SunFounder Universal Maker Sensor Kit

www.sunfounder.com

Aug 13, 2024

CONTENTS

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

-

Thanks for choosing our .

Note: This document is available in the following languages.

-
-
-

Please click on the respective links to access the document in your preferred language.

Have you ever ordered an electronic kit online, only to find it came with a basic PDF or a limited booklet that barely scratches the surface of your project's potential? Or are you eager to build your own smart devices but feel daunted by the complexity and high costs of available kits? Maybe you've admired the advanced projects others have created but didn't know where to start?

Introducing our "Universal Maker Sensor Kit" – the solution to all these challenges and the gateway to mastering modern electronics.

Within the Universal Maker Sensor Kit, you'll find an extensive range of components, from basic breadboards to intricate sensors like flame detectors, gas sensors, and much more – over 25 sensors, actuators, and display modules in total. Each component comes with an easy-to-follow tutorial supporting Arduino Uno, ESP32 modules, Raspberry Pi Pico, and Raspberry Pi, making your learning journey smooth and engaging.

Our kit proudly supports the latest in the Arduino UNO series, the UNO R4, as well as the latest Raspberry Pi release, the Raspberry Pi 5, ensuring you stay at the forefront of technological advancements. Whether you're a beginner or an experienced maker, our kit can enhance your skills through cutting-edge components.

This kit isn't just about assembling parts; it's about unleashing your creativity. You'll learn to write your own code, develop unique projects, and understand the intricacies of each component. Whether you're a beginner or an experienced maker, our kit is tailored to elevate your skills in electronics.

And for those just starting, we offer a range of engaging projects, perfect for easing into the world of programming and electronics. You'll gain the knowledge and confidence to progress from a beginner to an expert, creating your own smart devices and electronic projects.

Embrace the world of electronic innovation today with our Universal Maker Sensor Kit. Transform your ideas into reality, and evolve from zero to hero in the realm of electronics and programming!

If you have any questions or other interesting ideas, please feel free to email us at service@sunfounder.com.

CHAPTER

ONE

TABLE OF CONTENTS

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.1 Download the Code

Download the relevant code from the link below.

- `SunFounder Universal Maker Sensor Kit Code`
- Or check out the code at SunFounder Universal Maker Sensor Kit - GitHub

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2 Learn about the Components in Your Kit

After opening the package, please check whether the quantity of components is compliance with product description and whether all components are in good condition.

- `SunFounder Universal Maker Sensor Kit Components List`

The following is an introduction to each component, including its working principle and corresponding project. **Each component has a simple code example to help you get started quickly.**

Basic

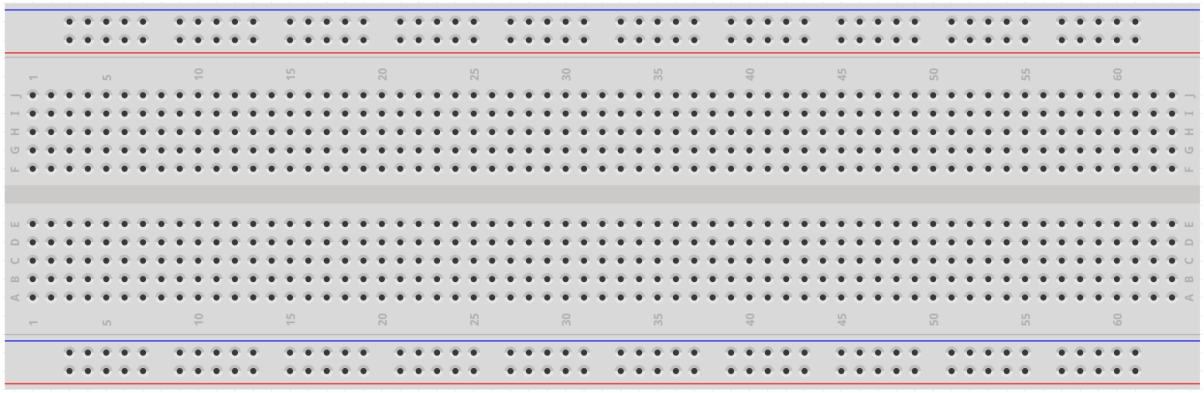
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

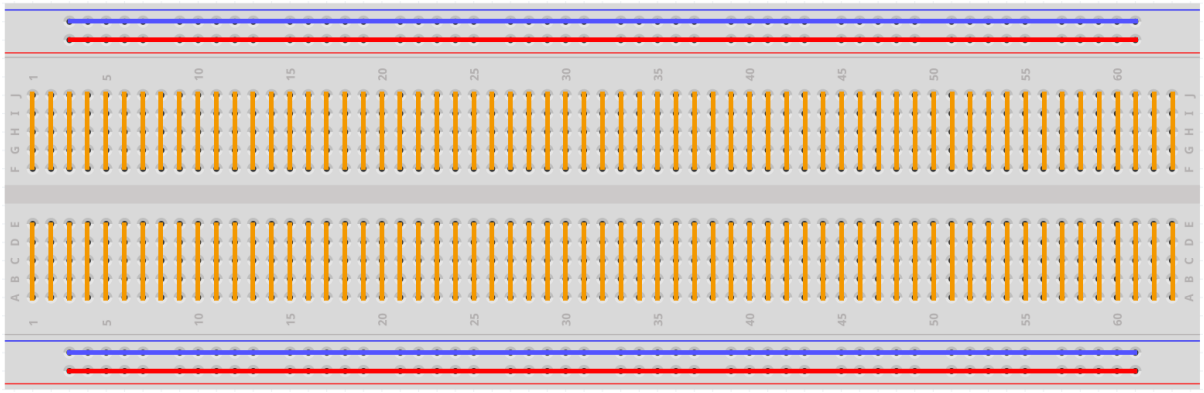
1.2.1 Breadboard



A breadboard is a construction base for prototyping of electronics. Originally the word referred to a literal bread board, a polished piece of wood used for slicing bread. In the 1970s the solderless breadboard (a.k.a. plugboard, a terminal array board) became available and nowadays the term "breadboard" is commonly used to refer to these.

It is used to build and test circuits quickly before finishing any circuit design. And it has many holes into which components mentioned above can be inserted like ICs and resistors as well as jumper wires. The breadboard allows you to plug in and remove components easily.

The picture shows the internal structure of a breadboard. Although these holes on the breadboard appear to be independent of each other, they are actually connected to each other through metal strips internally.



If you want to know more about breadboard, refer to:

Sensor

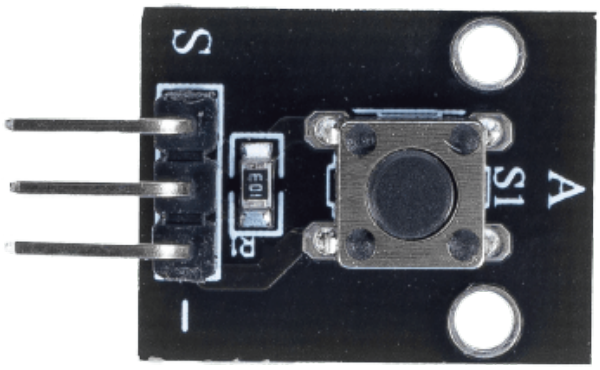
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.2 Button Module

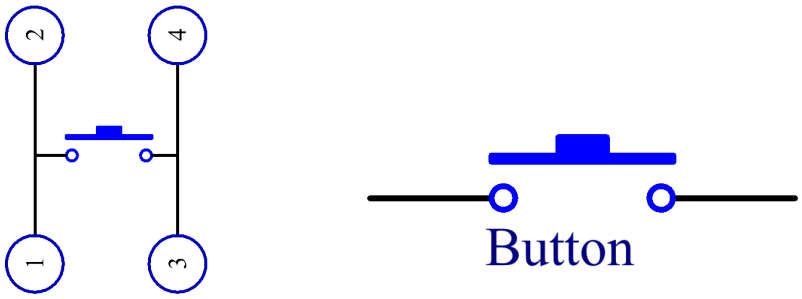


The button module is an electronic device that detects the state of a button.They are usually used as switches to connect or break circuits.Buttons are used in many scenarios, such as doorbells, desk lamps, remote controls, elevators, fire alarms, etc.

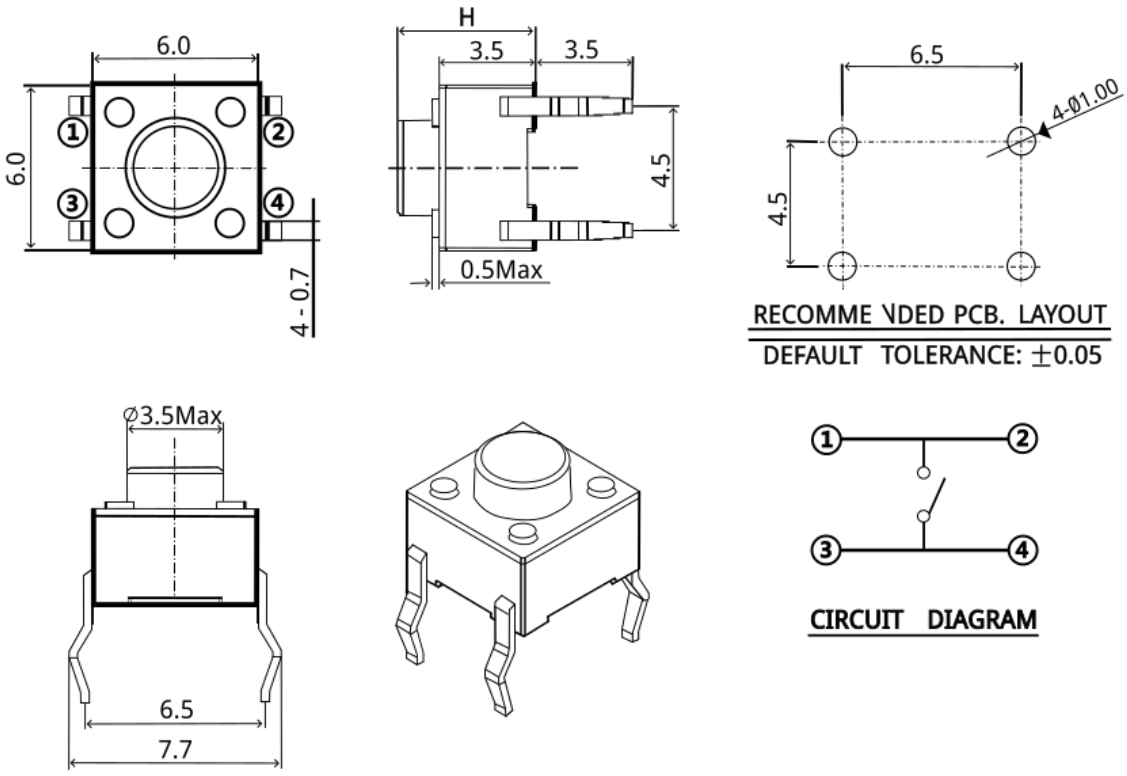
Principle

The button module works on the principle of a switch. A switch is an electrical component that can be used to open or close a circuit.

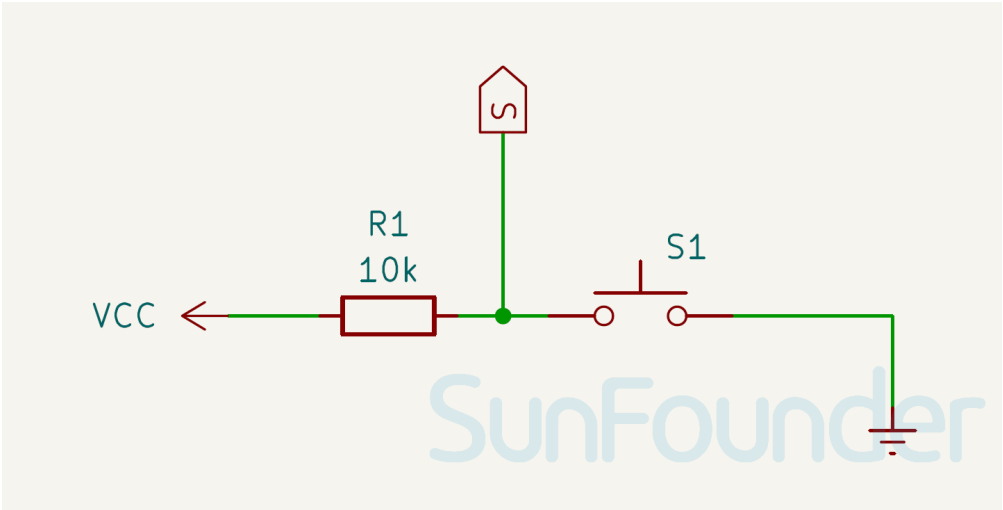
The following is the internal structure of a button. The symbol on the right below is usually used to represent a button in circuits.



Since the pin 1 is connected to pin 2, and pin 3 to pin 4, when the button is pressed, the 4 pins are connected, thus closing the circuit.



Schematic diagram



Example

- *Lesson 01: Button Module* (Arduino UNO)
- *Lesson 01: Button Module* (ESP32)
- *Lesson 01: Button Module* (Raspberry Pi Pico)
- *Lesson 01: Button Module* (Raspberry Pi)
- *Lesson 47: IoT Communication with MQTT* (ESP32)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.3 Capacitive Soil Moisture Module

The Soil Moisture Module is a sensor used in agriculture to measure the moisture content of soil, helping farmers monitor soil moisture levels and determine when to water their crops. This capacitive soil moisture sensor differs from the resistive sensors on the market, using the principle of capacitive induction to detect soil moisture. It avoids the problem of easy corrosion in resistive sensors and greatly extends its working life.

Pinout

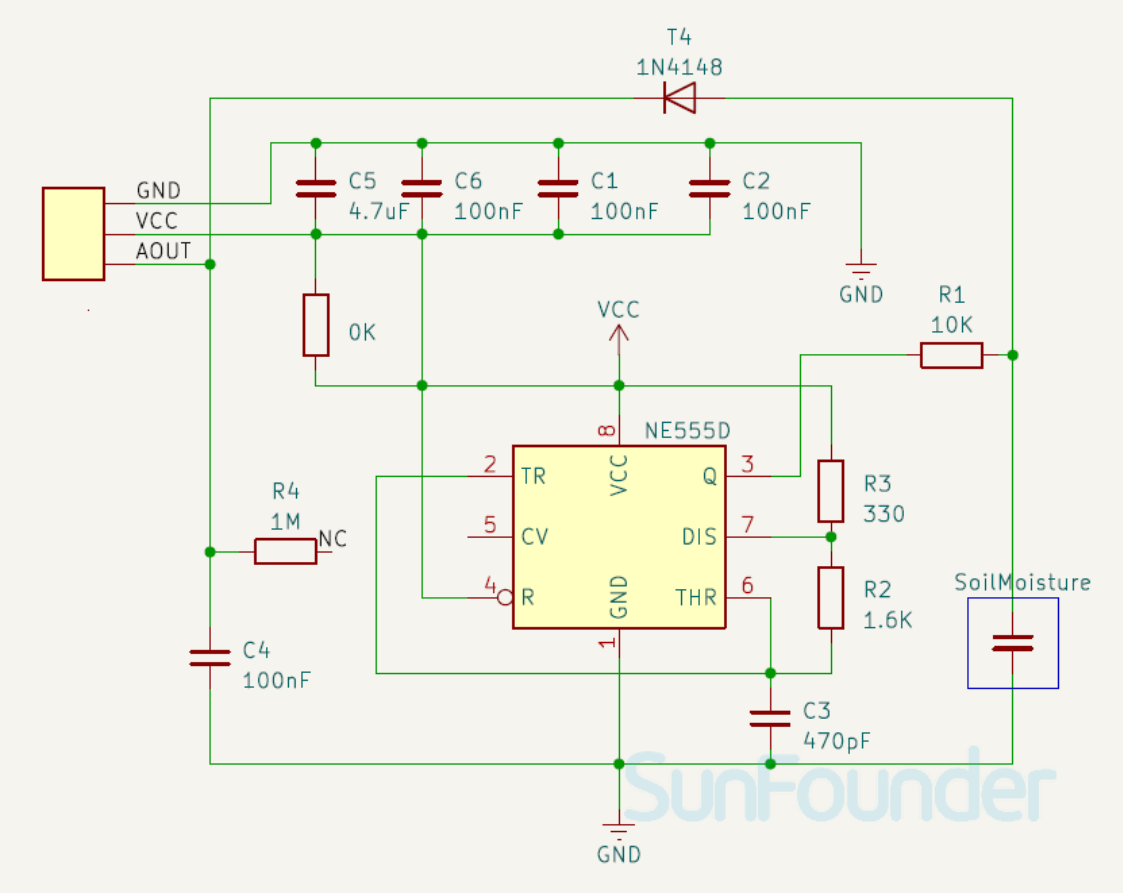
- **VCC**: This is the positive power supply input from the main control.
- **GND**: Ground connection.
- **AUOT**: Analog output. The higher the soil moisture content, the lower the analog output value.

Principle

This capacitive soil moisture sensor is different from most of the resistive sensors on the market, using the principle of capacitive induction to detect soil moisture. It avoids the problem that resistive sensors are highly susceptible to corrosion and greatly extends its working life.

It is made of corrosion-resistant materials and has an excellent service life. Insert it into the soil around plants and monitor real-time soil moisture data. The module includes an on-board voltage regulator that allows it to operate over a voltage range of 3.3 ~ 5.5 V. It is ideal for low-voltage microcontrollers with 3.3 V and 5 V supplies.

The hardware schematic of the capacitive soil moisture sensor is shown below.



There is a fixed frequency oscillator, which is built with a 555 timer IC. The generated square wave is then fed to the sensor like a capacitor. However, for the square wave signal, the capacitor has a certain reactance or, for the sake of argument, a resistor with a pure ohmic resistor (10k resistor on pin 3) to form a voltage divider.

The higher the soil moisture, the higher the capacitance of the sensor. As a result, the square wave has less reactance, which reduces the voltage on the signal line, and the smaller the value of the analog input through the microcontroller.

Example

- *Lesson 02: Capacitive Soil Moisture Module* (Arduino UNO)
- *Lesson 02: Capacitive Soil Moisture Module* (ESP32)
- *Lesson 02: Capacitive Soil Moisture Module* (Raspberry Pi Pico)
- *Lesson 02: Capacitive Soil Moisture Module* (Raspberry Pi Pi)
- *Lesson 45: Plant Monitor* (Arduino UNO)
- *Lesson 43: Plant Monitor* (ESP32)

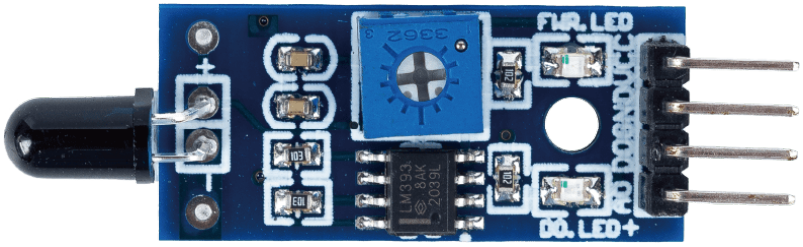
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.4 Flame Sensor Module



Tip: Keep a specific distance between the sensor and the flame to prevent damage from high temperatures.

Note: **Notice**: Due to a production error, some of the flame sensors included in our kits may be the 3-pin version, which lacks the AO (Analog Output). This version is suitable for most projects and does not impact general usage. If you still require the 4-pin version, please contact our customer service at service@sunfounder.com. We will provide a free replacement to meet your needs.

The Flame sensor is a sensor that can detect the presence of fire or flames. The flame sensor works based on infrared radiation. The IR photodiode will detect the IR radiation from any hot body. This value is then compared with a set value. Once the radiation reaches the threshold value, the sensor will change its output accordingly. It is widely used in fire detection systems in homes and industries.

The Flame sensor works on the principle of infrared (IR) detection. The sensor has an IR receiver that detects the IR radiation emitted by flames. When fire burns it emits a small amount of Infra-red light, this light will be received by

the Photodiode (IR receiver) on the sensor module. Then we use an Op-Amp to check for a change in voltage across the IR Receiver, so that if a fire is detected the output pin (DO) will give 0V(LOW), and if the is no fire the output pin will be 5V(HIGH).

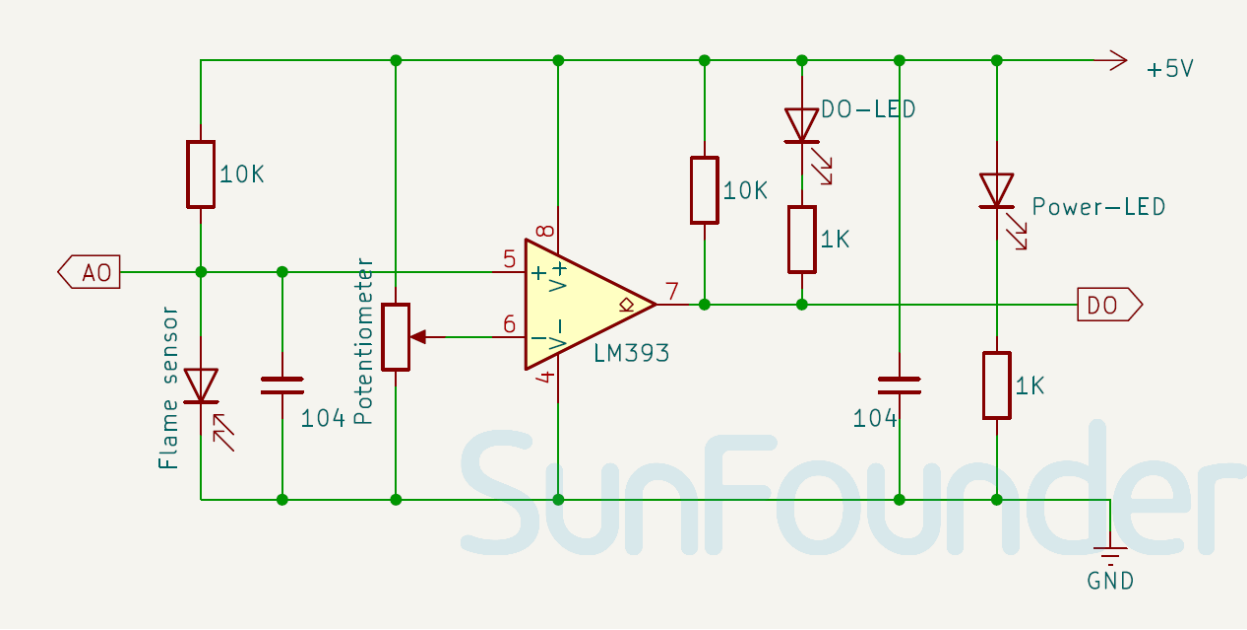
Specification

- Supply Voltage: 3.3V - 5V
- PCB Size: 31 x 14mm
- Output Signal Type: DO and AO
- Detection Angle: 60 degrees

Pinout

- **VCC**: This is the positive power supply input from the main control.
- **GND**: Ground connection.
- **DO**: Digital output. It indicates the presence of a flame. When the infrared radiation exceeds the threshold value (set by the potentiometer), DO becomes LOW; otherwise, it remains HIGH.
- **AO**: Analog output. It generates an output voltage that is inversely proportional to the intensity of infrared radiation (flame size). Therefore, higher infrared radiation will result in a lower voltage, while lower infrared radiation will result in a higher voltage.

Schematic diagram



Example

- *Lesson 03: Flame Sensor Module* (Arduino UNO)
- *Lesson 03: Flame Sensor Module* (ESP32)
- *Lesson 03: Flame Sensor Module* (Raspberry Pi Pico)
- *Lesson 03: Flame Sensor Module* (Raspberry Pi)
- *Lesson 50: Flame Alert System with Blynk* (Arduino UNO)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.5 Gas/Smoke Sensor Module (MQ2)

Tip: MQ2 is a heating-driven sensor that usually requires preheating before use. During the preheating period, the sensor typically reads high and gradually decreases until it stabilizes.

The MQ-2 sensor is a versatile gas sensor capable of detecting a wide range of gases including alcohol, carbon monoxide, hydrogen, isobutene, liquefied petroleum gas, methane, propane, and smoke. It is popular among beginners due to its low cost and easy-to-use features.

Principle

The MQ-2 sensor works on the principle of resistance changes in the presence of different gases. When the target gas comes in contact with the heated MOS(Metal Oxide Semiconductor) material, it undergoes oxidation or reduction reactions that change the resistance of the MOS material. **It is noteworthy that the MQ2 gas sensor is capable of detecting multiple gases, but lacks the ability to differentiate between them.** This is a common characteristic of most gas sensors.

The sensor has a built-in potentiometer that allows you to adjust the sensor digital output (D0) threshold. When the concentration of gas in the air exceeds a certain threshold value, the resistance of the sensor changes. This change in resistance is then converted into an electrical signal that can be read by an Arduino board.

Calibrating the MQ2 Gas Sensor

Because the MQ2 is a heater-driven sensor, the calibration of the sensor may drift if it is left in storage for an extended period of time. When first used after a long period of storage (a month or more), the sensor must be fully warmed up for 24-48 hours to ensure maximum accuracy. If the sensor has recently been used, it will only take 5-10 minutes to fully warm up. During the warm-up period, the sensor typically reads high and gradually decreases until it stabilizes.

Specification

- Model: MQ2
- Supply Voltage: 5V
- PCB Size: 32 x 20mm
- Output Signal Type: DO and AO
- Detection Concentration: 300 to 10000ppm
- Preheat Duration: Over 24 hours (first time)
- Detect Gas: LPG, Alcohol, Propane, Hydrogen, CO and even methane

Pinout

- **VCC**: This is the positive power supply input from the main control.
- **GND**: Ground connection.
- **DO**: Digital output. It indicates the presence of combustible gases. When the gas concentration exceeds the threshold value (as set by the potentiometer), D0 becomes LOW; otherwise, it is HIGH.
- **AO**: Analog output. It produces an analog output voltage proportional to gas concentration, so a higher concentration results in a higher voltage and a lower concentration results in a lower voltage.

Example

- *Lesson 04: Gas Sensor Module (MQ-2)* (Arduino UNO)
- *Lesson 04: Gas Sensor Module (MQ-2)* (ESP32)
- *Lesson 04: Gas Sensor Module (MQ-2)* (Raspberry Pi Pico)
- *Lesson 04: Gas Sensor Module (MQ-2)* (Raspberry Pi)
- *Lesson 38: Gas leak alarm* (Arduino UNO)
- *Lesson 36: Gas leak alarm* (ESP32)

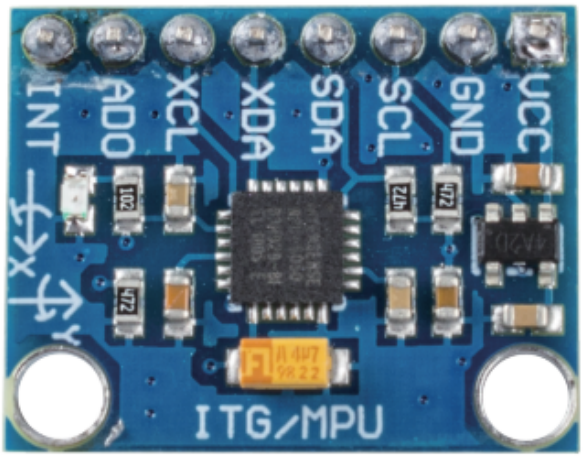
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.6 Gyroscope & Accelerometer Module (MPU6050)



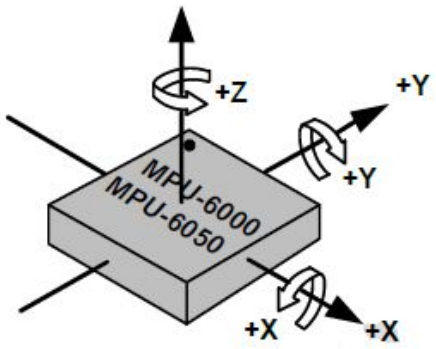
The MPU-6050 is a 6-axis(combines 3-axis Gyroscope, 3-axis Accelerometer) motion tracking devices.Changes in motion, acceleration and rotation can be detected.It is commonly used in robotics, gaming controllers, and other electronic devices that require motion detection. Its high accuracy and cheap cost make it very popular among the DIY community.

Principle

An MPU-6050 sensor module consists of a 3-axis accelerometer and a 3-axis gyroscope.

Its three coordinate systems are defined as follows:

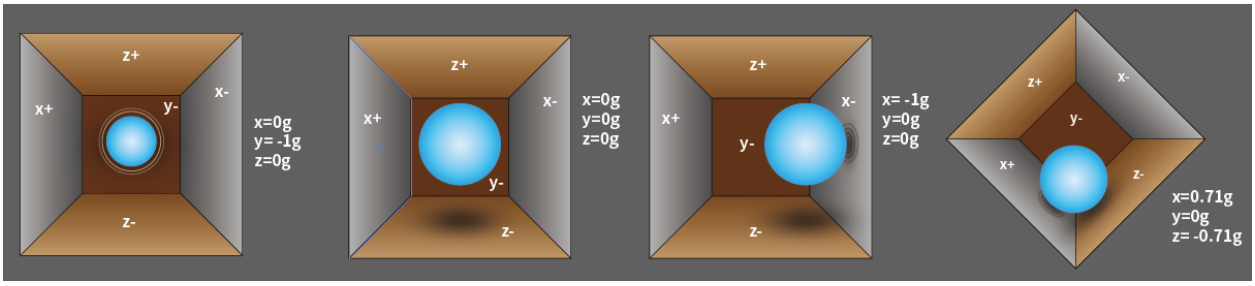
Put MPU6050 flat on the table, assure that the face with label is upward and a dot on this surface is on the top left corner. Then the upright direction upward is the z-axis of the chip. The direction from left to right is regarded as the X-axis. Accordingly the direction from back to front is defined as the Y-axis.



3-axis Accelerometer

The accelerometer works on the principle of piezo electric effect, the ability of certain materials to generate an electric charge in response to applied mechanical stress.

Here, imagine a cuboidal box, having a small ball inside it, like in the picture above. The walls of this box are made with piezo electric crystals. Whenever you tilt the box, the ball is forced to move in the direction of the inclination, due to gravity. The wall with which the ball collides, creates tiny piezo electric currents. There are totally, three pairs of opposite walls in a cuboid. Each pair corresponds to an axis in 3D space: X, Y and Z axes. Depending on the current produced from the piezo electric walls, we can determine the direction of inclination and its magnitude.



We can use the MPU6050 to detect its acceleration on each coordinate axis (in the stationary desktop state, the Z-axis acceleration is 1 gravity unit, and the X and Y axes are 0). If it is tilted or in a weightless/overweight condition, the corresponding reading will change.

There are four kinds of measuring ranges that can be selected programmatically: +/-2g, +/-4g, +/-8g, and +/-16g (2g by default) corresponding to each precision. Values range from -32768 to 32767.

The reading of accelerometer is converted to an acceleration value by mapping the reading from the reading range to the measuring range.

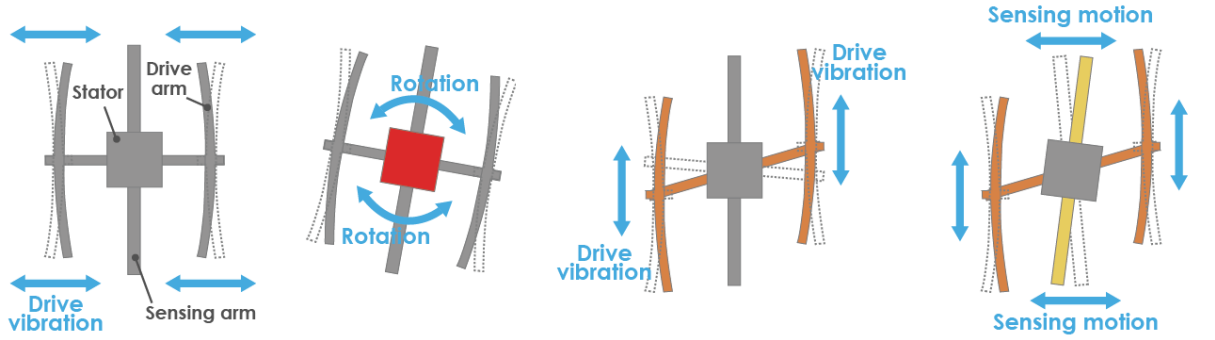
Acceleration = (Accelerometer axis raw data / 65536 * full scale Acceleration range) g

Take the X-axis as an example, when Accelerometer X axis raw data is 16384 and the range is selected as +/-2g:

Acceleration along the X axis = (16384 / 65536 * 4) g =1g

3-axis Gyroscope

Gyroscopes work on the principle of Coriolis acceleration. Imagine that there is a fork like structure, that is in constant back and forth motion. It is held in place using piezo electric crystals. Whenever, you try to tilt this arrangement, the crystals experience a force in the direction of inclination. This is caused as a result of the inertia of the moving fork. The crystals thus produce a current in consensus with the piezo electric effect, and this current is amplified.



1.Normally, a drive arm vibrates in a certain direction.

2.Direction of rotation

3.When the gyro is rotated,the Coriolis force acts on the drive arms, producing vertical vibration.

4.The stationary part bends due to vertical drive arm vibration, producing a sensing motion in the sensing arms.

The Gyroscope also has four kinds of measuring ranges: +/- 250, +/- 500, +/- 1000, +/- 2000. The calculation method and Acceleration are basically consistent.

The formula for converting the reading into angular velocity is as follows:

Angular velocity = (Gyroscope axis raw data / 65536 * full scale Gyroscope range) °/s

The X axis, for example, the Accelerometer X axis raw data is 16384 and ranges + / - 250°/ s:

Angular velocity along the X axis = (16384 / 65536 * 500)°/s =125°/s

Example

- *Lesson 05: Gyroscope & Accelerometer Module (MPU6050)* (Arduino UNO)
- *Lesson 05: Gyroscope & Accelerometer Module (MPU6050)* (ESP32)
- *Lesson 05: Gyroscope & Accelerometer Module (MPU6050)* (Raspberry Pi Pico)
- *Lesson 05: Gyroscope & Accelerometer Module (MPU6050)* (Raspberry Pi Pi)
- *Lesson 52: Tilt Direction Indicator* (Arduino UNO)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

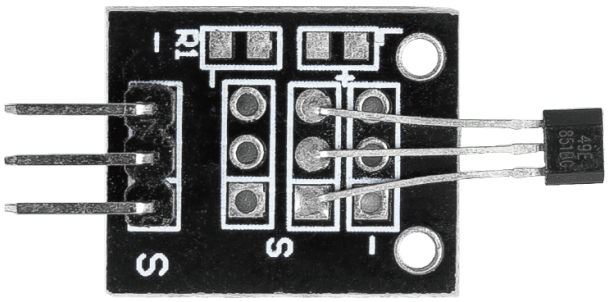
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.7 Hall Sensor Module



The Hall Sensor module is a magnetic non-contact sensor that produces an electrical signal proportional to the applied magnetic field. It can measure both north and south polarity of a magnetic field and the relative strength of the field. It's used for detecting magnetic fields, acting like a magnet detector that can identify nearby magnets. This sensor is useful in various projects, such as developing door alarm systems or measuring the speed of rotating objects.

Principle

The working principle of the Hall Sensor module is based on the , discovered by Edwin Hall. Here's how it works in simple terms: when electricity flows through a conductor (like a wire), and there's a magnetic field around it, the magnetic field pushes the moving electrons in the conductor to one side. This creates a voltage difference across the conductor - this is the Hall Effect.

In the Hall Sensor module, when a magnet comes close, the magnetic field affects the electrons in the semiconductor material inside the sensor. This changes the voltage across the sensor, which the sensor detects. The Arduino can read this voltage change and understand whether there's a magnet nearby and how strong its magnetic field is.

The Hall Sensor module is equipped with a 49E Linear Hall-Effect Sensor, capable of measuring both the north and south polarity of a magnetic field as well as the relative strength of the field. The output pin provides an analog representation indicating the presence and strength of a magnetic field, along with its polarity (north or south). When

no magnetic field is present, the 49E outputs a voltage around half of the source voltage. If the south pole of a magnet is placed near the labeled side of the 49E (the side with text etched on it), then the output voltage will linearly increase towards the source voltage in proportion to the strength of the applied magnetic field. Conversely, if you place a north pole near this side, then there will be a linear decrease in output voltage relative to the strength of that magnetic field.

For instance, when powering 49E with 5V and no magnetic field present, its output will be approximately 2.5V. In this scenario, placing a strong magnet's south pole near it would cause an increase in output voltage up to around 4.2V; while placing its north pole nearby would result in dropping down to about 0.86V from source based on their respective strengths.

Example

- *Lesson 06: Hall Sensor Module* (Arduino UNO)
- *Lesson 06: Hall Sensor Module* (ESP32)
- *Lesson 06: Hall Sensor Module* (Raspberry Pi Pico)
- *Lesson 06: Hall Sensor Module* (Raspberry Pi Pi)

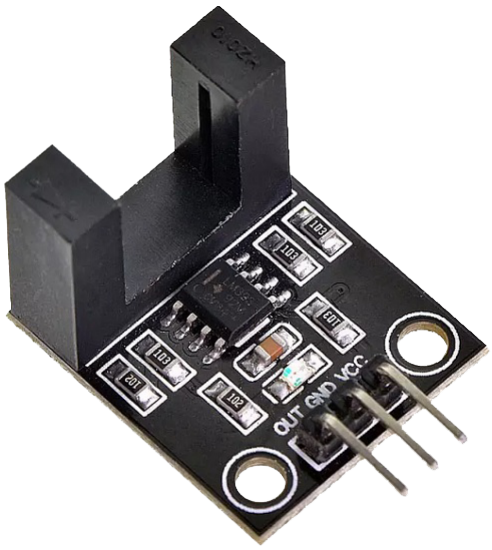
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.8 Infrared Speed Sensor Module



The Infrared Speed Sensor Module is an IR counter that has an IR transmitter and receiver. If any obstacle is placed between these sensors, a signal is sent to the microcontroller. The module can be used in association with a microcontroller for motor speed detection, pulse count, position limit, etc.

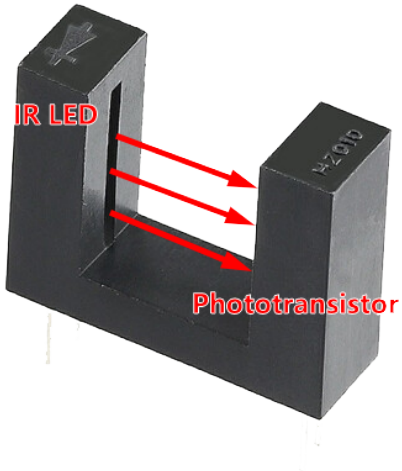
Pinout

- **VCC**: This is the positive power supply(3.3V or 5V) input from the main control.
- **GND**: Ground connection.
- **OUT**: Digital output. When the speed sensor is obstructed, it outputs a high level; when unobstructed, it outputs a low level.

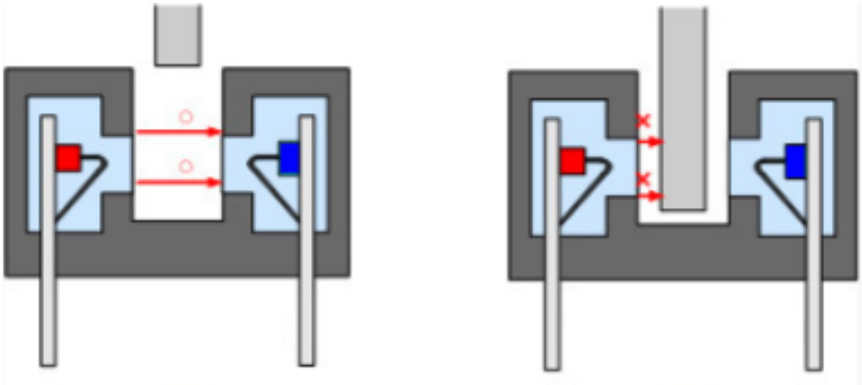
Principle

The speed sensor module is mainly used to detect changes in rotational speed or velocity. When an object passes by the H2010 sensor, it generates a pulse signal. The integrated LM393 comparator inside the module compares this pulse signal with a preset threshold, producing a stable high-level output signal.

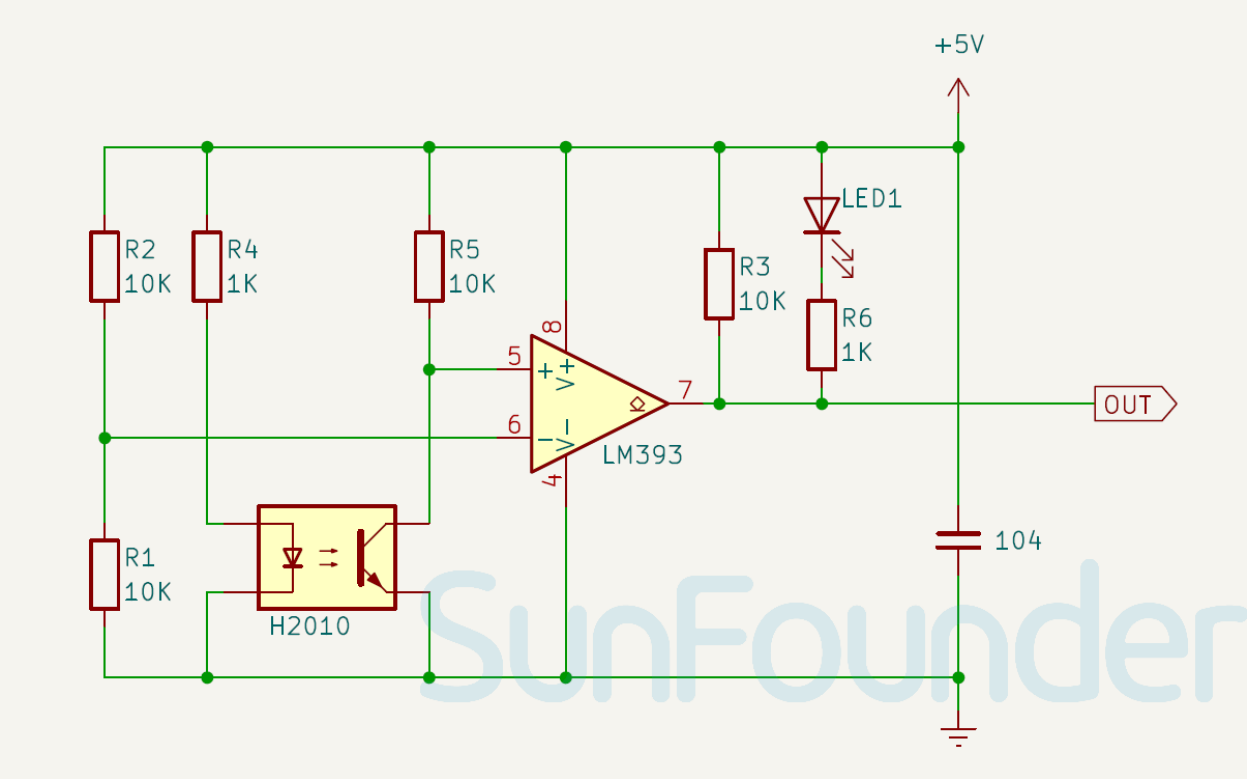
The Infrared Speed Sensor Module has 1 H2010 photocell, which consists of a phototransistor and an infrared light emitter packaged in a 10 cm wide black plastic housing.



When operating, the infrared light-emitting diode continuously emits infrared light (invisible light), and the photosensitive triode will conduct if it receives it.



Schematic diagram



Example

- *Lesson 07: Infrared Speed Sensor Module* (Arduino UNO)
- *Lesson 07: Infrared Speed Sensor Module* (ESP32)
- *Lesson 07: Infrared Speed Sensor Module* (Raspberry Pi Pico)
- *Lesson 07: Infrared Speed Sensor Module* (Raspberry Pi)

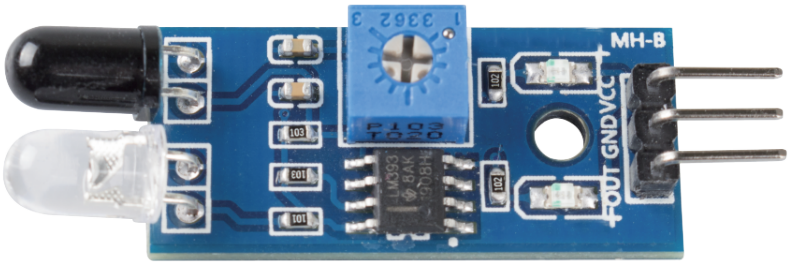
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.9 IR Obstacle Avoidance Sensor Module



This module can adapt to ambient light and includes a pair of infrared emitting and receiving tubes. The emitting tube sends out infrared at a specific frequency, and when the detection direction encounters an obstacle (reflective surface), the receiving tube picks up the reflected infrared. After being processed by the comparator circuit, the green indicator light will turn on, and simultaneously, the signal output interface will produce a digital signal (a low level signal). The detection distance can be adjusted using a potentiometer knob.

Specification

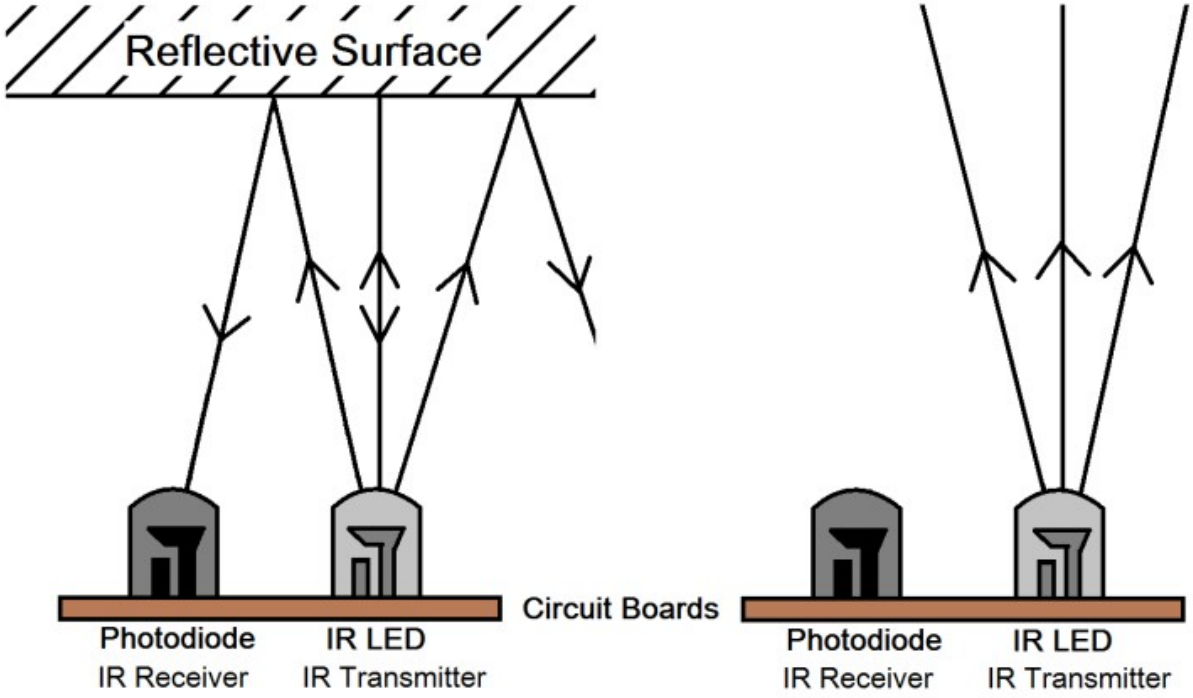
- Supply Voltage: 3.3V - 5V
- PCB Size: 32 x 14mm
- Output Signal Type: Digital Output
- Detection Angle: 35°
- Detection distance: 230cm

Pinout

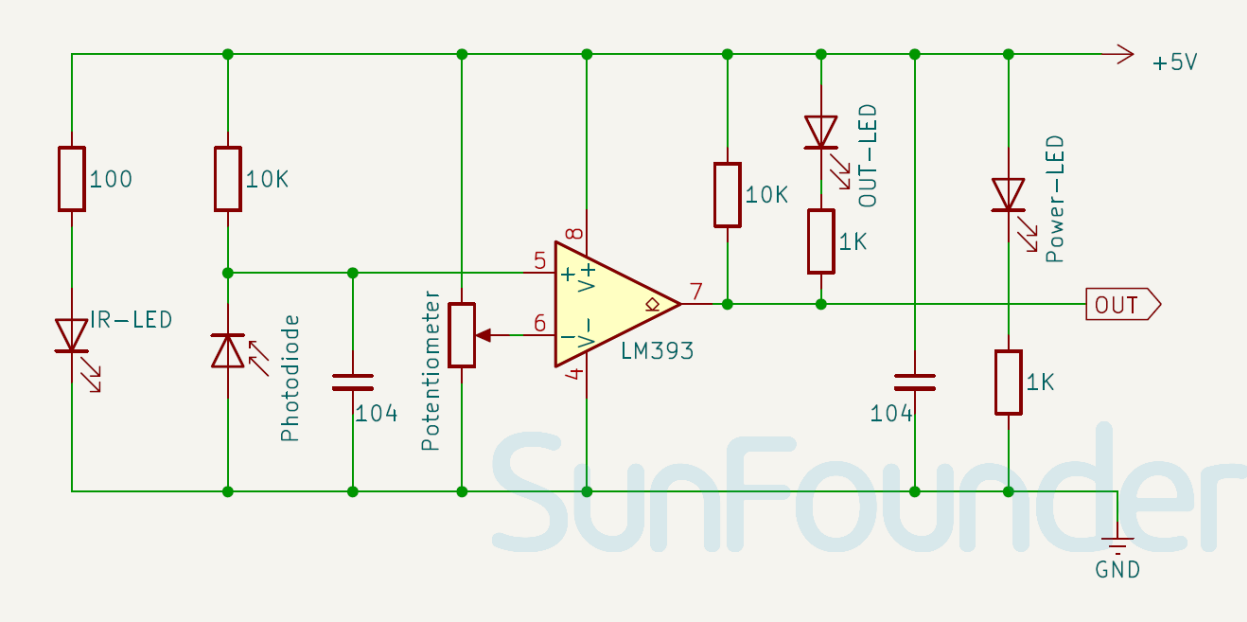
- **VCC**: This is the positive power supply input from the main control.
- **GND**: Ground connection.
- **OUT**: Digital output. Outputs high level when there is no obstacle, and outputs low level when an obstacle is detected. The detection distance of obstacles can be adjusted by the potentiometer on the module.

Principle

An obstacle avoidance sensor mainly consists of an infrared transmitter, an infrared receiver and a potentiometer. According to the reflecting character of an object, if there is no obstacle, the emitted infrared ray will weaken with the distance it spreads and finally disappear. If there is an obstacle, when the infrared ray encounters it, the ray will be reflected back to the infrared receiver. Then the infrared receiver detects this signal and confirms an obstacle in front. The detection range can be adjusted by the built-in potentiometer.



Schematic diagram



Example

- *Lesson 08: IR Obstacle Avoidance Sensor Module* (Arduino UNO)
- *Lesson 08: IR Obstacle Avoidance Sensor Module* (ESP32)
- *Lesson 08: IR Obstacle Avoidance Sensor Module* (Raspberry Pi Pico)
- *Lesson 08: IR Obstacle Avoidance Sensor Module* (Raspberry Pi)
- *Lesson 39: Automatic soap dispenser* (Arduino UNO)
- *Lesson 37: Automatic soap dispenser* (ESP32)

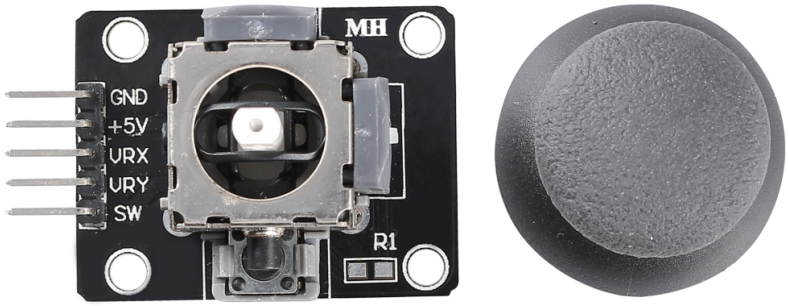
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.10 Joystick Module



A joystick module is a device that can measure the movement of a knob in two directions: horizontal (X-axis) and vertical (Y-axis). A joystick module can be used to control various things such as games, robots, cameras, etc.

Specification

- Supply Voltage: 3.3V or 5V
- PCB Size: 34 x 26mm
- Output Signal Type: DO and AO
- Analog Output: X, Y, 2 Axis analog output
- Digital Output: Z, digital output

Pinout

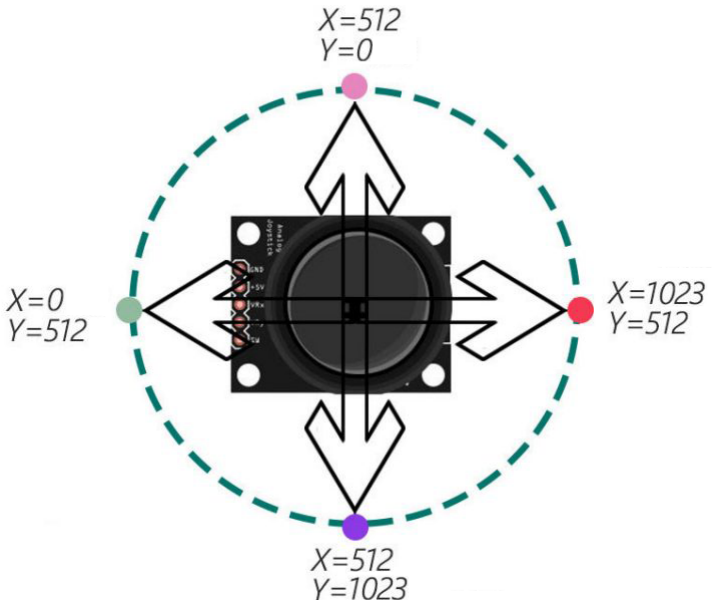
- **+5V**: This is the positive power supply input from the main control.
- **GND**: Ground connection.
- **VRX**: Analog output. X-axis analog output voltage. Moving the joystick from left to right will cause the output voltage to change from 0 to VCC. When the joystick is in the center position (idle state), it will read about half of VCC.
- **VRY**: Analog output. Y-axis analog output voltage. Moving the joystick up or down will cause the output voltage to change from 0 to VCC. When the joystick is in the center position (at rest), it will read approximately half of VCC.
- **SW**: Digital output. The pushbutton switch outputs a floating signal by default.

Tip: To read the pushbutton switch, a pull-up resistor is needed. When the joystick knob is pressed, the switch output becomes LOW; otherwise, it remains HIGH. Ensure that the input pin connected to the switch has either internal pull-up enabled or an external pull-up resistor connected.

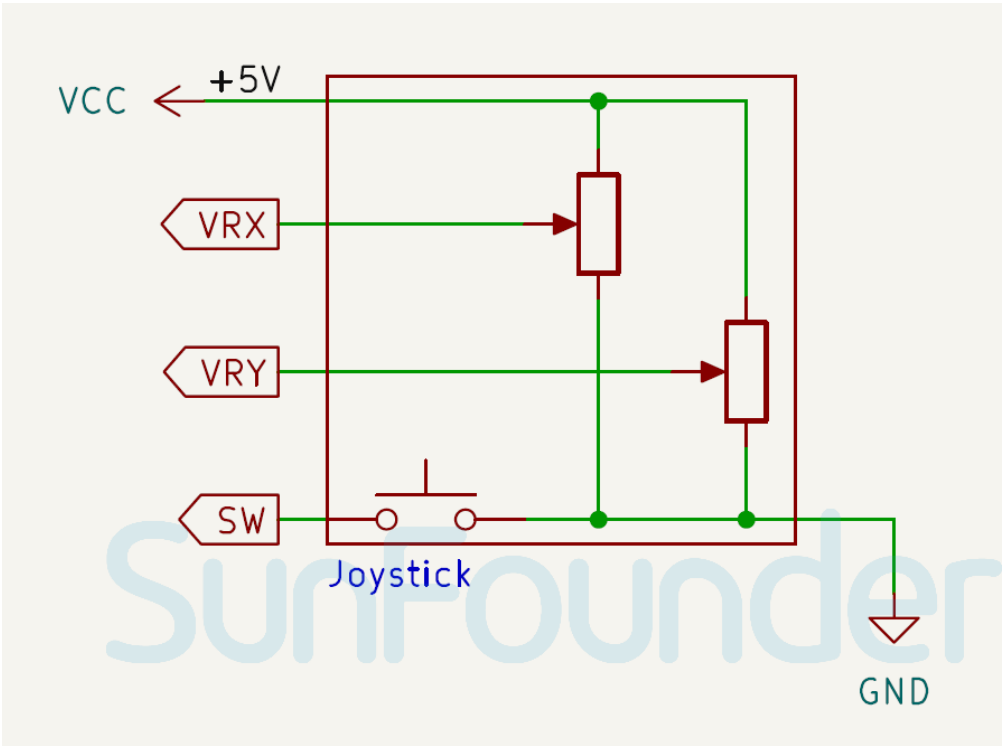
Principle

Joystick operates based on the resistance change of two potentiometers (usually 10-kilo ohms). By changing resistance in x and y directions, Arduino receives varying voltages which are interpreted to x and y coordinates. The processor needs an ADC unit to change the joystick's analog values into digital values and perform necessary processing.

Arduino boards have six 10-bits ADC channels. It means the Arduino's reference voltage (5 volts) is divided to 1024 segments. When joystick moves along the x-axis, the ADC value rises from 0 to 1023, with the value 512 in the middle. The image below displays the ADC approximate value based on the joystick position.



Schematic diagram



Example

- *Lesson 09: Joystick Module* (Arduino UNO)
- *Lesson 09: Joystick Module* (ESP32)
- *Lesson 09: Joystick Module* (Raspberry Pi Pico)
- *Lesson 09: Joystick Module* (Raspberry)
- *Lesson 53: Direction Indicator* (Arduino UNO)

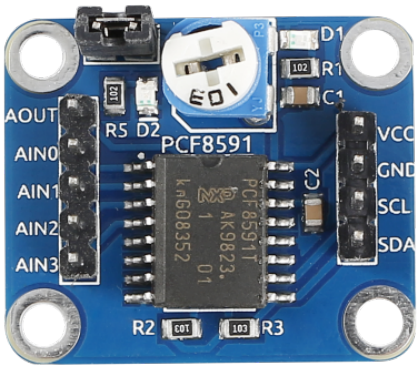
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.11 PCF8591 ADC DAC Converter Module



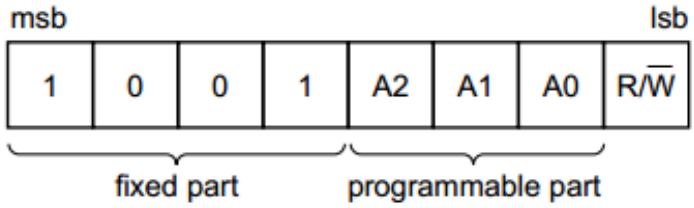
The PCF8591 is a single-chip, single-supply low-power 8-bit CMOS data acquisition device with four analog inputs, one analog output and a serial I2C-bus interface. Three address pins A0, A1 and A2 are used for programming the hardware address, allowing the use of up to eight devices connected to the I2C-bus without additional hardware. Address, control and data to and from the device are transferred serially via the two-line bidirectional I2C-bus.

The functions of the device include analog input multiplexing, on-chip track and hold function, 8-bit analog-to-digital conversion and an 8-bit digital-to-analog conversion. The maximum conversion rate is given by the maximum speed of the I2C-bus.

Principle

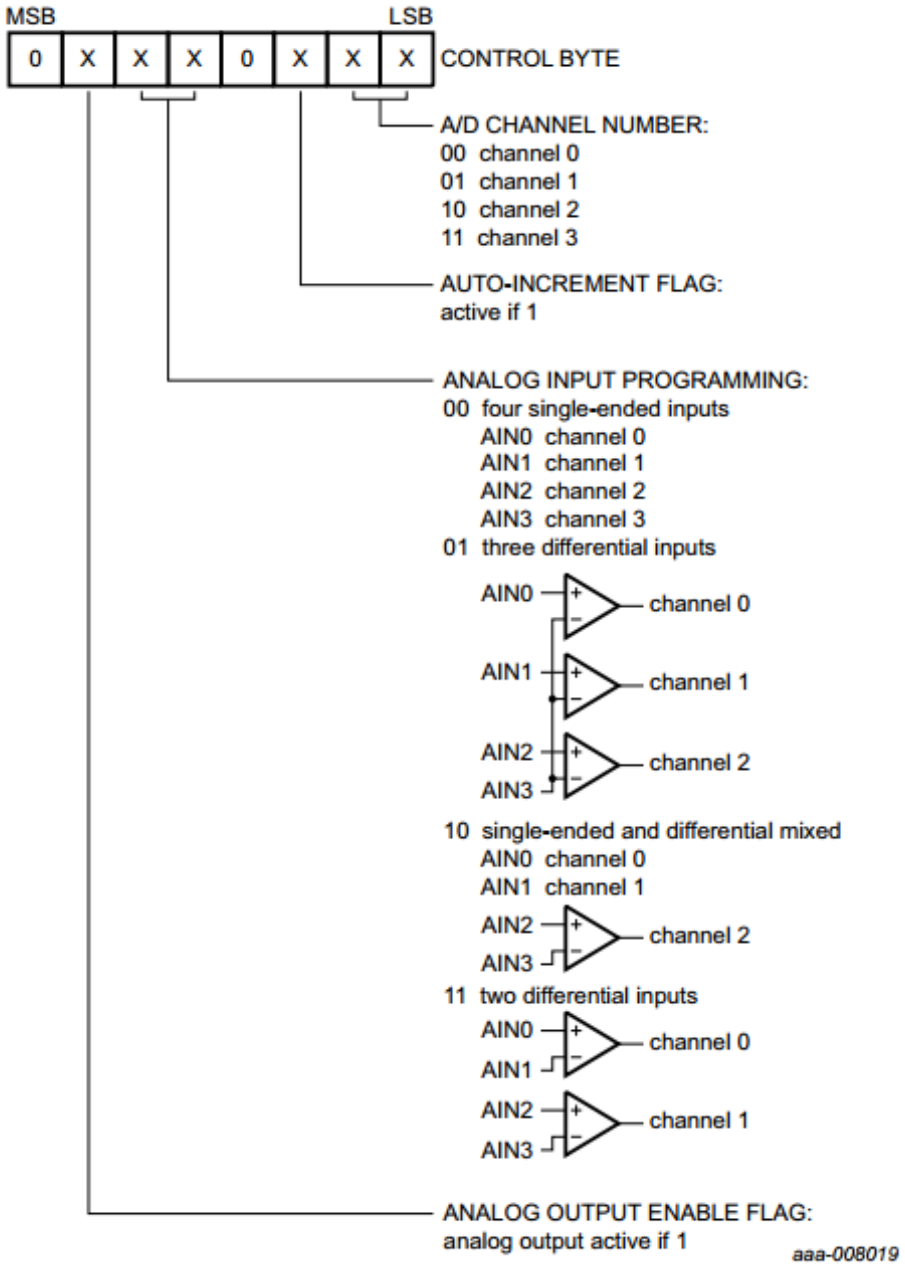
Addressing:

Each PCF8591 device in an I2C-bus system is activated by sending a valid address to the device. The address consists of a fixed part and a programmable part. The programmable part must be set according to the address pins A0, A1 and A2. The address always has to be sent as the first byte after the start condition in the I2C-bus protocol. The last bit of the address byte is the read/write-bit which sets the direction of the following data transfer (see as below).

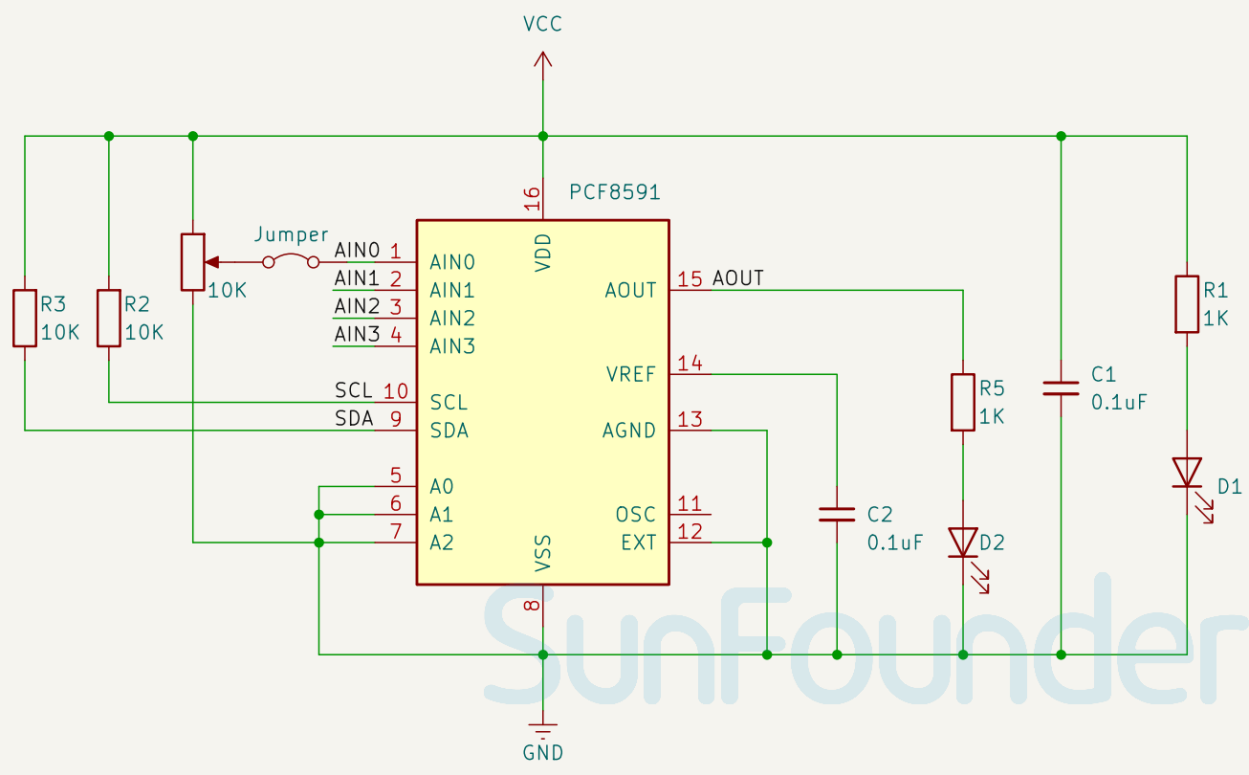


Control byte:

The second byte sent to a PCF8591 device will be stored in its control register and is required to control the device function. The upper nibble of the control register is used for enabling the analog output, and for programming the analog inputs as single-ended or differential inputs. The lower nibble selects one of the analog input channels defined by the upper nibble. If the auto-increment flag is set, the channel number is incremented automatically after each A/D conversion. See the figure below.


MSB
LSB
0 X X X 0 X X X
CONTROL BYTE
A/D CHANNEL NUMBER:
00 channel 0
01 channel 1
10 channel 2
11 channel 3
AUTO-INCREMENT FLAG:
active if 1
ANALOG INPUT PROGRAMMING:
00 four single-ended inputs
AIN0 channel 0
AIN1 channel 1
AIN2 channel 2
AIN3 channel 3
01 three differential inputs
AIN0
channel 0
AIN1
channel 1
AIN2
AIN3
channel 2
10 single-ended and differential mixed
AIN0 channel 0
AIN1 channel 1
AIN2
AIN3
channel 2
11 two differential inputs
AIN0
AIN1
channel 0
AIN2
AIN3
channel 1
ANALOG OUTPUT ENABLE FLAG:
analog output active if 1
aaa-008019

Schematic diagram



Example

- *Lesson 10: PCF8591 ADC DAC Converter Module* (Arduino UNO)
- *Lesson 10: PCF8591 ADC DAC Converter Module* (ESP32)
- *Lesson 10: PCF8591 ADC DAC Converter Module* (Raspberry Pi Pico)
- *Lesson 10: PCF8591 ADC DAC Converter Module* (Raspberry Pi)
- *Lesson 02: Capacitive Soil Moisture Module* (Raspberry Pi)
- *Lesson 09: Joystick Module* (Raspberry Pi)
- *Lesson 11: Photoresistor Module* (Raspberry Pi)
- *Lesson 13: Potentiometer Module* (Raspberry Pi)
- *Lesson 25: Water Level Sensor Module* (Raspberry Pi)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

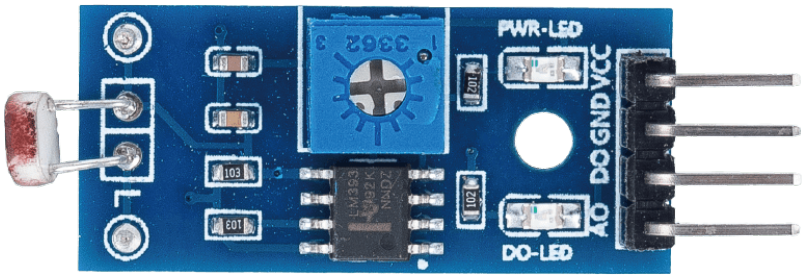
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.

- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.12 Photoresistor Module

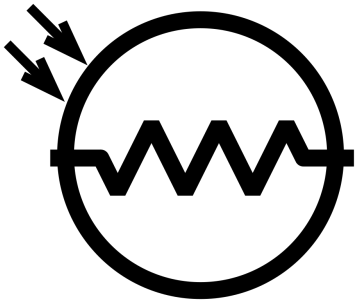


The photoresistor module is a device that can detect the intensity of light in the environment. It can be used for various purposes, such as adjusting the brightness of a device, detecting day and night, or activating a light switch.

An important component of the photoresistor module is the photoresistor. A photoresistor is a light-controlled variable resistor. The resistance of a photoresistor decreases with increasing incident light intensity; in other words, it exhibits photo conductivity.

A photoresistor can be applied in light-sensitive detector circuits and light-activated and dark-activated switching circuits acting as a resistance semiconductor. In the dark, a photoresistor can have a resistance as high as several megaohms (M), while in the light, a photoresistor can have a resistance as low as a few hundred ohms.

Here is the electronic symbol of photoresistor.



Specification

- Supply Voltage: 3.3V - 5V
- PCB Size: 32 x 14mm
- Output Signal Type: DO and AO

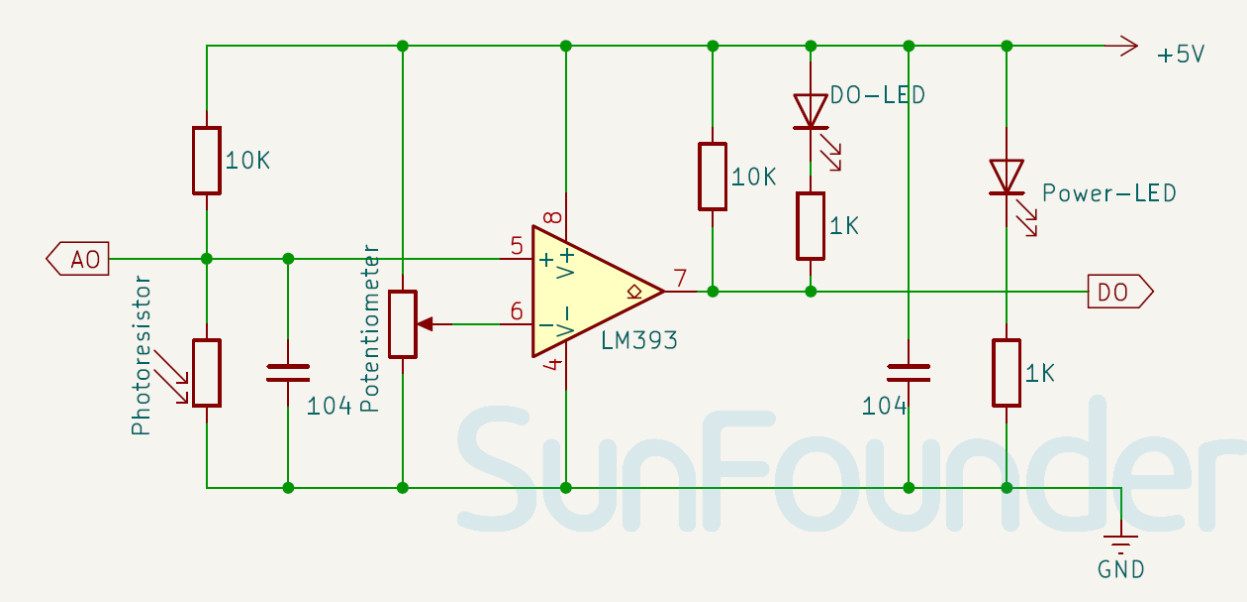
Pinout

- **VCC**: This is the positive power supply input from the main control.
- **GND**: Ground connection.
- **DO**: Digital output. When the intensity of the light exceeds the threshold value (set by the potentiometer), D0 becomes LOW; otherwise, it remains HIGH.
- **AO**: Analog output. The stronger the light, the lower the output value; conversely, the weaker the light, the higher the output value.

Principle

The photoresistor module works on the principle of changing resistance in response to different light intensities. The sensor has a built-in potentiometer that adjusts the sensor's digital output (D0) threshold.

Schematic diagram



Example

- *Lesson 11: Photoresistor Module* (Arduino UNO)
- *Lesson 11: Photoresistor Module* (ESP32)
- *Lesson 11: Photoresistor Module* (Raspberry Pi Pico)
- *Lesson 11: Photoresistor Module* (Raspberry Pi)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.

- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.13 PIR Motion Module (HC-SR501)

The Passive Infrared(PIR) Motion Sensor is a sensor that detects motion. It is commonly used in security systems and automatic lighting systems. The sensor has two slots that detect infrared radiation. When an object, such as a person, passes in front of the sensor, it detects a change in the amount of infrared radiation and triggers an output signal.

Specification

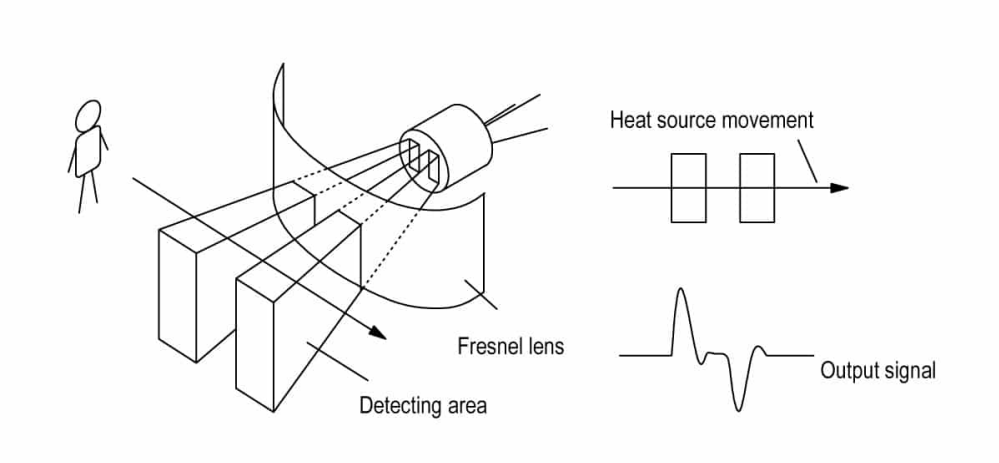
- Supply Voltage: 5V~20V;
- Output: Defaults to low; goes high when someone passes by.
- Delay Time: 5~200s(adjustable)
- Blocking Time: 8s
- Sensing Range: <120°, within 7 meters(adjustable)
- Trigger Mode: L Non-repeatable trigger mode, H Repeatable trigger mode
- PCB size: 32 x 24mm
- Lens size: 23mm
- Working temperature: -15~+70℃

Pinout

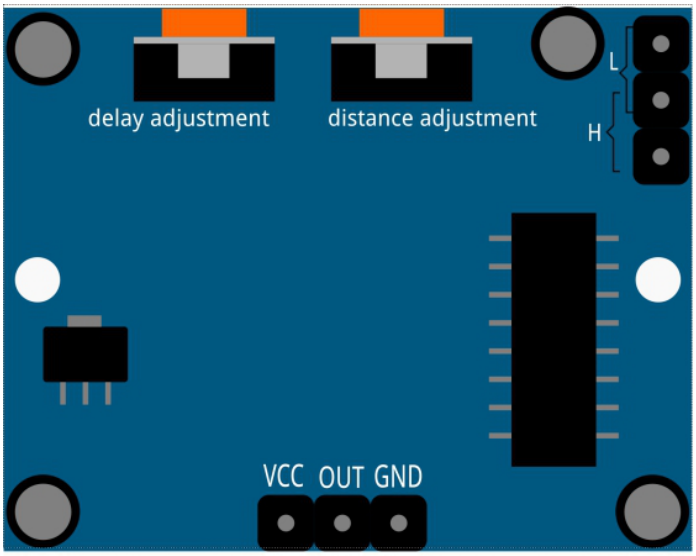
- **VCC**: This is the positive power supply input from the main control.
- **GND**: Ground connection.
- **DO**: Digital output. Defaults to low; goes high when someone passes by.

Principle

The PIR sensor is split into two slots that are connected to a differential amplifier. Whenever a stationary object is in front of the sensor, the two slots receive the same amount of radiation and the output is zero. Whenever a moving object is in front of the sensor, one of the slots receives more radiation than the other , which makes the output fluctuate high or low. This change in output voltage is a result of detection of motion.



After the sensing module is wired, there is a one-minute initialization. During the initialization, module will output for 0~3 times at intervals. Then the module will be in the standby mode. Please keep the interference of light source and other sources away from the surface of the module so as to avoid the misoperation caused by the interfering signal. Even you'd better use the module without too much wind, because the wind can also interfere with the sensor.



Distance Adjustment

Turning the knob of the distance adjustment potentiometer clockwise, the range of sensing distance increases, and the maximum sensing distance range is about 0-7 meters. If turn it anticlockwise, the range of sensing distance is reduced, and the minimum sensing distance range is about 0-3 meters.

Delay adjustment

Rotate the knob of the delay adjustment potentiometer clockwise, you can also see the sensing delay increasing. The maximum of the sensing delay can reach up to 300s. On the contrary, if rotate it anticlockwise, you can shorten the delay with a minimum of 5s.

Two Trigger Modes

Choosing different modes by using the jumper cap.

- H: Repeatable trigger mode, after sensing the human body, the module outputs high level. During the subsequent delay period, if somebody enters the sensing range,the output will keep being the high level.
- L: Non-repeatable trigger mode, outputs high level when it senses the human body. After the delay, the output will change from high level into low level automatically.

Example

- *Lesson 12: PIR Motion Module (HC-SR501)* (Arduino UNO)
- *Lesson 12: PIR Motion Module (HC-SR501)* (ESP32)
- *Lesson 12: PIR Motion Module (HC-SR501)* (Raspberry Pi Pico)
- *Lesson 12: PIR Motion Module (HC-SR501)* (Raspberry Pi)
- *Lesson 40: Motion triggered relay* (Arduino UNO)
- *Lesson 51: Intrusion Alert System with Blynk* (Arduino UNO)
- *Lesson 38: Motion triggered relay* (ESP32)
- *Lesson 49: Blynk-based Intrusion Notification System* (ESP32)

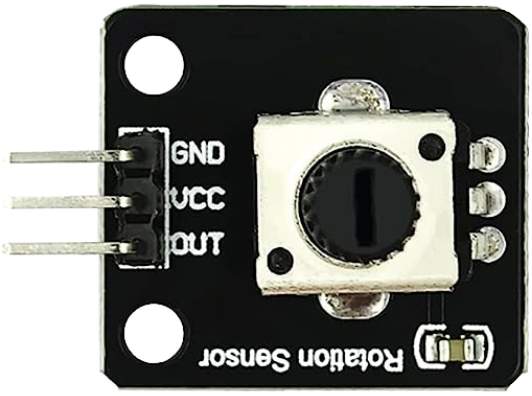
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.14 Potentiometer Module



The potentiometer module is an electronic component that changes its resistance depending on the position of the twist knob.It can be used for various purposes, such as controlling the volume of a speaker, the brightness of a LED, or the speed of a motor.

Pinout

- **VCC**: This is the positive power supply input from the main control.
- **GND**: Ground connection.
- **AO**: Analog output.

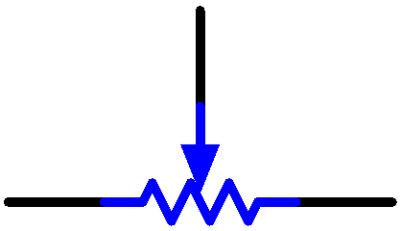
Principle

Potentiometer is also a resistance component with 3 terminals and its resistance value can be adjusted according to some regular variation.

Potentiometers come in various shapes, sizes, and values, but they all have the following things in common:

- They have three terminals (or connection points).
- They have a knob, screw, or slider that can be moved to vary the resistance between the middle terminal and either one of the outer terminals.
- The resistance between the middle terminal and either one of the outer terminals varies from 0 to the maximum resistance of the pot as the knob, screw, or slider is moved.

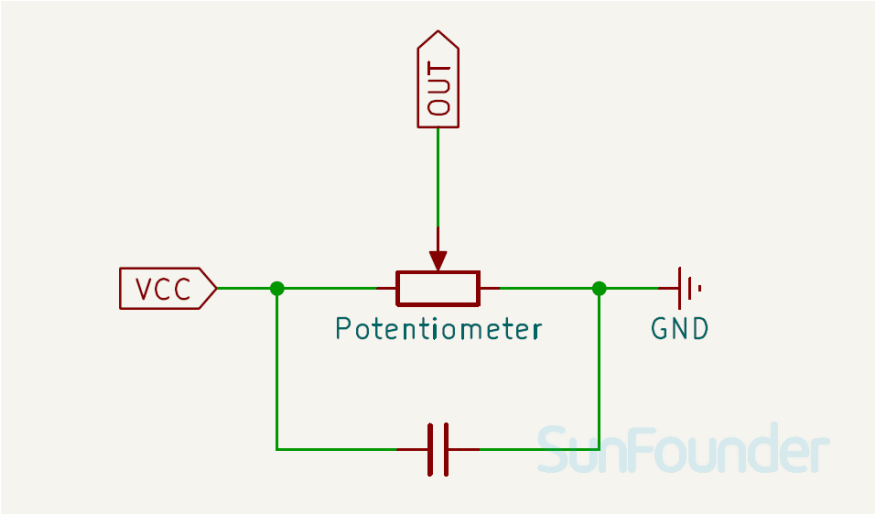
Here is the circuit symbol of potentiometer.



The functions of the potentiometer in the circuit are as follows:

1. **Serving as a voltage divider**
   Potentiometer is a continuously adjustable resistor. When you adjust the shaft or sliding handle of the potentiometer, the movable contact will slide on the resistor. At this point, a voltage can be output depending on the voltage applied onto the potentiometer and the angle the movable arm has rotated to or the travel it has made.
2. **Serving as a rheostat**
   When the potentiometer is used as a rheostat, connect the middle pin and one of the other 2 pins in the circuit. Thus you can get a smoothly and continuously changed resistance value within the travel of the moving contact.
3. **Serving as a current controller**
   When the potentiometer acts as a current controller, the sliding contact terminal must be connected as one of the output terminals.

Schematic diagram



Example

- *Lesson 13: Potentiometer Module* (Arduino UNO)
- *Lesson 13: Potentiometer Module* (ESP32)
- *Lesson 13: Potentiometer Module* (Raspberry Pi Pico)
- *Lesson 13: Potentiometer Module* (Raspberry Pi)
- *Lesson 43: Potentiometer scale value* (Arduino UNO)
- *Lesson 41: Potentiometer scale value* (ESP32)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

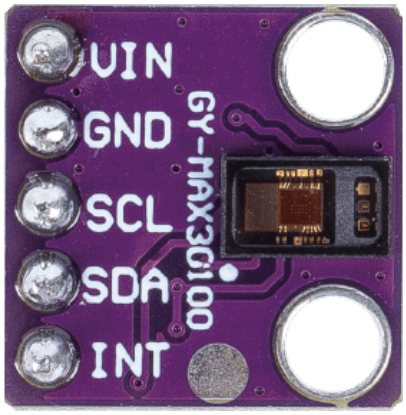
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.15 Pulse Oximeter and Heart Rate Sensor Module (MAX30102)



The MAX30102 is an advanced sensor module designed for tracking heart rate and blood oxygen levels (SpO2). Manufactured by Maxim Integrated, it combines pulse oximetry and heart rate monitoring into a compact package, making it a popular choice for wearable health and fitness applications.

Specification

- Chip Type: MAX30102
- LED Peak Wavelength: 660nm/880nm
- Supply Voltage: 3.3V or 5V;
- Detection Signal Type: Optical Reflection Signal (PPG)
- Output Signal Interface: I2C Interface
- PCB size: 14 x 14mm
- Working temperature: -40 ~ +85℃

Pinout

- **VCC**: This is the positive power supply input from the main control.
- **GND**: Ground connection.
- **SCL**: serial clock pin for the I2C interface.
- **SDA**: serial data pin for the I2C interface.
- **INT**: the Interrupt pin of the IC.

Principle

MAX30102 is a sensor that combines a pulse oximeter and a heart rate monitor. It's an optical sensor that measures the absorbance of pulsating blood through a photodetector after emitting two wavelengths of light from two LEDs - a red and an infrared one. This particular LED colour combination is designed to allow data to be read with the tip of one's finger.

The MAX30102 works by shining both lights onto the finger or earlobe (or essentially anywhere where the skin isn't too thick, so both lights can easily penetrate the tissue) and measuring the amount of reflected light using a photodetector. This method of pulse detection through light is called Photoplethysmogram.

The working of MAX30102 can be divided into two parts: Heart Rate Measurement and Pulse Oximetry (measuring the oxygen level of the blood).

Heart Rate Measurement

The oxygenated hemoglobin (HbO2) in the arterial blood has the characteristic of absorbing IR light. The redder the blood (the higher the hemoglobin), the more IR light is absorbed. As the blood is pumped through the finger with each heartbeat, the amount of reflected light changes, creating a changing waveform at the output of the photodetector. As you continue to shine light and take photodetector readings, you quickly start to get a heart-beat (HR) pulse reading.

Pulse Oximetry

Pulse oximetry is based on the principle that the amount of RED and IR light absorbed varies depending on the amount of oxygen in your blood.

Example

- *Lesson 14: Pulse Oximeter and Heart Rate Sensor Module (MAX30102)* (Arduino UNO)
- *Lesson 14: Pulse Oximeter and Heart Rate Sensor Module (MAX30102)* (ESP32)
- *Lesson 14: Pulse Oximeter and Heart Rate Sensor Module (MAX30102)* (Raspberry Pi Pico)
- *Lesson 14: Pulse Oximeter and Heart Rate Sensor Module (MAX30102)* (Raspberry Pi)
- *Lesson 41: Heart rate monitor* (Arduino UNO)
- *Lesson 39: Heart rate monitor* (ESP32)

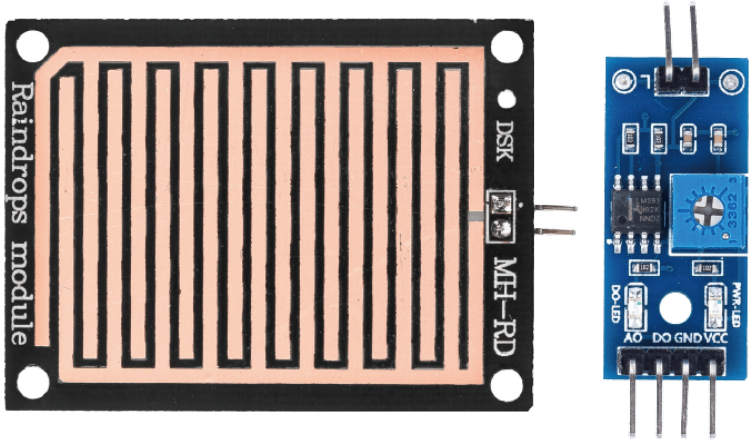
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.16 Raindrop Detection Module



The Raindrop Detection Sensor Module is a weather sensor that detects rainfall presence and intensity. It includes a raindrop sensor board with printed tracks, usually paired with a comparator module. When raindrops hit the sensor board, they create a conductive path between tracks, changing the resistance. This change is then converted into an analog or digital signal to show the rainfall intensity.

Specification

- Supply Voltage: 3.3V - 5V
- PCB Size: 32 x 14mm
- Output Signal Type: DO and AO

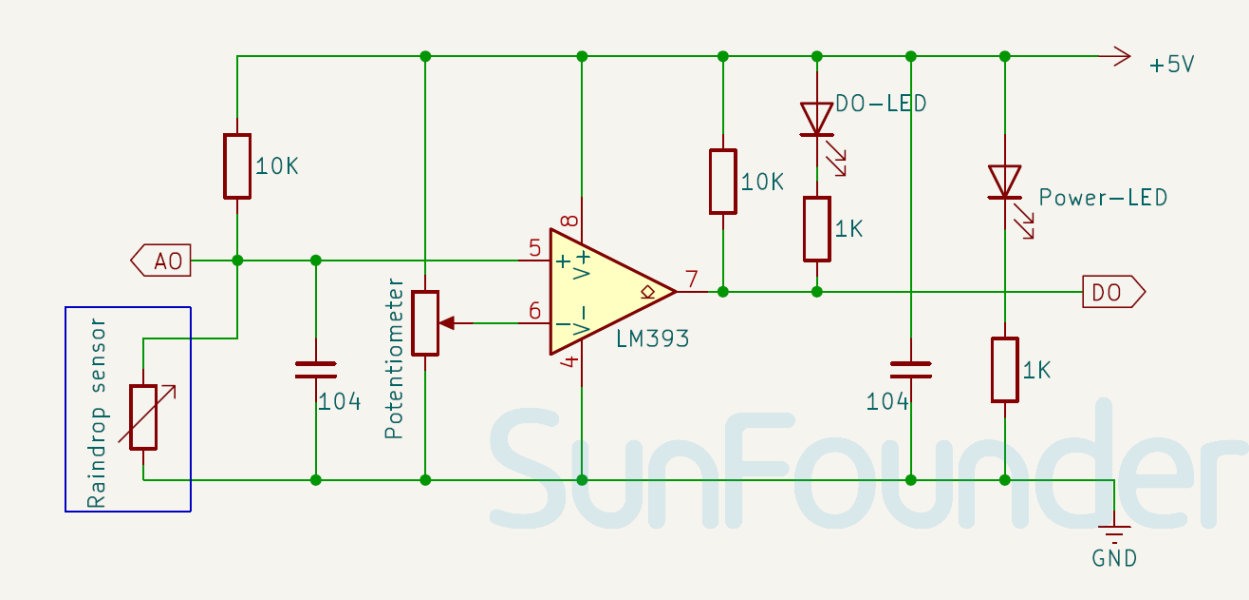
Pinout

- **VCC**: This is the positive power supply input from the main control.
- **GND**: Ground connection.
- **DO**: Digital output. Outputs a low level when raindrops are detected, and a high level when dry.
- **AO**: Analog output. The more rainwater, the smaller the analog output value.

Principle

Raindrop sensor is basically a board on which nickel is coated in the form of lines. It works on the principal of resistance. When there is no rain drop on board. Resistance is high so we gets high voltage according to V=IR. When rain drop present it reduces the resistance because water is conductor of electricity and presence of water connects nickel lines in parallel so reduced resistance and reduced voltage drop across it. The more intense the rainfall the lower the resistance.

Schematic diagram



Example

- *Lesson 15: Raindrop Detection Module* (Arduino UNO)
- *Lesson 15: Raindrop Detection Module* (ESP32)
- *Lesson 15: Raindrop Detection Module* (Raspberry Pi Pico)
- *Lesson 15: Raindrop Detection Module* (Raspberry Pi)

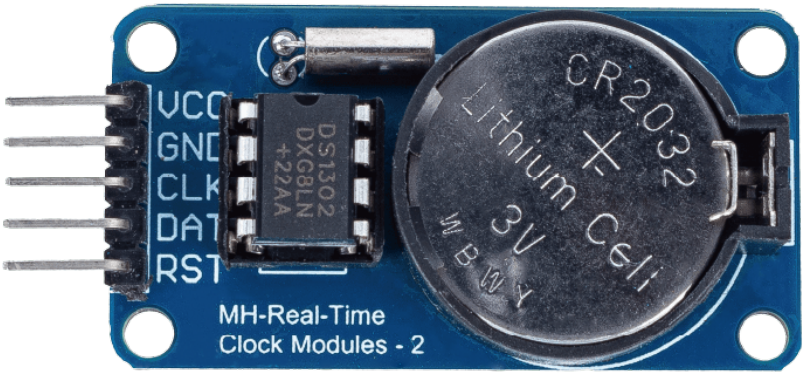
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.17 Real Time Clock Module (DS1302)



The DS1302 module is a Real-Time Clock (RTC) module that can track years, months, days, weekdays, hours, minutes, and seconds. It also has the ability to adjust for leap years. It is useful for creating projects requiring precise timing and scheduling.

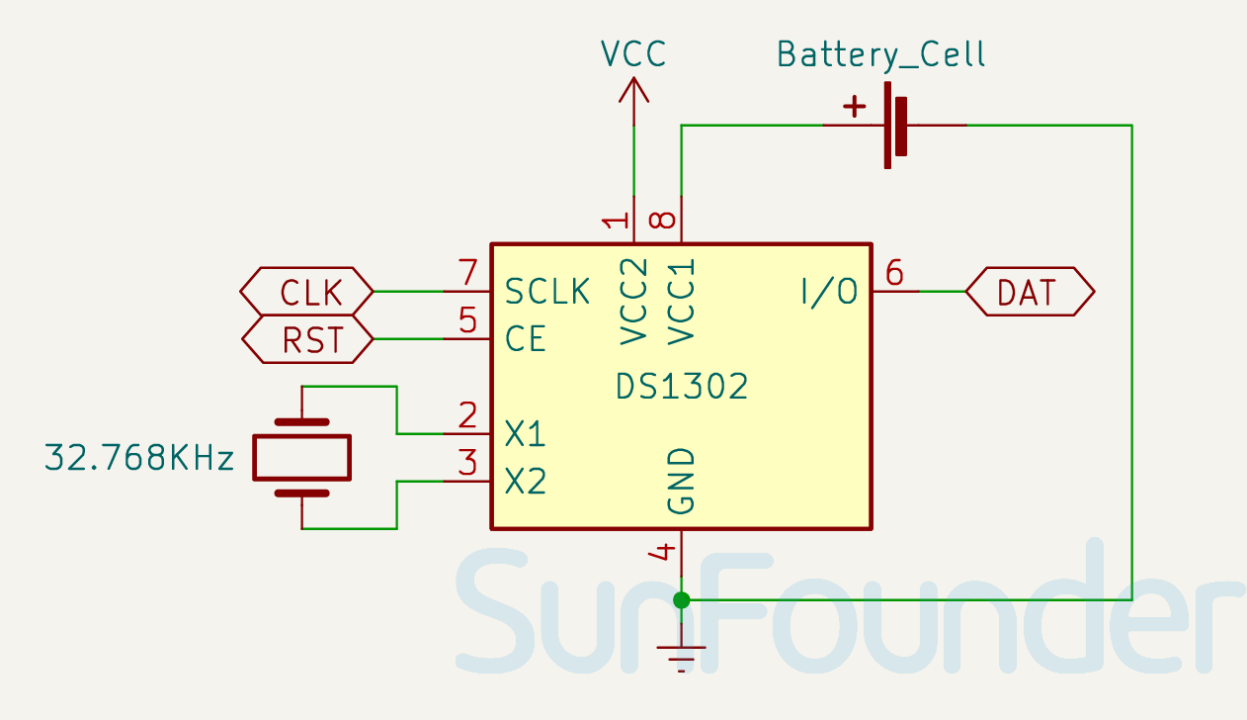
Specification

- Supply Voltage: 3.3V - 5V
- PCB Size: 44 x 23mm
- Clock IC: DS1302
- Operating temperature: 0℃ - 70℃

Pinout

- **VCC**: Module power supply
- **GND**: Ground
- **CLK**: Clock pin
- **DAT**: Data pin
- **RST**: Reset pin

Schematic diagram



Example

- *Lesson 16: Real Time Clock Module (DS1302)* (Arduino UNO)
- *Lesson 16: Real Time Clock Module (DS1302)* (ESP32)
- *Lesson 16: Real Time Clock Module (DS1302)* (Raspberry Pi Pico)

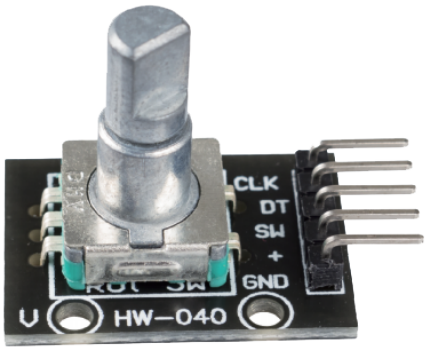
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.18 Rotary Encoder Module



A rotary encoder is a position sensor that converts the rotation of a knob into an output signal, indicating the direction in which the knob is turned.

Rotary encoders are digital versions of potentiometers, offering greater versatility. They can rotate continuously, while potentiometers have limited rotation. Potentiometers indicate exact knob position, while rotary encoders show changes in position.

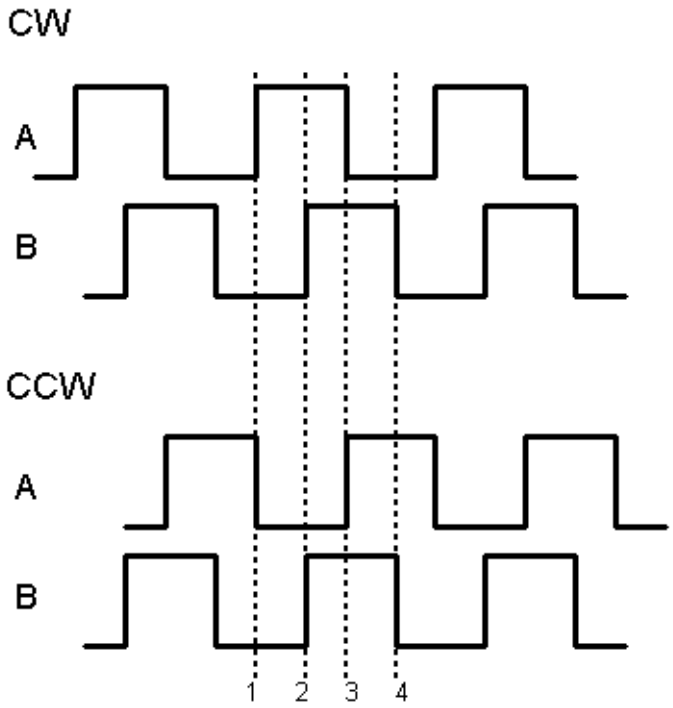
Pinout

- **VCC**: This is the positive power supply input from the main control.
- **GND**: Ground connection.
- **SW**: Digital output.
- **CLK**: is similar to CLK output, but it lags behind CLK by a 90° phase shift. This output is used to determine the direction of rotation.
- **DT**: is the primary output pulse used to determine the amount of rotation. Each time the knob is turned in either direction by just one detent (click), the 'CLK' output goes through one cycle of going HIGH and then LOW.

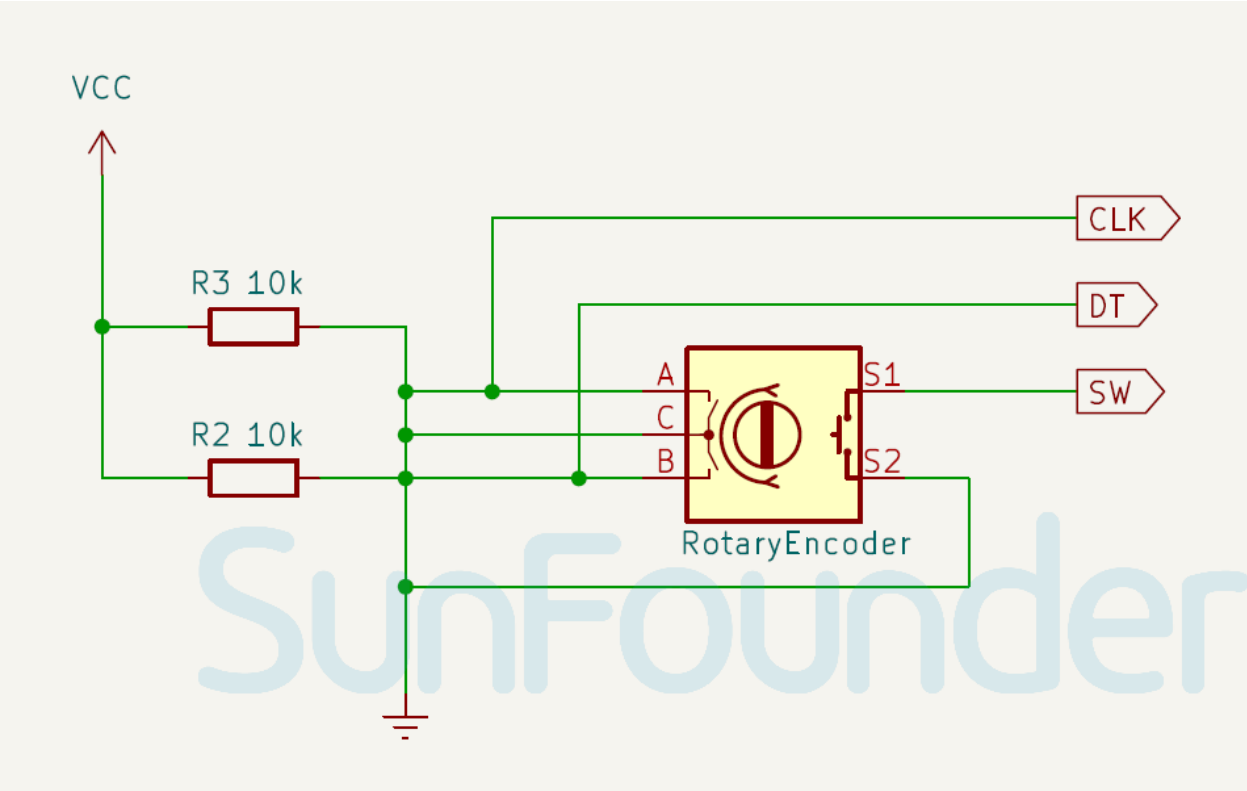
Principle

Incremental encoders produce two-phase square waves, with a 90-degree phase difference commonly referred to as the A and B channels.

As illustrated below, when channel A transitions from a high level to a low level, if channel B is at a high level, it indicates that the rotary encoder is rotating clockwise (CW); if at that moment channel B is at a low level, it means the rotation is counterclockwise (CCW). Therefore, by reading the value of channel B when channel A is at a low level, we can determine the direction in which the rotary encoder rotates.



Schematic diagram



Example

- *Lesson 17: Rotary Encoder Module* (Arduino UNO)
- *Lesson 17: Rotary Encoder Module* (ESP32)
- *Lesson 17: Rotary Encoder Module* (Raspberry Pi Pico)
- *Lesson 17: Rotary Encoder Module* (Raspberry Pi)

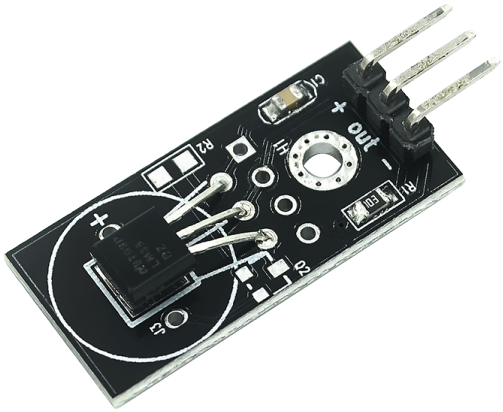
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.19 Temperature Sensor Module (DS18B20)



The DS18B20 is a digital temperature sensor that can measure temperatures ranging from -67°F to +257°F with an accuracy of ±0.5°C. It follows the single wire protocol and can communicate with a microcontroller using only one pin. The sensor can be powered directly from the data line, eliminating the need for an external power supply. The applications of the DS18B20 temperature sensor include industrial systems, consumer products, systems which are sensitive thermally, thermostatic controls, and thermometers.

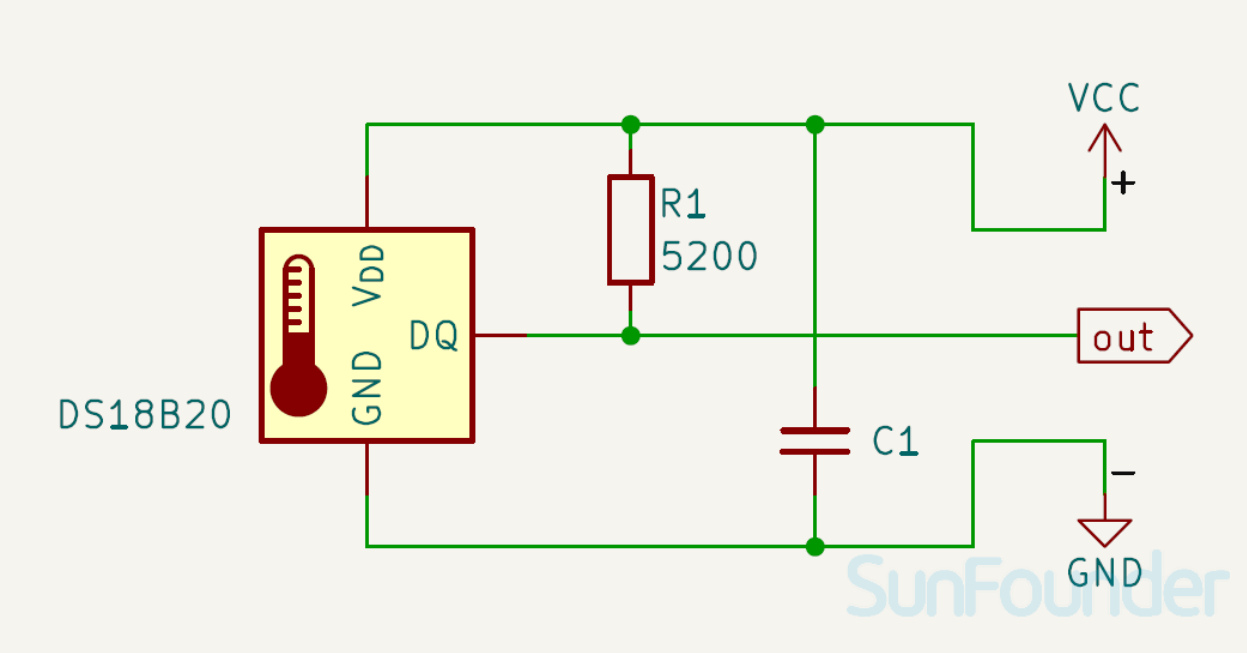
Specification

- PCB Size: 13 x 27.9mm
- Power Supply: 3V to 5.5V
- Temperature Range: -55 to 125°C
- Accuracy: ±0.5°C
- Resolution: 9 to 12 bit (selectable)

Pinout

- **VCC**: This is the positive power supply input from the main control.
- **GND**: Ground connection.
- **OUT**: The 1-Wire Data Bus that should be connected to a digital pin on the microcontroller.

Schematic diagram



Example

- *Lesson 18: Temperature Sensor Module (DS18B20)* (Arduino UNO)
- *Lesson 18: Temperature Sensor Module (DS18B20)* (ESP32)
- *Lesson 18: Temperature Sensor Module (DS18B20)* (Raspberry Pi Pico)
- *Lesson 18: Temperature Sensor Module (DS18B20)* (Raspberry Pi)

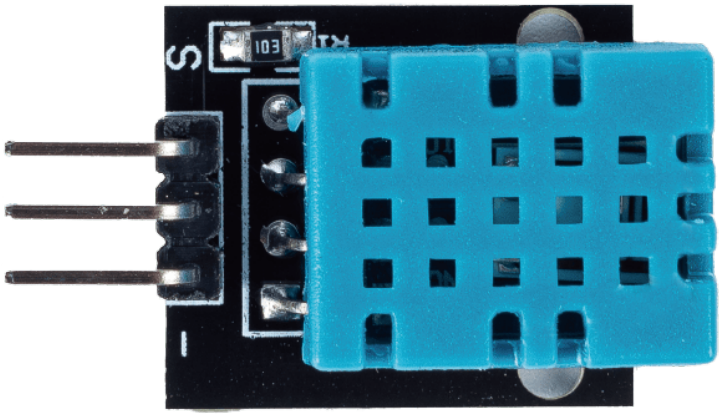
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.20 Temperature and Humidity Sensor Module (DHT11)



The digital temperature and humidity sensor DHT11 is a composite sensor that contains a calibrated digital signal output of temperature and humidity. The technology of a dedicated digital modules collection and the temperature and humidity sensing technology are applied to ensure that the product has high reliability and excellent long-term stability.

Specification

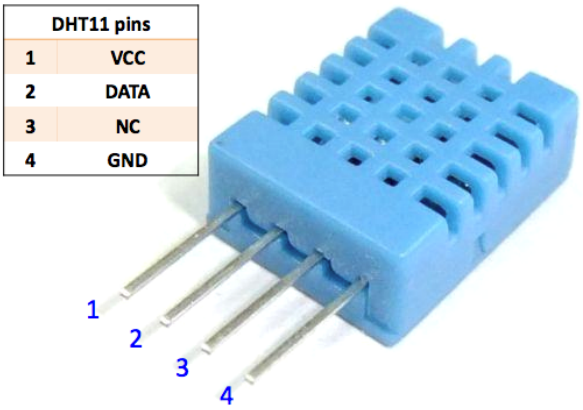
- Supply Voltage: 3.3V - 5V
- Output Signal Type: Digital output
- Temperature Measurement Range: 0-50℃ ± 2℃
- Humidity Measurement Range: 20-90%RH ± 5%RH
- Temperature Accuracy: ±2°C
- Humidity Accuracy: ±5% RH

Pinout

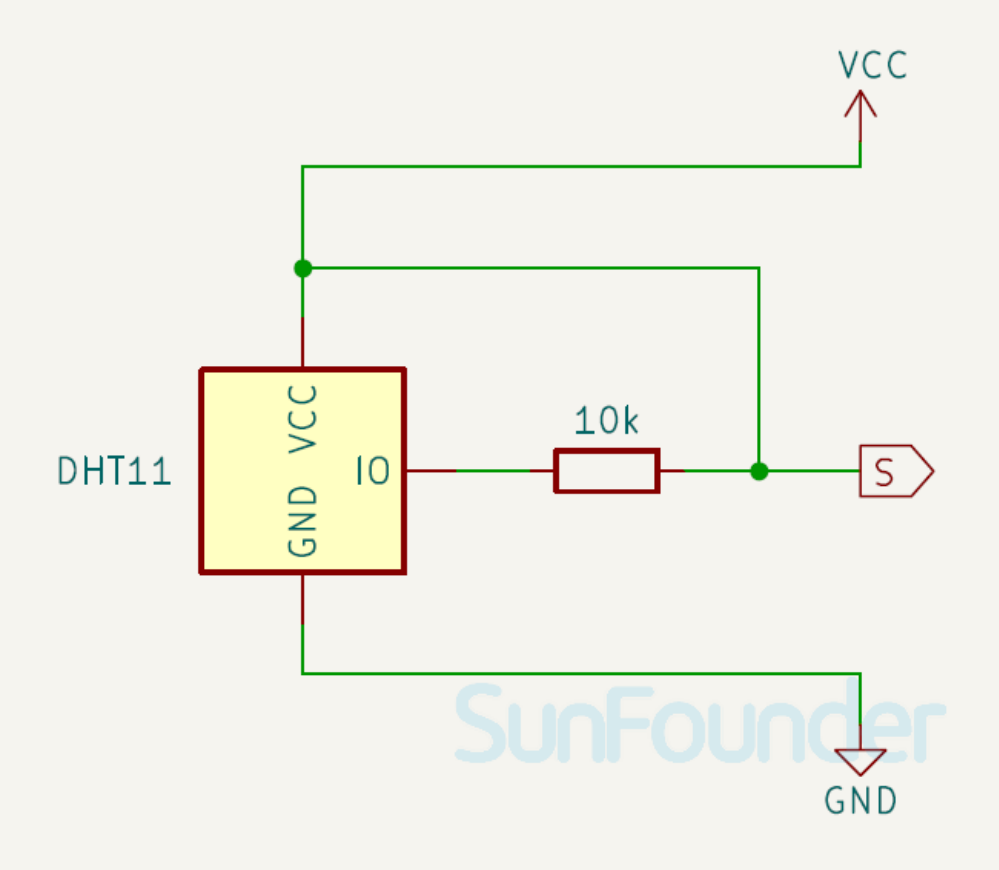
- **VCC**: This is the positive power supply input from the main control.
- **GND**: Ground connection.
- **S**: This pin is used for transmitting temperature and humidity data to the microcontroller using a single-wire bi-directional protocol.

Principle

Only three pins are available for use: VCC, GND, and DATA. The communication process begins with the DATA line sending start signals to DHT11, and DHT11 receives the signals and returns an answer signal. Then the host receives the answer signal and begins to receive 40-bit humidity and temperature data (8-bit humidity integer + 8-bit humidity decimal + 8-bit temperature integer + 8-bit temperature decimal + 8-bit checksum).



Schematic diagram



Example

- *Lesson 19: Temperature and Humidity Sensor Module (DHT11)* (Arduino UNO)
- *Lesson 19: Temperature and Humidity Sensor Module (DHT11)* (ESP32)
- *Lesson 19: Temperature and Humidity Sensor Module (DHT11)* (Raspberry Pi Pico)
- *Lesson 19: Temperature and Humidity Sensor Module (DHT11)* (Raspberry Pi)
- *Lesson 45: Plant Monitor* (Arduino UNO)
- *Lesson 43: Plant Monitor* (ESP32)
- *Lesson 48: Temperature and Humidity Monitoring with Adafruit IO* (ESP32)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

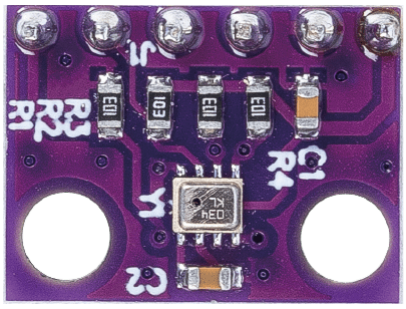
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.

- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.21 Temperature, Humidity & Pressure Sensor (BMP280)



The BMP280, developed by Bosch Sensortec, is a high-precision, low-power digital sensor module for measuring barometric pressure and temperature. It is widely used in mobile devices, weather monitoring, altitude estimations, and various other applications that require accurate atmospheric pressure and temperature data due to its small size and superior performance.

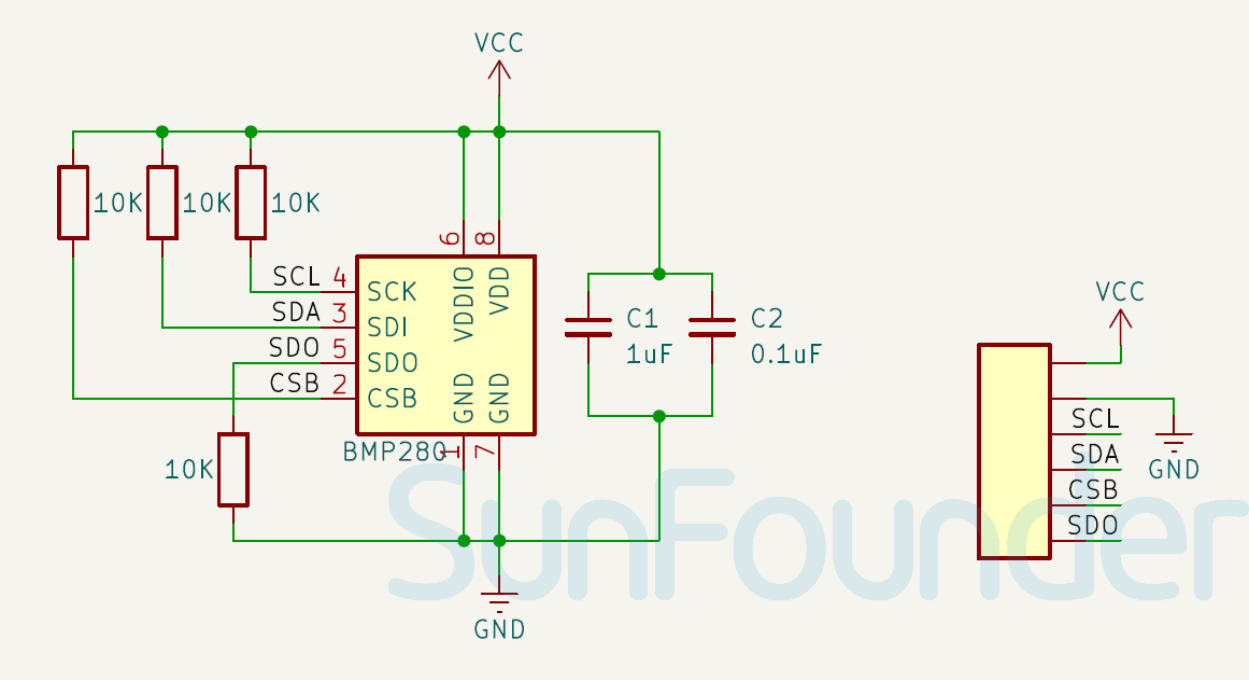
Specification

- Supply Voltage: 3.3V or 5V
- PCB size: 15 x 11mm
- Working temperature range: -40 ~ +85°C
- Air pressure measurement range: 300 ~ 1100hPa
- Interface: I2C (up to 3.4MHz), SPI (up to 10MHz)

Pinout

- **VCC**: This is the positive power supply input from the main control.
- **GND**: Ground connection.
- **SCL**: serial clock pin for the I2C interface.
- **SDA**: serial data pin for the I2C interface.
- **CSB**: the chip select pin of the module, if you are communicating with the device with SPI you can use this pin to communicate to select one if multiple devices are connected in the same bus.
- **SDO**: Serial Data out pin of the module. An output signal on a device where data is sent out to another SPI device.

Schematic diagram



Example

- *Lesson 20: Temperature, Humidity & Pressure Sensor (BMP280)* (Arduino UNO)
- *Lesson 20: Temperature, Humidity & Pressure Sensor (BMP280)* (ESP32)
- *Lesson 20: Temperature, Humidity & Pressure Sensor (BMP280)* (Raspberry Pi Pico)
- *Lesson 20: Temperature, Humidity & Pressure Sensor (BMP280)* (Raspberry Pi)
- *Lesson 48: Weather Monitor with ThingSpeak* (Arduino UNO)

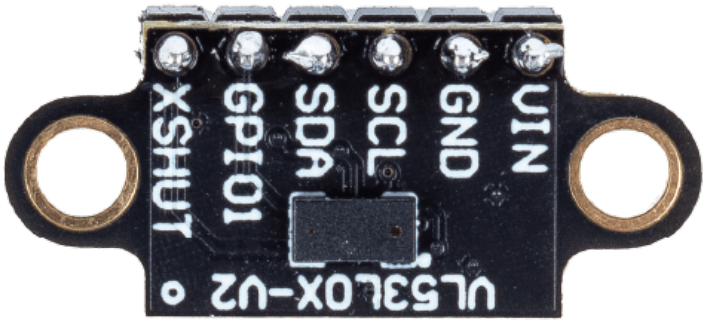
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.22 Time of Flight Micro-LIDAR Distance Sensor (VL53L0X)



The VL53L0X module is an advanced time-of-flight (ToF) ranging sensor that offers highly accurate distance measurement, regardless of the target's color and reflectance. Manufactured by STMicroelectronics, this sensor excels in measuring absolute distances up to 2 meters, making it well-suited for various applications in fields such as robotics, drones, and wearable devices.

Specification

- Supply Voltage: 3.3V or 5V
- PCB Size: 11 x 25mm
- Communication method: I2C
- ToF ranging length: 2M

Pinout

- **VIN**: This is the power pin.
- **GND**: Common ground for power and logic.
- **SCL**: I2C clock pin, connect to your microcontrollers I2C clock line.
- **SDA**: I2C data pin, connect to your microcontrollers I2C data line.
- **GPIO1**: Programmable interrupt output. This output is not level-shifted.
- **XSHUT**: This pin is an active-low shutdown input; Driving this pin low puts the sensor into hardware standby. This input is not level-shifted.

Example

- *Lesson 21: Time of Flight Micro-LIDAR Distance Sensor (VL53L0X)* (Arduino UNO)
- *Lesson 21: Time of Flight Micro-LIDAR Distance Sensor (VL53L0X)* (ESP32)
- *Lesson 21: Time of Flight Micro-LIDAR Distance Sensor (VL53L0X)* (Raspberry Pi Pico)
- *Lesson 21: Time of Flight Micro-LIDAR Distance Sensor (VL53L0X)* (Raspberry Pi)

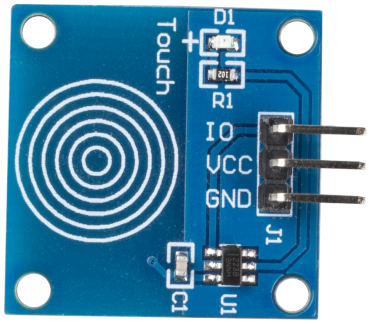
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.23 Touch Sensor Module



The Touch switch sensor (also called touch button or touch switch) is widely used to control devices (e.g. touchable lamp). It has the same functionality as a button. It is used instead of the button on many new devices because it makes the product look neat.

Pinout

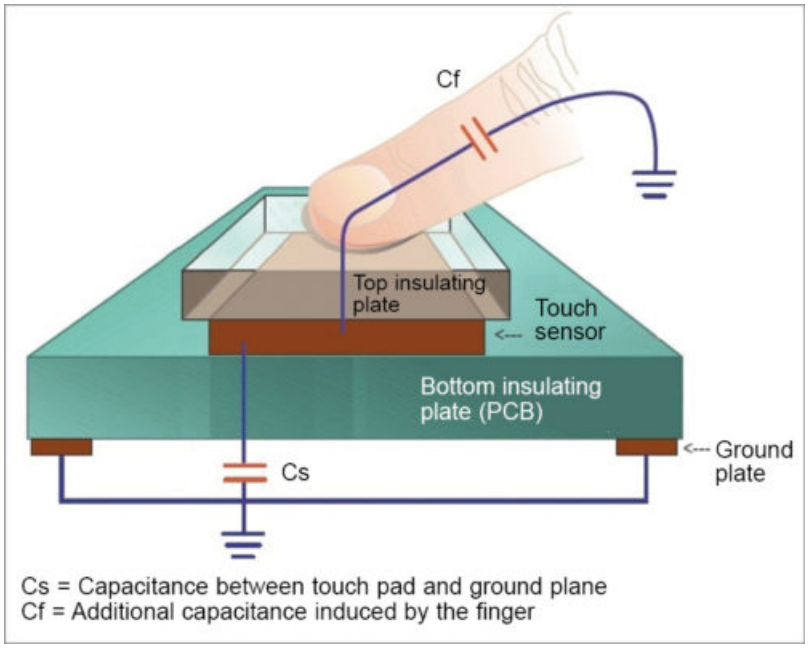
- **VCC**: This is the positive power supply input from the main control.
- **GND**: Ground connection.
- **IO**: Digital output. High level with a touch, low level without touching.

Principle

This module is a capacitive touch switch module based on a touch sensor IC (TTP223B). In the normal state, the module outputs a low level with low power consumption; when a finger touches the corresponding position, the module outputs a high level and becomes low level again after the finger is released.

Here is how the capacitive touch switch works:

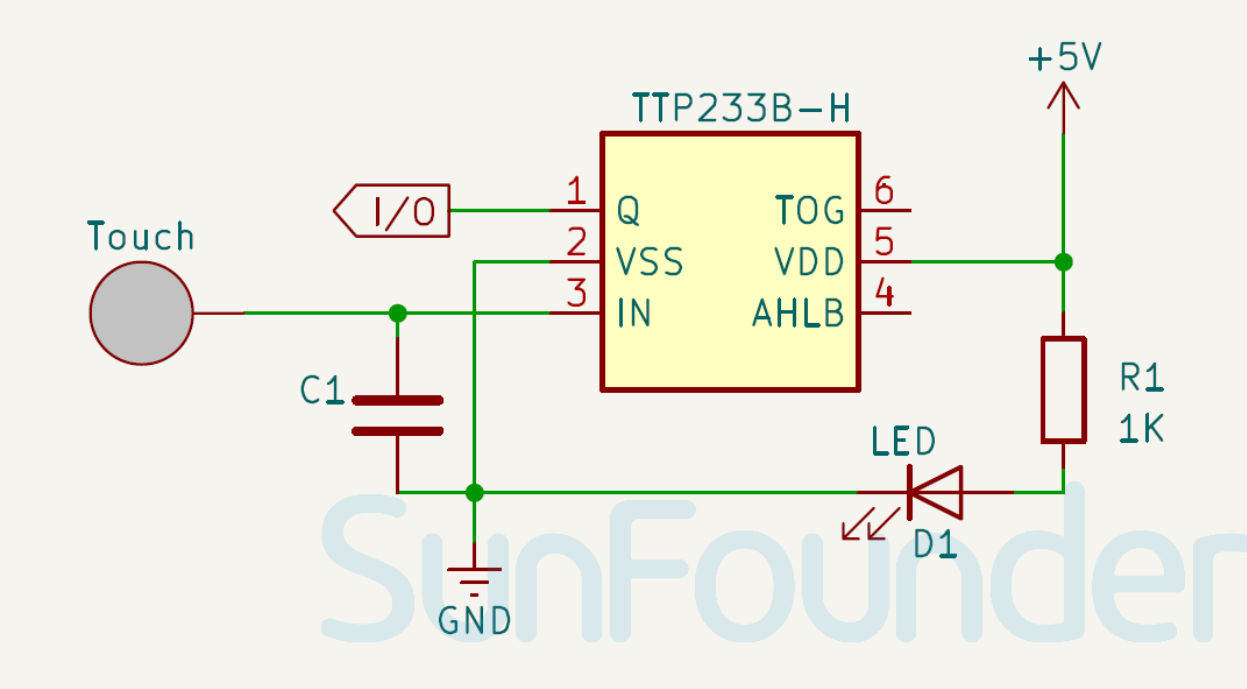
A capacitive touch switch has different layers—top insulating face plate followed by touch plate, another insulating layer and then ground plate.



In practice, a capacitive sensor can be made on a double-sided PCB by regarding one side as the touch sensor and the opposite side as ground plate of the capacitor. When power is applied across these plates, the two plates get charged. In equilibrium state, the plates have the same voltage as the power source.

The touch detector circuit has an oscillator whose frequency is dependent on capacitance of the touchpad. When a finger is moved close to the touchpad, additional capacitance causes frequency of this internal oscillator to change. The detector circuit tracks oscillator frequency at timed intervals, and when the shift crosses the threshold change, the circuit triggers a key-press event.

Schematic diagram



Example

- *Lesson 22: Touch Sensor Module* (Arduino UNO)
- *Lesson 22: Touch Sensor Module* (ESP32)
- *Lesson 22: Touch Sensor Module* (Raspberry Pi Pico)
- *Lesson 22: Touch Sensor Module* (Raspberry Pi)
- *Lesson 42: Touch toggle light* (Arduino UNO)
- *Lesson 40: Touch toggle light* (ESP32)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.24 Ultrasonic Sensor Module (HC-SR04)

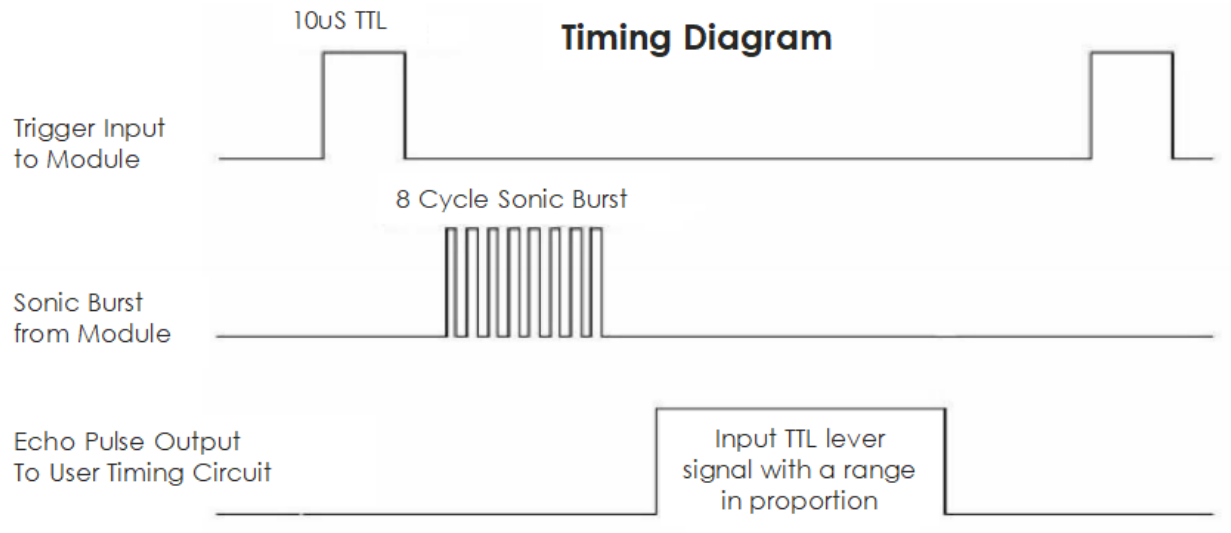
The Ultrasonic Module(HC-SR04) is a sensor that can measure distances between 2cm and 400cm using ultrasonic waves. It is commonly used in robotics and automation projects to detect objects and measure distances. The module consists of an ultrasonic transmitter and receiver, which work together to send and receive ultrasonic waves.

Principle

The module includes ultrasonic transmitters, receiver and control circuit. The basic principles are as follows:

1. Use an IO flip-flop to process a high level signal of at least 10us.
2. The module automatically sends eight 40khz and detects if there is a pulse signal return.
3. If the signal returns, passing the high level, the high output IO duration is the time from the transmission of the ultrasonic wave to the return of it. Here, test distance = (high time x sound speed (340 m / s) / 2.

The timing diagram is shown below.



You only need to supply a short 10us pulse for the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. You can calculate the range through the time interval between sending trigger signal and receiving echo signal.

Note: It is recommended to use measurement cycle over 60ms in order to prevent signal collisions of trigger signal and the echo signal.

Formula:

- us / 58 = centimeters
- us / 148 = inch
- distance = high level time * speed of sound (340m/s) / 2;

Example

- *Lesson 23: Ultrasonic Sensor Module (HC-SR04)* (Arduino UNO)
- *Lesson 23: Ultrasonic Sensor Module (HC-SR04)* (ESP32)
- *Lesson 23: Ultrasonic Sensor Module (HC-SR04)* (Raspberry Pi Pico)
- *Lesson 23: Ultrasonic Sensor Module (HC-SR04)* (Raspberry Pi)
- *Lesson 37: Smart trashcan* (Arduino UNO)
- *Lesson 35: Smart trashcan* (ESP32)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.25 Vibration Sensor Module (SW-420)

The SW-420 vibration sensor is a module that detects vibrations or shocks on a surface. It can be used for various purposes, such as detecting door knocks, machine malfunctions, car collisions, or alarm systems. It operates within a voltage range of 3.3 V to 5 V and has three peripherals: two LEDs (one for power status and the other for sensor output) and a potentiometer that can be used to control the vibration threshold point.

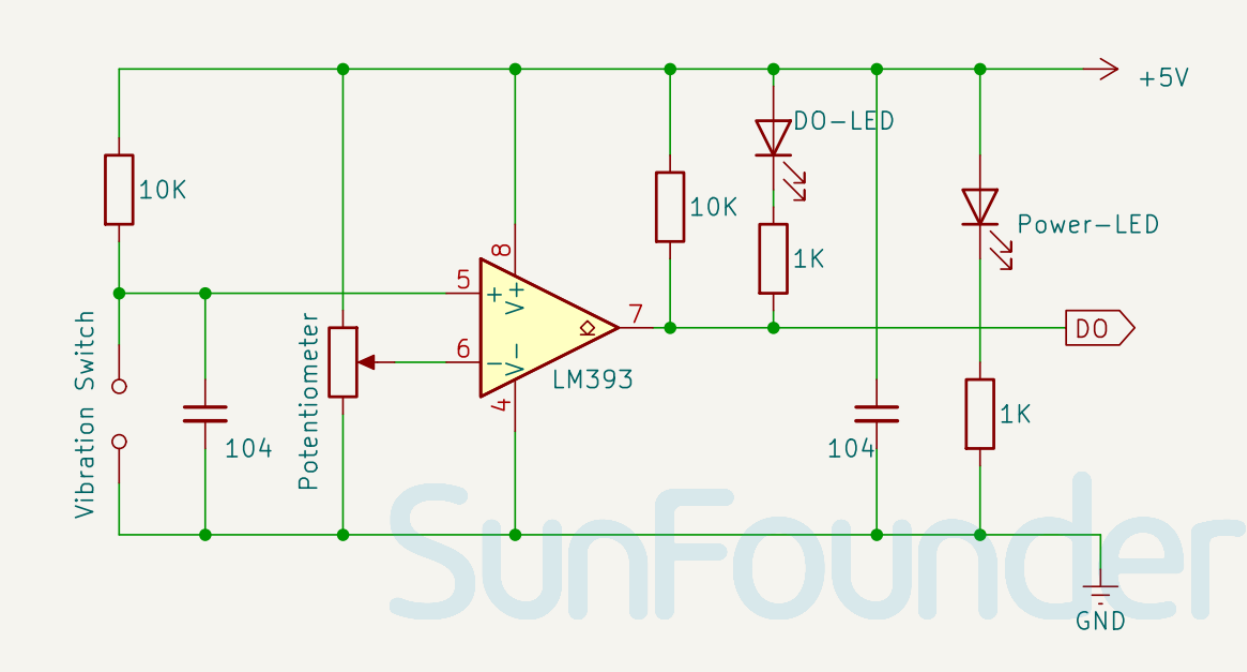
Pinout

- **VCC**: This is the positive power supply input from the main control.
- **GND**: Ground connection.
- **DO**: Digital output. During normal operation, the sensor outputs a Logic Low signal. When vibration is detected, the sensor outputs a Logic High signal.

Principle

SW-420 vibration sensor module consists of a SW-420 vibration switch and an LM393 voltage comparator. A SW-420 vibration switch is a device that has a spring and a rod inside a tube. When the switch is exposed to a vibration, the spring touches the rod and closes the circuit. The vibration sensor in the module detects these oscillations and converts them into electrical signals. The LM393 comparator chip then compares these signals with a reference voltage set by the potentiometer. If the amplitude of the signal exceeds this reference voltage, the output of the comparator goes high (1), otherwise it goes low (0).

Schematic diagram



Example

- *Lesson 24: Vibration Sensor Module (SW-420)* (Arduino UNO)
- *Lesson 24: Vibration Sensor Module (SW-420)* (ESP32)
- *Lesson 24: Vibration Sensor Module (SW-420)* (Raspberry Pi Pico)
- *Lesson 24: Vibration Sensor Module (SW-420)* (Raspberry Pi)
- *Lesson 44: Digital Dice* (Arduino UNO)
- *Lesson 49: Vibration Alert System with IFTTT* (Arduino UNO)

- *Lesson 42: Digital Dice* (ESP32)

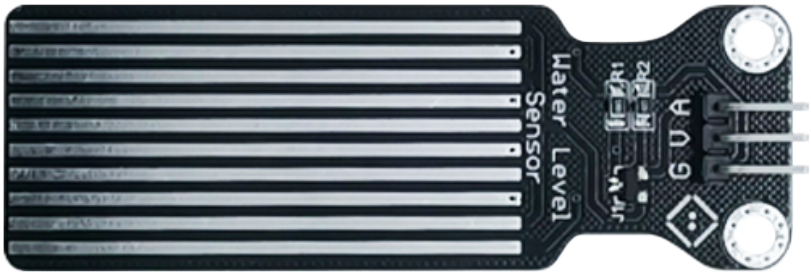
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.26 Water Level Sensor Module



The water level sensor is an affordable, user-friendly device that is compact and lightweight. It uses exposed parallel wire traces to measure the size of water droplets or volume, thus determining the water level. This sensor effortlessly converts water levels into analog signals, which can be readily utilized by program functions for triggering water level alarms. Its low power consumption and high sensitivity are also notable features.

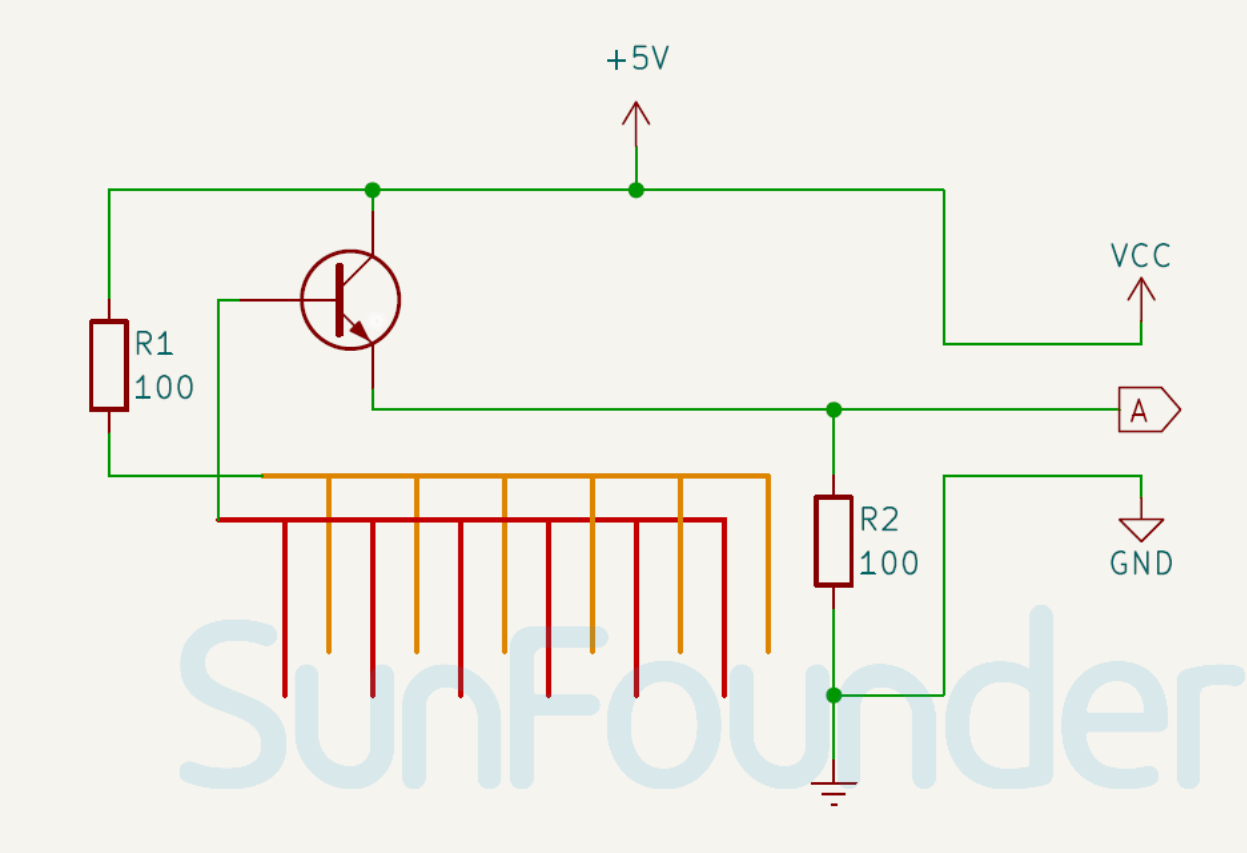
Specification

- Supply Voltage: 3.3V or 5V
- PCB size: 22 x 60mm
- Working temperature range: 10℃ - 30℃
- Working humidity range: 10% - 90%

Pinout

- **V**: This is the positive power supply input from the main control.
- **G**: Ground connection.
- **A**: Analog output. The higher the water level, the greater the output voltage.

Schematic diagram



Example

- *Lesson 25: Water Level Sensor Module* (Arduino UNO)
- *Lesson 25: Water Level Sensor Module* (ESP32)
- *Lesson 25: Water Level Sensor Module* (Raspberry Pi Pico)
- *Lesson 25: Water Level Sensor Module* (Raspberry Pi)

Display

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

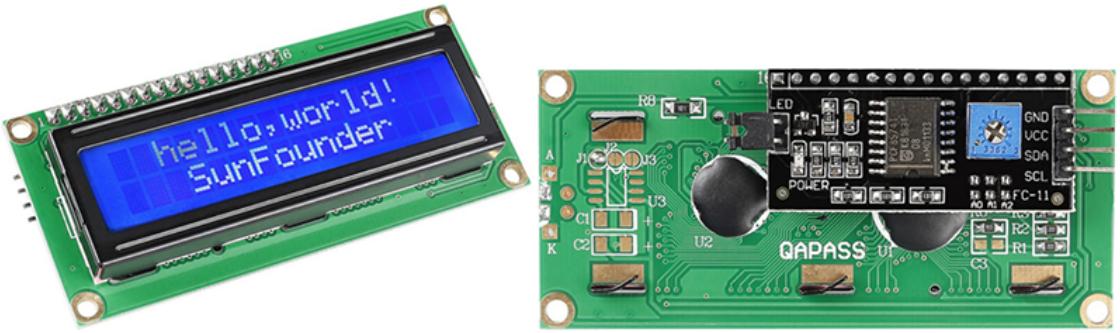
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.

- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.27 I2C LCD 1602



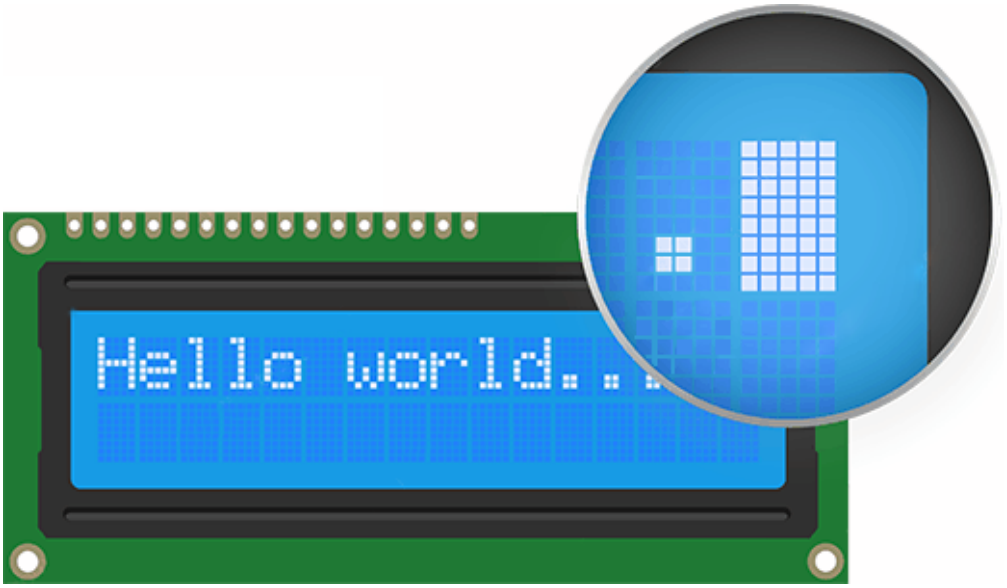
An I2C LCD1602 is a device that can display text and characters on a 16x2 (16 columns and 2 rows) liquid crystal display (LCD) using the I2C protocol. You can use an I2C LCD1602 to show information from your Arduino projects, such as sensor readings, messages, menus, etc. The I2C module has a built-in PCF8574 I2C chip that converts I2C serial data to parallel data for the LCD display.

-

Principle

An I2C LCD1602 consists of a normal LCD1602 and an I2C module that is attached to the back of the LCD. The I2C module is a chip that can expand the I/O ports of the Arduino using the I2C protocol. The I2C protocol is a serial communication protocol that uses two wires: SDA (serial data) and SCL (serial clock). The I2C protocol allows multiple devices to communicate with each other using only two wires and unique addresses.

The I2C module converts the signals from the Arduino into commands for the LCD. The LCD has 16x2 cells that can display characters or symbols. Each cell consists of 5x8 dots that can be turned on or off by applying voltage. The LCD can display different characters or symbols by turning on or off different combinations of dots.



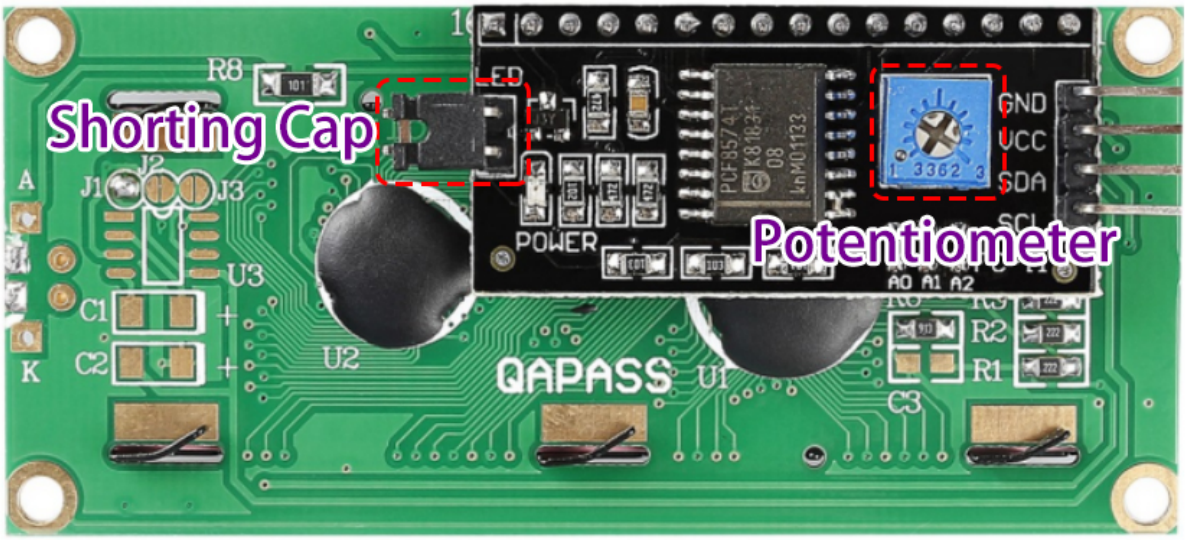
I2C Address

The default address is basically 0x27, in a few cases it may be 0x3F.

Taking the default address of 0x27 as an example, the device address can be modified by shorting the A0/A1/A2 pads; in the default state, A0/A1/A2 is 1, and if the pad is shorted, A0/A1/A2 is 0.

Backlight/Contrast

Backlight can be enabled by jumper cap, unplugg the jumper cap to disable the backlight. The blue potentiometer on the back is used to adjust the contrast (the ratio of brightness between the brightest white and the darkest black).



- **Shorting Cap**: Backlight can be enabled by this capunplugg this cap to disable the backlight.
- **Potentiometer**: It is used to adjust the contrast (the clarity of the displayed text), which is increased in the clockwise direction and decreased in the counterclockwise direction.

Note: After wiring the LCD, you should turn on the Arduino and adjust the contrast by rotating the potentiometer on the I2C module until the first row of rectangles appear to ensure proper LCD operation.

Example

- *Lesson 26: I2C LCD 1602* (Arduino UNO)
- *Lesson 26: I2C LCD 1602* (ESP32)
- *Lesson 26: I2C LCD 1602* (Raspberry Pi Pico)
- *Lesson 26: I2C LCD 1602* (Raspberry Pi)
- *Lesson 43: Potentiometer scale value* (Arduino UNO)
- *Lesson 45: Plant Monitor* (Arduino UNO)
- *Lesson 46: Bluetooth LCD* (Arduino UNO)
- *Lesson 41: Potentiometer scale value* (ESP32)
- *Lesson 43: Plant Monitor* (ESP32)
- *Lesson 46: Real-time Weather From @OpenWeatherMap* (ESP32)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.

- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.28 OLED Display Module (SSD1306)

An OLED (Organic Light-Emitting Diode) display module is a device that can display text, graphics and images on a thin and flexible screen using organic materials that emit light when electric current is applied.

The SSD1306 I2C OLED display module operates by controlling an OLED (Organic Light-Emitting Diode) display using the powerful single-chip CMOS OLED driver controller, the SSD1306. This controller manages all RAM buffering and demands minimal effort from the connected microcontroller, such as an Arduino. OLED displays are renowned for their extremely light and potentially flexible nature, producing brighter and crisper images compared to traditional displays due to being almost paper-thin.

The main advantage of an OLED Display is that it emits its own light and doesn't need another source of backlight. Due to this, OLED Displays often have better contrast, brightness and viewing angles when compared to LCD displays.

Another important feature of OLED Displays is deep black levels. Since each pixel emits its own light in an OLED Display, to produce black color, the individual pixel can be turned OFF.

Due to lower power consumption (only pixels which are lit up draw current), OLED displays are also popular in battery operated devices like Smart Watches, Health Trackers and other wearables.

Pinout

- **VIN**: This is the power pin.
- **GND**: Common ground for power and logic.
- **SCL**: The serial clock pin for the I2C interface.
- **SDA**: The serial data pin for the I2C interface.

Example

- *Lesson 27: OLED Display Module (SSD1306)* (Arduino UNO)
- *Lesson 27: OLED Display Module (SSD1306)* (ESP32)
- *Lesson 27: OLED Display Module (SSD1306)* (Raspberry Pi Pico)
- *Lesson 27: OLED Display Module (SSD1306)* (Raspberry Pi)
- *Lesson 41: Heart rate monitor* (Arduino UNO)
- *Lesson 44: Digital Dice* (Arduino UNO)
- *Lesson 52: Tilt Direction Indicator* (Arduino UNO)
- *Lesson 53: Direction Indicator* (Arduino UNO)
- *Lesson 39: Heart rate monitor* (ESP32)
- *Lesson 42: Digital Dice* (ESP32)

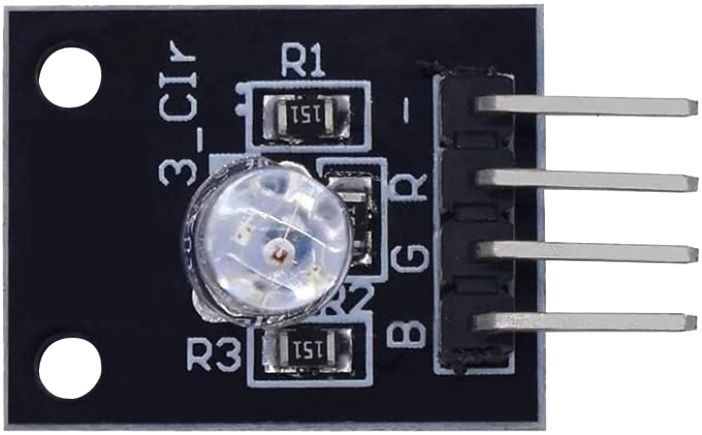
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.29 RGB LED Module



The RGB Full Color LED module emits a range of colors by mixing red, green, and blue light. Each color is adjusted by using PWM.It can be used to create colorful lighting effects or to learn how to use PWM (pulse-width modulation) with Arduino.

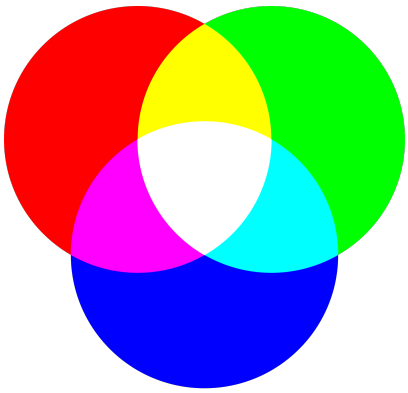
Pinout

- **GND**: Common ground for power.
- **B**: Controls the brightness of the red LED. By adjusting the current flowing through this pin, the intensity of the red light can be varied.
- **R**: Controls the brightness of the green LED. Similarly to the red pin, adjusting the current flow through this pin changes the intensity of the green light.
- **G**: Controls the brightness of the blue LED. By adjusting the current flowing through this pin, the intensity of the blue light can be altered.

Principle

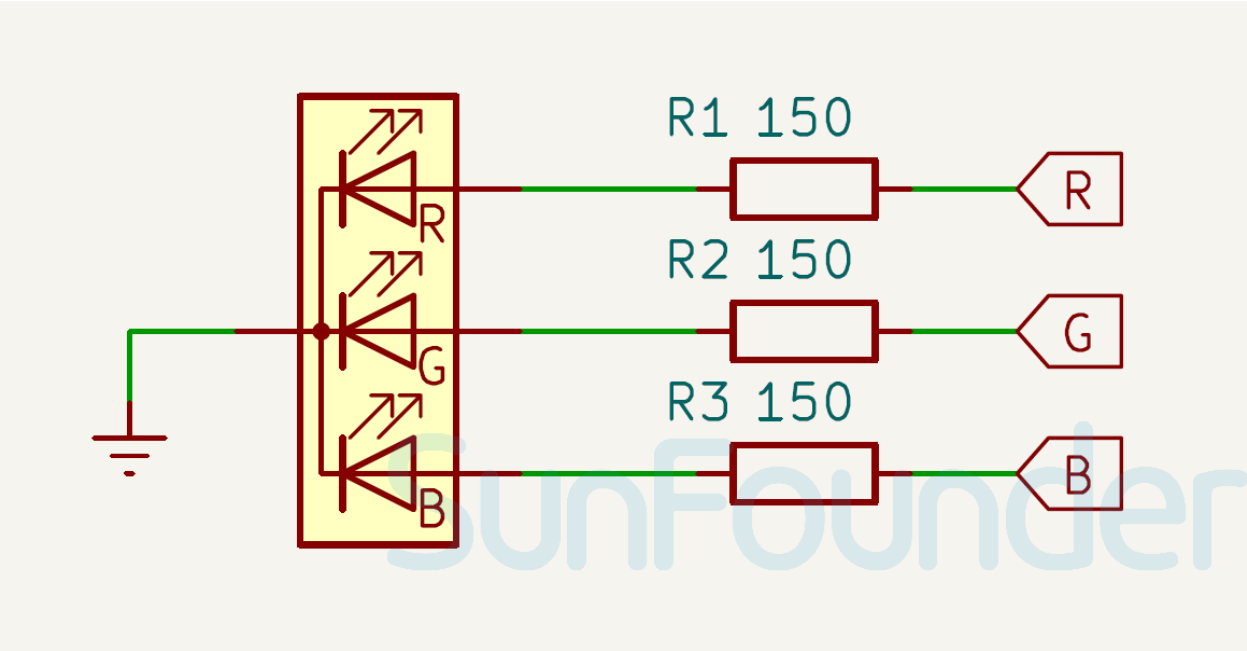
The RGB MODULE works by using a full-color LED that uses R, G, and B pins with adjustable PWM voltage input. Colors from the LED can be combined. For example, mix blue light and green light give cyan light, red light and green light give yellow light. This is called "The additive method of color mixing".

- Additive color - Wikipedia



Based on this method, we can use the three primary colors to mix the visible light of any color according to different proportions. For example, orange can be produced by more red and less green. The strength of the primary colors (red, blue, green) is adjusted in order to achieve full color mixing effect.PWM is a technique where the duty cycle of a digital signal is modified, adjusting the percentage of time that the signal remains active within a given period. By changing the duty cycle, we can make the LED appear brighter or dimmer.

Schematic diagram



Example

- *Lesson 28: RGB LED Module* (Arduino UNO)
- *Lesson 28: RGB LED Module* (ESP32)
- *Lesson 28: RGB LED Module* (Raspberry Pi Pico)
- *Lesson 28: RGB Module* (Raspberry Pi)
- *Lesson 30: Relay Module* (ESP32)
- *Lesson 30: Relay Module* (Raspberry Pi Pico)
- *Lesson 30: Relay Module* (Raspberry Pi)
- *Lesson 38: Gas leak alarm* (Arduino UNO)
- *Lesson 40: Motion triggered relay* (Arduino UNO)
- *Lesson 36: Gas leak alarm* (ESP32)
- *Lesson 38: Motion triggered relay* (ESP32)
- *Lesson 45: Bluetooth Control RGB LED* (ESP32)
- *Lesson 47: IoT Communication with MQTT* (ESP32)
- *Lesson 48: Temperature and Humidity Monitoring with Adafruit IO* (ESP32)

- *Lesson 50: Android Application - RGB LED Operation via Arduino and Bluetooth* (ESP32)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

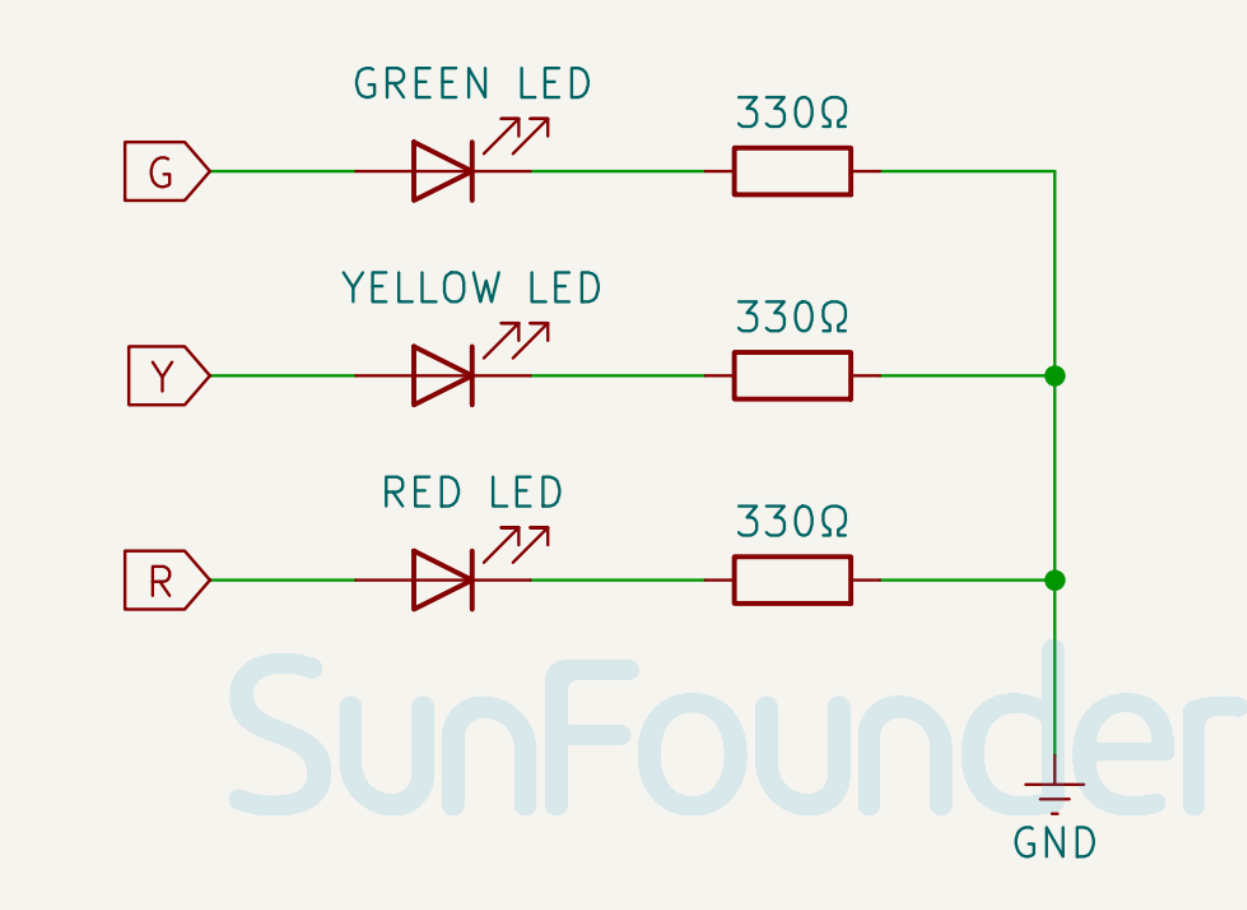
1.2.30 Traffic Light Module

The traffic light module is a small device that can display red, yellow and green lights, just like a real traffic light. It can be used to make a traffic light system model or to learn how to control LEDs with Arduino. It is featured with its small size, simple wiring, targeted, and custom installation. It can be connected PWM pin to control the brightness of the LED.

Principle

The traffic light module can be controlled in two primary ways. The more straightforward method involves using digital inputs from the Arduino, where a HIGH or LOW signal directly turns the corresponding LED on or off. Alternatively, PWM (pulse-width modulation) can be used, especially when varying the brightness of the LED is desired. PWM is a technique where the duty cycle of a digital signal is changed to modulate the brightness of the LED. A duty cycle represents the percentage of time that a signal remains on during a specific period. For instance, a 50% duty cycle implies the signal is active for half the duration and inactive for the remainder. Adjusting the duty cycle allows for the LED's brightness modulation.

Schematic diagram



Example

- *Lesson 29: Traffic Light Module* (Arduino UNO)
- *Lesson 29: Traffic Light Module* (ESP32)
- *Lesson 30: Relay Module* (Raspberry Pi Pico)
- *Lesson 30: Relay Module* (Raspberry Pi)
- *Lesson 30: Relay Module* (Arduino UNO)
- *Lesson 42: Touch toggle light* (Arduino UNO)
- *Lesson 47: Bluetooth Traffic Light* (Arduino UNO)
- *Lesson 40: Touch toggle light* (ESP32)

Actuator

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

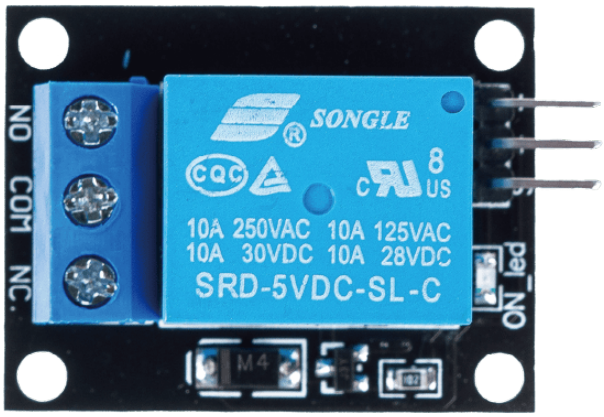
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.

- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.31 5V Relay Module

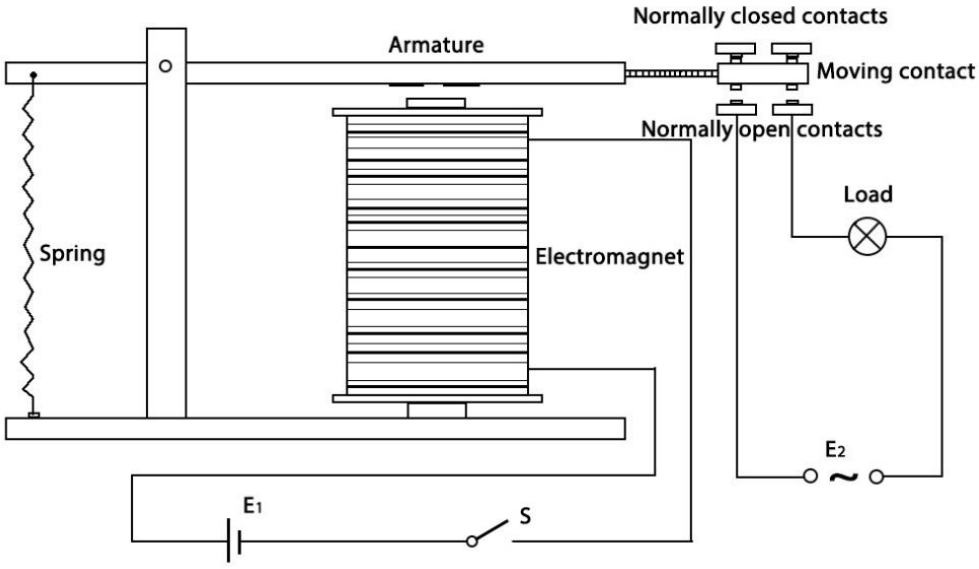


5V relay modules are devices that can switch high voltage or high current devices on and off using a 5V signal from Arduino. They can be used to control devices such as lights, fans, motors, solenoids, etc. 5V relay has three high voltage terminals (NC, C, and NO) which connect to the device you want to control. The other side has three low voltage pins (Ground, Vcc, and Signal) which connect to the Arduino.

Principle

A relay is a device which is used to provide connection between two or more points or devices in response to the input signal applied. In other words, relays provide isolation between the controller and devices, which may operate on either AC or DC. However, they receive signals from a microcontroller which works on DC hence requiring a relay to bridge the gap. Relay is extremely useful when you need to control a large amount of current or voltage with small electrical signal.

There are 5 parts in every relay:



Electromagnet - It consists of an iron core wounded by coil of wires. When electricity is passed through, it becomes magnetic. Therefore, it is called electromagnet.

Armature - The movable magnetic strip is known as armature. When current flows through them, the coil gets energized thus producing a magnetic field which is used to make or break the normally open (N/O) or normally close (N/C) points. And the armature can be moved with direct current (DC) as well as alternating current (AC).

Spring - When no currents flow through the coil on the electromagnet, the spring pulls the armature away so the circuit cannot be completed.

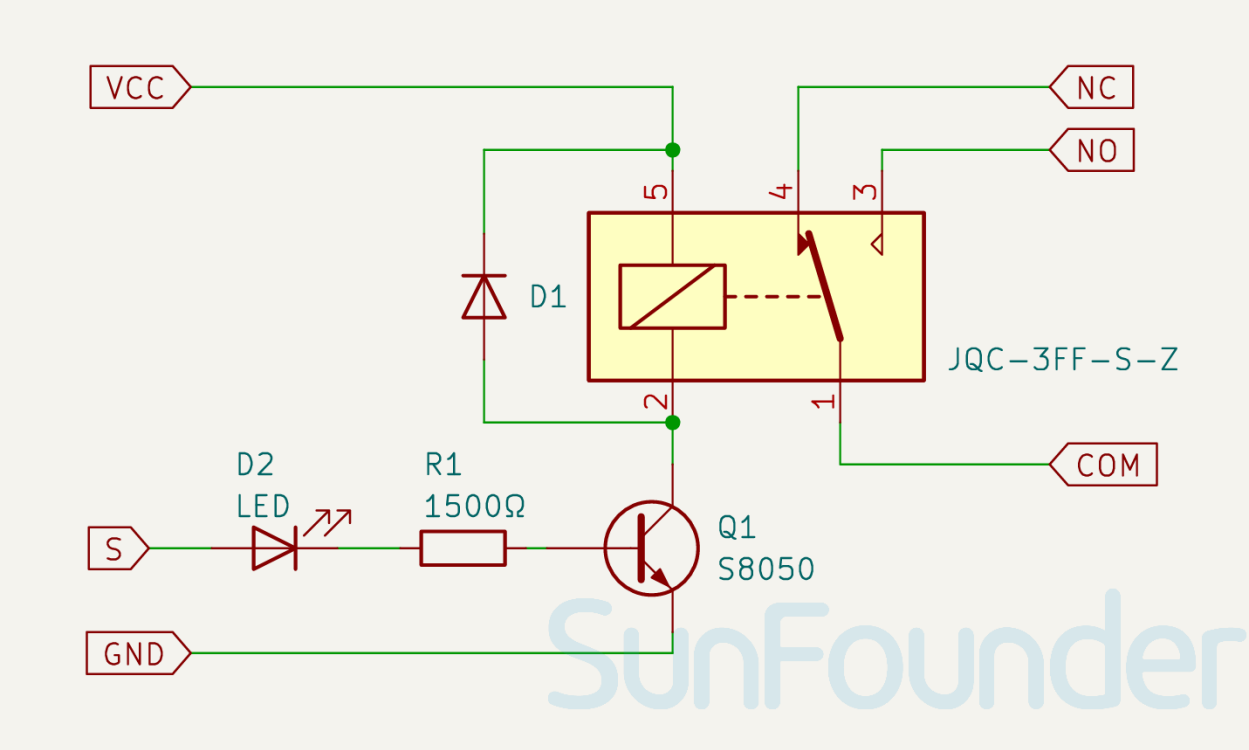
Set of electrical contacts - There are two contact points:

- Normally open - connected when the relay is activated, and disconnected when it is inactive.
- Normally closed - not connected when the relay is activated, and connected when it is inactive.

Molded frame - This is typically made of plastic and provides structural support and protection for the relay.

The working principle of relay is simple. When power is supplied to the relay, currents start flowing through the control coil; as a result, the electromagnet starts energizing. Then the armature is attracted to the coil, pulling down the moving contact together thus connecting with the normally open contacts. So the circuit with the load is energized. Then breaking the circuit would be a similar case, as the moving contact will be pulled up to the normally closed contacts under the force of the spring. In this way, the switching on and off of the relay can control the state of a load circuit.

Schematic diagram



Example

- *Lesson 30: Relay Module* (Arduino UNO)
- *Lesson 30: Relay Module* (ESP32)
- *Lesson 30: Relay Module* (Raspberry Pi Pico)
- *Lesson 30: Relay Module* (Raspberry Pi)
- *Lesson 40: Motion triggered relay* (Arduino UNO)
- *Lesson 38: Motion triggered relay* (ESP32)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.32 Centrifugal Pump

A centrifugal pump is a device that can move liquids from one place to another by using a rotating impeller. It can be used to pump water, oil, chemicals, etc. A centrifugal pump has two main parts: a motor and a pump. The motor provides power to the pump and the pump converts the rotational energy into pressure and flow.

Specification

- **Voltage Scope**: DC 3 ~ 4.5V
- **Operating Current**: 120 ~ 180mA
- **Power**: 0.36 ~ 0.91W
- **Max Water Head**: 0.35 ~ 0.55M
- **Max Flow Rate**: 80 ~ 100 L/H
- **Continuous Working Life**: 100 hours
- **Water Fing Grade**: IP68
- **Driving Mode**: DC, Magnetic Driving
- **Material**: Engineering Plastic
- **Outlet Outside Diameter**: 7.8 mm
- **Outlet Inside Diameter**: 6.5 mm
- It is a submersible pump and should be used that way. It tends to heat too much that there's a risk of overheating if you turn it on unsubmerged.

Example

- *Lesson 31: Centrifugal Pump* (Arduino UNO)
- *Lesson 31: Centrifugal Pump* (ESP32)
- *Lesson 31: Centrifugal Pump* (Raspberry Pi Pico)
- *Lesson 31: Centrifugal Pump* (Raspberry Pi)
- *Lesson 39: Automatic soap dispenser* (Arduino UNO)
- *Lesson 45: Plant Monitor* (Arduino UNO)
- *Lesson 37: Automatic soap dispenser* (ESP32)

- *Lesson 43: Plant Monitor* (ESP32)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.33 L9110 Motor Driver Module

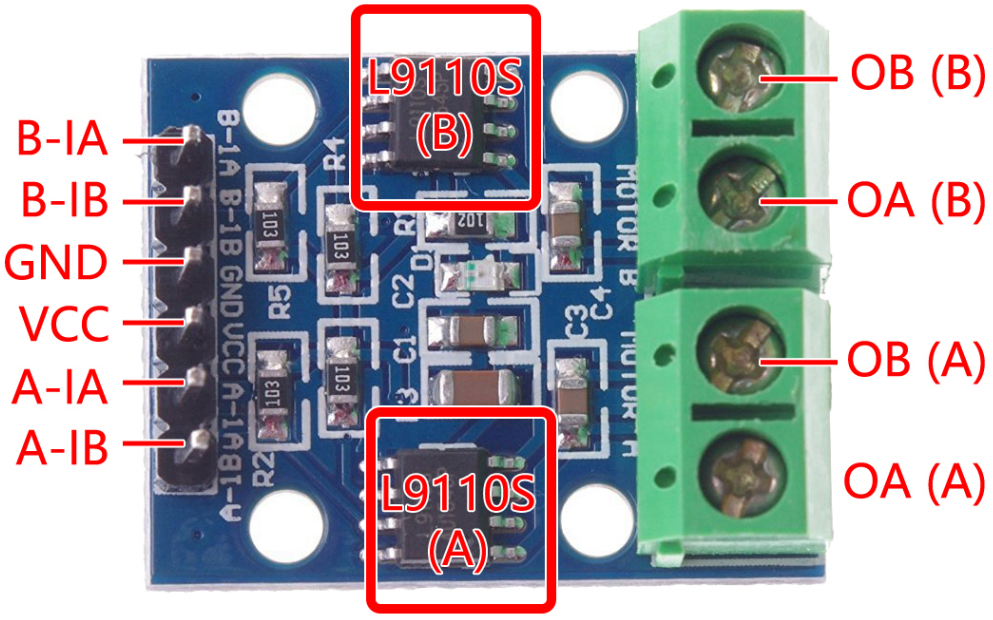
The L9110 motor driver module is adept at driving two motors in tandem. It houses a pair of independent L9110S driver chips, each channel boasting a steady current output of up to 800mA.

Spanning a voltage range from 2.5V to 12V, the module comfortably pairs with both 3.3V and 5V microcontrollers.

Serving as a streamlined solution, the L9110 motor driver module facilitates motor control across a spectrum of applications. Thanks to its dual-channel architecture, it enables the independent orchestration of two motors—ideal for projects where dual motor operations are paramount.

Given its potent continuous current output, this module confidently powers motors from the petite to the moderately sized, paving the way for diverse robotic, automation, and motor-centric endeavors. Its expansive voltage range further injects adaptability, aligning with varied power supply setups.

Designed with user-friendliness in mind, the module offers intuitive input and output terminals, simplifying connections to microcontrollers or akin control devices. Plus, it doesn't skimp on safety—integrated overcurrent and overtemperature safeguards bolster the trustworthiness and security of motor operations.



- **B-1A & B-1B(B-2A)**: Input pins for controlling the spinning direction of Motor B.
- **A-1A & A-1B**: Input pins for controlling the spinning direction of Motor A.
- **0A & OB(A)**: Output pins of Motor A.
- **0A & OB(B)**: Output pins of Motor B.
- **VCC**: Power input pin (2.5V-12V).
- **GND**: Ground pin.

Features

- On-board 2 L9110S motor control chip
- Dual-channel motor control.
- Independent motor spinning direction control.
- High current output (800mA per channel).
- Wide voltage range (2.5V-12V).
- Compact design.
- Convenient input and output terminals.
- Built-in protective features.
- Versatile applications.
- PCB Size: 29.2mm x 23mm
- Operating Temperature: -20°C ~ 80°C
- Power-On LED indicator

Operating Principle

Here is the truth table of Motor B:

This truth table shows the different states of Motor B based on the values of input pins B-1A and B-1B(B-2A). It indicates the direction of rotation (clockwise or counterclockwise), braking, or stopping of Motor B.

B-1A	B-1B(B-2A)	The state of Motor B
1	0	Rotate clockwise
0	1	Rotate counterclockwise
0	0	Brake
1	1	Stop

Here is the truth table of Motor A:

This truth table shows the different states of Motor A based on the values of input pins A-1A and A-1B. It indicates the direction of rotation (clockwise or counterclockwise), braking, or stopping of Motor A.

A-1A	A-1B	The state of Motor B
1	0	Rotate clockwise
0	1	Rotate counterclockwise
0	0	Brake
1	1	Stop

Example

- *Lesson 31: Centrifugal Pump* (Arduino UNO)
- *Lesson 31: Centrifugal Pump* (ESP32)
- *Lesson 31: Centrifugal Pump* (Raspberry Pi Pico)
- *Lesson 31: Centrifugal Pump* (Raspberry Pi)
- *Lesson 34: TT Motor* (Arduino UNO)
- *Lesson 34: TT Motor* (ESP32)
- *Lesson 34: TT Motor* (Raspberry Pi Pico)
- *Lesson 34: TT Motor* (Raspberry Pi)
- *Lesson 07: Infrared Speed Sensor Module* (Arduino UNO)
- *Lesson 07: Infrared Speed Sensor Module* (Raspberry Pi)
- *Lesson 39: Automatic soap dispenser* (Arduino UNO)
- *Lesson 45: Plant Monitor* (Arduino UNO)
- *Lesson 37: Automatic soap dispenser* (ESP32)
- *Lesson 43: Plant Monitor* (ESP32)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

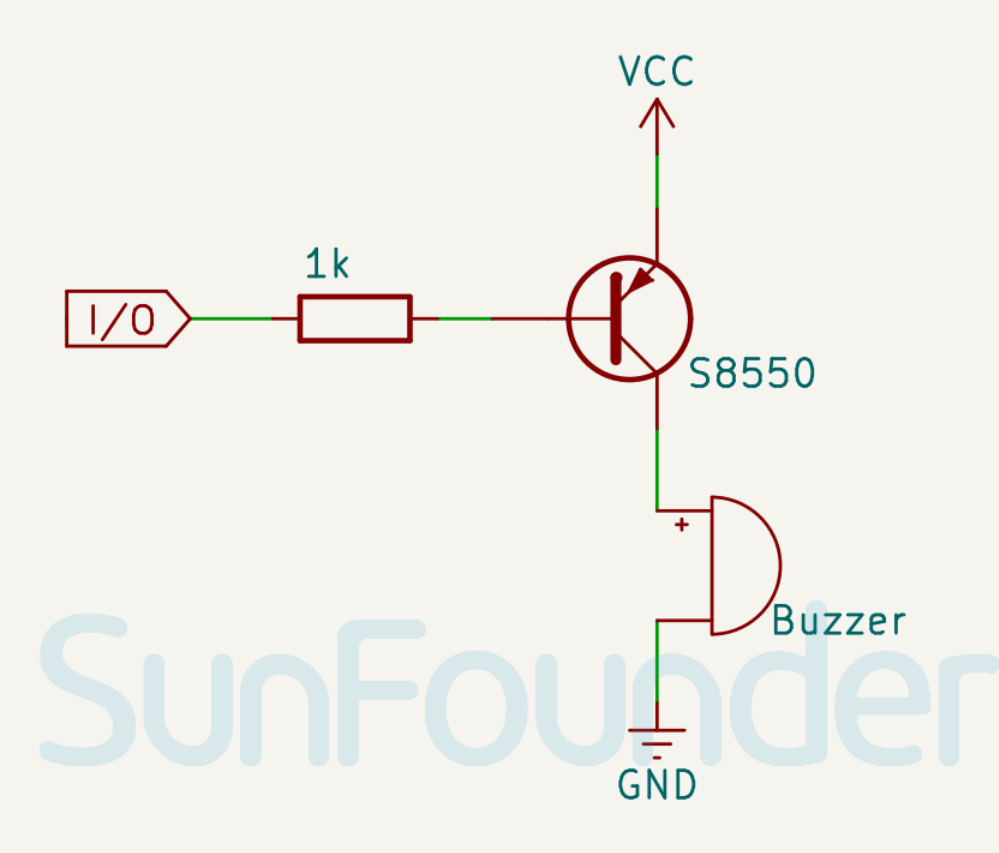
1.2.34 Passive Buzzer Module

The passive buzzer is a device that generates sound when an electrical signal is applied to it. It is called passive because it does not have an internal oscillator to generate sound on its own. Instead, it relies on an external signal from a microcontroller like Arduino to produce sound. The passive buzzer module is a small electronic component that contains a passive buzzer and some additional circuitry that makes it easier to use with Arduino.

Pinout

- **VCC**: This is the positive power supply input from the main control.
- **GND**: Ground connection.
- **I/O**: Through this pin, you can send control signals to control the tone and frequency of the buzzer.

Schematic diagram



Example

- *Lesson 32: Passive Buzzer Module* (Arduino UNO)
- *Lesson 32: Passive Buzzer Module* (ESP32)
- *Lesson 32: Passive Buzzer Module* (Raspberry Pi Pico)
- *Lesson 32: Passive Buzzer Module* (Raspberry Pi)
- *Lesson 38: Gas leak alarm* (Arduino UNO)
- *Lesson 36: Gas leak alarm* (ESP32)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.35 Servo Motor (SG90)

Servo motors are devices that can rotate to a specific angle or position. They can be used to move robotic arms, steering wheels, camera gimbals, etc. Servo motors have three wires: power, ground and signal. The power wire is usually red and should be connected to the 5V pin on the Arduino board. The ground wire is usually black or brown and should be connected to a ground pin on the board. The signal wire is usually yellow or orange and should be connected to a PWM pin on the board.

Pinout

- Brown wire: GND
- Orange wire: Signal pin, connect to the PWM pin of main board.
- Red wire: VCC

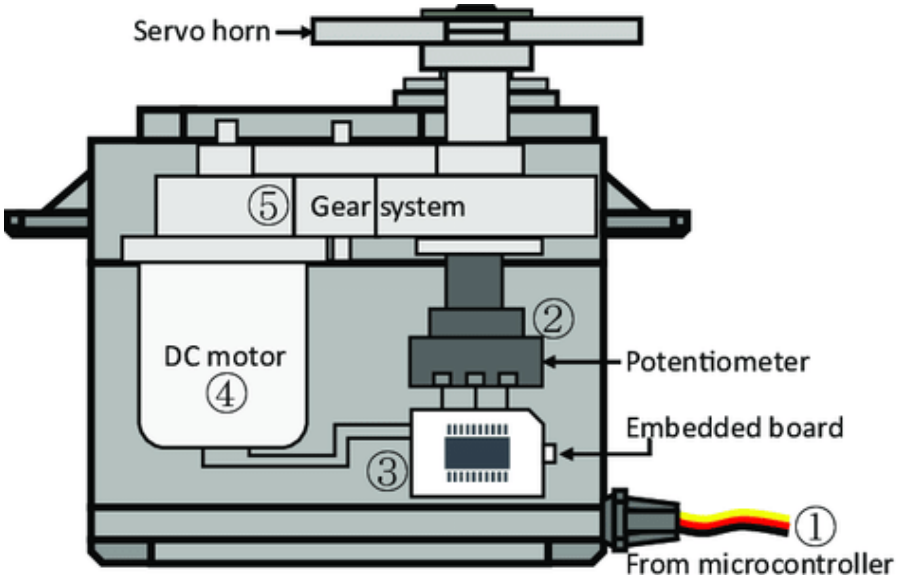
Principle

A servo is generally composed of the following parts: case, shaft, gear system, potentiometer, DC motor, and embedded board.

It works like this:

- The microcontroller sends out PWM signals to the servo, and then the embedded board in the servo receives the signals through the signal pin and controls the motor inside to turn.
- As a result, the motor drives the gear system and then motivates the shaft after deceleration.

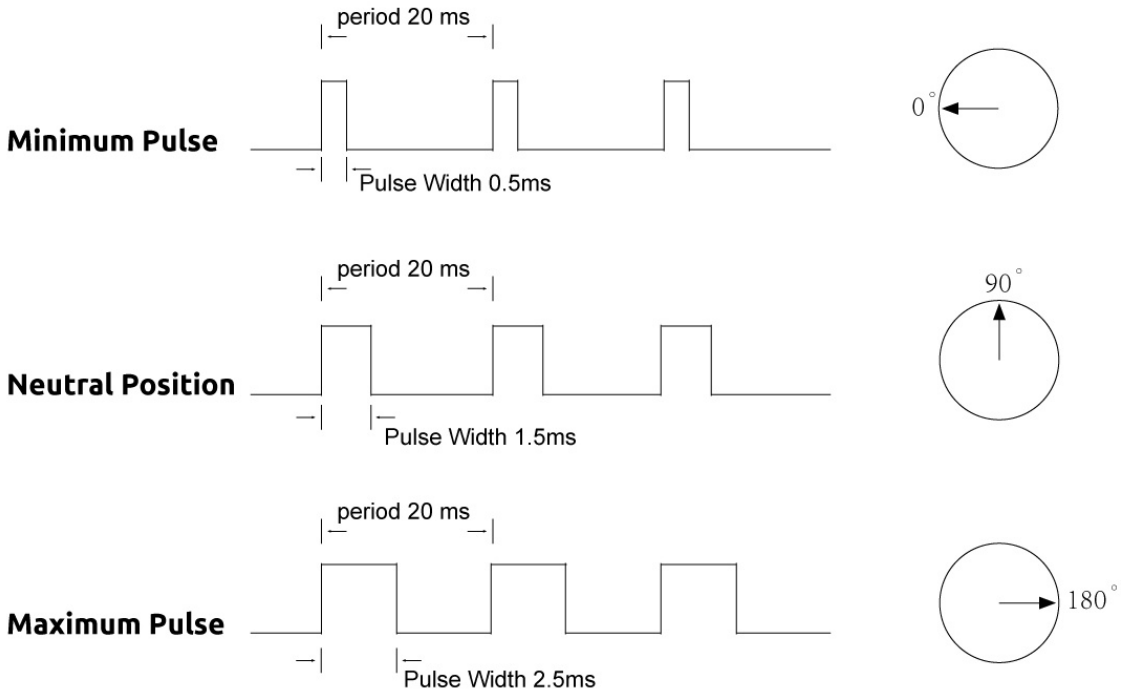
- The shaft and potentiometer of the servo are connected together.
- When the shaft rotates, it drives the potentiometer, so the potentiometer outputs a voltage signal to the embedded board.
- Then the board determines the direction and speed of rotation based on the current position, so it can stop exactly at the right position as defined and hold there.



Work Pulse

The angle is determined by the duration of a pulse that is applied to the control wire. This is called Pulse width Modulation.

- The servo expects to see a pulse every 20 ms. The length of the pulse will determine how far the servo turns.
- For example, a 1.5ms pulse will make the servo turn to the 90 degree position (neutral position).
- When a pulse is sent to a servo that is less than 1.5 ms, the servo rotates to a position and holds its output shaft some number of degrees counterclockwise from the neutral point.
- When the pulse is wider than 1.5 ms the opposite occurs.
- The minimal width and the maximum width of pulse that will command the servo to turn to a valid position are functions of each servo.
- Generally the pulse will be about 0.5 ms ~ 2.5 ms wide.



Example

- *Lesson 33: Servo Motor (SG90)* (Arduino UNO)
- *Lesson 33: Servo Motor (SG90)* (ESP32)
- *Lesson 33: Servo Motor (SG90)* (Raspberry Pi Pico)
- *Lesson 33: Servo Motor (SG90)* (Raspberry Pi)
- *Lesson 37: Smart trashcan* (Arduino UNO)
- *Lesson 35: Smart trashcan* (ESP32)

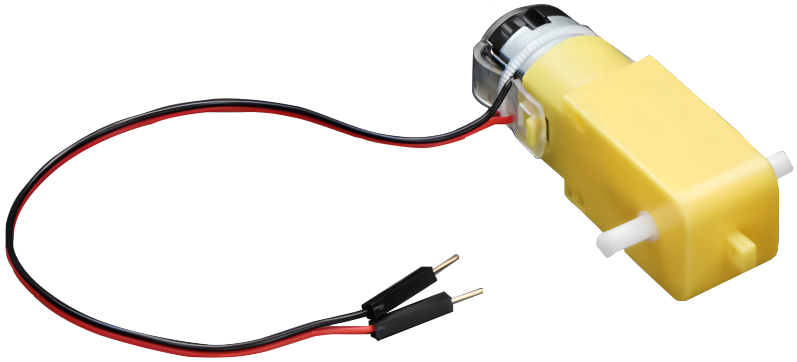
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.36 TT Motor



A TT motor is a type of DC motor that has a gearbox attached to it. The gearbox reduces the speed of the motor and increases its torque. A TT motor is commonly used in applications such as driving wheels, propellers, fans, among others. A TT motor has two wires: a positive wire and a negative wire. The positive wire is usually red and the negative wire is usually black.

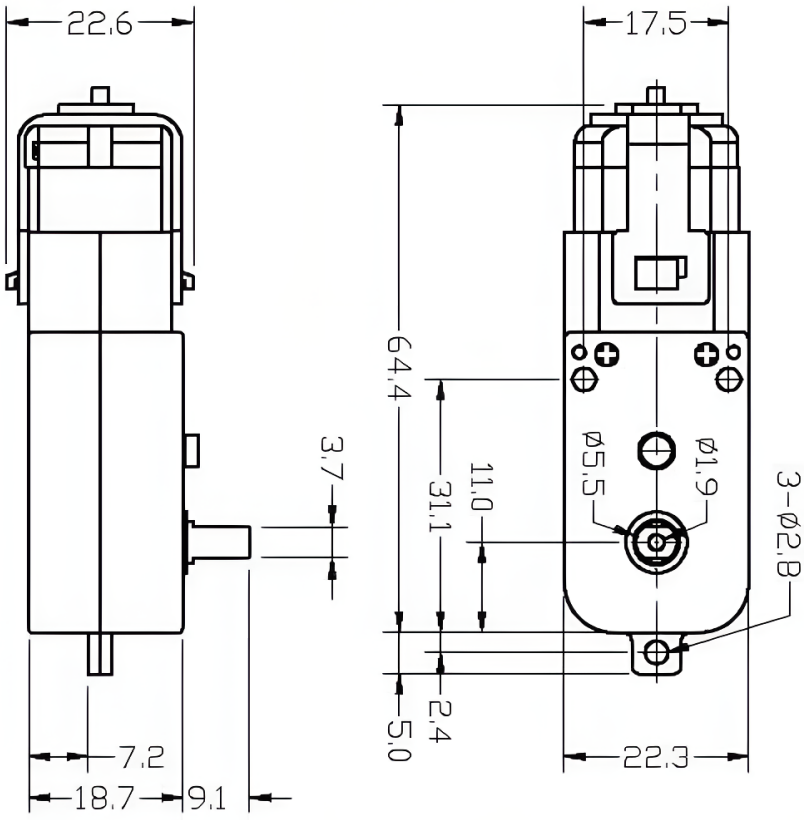
A TT DC gearbox motor with a 1:48 gear ratio is used in the product, it comes with 2 x 200mm wires with 0.1" male connectors that fit into a breadboard. Perfect for plugging into a breadboard or terminal block.

You can power these motors with 3 ~ 6VDC, but of course, they will go a little faster at higher voltages.

Technical Details

- Rated Voltage: 3~6V
- Continuous No-Load Current: 150mA +/- 10%
- Min. Operating Speed (3V): 90+/- 10% RPM
- Min. Operating Speed (6V): 200+/- 10% RPM
- Stall Torque (3V): 0.4kg.cm
- Stall Torque (6V): 0.8kg.cm
- Gear Ratio: 1:48
- Body Dimensions: 70 x 22 x 18mm
- Wires Length: 200mm & 28 AWG
- Weight: 30.6g

Dimensional Drawing



Example

- *Lesson 34: TT Motor* (Arduino UNO)
- *Lesson 34: TT Motor* (ESP32)
- *Lesson 34: TT Motor* (Raspberry Pi Pico)
- *Lesson 34: TT Motor* (Raspberry Pi)
- *Lesson 07: Infrared Speed Sensor Module* (Arduino UNO)
- *Lesson 07: Infrared Speed Sensor Module* (Raspberry Pi)

Wireless & IoT

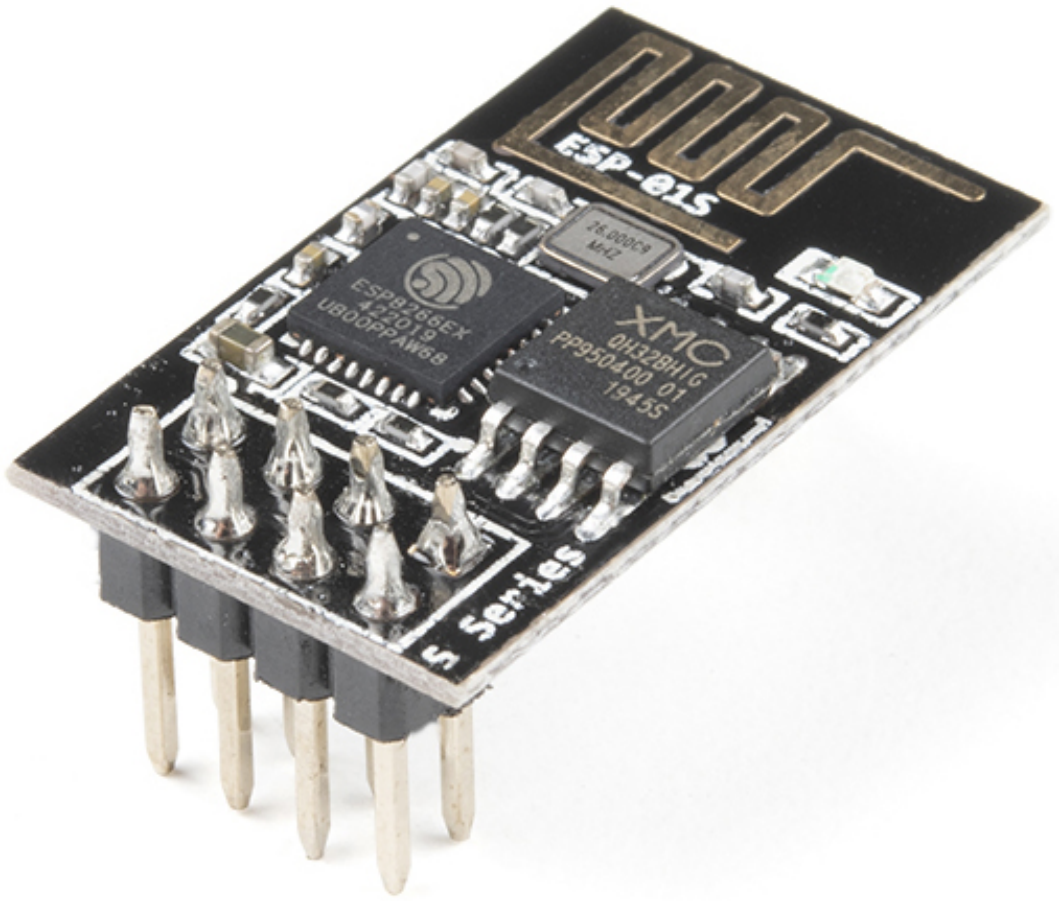
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.37 ESP8266 Module



The ESP8266 is a low-cost Wi-Fi microchip, with built-in TCP/IP networking software, and microcontroller capability, produced by Espressif Systems in Shanghai, China.

The chip first came to the attention of Western makers in August 2014 with the ESP-01 module, made by a third-party manufacturer Ai-Thinker. This small module allows microcontrollers to connect to a Wi-Fi network and make simple TCP/IP connections using Hayes-style commands. However, at first, there was almost no English-language documentation on the chip and the commands it accepted. The very low price and the fact that there were very few external components on the module, which suggested that it could eventually be very inexpensive in volume, attracted many hackers to explore the module, the chip, and the software on it, as well as to translate the Chinese documentation.

Pins of ESP8266 and their functions:

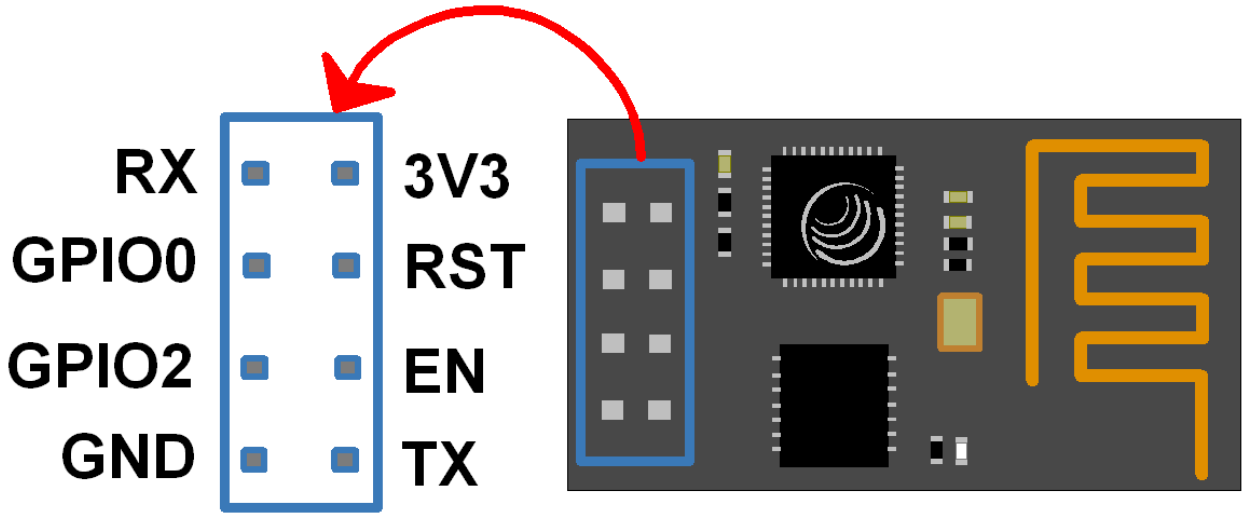
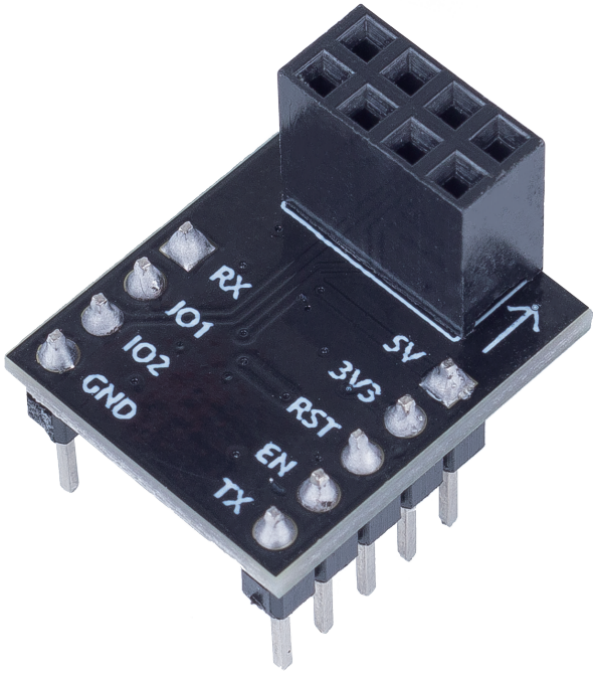


Table 1: ESP8266-01 Pins

Pin	Name	Description
1	TXD	UART_TXD, sending; General Purpose Input/Output: GPIO1; Pull-down is not allowed when startup.
2	GND	GND
3	CU_PD	Working at high level; Power off when low level is supplied.
4	GPIO2	It should be high level when power on, hardware pull-down is not allowed; Pull-up by default;
5	RST	External Reset signal, reset when low level is supplied; work when high level is supplied (high level by default);
6	GPIO0	WiFi Status indicator; Operation mode selection: Pull-up: Flash Boot, operation mode; Pull-down: UART Download, download mode
7	VCC	Power Supply(3.3V)
8	RXD	UART_RXDReceiving; General Purpose Input/Output: GPIO3;

- ESP8266 - Espressif
-

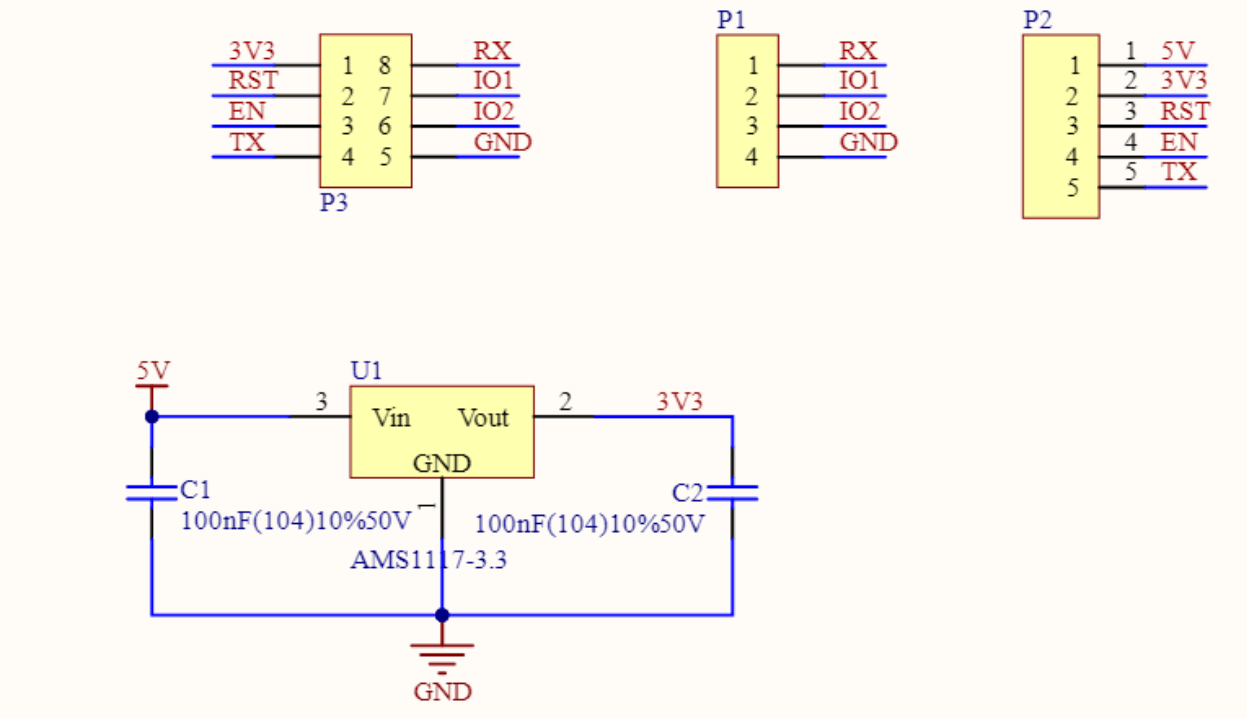
ESP8266 Adapter



The ESP8266 adapter is an expansion board that allows the ESP8266 module to be used on a breadboard.

It perfectly matches the pins of the ESP8266 itself, and also adds a 5V pin to receive the voltage from the Arduino board. The integrated AMS1117 chip is used to drive the ESP8266 module after dropping the voltage to 3.3V.

The schematic diagram is as follows:



Example

- *Lesson 35: Get Started with ESP8266 Module* (Arduino UNO)
- *Lesson 48: Weather Monitor with ThingSpeak* (Arduino UNO)
- *Lesson 49: Vibration Alert System with IFTTT* (Arduino UNO)
- *Lesson 50: Flame Alert System with Blynk* (Arduino UNO)
- *Lesson 51: Intrusion Alert System with Blynk* (Arduino UNO)

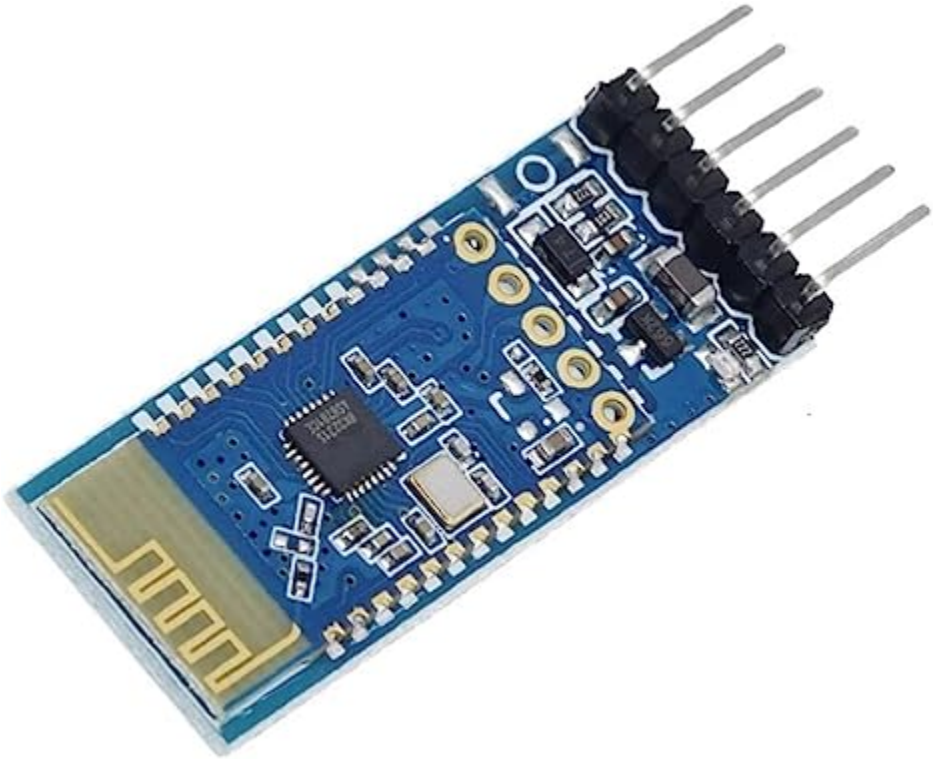
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.38 JDY-31 Bluetooth Module



Warning: This module **does not support Apple device** connections, so tutorials involving this module require an Android phone or tablet.

The JDY-31 Bluetooth module is a pin-compatible replacement for the HC-06 Bluetooth module. It is simpler and easier to use than the HC-06 and is often available at a slightly lower cost.

The JDY-31 Bluetooth module is based on Bluetooth 3.0 SPP design and can support Windows, Linux, and Android data transmission. The working frequency of the JDY-31 Bluetooth module is 2.4 GHz with modulation mode GFSK. The maximum transmission power is 8 dB, and the maximum transmission distance is 30 meters. Users can modify the device name through AT command, baud rate, and other instructions.

Pins of JDY-31 and their functions:

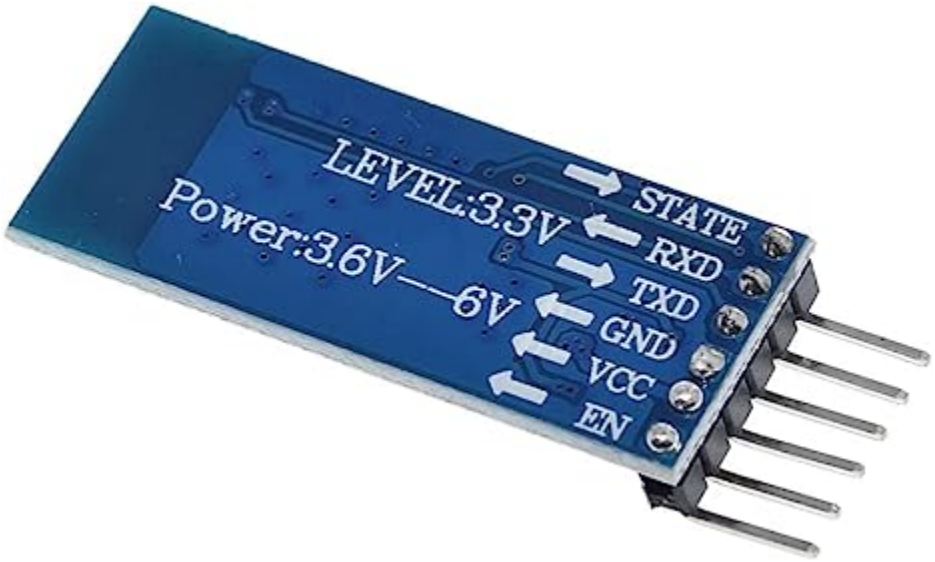


Table 2: JDY-31 Pins

Pin	Name	Description
1	STATE	Connection status pin (not connected low level, output high level after connectio)
2	RXD	Receiver pin, this pin must connect to TX pin of next device.
3	TXD	Transmitter pin, this pin must connect to RX pin of next device.
4	GND	GND
5	VCC	Power Supply(1.8-3.6V, 3.3v recommended)
6	EN	enable or disable the module. When this pin is held high, the module is enabled and begins transmitting and receiving data.

patch application: general application only need to connect VCC, GND, RXD, TXD 4 pins, if you need to actively disconnect in the connection state, send AT+DISC in the connection state.

AT Command Set

Command	Function	Default
AT+VERSION	Version Number	JDY-31-V1.2
AT+RESET	Soft reset	
AT+DISC	Disconnect (valid when connected)	
AT+LADDR	Query the MAC address of the module	
AT+PIN	Set or query connection password	1234
AT+BAUD	Set or query baud rate	9600
AT+NAME	Set or query broadcast name	JDY-31-SPP
AT+DEFAULT	Factory reset	
AT+ENLOG	Serial port status output	1

Example

- *Lesson 36: Get Started with Bluetooth Module* (Arduino UNO)
- *Lesson 46: Bluetooth LCD* (Arduino UNO)
- *Lesson 47: Bluetooth Traffic Light* (Arduino UNO)

Power

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

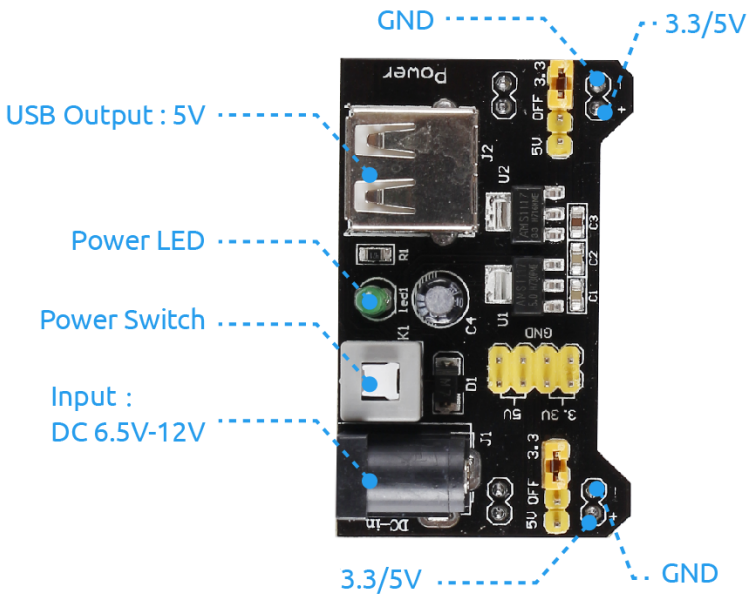
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.2.39 Power Supply Module

The breadboard power module provides 3.3V and 5V with a series diode and reverse polarity protection. It accepts input from 6.5V to 12V and outputs 3.3V and +5V. This power supply module is essential for experimenters who need to test electronic circuits on breadboards or perforated/veroboards.



Features

1. Plug directly to MB102 Standard breadboard.
2. Input voltage: 6.5-12 V (DC) or 5V USB power supply.
3. Output voltage: 3.3V and 5V can switch over.
4. Maximum output current: <700 mA.
5. External Input voltage ON/OFF switch.
6. Independent control of upper and Lower Bread Board Power Rails. Can switch over to 0V, 3.3V, 5V using jumpers on any rail.
7. On-board two groups of 3.3V, 5V DC output plug pin, convenient external lead use.
8. USB device connector onboard for power output to external device.
9. Size: 5.3cm x 3.5cm.

Example

- *Lesson 39: Automatic soap dispenser* (Arduino UNO)
- *Lesson 37: Automatic soap dispenser* (ESP32)
- *Lesson 45: Plant Monitor* (Arduino UNO)
- *Lesson 43: Plant Monitor* (ESP32)
- *Lesson 39: Automatic soap dispenser* (Arduino UNO)
- *Lesson 37: Automatic soap dispenser* (ESP32)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.

- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3 For Arduino Uno

Arduino users, please refer to the following tutorial.

The following tutorial uses Arduino UNO R3 as example, but it also applies to Arduino UNO R4.

Arduino Config

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.1 Get Started with Arduino

If you have no idea about Arduino. There are several words I would like to show you: electronics, design, programming, and even Maker. Some of you may think these words are quite far away from us, but in fact, they are not far at all. Because Arduino can take us into the world of programming and help us realize the dream of being a Maker. In this session we will learn:

- What is Arduino?
- What can Arduino do?
- How to build an Arduino Project?

What is Arduino?

First of all, I will give you a brief introduction to Arduino.

Arduino is a convenient, flexible, and easy-to-use open-source electronic prototyping platform, including hardware Arduino boards of various models and software Arduino IDE. It is not only suitable for engineers for rapid prototyping, but also artists, designers, hobbyists, while it is almost a must-have tool for modern Makers.

Arduino is quite a large system. It has software, hardware, and a very huge online community of people who have never met each other but are able to work together because of a common hobby. Everyone in the Arduino family is using their wisdom, making with their hands, and sharing one great invention after another. And you can also be a part of it.

What can Arduino do?

Speaking of which, you may have doubts about what Arduino can actually do. Suffice it to say, Arduino will solve all your problems.

Technically speaking, Arduino is a programmable logic controller. It is a development board that can be used to create many exciting and creative electronic creations: such as remote-controlled cars, robotic arms, bionic robots, smart homes, etc.

Arduino boards are straightforward, simple, and powerful, suitable for students, makers and even professional programmers.

To this day, electronics enthusiasts worldwide continue to develop creative electronic creations based on Arduino development boards.

How to build an Arduino Project

Follow these steps to learn how to use Arduino from zero!

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Download and Install Arduino IDE 2.0

The Arduino IDE, known as Arduino Integrated Development Environment, provides all the software support needed to complete an Arduino project. It is a programming software specifically designed for Arduino, provided by the Arduino team, that allows us to write programs and upload them to the Arduino board.

The Arduino IDE 2.0 is an open-source project. It is a big step from its sturdy predecessor, Arduino IDE 1.x, and comes with revamped UI, improved board & library manager, debugger, autocomplete feature and much more.

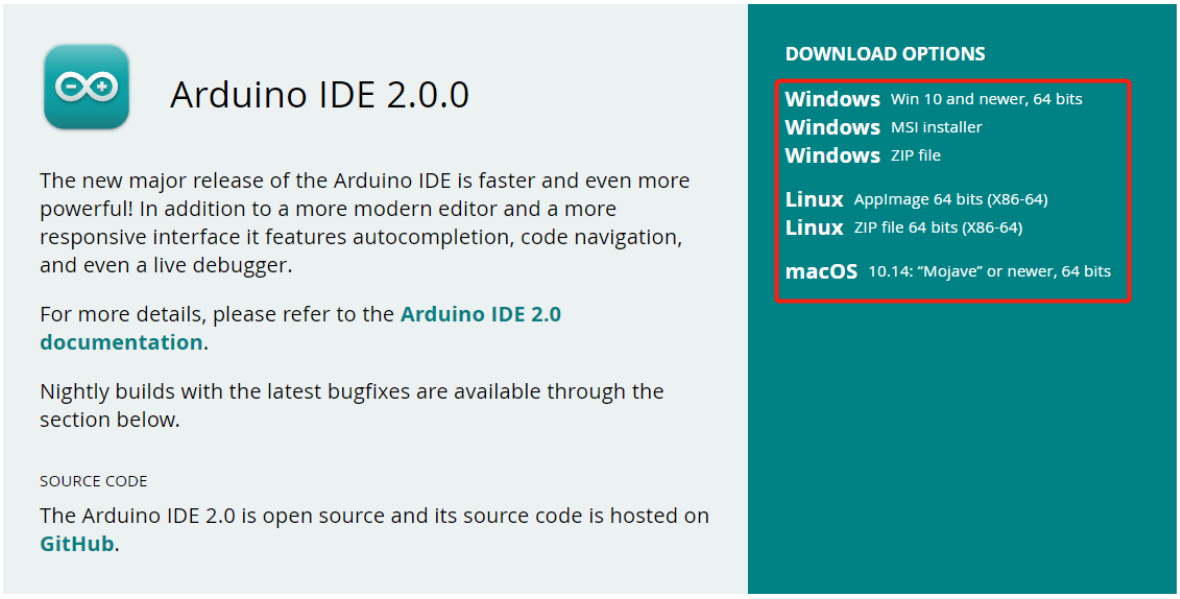
In this tutorial, we will show how to download and install the Arduino IDE 2.0 on your Windows, Mac, or Linux computer.

Requirements

- Windows - Win 10 and newer, 64 bits
- Linux - 64 bits
- Mac OS X - Version 10.14: “Mojave” or newer, 64 bits

Download the Arduino IDE 2.0

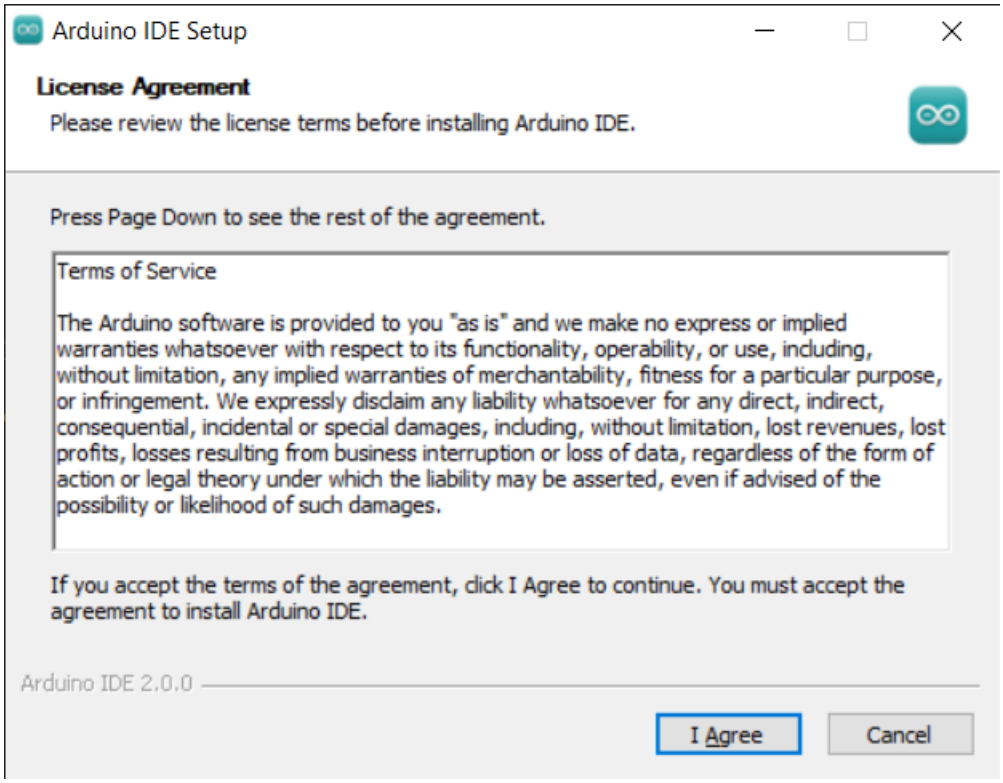
1. Visit .
2. Download the IDE for your OS version.



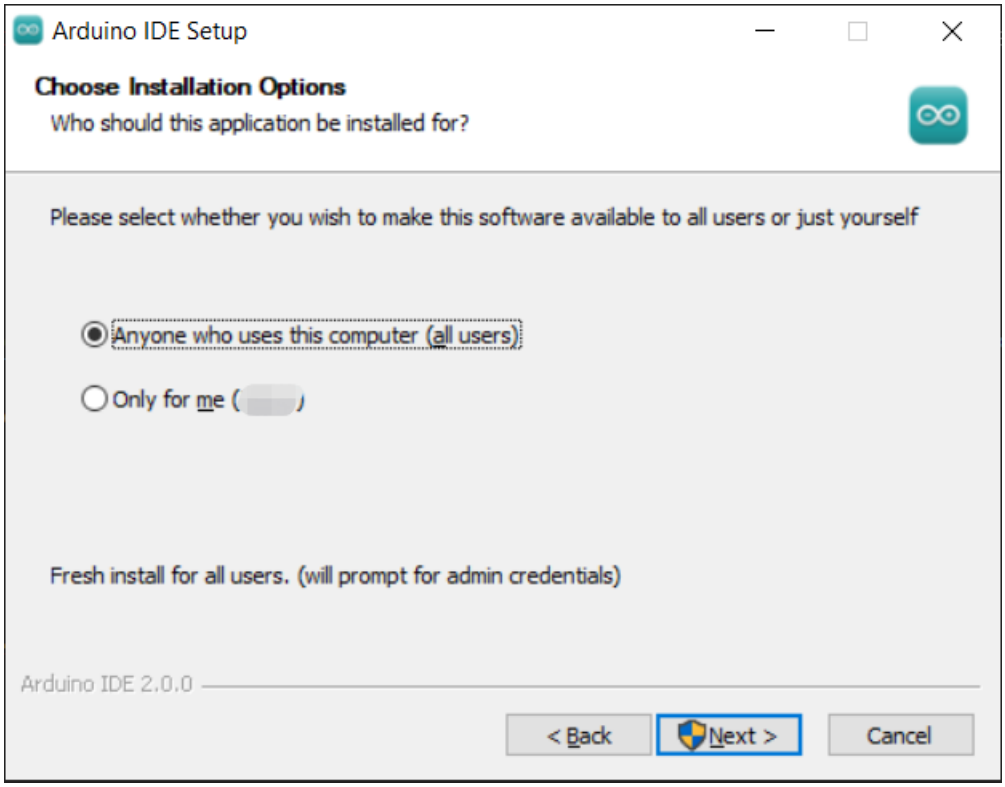
Installation

Windows

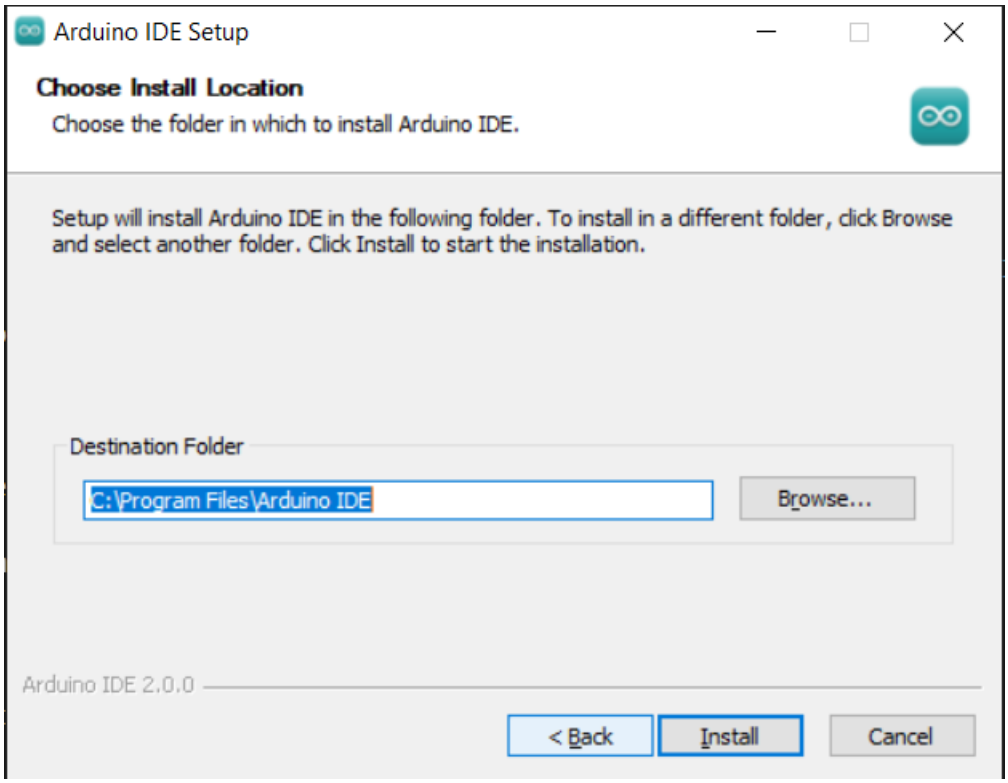
1. Double click the `arduino-ide_xxxx.exe` file to run the downloaded file.
2. Read the License Agreement and agree it.



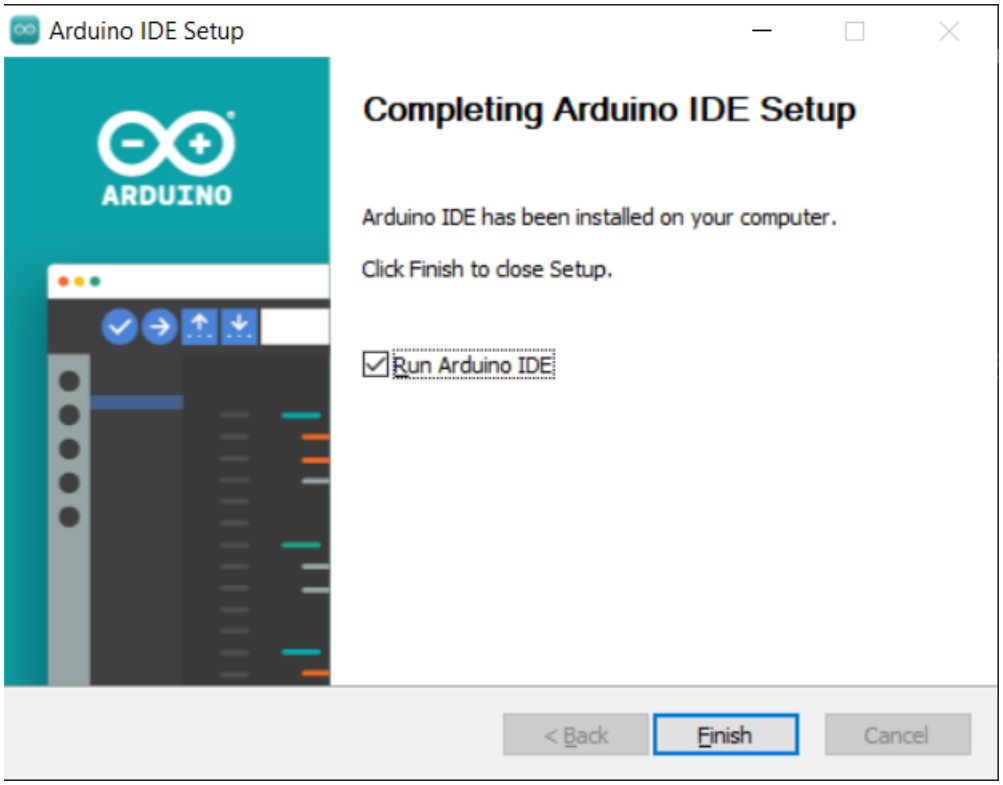
3. Choose installation options.



4. Choose install location. It is recommended that the software be installed on a drive other than the system drive.

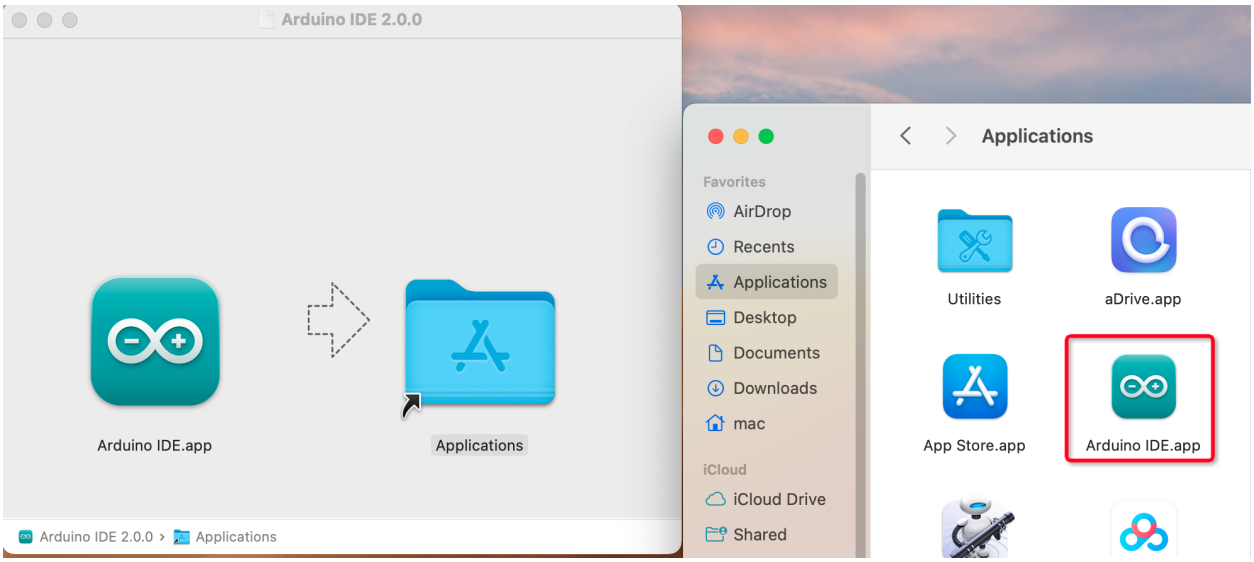


5. Then Finish.



macOS

Double click on the downloaded `arduino_ide_xxxx.dmg` file and follow the instructions to copy the **Arduino IDE.app** to the **Applications** folder, you will see the Arduino IDE installed successfully after a few seconds.

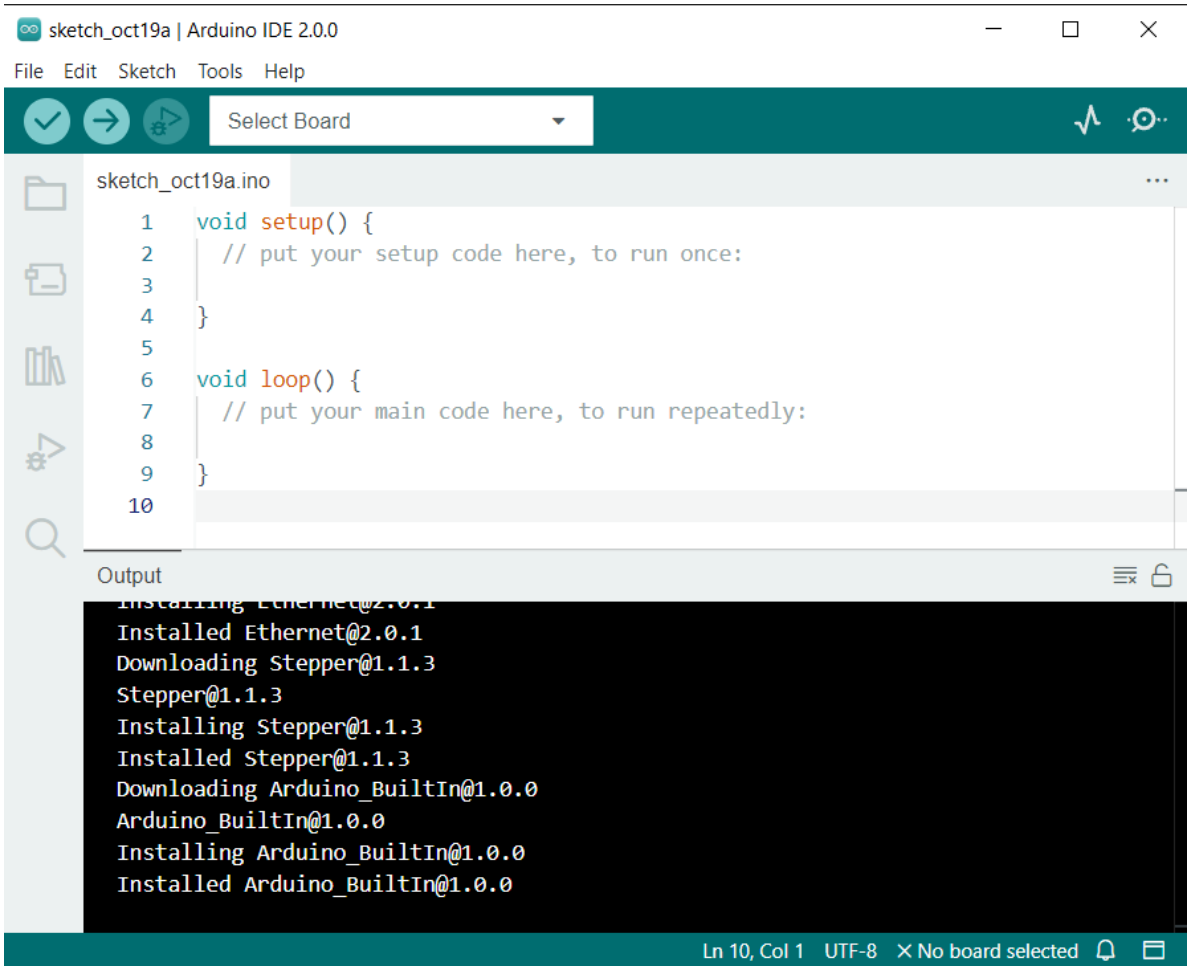


Linux

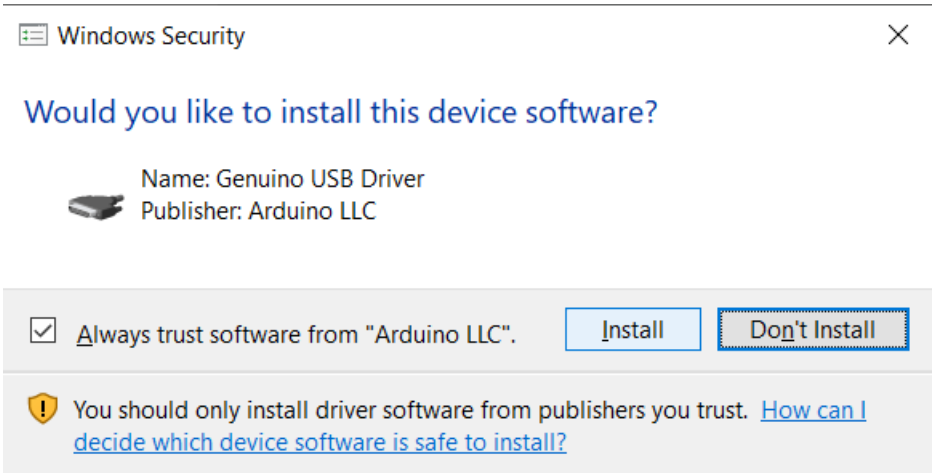
For the tutorial on installing the Arduino IDE 2.0 on a Linux system, please refer

Open the IDE

1. When you first open Arduino IDE 2.0, it automatically installs the Arduino AVR Boards, built-in libraries, and other required files.



2. In addition, your firewall or security center may pop up a few times asking you if you want to install some device driver. Please install all of them.



3. Now your Arduino IDE is ready!

Note: In the event that some installations didn't work due to network issues or other reasons, you can reopen the Arduino IDE and it will finish the rest of the installation. The Output window will not automatically open after all installations are complete unless you click Verify or Upload.

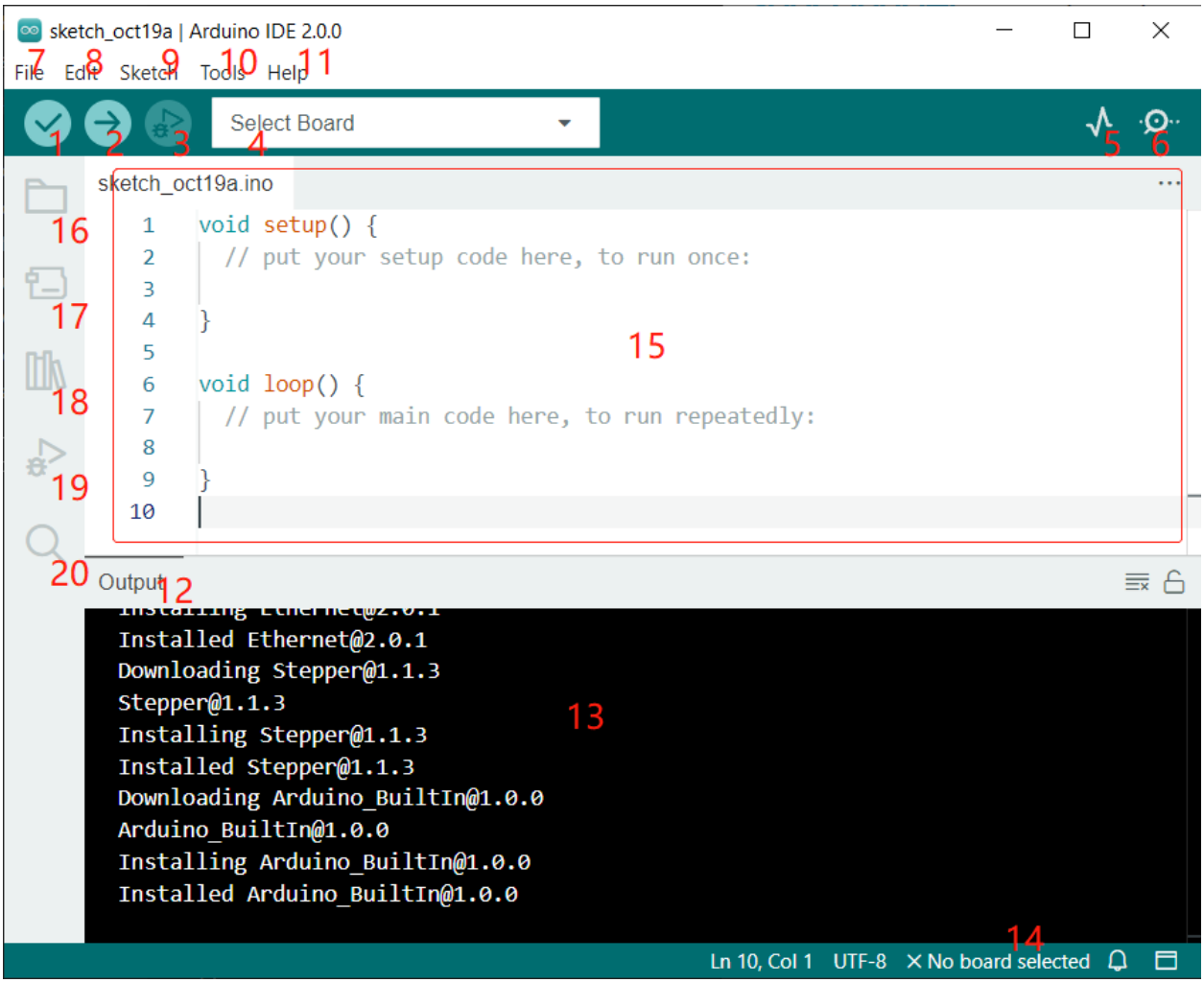
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Introduce of Arduino IDE



1. **Verify**: Compile your code. Any syntax problem will be prompted with errors.
2. **Upload**: Upload the code to your board. When you click the button, the RX and TX LEDs on the board will flicker fast and won't stop until the upload is done.
3. **Debug**: For line-by-line error checking.
4. **Select Board**: Quick setup board and port.
5. **Serial Plotter**: Check the change of reading value.
6. **Serial Monitor**: Click the button and a window will appear. It receives the data sent from your control board. It is very useful for debugging.
7. **File**: Click the menu and a drop-down list will appear, including file creating, opening, saving, closing, some parameter configuring, etc.
8. **Edit**: Click the menu. On the drop-down list, there are some editing operations like **Cut**, **Copy**, **Paste**, **Find**, and so on, with their corresponding shortcuts.
9. **Sketch**: Includes operations like **Verify**, **Upload**, **Add** files, etc. A more important function is **Include Library** – where you can add libraries.

10. **Tool**: Includes some tools – the most frequently used Board (the board you use) and Port (the port your board is at). Every time you want to upload the code, you need to select or check them.
11. **Help**: If you're a beginner, you may check the options under the menu and get the help you need, including operations in IDE, introduction information, troubleshooting, code explanation, etc.
12. **Output Bar**: Switch the output tab here.
13. **Output Window**: Print information.
14. **Board and Port**: Here you can preview the board and port selected for code upload. You can select them again by **Tools** -> **Board / Port** if any is incorrect.
15. The editing area of the IDE. You can write code here.
16. **Sketchbook**: For managing sketch files.
17. **Board Manager**: For managing board driver.
18. **Library Manager**: For managing your library files.
19. **Debug**: Help debugging code.
20. **Search**: Search the codes from your sketches.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

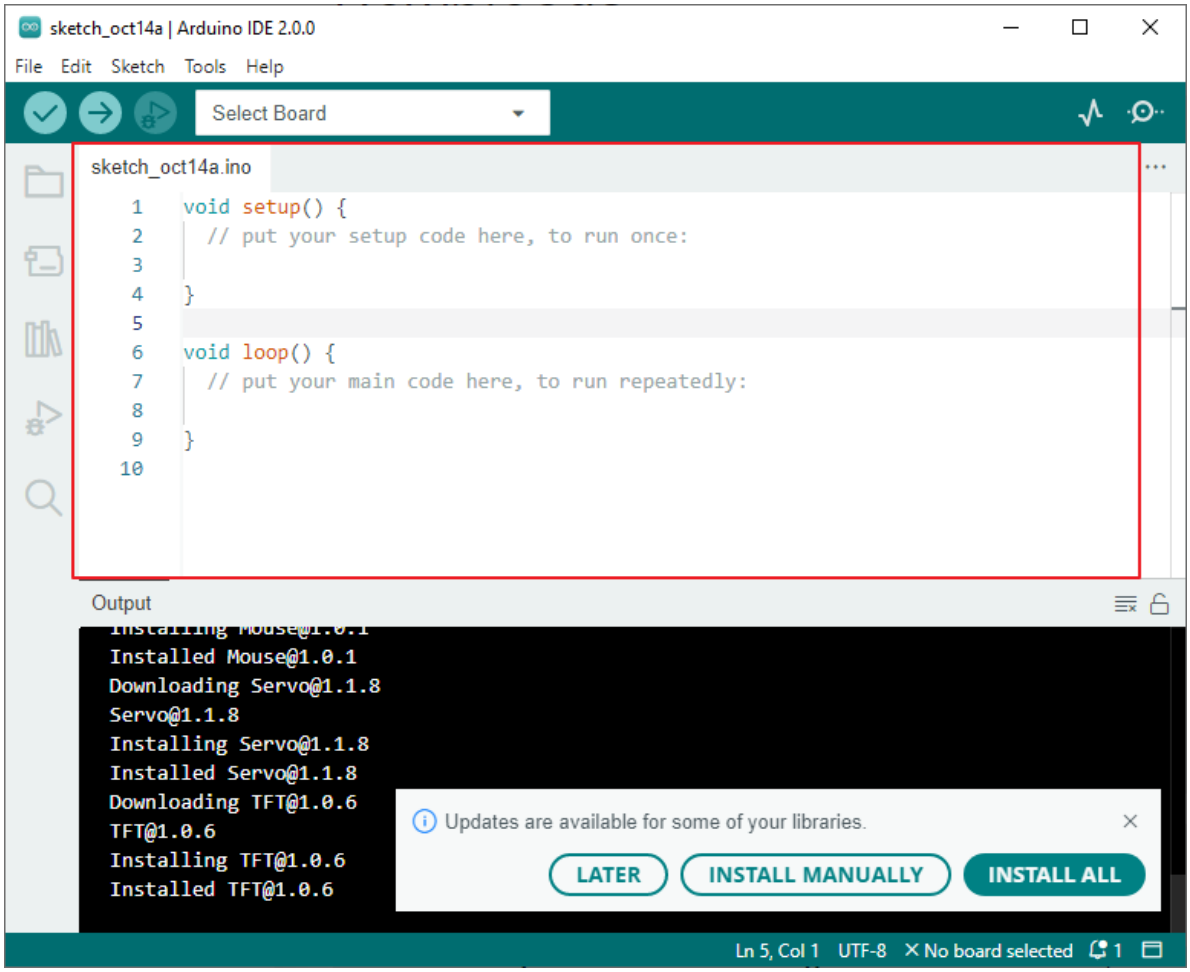
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

How to create, open or Save the Sketch?

1. When you open the Arduino IDE for the first time or create a new sketch, you will see a page like this, where the Arduino IDE creates a new file for you, which is called a "sketch".

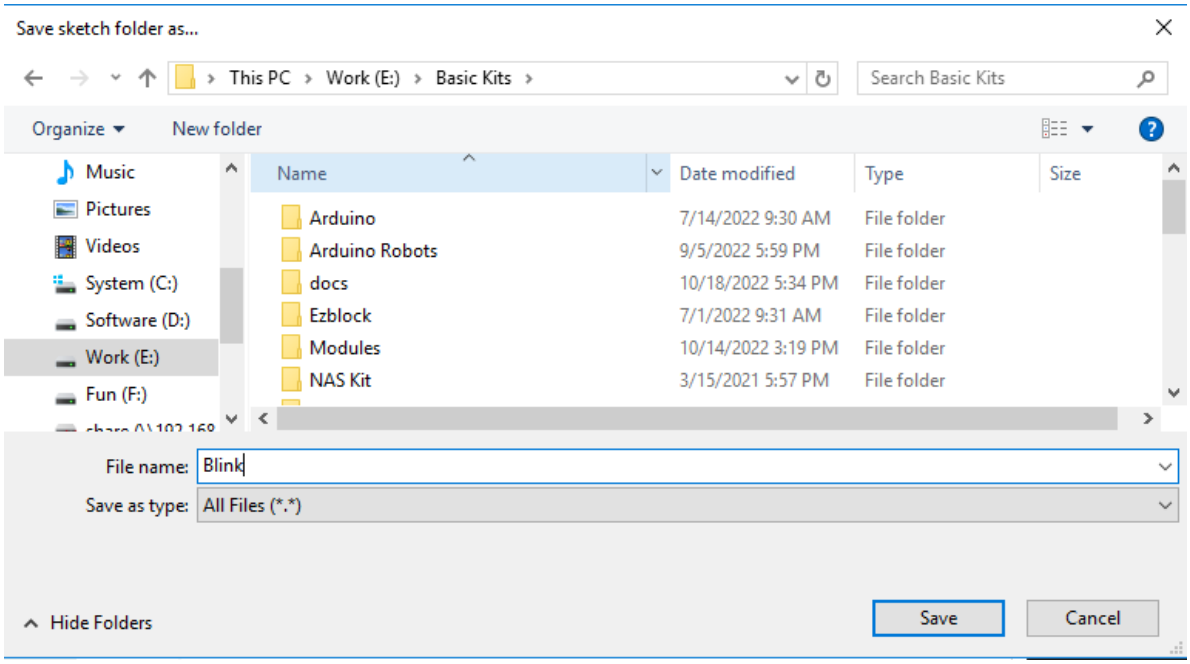


These sketch files have a regular temporary name, from which you can tell the date the file was created. `sketch_oct14a.ino` means October 14th first sketch, `.ino` is the file format of this sketch.

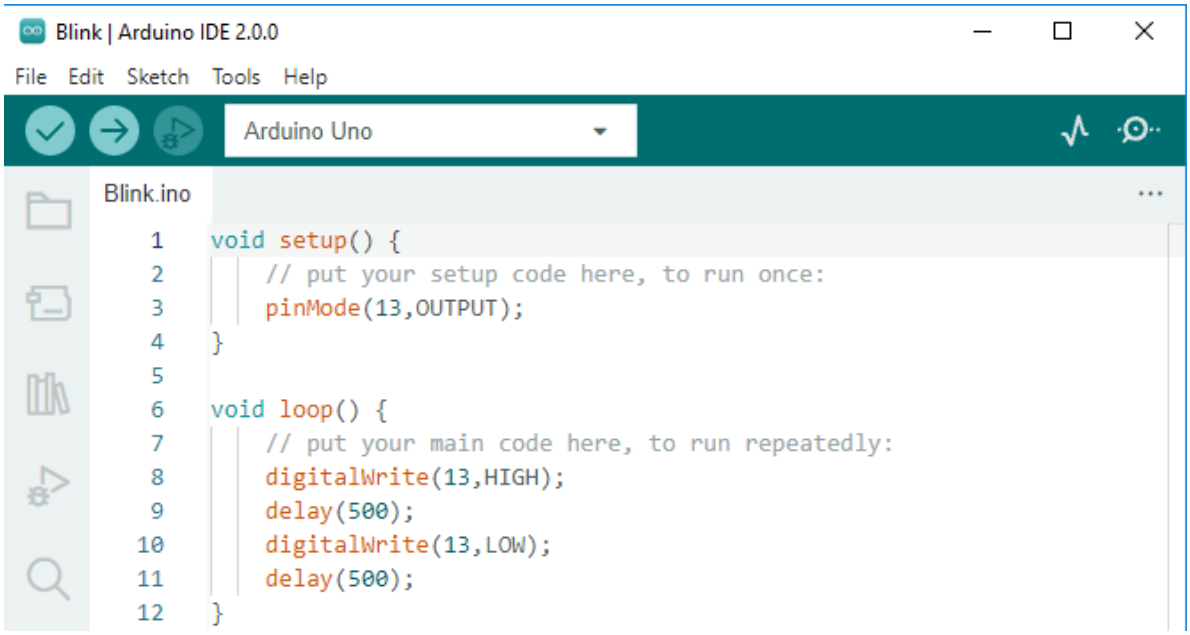
2. Now let's try to create a new sketch. Copy the following code into the Arduino IDE to replace the original code.

```
void setup() {
    // put your setup code here, to run once:
    pinMode(13,OUTPUT);
}

void loop() {
    // put your main code here, to run repeatedly:
    digitalWrite(13,HIGH);
    delay(500);
    digitalWrite(13,LOW);
    delay(500);
}
```

3. Press `Ctrl+S` or click **File** -> **Save**. The Sketch is saved in: `C:\Users\{your_user}\Documents\Arduino` by default, you can rename it or find a new path to save it.



4. After successful saving, you will see that the name in the Arduino IDE has been updated.



Please continue with the next section to learn how to upload this created sketch to your Arduino board.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

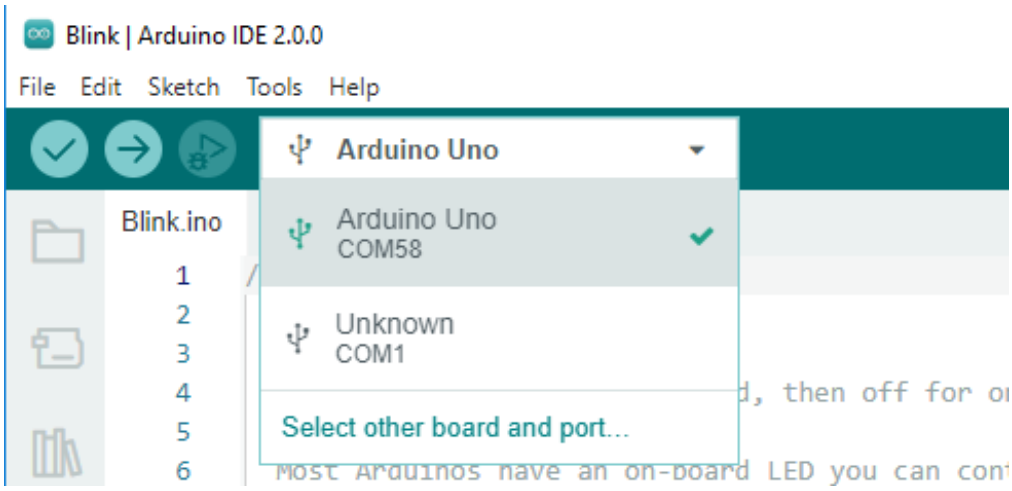
How to upload Sketch to the Board?

In this section, you will learn how to upload the sketch created previously to the Arduino board, as well as learn about some considerations.

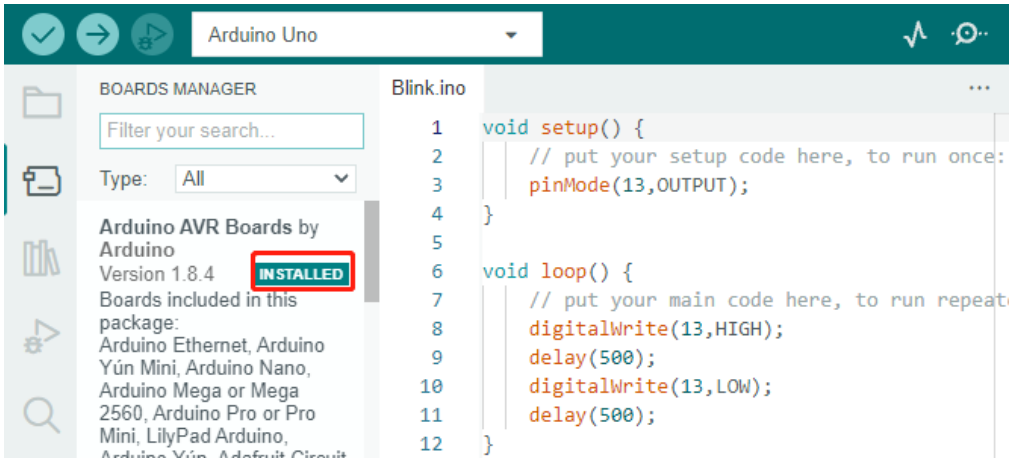
1. Choose Board and port

Arduino development boards usually come with a USB cable. You can use it to connect the board to your computer.

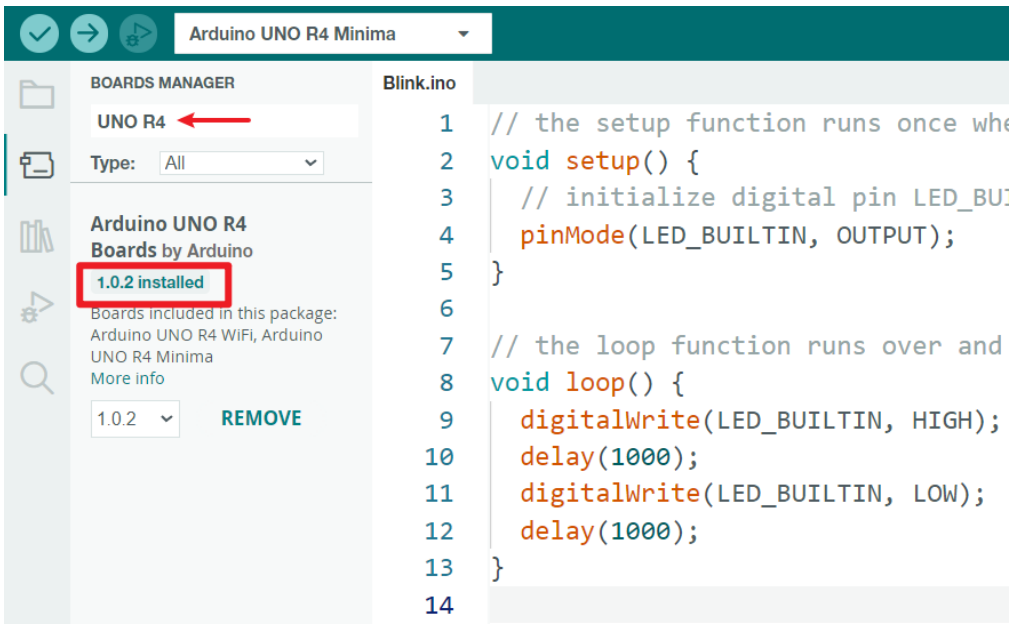
Select the correct **Board** and **Port** in the Arduino IDE. Normally, Arduino boards are recognized automatically by the computer and assigned a port, so you can select it here.



If your board is already plugged in, but not recognized, check if the **INSTALLED** logo appears in the **Arduino AVR Boards** section of the **Boards Manager**, if not, please scroll down a bit and click on **INSTALL**.



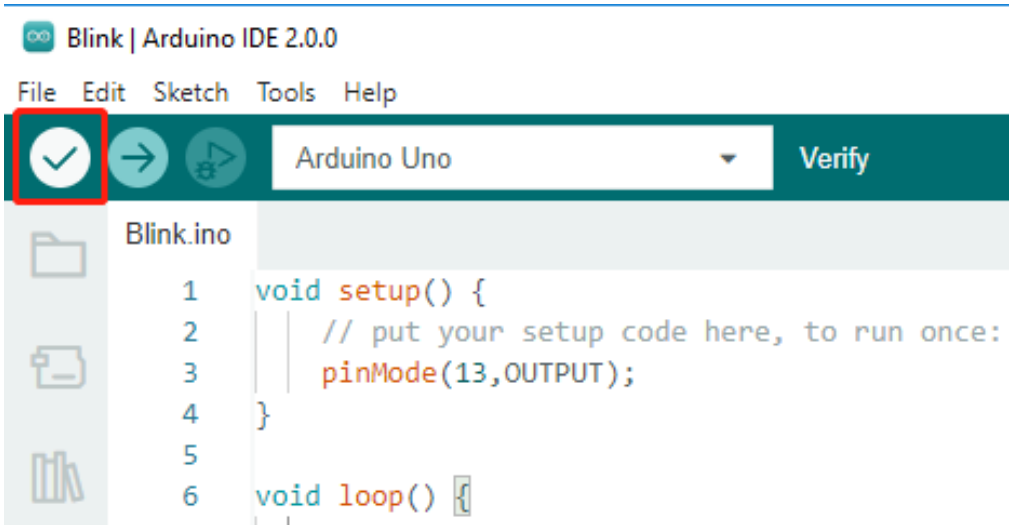
Specifically, for UNO R4, search **"UNO R4"** in **Boards Manager** and check if the corresponding library is installed.



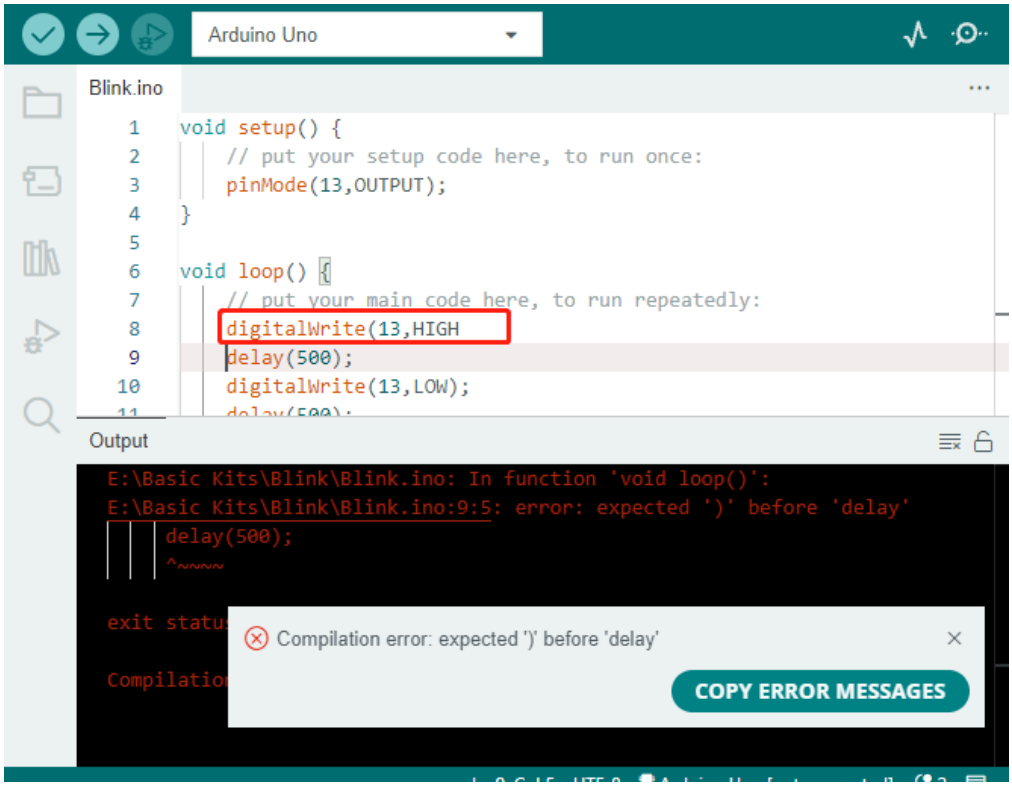
Reopening the Arduino IDE and re-plugging the Arduino board will fix most of the problems. You can also click **Tools** -> **Board** or **Port** to select them.

2. Verify the Sketch

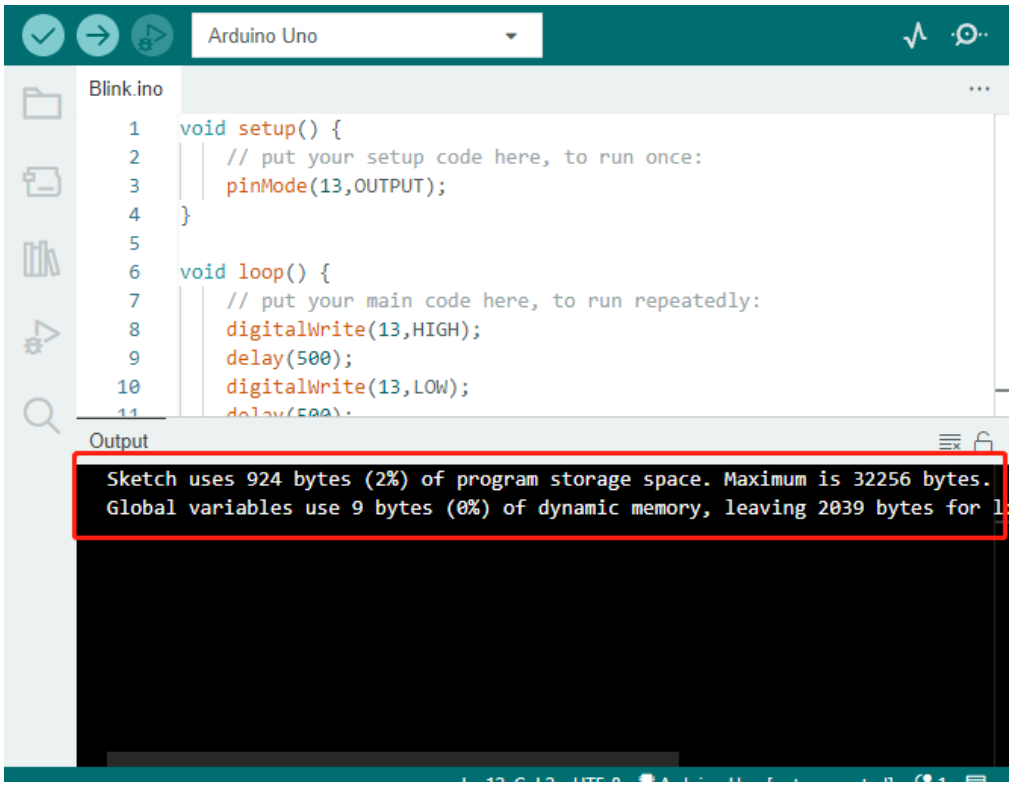
After clicking the Verify button, the sketch will be compiled to see if there are any errors.



You can use it to find mistakes if you delete some characters or type a few letters by mistake. From the message bar, you can see where and what type of errors occurred.

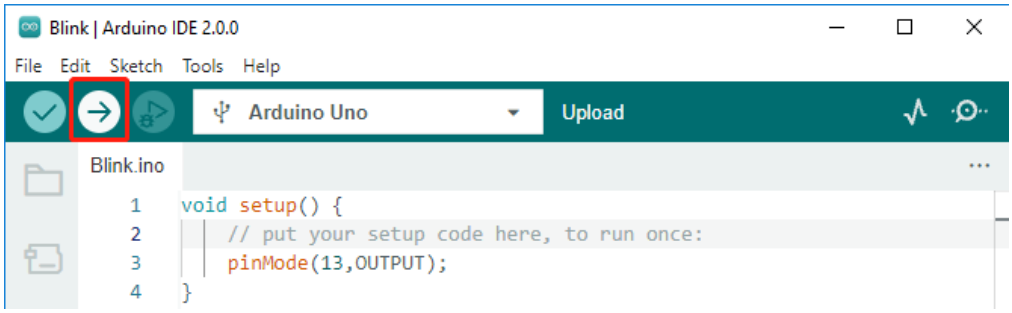


If there are no errors, you will see a message like the one below.

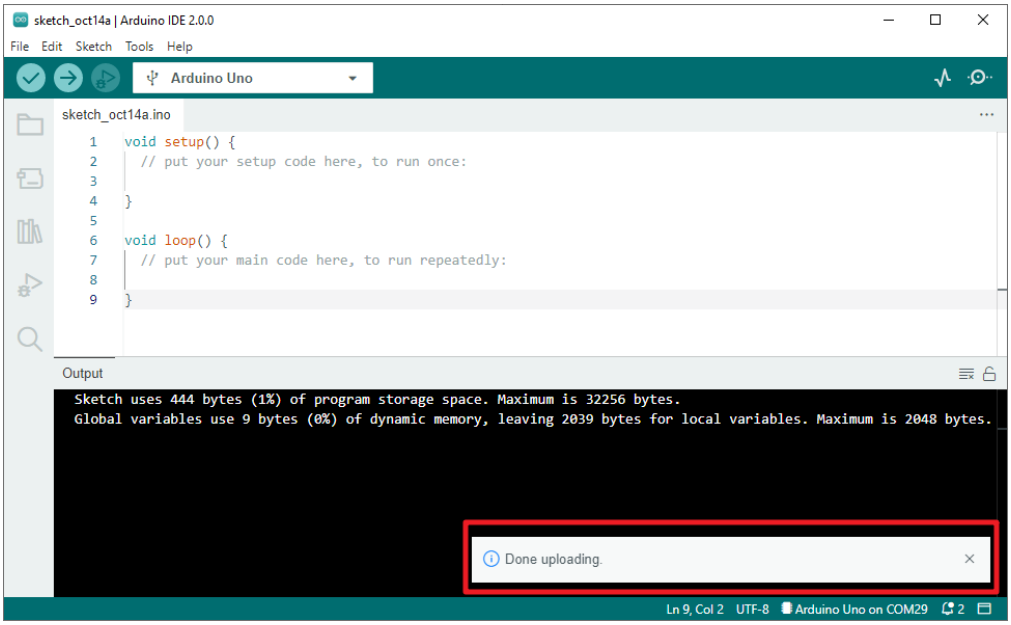


3. Upload sketch

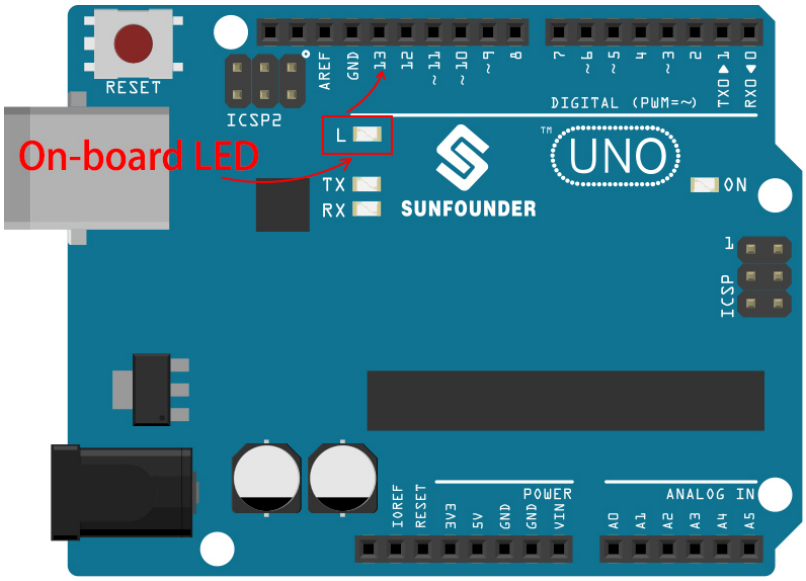
After completing the above steps, click the **Upload** button to upload this sketch to the board.



If successful, you will be able to see the following prompt.



At the same time, the on-board LED blink.



The Arduino board will automatically run the sketch after power is applied after the sketch is uploaded. The running program can be overwritten by uploading a new sketch.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.

- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Arduino Program Structure

Let's take a look at the new sketch file. Although it has a few lines of code itself, it is actually an "empty" sketch. Uploading this sketch to the development board will cause nothing to happen.

```
void setup() {
// put your setup code here, to run once:

}

void loop() {
// put your main code here, to run repeatedly:

}
```

If we remove `setup()` and `loop()` and make the sketch a real `blank` file, you will find that it does not pass the verification. They are the equivalent of the human skeleton, and they are indispensable.

During sketching, `setup()` is run first, and the code inside it (inside `{}`) is run after the board is powered up or reset and only once. `loop()` is used to write the main feature, and the code inside it will run in a loop after `setup()` is executed.

To better understand setup() and loop(), let's use four sketches. Their purpose is to make the on-board LED of the Arduino blink. Please run each experiment in turn and record their specific effects.

- Sketch 1: Make the on-board LED blink continuously.

```
void setup() {
    // put your setup code here, to run once:
    pinMode(13,OUTPUT);
}

void loop() {
    // put your main code here, to run repeatedly:
    digitalWrite(13,HIGH);
    delay(500);
    digitalWrite(13,LOW);
    delay(500);
}
```

- Sketch 2: Make the on-board LED blink only once.

```
void setup() {
    // put your setup code here, to run once:
    pinMode(13,OUTPUT);
    digitalWrite(13,HIGH);
    delay(500);
    digitalWrite(13,LOW);
    delay(500);
}
```

(continues on next page)

(continued from previous page)

```
void loop() {
    // put your main code here, to run repeatedly:
}
```

- Sketch 3: Make the on-board LED blink slowly once and then blink quickly.

```
void setup() {
    // put your setup code here, to run once:
    pinMode(13,OUTPUT);
    digitalWrite(13,HIGH);
    delay(1000);
    digitalWrite(13,LOW);
    delay(1000);
}

void loop() {
    // put your main code here, to run repeatedly:
    digitalWrite(13,HIGH);
    delay(200);
    digitalWrite(13,LOW);
    delay(200);
}
```

- Sketch 4: Report an error.

```
void setup() {
    // put your setup code here, to run once:
    pinMode(13,OUTPUT);
}

digitalWrite(13,HIGH);
delay(1000);
digitalWrite(13,LOW);
delay(1000);

void loop() {
    // put your main code here, to run repeatedly:
}
```

With the help of these sketches, we can summarize several features of `setup-loop`.

- `loop()` will be run repeatedly after the board is powered up.
- `setup()` will run only once after the board is powered up.
- After the board is powered up, `setup()` will run first, followed by `loop()`.
- The code needs to be written within the `{}` scope of `setup()` or `loop()`, out of the framework will be an error.

Note: Statements such as `digitalWrite(13,HIGH)` are used to control the on-board LED, and we will talk about their usage in detail in later chapters.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Sketch Writing Rule

If you ask a friend to turn on the lights for you, you can say "Turn on the lights.", or "Lights on, bro.", you can use any tone of voice you want.

However, if you want the Arduino board to do something for you, you need to follow the Arduino program writing rules to type in the commands.

This chapter contains the basic rules of the Arduino language and will help you understand how to translate natural language into code.

Of course, this is a process that takes time to get familiar with, and it is also the most error-prone part of the process for newbies, so if you make mistakes often, it's okay, just try a few more times.

Semicolon ;

Just like writing a letter, where you write a period at the end of each sentence as the end, the Arduino language requires you to use ; to tell the board the end of the command.

Take the familiar "onboard LED blinking" example. A healthy sketch should look like this.

Example:

```
void setup() {
    // put your setup code here, to run once:
    pinMode(13,OUTPUT);
}

void loop() {
    // put your main code here, to run repeatedly:
    digitalWrite(13,HIGH);
    delay(500);
    digitalWrite(13,LOW);
    delay(500);
}
```

Next, let's take a look at the following two sketches and guess if they can be correctly recognized by Arduino before running them.

Sketch A:

```
void setup() {
    // put your setup code here, to run once:
    pinMode(13,OUTPUT);
}

void loop() {
    // put your main code here, to run repeatedly:
    digitalWrite(13,HIGH)
    delay(500)
    digitalWrite(13,LOW)
    delay(500)
}
```

Sketch B:

```
void setup() {
    // put your setup code here, to run once:
    pinMode(13,OUTPUT);
}

void loop() {
    // put your main code here, to run repeatedly:
    digitalWrite(13,
HIGH);  delay
    (500
    );
    digitalWrite(13,

    LOW);
            delay(500)
    ;
}
```

The result is that **Sketch A** reports an error and **Sketch B** runs.

- The errors in **Sketch A** are missing `;` and although it looks normal, the Arduino can't read it.
- **Sketch B**, looks anti-human, but in fact, indentation, line breaks and spaces in statements are things that do not exist in Arduino programs, so to the Arduino compiler, it looks the same as in the example.

However, please don't write your code as **Sketch B**, because it is usually natural people who write and view the code, so don't get yourself into trouble.

Curlybraces {}

{} is the main component of the Arduino programming language, and they must appear in pairs. A better programming convention is to insert a structure that requires curly braces by typing the right curly brace directly after typing the left curly brace, and then moving the cursor between the curly braces to insert the statement.

Comment //

Comment is the part of the sketch that the compiler ignores. They are usually used to tell others how the program works.

If we write two adjacent slashes in a line of code, the compiler will ignore anything up to the end of the line.

If we create a new sketch, it comes with two comments, and if we remove these two comments, the sketch will not be affected in any way.

```
void setup() {
    // put your setup code here, to run once:

}

void loop() {
    // put your main code here, to run repeatedly:

}
```

Comment is very useful in programming, and several common uses are listed below.

- Usage A: Tell yourself or others what this section of code does.

```
void setup() {
    pinMode(13,OUTPUT); //Set pin 13 to output mode, it controls the onboard LED
}

void loop() {
    digitalWrite(13,HIGH); // Activate the onboard LED by setting pin 13 high
    delay(500); // Status quo for 500 ms
    digitalWrite(13,LOW); // Turn off the onboard LED
    delay(500);// Status quo for 500 ms
}
```

- Usage B: Temporarily invalidate some statements (without deleting them) and uncomment them when you need to use them, so you don't have to rewrite them. This is very useful when debugging code and trying to locate program errors.

```
void setup() {
    pinMode(13,OUTPUT);
    // digitalWrite(13,HIGH);
    // delay(1000);
    // digitalWrite(13,LOW);
    // delay(1000);
}

void loop() {
    digitalWrite(13,HIGH);
    delay(200);
    digitalWrite(13,LOW);
    delay(200);
}
```

Note: Use the shortcut `Ctrl+/` to help you quickly comment or uncomment your code.

Comment /**/

Same as // for comments. This type of comment can be more than one line long, and once the compiler reads /*, it ignores anything that follows until it encounters */.

Example 1:

```
/* Blink */

void setup() {
    pinMode(13,OUTPUT);
}

void loop() {
    /*
    The following code will blink the onboard LED
    You can modify the number in delay() to change the blinking frequency
    */
    digitalWrite(13,HIGH);
    delay(500);
    digitalWrite(13,LOW);
    delay(500);
}
```

#define

This is a useful C++ tool.

```
#define identifier token-string
```

The compiler automatically replaces `identifier` with `token-string` when it reads it, which is usually used for constant definitions.

As an example, here is a sketch that uses define, which improves the readability of the code.

```
#define ONBOARD_LED 13
#define DELAY_TIME 500

void setup() {
    pinMode(ONBOARD_LED,OUTPUT);
}

void loop() {
    digitalWrite(ONBOARD_LED,HIGH);
    delay(DELAY_TIME);
    digitalWrite(ONBOARD_LED,LOW);
    delay(DELAY_TIME);
}
```

To the compiler, it actually looks like this.

```
void setup() {
    pinMode(13,OUTPUT);
}

void loop() {
    digitalWrite(13,HIGH);
    delay(500);
    digitalWrite(13,LOW);
    delay(500);
}
```

We can see that the `identifier` is replaced and does not exist inside the program. Therefore, there are several caveats when using it.

1. A `token-string` can only be modified manually and cannot be converted into other values by arithmetic in the program.
2. Avoid using symbols such as `;`. For example.

```
#define ONBOARD_LED 13;

void setup() {
    pinMode(ONBOARD_LED,OUTPUT);
}

void loop() {
    digitalWrite(ONBOARD_LED,HIGH);
}
```

The compiler will recognize it as the following, which is what will be reported as an error.

```
void setup() {
    pinMode(13;,OUTPUT);
}

void loop() {
    digitalWrite(13;,HIGH);
}
```

Note: A naming convention for `#define` is to capitalize `identifier` to avoid confusion with variables.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.

- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Variable

The variable is one of the most powerful and critical tools in a program. It helps us to store and call data in our programs.

The following sketch file uses variables. It stores the pin numbers of the on-board LED in the variable `ledPin` and a number "500" in the variable `delayTime`.

```
int ledPin = 13;
int delayTime = 500;

void setup() {
    pinMode(ledPin,OUTPUT);
}

void loop() {
    digitalWrite(ledPin,HIGH);
    delay(delayTime);
    digitalWrite(ledPin,LOW);
    delay(delayTime);
}
```

Wait, is this a duplicate of what `#define` does? The answer is NO.

- The role of `#define` is to simply and directly replace text, it is not considered by the compiler as part of the program.
- A `variable`, on the other hand, exists within the program and is used to store and call value. A variable can also modify its value within the program, something that a define cannot do.

The sketch file below self-adds to the variable and it will cause the on-board LED to blink longer after each blink.

```
int ledPin = 13;
int delayTime = 500;

void setup() {
    pinMode(ledPin,OUTPUT);
}

void loop() {
    digitalWrite(ledPin,HIGH);
    delay(delayTime);
    digitalWrite(ledPin,LOW);
    delay(delayTime);
    delayTime = delayTime+200; //Each execution increments the value by 200
}
```

Declare a variable

Declaring a variable means creating a variable.

To declare a variable, you need two things: the data type, and the variable name. The data type needs to be separated from the variable by a space, and the variable declaration needs to be terminated by a `;`.

Let's use this variable as an example.

```
int delayTime;
```

Data Type

Here `int` is a data type called integer type, which can be used to store integers from -32768 to 32766. It can also not be used to store decimals.

Variables can hold different kinds of data other than integers. The Arduino language (which, remember, is C++) has built-in support for a few of them (only the most frequently used and useful are listed here):

- `float`: Store a decimal number, for example 3.1415926.
- `byte`: Can hold numbers from 0 to 255.
- `boolean`: Holds only two possible values, `True` or `False`, even though it occupies a byte in memory.
- `char`: Holds a number from -127 to 127. Because it is marked as a `char` the compiler will try to match it to a character from the .
- `string`: Can stores a string of characters, e.g. `Halloween`.

Variable Name

You can set the variable to any name you want, such as `i`, `apple`, `Bruce`, `R2D2`, `Sectumsempra`, but there are some basic rules to follow.

1. describe what it is used for. Here, I named the variable delayTime, so you can easily understand what it does. It works fine if I name the variable `barryAllen`, but it confuses the person looking at the code.
2. Use regular nomenclature. You can use CamelCase like I did, with the initial T in `delayTime` so that it is easy to see that the variable consists of two words. Also, you can use UnderScoreCase to write the variable as `delay_time`. It doesn't affect the program's running, but it would help the programmer to read the code if you use the nomenclature you prefer.
3. Don't use keywords. Similar to what happens when we type "int", the Arduino IDE will color it to remind you that it is a word with a special purpose and cannot be used as a variable name. Change the name of the variable if it is colored.
4. Special symbols are not allowed. For example, space, #, $, /, +, %, etc. The combination of English letters (case sensitive), underscores, and numbers (but numbers cannot be used as the first character of a variable name) is rich enough.

Assign a value to a variable

Once we have declared the variable, it is time to store the data. We use the assignment operator (i.e. =) to put value into the variable.

We can assign values to the variable as soon as we declare it.

```
int delayTime = 500;
```

It is also possible to assign a new value to it at some time.

```
int delayTime; // no value
delayTime = 500; // value is 500
delayTime = delayTime +200; // value is 700
```

Basic Project

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.2 Lesson 01: Button Module

In this lesson, you will learn how a button interacts with an LED using Arduino. We'll see how pressing the button lights up the LED and releasing it turns off the LED. This project is ideal for beginners as it provides a practical understanding of input and output operations on the Arduino platform.

Required Components

In this project, we need the following components.

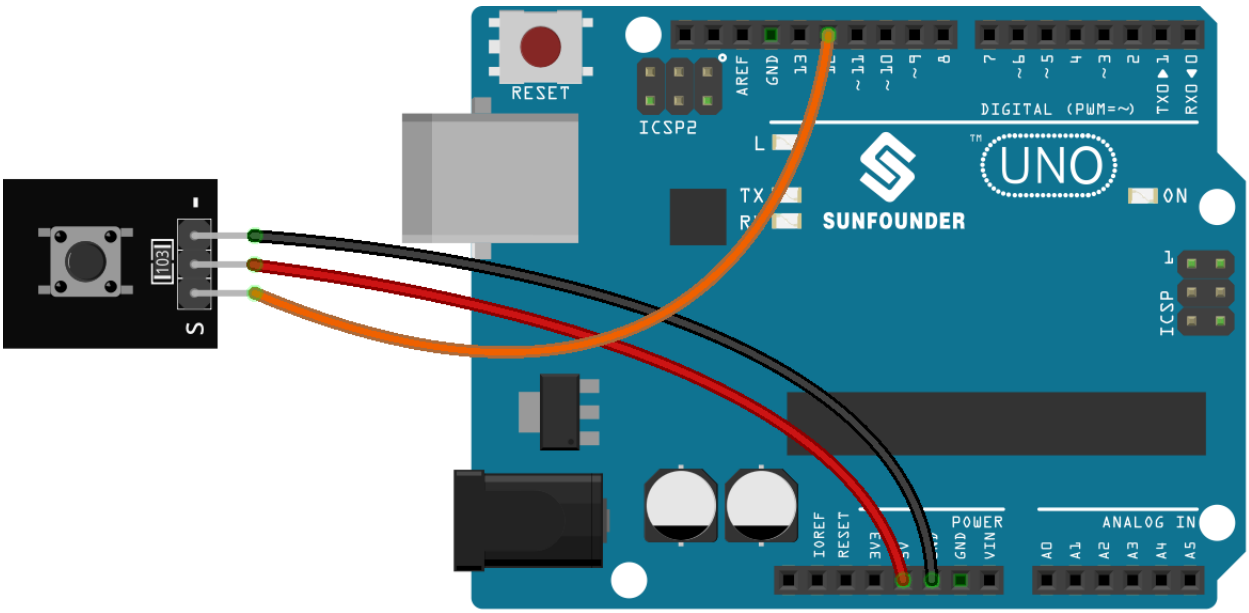
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Button Module	-

Wiring



Code

Code Analysis

1. Initialization of Pins

   The pins for the button and LED are defined and initialized. The `buttonPin` is set as an input to read the button's state, and `ledPin` is set as an output to control the LED.

   Note: Most Arduino boards have a pin connected to an on-board LED in series with a resistor. The constant `LED_BUILTIN` is the number of the pin to which the on-board LED is connected. Most boards have this LED connected to digital pin 13.

```
const int buttonPin = 12;         // Pin number for the button
const int ledPin = LED_BUILTIN;   // Pin number for the LED
int buttonState = 0;  // Variable to hold the current state of the button
```

2. Setup Function

   This function runs once and sets up the pin modes. `pinMode(buttonPin, INPUT)` configures the button pin as an input. `pinMode(ledPin, OUTPUT)` sets the LED pin as an output.

```
void setup() {
  pinMode(buttonPin, INPUT);  // Initialize buttonPin as an input pin
  pinMode(ledPin, OUTPUT);    // Initialize ledPin as an output pin
}
```

3. Main Loop Function

   This is the core of the program where the button state is continuously read and the LED state is controlled. `digitalRead(buttonPin)` reads the button's state. If the button is pressed (state is LOW), the LED is turned

on by `digitalWrite(ledPin, HIGH)`. If not pressed, the LED is turned off (`digitalWrite(ledPin, LOW)`).

The *button module* used in this project has an internal pull-up resistor (see its *schematic diagram*), causing the button to be at a low level when pressed and remain at a high level when released.

```
void loop() {
  // Read the current state of the button
  buttonState = digitalRead(buttonPin);

  // Check if the button is pressed (LOW)
  if (buttonState == LOW) {
    digitalWrite(ledPin, HIGH);  // Turn the LED on
  } else {
    digitalWrite(ledPin, LOW);  // Turn the LED off
  }
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.3 Lesson 02: Capacitive Soil Moisture Module

In this lesson, you will learn how to connect a capacitive soil moisture sensor to an Arduino and interpret its readings. The project includes reading the sensor's analog output with the Arduino and understanding that lower readings indicate higher soil moisture levels. You'll gain practical experience in handling analog input and serial communication with the Arduino by using the provided code as a hands-on example.

Required Components

In this project, we need the following components.

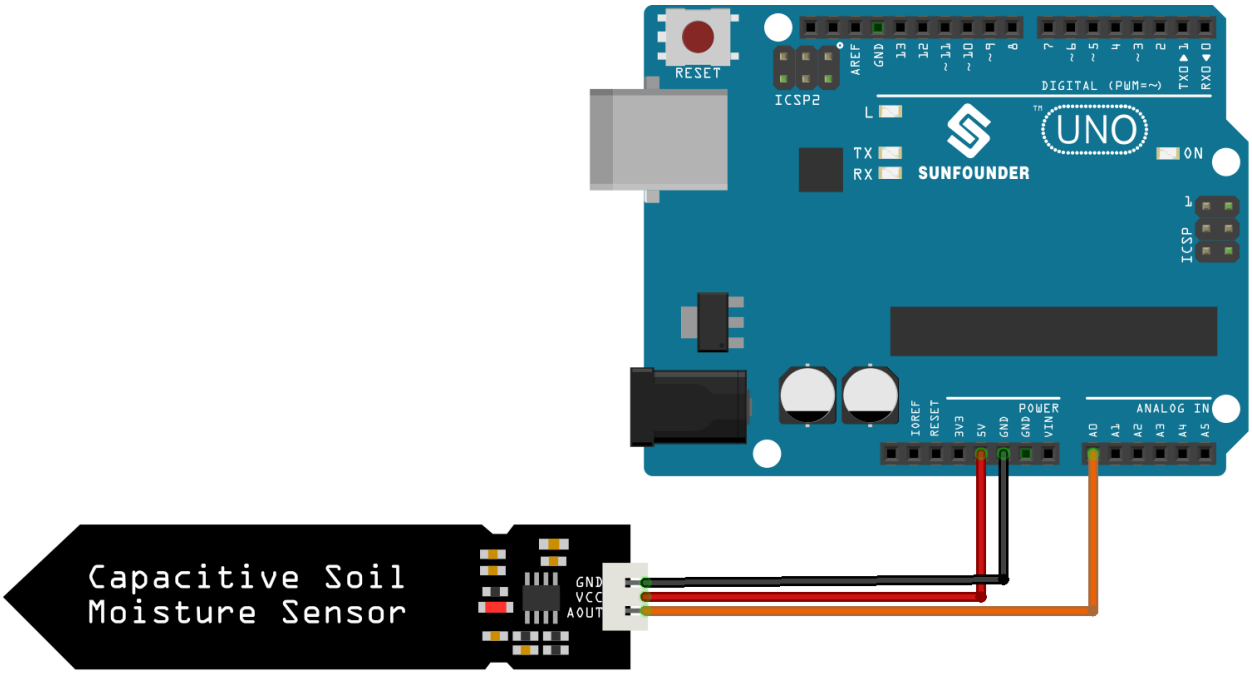
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Capacitive Soil Moisture Module	

Wiring



Code

Code Analysis

1. Defining the sensor pin:

   This line of code declares a constant integer `sensorPin` and assigns it the value of `A0`, which is the analog input pin the sensor is connected to.

   ```
   const int sensorPin = A0;
   ```

2. Setup function:

   The `setup()` function is executed once when the program starts. It initializes serial communication at 9600 baud rate. This setup is necessary for sending data to the serial monitor.

   ```
   void setup() {
     Serial.begin(9600);
   }
   ```

3. Loop function:

The `loop()` function runs continuously after `setup()`. It reads the sensor value from pin A0 using `analogRead()` and prints this value to the serial monitor. The `delay(500)` statement pauses the loop for 500 milliseconds before the next reading, thus controlling the rate of data acquisition.

```
void loop() {
  Serial.println(analogRead(A0));
  delay(500);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.4 Lesson 03: Flame Sensor Module

In this lesson, you will learn how to integrate a flame sensor with an Arduino board to detect the presence of fire. We will see how the flame sensor, when detecting a flame, triggers the Arduino's built-in LED to light up and sends a warning message to the serial monitor. Conversely, in the absence of a flame, the LED stays off, and a different message is relayed to the monitor. This project is an excellent starting point for beginners, offering a comprehensive understanding of how to manage digital inputs and outputs on the Arduino platform. It provides a hands-on approach to learning about sensor integration and real-time response mechanisms in an Arduino-based system.

Required Components

In this project, we need the following components.

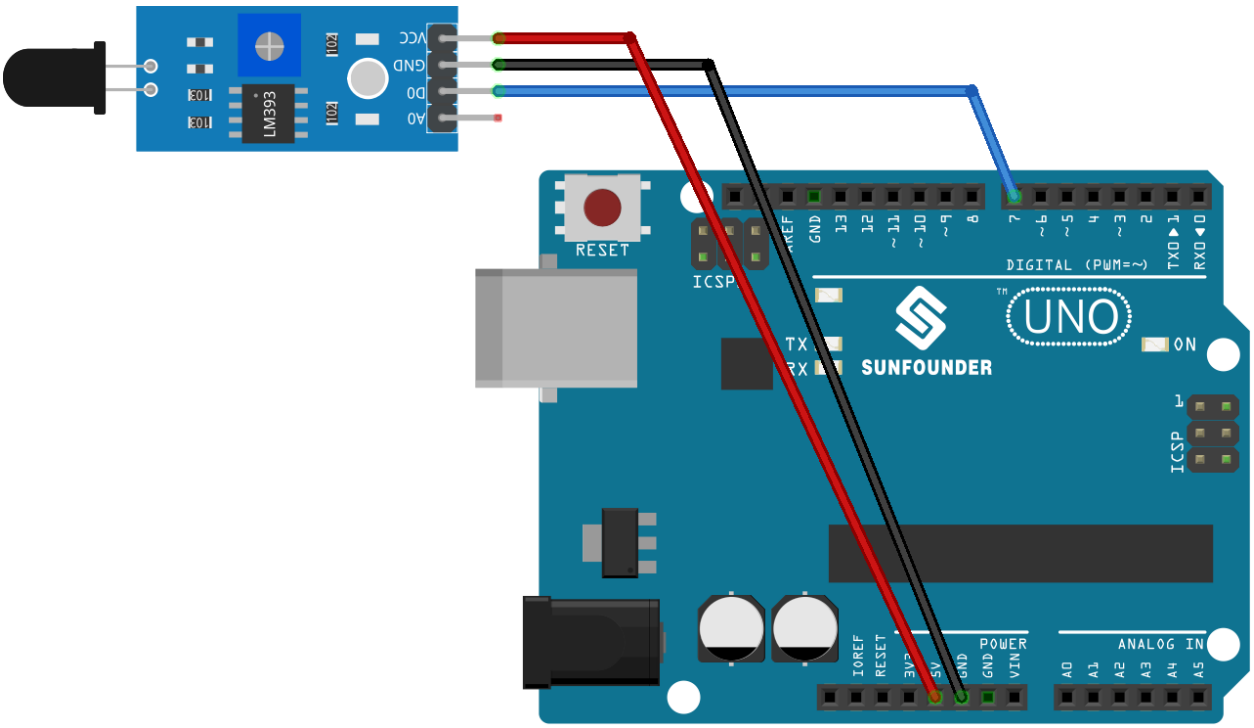
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Flame Sensor Module	

Wiring



Code

Code Analysis

1. The first line of code is a constant integer declaration for the flame sensor pin. We use the digital pin 7 to read the output from the flame sensor.

```
const int sensorPin = 7;
```

2. The `setup()` function initializes the flame sensor pin as an input and the built-in LED pin as an output. It also starts the serial communication at a baud rate of 9600 for printing messages to the serial monitor.

```
void setup() {
  pinMode(sensorPin, INPUT);      // Set the flame sensor pin as input
  pinMode(LED_BUILTIN, OUTPUT);   // Set the built-in LED pin as output
  Serial.begin(9600);             // Initialize the serial monitor at a baud rate of␣
↪9600
}
```

3. The `loop()` function is where we continuously check the status of the flame sensor. If the sensor detects a flame, the built-in LED is turned on and a message is printed to the serial monitor. If no flame is detected, the LED is turned off and a different message is printed. The process repeats every 100 milliseconds.

Note: You can change the threshold for detecting flames by adjusting the potentiometer on the flame sensor module.

```
void loop() {
  // Check if the sensor is detecting a fire
  if (digitalRead(sensorPin) == 0) {
    digitalWrite(LED_BUILTIN, HIGH);  // Turn on the built-in LED
    Serial.println("** Fire detected!!! **");
  } else {
    digitalWrite(LED_BUILTIN, LOW);  // Turn off the built-in LED
    Serial.println("No Fire detected");
  }
  delay(100);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.5 Lesson 04: Gas Sensor Module (MQ-2)

In this lesson, you will learn how to use the MQ-2 Gas Sensor with an Arduino Uno to measure gas concentrations. We'll explore how the sensor reads analog output values ranging from 0 to 1023, which represent the concentration of gases in the air. This project is essential for understanding environmental sensing and analog signal processing in Arduino, as well as a great introduction to working with sensors and interpreting their outputs. We'll discuss the importance of preheating the sensor for accurate readings and delve into the basics of serial communication for data visualization. This lesson is ideal for beginners interested in Arduino and environmental monitoring projects.

Required Components

In this project, we need the following components.

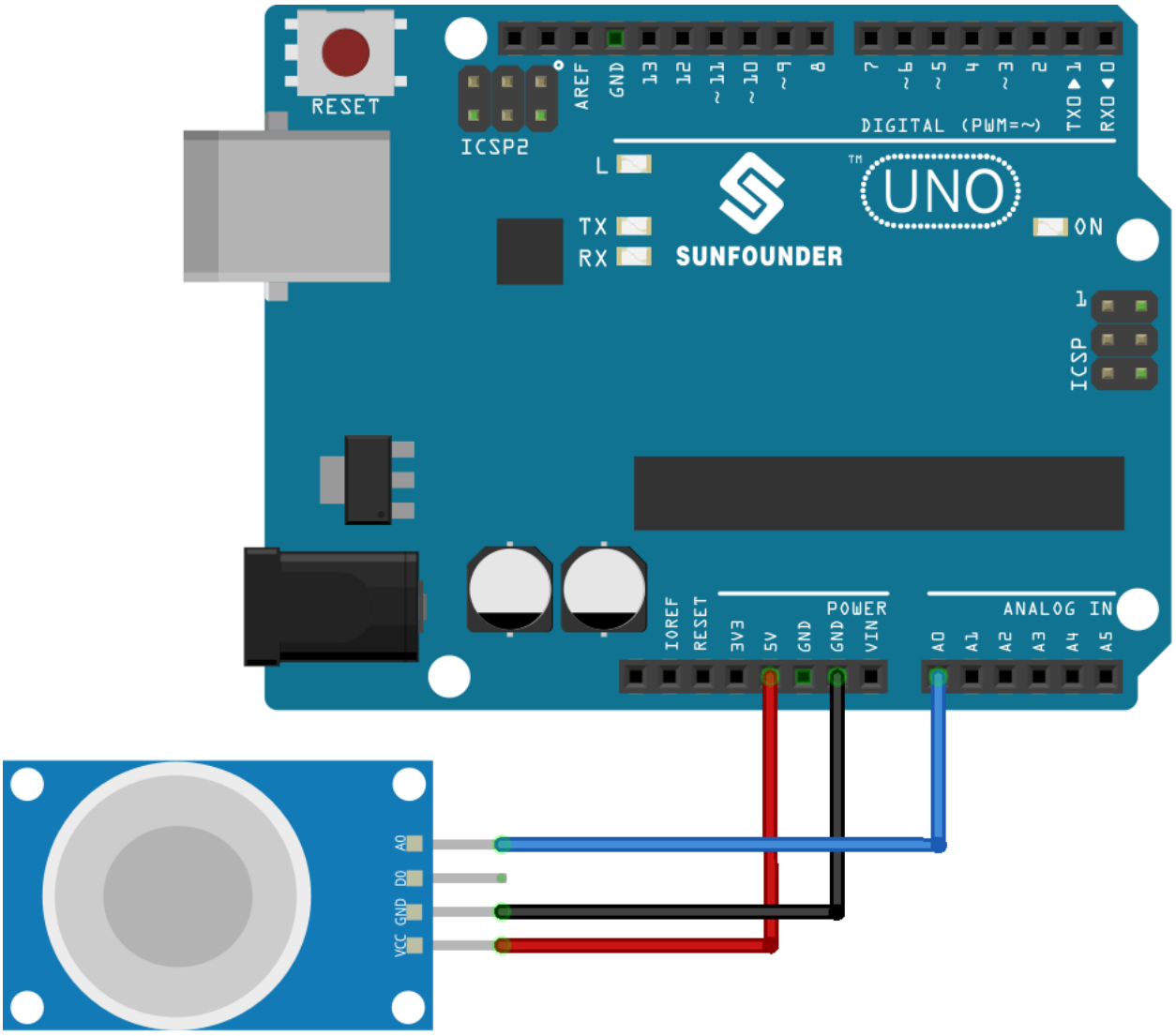
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Gas/Smoke Sensor Module (MQ2)	

Wiring



Code

Code Analysis

1. The first line of code is a constant integer declaration for the gas sensor pin. We use the analog pin A0 to read the output from the gas sensor.

```
const int sensorPin = A0;
```

2. The `setup()` function is where we initialize our serial communication at a baud rate of 9600. This is necessary to print the readings from the gas sensor to the serial monitor.

```
void setup() {
  Serial.begin(9600);  // Start serial communication at 9600 baud rate
}
```

3. The `loop()` function is where we continuously read the analog value from the gas sensor and print it to the serial monitor. We use the `analogRead()` function to read the analog value from the sensor. We then wait for 50 milliseconds before the next reading. This delay gives some breathing space for the serial monitor to process the data.

Note: MQ2 is a heating-driven sensor that usually requires preheating before use. During the preheating period, the sensor typically reads high and gradually decreases until it stabilizes.

```
void loop() {
  Serial.print("Analog output: ");
  Serial.println(analogRead(sensorPin));  // Read the analog value of the gas␣
↪sensor and print it to the serial monitor
  delay(50);                              // Wait for 50 milliseconds
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.6 Lesson 05: Gyroscope & Accelerometer Module (MPU6050)

In this lesson, you will learn how to use the MPU6050 sensor with an Arduino to measure acceleration, rotation, and temperature. We'll explore initializing the sensor, setting its ranges, and reading data to display on the serial monitor. This project offers a hands-on approach to working with motion sensors and integrating them with Arduino, perfect for those looking to dive into the world of electronics and sensor data handling.

Required Components

In this project, we need the following components.

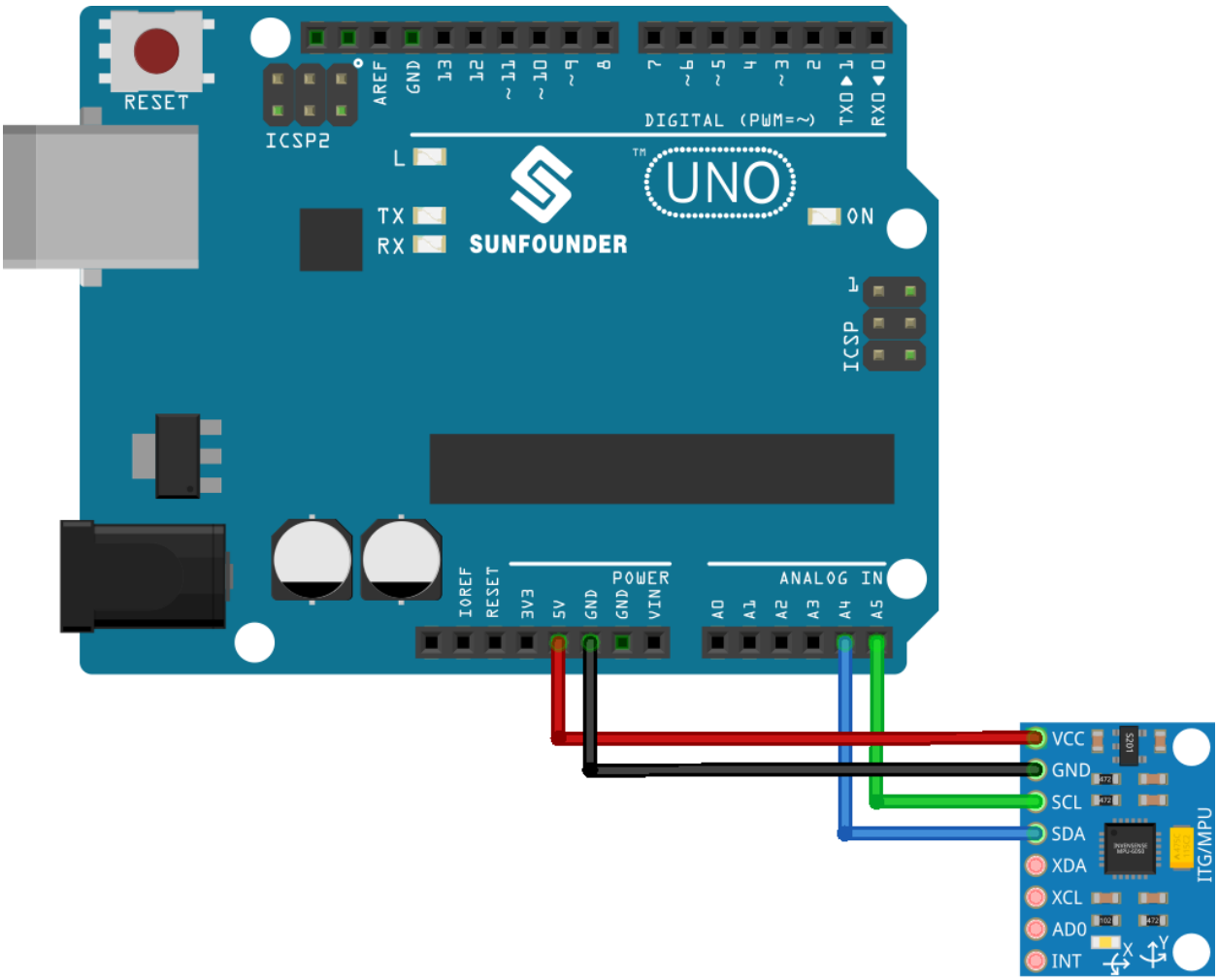
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Gyroscope & Accelerometer Module (MPU6050)	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"Adafruit MPU6050"** and install it.

Code Analysis

1. The code starts by including the necessary libraries and creating an object for the MPU6050 sensor. This code uses the Adafruit_MPU6050 library, Adafruit_Sensor library, and Wire library. The `Adafruit_MPU6050` library is used to interact with the MPU6050 sensor and retrieve acceleration, rotation, and temperature data. The `Adafruit_Sensor` library provides a common interface for various types of sensors. The `Wire` library is used for I2C communication, which is necessary to communicate with the MPU6050 sensor.

   Note: To install the library, use the Arduino Library Manager and search for **"Adafruit MPU6050"** and install it.

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
Adafruit_MPU6050 mpu;
```

2. The `setup()` function initializes the serial communication and checks if the sensor is detected. If the sensor is not found, the Arduino enters an infinite loop with a "Failed to find MPU6050 chip" message. If found, the accelerometer range, gyro range, and filter bandwidth are set, and a delay is added for stability.

```
void setup(void) {
  // Initialize the serial communication
  Serial.begin(9600);

  // Check if the MPU6050 sensor is detected
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");

  // set accelerometer range to +-8G
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);

  // set gyro range to +- 500 deg/s
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);

  // set filter bandwidth to 21 Hz
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);

  // Add a delay for stability
  delay(100);
}
```

3. In the `loop()` function, the program creates events to store the sensor readings and then retrieves the readings. The acceleration, rotation, and temperature values are then printed to the serial monitor.

```
void loop() {
  // Get new sensor events with the readings
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  // Print out the acceleration, rotation, and temperature readings
  // ...

  // Add a delay to avoid flooding the serial monitor
  delay(1000);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.7 Lesson 06: Hall Sensor Module

In this lesson, you will learn how a Hall sensor detects magnetic fields using Arduino. We'll explore how to read the sensor's analog signal with Arduino Uno, interpreting the values to determine the polarity of a magnetic field. You'll understand the workings of the Hall sensor, and how the Arduino board processes and displays these readings in real-time.

Required Components

In this project, we need the following components.

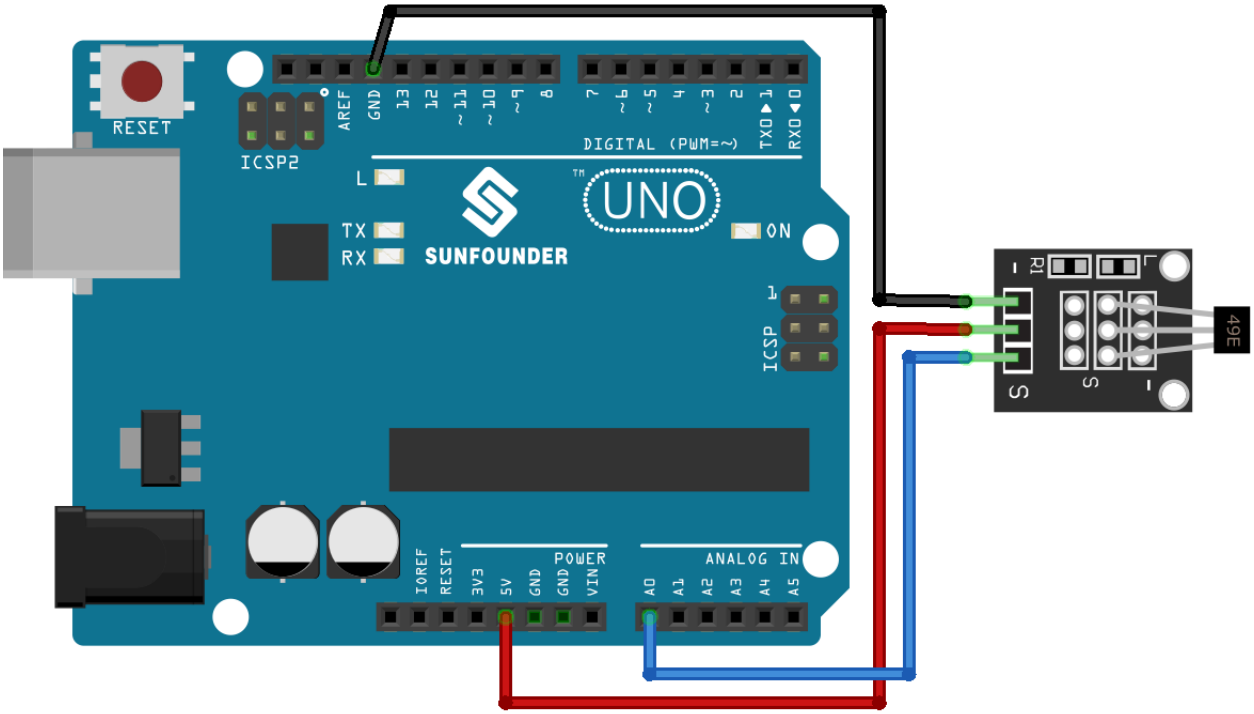
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Hall Sensor Module	-

Wiring



Code

Code Analysis

1. Setting up the Hall Sensor

```
const int hallSensorPin = A0;  // Pin A0 connected to the Hall sensor output
void setup() {
  Serial.begin(9600);             // Initialize serial communication at 9600 bps
  pinMode(hallSensorPin, INPUT);  // Set hall sensor pin as input
}
```

The hall sensor's output is connected to pin A0 on the Arduino. The `setup()` function is used to initialize serial communication at 9600 bits per second (bps) for displaying data on the serial monitor. The `pinMode()` function is used to configure A0 as an input pin.

2. Reading from the Hall Sensor and Determining Polarity

The Hall sensor module is equipped with a 49E linear Hall effect sensor, which can measure the polarity of the magnetic field's north and south poles as well as the relative strength of the magnetic field. If you place a magnet's south pole near the side marked with 49E (the side with text engraved on it), the value read by the code will increase linearly in proportion to the applied magnetic field strength. Conversely, if you place a north pole near this side, the value read by the code will decrease linearly in proportion to that magnetic field strength. For more details, please refer to *Hall Sensor Module*.

```
void loop() {
  int sensorValue = analogRead(hallSensorPin);  // Read analog value from Hall␣
↪sensor
```

(continues on next page)

(continued from previous page)

```
  Serial.print(sensorValue);                    // Output raw sensor value to␣
↪Serial Monitor
  delay(200);                                   // Delay for 200 milliseconds

  // Determine magnetic pole based on sensor value
  if (sensorValue >= 700) {
    Serial.print(" - South pole detected");  // South pole detected if value >= 700
  } else if (sensorValue <= 300) {
    Serial.print(" - North pole detected");  // North pole detected if value <= 300
  }

  Serial.println();  // New line for next output
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.8 Lesson 07: Infrared Speed Sensor Module

In this lesson, you will learn how to measure motor speed using a speed sensor module with an Arduino Uno. We'll cover setting up the motor and sensor, programming the Arduino to calculate revolutions per second, and displaying the data. This project is great for intermediate learners as it provides hands-on experience with real-time data processing and motor control on the Arduino platform.

Required Components

In this project, we need the following components.

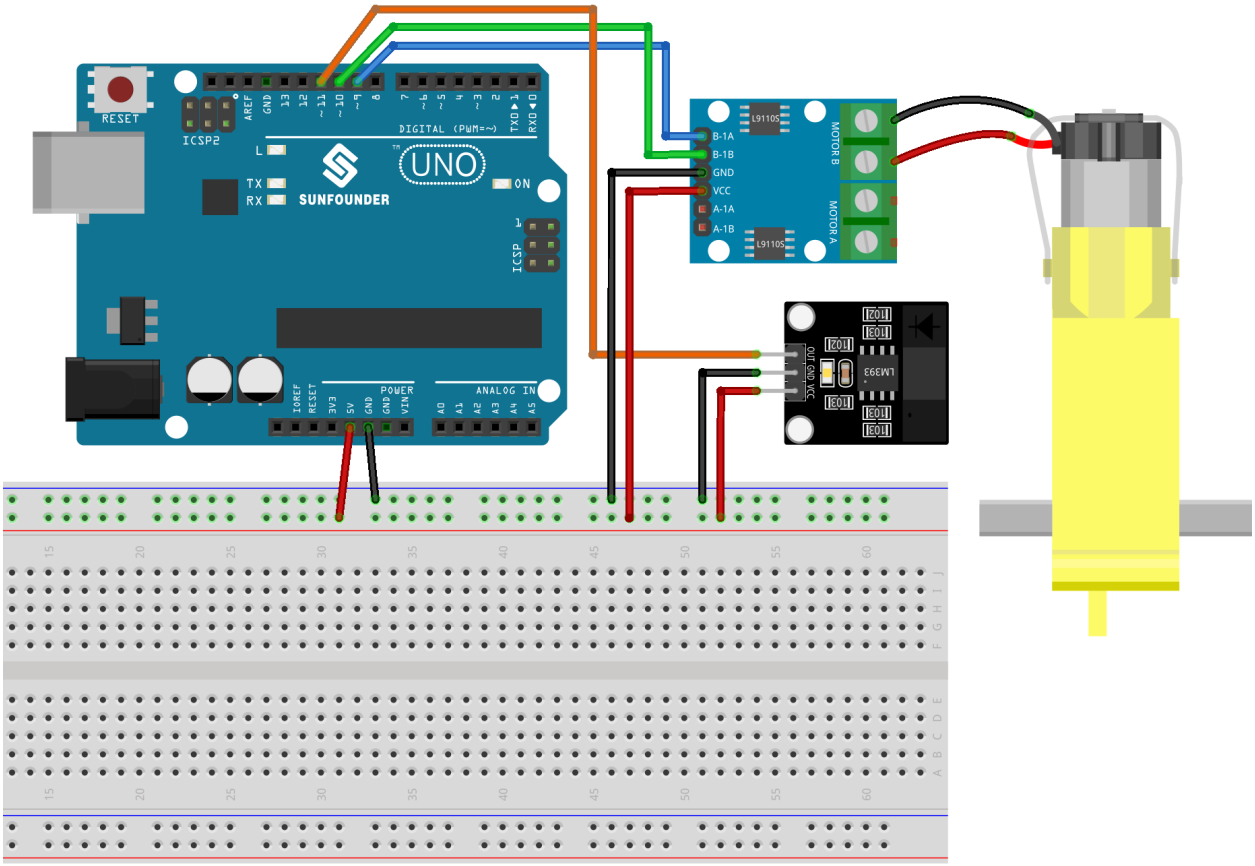
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Breadboard	
Infrared Speed Sensor Module	
TT Motor	-
L9110 Motor Driver Module	-

Wiring



Code

Code Analysis

1. Setting up the pins and initializing variables. Here, we define the pins for the motor and the speed sensor. We also initialize variables that will be used to measure and calculate the speed of the motor.

```
// Define the sensor and motor pins
const int sensorPin = 11;
```

(continues on next page)

(continued from previous page)

```
const int motorB_1A = 9;
const int motorB_2A = 10;

// Define variables for measuring speed
unsigned long start_time = 0;
unsigned long end_time = 0;
int steps = 0;
float steps_old = 0;
float temp = 0;
float rps = 0;
```

2. Initialization in the `setup()` function. This section sets up the serial communication, configures the pins' modes, and sets the initial motor speed.

```
void setup() {
  Serial.begin(9600);
  pinMode(sensorPin, INPUT);
  pinMode(motorB_1A, OUTPUT);
  pinMode(motorB_2A, OUTPUT);
  analogWrite(motorB_1A, 160);
  analogWrite(motorB_2A, 0);
}
```

3. Measuring the motor's speed in the `loop()` function. In this segment, the motor's steps are measured for a duration of 1 second. These steps are then used to calculate the revolutions per second (rps), which is then printed to the serial monitor.

   `millis()` returns the number of milliseconds passed since the Arduino board began running the current program.

```
void loop() {
  start_time = millis();
  end_time = start_time + 1000;
  while (millis() < end_time) {
    if (digitalRead(sensorPin)) {
      steps = steps + 1;
      while (digitalRead(sensorPin))
        ;
    }
  }
  temp = steps - steps_old;
  steps_old = steps;
  rps = (temp / 20);
  Serial.print("rps:");
  Serial.println(rps);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.9 Lesson 08: IR Obstacle Avoidance Sensor Module

In this lesson, you will learn how to use an Infrared obstacle avoidance sensor with an Arduino Uno. We will explore how to read digital signals from the sensor to detect obstacles. You'll see how the sensor's red indicator light illuminates in the presence of obstacles and how it sends a low-level signal to the Arduino. This lesson is perfect for beginners, providing hands-on experience with reading digital inputs and practicing serial communication on the Arduino platform.

Required Components

In this project, we need the following components.

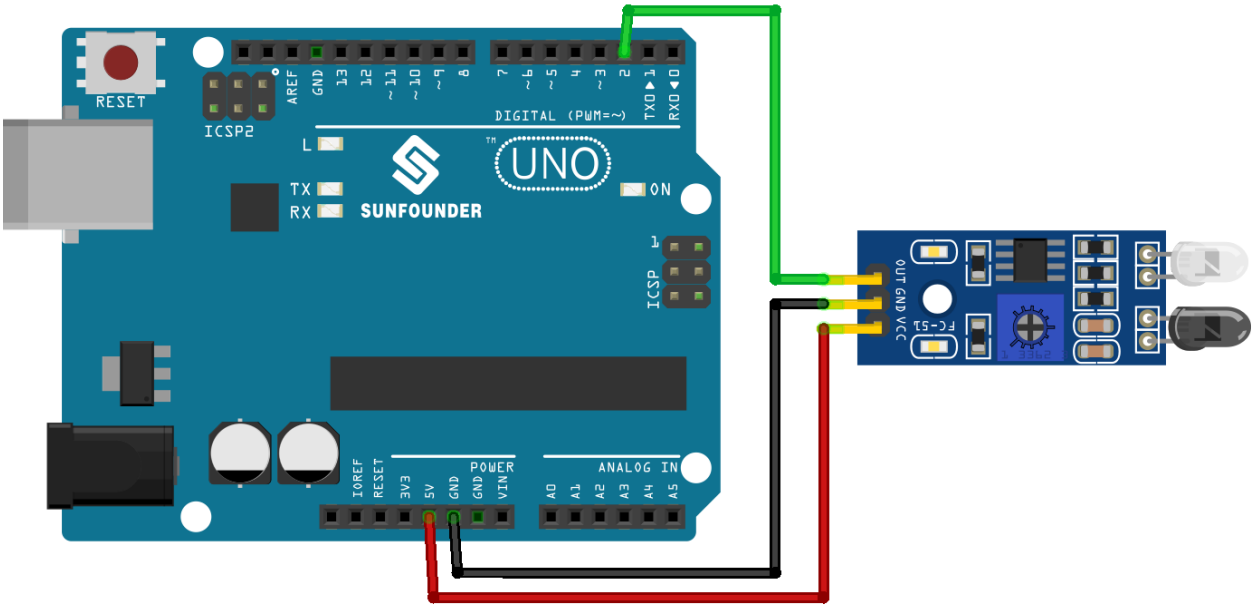
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
IR Obstacle Avoidance Sensor Module	

Wiring



Code

Code Analysis

1. Define pin number for sensor connection:

   ```
   const int sensorPin = 2;
   ```

   Connect the sensor's output pin to Arduino pin 2.

2. Setup serial communication and define sensor pin as input:

   ```
   void setup() {
     pinMode(sensorPin, INPUT);
     Serial.begin(9600);
   }
   ```

   Initialize serial communication at 9600 baud rate to print to serial monitor. Set sensor pin as input to read input signal.

3. Read sensor value and print to serial monitor:

   ```
   void loop() {
     Serial.println(digitalRead(sensorPin));
     delay(50);
   }
   ```

   Continuously read digital value from sensor pin using `digitalRead()` and print value to serial monitor using `Serial.println()`. Add 50ms delay between prints for better viewing.

Note: If the sensor is not working properly, adjust the IR transmitter and receiver to make them parallel. Additionally, you can adjust the detection range using the built-in potentiometer.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.10 Lesson 09: Joystick Module

In this lesson, you will learn how to read values from a joystick module using an Arduino Uno. We will explore connecting the joystick's X and Y axes to the Arduino and how to display their values on the serial monitor. Additionally, we'll cover the usage of a switch button on the joystick. This project is perfect for beginners, offering hands-on experience with analog and digital inputs on the Arduino platform.

Required Components

In this project, we need the following components.

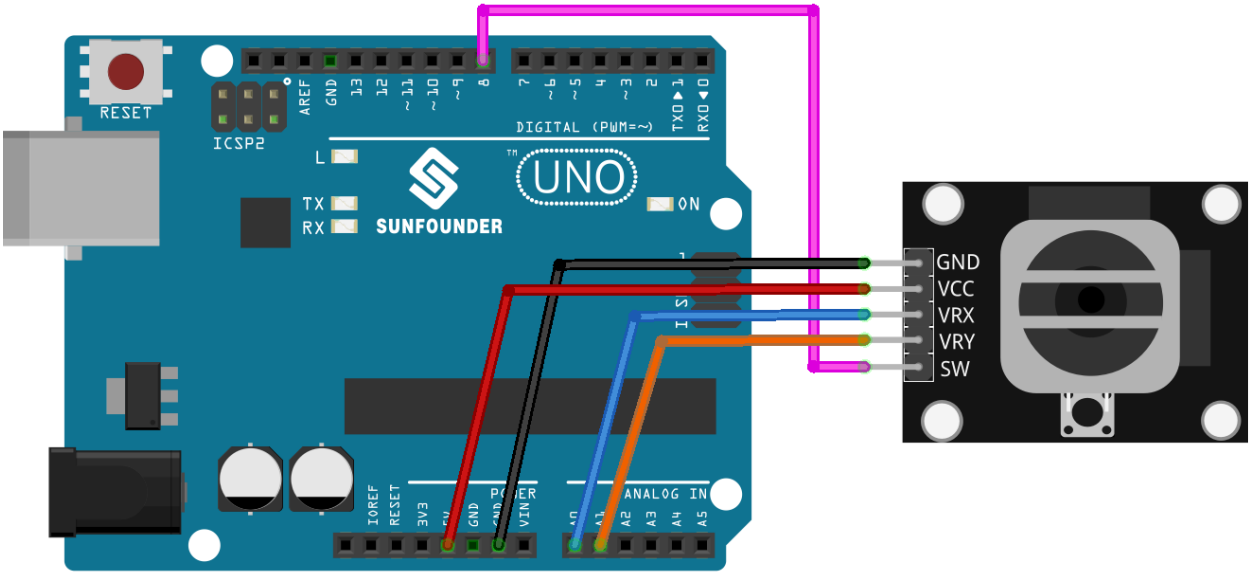
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Joystick Module	

Wiring



Code

Code Analysis

1. Pin Definitions:

```
const int xPin = A0;  //the VRX attach to
const int yPin = A1;  //the VRY attach to
const int swPin = 8;  //the SW attach to
```

Constants for the joystick pins are defined. `xPin` and `yPin` are analog pins for the joystick's X and Y axes. `swPin` is a digital pin for the joystick's switch.

2. Setup Function:

```
void setup() {
  pinMode(swPin, INPUT_PULLUP);
  Serial.begin(9600);
}
```

Initializes `swPin` as an input with a pull-up resistor, essential for the switch's functionality. Starts serial communication at 9600 baud.

3. Main Loop:

```
void loop() {
  Serial.print("X: ");
  Serial.print(analogRead(xPin));  // print the value of VRX
  Serial.print("|Y: ");
  Serial.print(analogRead(yPin));  // print the value of VRY
  Serial.print("|Z: ");
  Serial.println(digitalRead(swPin));  // print the value of SW
```

(continues on next page)

(continued from previous page)

```
  delay(50);
}
```

Continuously reads and prints the values from the joystick's axes and switch to the Serial Monitor, with a delay of 50 ms between readings.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.11 Lesson 10: PCF8591 ADC DAC Converter Module

In this lesson, you'll learn how to connect the Arduino Uno R4 (or R3) with a PCF8591 ADC DAC Converter Module. We'll cover reading analog values from input AIN0, sending these values to the DAC(AOUT), and displaying both the raw and voltage-converted readings on the serial monitor. The module's potentiometer is connected to AIN0 using jumper caps, and the D2 LED on the module is connected to AOUT, so you can see that the brightness of D2 LED changes as you rotate the potentiometer.

Required Components

In this project, we need the following components.

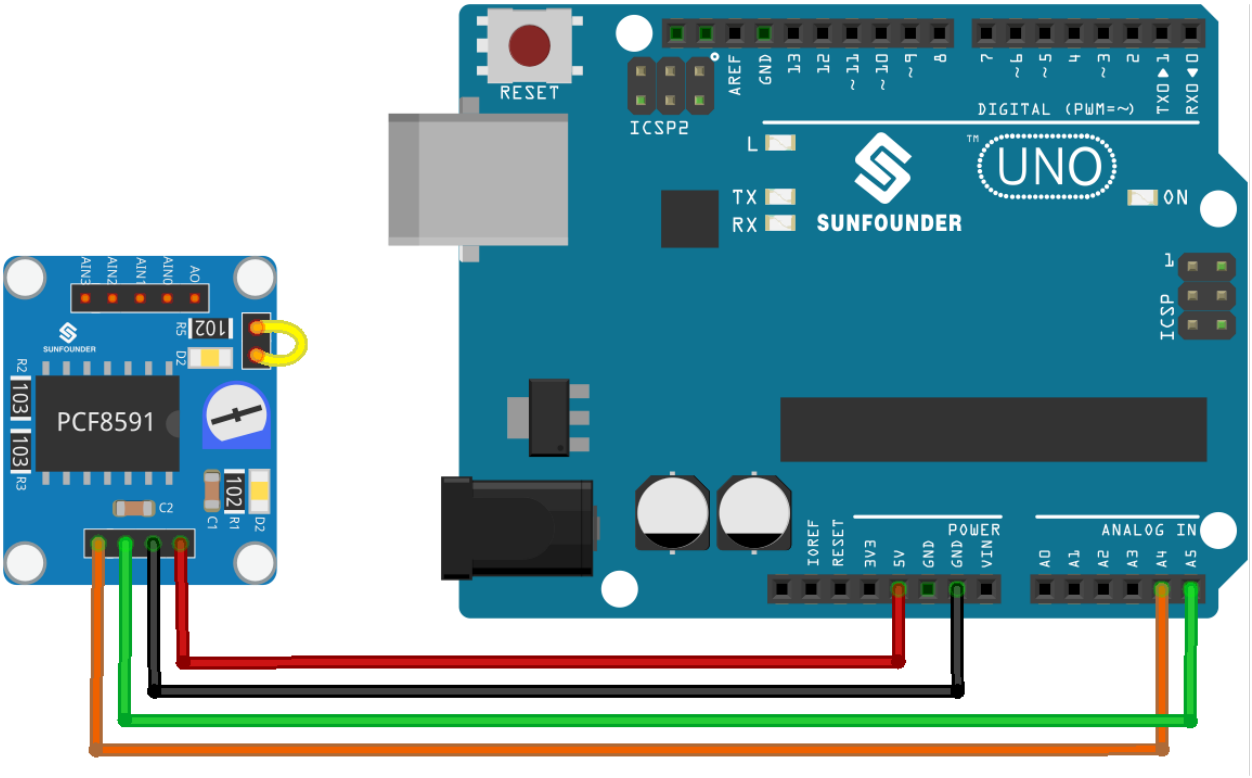
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
PCF8591 ADC DAC Converter Module	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"Adafruit PCF8591"** and install it.

Code Analysis

1. **Including the Library and Defining Constants**

   Note: To install the library, use the Arduino Library Manager and search for **"Adafruit PCF8591"** and install it.

   ```
   // Include Adafruit PCF8591 library
   #include <Adafruit_PCF8591.h>
   // Define the reference voltage for ADC conversion
   #define ADC_REFERENCE_VOLTAGE 5.0
   ```

   This section includes the Adafruit PCF8591 library, which provides functions for interacting with the PCF8591 module. The ADC reference voltage is set to 5.0 volts, which is the maximum voltage that the ADC can measure.

2. **Setting Up the PCF8591 Module**

```
// Create an instance of the PCF8591 module
Adafruit_PCF8591 pcf = Adafruit_PCF8591();
void setup() {
  Serial.begin(9600);
  Serial.println("# Adafruit PCF8591 demo");
  if (!pcf.begin()) {
    Serial.println("# PCF8591 not found!");
    while (1) delay(10);
  }
  Serial.println("# PCF8591 found");
  pcf.enableDAC(true);
}
```

In the setup function, serial communication is started, and an instance of the PCF8591 module is created. The `pcf.begin()` function checks if the module is connected properly. If not, it prints an error message and halts the program. If the module is found, it enables the DAC.

3. **Reading from ADC and Writing to DAC**

```
void loop() {
  AIN0 = pcf.analogRead(0);
  pcf.analogWrite(AIN0);
  Serial.print("AIN0: ");
  Serial.print(AIN0);
  Serial.print(", ");
  Serial.print(int_to_volts(AIN0, 8, ADC_REFERENCE_VOLTAGE));
  Serial.println("V");
  delay(500);
}
```

The loop function continuously reads the analog value from AIN0 (analog input 0) of the PCF8591 module, then writes this value back to the DAC. It also prints the raw value and the voltage-converted value of AIN0 to the Serial Monitor.

Jumper caps link the module's potentiometer to AIN0, and the D2 LED is connected to AOUT; please refer to the PCF8591 module *schematic* for details. The brightness of the LED changes as the potentiometer is rotated.

4. **Digital to Voltage Conversion Function**

```
float int_to_volts(uint16_t dac_value, uint8_t bits, float logic_level) {
  return (((float)dac_value / ((1 << bits) - 1)) * logic_level);
}
```

This function converts the digital value back to its corresponding voltage. It takes the digital value (`dac_value`), the number of bits of resolution (`bits`), and the logic level voltage (`logic_level`) as arguments. The formula used is a standard approach to convert a digital value to its equivalent voltage.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.

- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.12 Lesson 11: Photoresistor Module

In this lesson, you will learn how to measure light intensity using a photoresistor sensor with an Arduino Uno. We'll cover reading and displaying the analog values from the sensor, which reflect the amount of light it detects. This project is ideal for beginners as it provides hands-on experience in working with sensors and understanding analog input on the Arduino platform. You'll also improve your proficiency in serial communication by outputting sensor readings to the serial monitor.

Required Components

In this project, we need the following components.

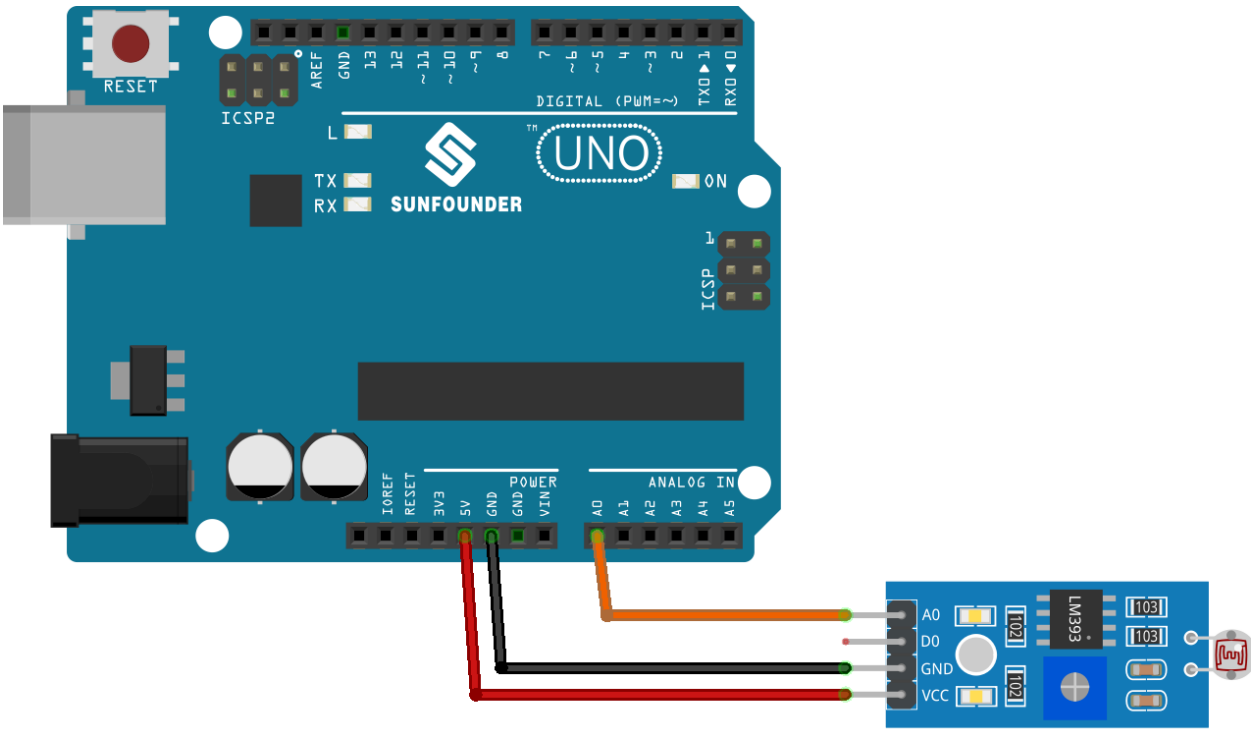
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Photoresistor Module	

Wiring



Code

Code Analysis

1. **Setting Up the Sensor Pin and Serial Communication**

   We start by defining the sensor pin and initializing serial communication in the setup function. The photoresistor is connected to the analog pin A0.

```
const int sensorPin = A0;  // Pin connected to the photoresistor

void setup() {
  Serial.begin(9600);  // Start serial communication at 9600 baud rate
}
```

2. **Reading and Displaying Sensor Data**

   In the loop function, we continuously read the analog value from the sensor and print it to the Serial Monitor. We also add a short delay to stabilize the readings.

```
void loop() {
  Serial.println(analogRead(sensorPin));  // Read and print the analog value
  delay(50);                              // Short delay to stabilize readings
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.13 Lesson 12: PIR Motion Module (HC-SR501)

In this lesson, you will learn how to use a PIR (Passive Infrared) motion sensor with an Arduino Uno. We'll see how the sensor detects movement and sends a signal to the Arduino, which then triggers a response. This project is ideal for beginners as it provides hands-on experience with digital inputs, serial communication, and conditional programming on the Arduino platform.

Required Components

In this project, we need the following components.

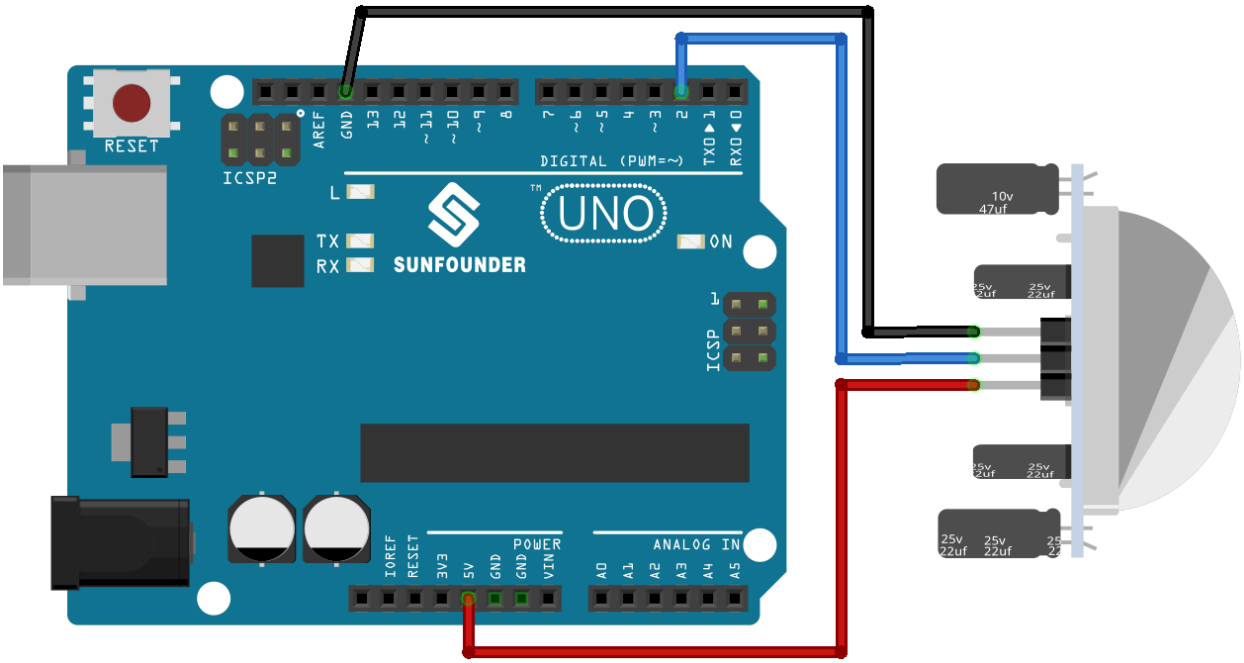
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
PIR Motion Module (HC-SR501)	-

Wiring



Code

Code Analysis

1. Setting up the PIR Sensor Pin. The pin for the PIR sensor is defined as pin 2.

```
const int pirPin = 2;
int state = 0;
```

2. Initializing the PIR Sensor. In the `setup()` function, the PIR sensor pin is set as an input. This allows the Arduino to read the state of the PIR sensor.

```
void setup() {
  pinMode(pirPin, INPUT);
  Serial.begin(9600);
}
```

3. Reading from the PIR Sensor and Displaying the Results. In the `loop()` function, the state of the PIR sensor is continuously read.

```
void loop() {
  state = digitalRead(pirPin);
  if (state == HIGH) {
    Serial.println("Somebody here!");
  } else {
    Serial.println("Monitoring...");
    delay(100);
  }
}
```

If the state is `HIGH`, meaning motion is detected, a message "Somebody here!" is printed to the serial monitor. Otherwise, "Monitoring…" is printed.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.14 Lesson 13: Potentiometer Module

In this lesson, you'll learn how to read the analog value of a potentiometer with an Arduino Uno. We'll connect the potentiometer to pin A0 and use the Arduino to measure its value from 0 to 1023. This tutorial will walk you through setting up the circuit, writing code to read the sensor, and displaying the readings on the serial monitor. It's a great project for beginners, providing hands-on experience with analog input and serial communication on the Arduino platform.

Required Components

In this project, we need the following components.

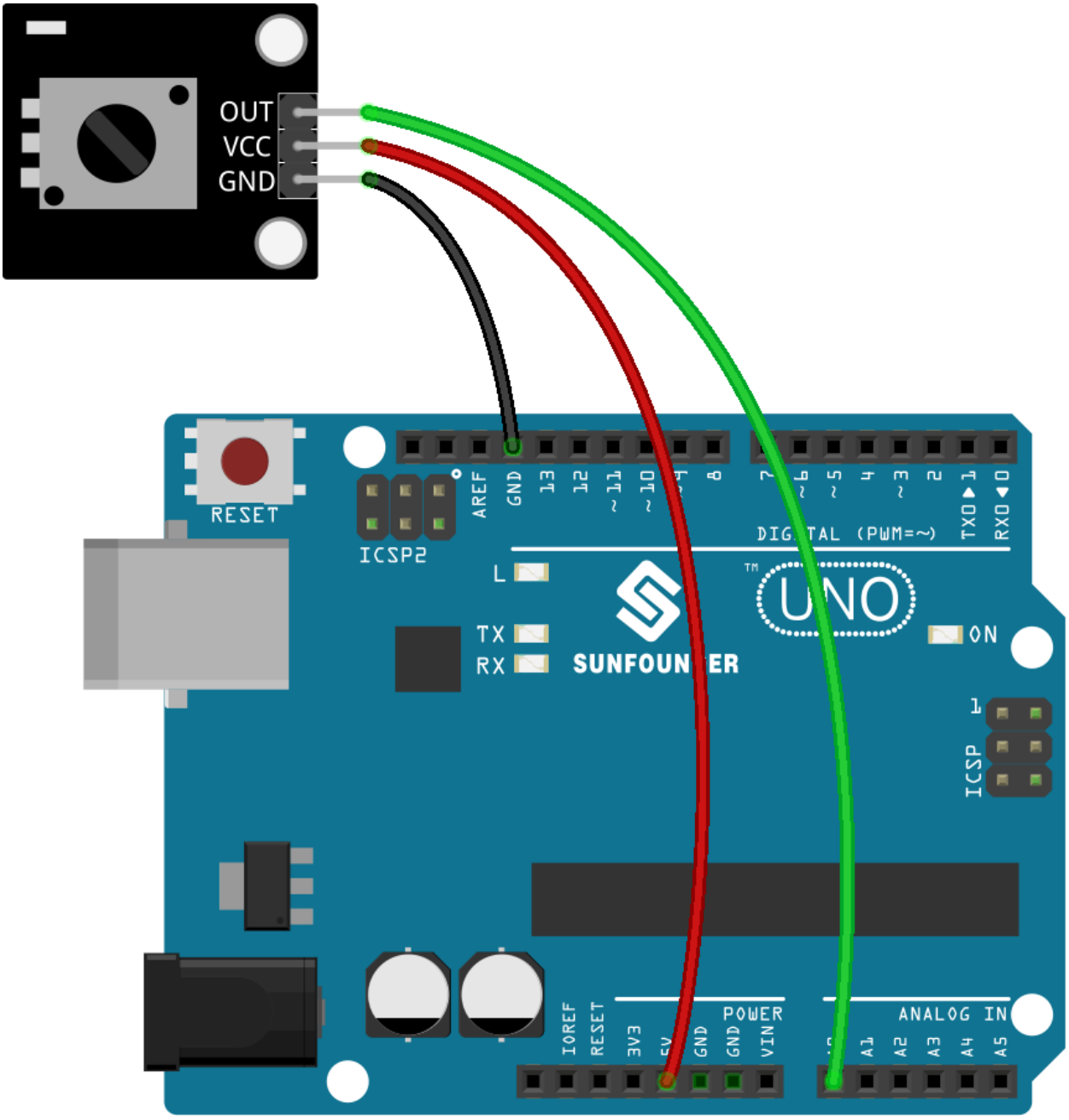
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Potentiometer Module	

Wiring


OUT
VCC
GND
RESET
ICSP2
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM=~)
L
TX
RX
ON
ICSP
IOREF
RESET
3V3
5V
GND
GND
VIN
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN

Code

Code Analysis

1. This line of code defines the pin number to which the potentiometer is connected on the Arduino board.

```
const int sensorPin = A0;
```

2. The `setup()` function is a special function in Arduino that is executed only once when the Arduino is powered on or reset. In this project, the `Serial.begin(9600)` command initiates serial communication at a baud rate of 9600.

```
void setup() {
  Serial.begin(9600);
}
```

3. The `loop()` function is the main function where the program runs repeatedly. In this function, the `analogRead()` function reads the analog value from the potentiometer and prints it to the serial monitor using `Serial.println()`. The `delay(50)` command makes the program wait for 50 milliseconds before taking the next reading.

```
void loop() {
  Serial.println(analogRead(sensorPin));
  delay(50);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.15 Lesson 14: Pulse Oximeter and Heart Rate Sensor Module (MAX30102)

In this lesson, you'll learn how to measure heart rate using a MAX30102 sensor and Arduino Uno. You'll learn to set up the sensor, read infrared values, calculate BPM, and average readings over time. This project is perfect for those interested in health monitoring with Arduino, combining hardware interfacing and software logic.

Warning: This project detects heart-rate optically. This method is tricky and prone to give false readings. So please **DO NOT** use it for actual medical diagnosis.

Required Components

In this project, we need the following components.

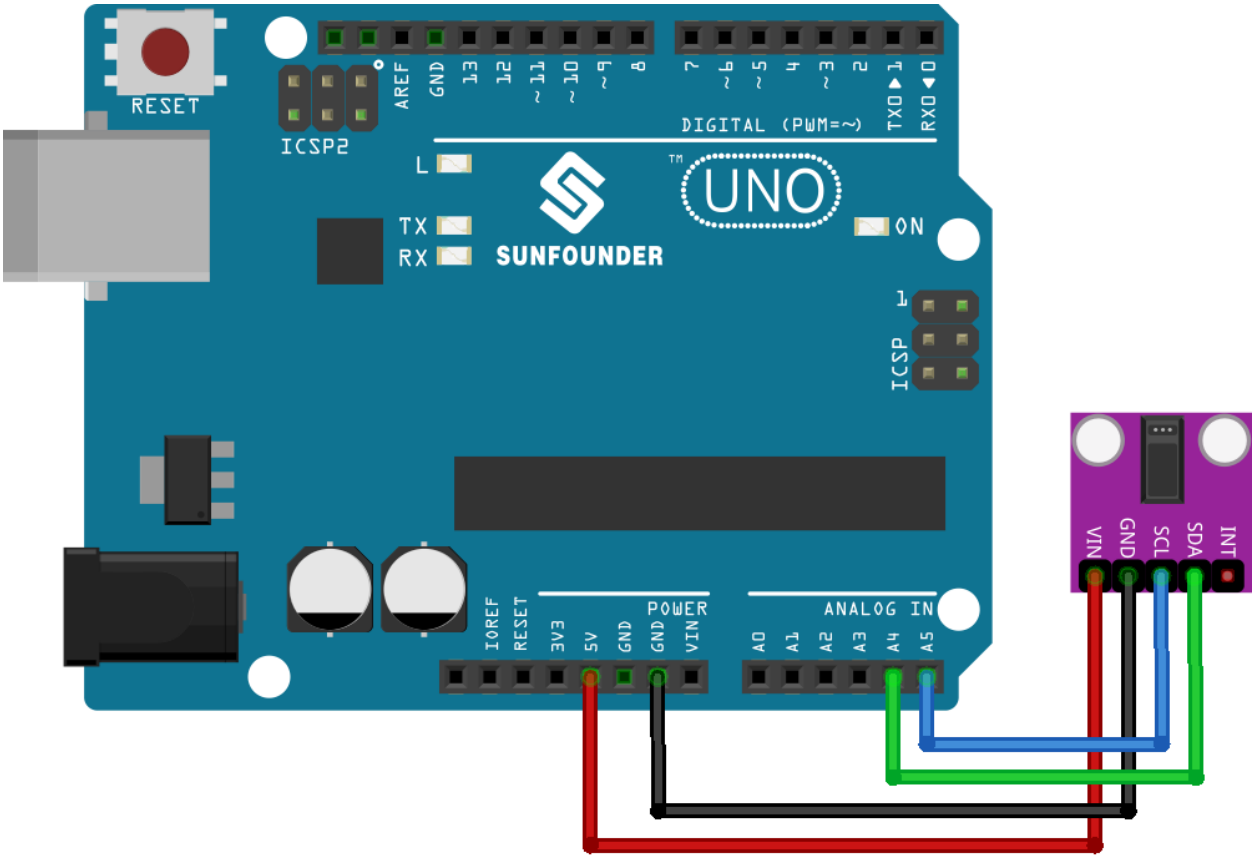
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Pulse Oximeter and Heart Rate Sensor Module (MAX30102)	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"SparkFun MAX3010x"** and install it.

Code Analysis

1. **Including Libraries & Initializing Global Variables**:

   The essential libraries are imported, the sensor object is instantiated, and global variables for data management are set.

   Note: To install the library, use the Arduino Library Manager and search for **"SparkFun MAX3010x"** and install it.

```
#include <Wire.h>
#include "MAX30105.h"
#include "heartRate.h"
MAX30105 particleSensor;
// ... (other global variables)
```

2. **Setup Function & Sensor Initialization**:

   The Serial communication is initialized at a baud rate of 9600. The sensor's connection is checked, and if successful, an initialization sequence is run. An error message is displayed if the sensor isn't detected.

```
void setup() {
  Serial.begin(9600);
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) {
    Serial.println("MAX30102 not found.");
    while (1) ;  // Infinite loop if sensor not detected.
  }
  // ... (further setup)
```

3. **Reading IR Value & Checking for Heartbeat**:

   The IR value, which is indicative of the blood flow, is fetched from the sensor. The `checkForBeat()` function assesses if a heartbeat is detected based on this value.

```
long irValue = particleSensor.getIR();
if (checkForBeat(irValue) == true) {
    // ... (heartbeat detected actions)
}
```

4. **Calculating Beats Per Minute (BPM)**:

   Upon detecting a heartbeat, the BPM is calculated based on the time difference since the last detected heartbeat. The code also ensures the BPM falls within a realistic range before updating the average.

```
long delta = millis() - lastBeat;
beatsPerMinute = 60 / (delta / 1000.0);
if (beatsPerMinute < 255 && beatsPerMinute > 20) {
```

(continues on next page)

(continued from previous page)

```
    // ... (store and average BPM)
}
```

5. **Printing Values to the Serial Monitor**:

   The IR value, current BPM, and average BPM are printed to the Serial Monitor. Additionally, the code checks if the IR value is too low, suggesting the absence of a finger.

```
//Print the IR value, current BPM value, and average BPM value to the serial monitor
Serial.print("IR=");
Serial.print(irValue);
Serial.print(", BPM=");
Serial.print(beatsPerMinute);
Serial.print(", Avg BPM=");
Serial.print(beatAvg);

if (irValue < 50000)
  Serial.print(" No finger?");
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.16 Lesson 15: Raindrop Detection Module

In this lesson, you will learn how to use a Raindrop Detection Sensor Module with an Arduino. We will see how the sensor detects rain by measuring changes in resistance caused by raindrops completing circuits on its nickel-coated surface.

Required Components

In this project, we need the following components.

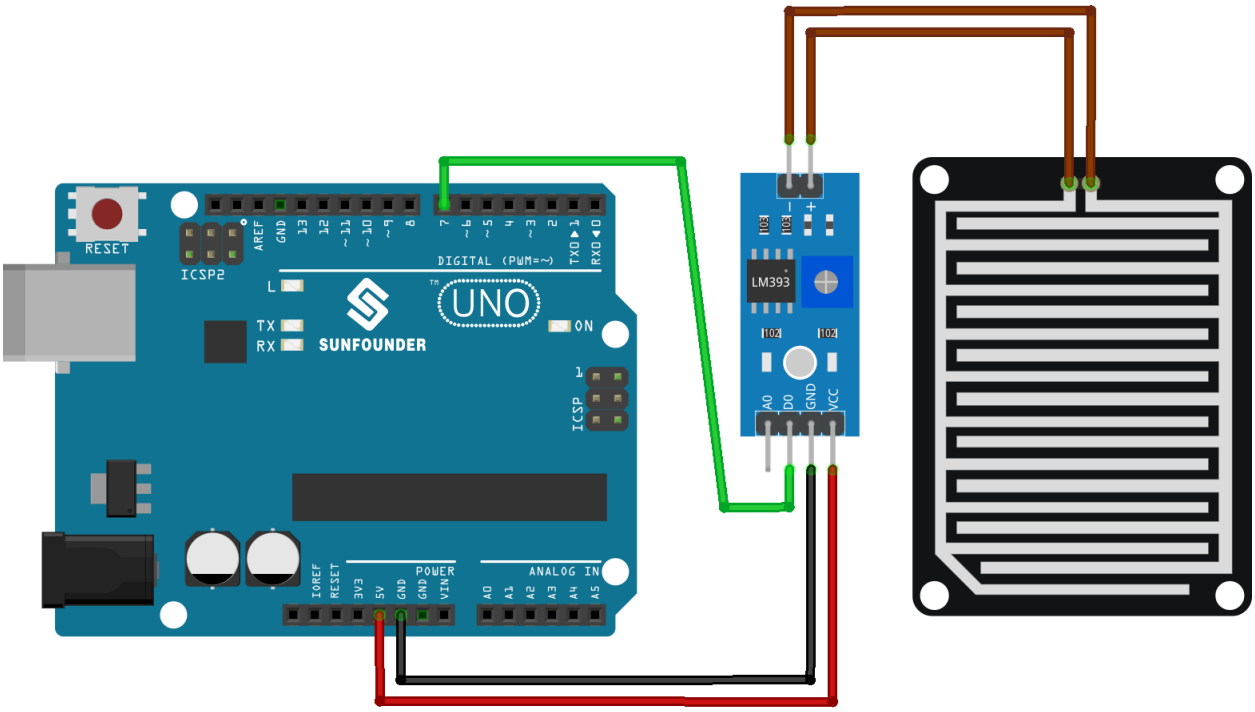
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Raindrop Detection Module	

Wiring



Code

Code Analysis

1. Defining sensor pin

   Here, a constant integer named `sensorPin` is defined and assigned the value 7. This corresponds to the digital pin on the Arduino board where the raindrops detection sensor is connected.

   ```
   const int sensorPin = 7;
   ```

2. Setting up the pin mode and initiating serial communication.

   In the `setup()` function, two essential steps are performed. Firstly, `pinMode()` is used to set the `sensorPin` as an input, enabling us to read digital values from the raindrops sensor. Secondly, serial communication is initialized with a baud rate of 9600.

   ```
   void setup() {
     pinMode(sensorPin, INPUT);
   ```

(continues on next page)

(continued from previous page)

```
  Serial.begin(9600);
}
```

3. Reading the digital value and sending it to the serial monitor.

   The `loop()` function reads the digital value from the raindrops sensor using `digitalRead()`. This value (either HIGH or LOW) is printed to the Serial Monitor. When raindrops are detected, the serial monitor will display 0; when no raindrops are detected, it will display 1. The program then waits for 50 milliseconds before the next reading.

```
void loop() {
  Serial.println(digitalRead(sensorPin));
  delay(50);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.17 Lesson 16: Real Time Clock Module (DS1302)

In this lesson, you will learn how to set up and use a Real Time Clock (RTC) module with an Arduino. We'll cover initializing the RTC DS1302 module, displaying the current date and time on the serial monitor, and ensuring accurate timekeeping. This session is ideal for those interested in time-based operations in embedded systems, offering hands-on experience with managing date and time settings, using RTC libraries, and troubleshooting common issues. This project is suitable for intermediate learners familiar with Arduino basics.

Required Components

In this project, we need the following components.

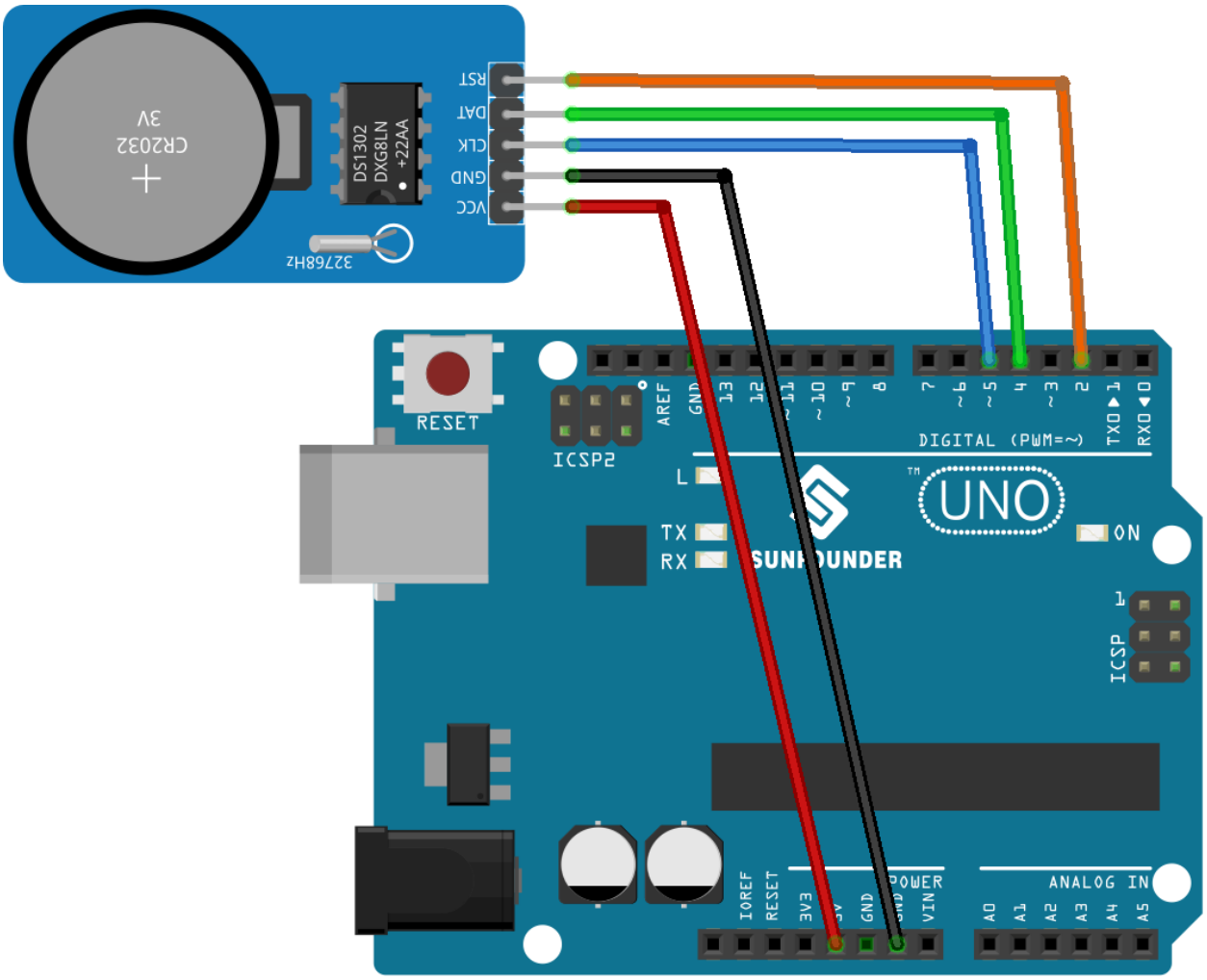
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Real Time Clock Module (DS1302)	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"Rtc by Makuna"** and install it.

Code Analysis

1. Initialization and library inclusion

   Note: To install the library, use the Arduino Library Manager and search for **"Rtc by Makuna"** and install it.

   Here, necessary libraries are included for the DS1302 RTC module.

```
#include <ThreeWire.h>
#include <RtcDS1302.h>
```

2. Define pins and create RTC instance

   Pins for communication are defined and an instance of the RTC is created.

```
const int IO = 4;     // DAT
const int SCLK = 5;   // CLK
const int CE = 2;     // RST

ThreeWire myWire(4, 5, 2);  // IO, SCLK, CE
RtcDS1302<ThreeWire> Rtc(myWire);
```

3. `setup()` function

   This function initializes the serial communication and sets up the RTC module. Various checks are made to ensure the RTC is running correctly.

```
void setup() {
  Serial.begin(9600);

  Serial.print("compiled: ");
  Serial.print(__DATE__);
  Serial.println(__TIME__);

  Rtc.Begin();

  RtcDateTime compiled = RtcDateTime(__DATE__, __TIME__);
  printDateTime(compiled);
  Serial.println();

  if (!Rtc.IsDateTimeValid()) {
    // Common Causes:
    //    1) first time you ran and the device wasn't running yet
    //    2) the battery on the device is low or even missing

    Serial.println("RTC lost confidence in the DateTime!");
    Rtc.SetDateTime(compiled);
```

(continues on next page)

(continued from previous page)

```
  }

  if (Rtc.GetIsWriteProtected()) {
    Serial.println("RTC was write protected, enabling writing now");
    Rtc.SetIsWriteProtected(false);
  }

  if (!Rtc.GetIsRunning()) {
    Serial.println("RTC was not actively running, starting now");
    Rtc.SetIsRunning(true);
  }

  RtcDateTime now = Rtc.GetDateTime();
  if (now < compiled) {
    Serial.println("RTC is older than compile time!  (Updating DateTime)");
    Rtc.SetDateTime(compiled);
  } else if (now > compiled) {
    Serial.println("RTC is newer than compile time. (this is expected)");
  } else if (now == compiled) {
    Serial.println("RTC is the same as compile time! (not expected but all is fine)
↪");
  }
}
```

4. `loop()` function

   This function periodically fetches the current date and time from the RTC and prints it on the serial monitor. It also checks if the RTC is still maintaining a valid date and time.

```
void loop() {
  RtcDateTime now = Rtc.GetDateTime();

  printDateTime(now);
  Serial.println();

  if (!now.IsValid()) {
    // Common Causes:
    //    1) the battery on the device is low or even missing and the power line␣
↪was disconnected
    Serial.println("RTC lost confidence in the DateTime!");
  }

  delay(5000);  // five seconds
}
```

5. Date and time printing function

   A helper function that takes a `RtcDateTime` object and prints the formatted date and time to the serial monitor.

```
void printDateTime(const RtcDateTime& dt) {
  char datestring[20];

  snprintf_P(datestring,
```

(continues on next page)

(continued from previous page)

```
            countof(datestring),
            PSTR("%02u/%02u/%04u %02u:%02u:%02u"),
            dt.Month(),
            dt.Day(),
            dt.Year(),
            dt.Hour(),
            dt.Minute(),
            dt.Second());
  Serial.print(datestring);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.18 Lesson 17: Rotary Encoder Module

In this lesson, you will learn how to monitor and control a rotary encoder with an Arduino Uno. We'll cover tracking the rotation direction (clockwise or counterclockwise), counting rotations, and detecting button presses on the encoder module. This project is ideal for those seeking to enhance their understanding of rotary encoders and input/output operations in Arduino, providing practical insight into physical control systems.

Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

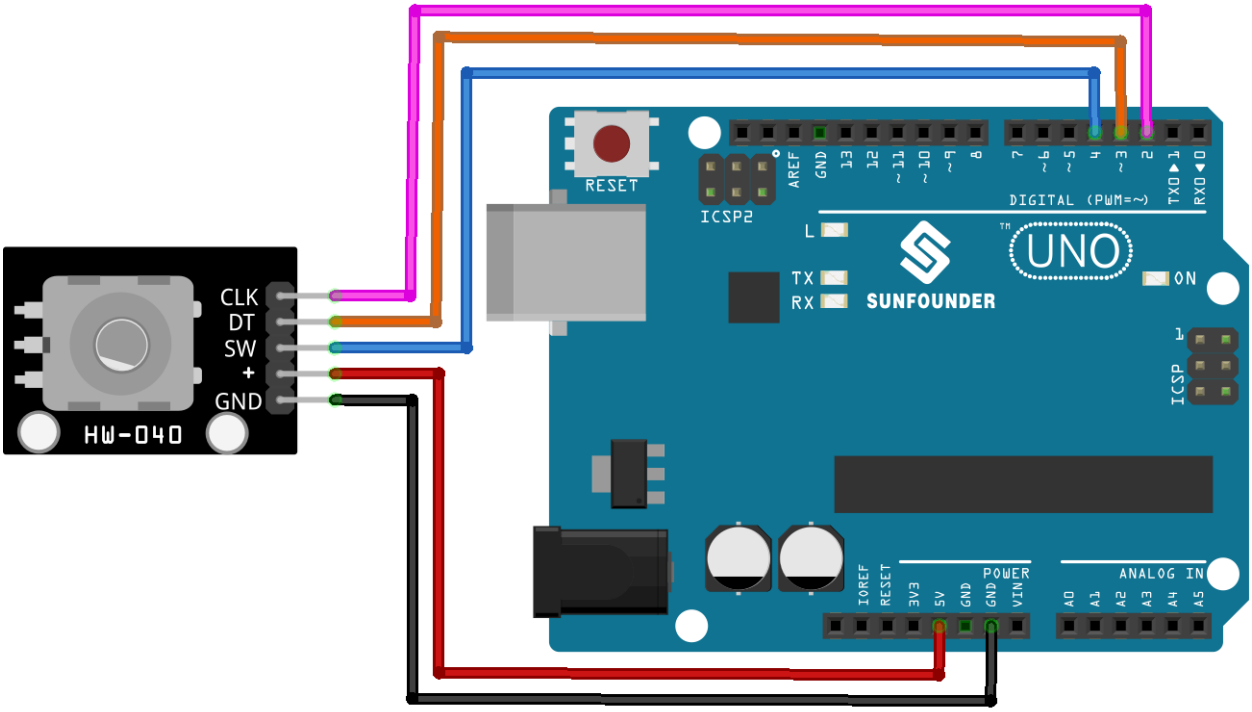
Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Rotary Encoder Module	-

- Arduino UNO R3 or R4
- *Rotary Encoder Module*

Wiring



Code

Code Analysis

1. **Setup and Initialization**

```
void setup() {
  pinMode(CLK, INPUT);
  pinMode(DT, INPUT);
  pinMode(SW, INPUT_PULLUP);
  Serial.begin(9600);
  lastStateCLK = digitalRead(CLK);
}
```

In the setup function, the digital pins connected to the encoder's CLK and DT are set as inputs. The SW pin, which is connected to the button, is set as an input with an internal pull-up resistor. This setup prevents the need for an external pull-up resistor. The Serial communication is started at a baud rate of 9600 to enable data visualization on the Serial Monitor. The initial state of the CLK pin is read and stored.

2. **Main Loop: Reading Encoder and Button State**

```
void loop() {
  currentStateCLK = digitalRead(CLK);
```

(continues on next page)

(continued from previous page)

```
  if (currentStateCLK != lastStateCLK && currentStateCLK == 1) {
    if (digitalRead(DT) != currentStateCLK) {
      counter--;
      currentDir = "CCW";
    } else {
      counter++;
      currentDir = "CW";
    }
    Serial.print("Direction: ");
    Serial.print(currentDir);
    Serial.print(" | Counter: ");
    Serial.println(counter);
  }
  lastStateCLK = currentStateCLK;
  int btnState = digitalRead(SW);
  if (btnState == LOW) {
    if (millis() - lastButtonPress > 50) {
      Serial.println("Button pressed!");
    }
    lastButtonPress = millis();
  }
  delay(1);
}
```

In the loop function, the program continually reads the current state of the CLK pin. If there's a change in the state, it implies a rotation has occurred. The direction of rotation is determined by comparing the states of CLK and DT pins. If they are different, it indicates counterclockwise (CCW) rotation; otherwise, it's clockwise (CW). The encoder's count is incremented or decremented accordingly. This information is then sent to the Serial Monitor.

The button state is read from the SW pin. If it's LOW (pressed), a debounce mechanism is implemented by checking the time elapsed since the last button press. If more than 50 milliseconds have passed, it's considered a valid press, and a message is sent to the Serial Monitor. The *delay(1)* at the end helps in debouncing.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.19 Lesson 18: Temperature Sensor Module (DS18B20)

In this lesson, you'll learn how to read temperature data from a DS18B20 sensor using Arduino. We'll cover using the DallasTemperature library to communicate with the sensor and display readings in both Celsius and Fahrenheit on the Serial Monitor. This project is ideal for Arduino beginners, providing practical experience with temperature sensors and data processing.

Required Components

In this project, we need the following components.

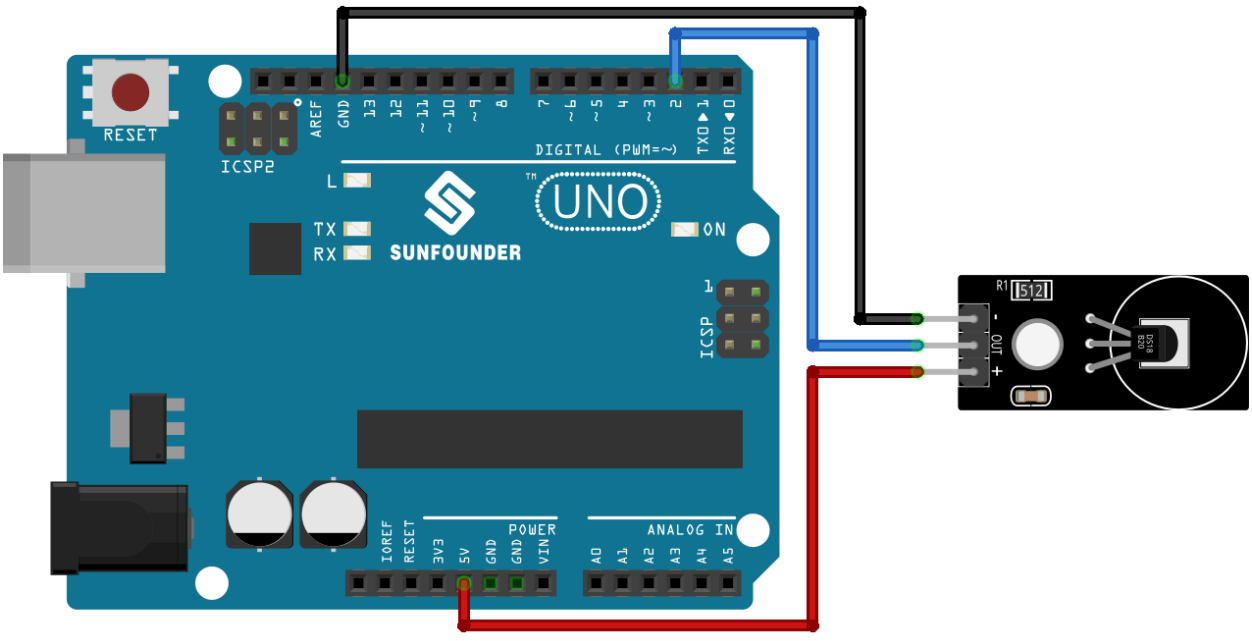
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Temperature Sensor Module (DS18B20)	-

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"DallasTemperature"** and install it.

Code Analysis

1. Library inclusion

   The inclusion of the OneWire and DallasTemperature libraries allows communication with the DS18B20 sensor.

   Note: To install the library, use the Arduino Library Manager and search for **"DallasTemperature"** and install it.

   ```
   #include <OneWire.h>
   #include <DallasTemperature.h>
   ```

2. Defining the sensor data pin

   The DS18B20 is connected to digital pin 2 of the Arduino.

   ```
   #define ONE_WIRE_BUS 2
   ```

3. Initializing the sensor

   The OneWire instance and DallasTemperature object are created and initialized.

   ```
   OneWire oneWire(ONE_WIRE_BUS);
   DallasTemperature sensors(&oneWire);
   ```

4. Setup function

   The `setup()` function initializes the sensor and sets up serial communication.

   ```
   void setup(void)
   {
      sensors.begin();       // Start up the library
      Serial.begin(9600);
   }
   ```

5. Main loop

   In the `loop()` function, the program requests temperature readings and prints them in both Celsius and Fahrenheit.

   ```
   void loop(void)
   {
      sensors.requestTemperatures();
      Serial.print("Temperature: ");
      Serial.print(sensors.getTempCByIndex(0));
      Serial.print("℃ | ");
      Serial.print((sensors.getTempCByIndex(0) * 9.0) / 5.0 + 32.0);
      Serial.println("");
   ```

(continues on next page)

(continued from previous page)

```
    delay(500);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.20 Lesson 19: Temperature and Humidity Sensor Module (DHT11)

In this lesson, you'll learn how to measure temperature and humidity, as well as calculate the heat index using a DHT11 sensor with an Arduino Uno. We'll cover reading and interpreting data from the DHT11 sensor, and displaying these values along with the heat index in both Celsius and Fahrenheit on the serial monitor. This project is perfect for Arduino beginners, providing hands-on experience with sensors and data handling in a simple yet engaging way.

Required Components

In this project, we need the following components.

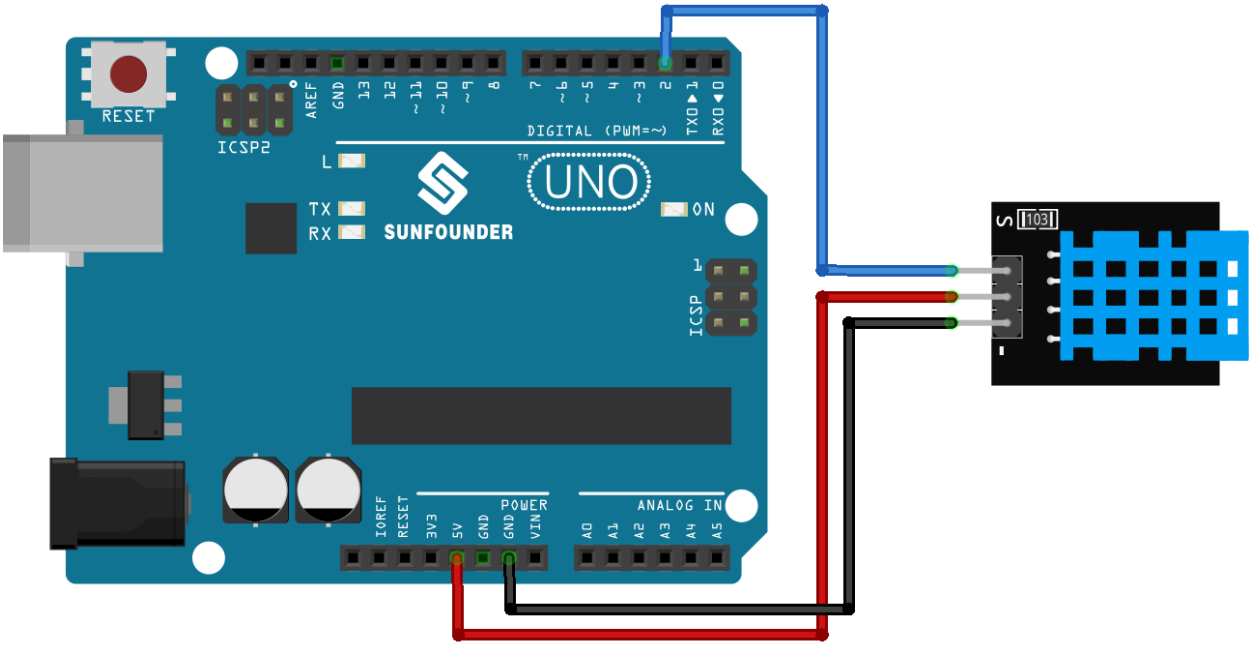
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Temperature and Humidity Sensor Module (DHT11)	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"DHT sensor library"** and install it.

Code Analysis

1. Inclusion of necessary libraries and definition of constants. This part of the code includes the DHT sensor library and defines the pin number and sensor type used in this project.

   Note: To install the library, use the Arduino Library Manager and search for **"DHT sensor library"** and install it.

   ```
   #include <DHT.h>
   #define DHTPIN 2       // Define the pin used to connect the sensor
   #define DHTTYPE DHT11  // Define the sensor type
   ```

2. Creation of DHT object. Here we create a DHT object using the defined pin number and sensor type.

   ```
   DHT dht(DHTPIN, DHTTYPE);  // Create a DHT object
   ```

3. This function is executed once when the Arduino starts. We initialize the serial communication and the DHT sensor in this function.

   ```
   void setup() {
     Serial.begin(9600);
   ```

(continues on next page)

(continued from previous page)

```
  Serial.println(F("DHT11 test!"));
  dht.begin();  // Initialize the DHT sensor
}
```

4. Main loop. The `loop()` function runs continuously after the setup function. Here, we read the humidity and temperature values, calculate the heat index, and print these values to the serial monitor. If the sensor read fails (returns NaN), it prints an error message.

Note: The is a way to measure how hot it feels outside by combining the air temperature and the humidity. It is also called the "felt air temperature" or "apparent temperature".

```
void loop() {
  delay(2000);
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  float f = dht.readTemperature(true);
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }
  float hif = dht.computeHeatIndex(f, h);
  float hic = dht.computeHeatIndex(t, h, false);
  Serial.print(F("Humidity: "));
  Serial.print(h);
  Serial.print(F("%  Temperature: "));
  Serial.print(t);
  Serial.print(F("°C "));
  Serial.print(f);
  Serial.print(F("°F  Heat index: "));
  Serial.print(hic);
  Serial.print(F("°C "));
  Serial.print(hif);
  Serial.println(F("°F"));
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.21 Lesson 20: Temperature, Humidity & Pressure Sensor (BMP280)

In this lesson, you will learn how to use the BMP280 sensor with an Arduino Uno to read atmospheric pressure, temperature, and approximate altitude. We'll cover integrating the sensor with Arduino using the Adafruit BMP280 library and displaying readings on the Serial Monitor. This session is ideal for beginners in electronics and programming who want to understand sensor interfacing and data acquisition on the Arduino platform.

Required Components

In this project, we need the following components.

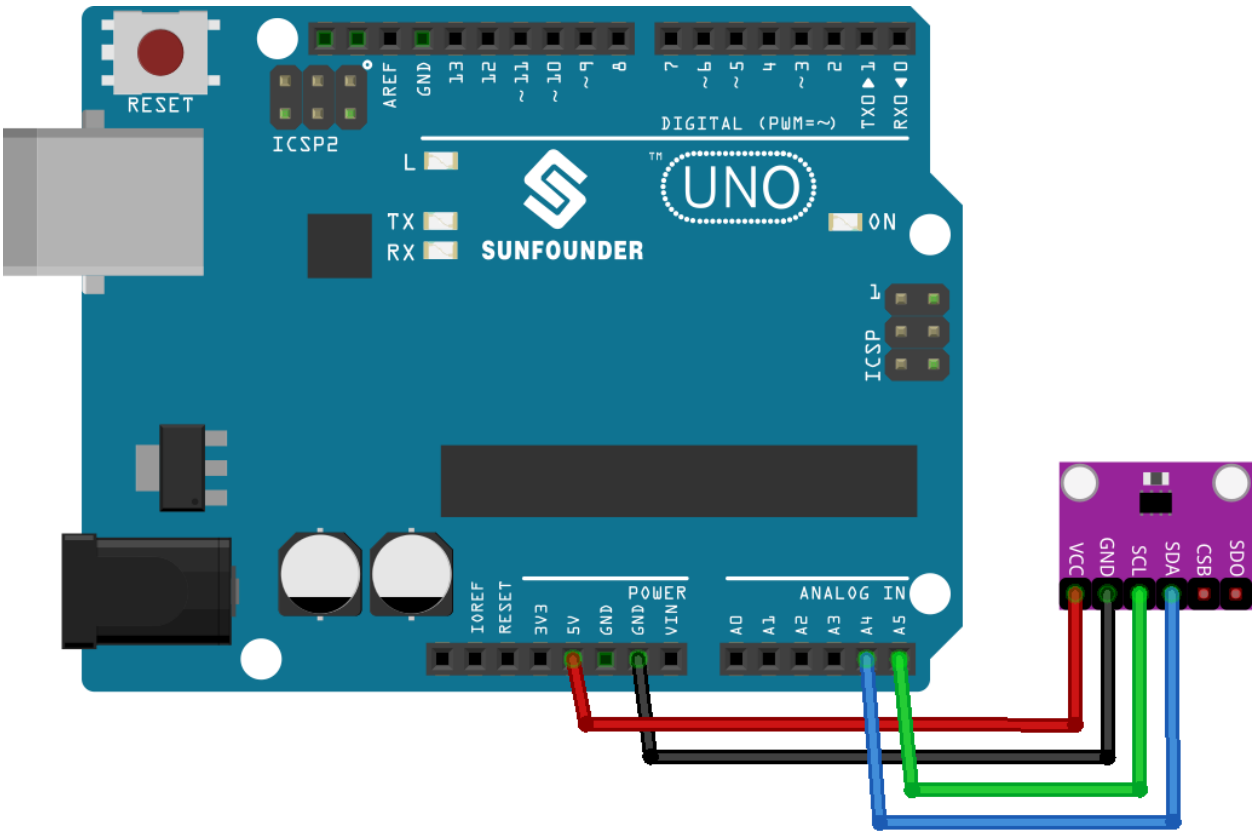
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Temperature, Humidity & Pressure Sensor (BMP280)	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"Adafruit BMP280"** and install it.

Code Analysis

1. Including Libraries and Initialization. Necessary libraries are included and the BMP280 sensor is initialized for communication using the I2C interface.

   Note: To install the library, use the Arduino Library Manager and search for **"Adafruit BMP280"** and install it.

   - Adafruit BMP280 Library: This library provides an easy-to-use interface for the BMP280 sensor, allowing the user to read temperature, pressure, and altitude.
   - Wire.h: Used for I2C communication.

```
#include <Wire.h>
#include <Adafruit_BMP280.h>
```

(continues on next page)

(continued from previous page)

```
#define BMP280_ADDRESS 0x76
Adafruit_BMP280 bmp;  // use I2C interface
```

2. The `setup()` function initializes the Serial communication, checks for the BMP280 sensor, and sets up the sensor with default settings.

```
void setup() {
  Serial.begin(9600);
  while (!Serial) delay(100);
  Serial.println(F("BMP280 test"));
  unsigned status;
  status = bmp.begin(BMP280_ADDRESS);
  // ... (rest of the setup code)
```

3. The `loop()` function reads data from the BMP280 sensor for temperature, pressure, and altitude. This data is printed to the Serial Monitor.

```
void loop() {
  // ... (read and print temperature, pressure, and altitude data)
  delay(2000);  // 2-second delay between readings.
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.22 Lesson 21: Time of Flight Micro-LIDAR Distance Sensor (VL53L0X)

In this lesson, you will learn how to use the VL53L0X Time of Flight Distance Sensor with an Arduino Uno. We'll cover the basics of connecting the sensor to measure distances in millimeters and displaying the readings on the serial monitor. This project provides hands-on experience with advanced sensors and their real-world applications, enhancing your Arduino skills.

Required Components

In this project, we need the following components.

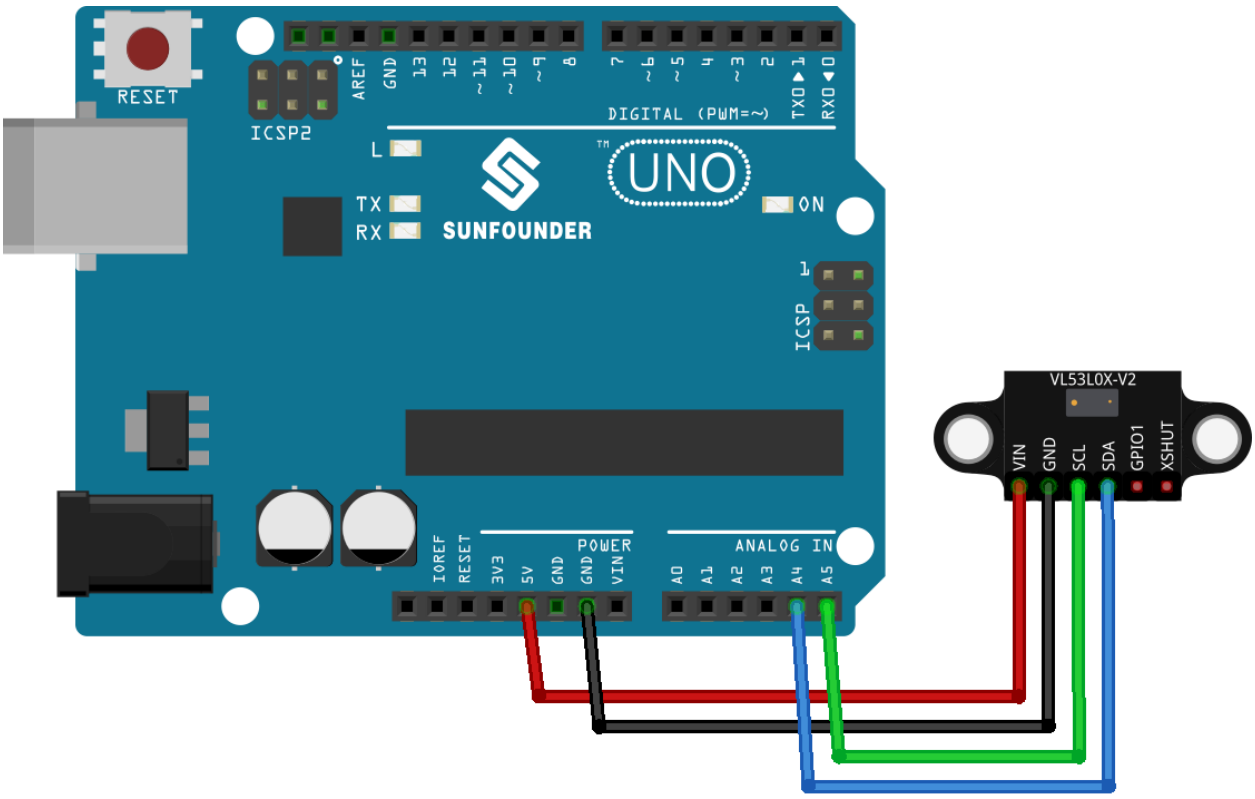
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Time of Flight Micro-LIDAR Distance Sensor (VL53L0X)	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"Adafruit_VL53L0X"** and install it.

Code Analysis

1. Including the necessary library and initializing the sensor object. We start by including the library for the VL53L0X sensor and creating an instance of the Adafruit_VL53L0X class.

   Note: To install the library, use the Arduino Library Manager and search for **"Adafruit_VL53L0X"** and install it.

```
#include <Adafruit_VL53L0X.h>
Adafruit_VL53L0X lox = Adafruit_VL53L0X();
```

2. Initialization in the `setup()` function. Here, we set up serial communication and initialize the distance sensor. If the sensor can't be initialized, the program halts.

```
void setup() {
  Serial.begin(115200);
  while (!Serial) {
    delay(1);
  }
  Serial.println("Adafruit VL53L0X test");
  if (!lox.begin()) {
    Serial.println(F("Failed to boot VL53L0X"));
    while (1)
      ;
  }
  Serial.println(F("VL53L0X API Simple Ranging example\n\n"));
}
```

3. Capturing and displaying the measurements in the `loop()` function. Continuously, the Arduino captures a distance measurement using the `rangingTest()` method. If the measurement is valid, it's printed to the serial monitor.

```
void loop() {
  VL53L0X_RangingMeasurementData_t measure;
  Serial.print("Reading a measurement... ");
  lox.rangingTest(&measure, false);
  if (measure.RangeStatus != 4) {
    Serial.print("Distance (mm): ");
    Serial.println(measure.RangeMilliMeter);
  } else {
    Serial.println(" out of range ");
  }
  delay(100);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.23 Lesson 22: Touch Sensor Module

In this lesson, you will learn how to integrate a touch sensor with an Arduino Uno. We'll focus on reading inputs from the touch sensor connected to the Arduino and how these inputs affect the program's flow. You'll discover how to use conditional statements to detect touch events and respond with appropriate actions and messages. This project is excellent for beginners, providing a clear understanding of working with digital inputs and basic Arduino programming concepts.

Required Components

In this project, we need the following components.

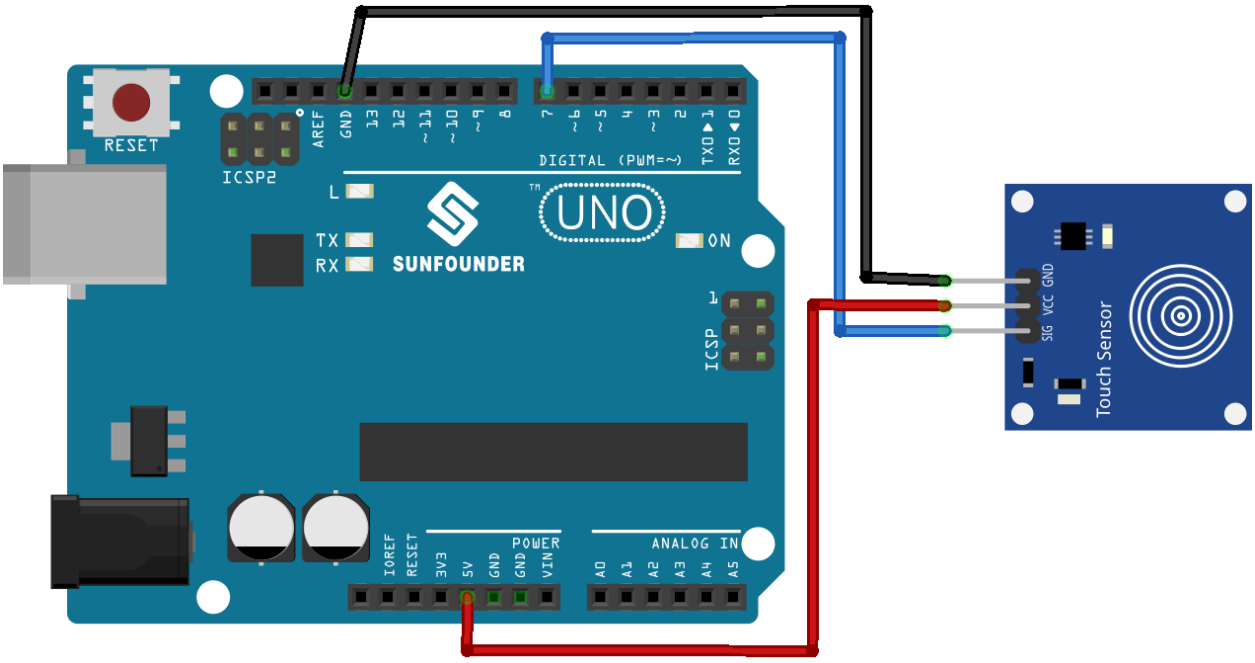
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Touch Sensor Module	

Wiring



Code

Code Analysis

1. Setting up the necessary variables. We start by defining the pin number where the touch sensor is connected.

```
const int sensorPin = 7;
```

2. Initialization in the `setup()` function. Here, we specify that the sensor pin will be used for input, the built-in LED will be used for output, and we start the serial communication to allow messages to be sent to the serial monitor.

```
void setup() {
  pinMode(sensorPin, INPUT);
  pinMode(LED_BUILTIN, OUTPUT);
  Serial.begin(9600);
}
```

3. Continuously, the Arduino checks if the touch sensor is activated. If touched, it turns on the LED and sends a "Touch detected!" message. If not touched, it turns off the LED and sends a "No touch detected..." message. A delay is introduced to prevent the sensor from being read too quickly.

```
void loop() {
  if (digitalRead(sensorPin) == 1) {
    digitalWrite(LED_BUILTIN, HIGH);
    Serial.println("Touch detected!");
  } else {
    digitalWrite(LED_BUILTIN, LOW);
    Serial.println("No touch detected...");
```

(continues on next page)

(continued from previous page)

```
  }
  delay(100);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.24 Lesson 23: Ultrasonic Sensor Module (HC-SR04)

In this lesson, you will learn how to use an ultrasonic sensor with Arduino to measure distances. We'll cover connecting the HC-SR04 sensor to the Arduino Uno R4 board and using it to calculate and display distance measurements in centimeters. This project is ideal for beginners, providing hands-on experience with Arduino's serial communication and sensor data processing. You'll gain valuable insights into working with digital signals and understanding the basics of ultrasonic sensing technology.

Required Components

In this project, we need the following components.

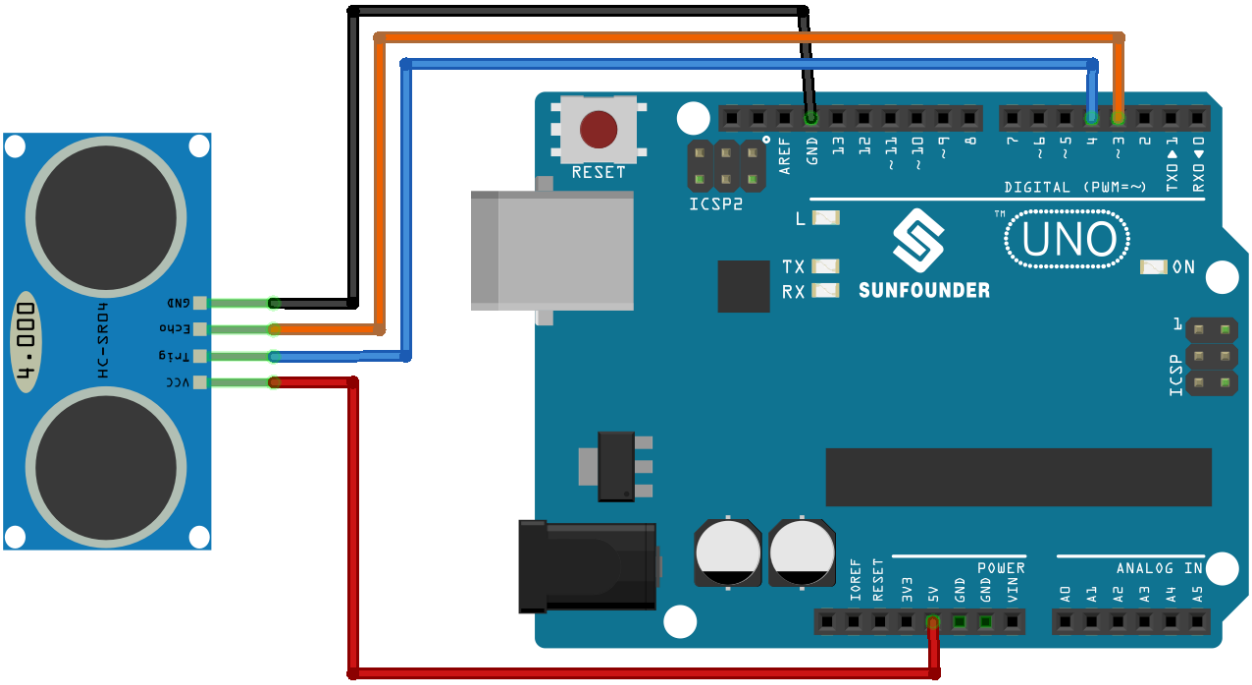
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Ultrasonic Sensor Module (HC-SR04)	

Wiring



Code

Code Analysis

1. Pin declaration:

   Start by defining the pins for the ultrasonic sensor. `echoPin` and `trigPin` are declared as integers and their values are set to match the physical connection on the Arduino board.

   ```
   const int echoPin = 3;
   const int trigPin = 4;
   ```

2. `setup()` function:

   The `setup()` function initializes the serial communication, sets the pin modes, and prints a message to indicate the ultrasonic sensor is ready.

   ```
   void setup() {
     Serial.begin(9600);
     pinMode(echoPin, INPUT);
     pinMode(trigPin, OUTPUT);
     Serial.println("Ultrasonic sensor:");
   }
   ```

3. `loop()` function:

   The `loop()` function reads the distance from the sensor and prints it to the serial monitor, then delays for 400 milliseconds before repeating.

```
void loop() {
  float distance = readDistance();
  Serial.print(distance);
  Serial.println(" cm");
  delay(400);
}
```

4. `readDistance()` function :

   The `readDistance()` function triggers the ultrasonic sensor and calculates the distance based on the time it takes for the signal to bounce back.

   For more details, please refer to the working *principle* of the ultrasonic sensor module.

```
float readDistance() {
  digitalWrite(trigPin, LOW);   // Set trig pin to low to ensure a clean pulse
  delayMicroseconds(2);         // Delay for 2 microseconds
  digitalWrite(trigPin, HIGH);  // Send a 10 microsecond pulse by setting trig pin
↪to high
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);  // Set trig pin back to low
  float distance = pulseIn(echoPin, HIGH) / 58.00;  // Formula: (340m/s * 1us) / 2
  return distance;
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.25 Lesson 24: Vibration Sensor Module (SW-420)

In this lesson, you will learn how to detect vibrations using a vibration sensor with an Arduino Uno. We'll explore how the sensor signals the presence of vibrations to the Arduino, triggering it to display a message. This project is perfect for beginners to understand digital input processing and serial communication in Arduino. You'll gain hands-on experience in reading sensor data and implementing conditional logic in your sketches.

Required Components

In this project, we need the following components.

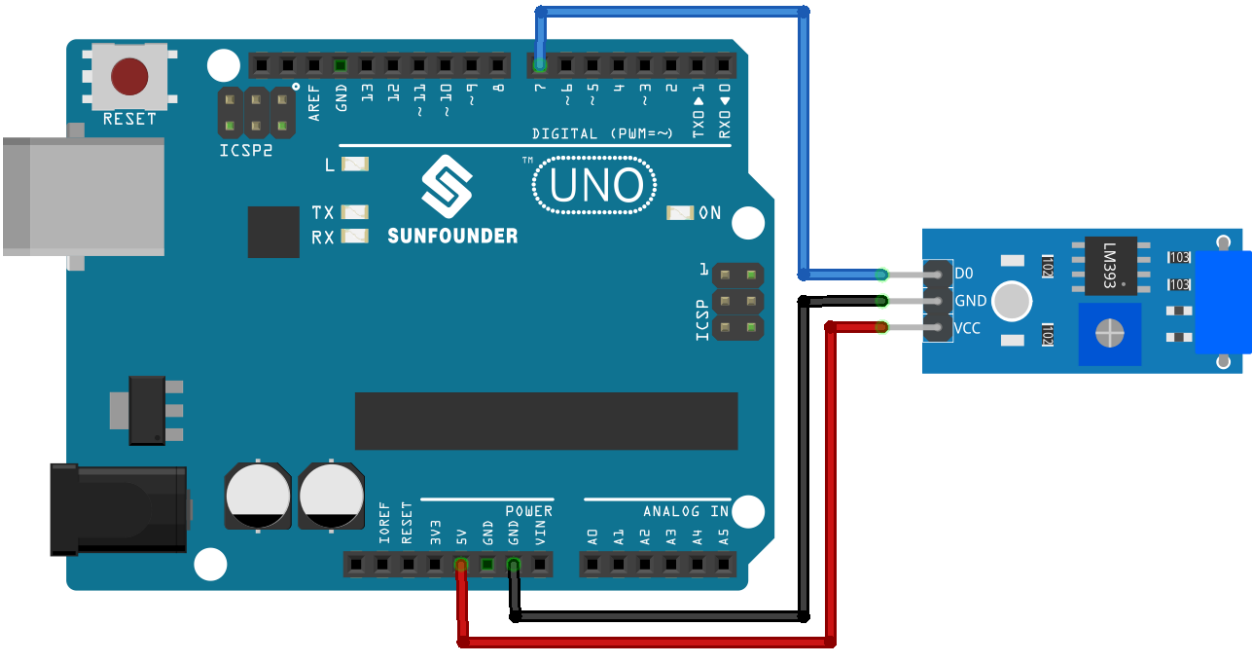
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Vibration Sensor Module (SW-420)	

Wiring



Code

Code Analysis

1. The first line of code is a constant integer declaration for the vibration sensor pin. We use digital pin 7 to read the output from the vibration sensor.

```
const int sensorPin = 7;
```

2. In the `setup()` function, we initialize the serial communication at a baud rate of 9600 to print readings from the vibration sensor to the serial monitor. We also set the vibration sensor pin as an input.

```
void setup() {
  Serial.begin(9600);         // Start serial communication at 9600 baud rate
  pinMode(sensorPin, INPUT);  // Set the sensorPin as an input pin
}
```

3. The `loop()` function is where we continuously check for any vibrations detected by the sensor. If the sensor detects a vibration, it prints "Detected vibration…" to the serial monitor. If no vibration is detected, it prints "…". The loop repeats every 100 milliseconds.

```
void loop() {
  if (digitalRead(sensorPin)) {               // Check if there is any vibration␣
↪detected by the sensor
    Serial.println("Detected vibration...");  // Print "Detected vibration..." if␣
↪vibration detected
  }
  else {
    Serial.println("...");  // Print "..." otherwise
  }
  // Add a delay to avoid flooding the serial monitor
  delay(100);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.26 Lesson 25: Water Level Sensor Module

In this lesson, you will learn how to measure water levels using Arduino. We'll look at how a water level sensor can produce different voltage levels based on the water height and how the Arduino reads these voltage levels. This project is ideal for beginners as it provides practical experience with analog sensors and introduces basic concepts of processing sensor data on the Arduino platform.

Required Components

In this project, we need the following components.

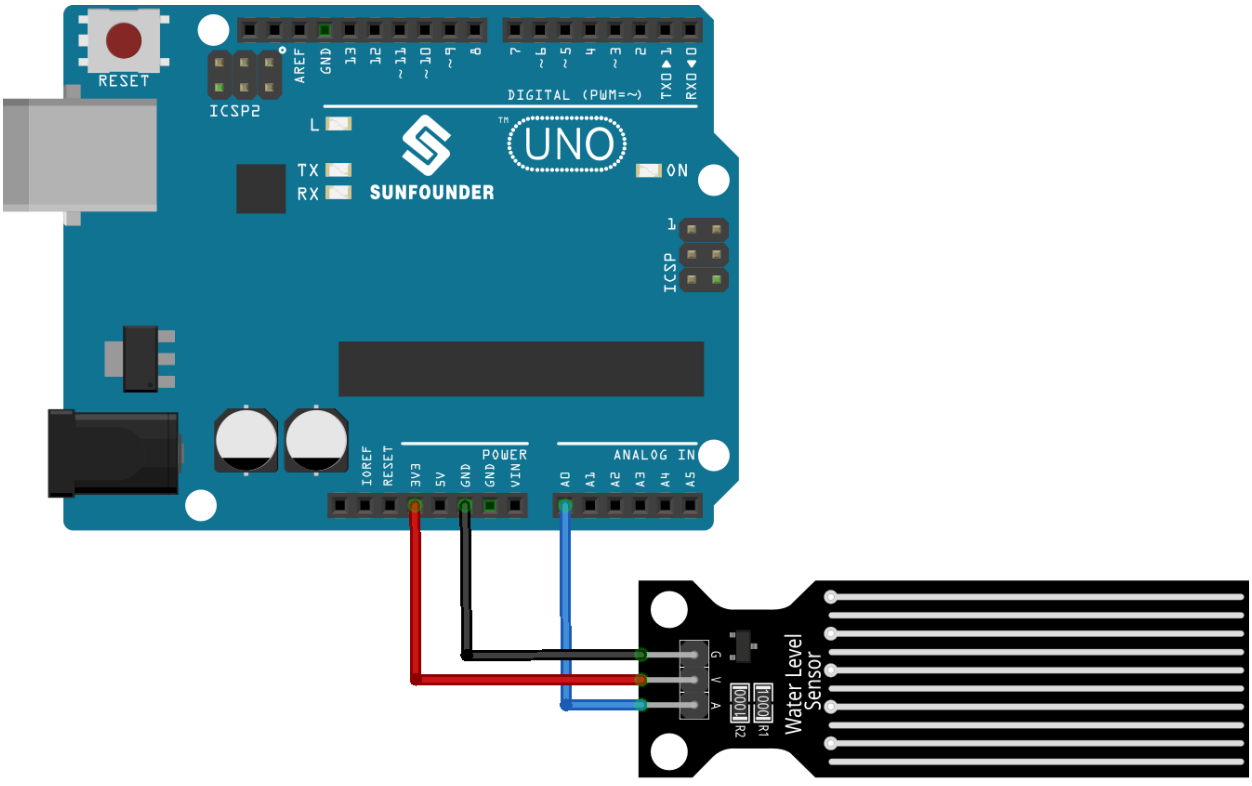
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Water Level Sensor Module	-

Wiring



Code

Code Analysis

1. **Initializing the Sensor Pin**:

   Before using the water level sensor, it's pin number is defined using a constant variable. This makes the code more readable and easier to modify.

```
const int sensorPin = A0;
```

2. **Setting Up Serial Communication**:

   In the `setup()` function, the baud rate for serial communication is set. This is crucial for the Arduino to communicate with the computer's serial monitor.

```
void setup() {
  Serial.begin(9600);  // Start serial communication at 9600 baud rate
}
```

3. **Reading Sensor Data and Outputting to Serial Monitor**:

   The `loop()` function continuously reads the sensor's analog value using `analogRead()` and outputs it to the serial monitor using `Serial.println()`. The `delay(100)` function makes the Arduino wait for 100 milliseconds before repeating the loop, controlling the rate of data reading and transmission.

```
void loop() {
  Serial.println(analogRead(sensorPin));  // Read the analog value of the sensor␣
↪and print it to the serial monitor
  delay(100);                             // Wait for 100 milliseconds
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.27 Lesson 26: I2C LCD 1602

In this lesson, you will learn how to set up and display messages on a 16x2 Liquid Crystal Display (LCD) with an I2C interface using Arduino. We'll cover the basics of using the LiquidCrystal I2C library to initialize the LCD, display text, and control the backlight. You'll see how to print "Hello world!" and "LCD Tutorial" on the display, providing a hands-on introduction to interfacing LCDs with Arduino. This tutorial is perfect for beginners as it offers a practical lesson in controlling electronic displays.

Required Components

In this project, we need the following components.

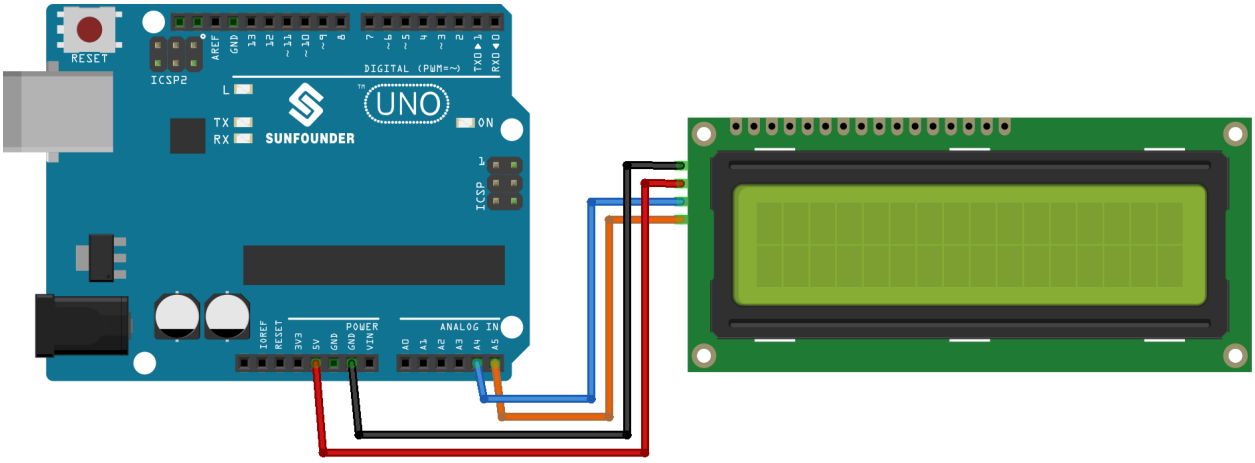
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
I2C LCD 1602	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"LiquidCrystal I2C"** and install it.

Code Analysis

1. Library Inclusion and LCD Initialization: The LiquidCrystal I2C library is included to provide functions and methods for LCD interfacing. Following that, an LCD object is created using the LiquidCrystal_I2C class, specifying the I2C address, number of columns, and number of rows.

   Note: To install the library, use the Arduino Library Manager and search for **"LiquidCrystal I2C"** and install it.

   ```
   #include <LiquidCrystal_I2C.h>
   LiquidCrystal_I2C lcd(0x27, 16, 2);
   ```

2. Setup Function: The `setup()` function is executed once when the Arduino starts. In this function, the LCD is initialized, cleared, and the backlight is turned on. Then, two messages are displayed on the LCD.

   ```
   void setup() {
     lcd.init();       // initialize the LCD
     lcd.clear();      // clear the LCD display
     lcd.backlight();  // Make sure backlight is on

     // Print a message on both lines of the LCD.
     lcd.setCursor(2, 0);  //Set cursor to character 2 on line 0
     lcd.print("Hello world!");

     lcd.setCursor(2, 1);  //Move cursor to character 2 on line 1
     lcd.print("LCD Tutorial");
   }
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.28 Lesson 27: OLED Display Module (SSD1306)

In this lesson, you will learn how to program an Arduino Uno board to control an OLED display (SSD1306). We'll cover using the Adafruit SSD1306 and GFX libraries to display text, numbers, and create scroll animations on the screen. This project is ideal for those seeking to enhance their knowledge of displaying graphics and text on small screens using the Arduino environment.

Required Components

In this project, we need the following components.

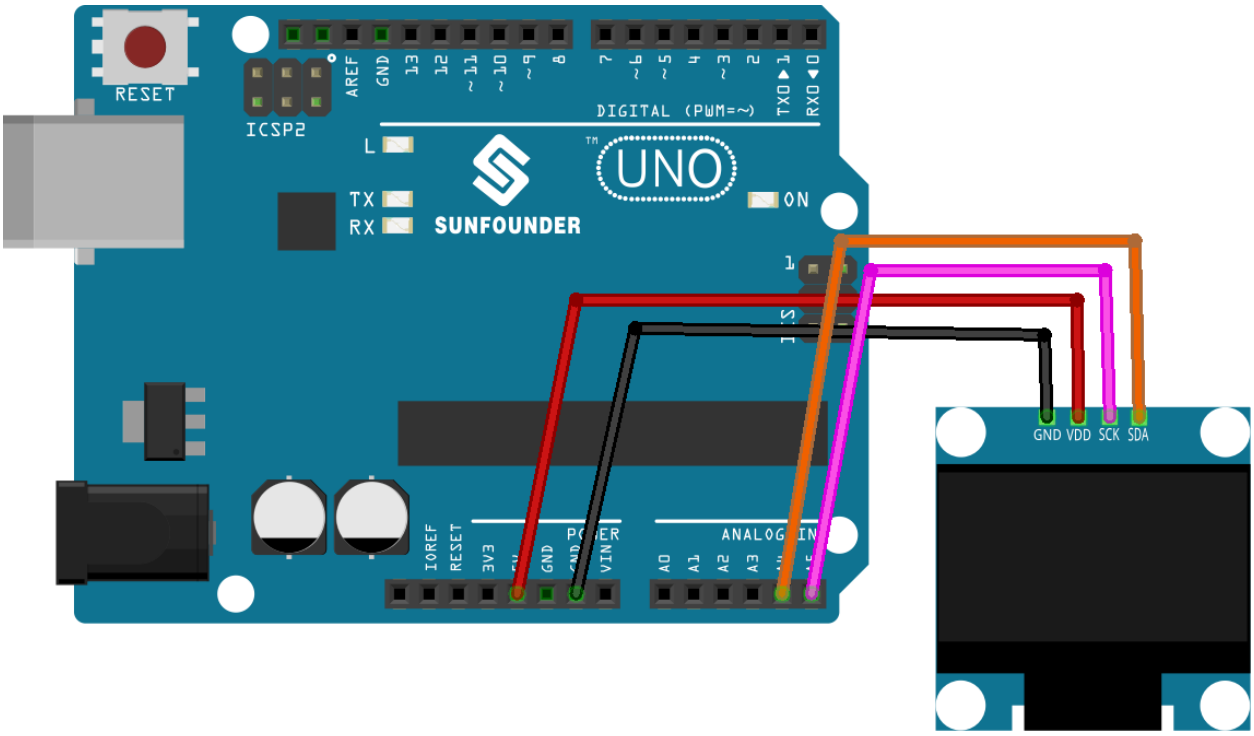
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
OLED Display Module (SSD1306)	-

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"Adafruit SSD1306"** and **"Adafruit GFX"** and install it.

Code Analysis

1. **Library Inclusion and Initial Definitions**: The necessary libraries for interfacing with the OLED are included. Following that, definitions regarding the OLED's dimensions and I2C address are provided.

    - **Adafruit SSD1306**: This library is designed to help with the interfacing of the SSD1306 OLED display. It provides methods to initialize the display, control its settings, and display content.
    - **Adafruit GFX Library**: This is a core graphics library for displaying text, producing colors, drawing shapes, etc., on various screens including OLEDs.

    Note: To install the library, use the Arduino Library Manager and search for **"Adafruit SSD1306"** and **"Adafruit GFX"** and install it.

    ```
    #include <SPI.h>
    #include <Wire.h>
    #include <Adafruit_GFX.h>
    #include <Adafruit_SSD1306.h>

    #define SCREEN_WIDTH 128  // OLED display width, in pixels
    #define SCREEN_HEIGHT 64  // OLED display height, in pixels

    #define OLED_RESET -1
    #define SCREEN_ADDRESS 0x3C
    ```

2. **Bitmap Data**: Bitmap data for displaying a custom icon on the OLED screen. This data represents an image in a format that the OLED can interpret.

    You can use this online tool called that can turn your image into an array.

    The `PROGMEM` keyword denotes that the array is stored in the program memory of the Arduino microcontroller. Storing data in program memory(PROGMEM) instead of RAM can be helpful for large amounts of data, which would otherwise take up too much space in RAM.

    ```
    static const unsigned char PROGMEM sunfounderIcon[] = {...};
    ```

3. **Setup Function (Initialization and Display)**: The `setup()` function initializes the OLED and displays a series of patterns, texts, and animations.

    ```
    void setup() {
       ...  // Serial initialization and OLED object initialization
       ...  // Displaying various text, numbers, and animations
    }
    ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.29 Lesson 28: RGB LED Module

In this lesson, you will learn how to control an RGB LED with Arduino. We'll cover setting up the LED and then delve into displaying primary colors and creating a vibrant rainbow spectrum. This hands-on project is ideal for beginners, providing practical experience with output operations and color mixing in the Arduino environment.

Required Components

In this project, we need the following components.

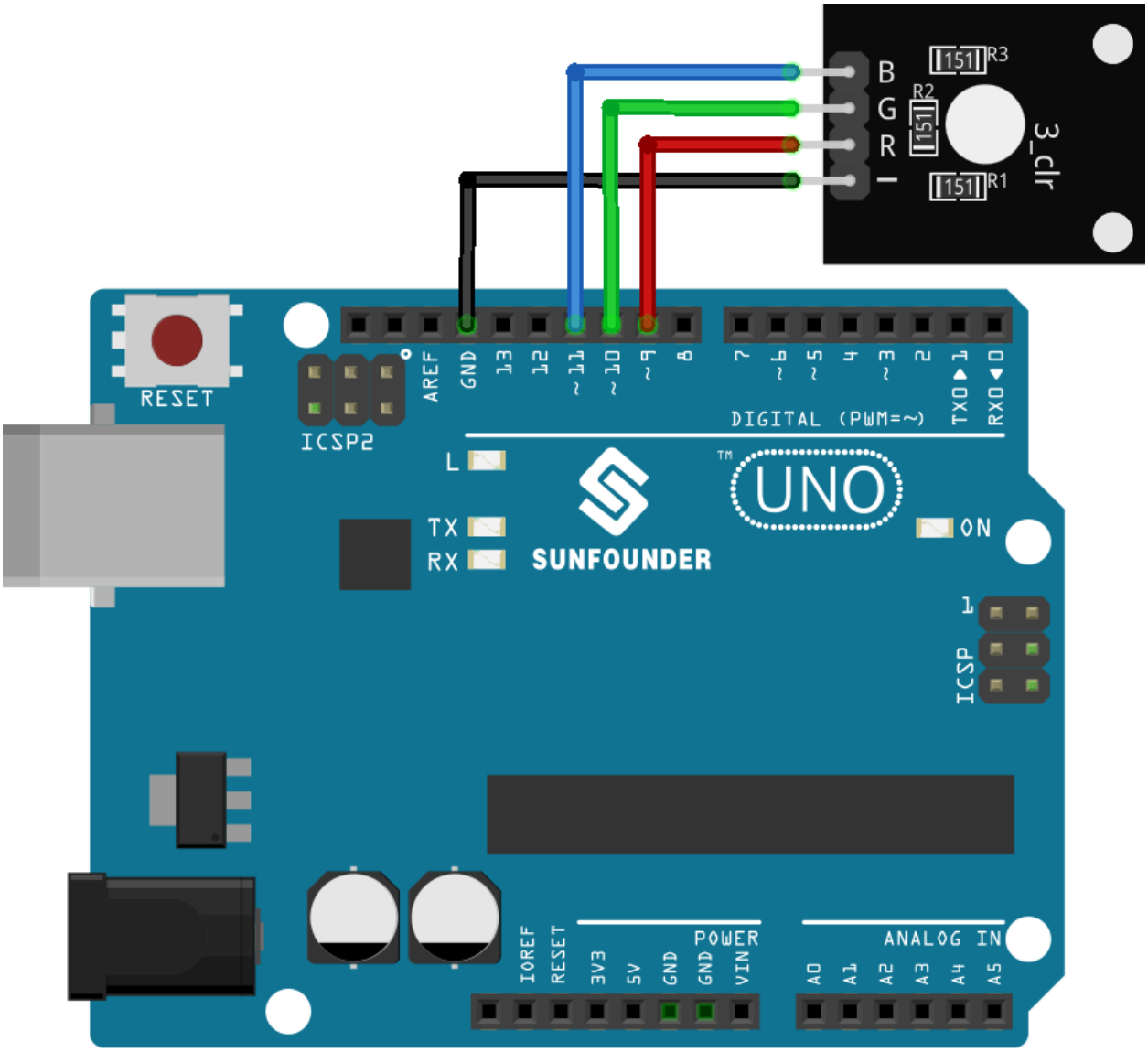
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
RGB LED Module	-

Wiring



Code

Code Analysis

1. The first segment of the code declares and initializes the pins to which each color channel of the RGB LED module is connected.

```
const int rledPin = 9;  // pin connected to the red color channel
const int gledPin = 10;   // pin connected to the green color channel
const int bledPin = 11;  // pin connected to the blue color channel
```

2. The `setup()` function initializes these pins as OUTPUT. This means we are sending signals OUT from these pins to the RGB LED module.

```
void setup() {
  pinMode(rledPin, OUTPUT);
  pinMode(gledPin, OUTPUT);
  pinMode(bledPin, OUTPUT);
}
```

3. In the `loop()` function, the `setColor()` function is called with different parameters to display different colors. The `delay()` function is used after setting each color to pause for 1000 milliseconds (or 1 second) before moving on to the next color.

```
void loop() {
  setColor(255, 0, 0);  // Set RGB LED color to red
  delay(1000);
  setColor(0, 255, 0);  // Set RGB LED color to green
  delay(1000);
  // The rest of the color sequence...
}
```

4. The `setColor()` function uses the `analogWrite()` function to adjust the brightness of each color channel on the RGB LED module. The `analogWrite()` function employs Pulse Width Modulation (PWM) to simulate varying voltage outputs. By controlling the PWM duty cycle (the percentage of time a signal is HIGH within a fixed period), the brightness of each color channel can be controlled, allowing the mixing of various colors.

```
void setColor(int R, int G, int B) {
  analogWrite(rledPin, R);  // Use PWM to control the brightness of the red color␣
↪channel
  analogWrite(gledPin, G);  // Use PWM to control the brightness of the green color␣
↪channel
  analogWrite(bledPin, B);  // Use PWM to control the brightness of the blue color␣
↪channel
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.30 Lesson 29: Traffic Light Module

In this lesson, you will learn how to use Arduino to control a mini LED traffic light. We'll cover programming the Arduino Uno to cycle through green, yellow, and red lights, simulating a real traffic signal. This project is ideal for beginners as it provides practical experience in coding light sequences and timing controls on the Arduino platform.

Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

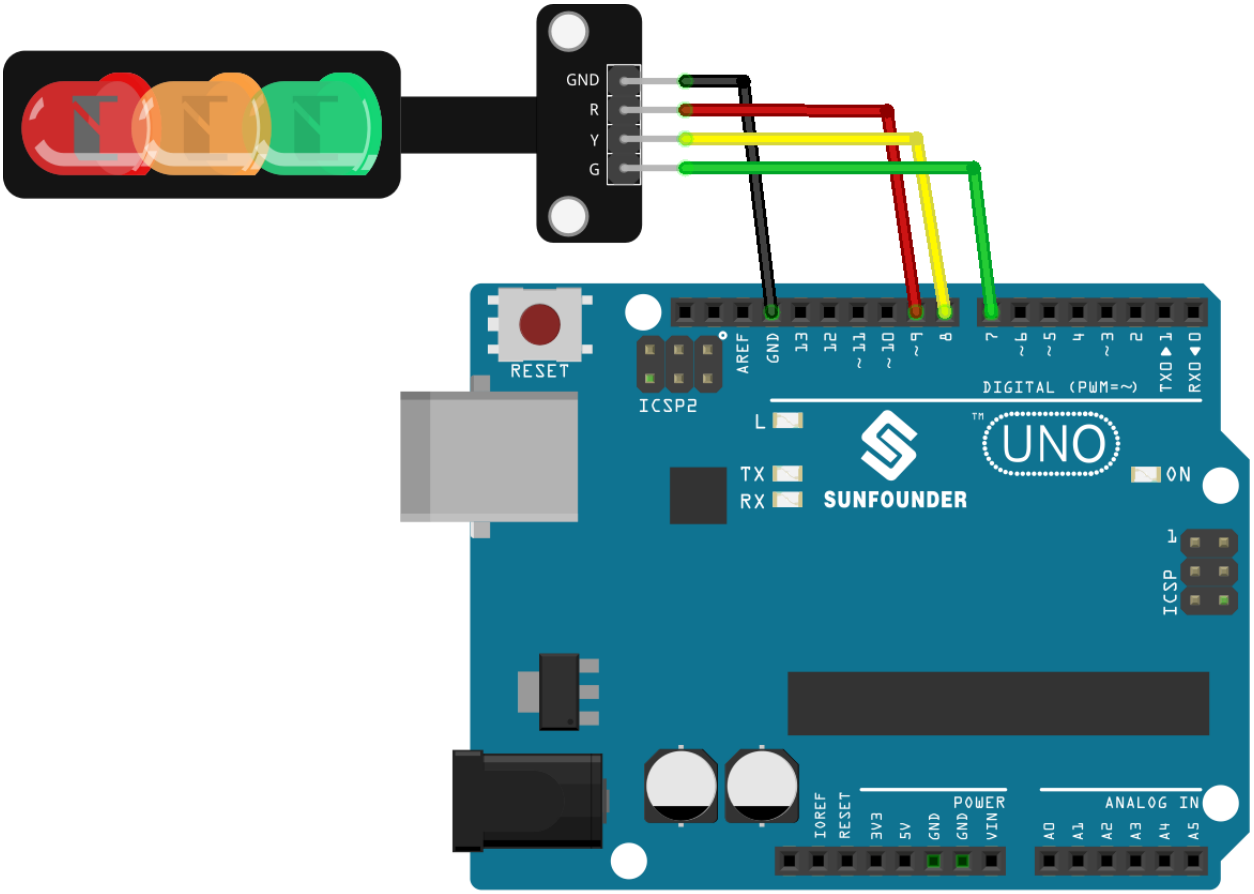
Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Traffic Light Module	

- Arduino UNO R3 or R4
- *Traffic Light Module*

Wiring



Code

Code Analysis

1. Before any operations, we define constants for the pins where LEDs are connected. This makes our code easier to read and modify.

```
const int rledPin = 9;  //red
const int yledPin = 8;  //yellow
const int gledPin = 7;  //green
```

2. Here, we specify the pin modes for our LED pins. They are all set to `OUTPUT` because we intend to send voltage to them.

```
void setup() {
  pinMode(rledPin, OUTPUT);
  pinMode(yledPin, OUTPUT);
  pinMode(gledPin, OUTPUT);
}
```

3. This is where our traffic light cycle logic is implemented. The sequence of operations is:

    - Turn the green LED on for 5 seconds.

- Blink the yellow LED three times (each blink lasts for 0.5 seconds).
- Turn the red LED on for 5 seconds.

```
void loop() {
  digitalWrite(gledPin, HIGH);
  delay(5000);
  digitalWrite(gledPin, LOW);

  digitalWrite(yledPin, HIGH);
  delay(500);
  digitalWrite(yledPin, LOW);
  delay(500);
  digitalWrite(yledPin, HIGH);
  delay(500);
  digitalWrite(yledPin, LOW);
  delay(500);
  digitalWrite(yledPin, HIGH);
  delay(500);
  digitalWrite(yledPin, LOW);
  delay(500);

  digitalWrite(rledPin, HIGH);
  delay(5000);
  digitalWrite(rledPin, LOW);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.31 Lesson 30: Relay Module

In this lesson, you will learn how to use a relay and an Arduino Uno to control a traffic light module. We'll demonstrate how to turn the red light of the traffic module on and off using the relay. This project is ideal for beginners in Arduino, providing hands-on experience in controlling external modules and gaining a fundamental understanding of relay operations.

Required Components

In this project, we need the following components.

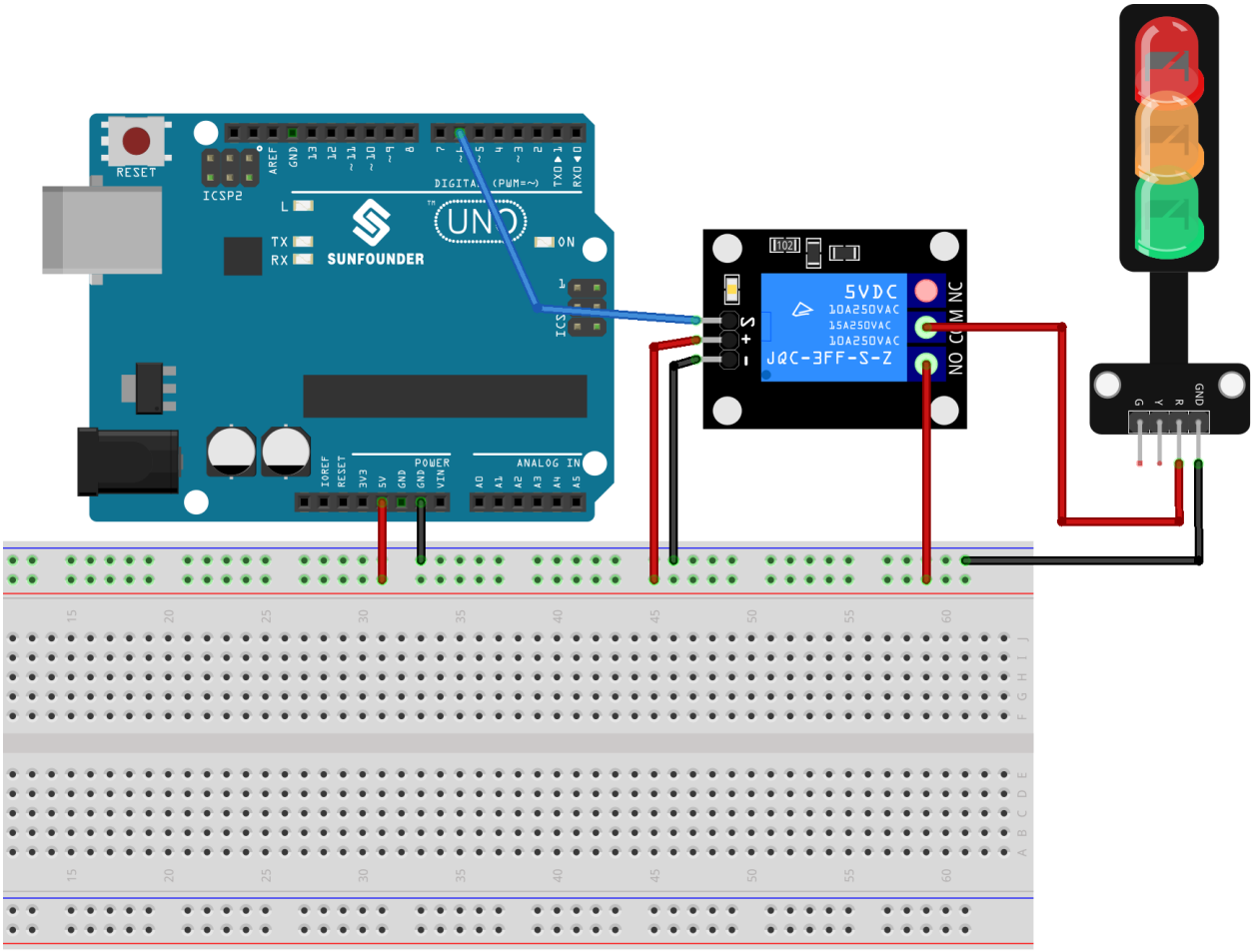
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Breadboard	
5V Relay Module	-
Traffic Light Module	

Wiring



Code

Code Analysis

1. Setting up the relay pin:
    - The relay module is connected to pin 6 of the Arduino. This pin is defined as `relayPin` for ease of reference in the code.

```
const int relayPin = 6;
```

2. Configuring the relay pin as an output:
    - In the `setup()` function, the relay pin is set as an OUTPUT using the `pinMode()` function. This means the Arduino will send signals (either HIGH or LOW) to this pin.

```
void setup() {
  pinMode(relayPin, OUTPUT);
}
```

3. Toggling the relay ON and OFF:

- In the `loop()` function, the relay is first set to the OFF state using `digitalWrite(relayPin, LOW)`. It remains in this state for 3 seconds (`delay(3000)`).
- Then, the relay is set to the ON state using `digitalWrite(relayPin, HIGH)`. Again, it remains in this state for 3 seconds.
- This cycle repeats indefinitely.

```
void loop() {
  digitalWrite(relayPin, LOW);
  delay(3000);

  digitalWrite(relayPin, HIGH);
  delay(3000);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.32 Lesson 31: Centrifugal Pump

In this lesson, you'll learn how to control a centrifugal pump with an Arduino Uno R3 or R4 and an L9110 motor control board. You'll discover how to set up and program the Arduino to start the pump in one direction, run it for a specific duration, and then stop it. This hands-on experience is ideal for beginners and offers fundamental insight into managing motor operations and understanding output controls in Arduino projects.

Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

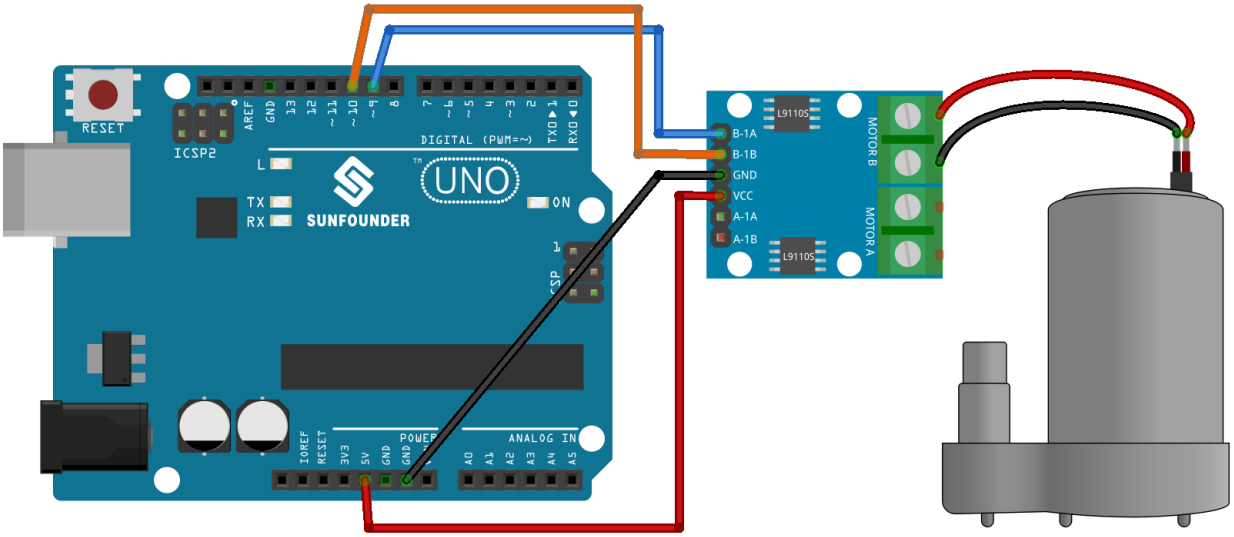
Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Centrifugal Pump	-
L9110 Motor Driver Module	-

- Arduino UNO R3 or R4
- *Centrifugal Pump*
- *L9110 Motor Driver Module*

Wiring



Code

Code Analysis

1. Two pins are defined for controlling the motor, specifically `motorB_1A` and `motorB_2A`. These pins will connect to the L9110 motor control board to control the direction and speed of the motor.

```
const int motorB_1A = 9;
const int motorB_2A = 10;
```

2. Configuring the pins and controlling the motor:
    - The `setup()` function initializes the pins as `OUTPUT` which means they can send signals to the motor control board.
    - The `analogWrite()` function is used to set the motor speed. Here, setting one pin to `HIGH` and the other to `LOW` makes the pump spin in one direction. After a 5-second delay, both pins are set to 0, turning off the motor.

```
void setup() {
   pinMode(motorB_1A, OUTPUT);  // set pump pin 1 as output
   pinMode(motorB_2A, OUTPUT);  // set pump pin 2 as output
   analogWrite(motorB_1A, HIGH);
   analogWrite(motorB_2A, LOW);
   delay(5000);// wait for 5 seconds
   analogWrite(motorB_1A, 0);  // turn off the pump
   analogWrite(motorB_2A, 0);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.33 Lesson 32: Passive Buzzer Module

In this lesson, you will learn how to play a melody on a passive buzzer module using Arduino. We'll cover programming the Arduino to control the buzzer and create various note durations. This project is ideal for beginners as it provides hands-on experience in producing sound and understanding musical notes within electronic components. You'll also gain practical insight into using the Arduino Uno board and the passive buzzer module.

Required Components

In this project, we need the following components.

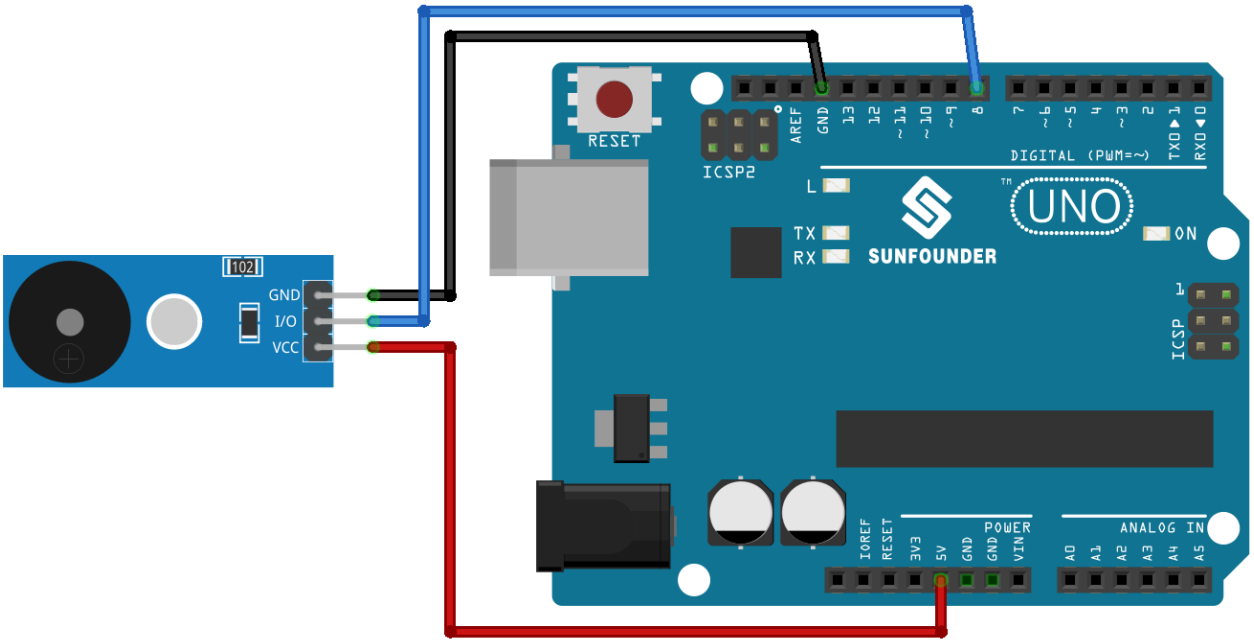
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Passive Buzzer Module	

Wiring



Code

Code Analysis

1. Including the pitches library: This library provides the frequency values for various musical notes, allowing you to use musical notation in your code.

```
#include "pitches.h"
```

2. Defining constants and arrays:
   - `buzzerPin` is the digital pin on the Arduino where the buzzer is connected.
   - `melody[]` is an array that stores the sequence of notes to be played.
   - `noteDurations[]` is an array that stores the duration of each note in the melody.

```
const int buzzerPin = 8;
int melody[] = {
  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
};
int noteDurations[] = {
  4, 8, 8, 4, 4, 4, 4, 4
};
```

3. Playing the melody:
   - The `for` loop iterates over each note in the melody.
   - The `tone()` function plays a note on the buzzer for a specific duration.
   - A delay is added between notes to distinguish them.

- The `noTone()` function stops the sound.

```
void setup() {
  for (int thisNote = 0; thisNote < 8; thisNote++) {
    int noteDuration = 1000 / noteDurations[thisNote];
    tone(buzzerPin, melody[thisNote], noteDuration);
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    noTone(buzzerPin);
  }
}
```

4. Empty loop function: Since the melody is played only once in the setup, there's no code in the loop function.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.34 Lesson 33: Servo Motor (SG90)

In this lesson, you will learn how to use Arduino to control a servo motor and make it rotate from 0 to 180 degrees and back. We will cover the usage of the Servo library, defining and using variables for servo control, as well as implementing a for loop for gradual movement. This project is ideal for beginners as it provides hands-on experience with motor control and basic programming principles in Arduino.

Required Components

In this project, we need the following components.

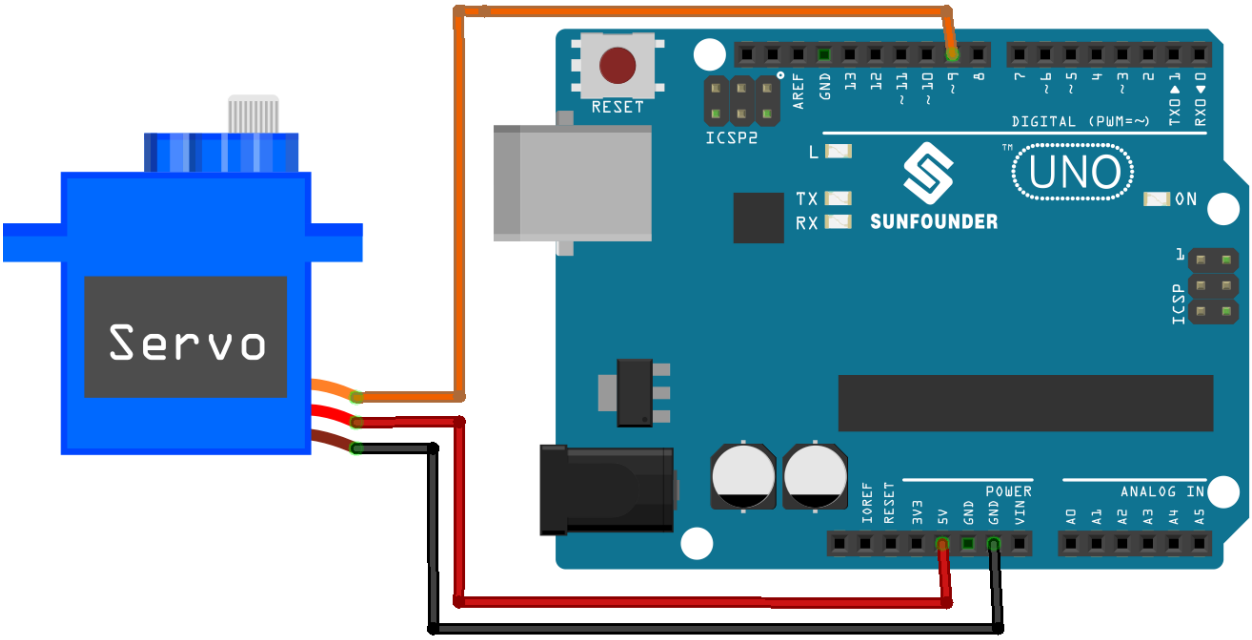
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Servo Motor (SG90)	

Wiring



Code

Code Analysis

1. Here, the `Servo` library is included which allows for easy control of the servo motor. The pin connected to the servo and the initial angle of the servo are also defined.

```
#include <Servo.h>
const int servoPin = 9;  // Define the servo pin
int angle = 0;           // Initialize the angle variable to 0 degrees
Servo servo;             // Create a servo object
```

2. The `setup()` function runs once when the Arduino starts. The servo is attached to the defined pin using the `attach()` function.

```
void setup() {
  servo.attach(servoPin);
}
```

3. The main loop has two `for` loops. The first loop increases the angle from 0 to 180 degrees, and the second loop decreases the angle from 180 to 0 degrees. The `servo.write(angle)` command sets the servo to the specified

angle. The `delay(15)` causes the servo to wait for 15 milliseconds before moving to the next angle, controlling the speed of the scanning movement.

```
void loop() {
  // scan from 0 to 180 degrees
  for (angle = 0; angle < 180; angle++) {
    servo.write(angle);
    delay(15);
  }
  // now scan back from 180 to 0 degrees
  for (angle = 180; angle > 0; angle--) {
    servo.write(angle);
    delay(15);
  }
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.35 Lesson 34: TT Motor

In this lesson, you will learn how to control a motor using an Arduino Uno R3 or R4 and an L9110 motor control board. We'll cover defining motor pins and setting their speed through programming. This tutorial will walk you through the process of connecting and controlling a motor, demonstrating the basic principles of motor operation and control in Arduino projects. Geared towards beginners, this lesson provides a hands-on approach to understanding output operations on the Arduino platform.

Required Components

In this project, we need the following components.

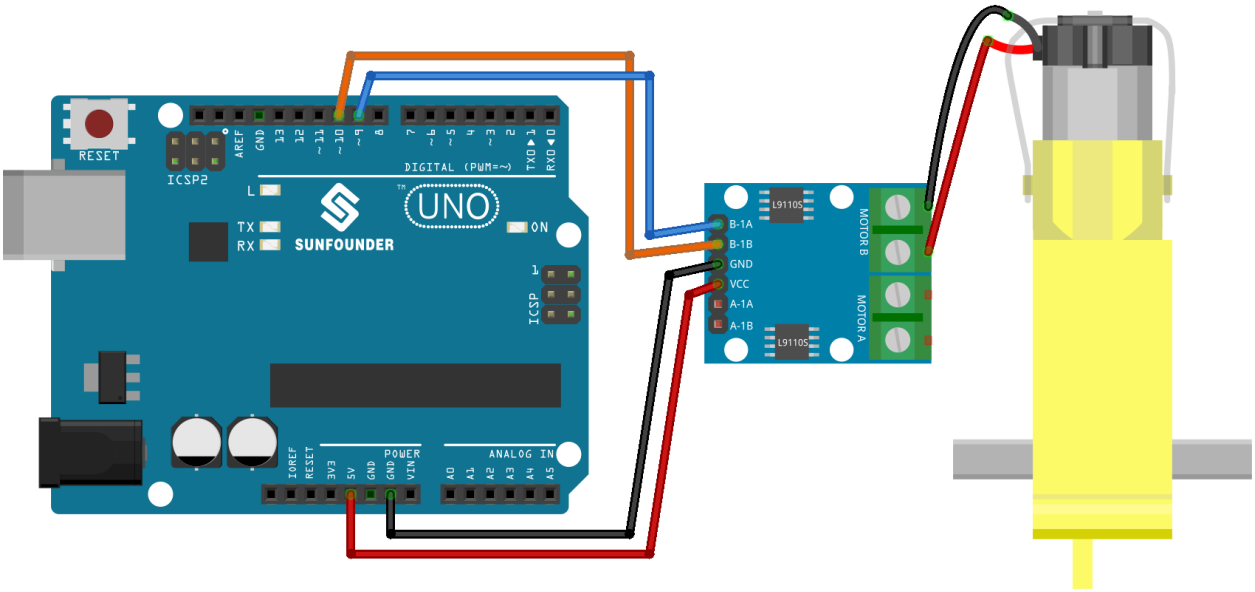
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
TT Motor	-
L9110 Motor Driver Module	-

Wiring



Code

Code Analysis

1. The first part of the code defines the motor control pins. These are connected to the L9110 motor control board.

```
// Define the motor pins
const int motorB_1A = 9;
const int motorB_2A = 10;
```

2. The `setup()` function initializes the motor control pins as output using the `pinMode()` function. Then it uses `analogWrite()` to set the speed of the motor. The value passed to `analogWrite()` can range from 0 (off) to 255 (full speed). A `delay()` function is then used to pause the code for 5000 milliseconds (or 5 seconds), after which the motor speed is set to 0 (off).

```
void setup() {
  pinMode(motorB_1A, OUTPUT);  // set motor pin 1 as output
  pinMode(motorB_2A, OUTPUT);  // set motor pin 2 as output

  analogWrite(motorB_1A, 255);  // set motor speed (0-255)
  analogWrite(motorB_2A, 0);
```

(continues on next page)

(continued from previous page)

```
  delay(5000);

  analogWrite(motorB_1A, 0);
  analogWrite(motorB_2A, 0);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

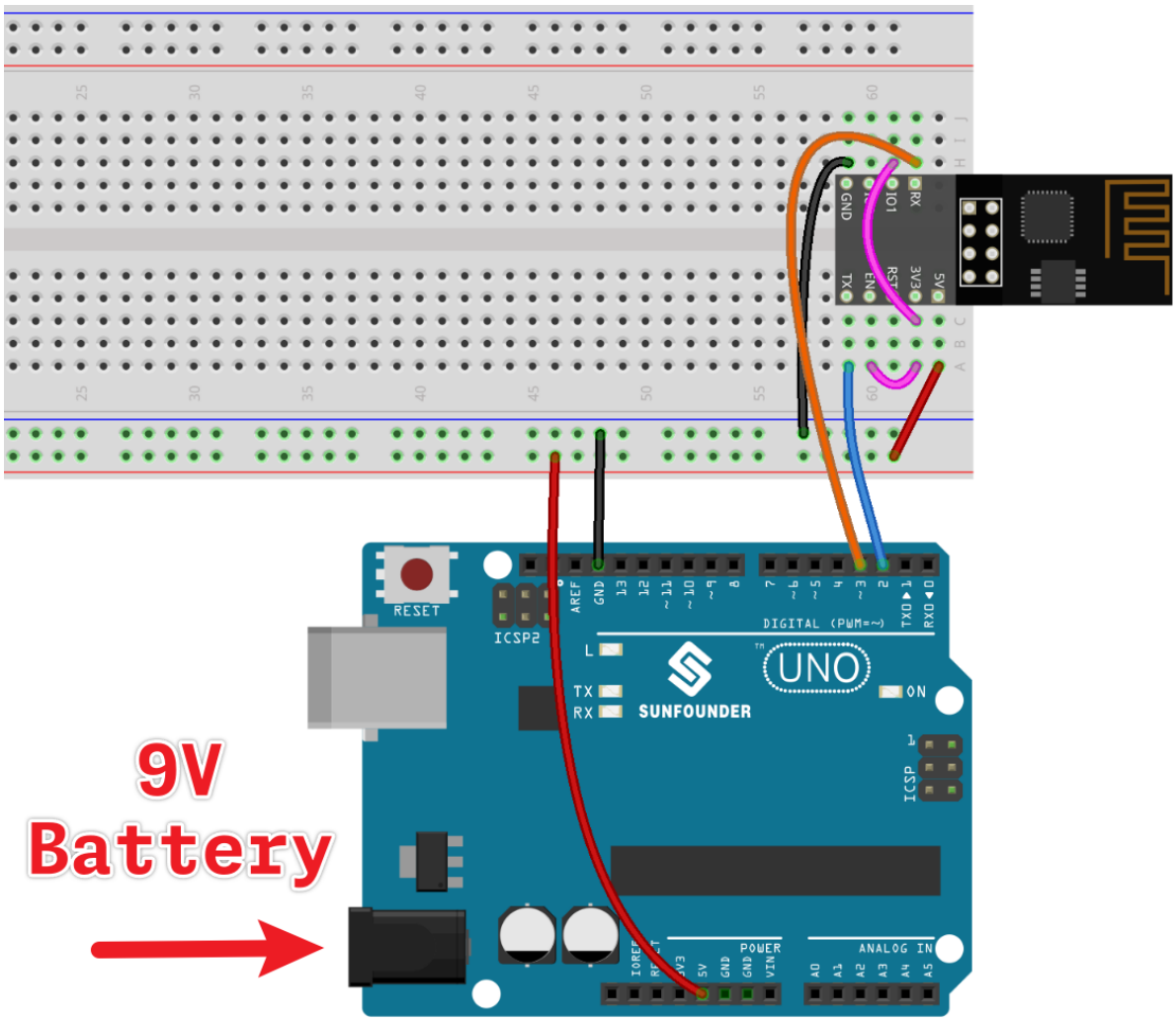
Ready to explore and create with us? Click [] and join today!

1.3.36 Lesson 35: Get Started with ESP8266 Module

The ESP8266 module that comes with the kit is already pre-burned with AT firmware, but you still need to modify its configuration by following the steps below.

1. Build the circuit.

   Note: To ensure the ESP8266 gets a stable voltage, please connect it to an external power source like the 9V battery that comes with the kit, by hooking it up to the Uno board.



2. Open the `.ino` file under the path of `universal-maker-sensor-kit\arduino_uno\Lesson_35_ESP8266`. Or copy this code into Arduino IDE. And upload the code.

   The code establishes a software serial communication using Arduino's SoftwareSerial library, allowing the Arduino to communicate with the ESP8266 module through its digital pins 2 and 3 (as Rx and Tx). It checks for data transfer between them, forwarding received messages from one to the other at a baud rate of 115200. **With this code, you can use the Arduino's serial monitor to send AT firmware commands to the ESP8266 module and receive its responses.**

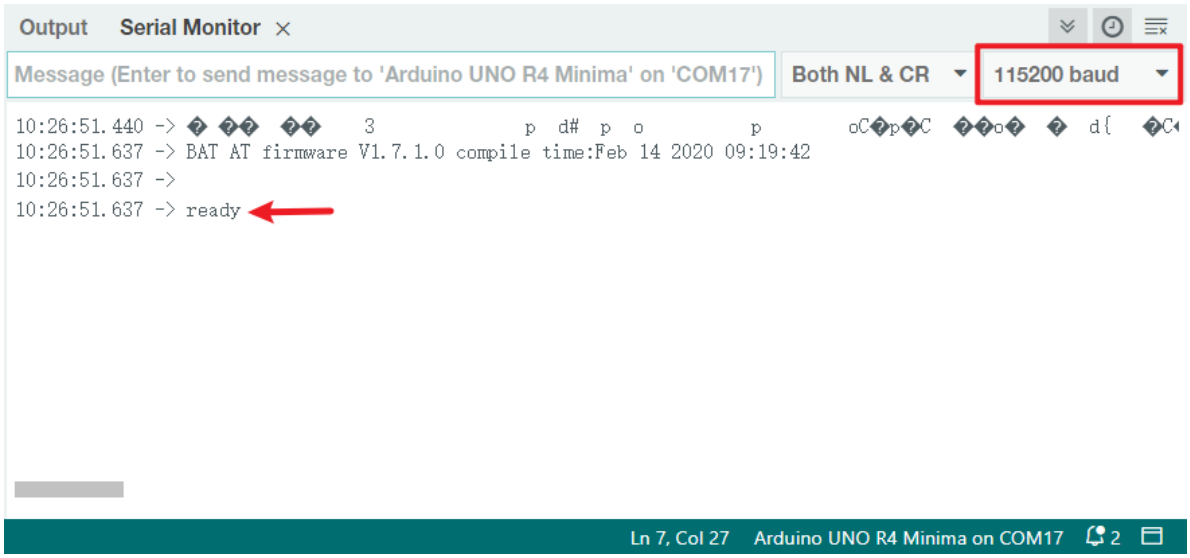
```
#include <SoftwareSerial.h>
SoftwareSerial espSerial(2, 3); //Rx,Tx

void setup() {
    // put your setup code here, to run once:
    Serial.begin(115200);
    espSerial.begin(115200);
}

void loop() {
    if (espSerial.available()) {
        Serial.write(espSerial.read());
```

(continues on next page)

(continued from previous page)

```
    }
    if (Serial.available()) {
        espSerial.write(Serial.read());
    }
}
```

3. Click the magnifying glass icon (Serial Monitor) in the upper right corner and set the baud rate to **115200**. (You may have some printed information like me, or you may not, it doesn't matter, just go to the next step.)

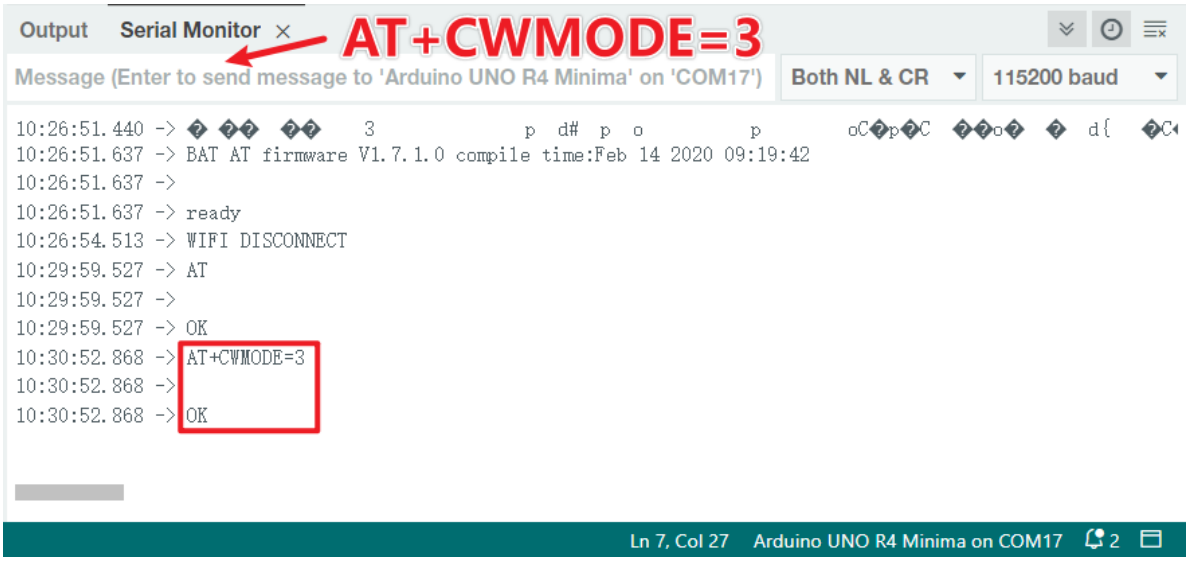


Warning:

- If `ready` doesn't appear, you can try to reset the ESP8266 module(connect RST to GND) and re-open the Serial Monitor.
- In addition, if the result is `OK`, you may need to re-burn the firmware, please refer to *How to re-burn the AT firmware for ESP8266 module?* for details. If you still can't solve it, please take a screenshot of the serial monitor and send it to service@sunfounder.com, we will help you solve the problem as soon as possible.

4. Click on **NEWLINE DROPDOWN BOX**, select `both NL & CR` in the drop down option, enter `AT`, if it returns OK, it means ESP8266 has successfully established connection with R4 board.

5. Enter `AT+CWMODE=3` and the managed mode will be changed to **Station and AP** coexistence.



Reference

-

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.37 Lesson 36: Get Started with Bluetooth Module

In this project, we demonstrate how to communicate with a Bluetooth module through Arduino.

Firstly, we need to set up the circuit and use software serial communication. Connect the TX pin of the Bluetooth module to pin 3 of the Uno board, and connect the RX pin of the Bluetooth module to pin 4 of the Uno board.

Required Components

In this project, we need the following components.

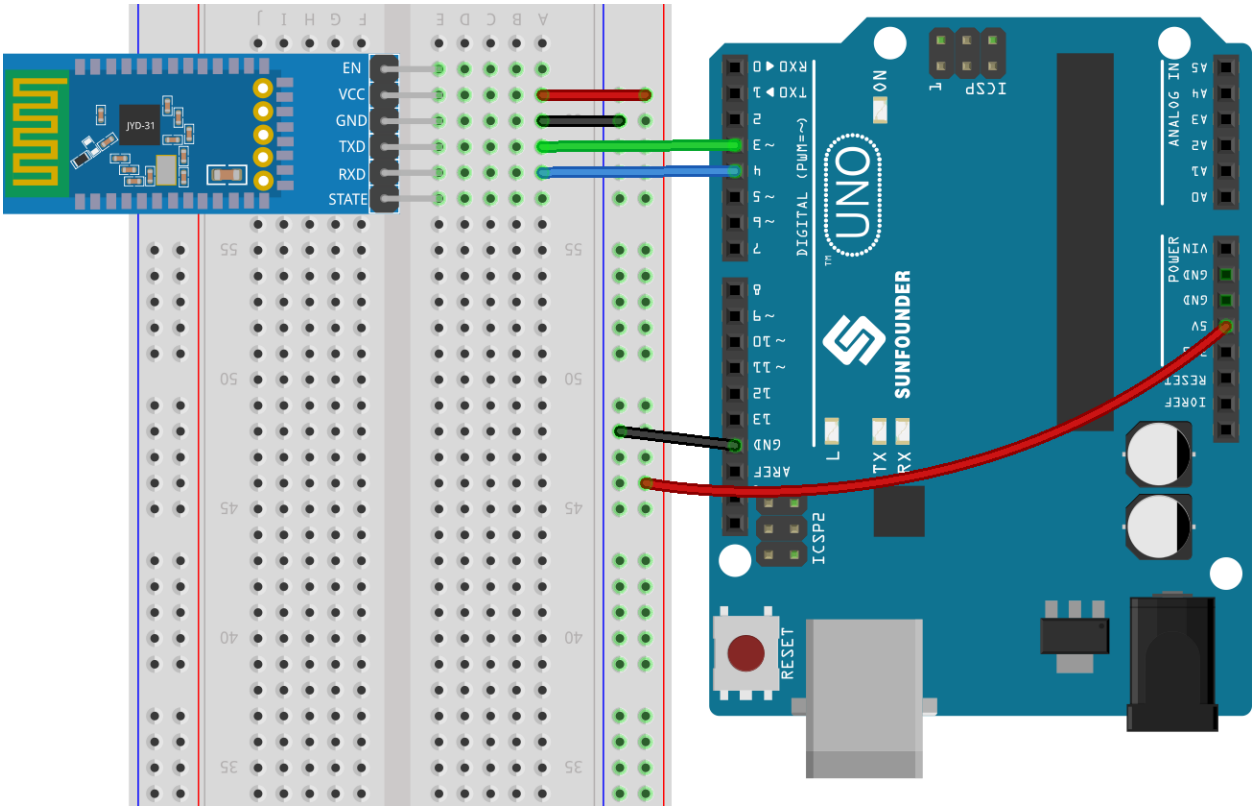
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
JDY-31 Bluetooth Module	
Breadboard	

1. Build the Circuit

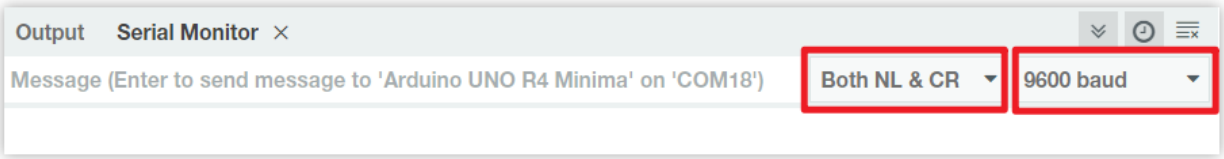


2. Upload the code

The code establishes a software serial communication using Arduino's SoftwareSerial library, allowing the Arduino to communicate with the JDY-31 Bluetooth module through its digital pins 3 and 4 (as Rx and Tx). It checks for data transfer between them, forwarding received messages from one to the other at a baud rate of 9600. **With this code, you can use the Arduino's serial monitor to send AT commands to the JDY-31 Bluetooth module and receive its responses**.

3. Configuring the Bluetooth module

Click the magnifying glass icon (Serial Monitor) in the upper right corner and set the baud rate to `9600`. Then, select `both NL & CR` from the drop-down option of the `New Line` dropdown box.



The following are some examples of using AT commands to configure Bluetooth modules: Enter `AT+NAME` to obtain the name of the Bluetooth device. If you want to modify the Bluetooth name, please add a new name after `AT+NAME`.

- **Query the name of a Bluetooth device:** `AT+NAME`

- **Set Bluetooth device name:** `AT+NAME` (following by the new name). `+OK` means the setting was successful. You can send `AT+NAME` again to verify.

Note: To ensure consistency in the learning experience, it is recommended not to modify the default baud rate of the Bluetooth module and **keep it at its default value of 4 (i.e. 9600 baud rate)**. In relevant courses, we communicate with Bluetooth using a baud rate of 9600.

- **Set Bluetooth baudrate:** `AT+BAUD` (followed by the number indicating the baudrate).
  - 4 == 9600
  - 5 == 19200
  - 6 == 38400
  - 7 == 57600
  - 8 == 115200
  - 9 == 128000

Please refer to the table below for more AT commands.

Command	Function	Default
AT+VERSION	Version Number	JDY-31-V1.2
AT+RESET	Soft reset	
AT+DISC	Disconnect (valid when connected)	
AT+LADDR	Query the MAC address of the module	
AT+PIN	Set or query connection password	1234
AT+BAUD	Set or query baud rate	9600
AT+NAME	Set or query broadcast name	JDY-31-SPP
AT+DEFAULT	Factory reset	
AT+ENLOG	Serial port status output	1

4. Communicating through Bluetooth debugging tools on mobile phones

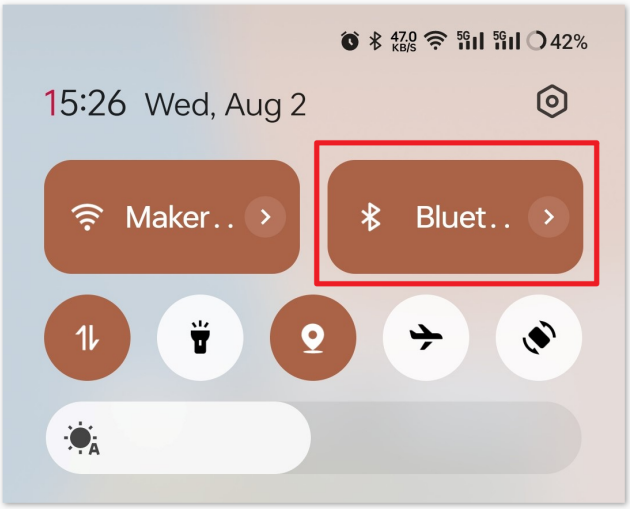
We can use an app called "Serial Bluetooth Terminal" to send messages from the Bluetooth module to Arduino, simulating the process of Bluetooth interaction. The Bluetooth module will send received messages to Arduino through serial port, and similarly, Arduino can also send messages to bluetooth module through serial port.

a. **Install Serial Bluetooth Terminal**

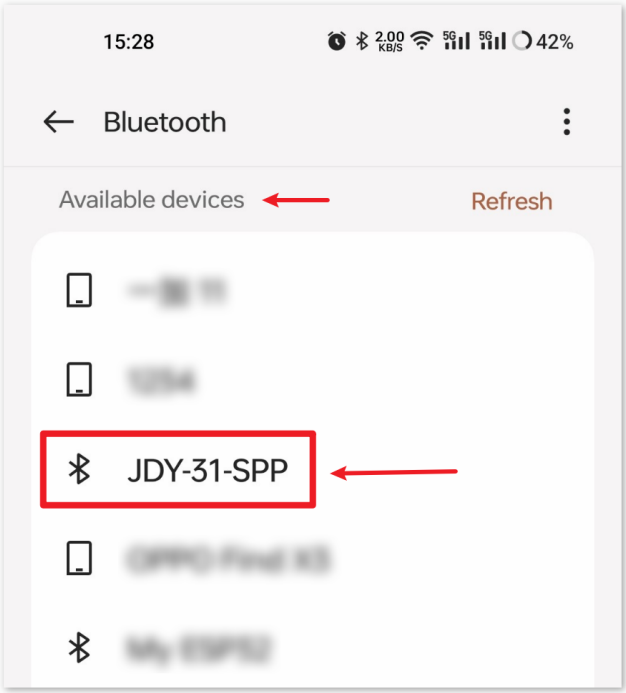
   Go to Google Play to download and install .

b. **Connect Bluetooth**

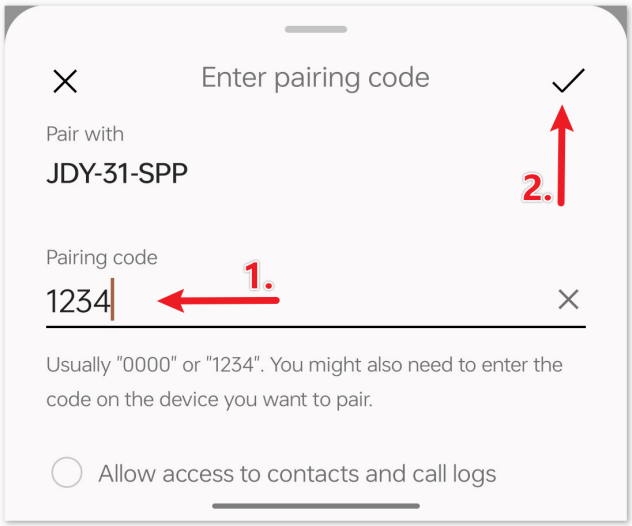
   Initially, turn on **Bluetooth** on your smartphone.



Navigate to the **Bluetooth settings** on your smartphone and look for names like **JDY-31-SPP**.

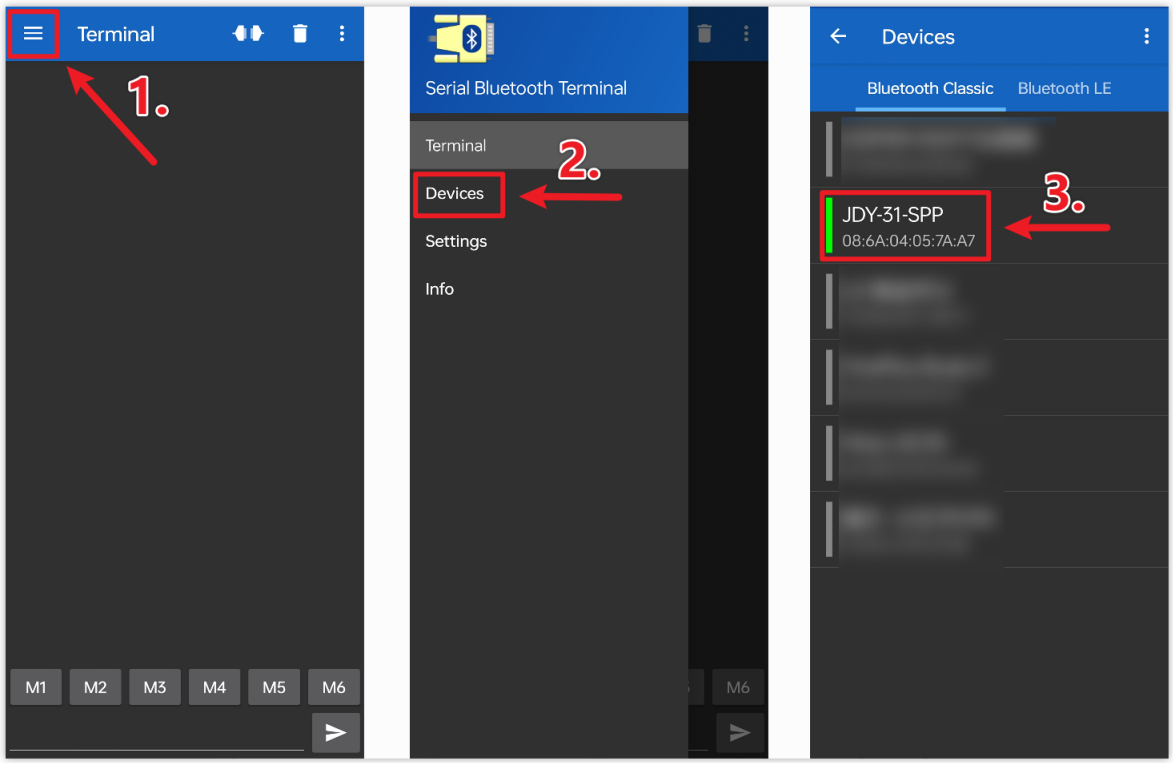


After clicking it, agree to the **Pair** request in the pop-up window. If prompted for a pairing code, please enter "1234".



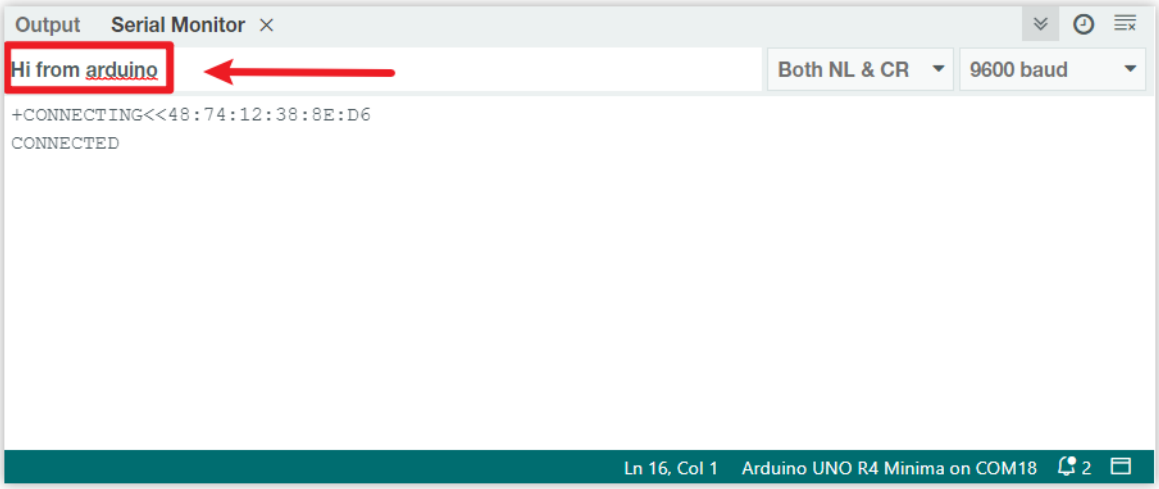
c. **Communicate with Bluetooth module**

Open the Serial Bluetooth Terminal. Connect to "JDY-31-SPP".

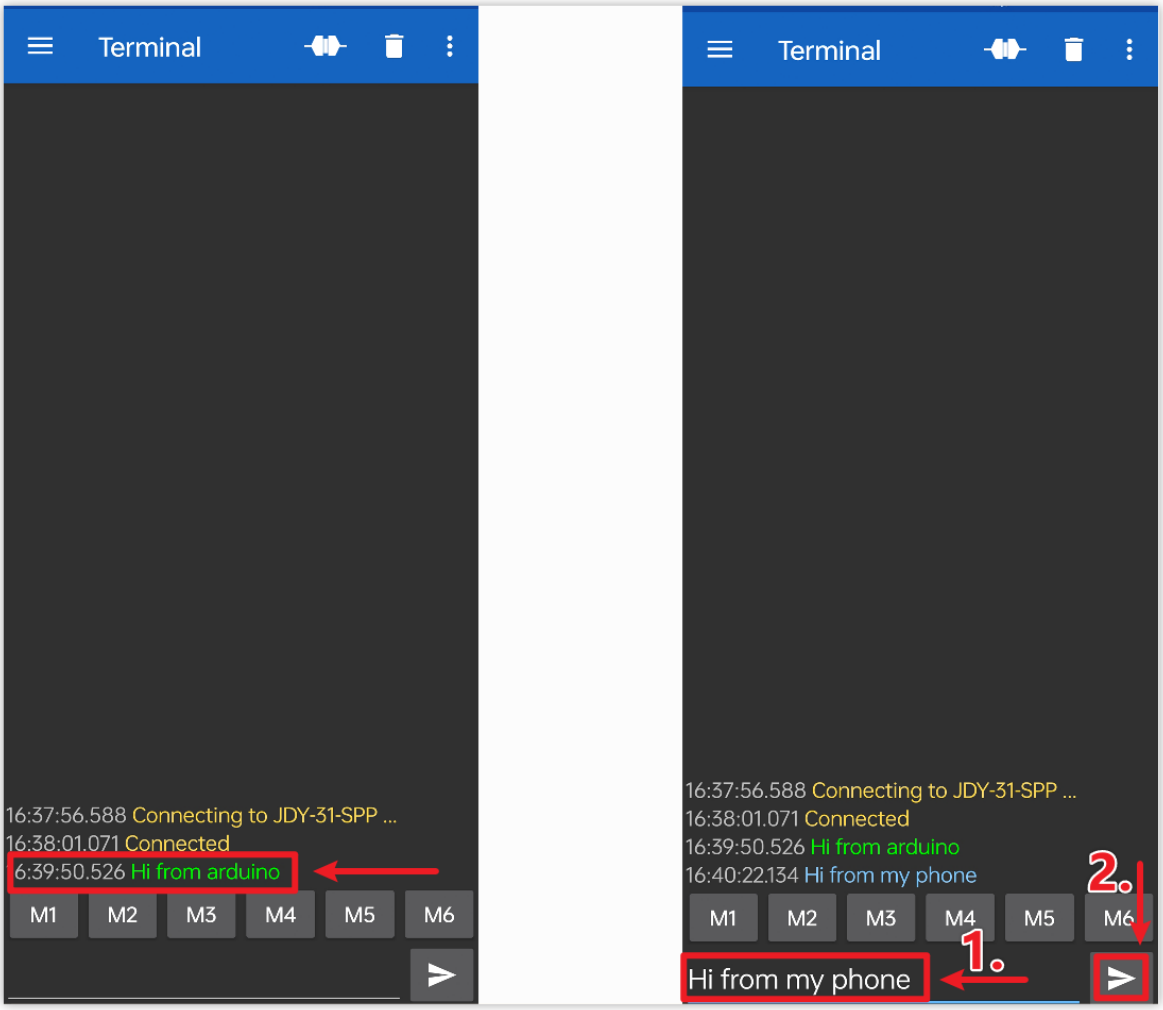


After successful connection, you can see the prompt of successful connection in the serial port monitor.

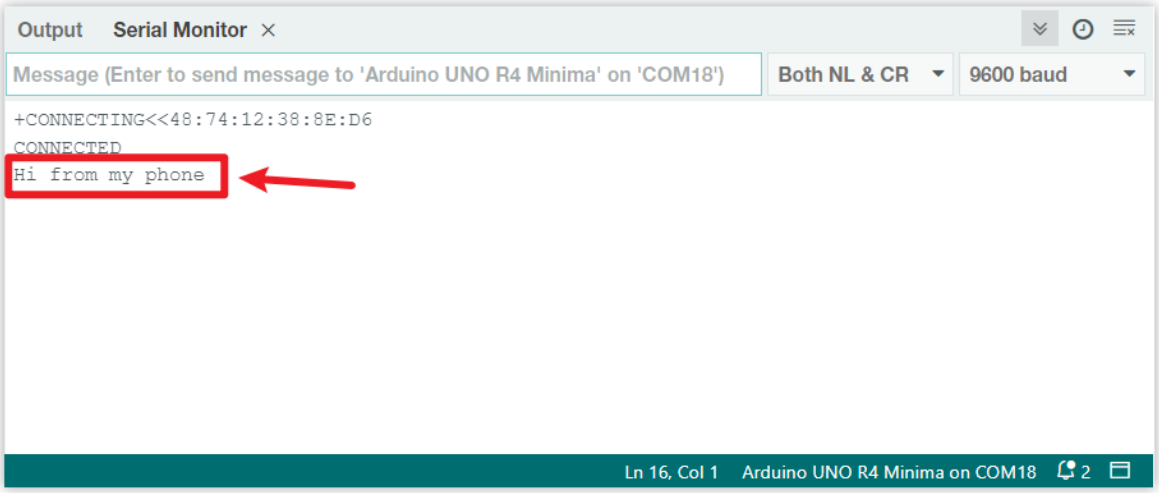
Input the message in the serial monitor and send it to the Bluetooth module.



After sending, you can see this message in the Serial Bluetooth Terminal APP. Similarly, data can be sent to Arduino via Bluetooth in **Serial Bluetooth Terminal** APP.



You can see this message from Bluetooth in the serial monitor.



Fun Project

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.38 Lesson 37: Smart trashcan

This project revolves around the concept of a smart trash can. The primary aim is to have the trash can's lid automatically open when an object approaches within a set distance (20cm in this case). The functionality is achieved by using an ultrasonic distance sensor paired with a servo motor. The distance between the object and the sensor is continually measured. If the object is close enough, the servo motor is triggered to open the lid.

Required Components

In this project, we need the following components.

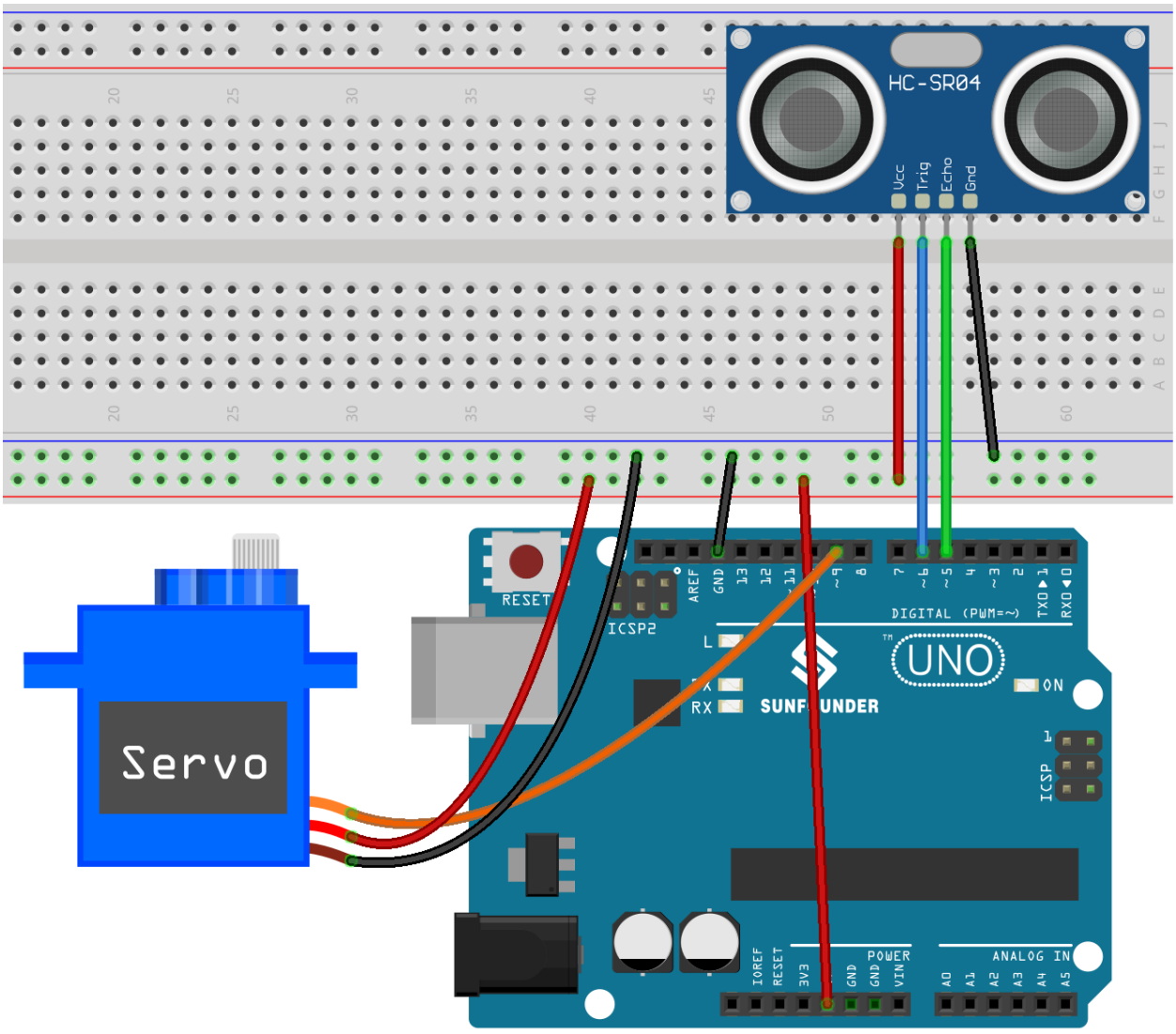
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Ultrasonic Sensor Module (HC-SR04)	
Servo Motor (SG90)	
Breadboard	

Wiring



Code

Code Analysis

The project is based on real-time monitoring of the distance between an object and a trash can. An ultrasonic sensor continuously measures this distance, and if an object approaches within 20cm, the trash can interprets it as an intention to dispose of waste and automatically opens its lid. This automation adds smartness and convenience to a regular trash can.

1. Initial Setup and Variable Declaration

   Here, we're including the `Servo` library and defining the constants and variables we'll use. The pins for the servo and the ultrasonic sensor are declared. We also have an array `averDist` to hold the three distance measurements.

```
#include <Servo.h>
Servo servo;
```

(continues on next page)

(continued from previous page)

```
const int servoPin = 9;
const int openAngle = 0;
const int closeAngle = 90;
const int trigPin = 6;
const int echoPin = 5;
long distance, averageDistance;
long averDist[3];
const int distanceThreshold = 20;
```

2. `setup()` Function

   The `setup()` function initializes serial communication, configures the ultrasonic sensor's pins, and sets the initial position of the servo to the closed position.

```
void setup() {
  Serial.begin(9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  servo.attach(servoPin);
  servo.write(closeAngle);
  delay(100);
}
```

3. `loop()` Function

   The `loop()` function is responsible for continuously measuring the distance, computing its average, and then making a decision whether to open or close the trash can's lid based on this averaged distance.

```
void loop() {
  for (int i = 0; i <= 2; i++) {
    distance = readDistance();
    averDist[i] = distance;
    delay(10);
  }
  averageDistance = (averDist[0] + averDist[1] + averDist[2]) / 3;
  Serial.println(averageDistance);
  if (averageDistance <= distanceThreshold) {
    servo.write(openAngle);
    delay(3500);
  } else {
    servo.write(closeAngle);
    delay(1000);
  }
}
```

4. Distance Reading Function

   This function, `readDistance()`, is what actually interacts with the ultrasonic sensor. It sends a pulse and waits for an echo. The time taken for the echo is then used to calculate the distance between the sensor and any object in front of it.

   You can refer to the *Principle* of the ultrasonic sensor.

```
float readDistance() {
  digitalWrite(trigPin, LOW);
```

(continues on next page)

(continued from previous page)

```
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  float distance = pulseIn(echoPin, HIGH) / 58.00;
  return distance;
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.39 Lesson 38: Gas leak alarm

This project revolves around simulating a gas leak detection scenario using an Arduino Uno board. By incorporating an MQ-2 gas sensor and an RGB LED, this demonstration continuously reads the gas concentration. If this concentration surpasses a predefined threshold, it activates an alarm (buzzer) and illuminates the RGB LED in red. Conversely, if the concentration remains below this threshold, the alarm remains inactive and the LED shines green. It's crucial to note that this demo is purely illustrative and shouldn't replace real gas leak detection systems.

Required Components

In this project, we need the following components.

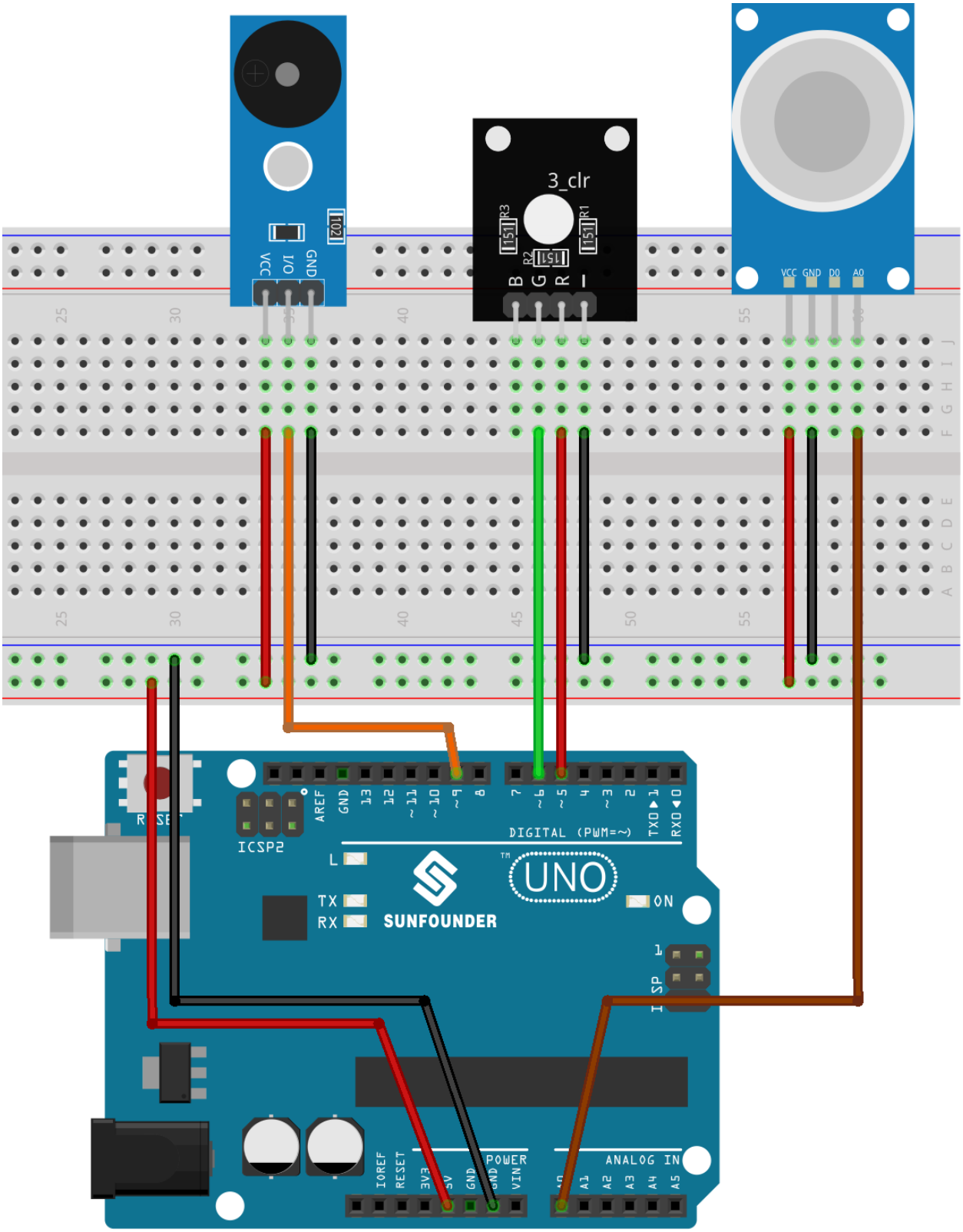
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Gas/Smoke Sensor Module (MQ2)	
Passive Buzzer Module	
RGB LED Module	-
Breadboard	

Wiring


VCC
I/O
GND
3_clr
R3
R2
R1
B
G
R
I
VCC
GND
DO
AO
ICSP2
DIGITAL (PWM=~)
L
TX
RX
SUNFOUNDER
UNO
ON
POWER
ANALOG IN

Code

Code Analysis

The core principle of the project revolves around continuously monitoring the gas concentration. When the detected gas concentration surpasses a certain threshold, it sets off an alarm and changes the LED's color to red. This serves as a simulated warning mechanism, indicative of potentially hazardous conditions. If the concentration drops below the threshold, the alarm is deactivated and the LED switches to green, indicating a safe environment.

1. Defining Constants and Variables

   These lines declare and initialize the pin numbers for various components. The `sensorPin` denotes the analog pin where the MQ-2 gas sensor is connected. `sensorValue` is an integer variable storing the sensor's analog output. The `buzzerPin` indicates the digital pin to which the buzzer is connected. Finally, the `RPin` and `GPin` are the pins for the red and green channels of the RGB LED, respectively.

```
// Define the pin numbers for the Gas Sensor
const int sensorPin = A0;
int sensorValue;

// Define the pin number for the buzzer
const int buzzerPin = 9;

// Define pin numbers for the RGB LED
const int RPin = 5;  // R channel of RGB LED
const int GPin = 6;  // G channel of RGB LED
```

2. Initialization in `setup()`

   The `setup()` function initializes the required settings. Serial communication begins at a baud rate of 9600, allowing us to view sensor readings on the Serial Monitor. Pins for the buzzer and RGB LED are set as `OUTPUT`, meaning they'll send signals out to external components.

```
void setup() {
  Serial.begin(9600);  // Start serial communication at 9600 baud rate

  // Initialize the buzzer and RGB LED pins as output
  pinMode(buzzerPin, OUTPUT);
  pinMode(RPin, OUTPUT);
  pinMode(GPin, OUTPUT);
}
```

3. Main Loop: Reading Sensor and Triggering Alarm

   The `loop()` function continually reads the gas sensor's output. The reading is then displayed on the Serial Monitor for observation. Depending on the sensor value, two scenarios can occur:

   - If the value exceeds 300, the buzzer is activated using `tone()`, and the RGB LED turns red.
   - If the value is below 300, the buzzer is silenced using `noTone()`, and the LED turns green.

   Lastly, a delay of 50 milliseconds is introduced before the next loop iteration to manage the read frequency and reduce the CPU load.

```
void loop() {
  // Read the analog value of the gas sensor
  sensorValue = analogRead(sensorPin);
```

(continues on next page)

(continued from previous page)

```
  // Print the sensor value to the serial monitor
  Serial.print("Analog output: ");
  Serial.println(sensorValue);

  // If the sensor value exceeds the threshold, trigger the alarm and make the RGB␣
↪LED red
  if (sensorValue > 300) {
    tone(buzzerPin, 500, 300);
    digitalWrite(GPin, LOW);
    digitalWrite(RPin, HIGH);
  } else {
    // If the sensor value is below the threshold, turn off the alarm and make the␣
↪RGB LED green
    noTone(buzzerPin);
    digitalWrite(RPin, LOW);
    digitalWrite(GPin, HIGH);
  }

  // Wait for 50 milliseconds before the next loop iteration
  delay(50);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.40 Lesson 39: Automatic soap dispenser

The Automatic Soap Dispenser project uses an Arduino Uno board along with an infrared obstacle avoidance sensor and a water pump. The sensor detects the presence of an object such as a hand, which activates the water pump to dispense soap.

Required Components

In this project, we need the following components.

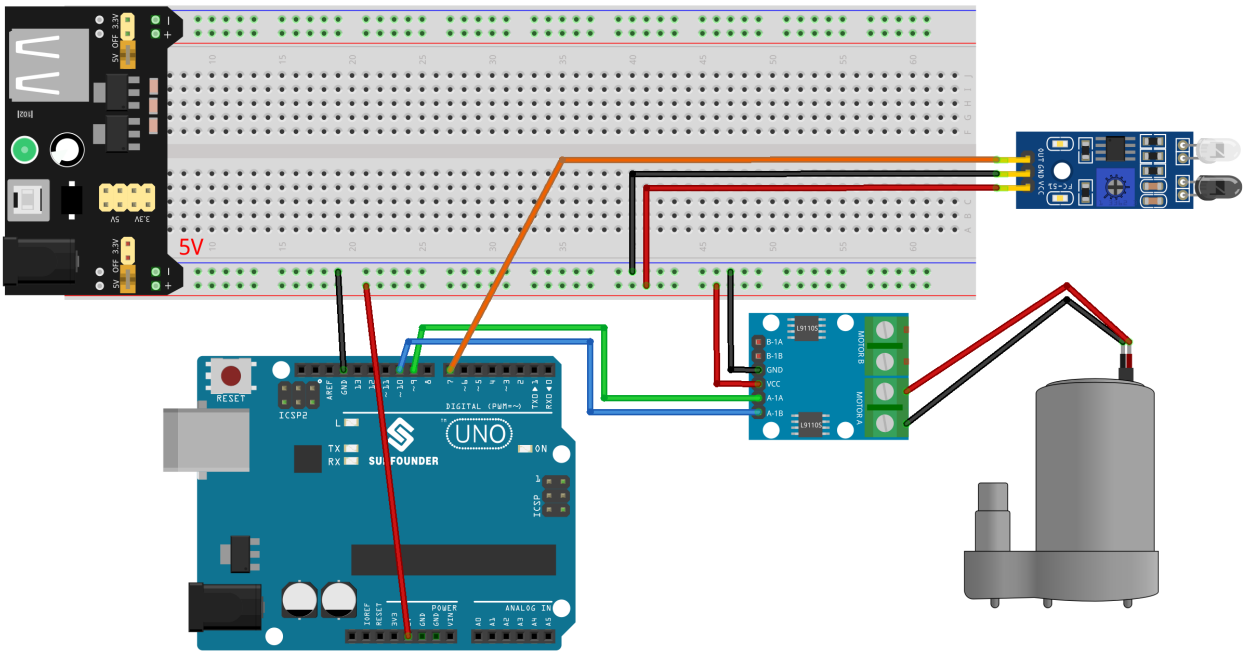
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
IR Obstacle Avoidance Sensor Module	
Centrifugal Pump	-
L9110 Motor Driver Module	-
Power Supply Module	-
Breadboard	

Wiring



Code

Code Analysis

The main idea behind this project is to create a hands-free soap dispensing system. The infrared obstacle avoidance sensor detects when an object (like a hand) is close. Upon detecting an object, the sensor sends a signal to the Arduino, which in turn triggers the water pump to dispense soap. The pump stays active for a brief period, dispensing soap, then turns off.

1. **Defining the pins for the sensor and the pump**

   In this code snippet, we define the Arduino pins that connect to the sensor and pump. We define pin 7 as the sensor pin and we will use the variable `sensorValue` to store the data read from this sensor. For the water pump, we use two pins, 9 and 10.

```
const int sensorPin = 7;
int sensorValue;
const int pump1A = 9;
const int pump1B = 10;
```

2. **Setting up the sensor and pump**

   In the `setup()` function, we define the modes for the pins we're using. The sensor pin is set to `INPUT` as it will be used to receive data from the sensor. The pump pins are set to `OUTPUT` as they will send commands to the pump. We ensure that the pin `pump1B` starts in a `LOW` state (off), and we start the serial communication with a baud rate of 9600.

```
void setup() {
  pinMode(sensorPin, INPUT);
  pinMode(pump1A, OUTPUT);
  pinMode(pump1B, OUTPUT);
  digitalWrite(pump1B, LOW);
  Serial.begin(9600);
}
```

3. **Continuously checking the sensor and controlling the pump**

   In the `loop()` function, the Arduino constantly reads the value from the sensor using `digitalRead()` and assigns it to `sensorValue()`. It then prints this value to the serial monitor for debugging purposes. If the sensor detects an object, `sensorValue()` will be 0. When this happens, `pump1A` is set to `HIGH`, activating the pump, and a delay of 700 milliseconds allows the pump to dispense soap. The pump is then deactivated by setting `pump1A` to `LOW`, and a 1-second delay gives the user time to move their hand away before the cycle repeats.

Note: If the sensor is not working properly, adjust the IR transmitter and receiver to make them parallel. Additionally, you can adjust the detection range using the built-in potentiometer.

```
void loop() {
  sensorValue = digitalRead(sensorPin);
  Serial.println(sensorValue);
  if (sensorValue == 0) {
    digitalWrite(pump1A, HIGH);
    delay(700);
    digitalWrite(pump1A, LOW);
    delay(1000);
```

(continues on next page)

(continued from previous page)

```
  }
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.41 Lesson 40: Motion triggered relay

This Arduino project aims to control a relay-operated light using a passive infrared (PIR) sensor. When the PIR sensor detects motion, the relay is activated, turning the light on. The light remains on for 5 seconds after the last detected motion.

Warning: As a demonstration, we are using a relay to control an RGB LED module. However, in real-life scenarios, this may not be the most practical approach.

While you can connect the relay to other appliances in actual applications, extreme caution is required when dealing with HIGH AC voltage. Improper or incorrect use can lead to severe injury or even death. Therefore, it is intended for people who are familiar with and knowledgeable about HIGH AC voltage. Always prioritize safety.

Required Components

In this project, we need the following components.

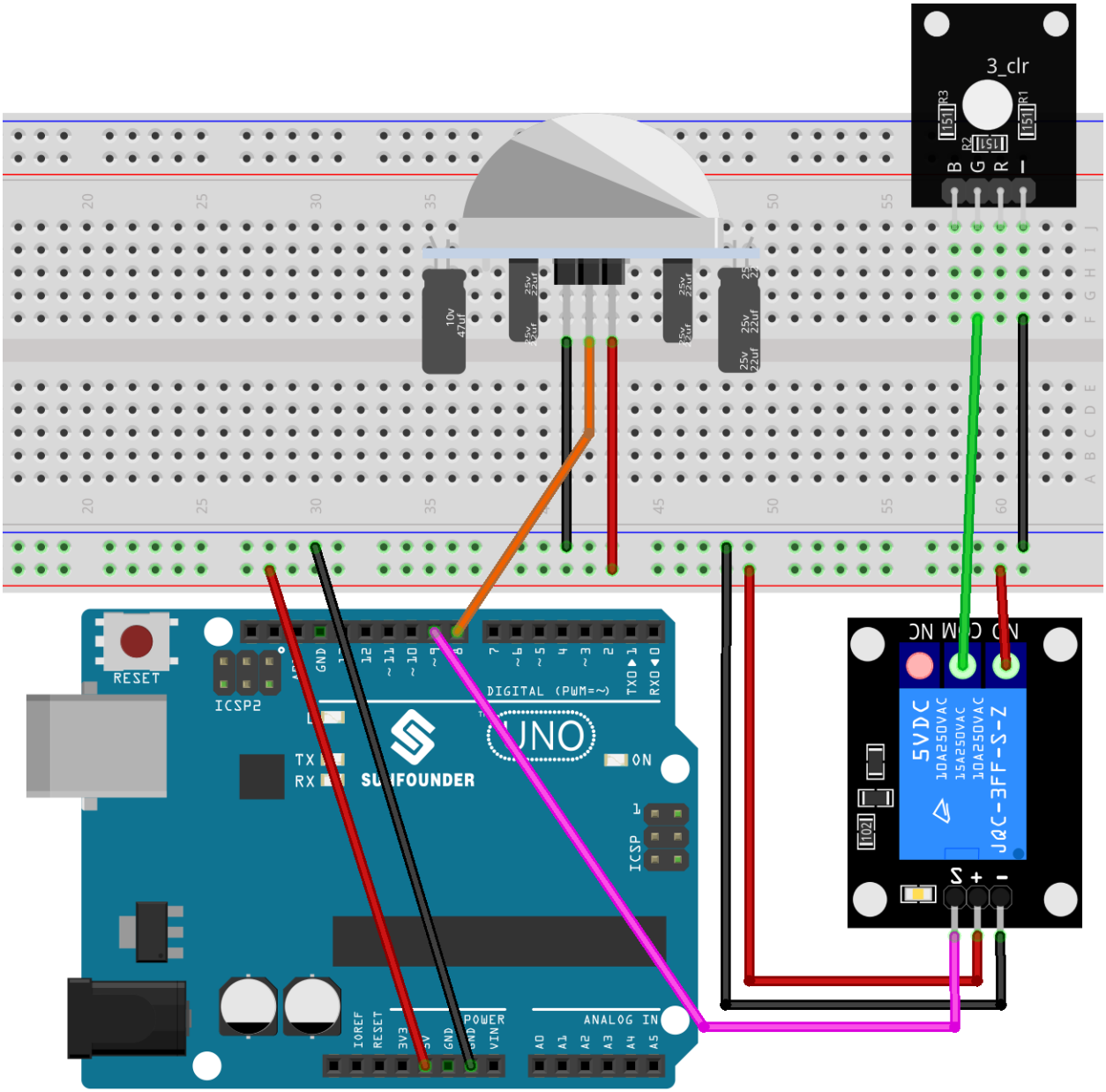
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
PIR Motion Module (HC-SR501)	-
5V Relay Module	-
RGB LED Module	-
Breadboard	

Wiring



Code

Code Analysis

The project revolves around the PIR motion sensor's capability to detect motion. When motion is detected, a signal is sent to the Arduino, triggering the relay module, which in turn activates a light. The light stays on for a specified duration (in this case, 5 seconds) after the last detected motion, ensuring the area remains illuminated for a short period even if motion ceases.

1. **Initial setup and variable declarations**

   This segment defines constants and variables that will be used throughout the code. We set up the relay and PIR pins and a delay constant for motion. We also have a variable to keep track of the last detected motion time and a flag to monitor if motion is detected.

```
// Define the pin number for the relay
const int relayPin = 9;

// Define the pin number for the PIR sensor
const int pirPin = 8;

// Motion delay threshold in milliseconds
const unsigned long MOTION_DELAY = 5000;

unsigned long lastMotionTime = 0;  // Timestamp of the last motion detection
bool motionDetected = false;       // Flag to track if motion is detected
```

2. **Configuration of pins in setup() function**

   In the `setup()` function, we configure the pin modes for both the relay and PIR sensor. We also initialize the relay to be off at the start.

```
void setup() {
  pinMode(relayPin, OUTPUT);    // Set relayPin as an output pin
  pinMode(pirPin, INPUT);       // Set the PIR pin as an input
  digitalWrite(relayPin, LOW);  // Turn off the relay initially
}
```

3. **Main logic in loop() function**

   The `loop()` function contains the primary logic. When the PIR sensor detects motion, it sends a `HIGH` signal, turning on the relay and updating the `lastMotionTime`. If there's no motion for the specified delay (5 seconds in this case), the relay is turned off.

   This approach ensures that even if motion is sporadic or brief, the light remains on for at least 5 seconds after the last detected motion, providing a consistent illumination duration.

```
void loop() {
  if (digitalRead(pirPin) == HIGH) {
    lastMotionTime = millis();     // Update the last motion time
    digitalWrite(relayPin, HIGH);  // Turn on the relay (and hence the light)
    motionDetected = true;
  }

  // If motion was detected earlier and 5 seconds have elapsed, turn off the relay
  if (motionDetected && (millis() - lastMotionTime >= MOTION_DELAY)) {
```

(continues on next page)

(continued from previous page)

```
    digitalWrite(relayPin, LOW);  // Turn off the relay
    motionDetected = false;
  }
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.42 Lesson 41: Heart rate monitor

This Arduino project aims to build a simple Heart Rate Monitor using a MAX30102 pulse oximeter sensor and an SSD1306 OLED Display. The code takes measurements of the heart rate by determining the time between heartbeats. By taking four measurements, it computes their average and presents the resultant average heart rate on an OLED screen. If the sensor doesn't detect a finger, it sends a prompt to the user to position their finger correctly on the sensor.

Required Components

In this project, we need the following components.

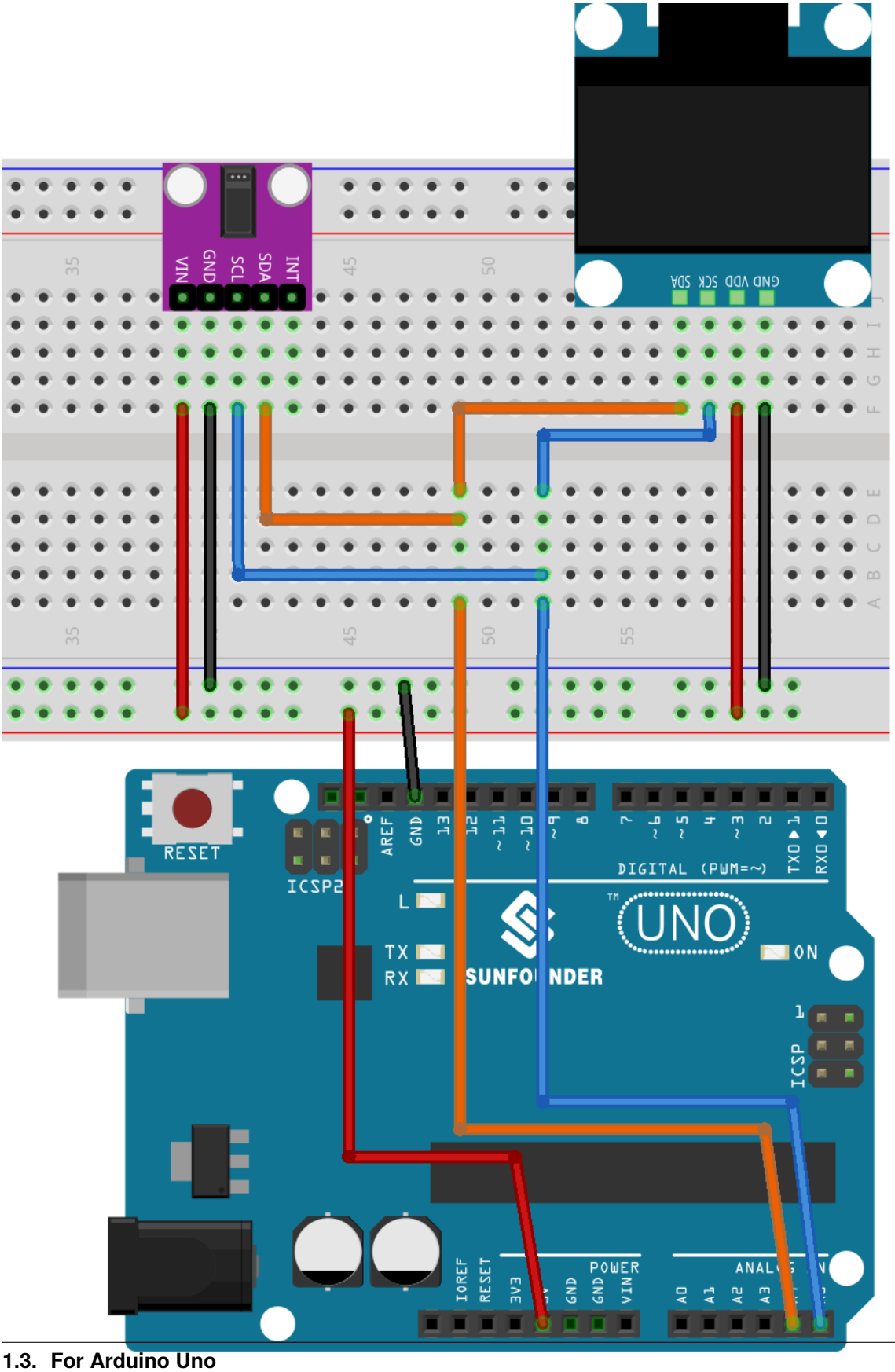
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Pulse Oximeter and Heart Rate Sensor Module (MAX30102)	
OLED Display Module (SSD1306)	-
Breadboard	

Wiring


VIN
GND
SCL
SDA
INT
GND VDD SCK SDA
RESET
ICSP2
AREF
GND
DIGITAL (PWM=~)
TX
RX
L
ON
UNO
ICSP
POWER
IOREF
RESET
3V3
GND
GND
VIN
A0
A1
A2
A3

Code

Note: To install the library, open the Arduino Library Manager, search for **"SparkFun MAX3010x"** , **"Adafruit SSD1306"** , and **"Adafruit GFX"**, then install them.

Code Analysis

The main principle behind this project is to capture the pulsation of blood flow through a finger using the MAX30102 sensor. As blood pumps through the body, it causes tiny changes in the volume of blood in the vessels of the fingertip. By shining light through the finger and measuring the amount of light that gets absorbed or reflected back, the sensor detects these minute volume changes. The time interval between subsequent pulses is then used to calculate the heart rate in beats per minute (BPM). This value is then averaged over four measurements and displayed on the OLED screen.

1. **Library Inclusions and Initial Declarations**:

   The code begins by including necessary libraries for the OLED display, MAX30102 sensor, and heart rate calculation. Additionally, the configuration for the OLED display and the variables for heart rate calculation are declared.

   Note: To install the library, open the Arduino Library Manager, search for **"SparkFun MAX3010x"** , **"Adafruit SSD1306"** , and **"Adafruit GFX"**, then install them.

   ```
   #include <Adafruit_GFX.h>  // OLED  libraries
   #include <Adafruit_SSD1306.h>
   #include <Wire.h>
   #include "MAX30105.h"   // MAX3010x library
   #include "heartRate.h"  // Heart rate  calculating algorithm

   // ... Variables and OLED configuration
   ```

   In this project, we've also whipped up a couple of bitmaps. The `PROGMEM` keyword denotes that the array is stored in the program memory of the Arduino microcontroller. Storing data in program memory(PROGMEM) instead of RAM can be helpful for large amounts of data, which would otherwise take up too much space in RAM.

   ```
   static const unsigned char PROGMEM beat1_bmp[] = {...}

   static const unsigned char PROGMEM beat2_bmp[] = {...}
   ```

2. **Setup Function**:

   Initializes I2C communication, starts serial communication, initializes the OLED display, and sets up the MAX30102 sensor.

   ```
   void setup() {
       Wire.setClock(400000);
       Serial.begin(9600);
       display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);
       // ... Rest of the setup code
   ```

3. **Main Loop**:

The core functionality resides here. The IR value is read from the sensor. If a finger is detected (IR value greater than 50,000), the program checks if a heartbeat is sensed. When a heartbeat is detected, the OLED screen displays the BPM and the time between heartbeats is used to calculate BPM. Otherwise, it prompts the user to place their finger on the sensor.

We have also prepared two bitmaps with heartbeats, and by switching between these two bitmaps, we can achieve a dynamic visual effect.

```
void loop() {
  // Get IR value from sensor
  long irValue = particleSensor.getIR();

  //If a finger is detected
  if (irValue > 50000) {

    // Check if a beat is detected
    if (checkForBeat(irValue) == true) {

      // Update OLED display
      // Calculate the BPM

      // Calculate the average BPM
      //Print the IR value, current BPM value, and average BPM value to the serial␣
↪monitor

      // Update OLED display

    }
  }
  else {
    // ... Prompt to place the finger on the sensor
  }
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.43 Lesson 42: Touch toggle light

This project is a simple implementation of a traffic light control system utilizing a touch sensor and a traffic light LED module. Activating the touch sensor initiates a sequence where LEDs illuminate in the following order: Red -> Yellow -> Green.

Required Components

In this project, we need the following components.

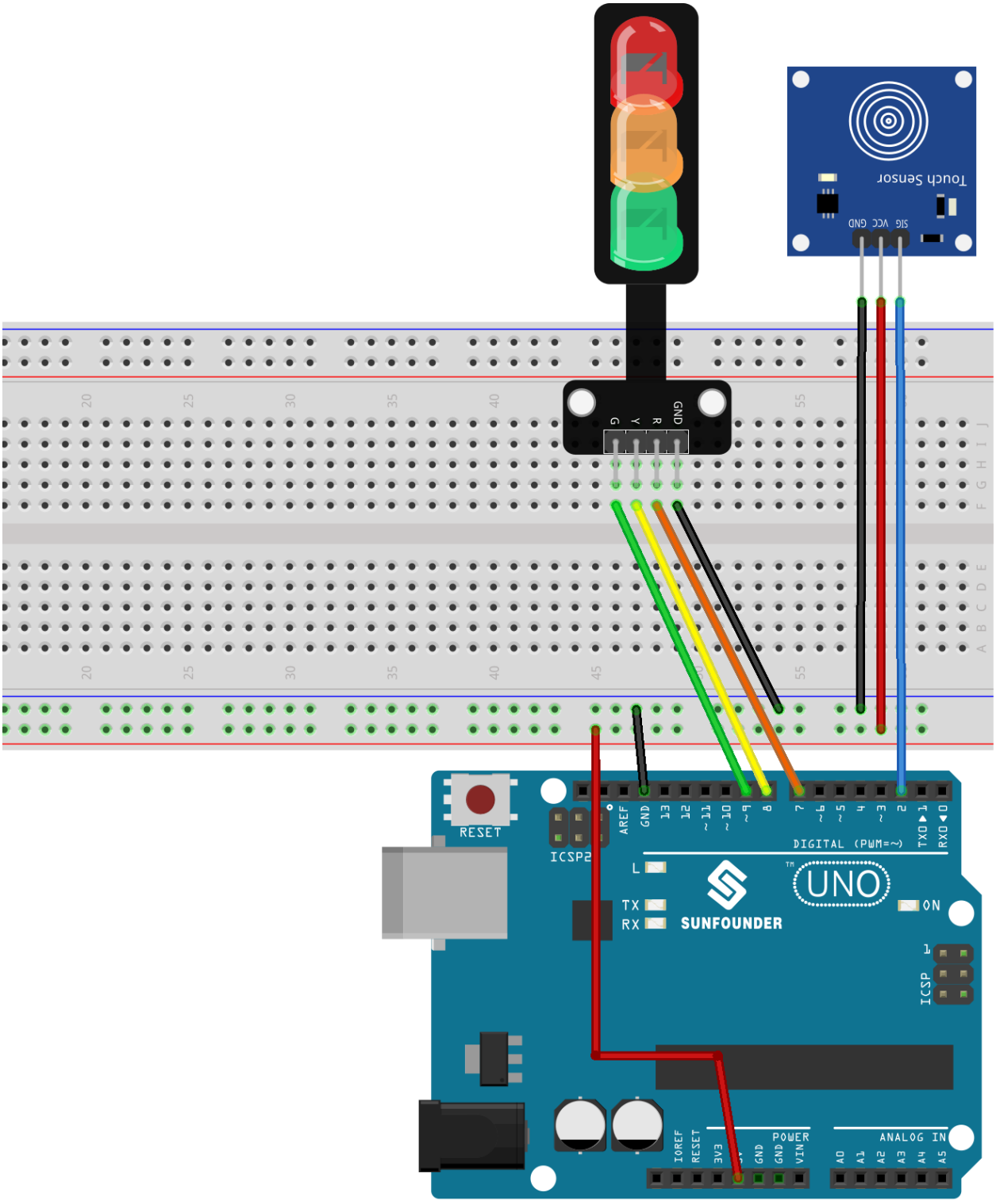
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Touch Sensor Module	-
Traffic Light Module	-
Breadboard	

Wiring


Touch Sensor
SIG VCC GND
G Y R GND
RESET
ICSP2
AREF GND 13 12 ~11 ~10 ~9 8 7 ~6 ~5 4 ~3 2 TX→1 RX←0
DIGITAL (PWM=~)
L
TX
RX
SUNFOUNDER
UNO
ON
ICSP
IOREF RESET 3V3 5V GND GND VIN
POWER
ANALOG IN
A0 A1 A2 A3 A4 A5

Code

Code Analysis

The operation of this project is straightforward: a touch detection on the sensor triggers the illumination of the next LED in the sequence (Red -> Yellow -> Green), controlled by the `currentLED` variable.

1. Define pins and initial values

```
const int touchSensorPin = 2;  // Touch sensor pin
const int rledPin = 7;         // Red LED pin
const int yledPin = 8;         // Yellow LED pin
const int gledPin = 9;         // Green LED pin
int lastTouchState;            // Previous touch sensor state
int currentTouchState;         // Current touch sensor state
int currentLED = 0;            // Current LED: 0->Red, 1->Yellow, 2->Green
```

   These lines establish the pin connections for the Arduino board components and initialize the touch sensor and LED states.

2. setup() function

```
void setup() {
  Serial.begin(9600);              // Initialize serial communication
  pinMode(touchSensorPin, INPUT);  // Set touch sensor pin as input
  // Configure LED pins as outputs
  pinMode(rledPin, OUTPUT);
  pinMode(yledPin, OUTPUT);
  pinMode(gledPin, OUTPUT);
  currentTouchState = digitalRead(touchSensorPin); // Read initial touch state
}
```

   This function configures the initial setup for the Arduino, defining input and output modes and starting serial communication for debugging.

3. loop() function

```
void loop() {
  lastTouchState = currentTouchState;                        // Store the last state
  currentTouchState = digitalRead(touchSensorPin);           // Read new touch state
  if (lastTouchState == LOW && currentTouchState == HIGH) {  // Detect touch
    Serial.println("Sensor touched");
    turnAllLEDsOff();  // Turn off all LEDs
    // Activate the next LED in sequence
    switch (currentLED) {
      case 0:
        digitalWrite(rledPin, HIGH);
        currentLED = 1;
        break;
      case 1:
        digitalWrite(yledPin, HIGH);
        currentLED = 2;
        break;
      case 2:
        digitalWrite(gledPin, HIGH);
```

(continues on next page)

(continued from previous page)

```
        currentLED = 0;
        break;
    }
  }
}
```

The loop continuously monitors the touch sensor, cycling through the LEDs when a touch is detected, ensuring only one LED is on at any given time.

4. Turn off LEDs function

```
void turnAllLEDsOff() {
  // Set all LED pins to LOW, turning them off
  digitalWrite(rledPin, LOW);
  digitalWrite(yledPin, LOW);
  digitalWrite(gledPin, LOW);
}
```

This auxiliary function turns off all LEDs, aiding in the cycling process.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.44 Lesson 43: Potentiometer scale value

This project focuses on reading a potentiometer's value and displaying it on an LCD 1620 equipped with an I2C interface. Additionally, the value is transmitted to the serial monitor for live monitoring. A distinctive aspect of this project is the graphical representation of the potentiometer's value on the LCD, which is depicted as a variable-length bar proportional to the potentiometer's reading.

Required Components

In this project, we need the following components.

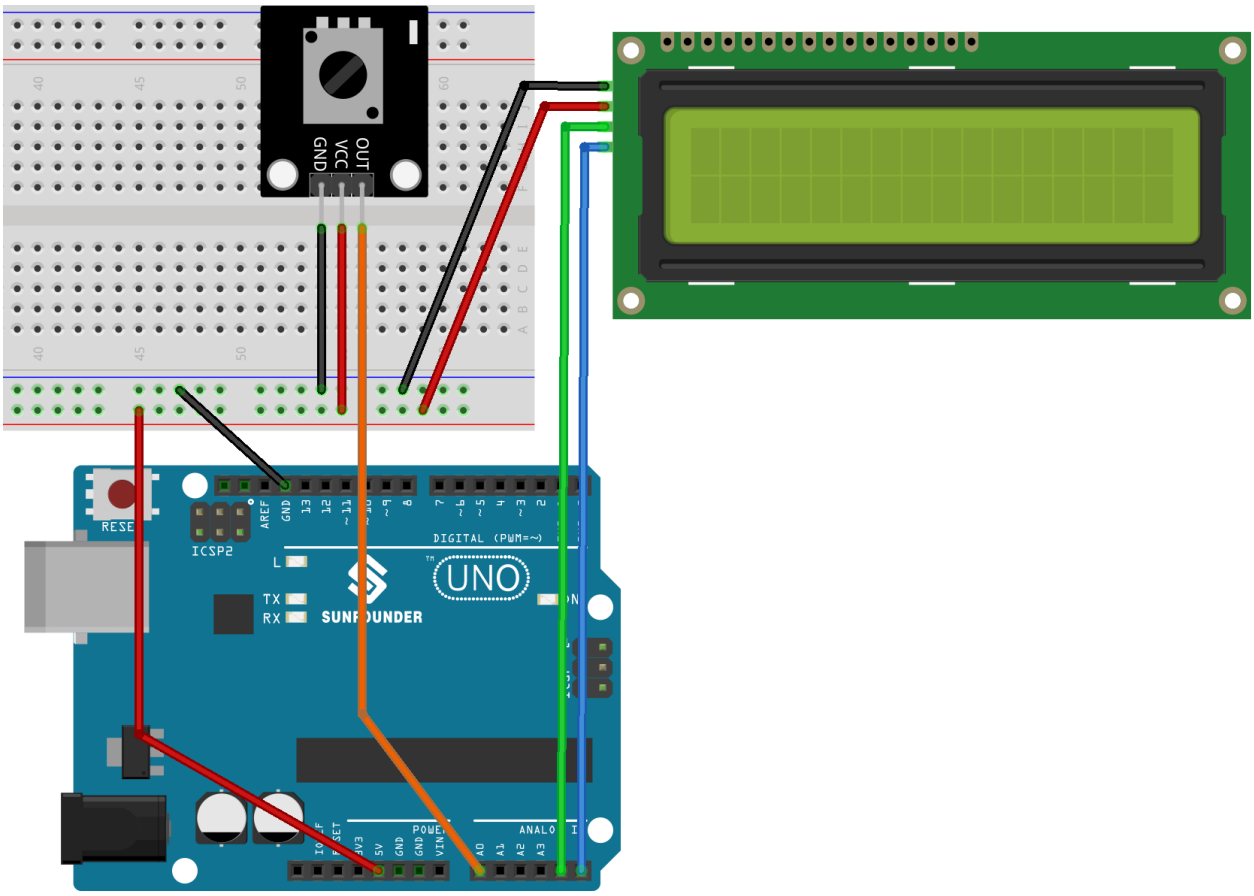
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Potentiometer Module	-
I2C LCD 1602	-
Breadboard	

Wiring



Code

Code Analysis

The core functionality of this project is to consistently read the potentiometer's value, map it to a scaled range (0-16), and display the result both numerically and graphically on the LCD. The implementation minimizes jitter by updating the display only when significant changes in the reading occur, thus maintaining a smooth visual experience.

1. **Library Inclusion and Initialization**:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
```

This segment incorporates the necessary libraries for I2C communication and LCD control. It then initializes an LCD instance with the I2C address of `0x27`, specifying its dimensions as `16 columns` and `2 rows`.

2. **Variable Declaration**:

```
int lastRead = 0;     // Stores the last read value from the potentiometer
int currentRead = 0;  // Holds the current read value from the potentiometer
```

Variables `lastRead` and `currentRead` are used to keep track of the potentiometer's readings across different moments.

3. **setup() Function**:

```
void setup() {
  lcd.init();          // Initiates the LCD
  lcd.backlight();     // Activates the LCD's backlight
  Serial.begin(9600);  // Commences serial communication at 9600 baud
}
```

This function prepares the LCD and starts serial communication, setting up the environment for the project's operation.

4. **Main Loop**:

```
void loop() {
  currentRead = analogRead(A0);
  int barLength = map(currentRead, 0, 1023, 0, 16);
  if (abs(lastRead - currentRead) > 2) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Value:");
    lcd.setCursor(7, 0);
    lcd.print(currentRead);
    Serial.println(currentRead);
    for (int i = 0; i < barLength; i++) {
      lcd.setCursor(i, 1);
      lcd.print(char(255));
    }
  }
  lastRead = currentRead;
  delay(200);
}
```

- Reads the potentiometer and converts its value to a scale suitable for visual representation.
- Updates the LCD only when a meaningful change is detected, displaying the numeric value and a corresponding bar graph.
- Also sends the reading to the serial monitor for external observation.
- Ensures stability and responsiveness by introducing a brief delay between iterations.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.45 Lesson 44: Digital Dice

This program simulates a dice roll using an OLED display. The simulation is triggered by shaking the vibration switch, causing the display to cycle through numbers 1 to 6, akin to rolling a dice. The display halts after a short duration, revealing a randomly selected number that represents the dice roll outcome.

Required Components

In this project, we need the following components.

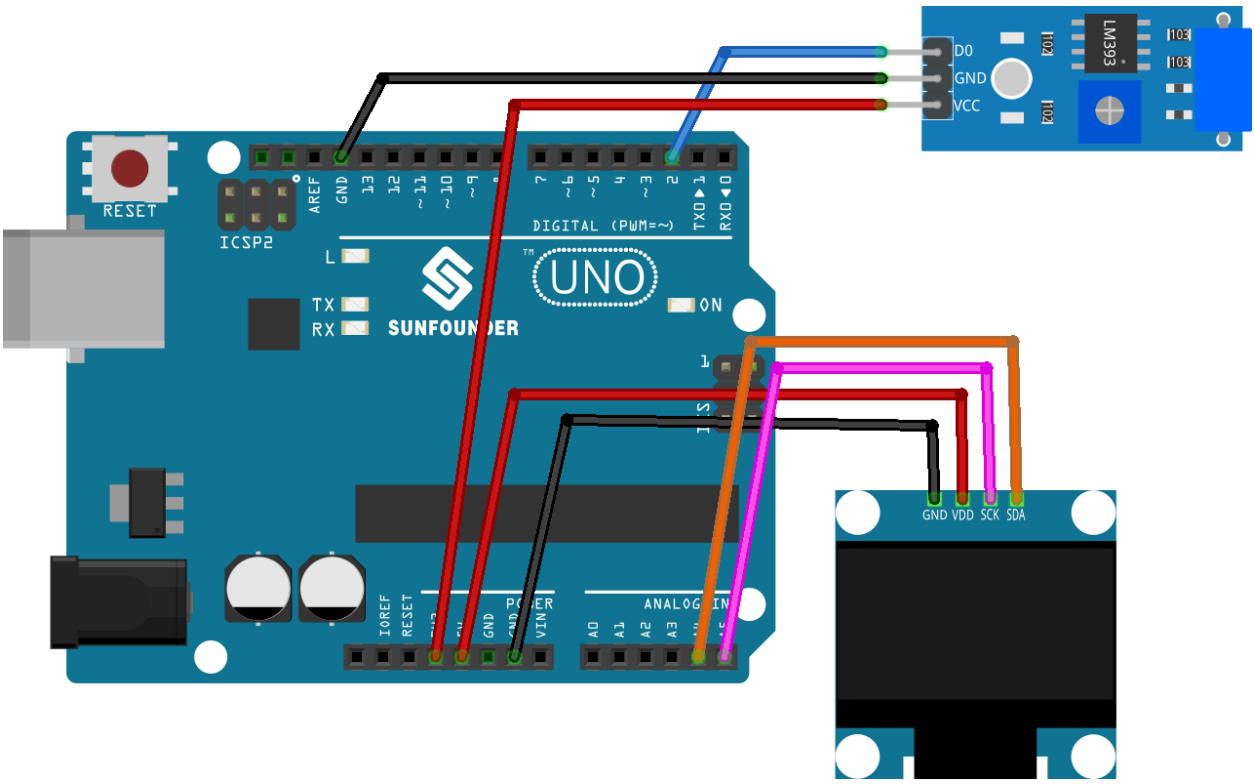
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Vibration Sensor Module (SW-420)	
OLED Display Module (SSD1306)	-
Breadboard	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"Adafruit SSD1306"** and **"Adafruit GFX"** and install it.

Code Analysis

A comprehensive breakdown of the code:

1. Initialization of variables:

   `vibPin`: Digital pin connected to the vibration sensor.

2. Volatile variables:

   `rolling`: A volatile flag that indicates the dice's rolling status. It is volatile as it is accessed within both the interrupt service routine and the main program.

3. `setup()`:

   Configures the vibration sensor's input mode. Assigns an interrupt to the sensor to trigger the rollDice function upon state change. Initializes the OLED display.

4. `loop()`:

   Continuously checks if `rolling` is true, displaying a random number between 1 and 6 during this state. The rolling ceases if the sensor has been shaken for over 500 milliseconds.

5. `rollDice()`:

   The interrupt service routine for the vibration sensor. It initiates the dice roll when the sensor is shaken by recording the current time.

6. `displayNumber()`:

   Displays a selected number on the OLED screen.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.46 Lesson 45: Plant Monitor

This project intelligently automates plant watering by triggering a water pump whenever the soil's moisture level dips below a predetermined threshold. It also features an LCD display that showcases the temperature, humidity, and soil moisture levels, offering users valuable insights into the plant's environmental conditions.

Required Components

In this project, we need the following components.

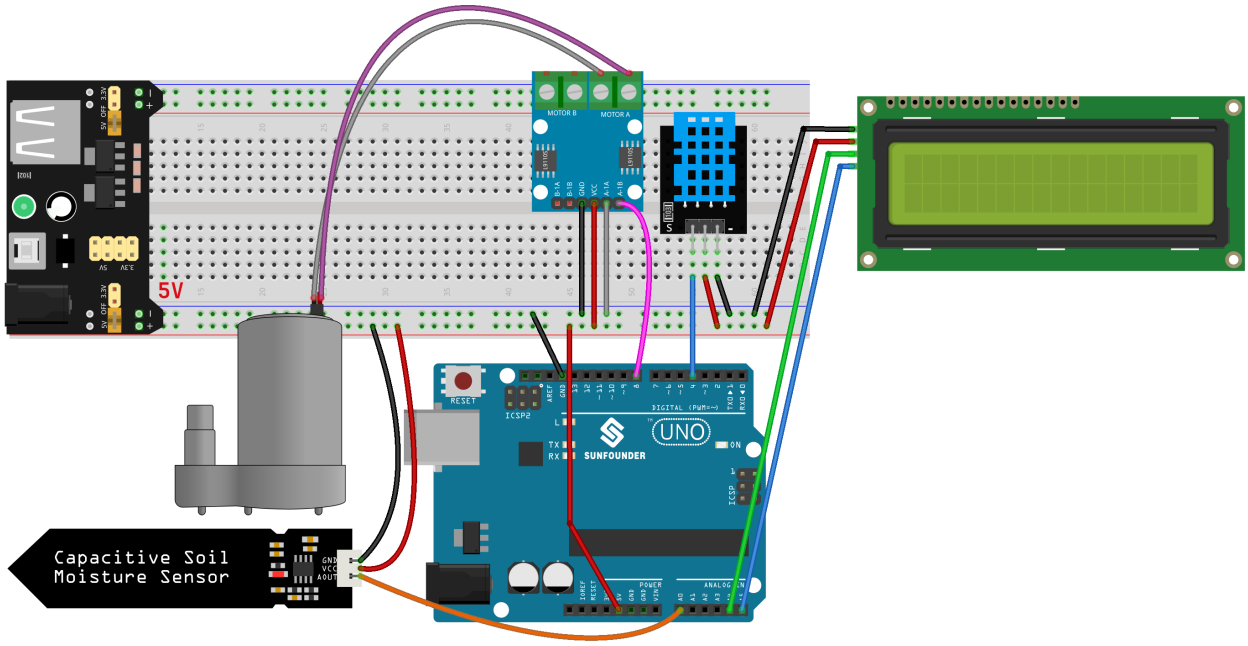
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Breadboard	
Power Supply Module	-
I2C LCD 1602	
Centrifugal Pump	-
L9110 Motor Driver Module	-
Capacitive Soil Moisture Module	
Temperature and Humidity Sensor Module (DHT11)	-

Wiring



Code

Code Analysis

The code is structured to seamlessly manage plant watering by monitoring environmental parameters:

1. Library Inclusions and Constants/Variables:

   Incorporate `Wire.h`, `LiquidCrystal_I2C.h`, and `DHT.h` libraries for functionality. Specify pin assignments and settings for the DHT11 sensor, soil moisture sensor, and water pump.

2. `setup()`:

   Configure pin modes for the moisture sensor and pump. Initially deactivate the pump. Initialize and backlight the LCD. Activate the DHT sensor.

3. `loop()`:

   Measure humidity and temperature via the DHT sensor. Gauge soil moisture through the soil moisture sensor. Display the temperature and humidity on the LCD, then show soil moisture levels. Assess soil moisture to decide on water pump activation; if soil moisture is under 500 (adjustable threshold), run the pump for 1 second.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.47 Lesson 46: Bluetooth LCD

This project enables the receipt of messages via a Bluetooth module connected to an Arduino UNO board and displays these messages on an LCD screen.

Required Components

In this project, we need the following components.

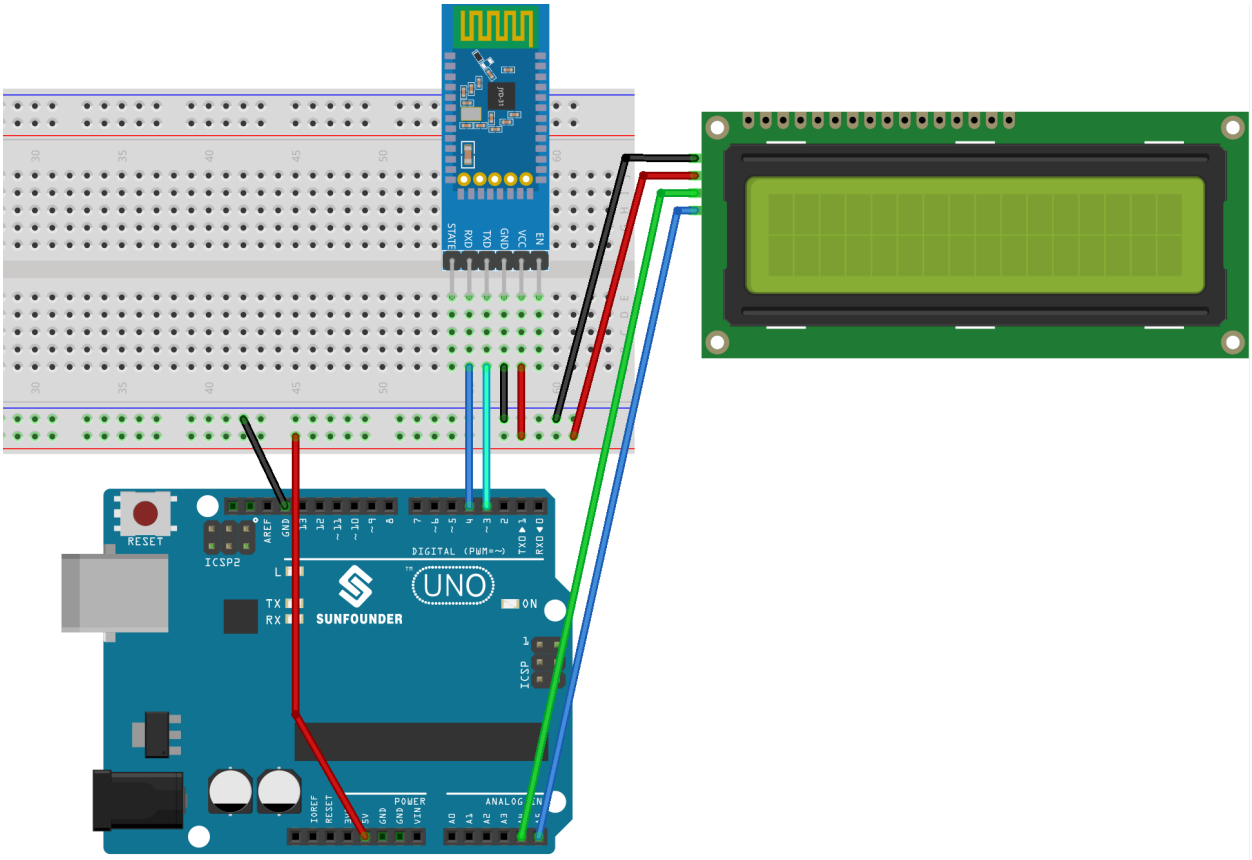
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Breadboard	
I2C LCD 1602	
JDY-31 Bluetooth Module	-

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"LiquidCrystal I2C"** and install it.

App and Bluetooth module Connection

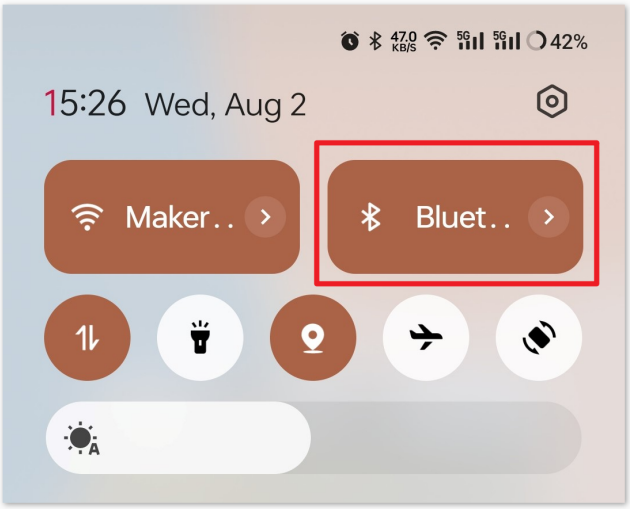
We can use an app called "Serial Bluetooth Terminal" to send messages from the Bluetooth module to Arduino.

a. **Install Serial Bluetooth Terminal**

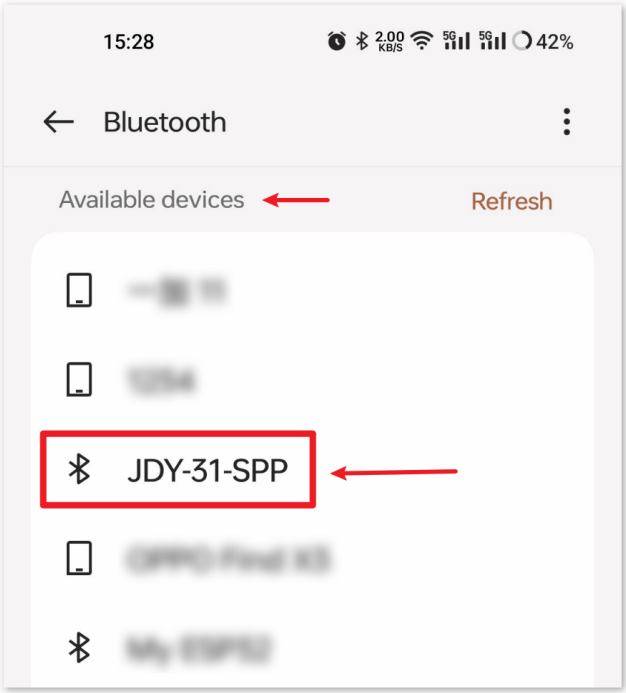
   Go to Google Play to download and install .

b. **Connect Bluetooth**

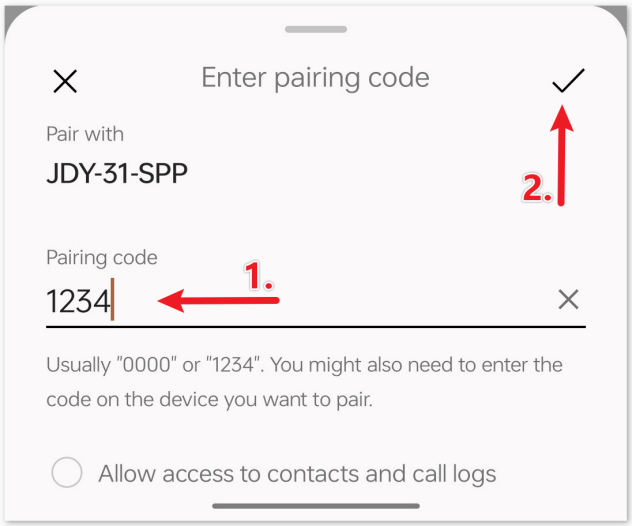
   Initially, turn on **Bluetooth** on your smartphone.



Navigate to the **Bluetooth settings** on your smartphone and look for names like **JDY-31-SPP**.

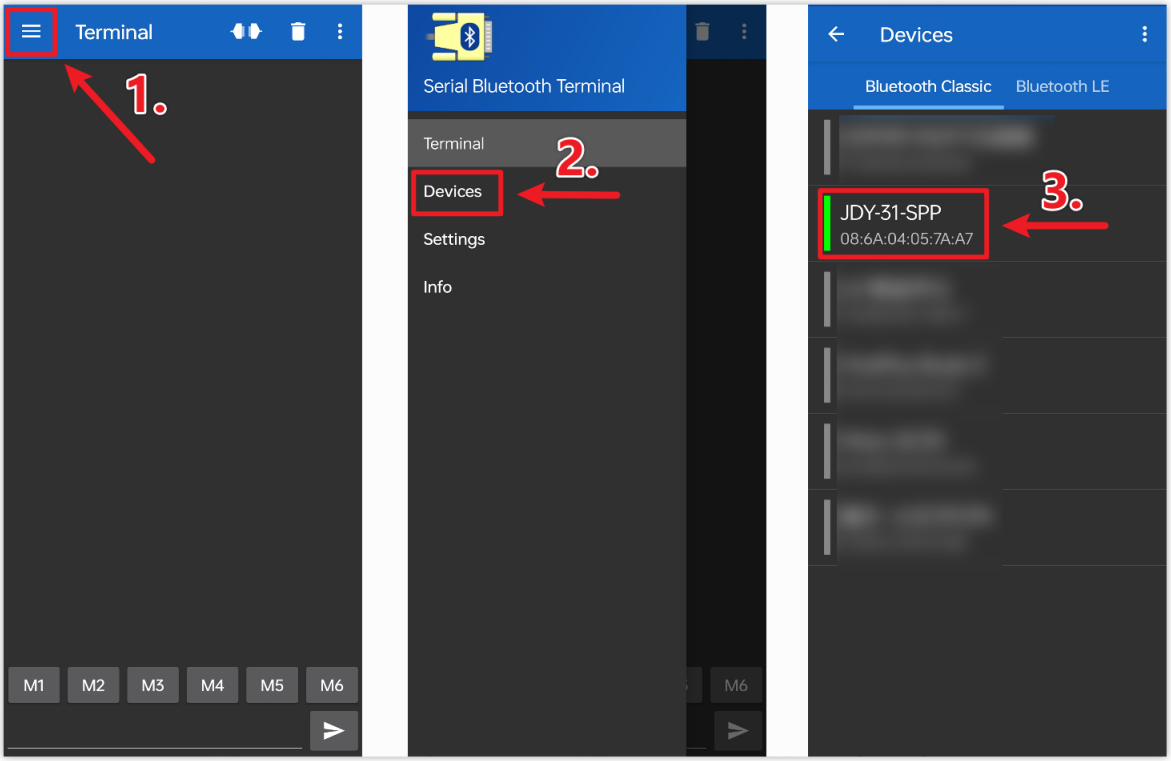


After clicking it, agree to the **Pair** request in the pop-up window. If prompted for a pairing code, please enter "1234".



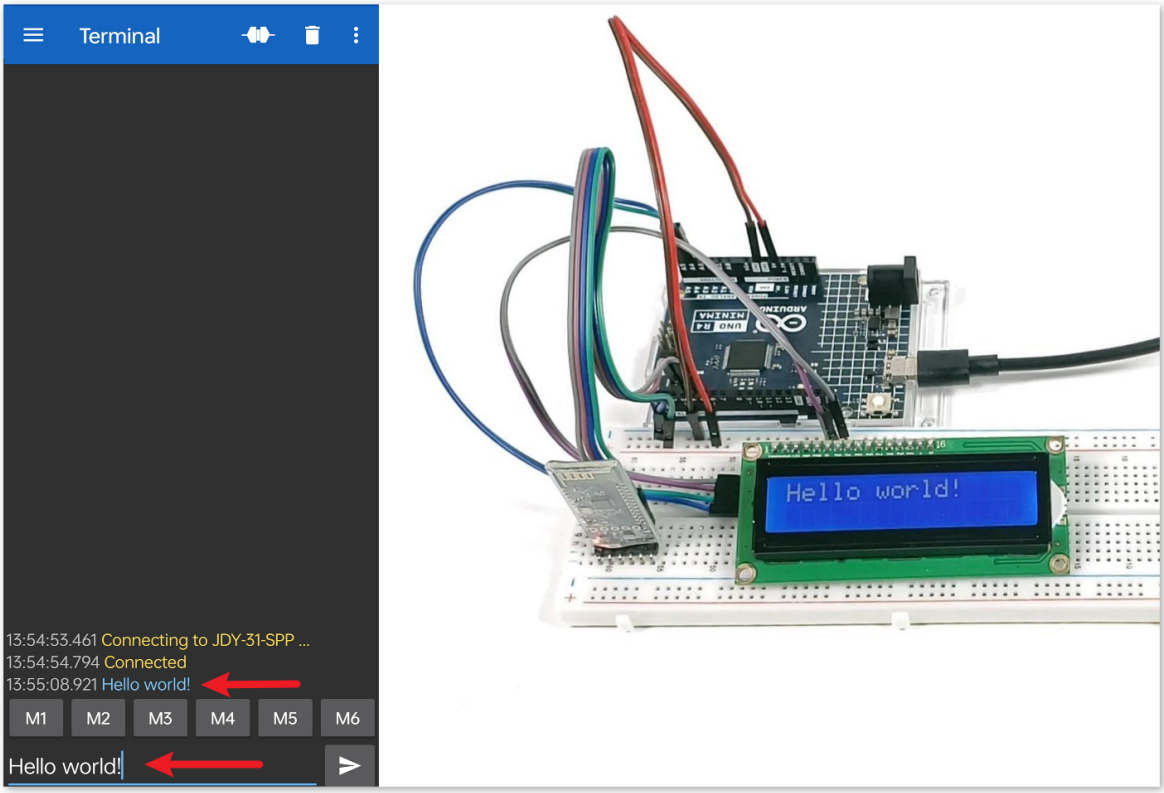
c. **Communicate with Bluetooth module**

Open the Serial Bluetooth Terminal. Connect to "JDY-31-SPP".



d. **Send command**

Use the Serial Bluetooth Terminal app to send messages to Arduino via Bluetooth. The message sent to Bluetooth will be displayed on the LCD.



Code Analysis

Note: To install library, use the Arduino Library Manager and search for **"LiquidCrystal I2C"** and install the library.

1. Setting up the LCD

```
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
```

This segment of code includes the LiquidCrystal_I2C library and initializes the LCD module with the I2C address as `0x27` and specifies that the LCD has `16` columns and `2` rows.

2. Setting up Bluetooth communication

```
#include <SoftwareSerial.h>
const int bluetoothTx = 3;
const int bluetoothRx = 4;
SoftwareSerial bleSerial(bluetoothTx, bluetoothRx);
```

Here, the SoftwareSerial library is included to allow the JDY-31 Bluetooth module to communicate with the Arduino using pins 3 (TX) and 4 (RX).

3. Initialization

```
void setup() {
   lcd.init();
```

(continues on next page)

(continued from previous page)

```
    lcd.clear();
    lcd.backlight();

    Serial.begin(9600);
    bleSerial.begin(9600);
}
```

The `setup()` function initializes the LCD and clears any existing content. It also turns on the backlight for the LCD. Communication is started with the serial monitor and the Bluetooth module, both at a baud rate of `9600`.

4. Main Loop

```
void loop() {
    String data;

    if (bleSerial.available()) {
        data += bleSerial.readString();
        data = data.substring(0, data.length() - 2);
        Serial.print(data);

        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print(data);
    }

    if (Serial.available()) {
        bleSerial.write(Serial.read());
    }
}
```

This is the main operational loop of the Arduino program. It continually checks for incoming data from both the Bluetooth module and the serial monitor. When data is received from the Bluetooth device, it's processed, displayed on the serial monitor, and shown on the LCD. If data is entered into the serial monitor, this data is sent to the Bluetooth module.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.48 Lesson 47: Bluetooth Traffic Light

This project is designed to control a traffic light (Red, Yellow, Green LEDs) using Bluetooth communication. The user can send a character ('R', 'Y', or 'G') from a Bluetooth device. When the Arduino receives one of these characters, it lights up the corresponding LED, while ensuring the other two LEDs are turned off.

Required Components

In this project, we need the following components.

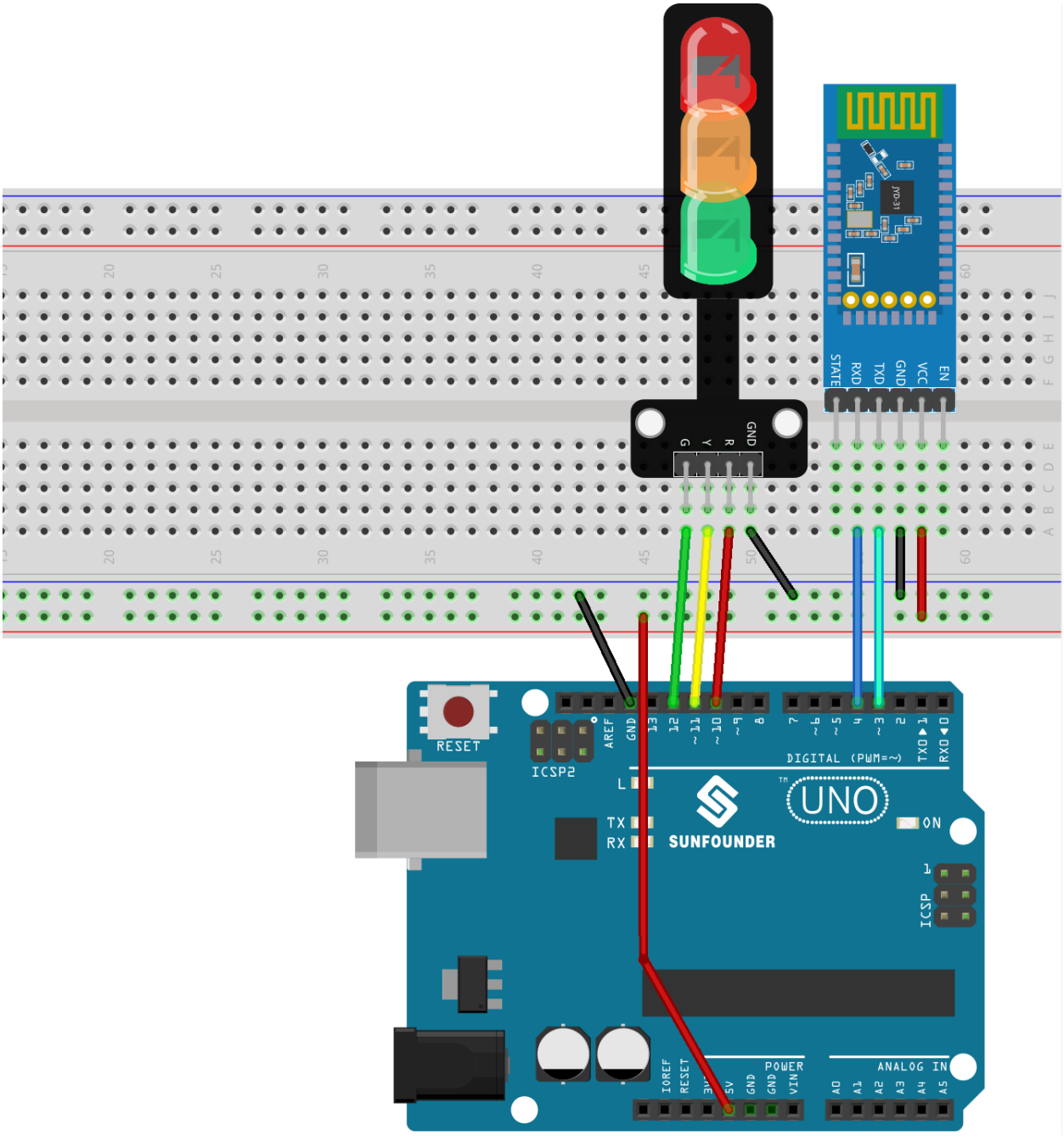
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Breadboard	
Traffic Light Module	-
JDY-31 Bluetooth Module	-

Wiring


GND
G
Y
R
STATE
RXD
TXD
GND
VCC
EN
RESET
ICSP2
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM=~)
L
TX
RX
SUNFOUNDER
UNO
ON
ICSP
POWER
ANALOG IN
IOREF
RESET
3V3
5V
GND
GND
VIN
A0
A1
A2
A3
A4
A5

Code

App and Bluetooth module Connection

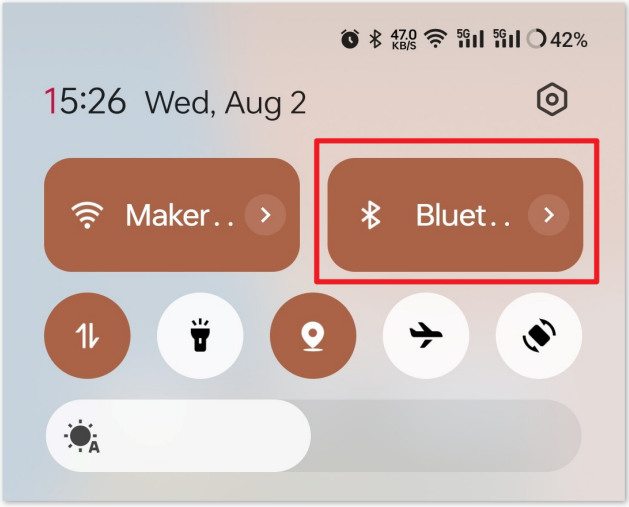
We can use an app called “Serial Bluetooth Terminal” to send messages from the Bluetooth module to Arduino.

a. **Install Serial Bluetooth Terminal**

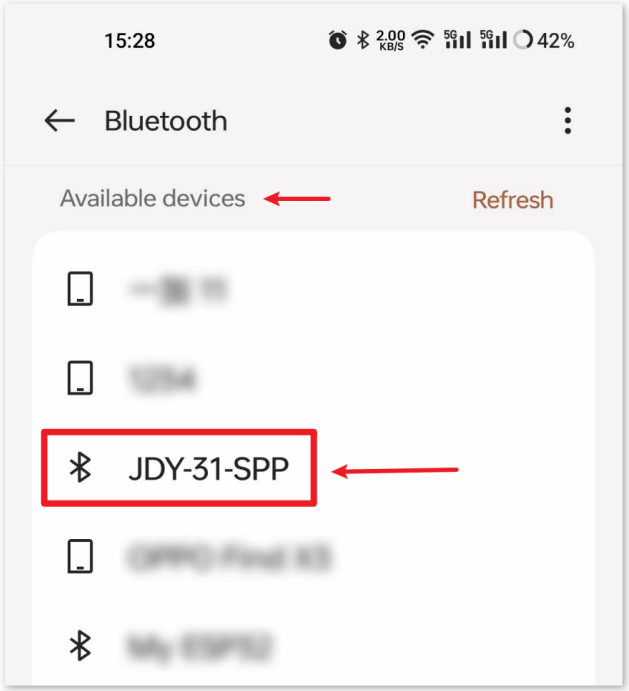
   Go to Google Play to download and install .

b. **Connect Bluetooth**

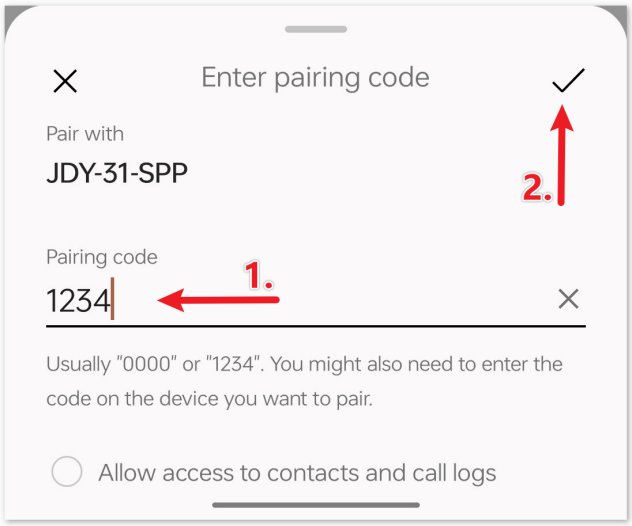
   Initially, turn on **Bluetooth** on your smartphone.



Navigate to the **Bluetooth settings** on your smartphone and look for names like **JDY-31-SPP**.

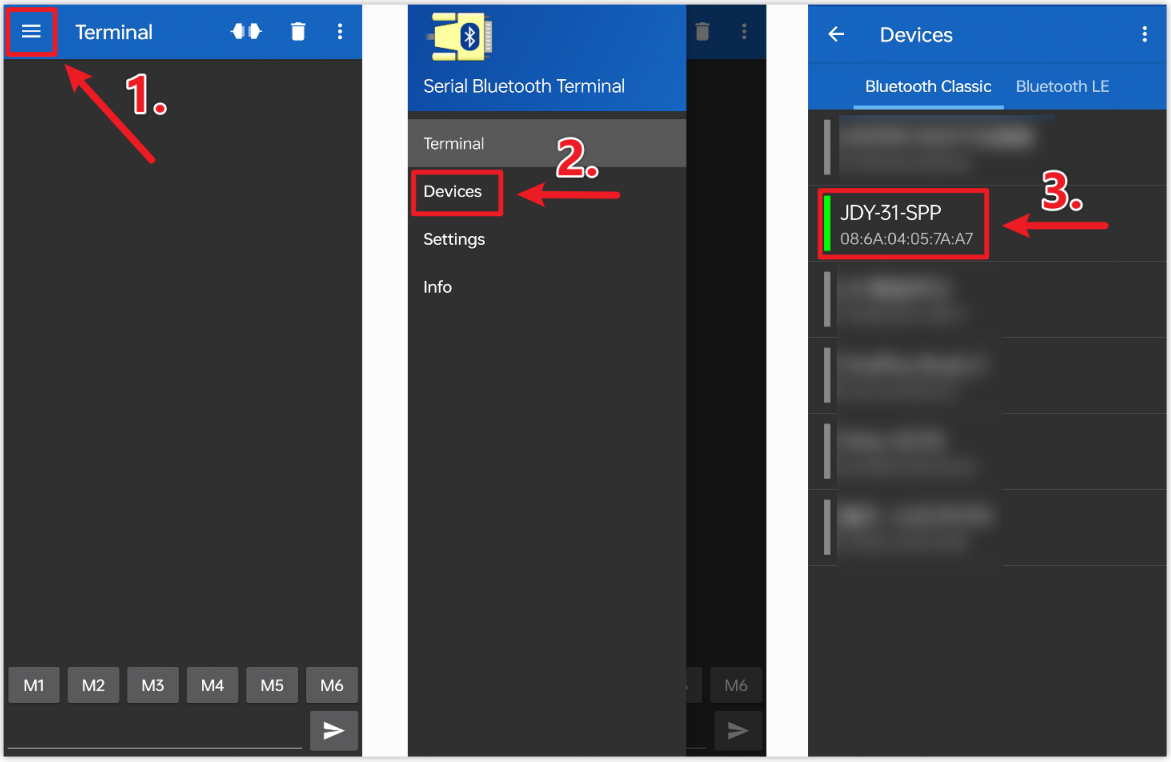


After clicking it, agree to the **Pair** request in the pop-up window. If prompted for a pairing code, please enter "1234".



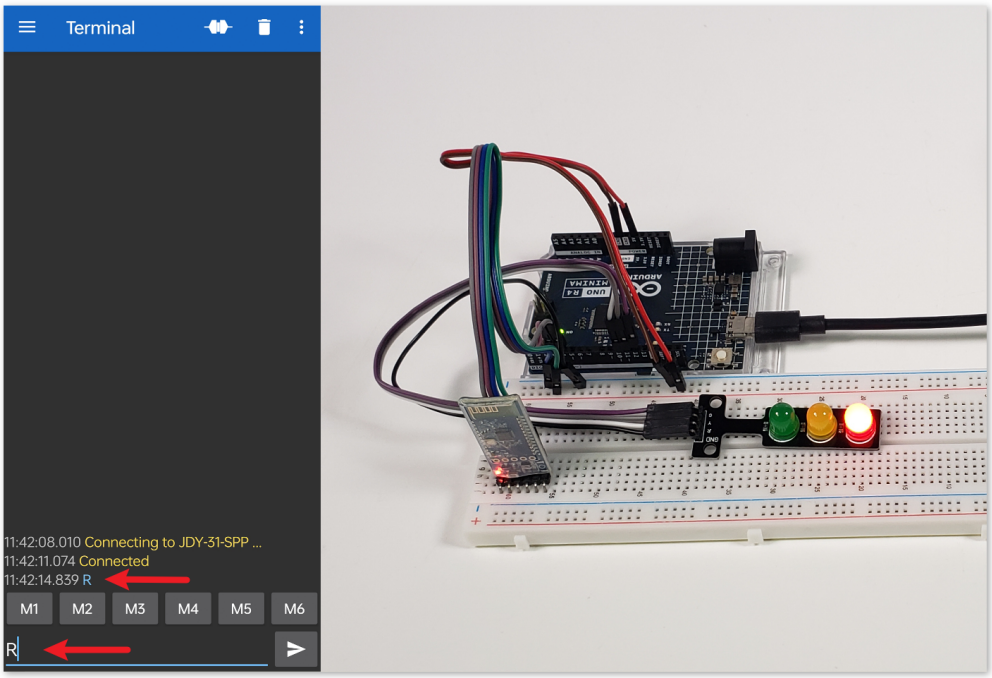
c. **Communicate with Bluetooth module**

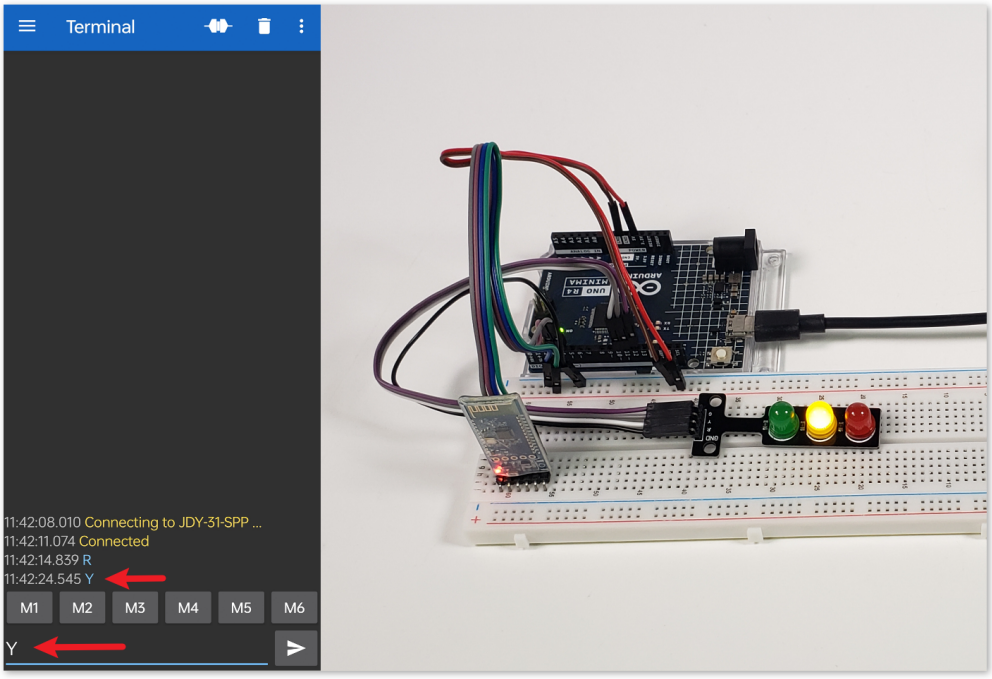
Open the Serial Bluetooth Terminal. Connect to "JDY-31-SPP".

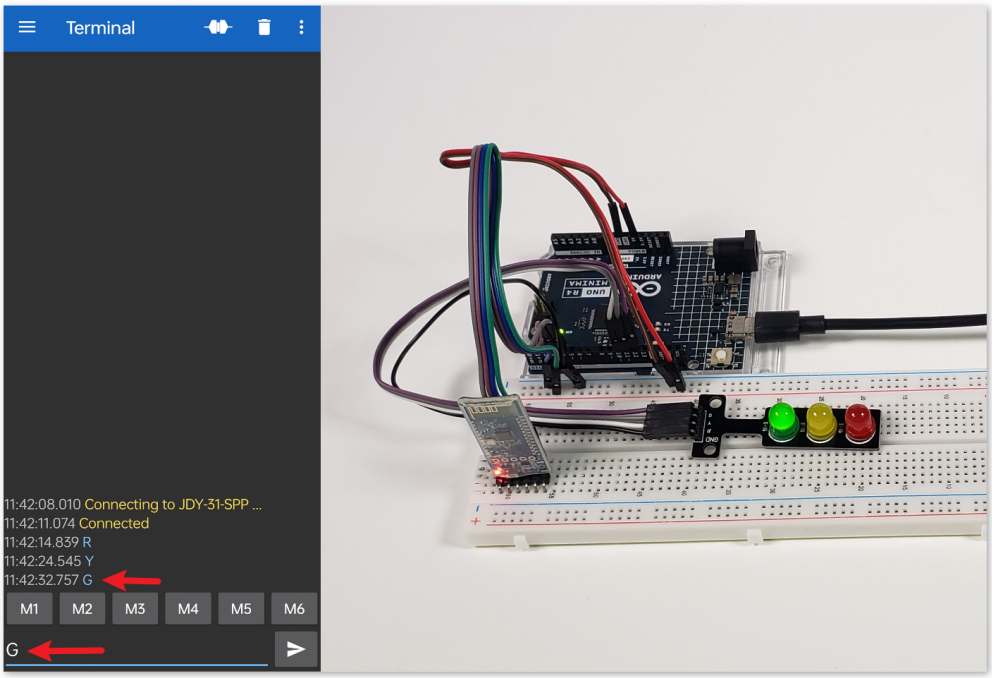


d. **Send command**

Use the Serial Bluetooth Terminal app to send commands to Arduino via Bluetooth. Send R to turn on the red light, Y for yellow, and G for green.


Terminal
11:42:08.010 Connecting to JDY-31-SPP ...
11:42:11.074 Connected
11:42:14.839 R
M1
M2
M3
M4
M5
M6
R


Terminal
11:42:08.010 Connecting to JDY-31-SPP ...
11:42:11.074 Connected
11:42:14.839 R
11:42:24.545 Y
M1
M2
M3
M4
M5
M6
Y



Code Analysis

1. Initialization and Bluetooth setup

```
// Set up Bluetooth module communication
#include <SoftwareSerial.h>
const int bluetoothTx = 3;
const int bluetoothRx = 4;
SoftwareSerial bleSerial(bluetoothTx, bluetoothRx);
```

We begin by including the SoftwareSerial library to help us with Bluetooth communication. The Bluetooth module's TX and RX pins are then defined and associated with pins 3 and 4 on the Arduino. Finally, we initialize the `bleSerial` object for Bluetooth communication.

2. LED Pin Definitions

```
// Pin numbers for each LED
const int rledPin = 10;  //red
const int yledPin = 11;  //yellow
const int gledPin = 12;  //green
```

Here, we're defining which Arduino pins our LEDs are connected to. The red LED is on pin 10, yellow on 11, and green on 12.

3. setup() Function

```
void setup() {
   pinMode(rledPin, OUTPUT);
   pinMode(yledPin, OUTPUT);
   pinMode(gledPin, OUTPUT);
```

(continues on next page)

(continued from previous page)

```
    Serial.begin(9600);
    bleSerial.begin(9600);
}
```

In the `setup()` function, we set the LED pins as `OUTPUT`. We also start serial communication for both the Bluetooth module and the default serial (connected to the computer) at a baud rate of 9600.

4. Main loop() for Bluetooth Communication

```
void loop() {
    while (bleSerial.available() > 0) {
        character = bleSerial.read();
        Serial.println(character);

        if (character == 'R') {
            toggleLights(rledPin);
        } else if (character == 'Y') {
            toggleLights(yledPin);
        } else if (character == 'G') {
            toggleLights(gledPin);
        }
    }
}
```

Inside our main `loop()`, we continuously check if data is available from the Bluetooth module. If we receive data, we read the character and display it in the serial monitor. Depending on the character received (R, Y, or G), we toggle the respective LED using the `toggleLights()` function.

5. Toggle Lights Function

```
void toggleLights(int targetLight) {
    digitalWrite(rledPin, LOW);
    digitalWrite(yledPin, LOW);
    digitalWrite(gledPin, LOW);

    digitalWrite(targetLight, HIGH);
}
```

This function, `toggleLights()`, turns off all the LEDs first. After ensuring they are all off, it turns on the specified target LED. This ensures that only one LED is on at a time.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.49 Lesson 52: Tilt Direction Indicator

This Arduino project uses an MPU6050 accelerometer and gyroscope sensor along with an OLED display. The project reads data from the MPU6050 sensor to detect the tilt direction and displays corresponding arrows (up, down, left, or right) or a circle (if there is no significant tilt) on the OLED screen based on the tilt direction.

Required Components

In this project, we need the following components.

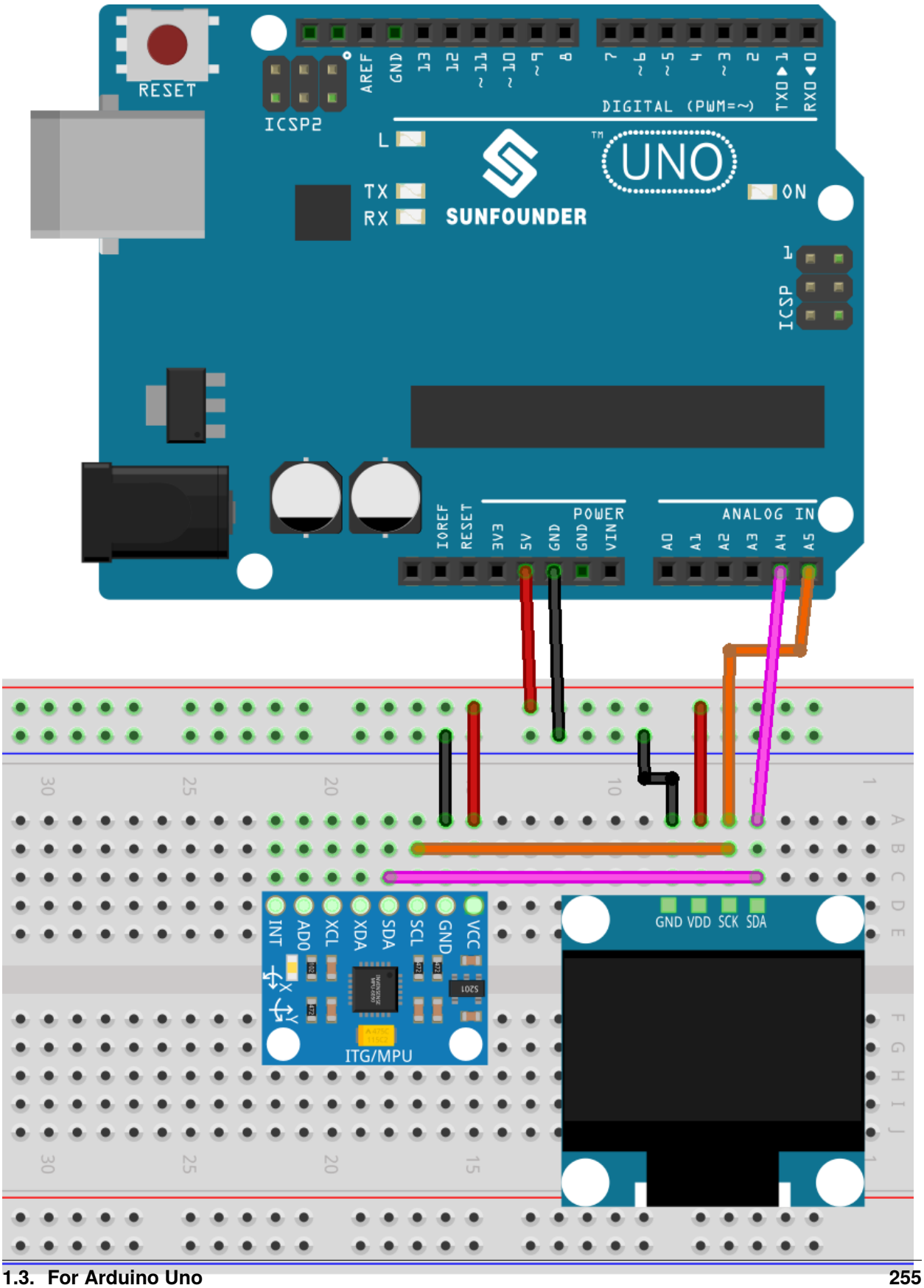
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Gyroscope & Accelerometer Module (MPU6050)	
OLED Display Module (SSD1306)	-
Breadboard	

Wiring


RESET
ICSP2
AREF
GND
13
12
11
10
9
8
7
6
5
4
3
2
1 TX0
0 RX0
DIGITAL (PWM=~)
L
TX
RX
SUNFOUNDER
UNO
ON
ICSP
IOREF
RESET
3V3
5V
GND
GND
VIN
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
INT
AD0
XCL
XDA
SDA
SCL
GND
VCC
ITG/MPU
GND VDD SCK SDA

Code

Note: To install the library, use the Arduino Library Manager and search for **"Adafruit SSD1306"** and **"Adafruit GFX"** and install it.

Code Analysis

1. Library inclusion and OLED display setup

   The project starts by including the necessary libraries to interface with the MPU6050 sensor and OLED display. The OLED display dimensions and I2C address are defined, followed by the creation of the display object.

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

Adafruit_MPU6050 mpu;
```

2. Setup function

   In the setup function, the serial communication is initialized, and the MPU6050 sensor is initialized with specific settings for accelerometer and gyroscope ranges. The OLED display is also initialized and cleared.

```
void setup(void) {
  Serial.begin(115200);

  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }

  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);

  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;)
      ;  // Don't proceed, loop forever
  }
  display.clearDisplay();
```

(continues on next page)

(continued from previous page)

```
  delay(100);
}
```

3. Loop function

   In the loop function, sensor data is continuously read, and the tilt direction is determined based on acceleration values. Depending on the tilt direction, different arrows or a circle are drawn on the OLED display.

   The code reads data from the MPU6050 sensor to detect the tilt direction and displays corresponding arrows (up, down, left, or right) or a circle (if there is no significant tilt) on the OLED screen based on the tilt direction.

```
void loop() {

  display.clearDisplay();

  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  Serial.print("acceleration:");
  Serial.print(a.acceleration.x);
  Serial.print(",");
  Serial.print(a.acceleration.y);
  Serial.print(",");
  Serial.println(a.acceleration.z);

  if (a.acceleration.x >= 2) {
    drawUpArrow();
  } else if (a.acceleration.x <= -2) {
    drawDownArrow();
  } else if (a.acceleration.y >= 2) {
    drawLeftArrow();
  } else if (a.acceleration.y <= -2) {
    drawRightArrow();
  } else {
    drawCircle();
  }
  display.display();

  delay(200);
}
```

4. Drawing functions

   Several helper functions are defined to draw different shapes on the OLED display. These functions use the `Adafruit_GFX` library to draw arrows and circles.

```
void drawUpArrow() {
  display.fillTriangle(49, 30, 64, 15, 79, 30, WHITE);
  display.fillRect(59, 30, 10, 20, WHITE);
}

void drawDownArrow() {
  display.fillTriangle(49, 36, 64, 51, 79, 36, WHITE);
```

(continues on next page)

(continued from previous page)

```
  display.fillRect(59, 16, 10, 20, WHITE);
}

void drawRightArrow() {
  display.fillTriangle(70, 15, 85, 30, 70, 45, WHITE);
  display.fillRect(50, 25, 20, 10, WHITE);
}

void drawLeftArrow() {
  display.fillTriangle(60, 15, 45, 30, 60, 45, WHITE);
  display.fillRect(60, 25, 20, 10, WHITE);
}

void drawCircle() {
  display.fillCircle(64, 32, 10, WHITE);
  display.fillCircle(64, 32, 8, BLACK);
}
```

Reference

-

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.50 Lesson 53: Direction Indicator

This Arduino project initializes an OLED display and reads input from a joystick connected to analog pins A0 and A1. It continuously monitors the joystick's position to determine its tilt direction and displays an appropriate arrow (up, down, left, or right) or a circle (if the joystick is centered) on the OLED display.

Required Components

In this project, we need the following components.

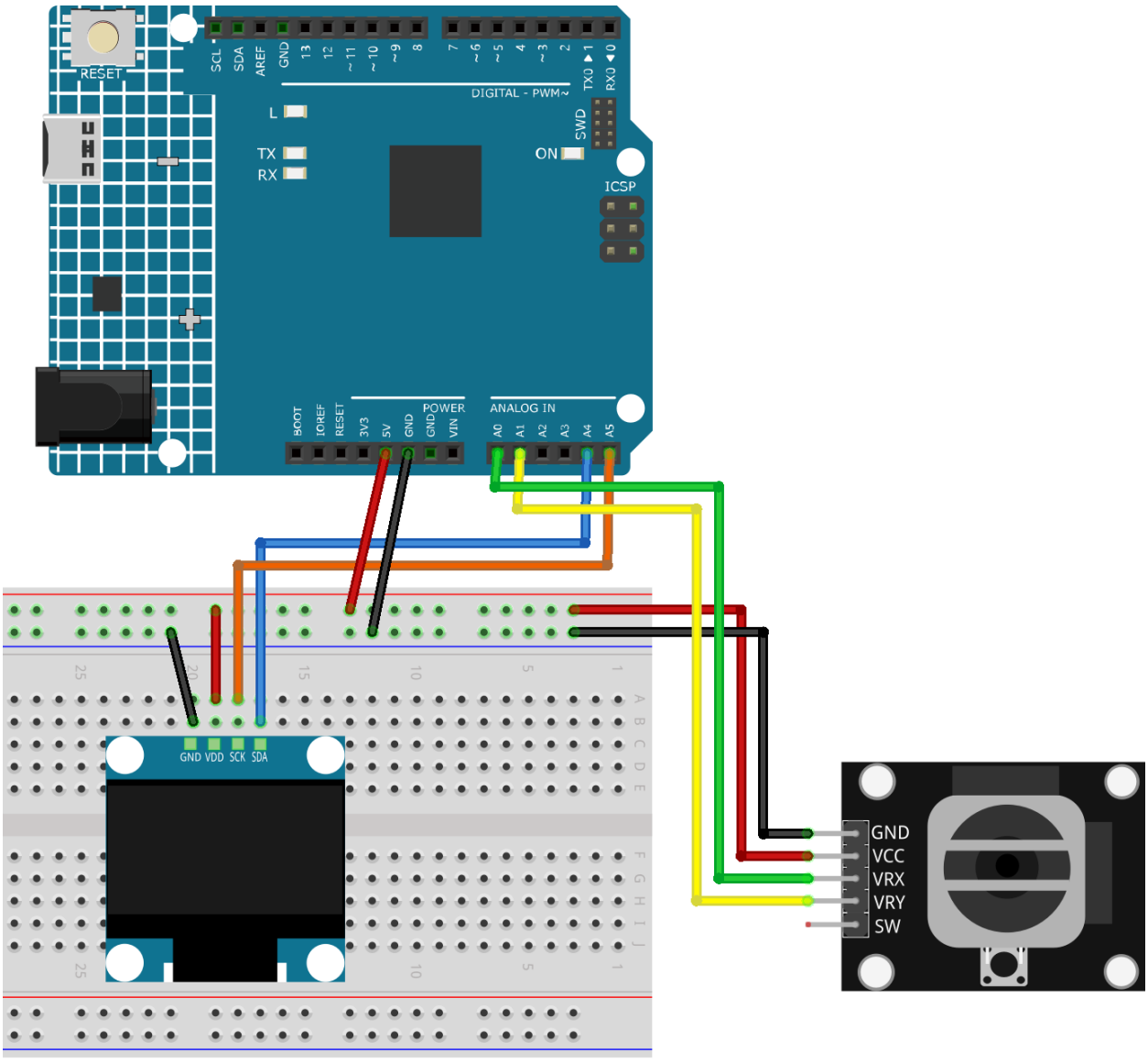
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Joystick Module	
OLED Display Module (SSD1306)	-
Breadboard	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"Adafruit SSD1306"** and **"Adafruit GFX"** and install it.

Code Analysis

1. Including necessary libraries

   The project uses three libraries: `Wire.h` for I2C communication, `Adafruit_GFX.h` for graphics primitives, and `Adafruit_SSD1306.h` for OLED display control.

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
```

2. Defining constants and creating an OLED display object

   Constants for the OLED display dimensions and address are defined. The OLED display object is created with these parameters.

```
#define SCREEN_WIDTH 128  // OLED display width, in pixels
#define SCREEN_HEIGHT 64  // OLED display height, in pixels
#define OLED_RESET -1  // Reset pin # (or -1 if sharing Arduino reset pin)
#define SCREEN_ADDRESS 0x3C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
```

3. Pin definitions and threshold for the joystick

   The analog pins A0 and A1 are used for the joystick, and a threshold is defined to determine if the joystick is centered.

```
const int xPin = A0;  // the VRX attach to
const int yPin = A1;  // the VRY attach to
const int threshold = 50;  // threshold to consider joystick in the center
```

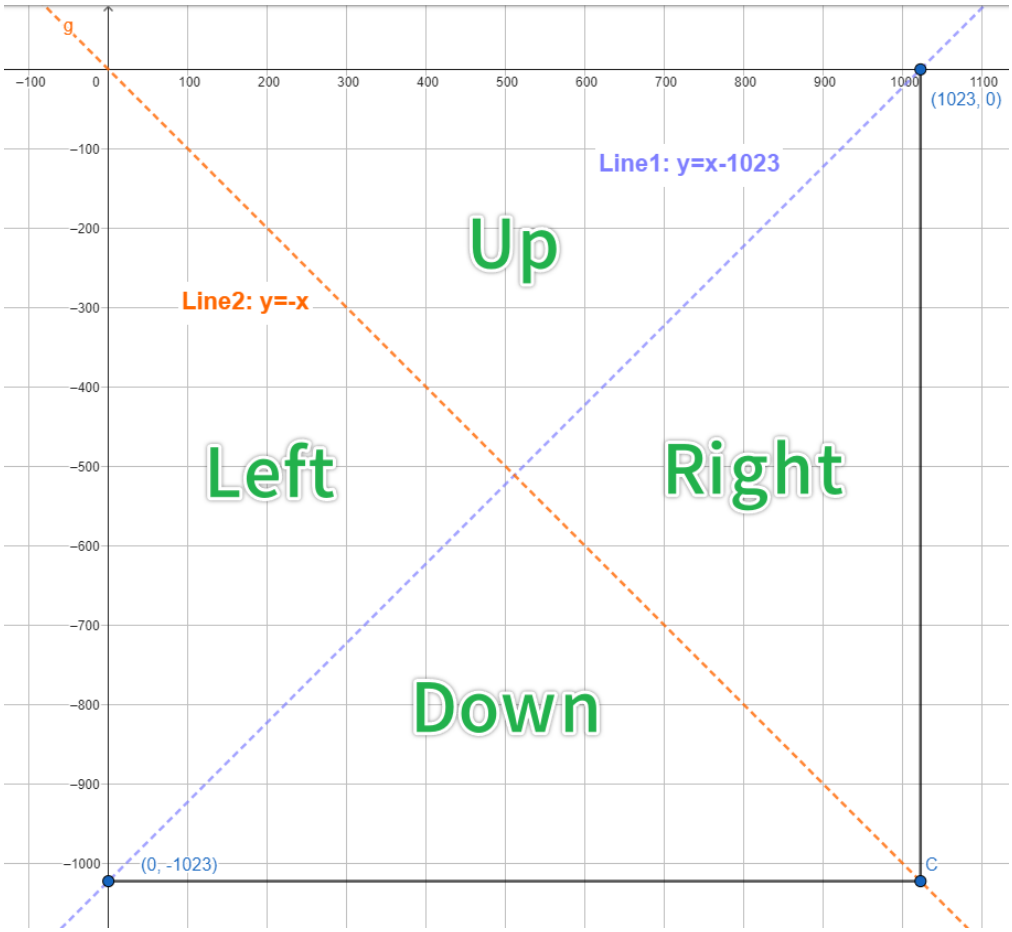
4. Setup function: initializing serial communication and the OLED display

   Serial communication is initialized for debugging, and the OLED display is initialized and cleared.

```
void setup() {
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;);
  }
  display.clearDisplay();
}
```

5. Main loop: reading joystick values, determining direction, and displaying shapes

   The main loop reads the joystick values, determines the direction based on these values, and displays the corresponding shape on the OLED display.




```
void loop() {
  display.clearDisplay();
  int xValue = analogRead(xPin);
  int yValue = analogRead(yPin) * -1;
  Serial.print("X: ");
  Serial.print(xValue);
  Serial.print("|Y: ");
  Serial.println(-yValue);

  float yLine1 = line1(xValue);
  float yLine2 = line2(xValue);

  int relX = xValue - 512;
  int relY = -yValue - 512;

  if (abs(relX) < threshold && abs(relY) < threshold) {
    drawCircle();
  } else if (yValue > yLine1 && yValue > yLine2) {
    drawUpArrow();
  } else if (yValue < yLine1 && yValue < yLine2) {
    drawDownArrow();
  } else if (yValue < yLine1 && yValue > yLine2) {
    drawRightArrow();
```

(continues on next page)

(continued from previous page)

```
  } else if (yValue > yLine1 && yValue < yLine2) {
    drawLeftArrow();
  }

  display.display();
  delay(80);
}
```

6. Helper functions: calculating lines and drawing shapes

   These functions help in calculating lines used for direction determination and drawing shapes on the OLED display.

```
float line1(float x) {
  return x - 1023;
}

float line2(float x) {
  return -x;
}

void drawUpArrow() {
  display.fillTriangle(49, 30, 64, 15, 79, 30, WHITE);
  display.fillRect(59, 30, 10, 20, WHITE);
}

void drawDownArrow() {
  display.fillTriangle(49, 36, 64, 51, 79, 36, WHITE);
  display.fillRect(59, 16, 10, 20, WHITE);
}

void drawRightArrow() {
  display.fillTriangle(70, 15, 85, 30, 70, 45, WHITE);
  display.fillRect(50, 25, 20, 10, WHITE);
}

void drawLeftArrow() {
  display.fillTriangle(60, 15, 45, 30, 60, 45, WHITE);
  display.fillRect(60, 25, 20, 10, WHITE);
}

void drawCircle() {
  display.fillCircle(64, 32, 10, WHITE);
  display.fillCircle(64, 32, 8, BLACK);
}
```

Reference

-

IoT Config

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

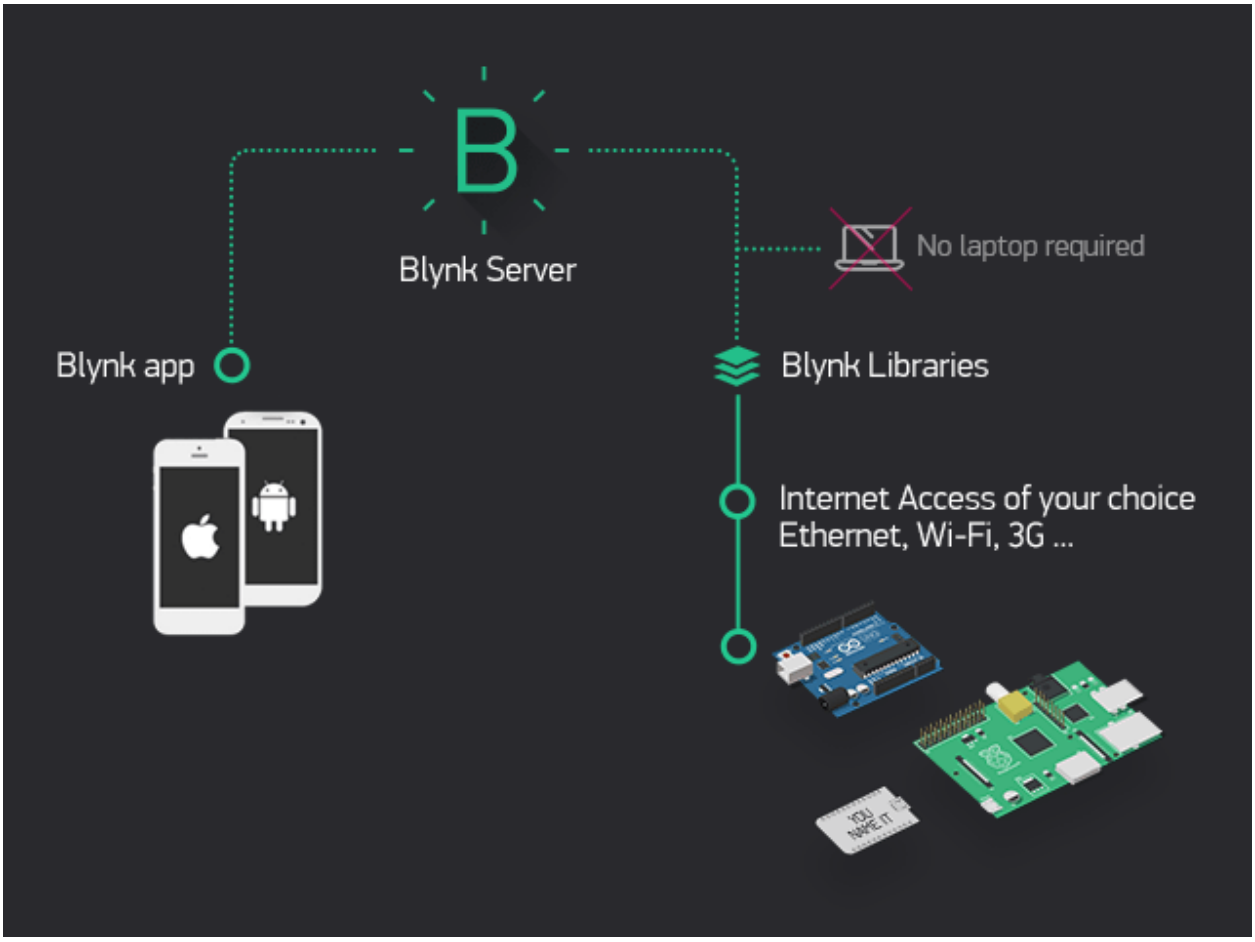
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.51 Get Started with Blynk

Blynk is a full suite of software required to prototype, deploy, and remotely manage connected electronic devices at any scale: from personal IoT projects to millions of commercial connected products. With Blynk anyone can connect their hardware to the cloud and build a no-code iOS, Android, and web applications to analyze real-time and historical data coming from devices, control them remotely from anywhere in the world, receive important notifications, and much more.



Getting the R4 board to communicate with Blynk requires some configuration when you first use Blynk.

Follow the steps below, and note that you must do them in order and not skip any chapters.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

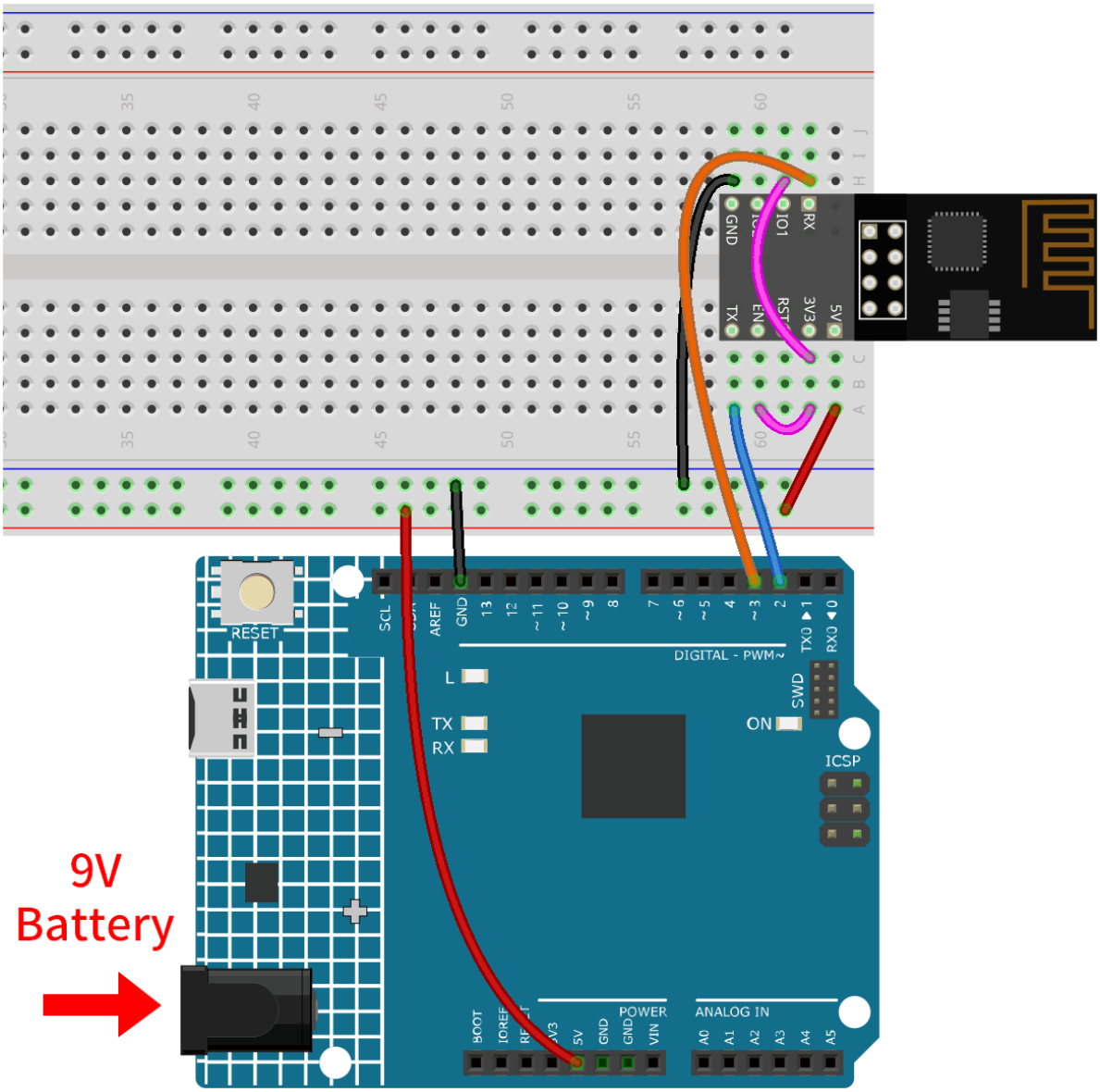
- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.1 Configuring the ESP8266

The ESP8266 module that comes with the kit is already pre-burned with AT firmware, but you still need to modify its configuration by following the steps below.

1. Build the circuit.



2. Open the `00-Set_software_serial.ino` file under the path of `ultimate-sensor-kit\iot_project\wifi\00-Set_software_serial`. Or copy this code into Arduino IDE. And upload the code.

   The code establishes a software serial communication using Arduino's SoftwareSerial library, allowing the Arduino to communicate with the ESP8266 module through its digital pins 2 and 3 (as Rx and Tx). It checks for data transfer between them, forwarding received messages from one to the other at a baud rate of 115200. **With this code, you can use the Arduino's serial monitor to send AT firmware commands to the ESP8266 module and receive its responses.**

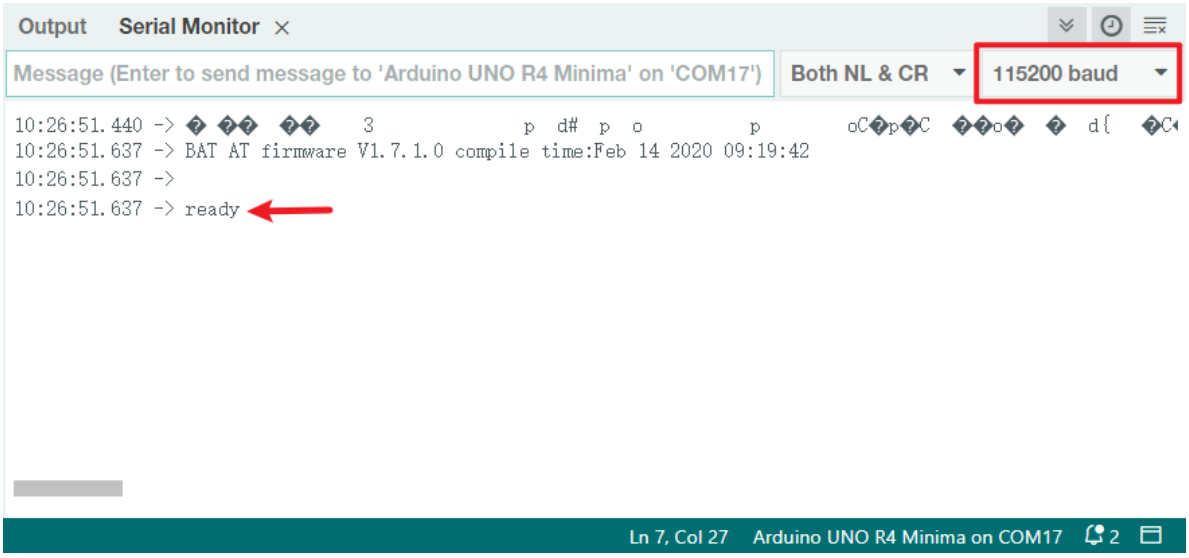
```
#include <SoftwareSerial.h>
SoftwareSerial espSerial(2, 3); //Rx,Tx

void setup() {
    // put your setup code here, to run once:
    Serial.begin(115200);
    espSerial.begin(115200);
```

(continues on next page)

(continued from previous page)

```
}

void loop() {
    if (espSerial.available()) {
        Serial.write(espSerial.read());
    }
    if (Serial.available()) {
        espSerial.write(Serial.read());
    }
}
```

3. Click the magnifying glass icon (Serial Monitor) in the upper right corner and set the baud rate to **115200**. (You may have some printed information like me, or you may not, it doesn't matter, just go to the next step.)

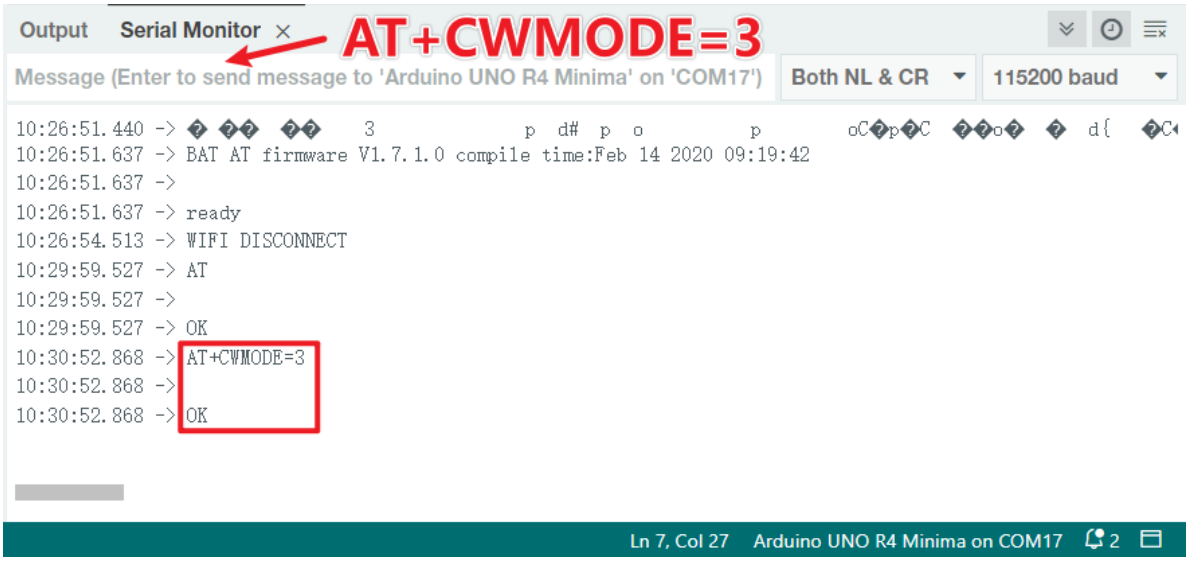


Warning:

- If `ready` doesn't appear, you can try to reset the ESP8266 module(connect RST to GND) and re-open the Serial Monitor.
- In addition, if the result is `OK`, you may need to re-burn the firmware, please refer to *How to re-burn the AT firmware for ESP8266 module?* for details. If you still can't solve it, please take a screenshot of the serial monitor and send it to service@sunfounder.com, we will help you solve the problem as soon as possible.

4. Click on **NEWLINE DROPDOWN BOX**, select `both NL & CR` in the drop down option, enter `AT`, if it returns OK, it means ESP8266 has successfully established connection with R4 board.

5. Enter `AT+CWMODE=3` and the managed mode will be changed to **Station and AP** coexistence.



Reference

-

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

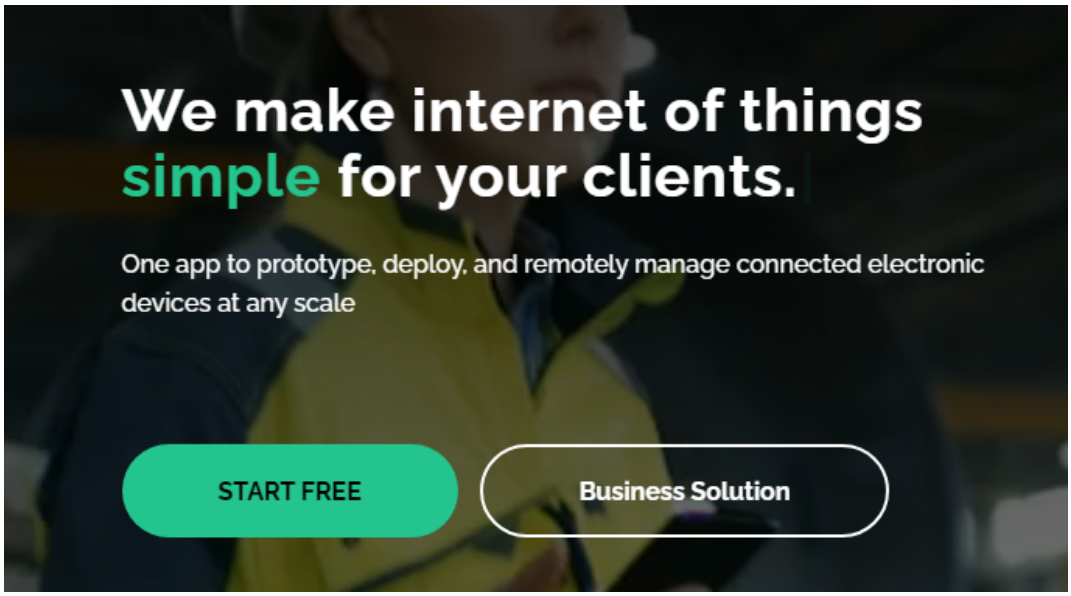
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

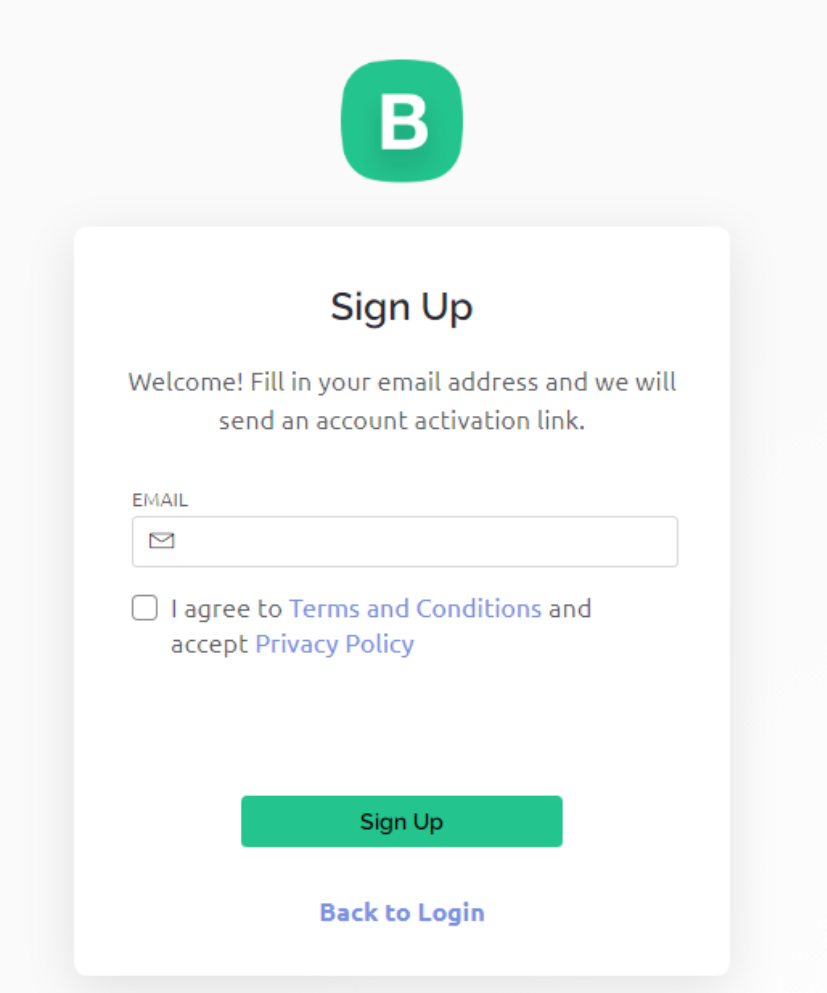
Ready to explore and create with us? Click [] and join today!

1.2 Configuring the Blynk

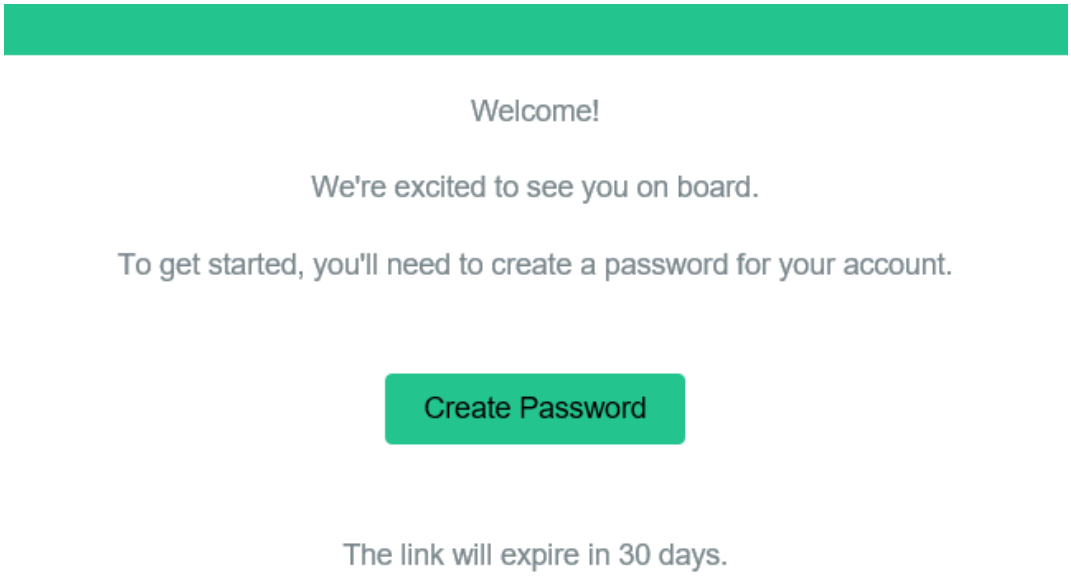
1. Go to the BLYNK and click **START FREE**.



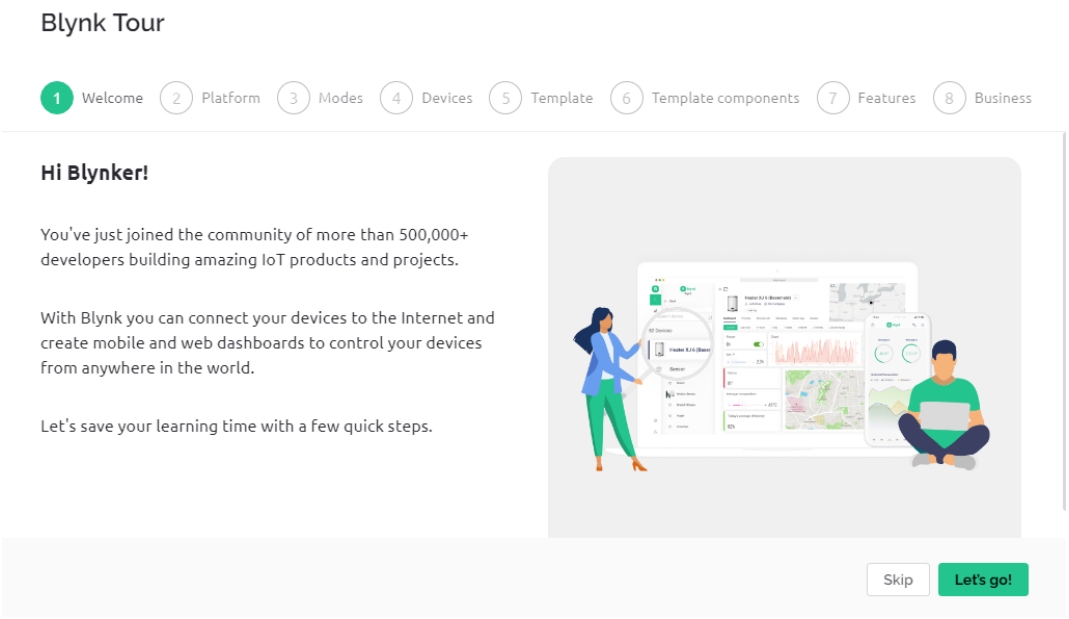
2. Fill in your email address to register an account.



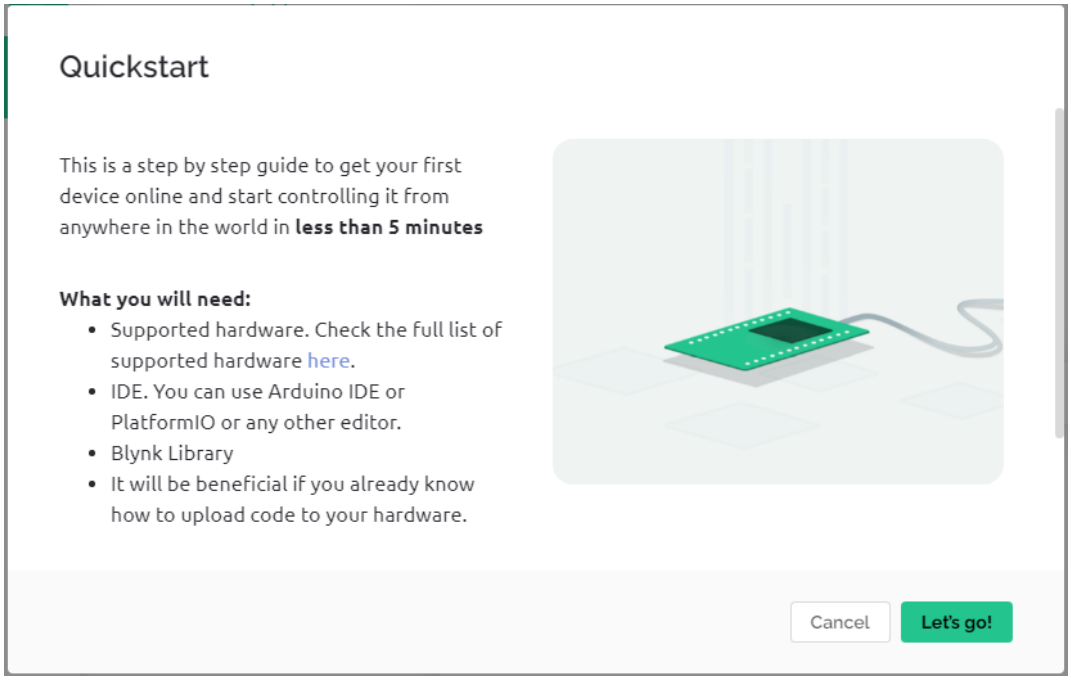
3. Go to your email address to complete your account registration.



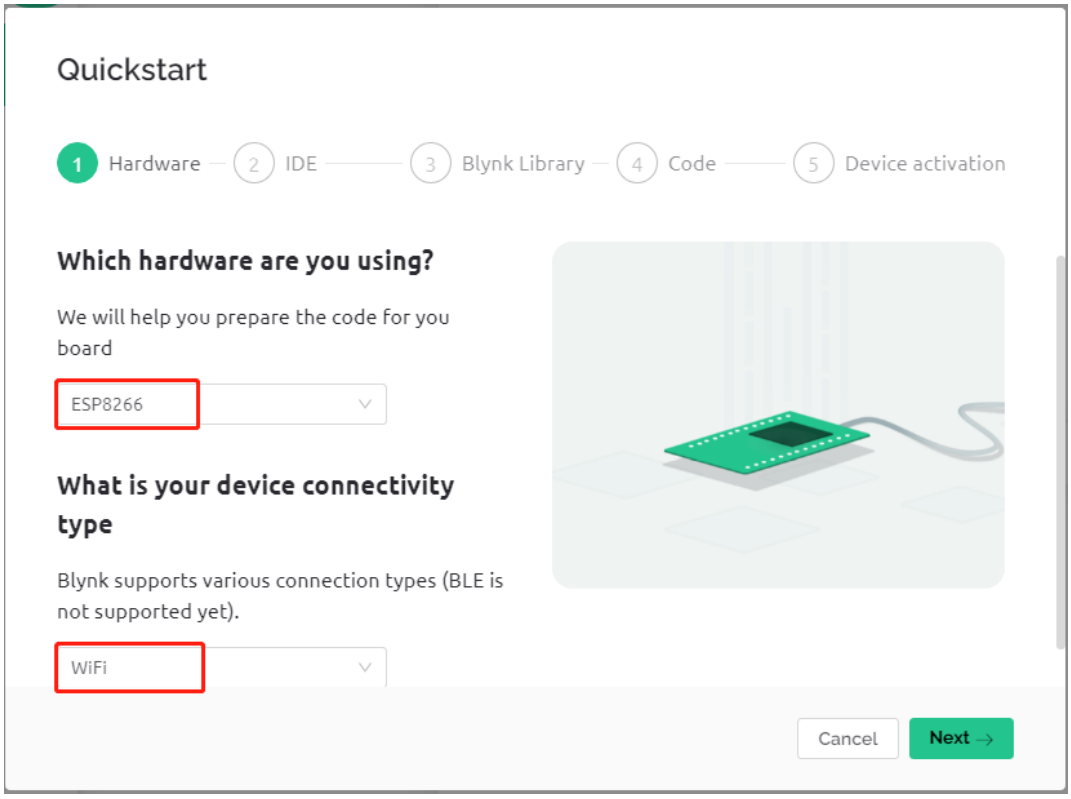
4. Afterwards, **Blynk Tour** will appear and you can read it to learn the basic information about the Blynk.



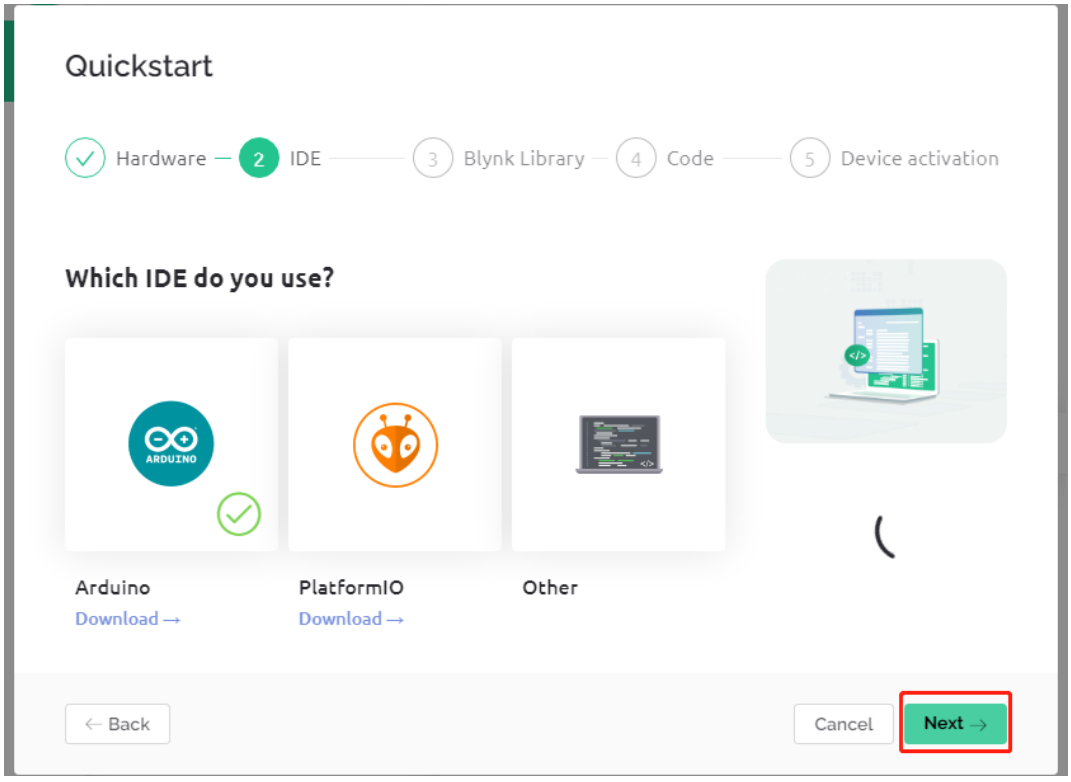
5. Next, we need to create a template and device with this **Quick Start**, click **Let's go**.



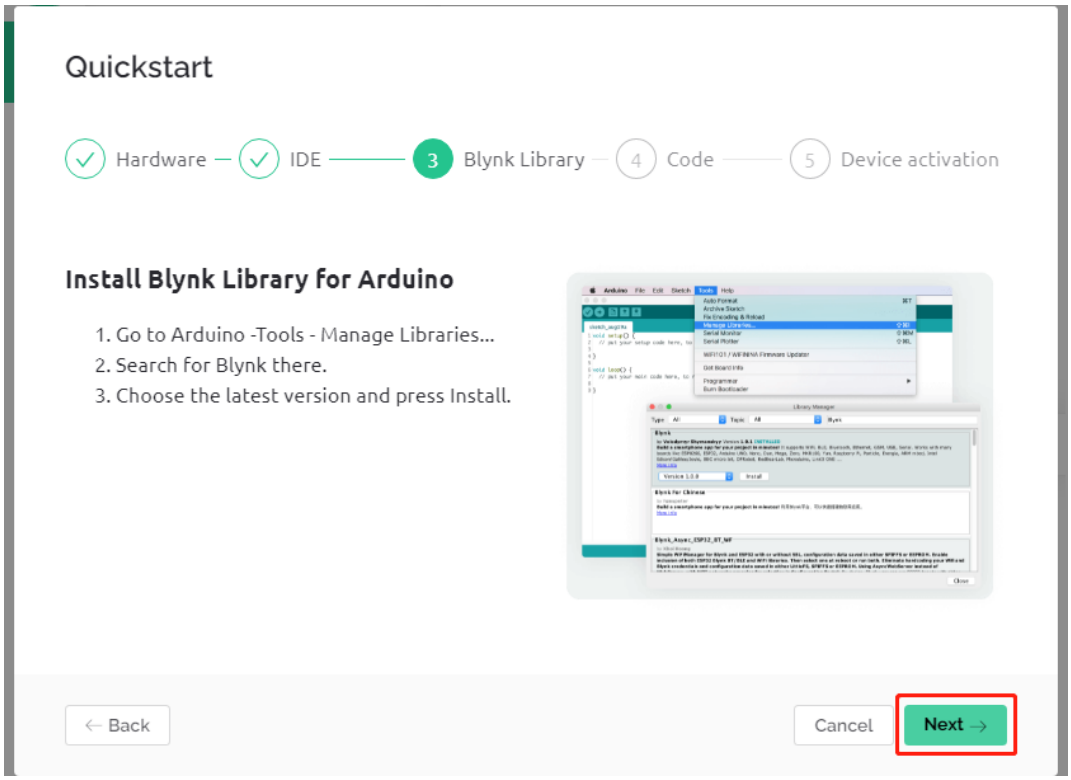
6. Select the hardware and connection type.



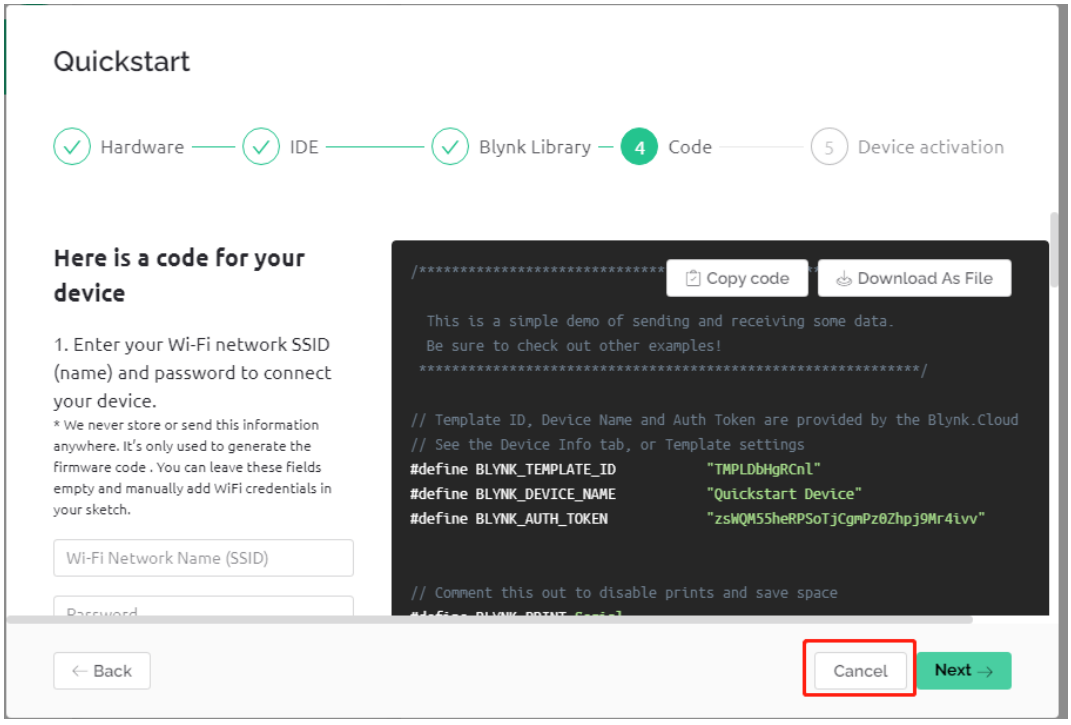
7. Here you are told which IDE you need to prepare, we recommend the **Arduino IDE**.



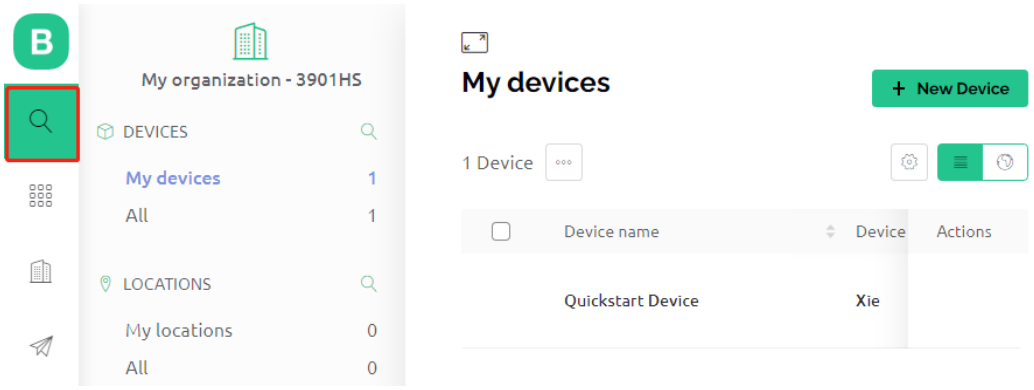
8. Here is the library you need to add, but the recommended library here is a bit problematic, we need to add other libraries manually (we will mention it later). Click **Next** here, and a new template and device will be created.



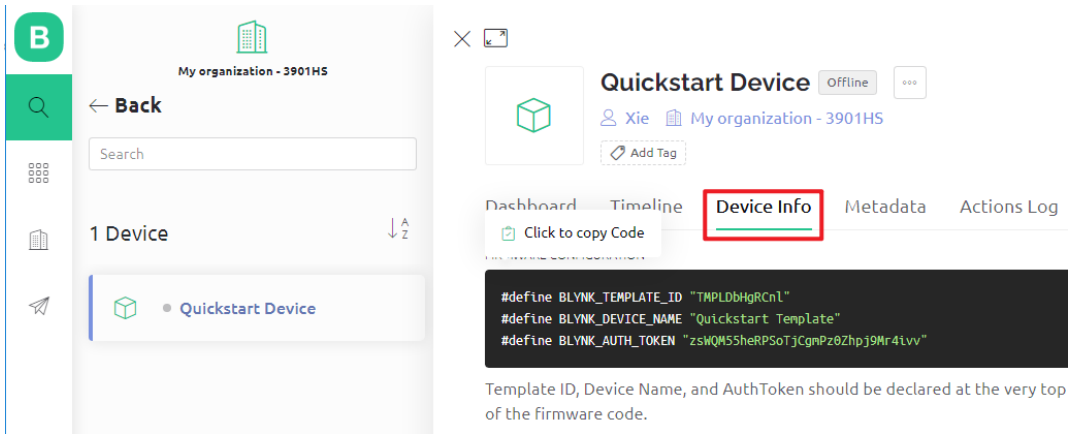
9. The next steps are to upload the relevant code and connect your board to Blynk, but since there is a problem with the library provided earlier, you need to add other libraries again. So click **Cancel** here to stop **Quick Start**.



10. Click the **Search** button and you will see the new device you just created.



11. Go to this **Quickstart Device** and click **Device Info**, you will see `TEMPLATE_ID`, `DEVICE_NAME`, and `AUTH_TOKEN` on the **Device info** page, and you will need to copy them later.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

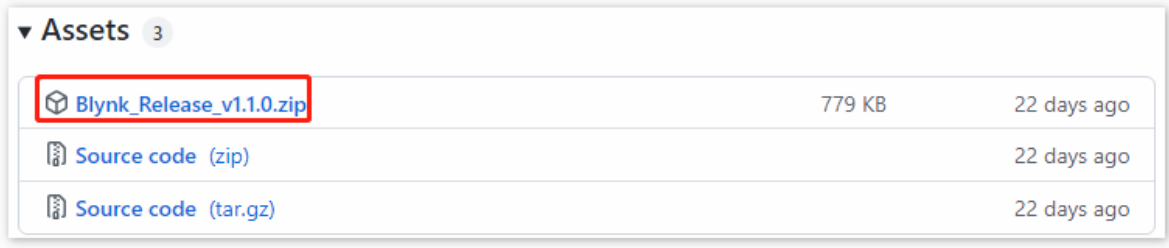
Ready to explore and create with us? Click [] and join today!

1.3 Adding the required libraries

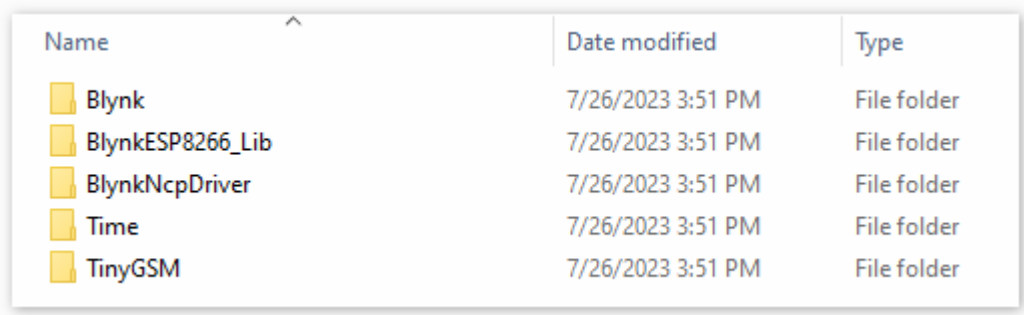
You need to add the correct libraries for the Arduino IDE to use Blynk.

1. Click , scroll down to **"Assets"** and download the first `.zip` file.

   Note: Please note that the version number shown in the image below may be outdated. We highly recommend downloading and installing the latest version available.

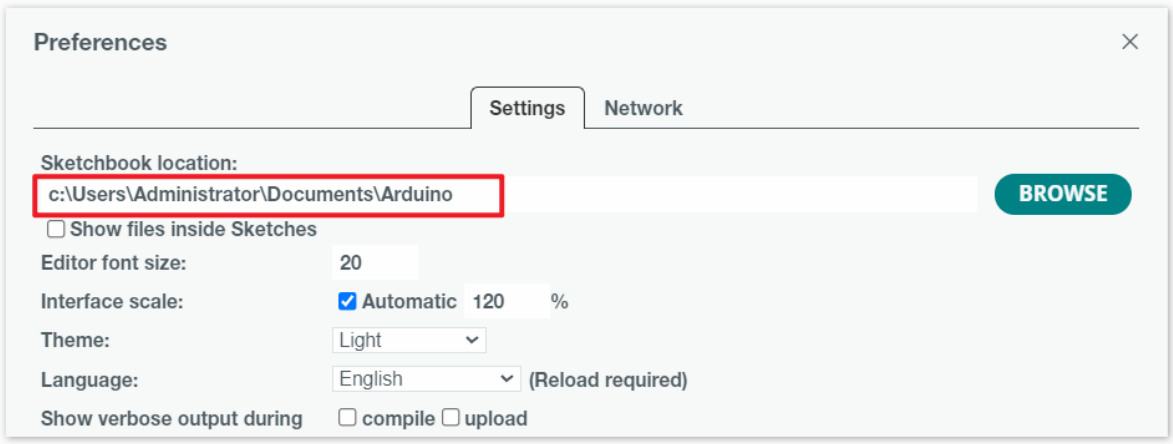


2. Unzip this file and then enter the `libraries` folder to see the following folders.

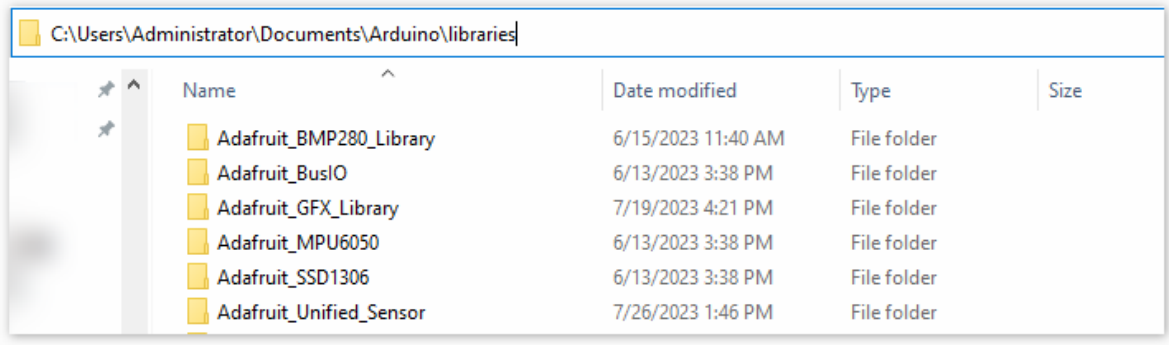


3. Copy them all and add them to the `libraries` folder of your sketchbook.

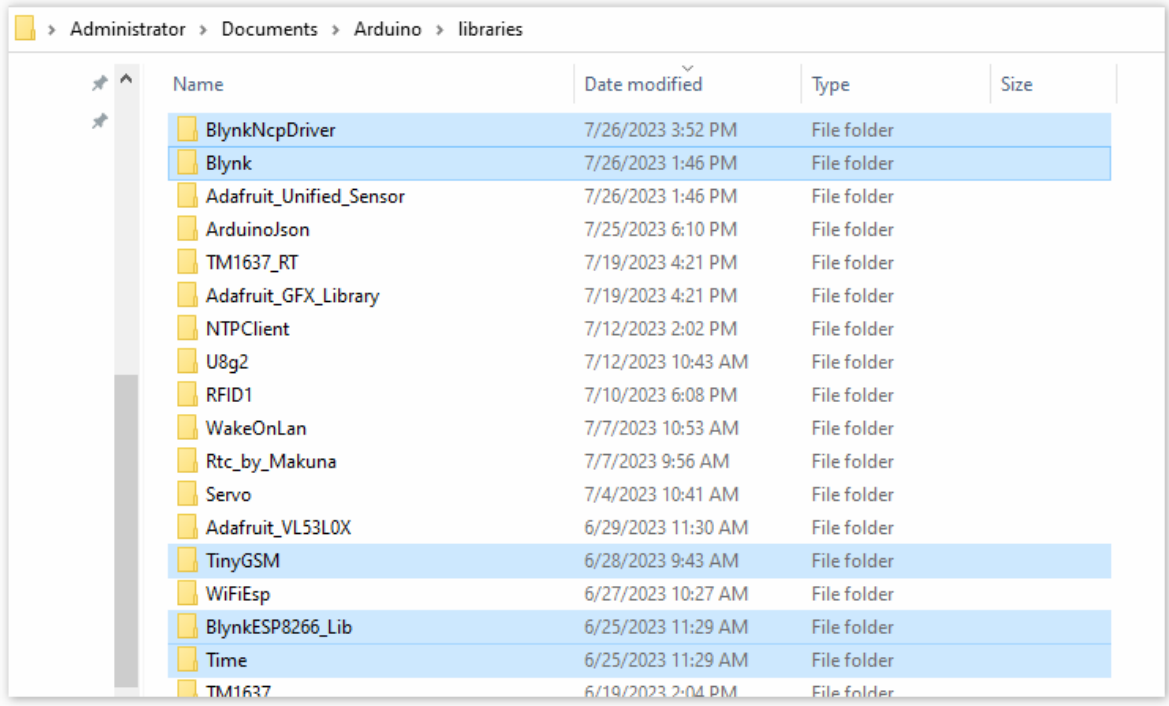
   Step 1: You can find or change the location of your libraries folder at `File` > `Preferences` > `Sketchbook location`.



   Step 2: Go to the location of your Sketchbook location(find from Arduino IDE). And find `libraries` folder, click to open it.



Step 3: Paste all the unzipped folders of `Blynk_Release_vx.x.x\libraries` into the libraries folder.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

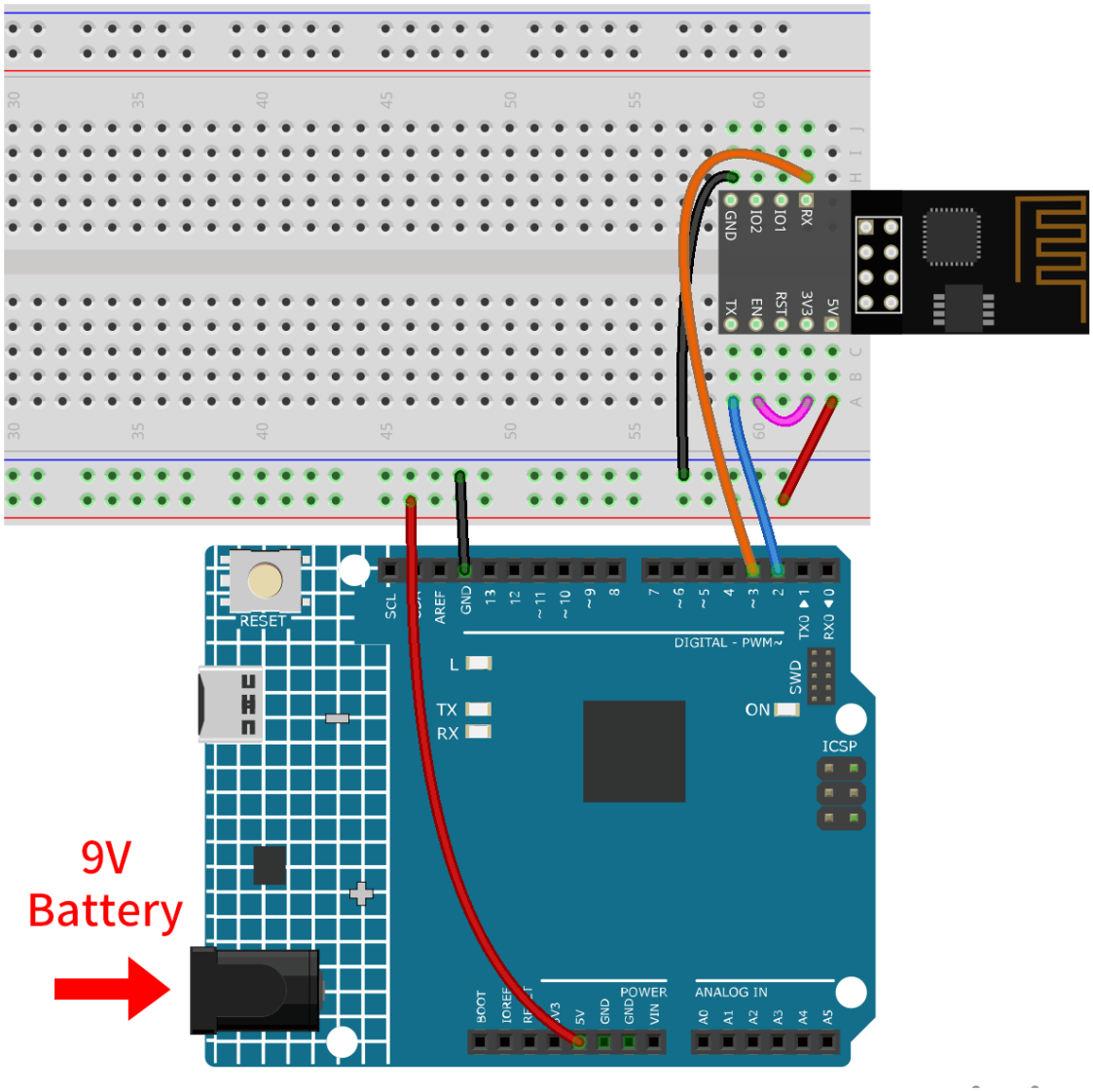
- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4 Connecting the R4 board to Blynk

1. Reconnect the ESP8266 module and R4 board, here the software serial is used, so TX and RX are connected to pins 2 and 3 of R4 board respectively.

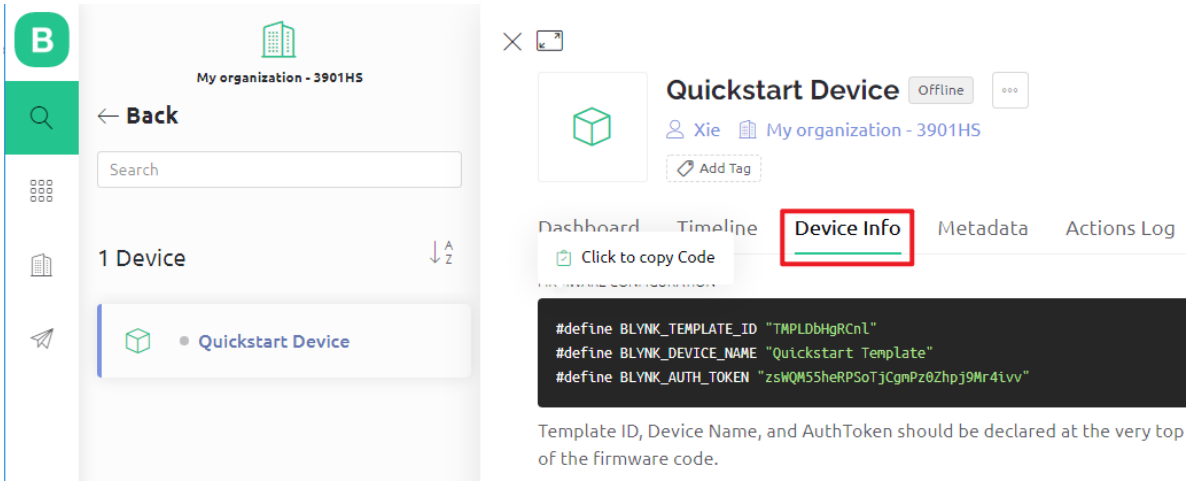
Note: The ESP8266 module requires a high current to provide a stable operating environment, so make sure the 9V battery is plugged in.



1. Open the `00-Blynk_quick_start.ino` file under the path of `ultimate-sensor-kit\iot_project\wifi\00-Blynk_quick_start`. Or copy this code into **Arduino IDE**.

2. Replace the following three lines of code that you can copy from your account's **Device info** page. These three lines of code will allow your R4 board to find your blynk account.

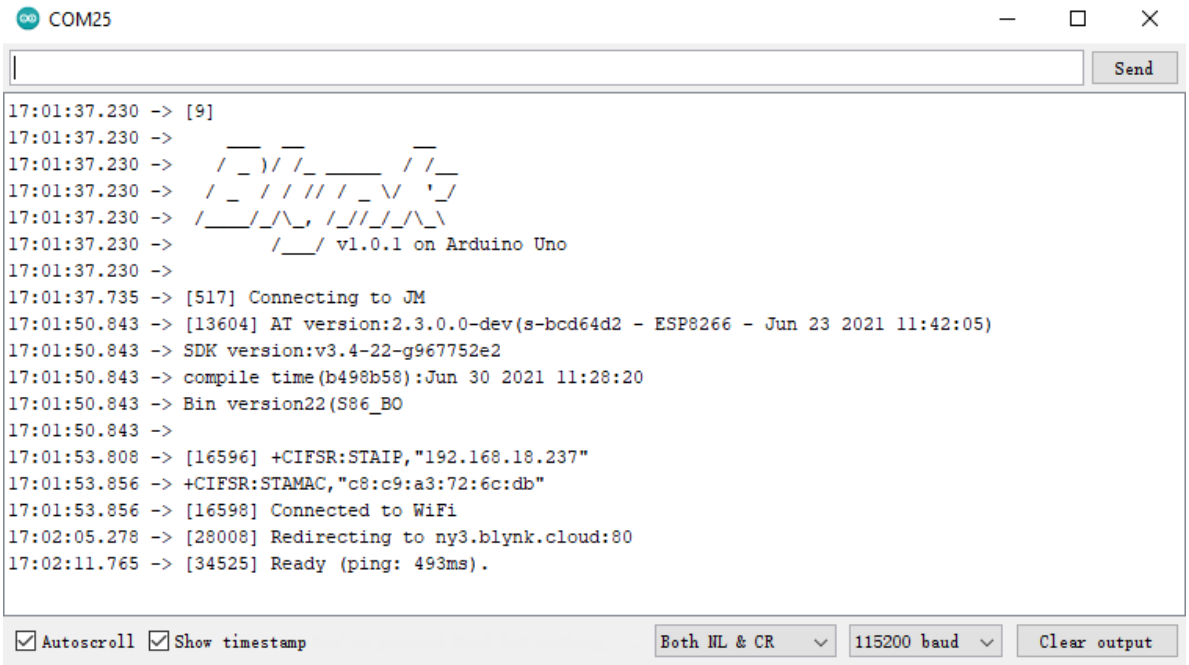
```
#define BLYNK_TEMPLATE_ID "TMPLxxxxxx"
#define BLYNK_DEVICE_NAME "Device"
#define BLYNK_AUTH_TOKEN "YourAuthToken"
```



3. Fill in the `ssid` and `password` of the WiFi you are using.

```
char ssid[] = "ssid";
char pass[] = "password";
```

4. Upload the code to the R4 board, then open the serial monitor and set the baud rate to 115200. when the R4 board communicates with Blynk successfully, the serial monitor will show the `ready` character.

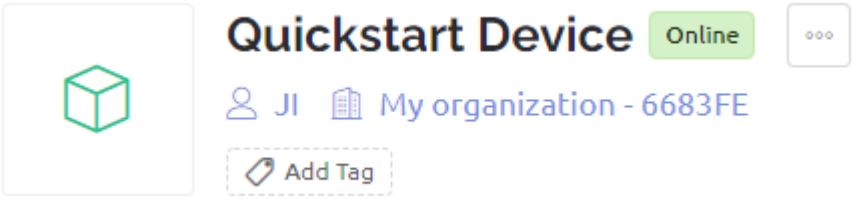


Note: If the message `ESP is not responding` appears when you connect, please follow these steps.

- Make sure the 9V battery is plugged in.
- Reset the ESP8266 module by connecting the pin RST to GND for 1 second, then unplug it.
- Press the reset button on the R4 board.

Sometimes, you may need to repeat the above operation 3-5 times, please be patient.

5. The status of Blynk will change from **offline** to **online**.



IoT Project

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.52 Lesson 48: Weather Monitor with ThingSpeak

This project collects temperature and pressure data using an Atmospheric Pressure Sensor. The collected data is then transmitted to the ThingSpeak cloud platform via an ESP8266 module and Wi-Fi network at regular time intervals.

Required Components

In this project, we need the following components.

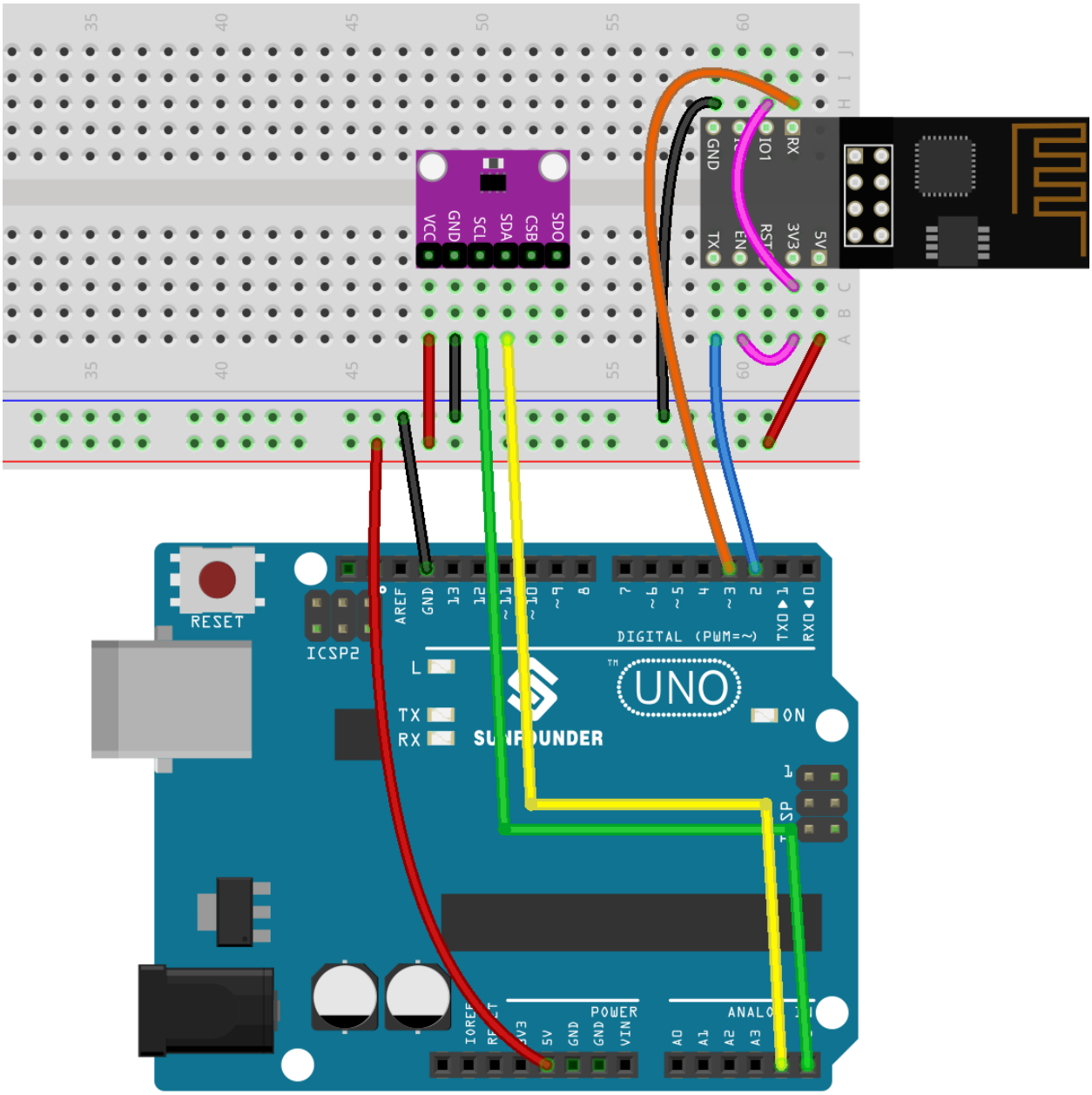
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

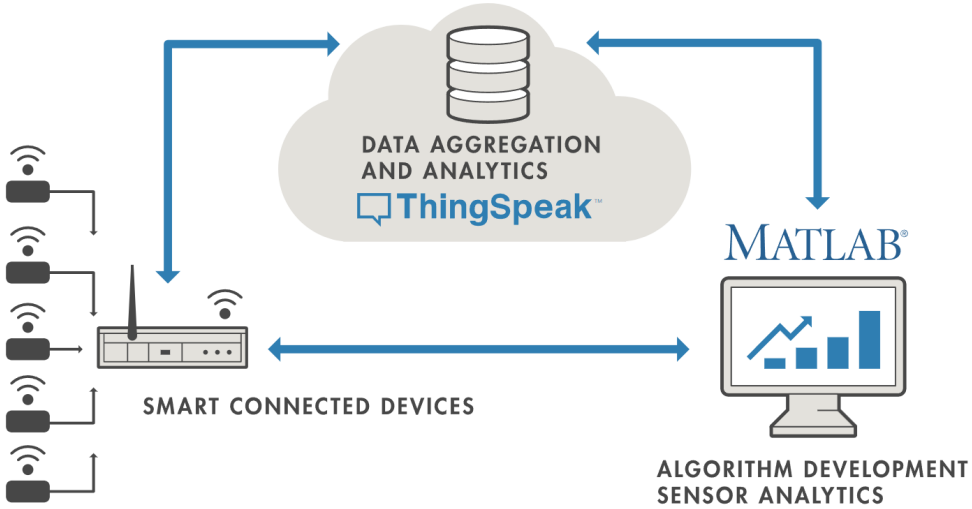
Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Breadboard	
ESP8266 Module	-
Temperature, Humidity & Pressure Sensor (BMP280)	-

Wiring


VCC
GND
SCL
SDA
CSB
SDO
GND
IO1
RX
TX
EN
RST
3V3
5V
RESET
ICSP2
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM=~)
L
TX
RX
UNO
ON
POWER
ANALOG IN
IOREF
RESET
3V3
5V
GND
GND
VIN
A0
A1
A2
A3

Configure ThingSpeak

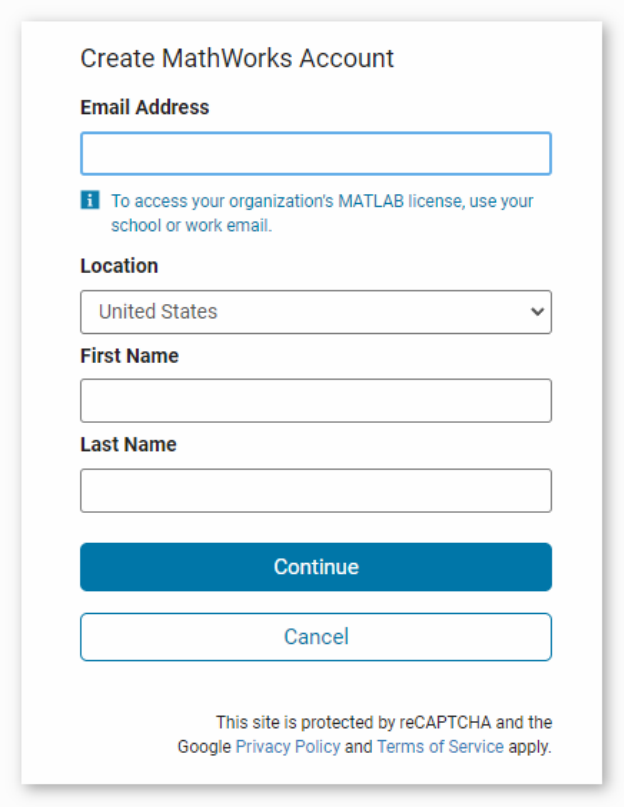
™ is an IoT analytics platform service that allows you to aggregate, visualize and analyze live data streams in the cloud. ThingSpeak provides instant visualizations of data posted by your devices to ThingSpeak. With the ability to execute MATLAB® code in ThingSpeak you can perform online analysis and processing of the data as it comes in. ThingSpeak is often used for prototyping and proof of concept IoT systems that require analytics.



1) Creating ThingSpeak Account

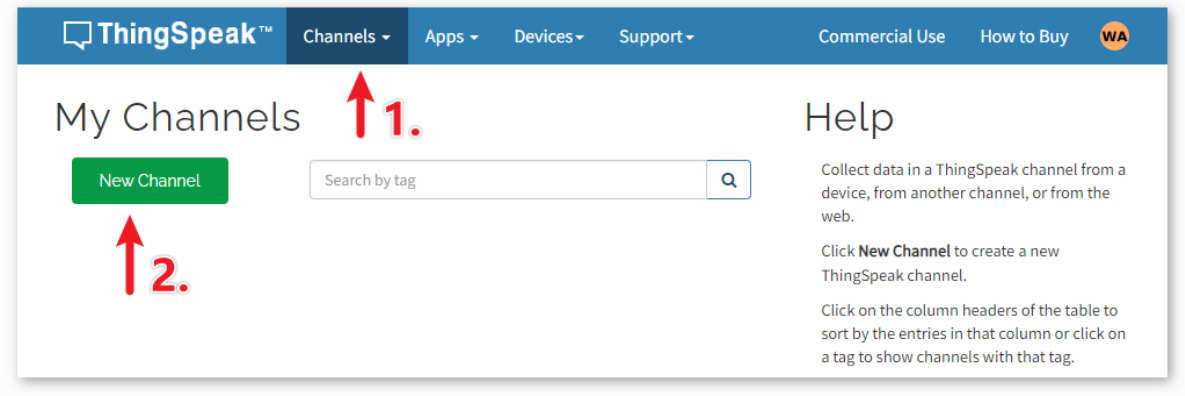
The first thing you need to do is to create an account with ThingSpeak. Since the collaboration with MATLAB, you can use your MathWorks credentials to login to .

If you do not have one, you need to create an account with MathWorks and login to ThingSpeak Application.

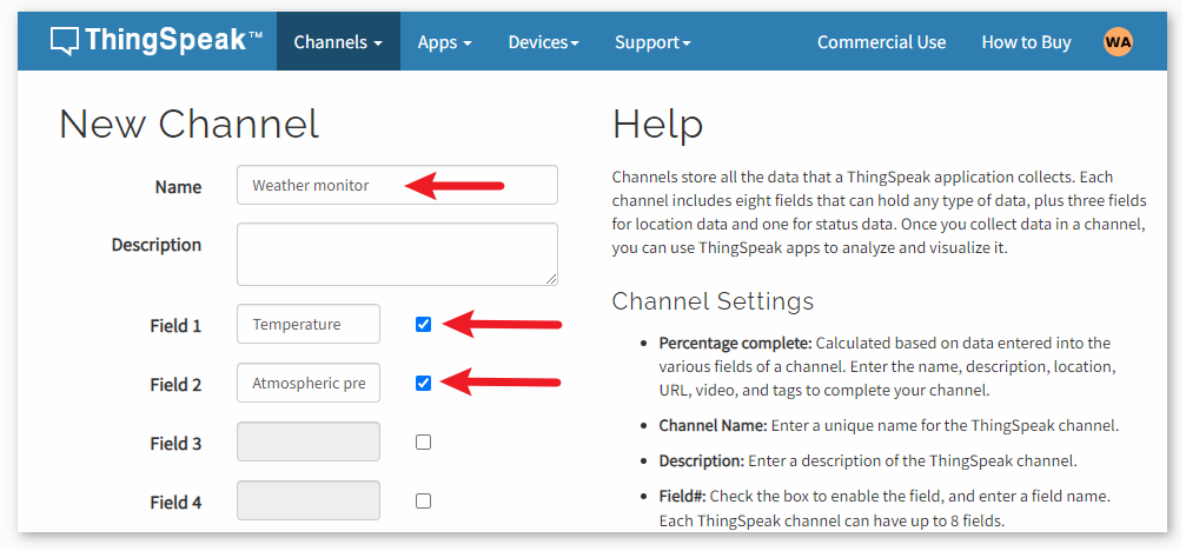


2) Creating the channel

After logging in, create a new channel to store the data by going to "Channels" > "My Channels" and clicking on "New Channel".



For this project, we need to create a channel called "**Weather Monitor**" with two fields: **Field 1** for "**Temperature**" and **Field 2** for "**Atmospheric Pressure**".



Code

1. Open the `Lesson_48_Iot_Weather_Monitor.ino` file under the path of `universal-maker-sensor-kit\arduino_uno\Lesson_48_Iot_Weather_Monitor`, or copy this code into **Arduino IDE**.

Note: To install the library, use the Arduino Library Manager and search for **"Adafruit BMP280"** and install it.

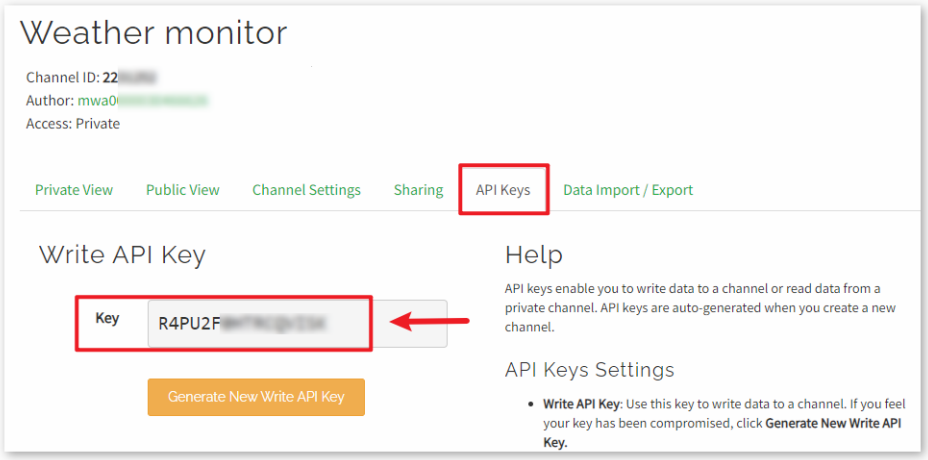
---

2. You need to enter the `mySSID` and `myPWD` of the WiFi you are using.

```
String mySSID = "your_ssid";     // WiFi SSID
String myPWD = "your_password";  // WiFi Password
```

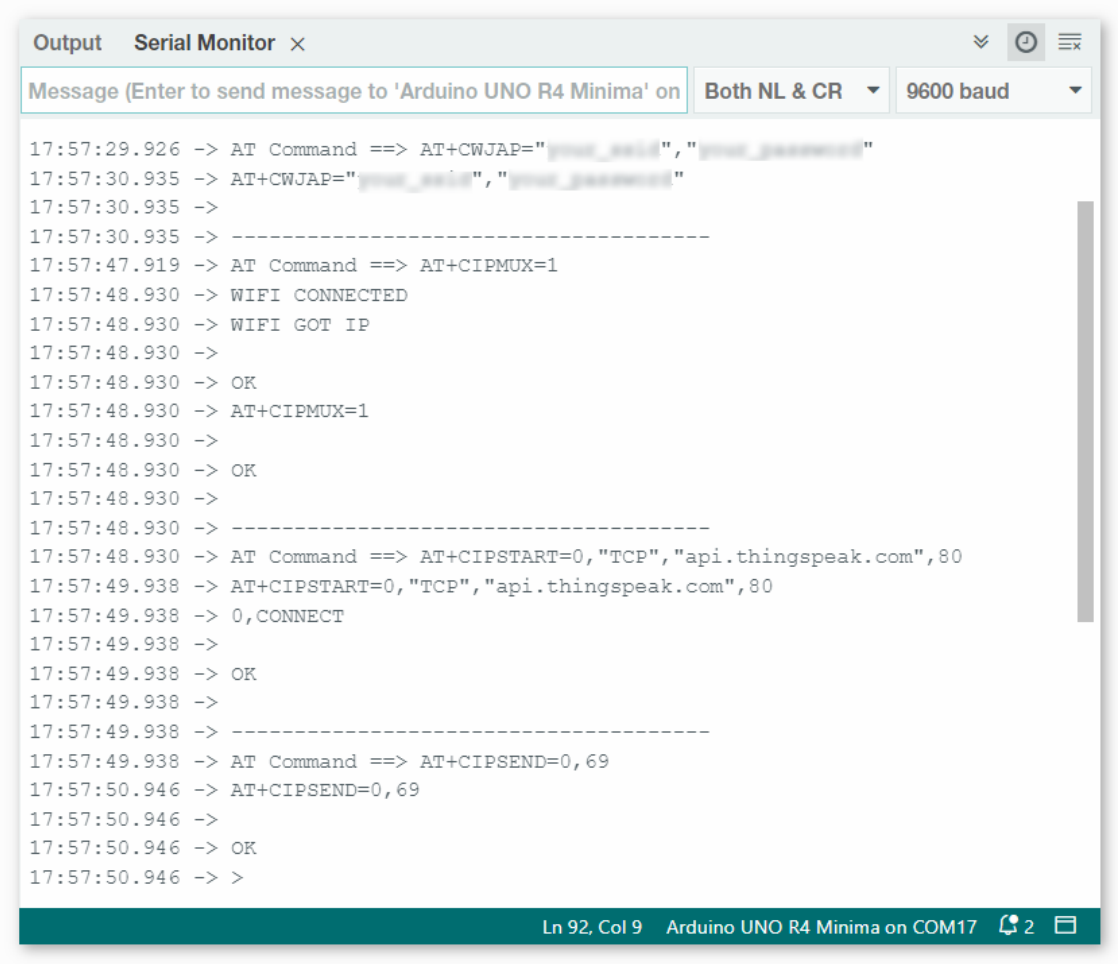
3. You also need to modify the `myAPI` with your ThingSpeak Channel API key.

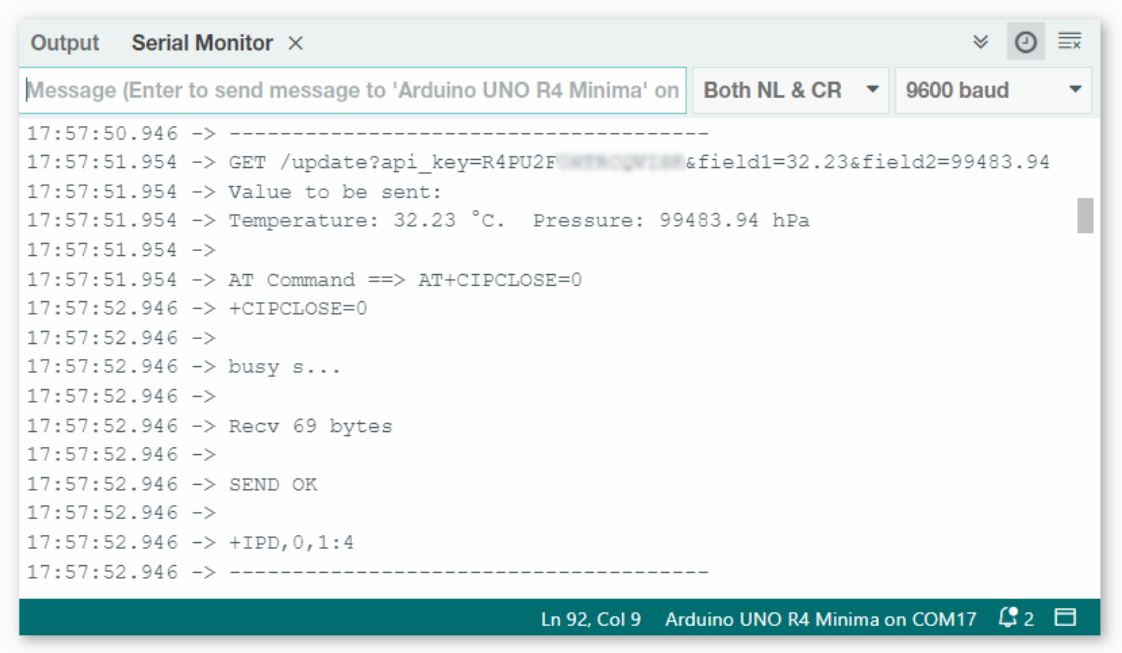
```
String myAPI = "xxxxxxxxxxxx";  // API Key
```



Here you can find **your unique API KEY that you must keep private**.

4. After selecting the correct board and port, click the **Upload** button.
5. Open the Serial monitor(set baudrate to **9600**) and wait for a prompt such as a successful connection to appear.





Code Analysis

1. Initialization and Bluetooth setup

```
// Set up Bluetooth module communication
#include <SoftwareSerial.h>
const int bluetoothTx = 3;
const int bluetoothRx = 4;
SoftwareSerial bleSerial(bluetoothTx, bluetoothRx);
```

We begin by including the SoftwareSerial library to help us with Bluetooth communication. The Bluetooth module's TX and RX pins are then defined and associated with pins 3 and 4 on the Arduino. Finally, we initialize the `bleSerial` object for Bluetooth communication.

2. LED Pin Definitions

```
// Pin numbers for each LED
const int rledPin = 10;  //red
const int yledPin = 11;  //yellow
const int gledPin = 12;  //green
```

Here, we're defining which Arduino pins our LEDs are connected to. The red LED is on pin 10, yellow on 11, and green on 12.

3. setup() Function

```
void setup() {
   pinMode(rledPin, OUTPUT);
   pinMode(yledPin, OUTPUT);
   pinMode(gledPin, OUTPUT);

   Serial.begin(9600);
```

(continues on next page)

(continued from previous page)

```
    bleSerial.begin(9600);
}
```

In the `setup()` function, we set the LED pins as `OUTPUT`. We also start serial communication for both the Bluetooth module and the default serial (connected to the computer) at a baud rate of 9600.

4. Main loop() for Bluetooth Communication

```
void loop() {
    while (bleSerial.available() > 0) {
        character = bleSerial.read();
        Serial.println(character);

        if (character == 'R') {
            toggleLights(rledPin);
        } else if (character == 'Y') {
            toggleLights(yledPin);
        } else if (character == 'G') {
            toggleLights(gledPin);
        }
    }
}
```

Inside our main `loop()`, we continuously check if data is available from the Bluetooth module. If we receive data, we read the character and display it in the serial monitor. Depending on the character received (R, Y, or G), we toggle the respective LED using the `toggleLights()` function.

5. Toggle Lights Function

```
void toggleLights(int targetLight) {
    digitalWrite(rledPin, LOW);
    digitalWrite(yledPin, LOW);
    digitalWrite(gledPin, LOW);

    digitalWrite(targetLight, HIGH);
}
```

This function, `toggleLights()`, turns off all the LEDs first. After ensuring they are all off, it turns on the specified target LED. This ensures that only one LED is on at a time.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.53 Lesson 49: Vibration Alert System with IFTTT

This project sets up a vibration detection system using an Arduino board (Uno R4 or R3) with an ESP8266 module and a vibration sensor (SW-420). When a vibration is detected, the system sends an HTTP request to an IFTTT server, potentially triggering various actions such as sending a notification or an email.

To avoid excessive alerts within a short timeframe, the system has been programmed to send these HTTP requests at a minimum interval of 2 minutes (120000 milliseconds). This interval could be adjusted based on the user's needs.

Required Components

In this project, we need the following components.

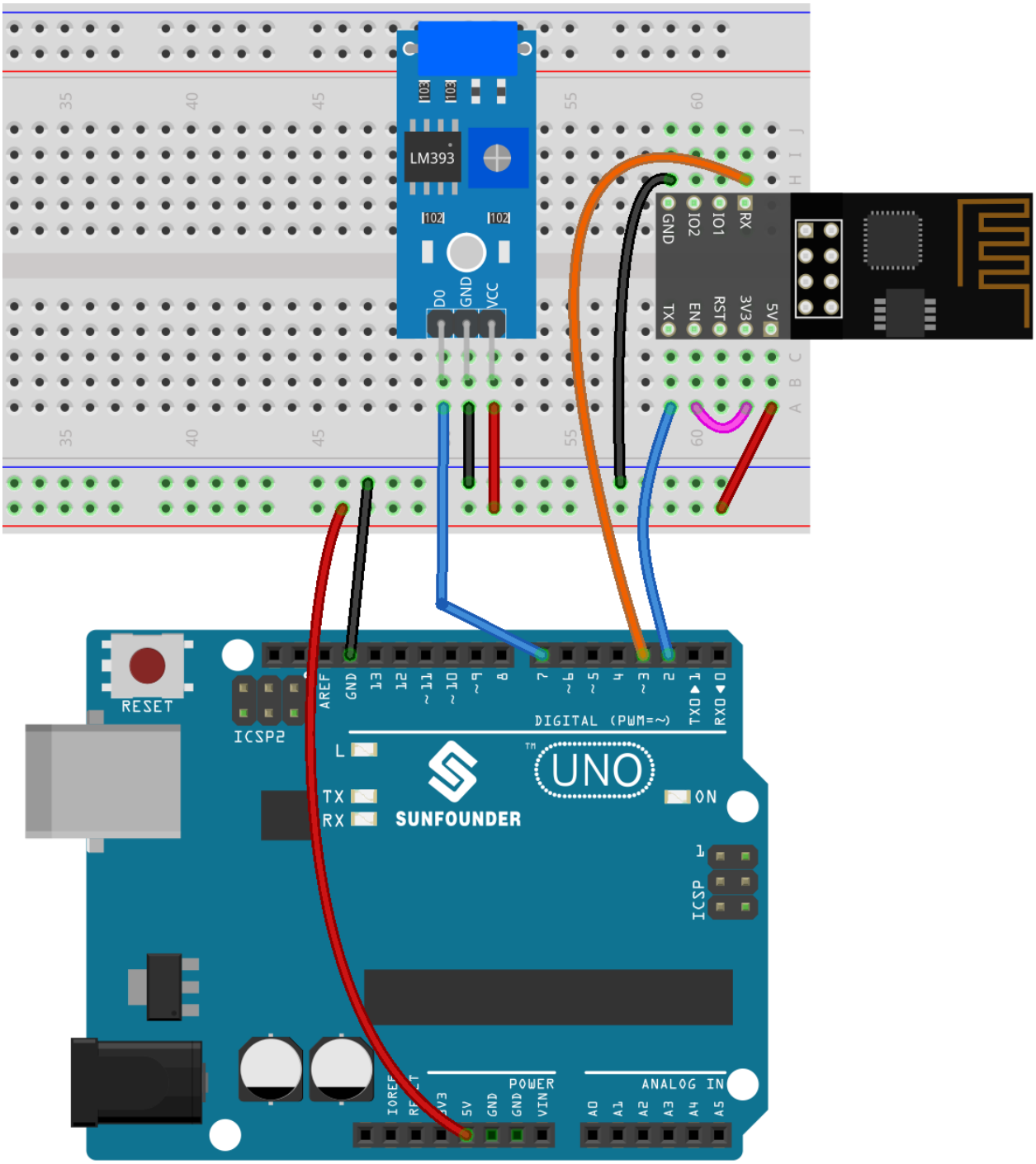
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

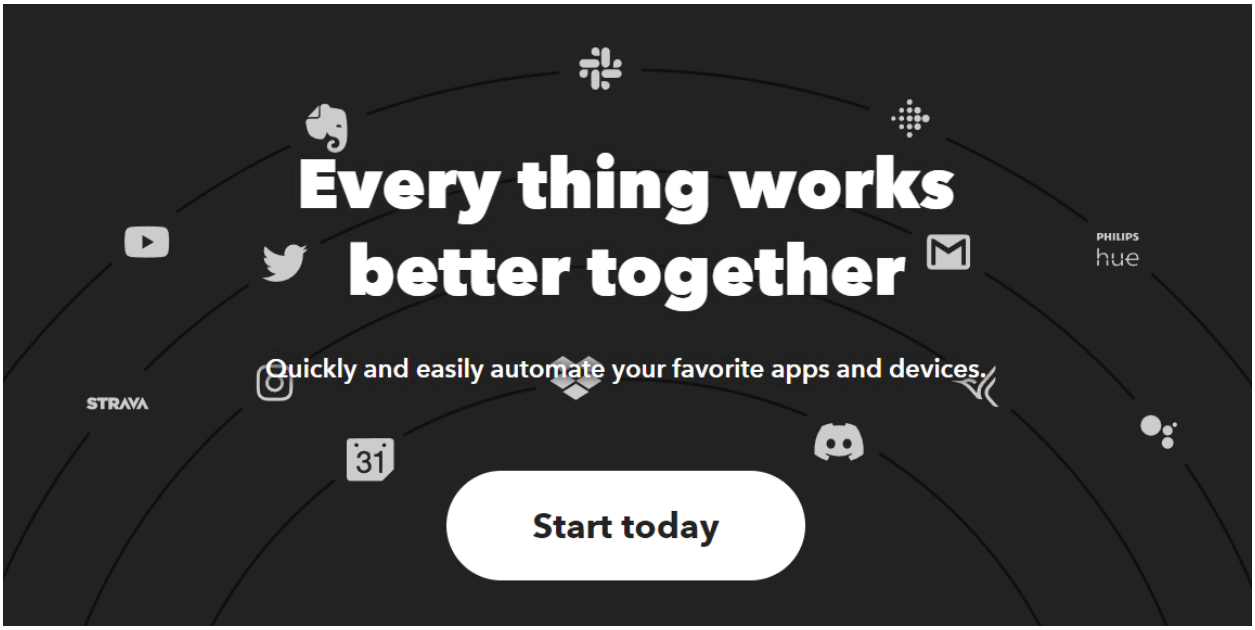
Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Breadboard	
ESP8266 Module	-
Vibration Sensor Module (SW-420)	-

Wiring

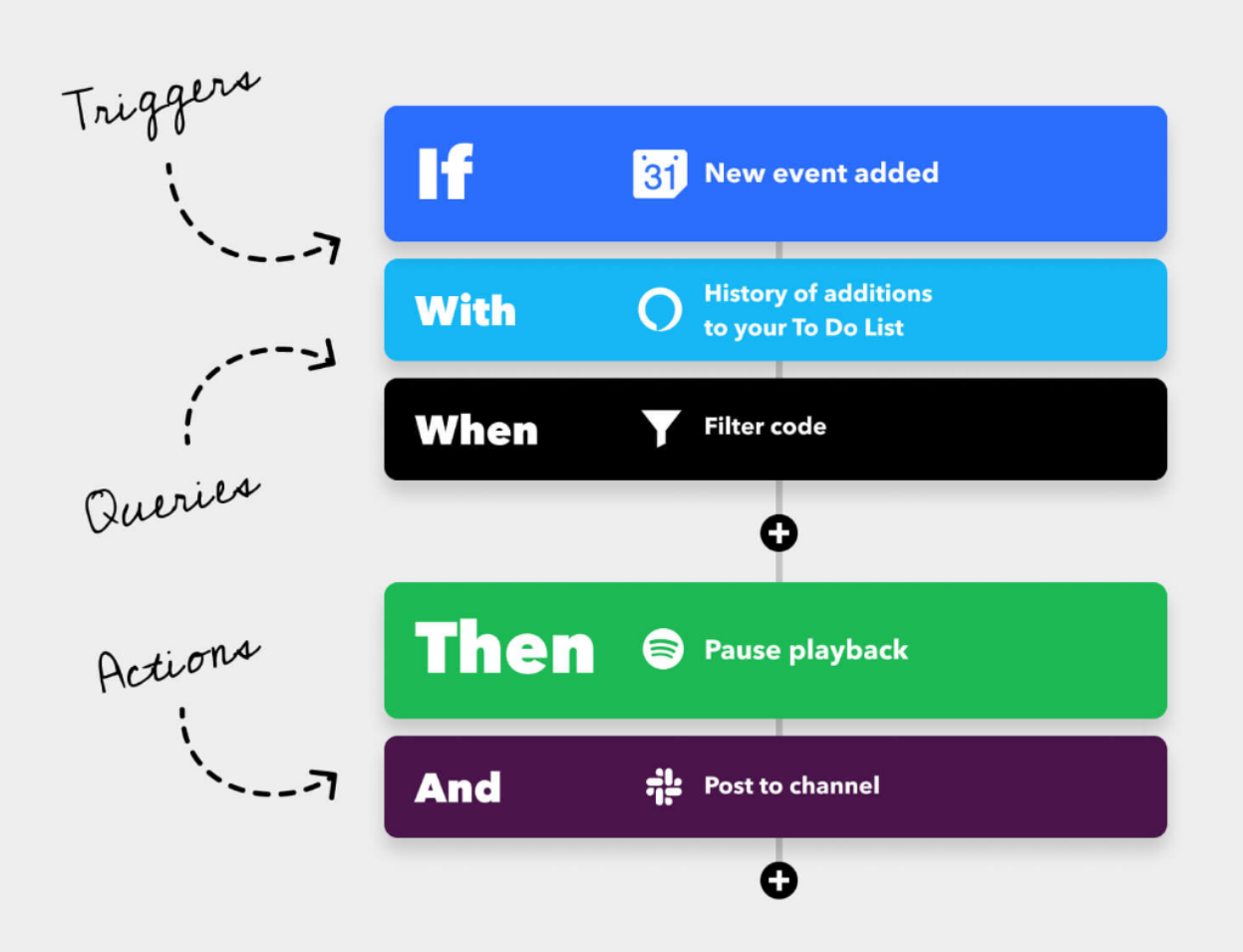

LM393
D0
GND
VCC
GND
IO2
IO1
RX
TX
EN
RST
3V3
5V
RESET
ICSP2
AREF
GND
DIGITAL (PWM=~)
TXD
RXD
L
TX
RX
SUNFOUNDER
UNO
ON
ICSP
POWER
ANALOG IN
VIN
A0
A1
A2
A3
A4
A5

Configure IFTTT

is a private commercial company founded in 2011 that runs online digital automation platforms which it offers as a service. Their platforms provide a visual interface for making cross-platform if statements to its users, which, as of 2020, numbered 18 million people.

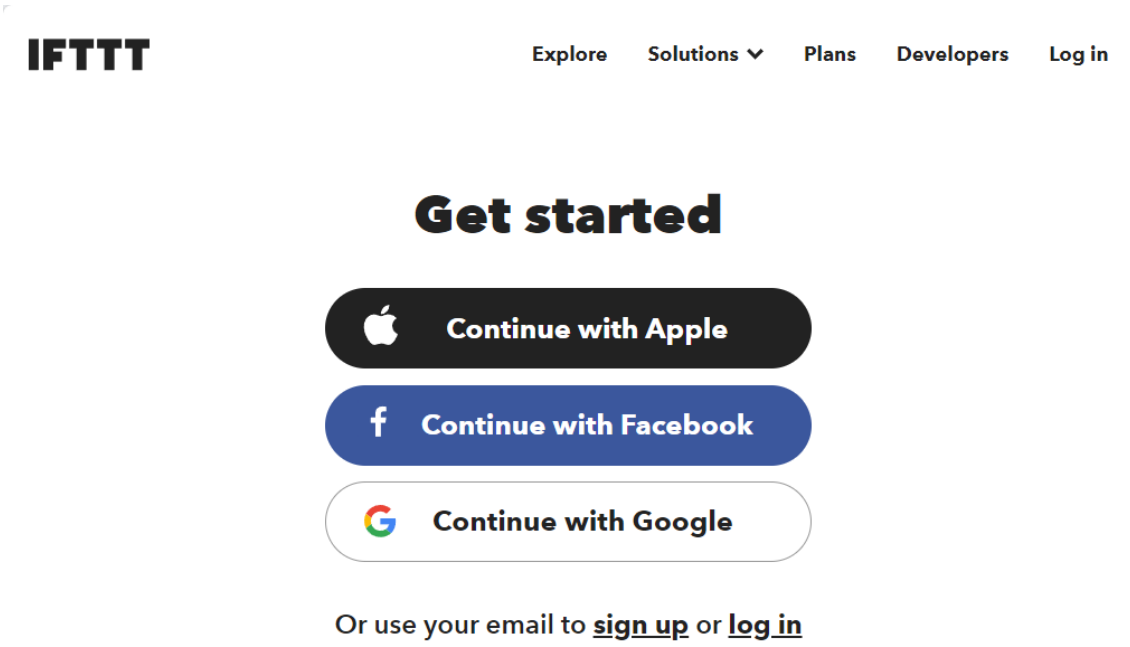


IFTTT stands for “If This Then That.” Basically, if certain conditions are met, then something else will happen. The “if this” part is called a trigger, and the “then that” part is called an action. It joins smart home devices, social media, delivery apps, and more so it can perform automated tasks.

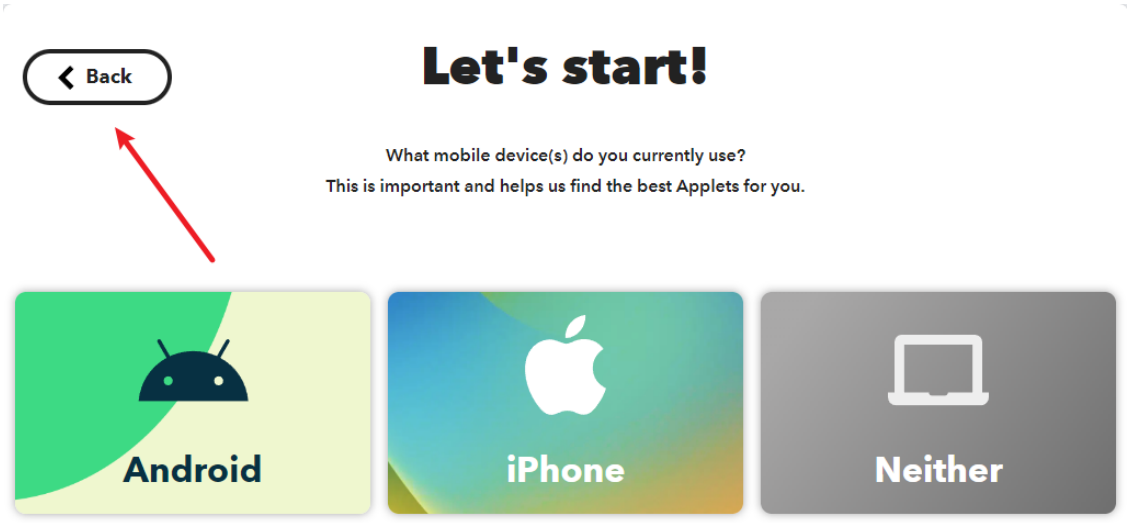


1) Sign up IFTTT

Type "https://ifttt.com" in your browser and click on the "Get started" button located at the center of the page. Fill out the form with your information to create an account.

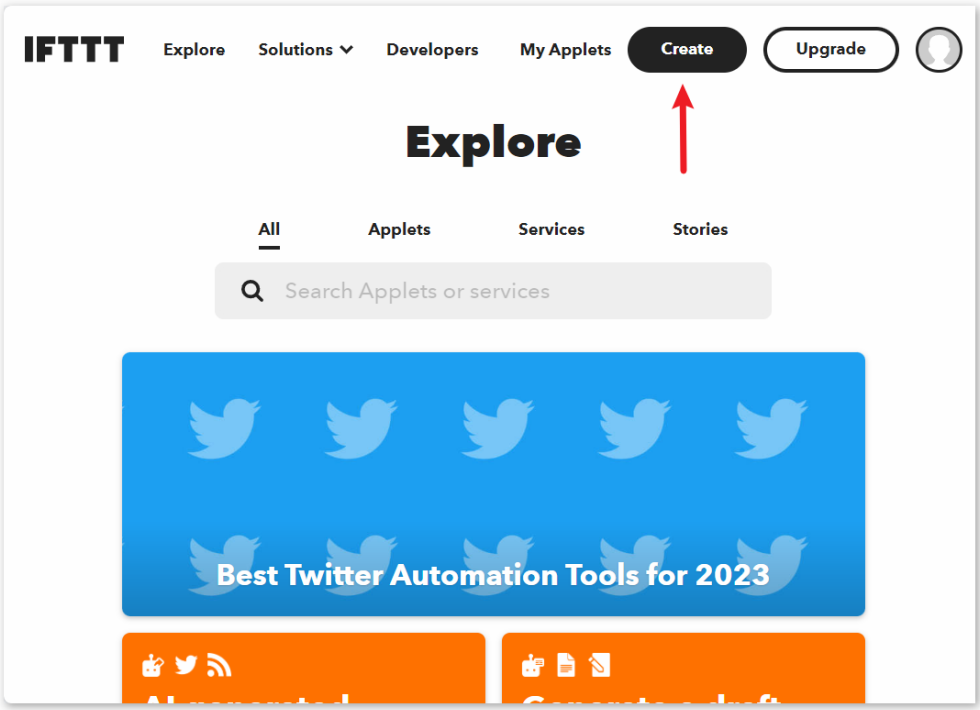


Click “Back” to exit quickstart, return to the IFTTT homepage, refresh the page and log in again.



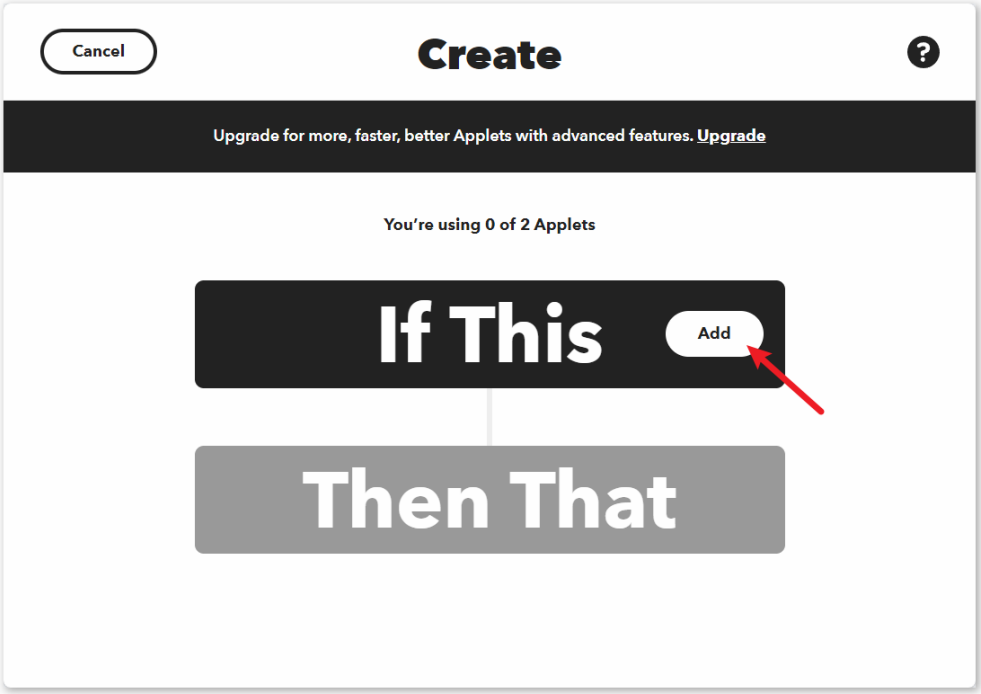
2) Creating the Applet

Click “Create” to start creating the Applet.

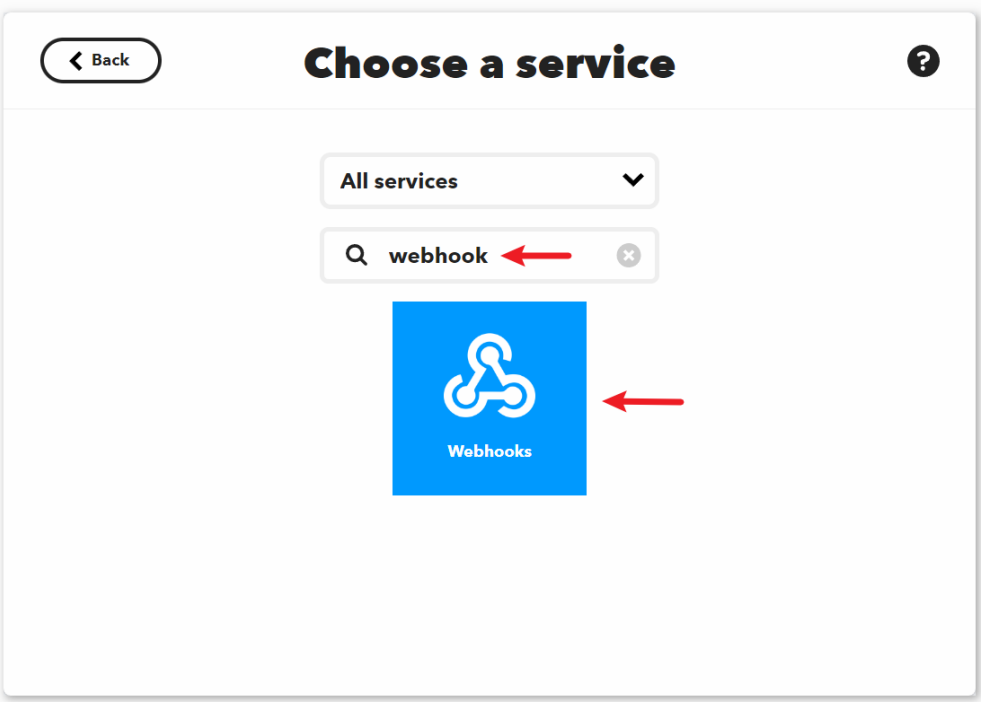


If This trigger

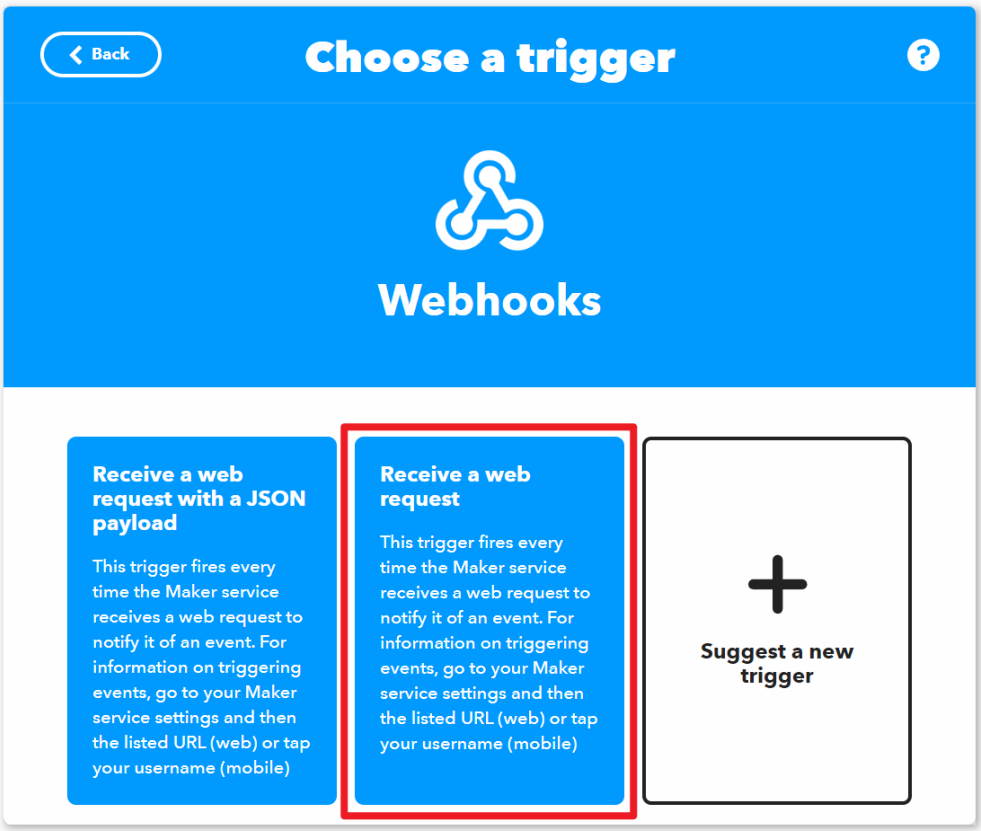
Click “Add” next to “If This” to add a trigger.



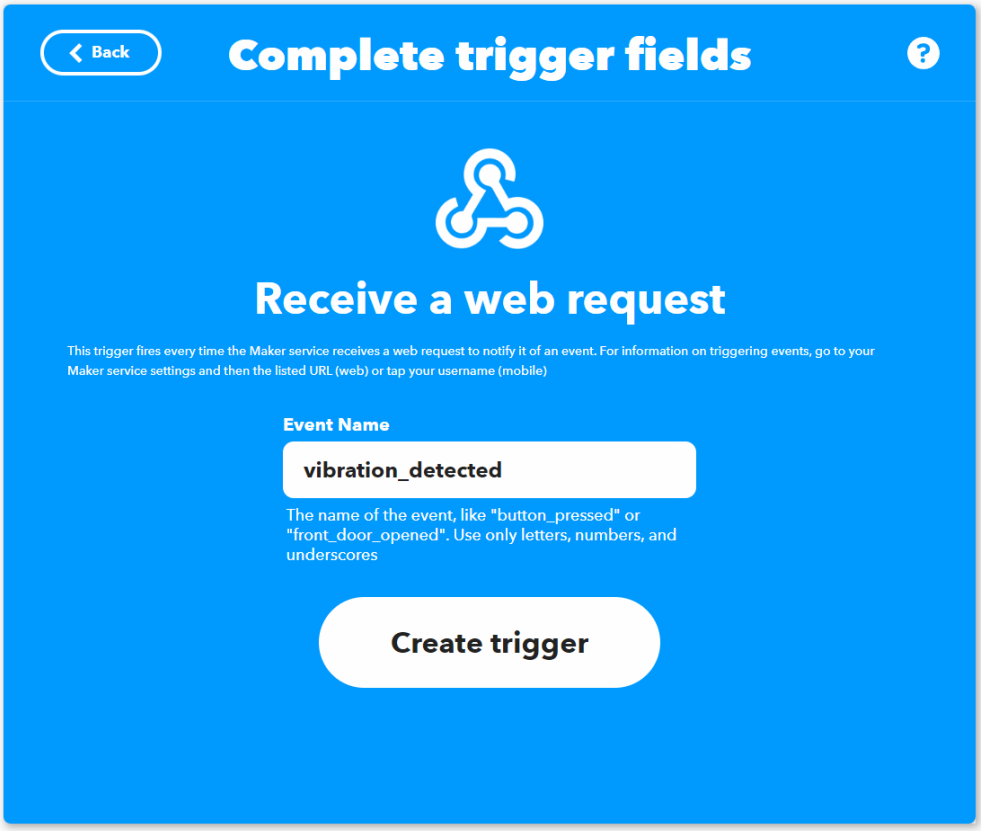
Search for “webhook” and click on “Webhooks”.



Click on “Receive a web request” on the page shown in the following image.

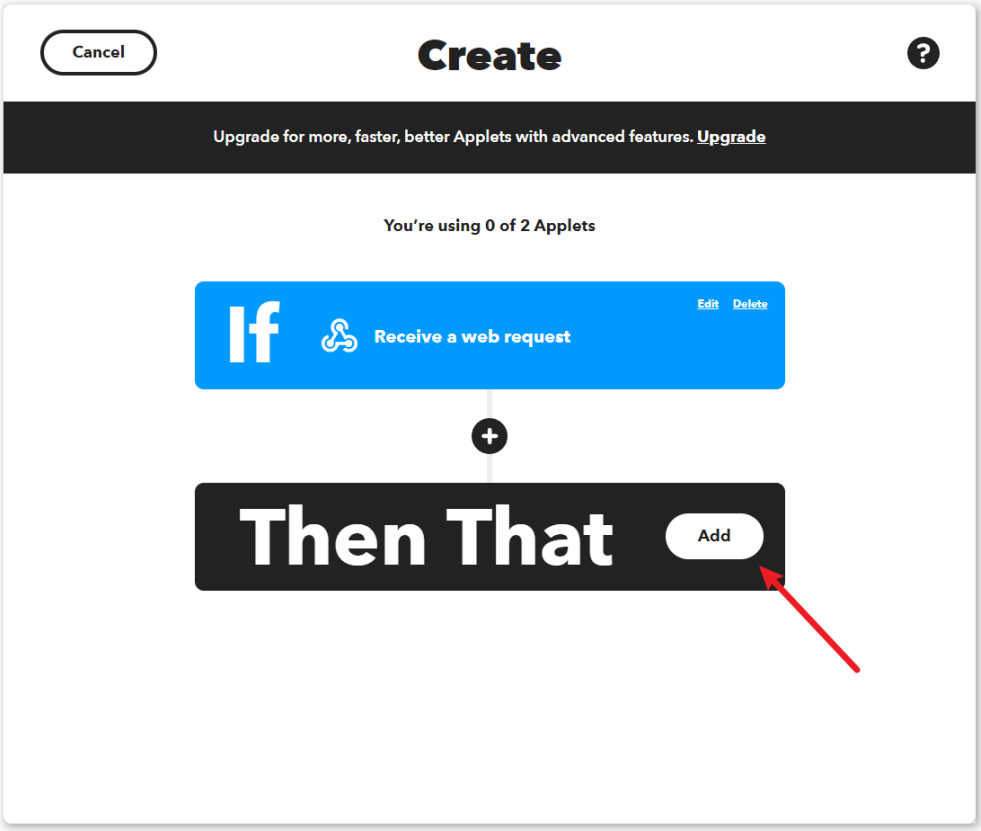


Set the “Event Name” to “vibration_detected”.

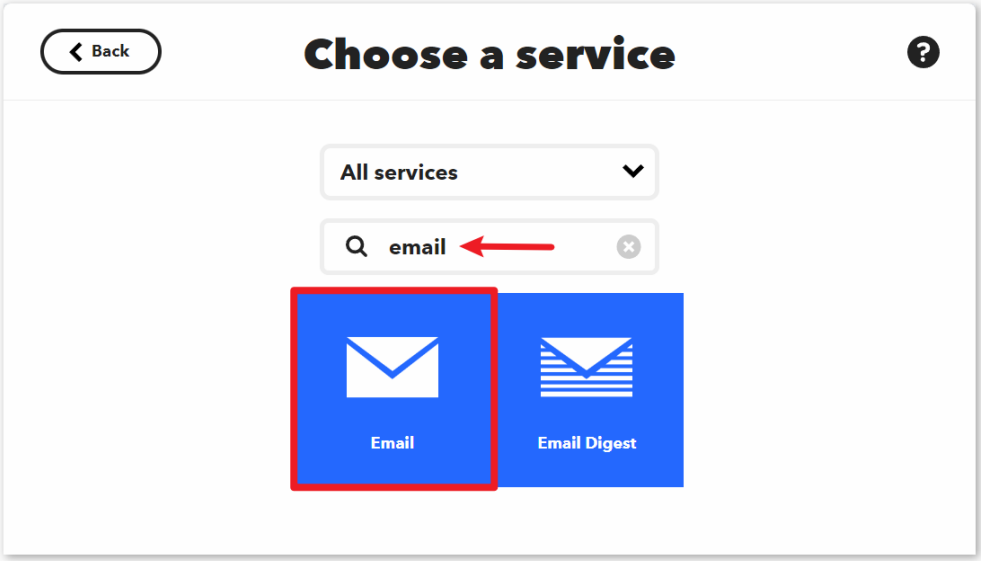


Then That action

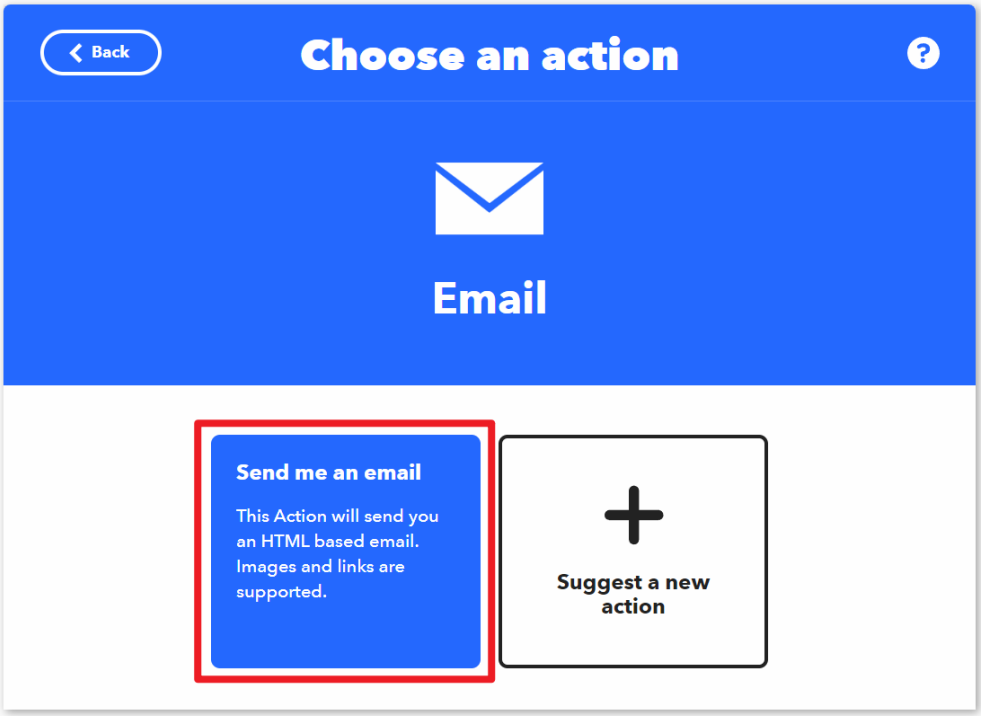
Click on “Add” next to “Then That” to add a action.



Search for "email" and click on "Email".

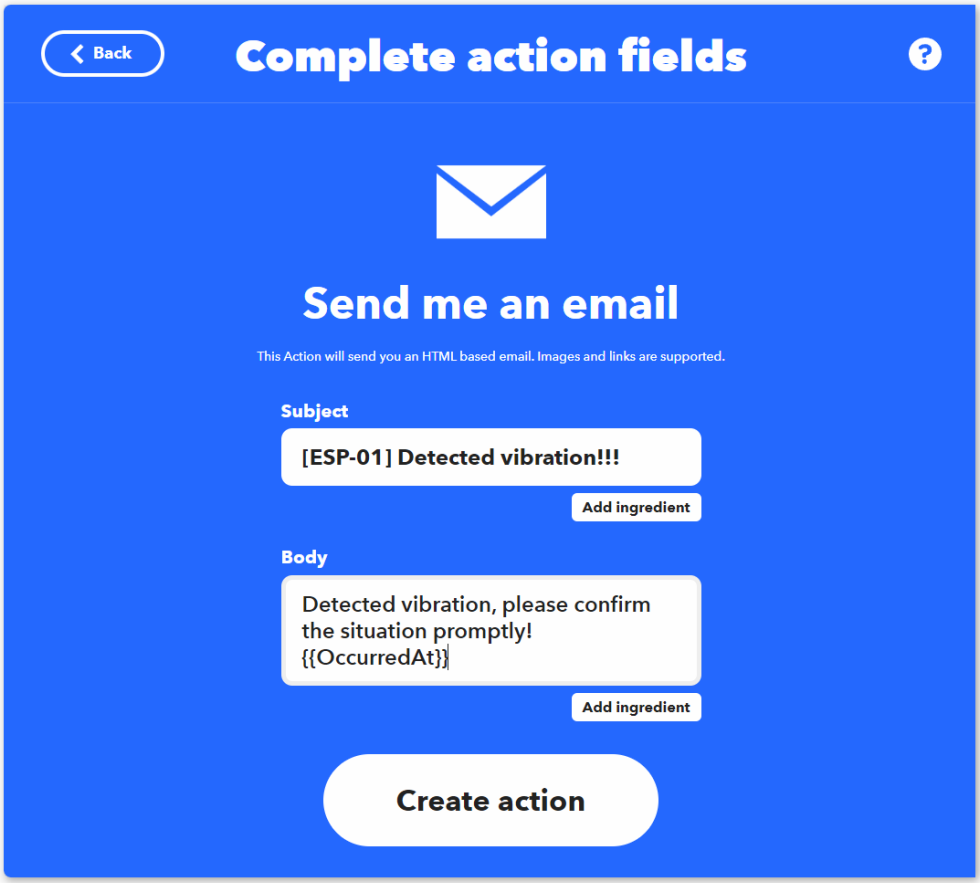


Click on "Send me a email" on the page shown in the following image.

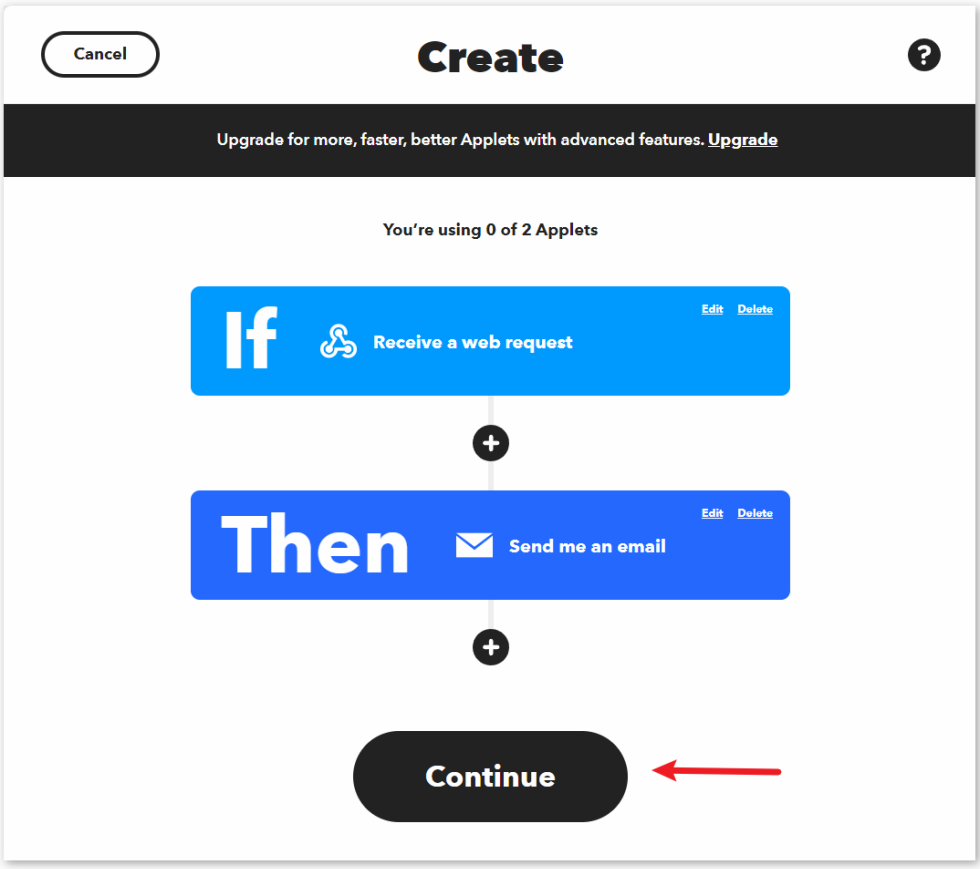


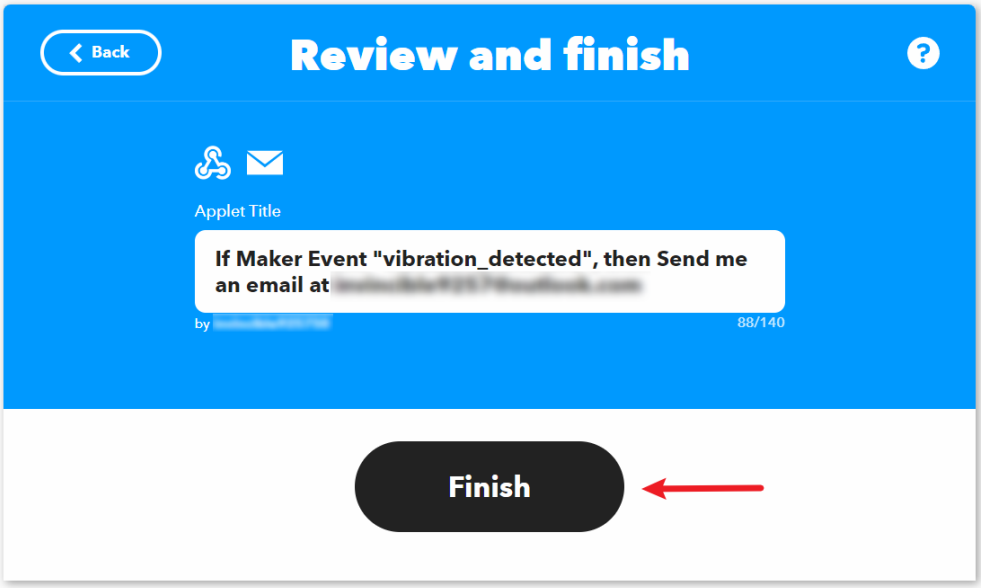
Set the subject and content of the email to be sent when vibration is detected.

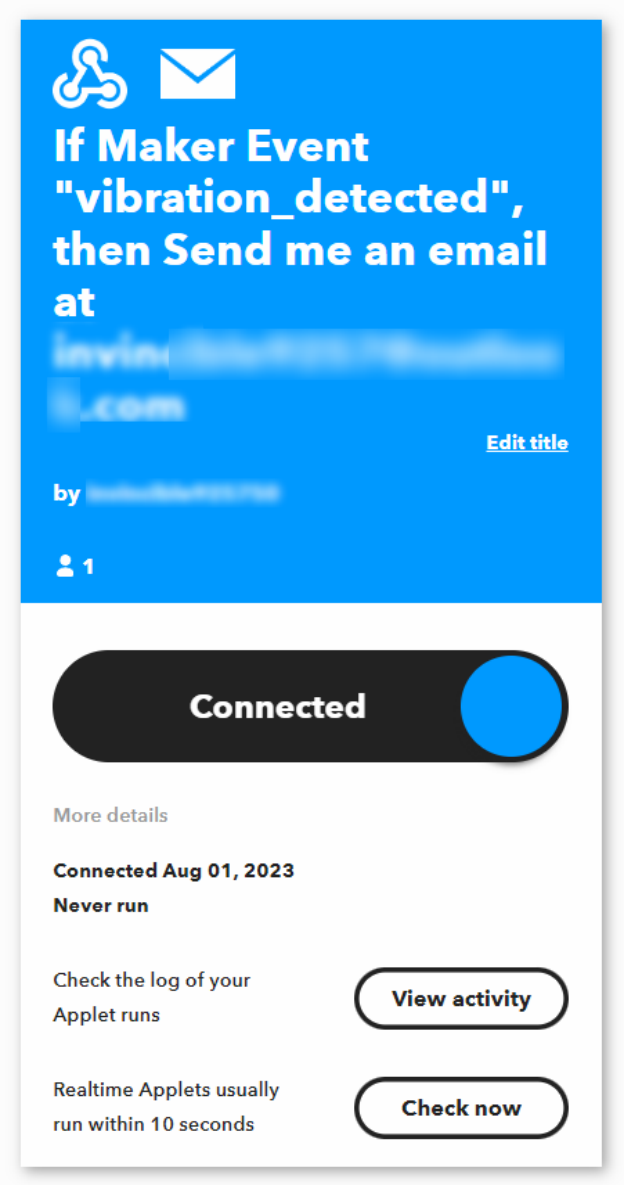
As a reference, the subject is set to "[ESP-01] Detected vibration!!!", and the content is set to "Detected vibration, please confirm the situation promptly! {{OccurredAt}}". When sending an email, `{{OccurredAt}}` will be automatically replaced with the time when the event occurred.



According to the following steps, complete the creation of the Applet.


Cancel
Create
Upgrade for more, faster, better Applets with advanced features. Upgrade
You're using 0 of 2 Applets
If
Receive a web request
Edit
Delete
Then
Send me an email
Edit
Delete
Continue


Back
Review and finish
Applet Title
If Maker Event "vibration_detected", then Send me an email at
by
88/140
Finish



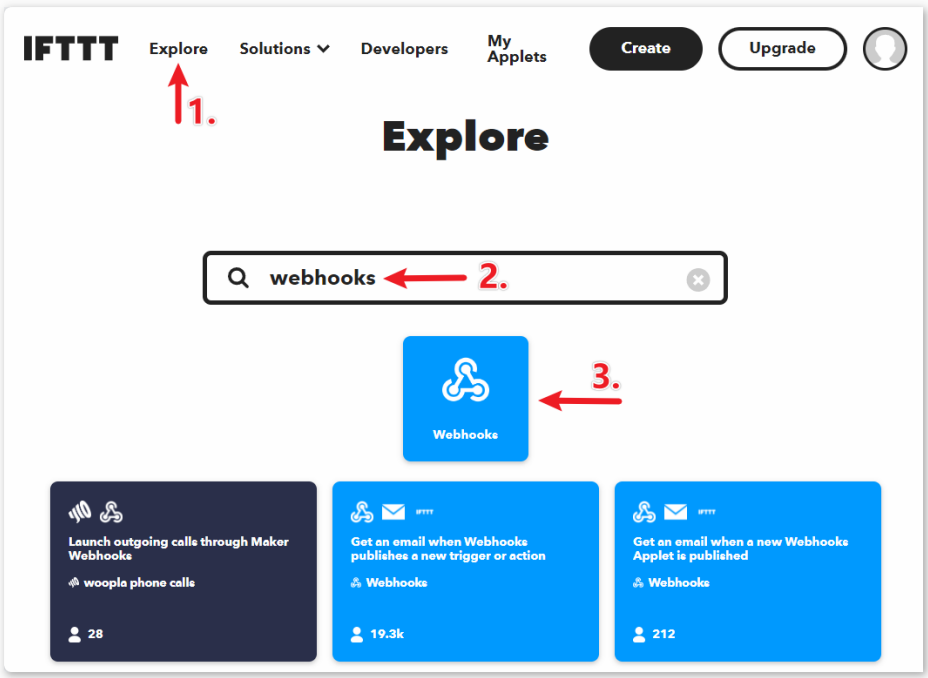
Code

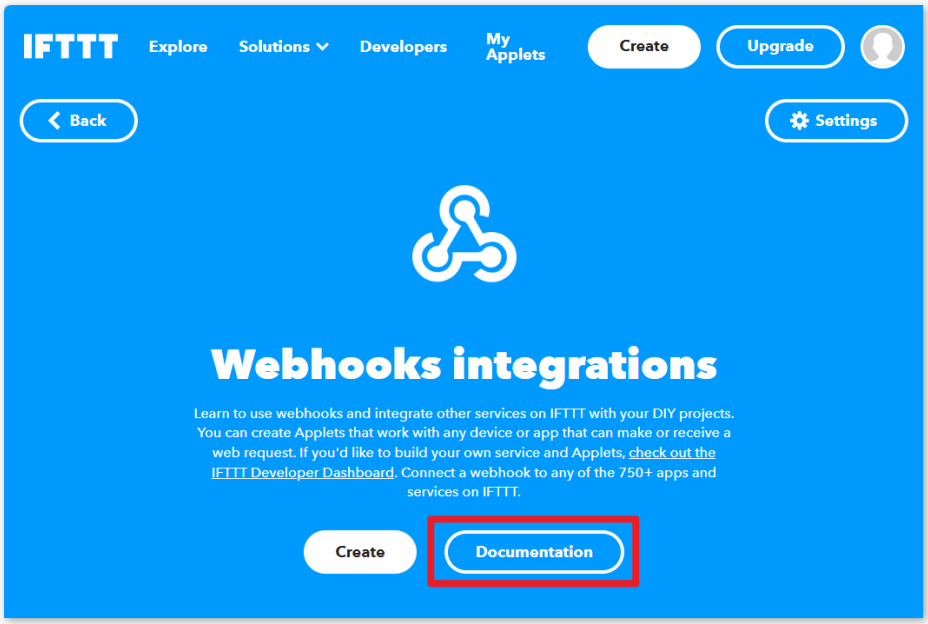
1. Open the `Lesson_49_Vibration_alert_system.ino` file under the path of `universal-maker-sensor-kit\arduino_uno\Lesson_49_Vibration_alert_system`, or copy this code into **Arduino IDE**.
2. You need to enter the `mySSID` and `myPWD` of the WiFi you are using.

```
String mySSID = "your_ssid";     // WiFi SSID
String myPWD = "your_password";  // WiFi Password
```

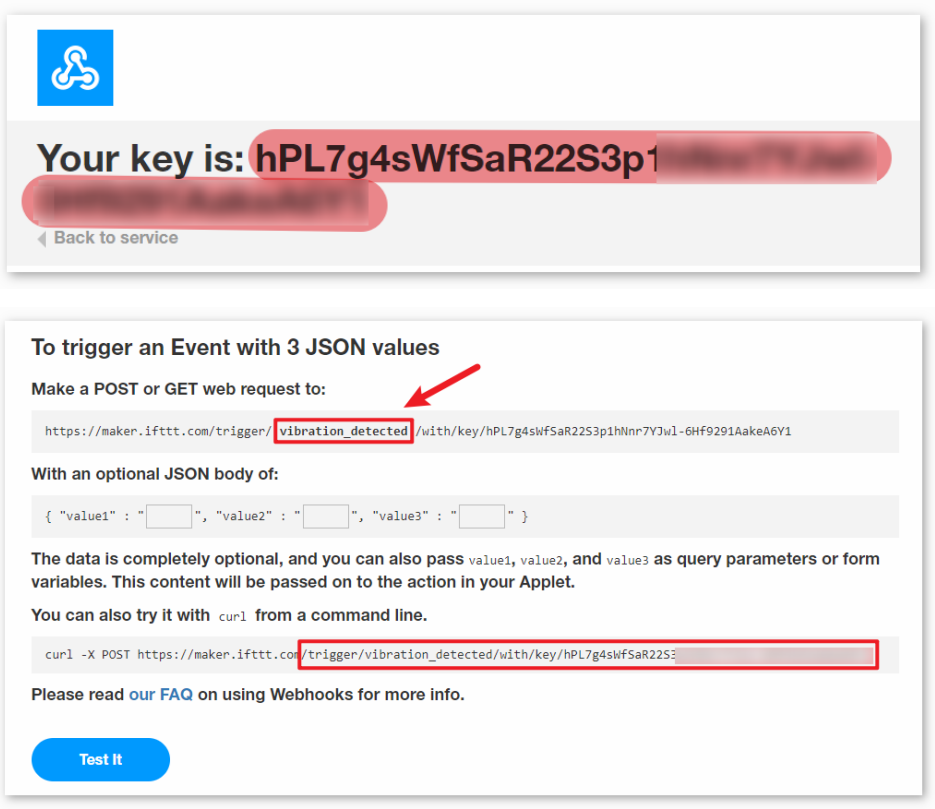
3. You also need to modify the `URL` with both the event name you set and your API key.

```
String URL = "/trigger/vibration_detected/with/key/xxxxxxxxxxxxxxxxx";
```

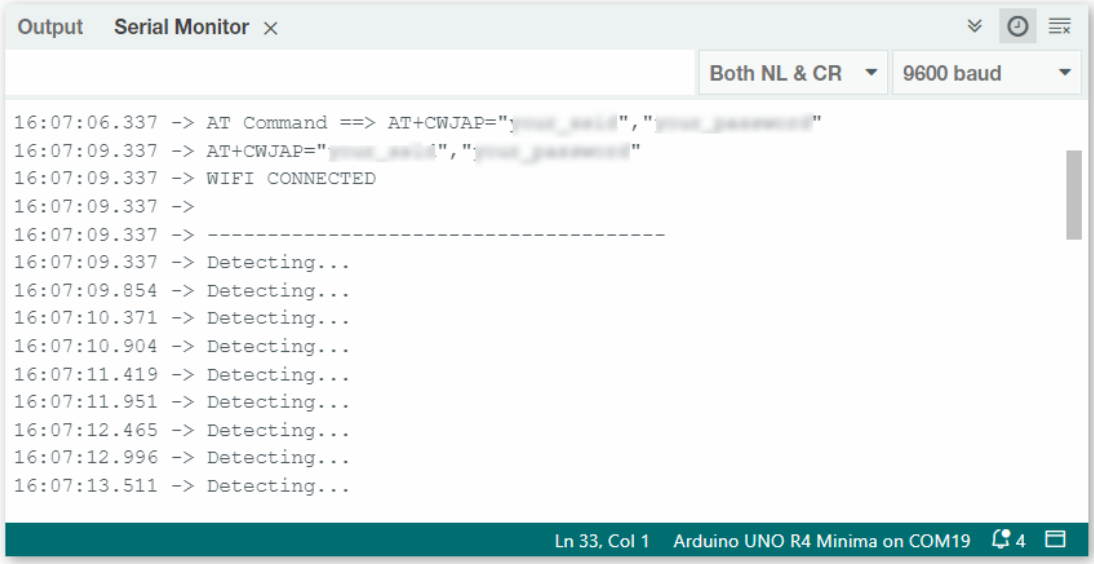




Here you can find **your unique API KEY that you must keep private**. Type in the event name as `vibration_detected`. Your final URL will appear at the bottom of the webpage. Copy this URL.



4. After selecting the correct board and port, click the **Upload** button.
5. Open the Serial monitor(set baudrate to **9600**) and wait for a prompt such as a successful connection to appear.



Code Analysis

The ESP8266 module that comes with the kit is already pre-burned with AT firmware. Therefore, the ESP8266 module can be controlled through AT commands. In this project, we use software serial to enable communication between the Arduino Uno board and the ESP8266 module. The Arduino Uno board sends AT commands to the ESP8266 module for network connection and sending requests. You can refer to .

The Uno board reads sensor values and sends AT commands to the ESP8266 module. The ESP8266 module connects to a network and sends requests to IFTTT servers.

1. Include SoftwareSerial library for serial communication between Arduino and ESP8266

```
#include <SoftwareSerial.h>
SoftwareSerial espSerial(2, 3);
```

2. Configure WiFi credentials and IFTTT server details

```
String mySSID = "your_ssid";
String myPWD = "your_password";
String myHOST = "maker.ifttt.com";
String myPORT = "80";
String URL = "/trigger/xxx/with/key/xxxx";
```

3. Define variables for the vibration sensor and alert frequency control

```
unsigned long lastAlertTime = 0;
const unsigned long postingInterval = 120000L;
const int sensorPin = 7;
```

4. In `setup()`, initialize serial communication, ESP8266 module and connect to WiFi

```
void setup() {
  Serial.begin(9600);
  espSerial.begin(115200);

  // Initialize the ESP8266 module
  sendATCommand("AT+RST", 1000, DEBUG);   //Reset the ESP8266 module
  sendATCommand("AT+CWMODE=1", 1000, DEBUG);  //Set the ESP mode as station mode
  sendATCommand("AT+CWJAP=\"" + mySSID + "\",\"" + myPWD + "\"", 3000, DEBUG);  //
↪Connect to WiFi network

  while (!espSerial.find("OK")) {
    //Wait for connection
  }
}
```

5. In `loop()`, detect vibration and send alert if time interval has passed

```
void loop() {

  if (digitalRead(sensorPin)) {
    if (lastAlertTime == 0 || millis() - lastAlertTime > postingInterval) {
      Serial.println("Detected vibration!!!");
      sendAlert();  //Send an HTTP request to IFTTT server
    } else {
```

(continues on next page)

(continued from previous page)

```
      Serial.print("Detected vibration!!! ");
      Serial.println("Since an email has been sent recently, no warning email will␣
↪be sent this time to avoid bombarding your inbox.");
    }
  } else {
    if (DEBUG) {
      Serial.println("Detecting...");
    }
  }
  delay(500);
}
```

6. sendAlert() constructs HTTP request and sends it via ESP8266

```
void sendAlert() {

  String sendData = "GET " + URL + " HTTP/1.1" + "\r\n";
  sendData += "Host: maker.ifttt.com\r\n";

  sendATCommand("AT+CIPMUX=0",1000,DEBUG);
  sendATCommand("AT+CIPSTART=...",3000,DEBUG);
  sendATCommand("AT+CIPSEND=" + String(sendData.length()),1000,DEBUG);
  espSerial.println(sendData);

}
```

7. Handling AT Commands sendATCommand()

   This function sends AT commands to the ESP8266 and collects responses.

```
void sendATCommand(String command, const int timeout, boolean debug) {
  // Print and send command
  Serial.print("AT Command ==> ");
  Serial.print(command);
  Serial.println();
  espSerial.println(command);  // Send the AT command

  // Get the response from the ESP8266 module
  String response = "";
  long int time = millis();
  while ((time + timeout) > millis()) {  // Wait for the response until the timeout
    while (espSerial.available()) {
      char c = espSerial.read();
      response += c;
    }
  }

  // Print response if debug mode is on
  if (debug) {
    Serial.println(response);
    Serial.println("--------------------------------------");
  }
```

Reference

-
-
-

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.54 Lesson 50: Flame Alert System with Blynk

In this chapter, we will guide you through the process of creating a home flame alarm system demo using Blynk. By utilizing a flame sensor, you can detect potential fires in your home. Sending the detected values to Blynk allows for remote monitoring of your home via the internet. In case of a fire, Blynk will promptly notify you via email.

Required Components

In this project, we need the following components.

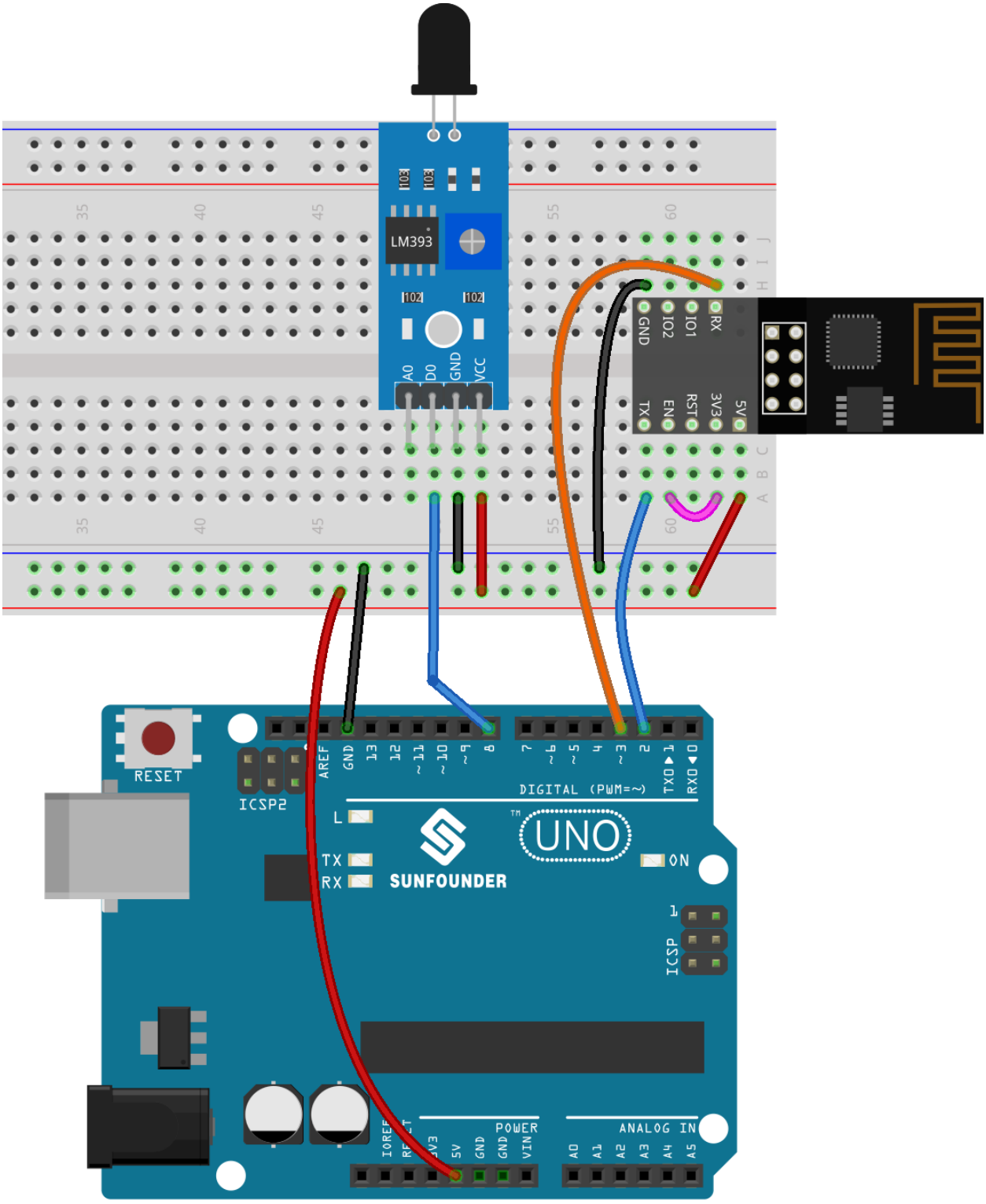
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Breadboard	
ESP8266 Module	-
Flame Sensor Module	-

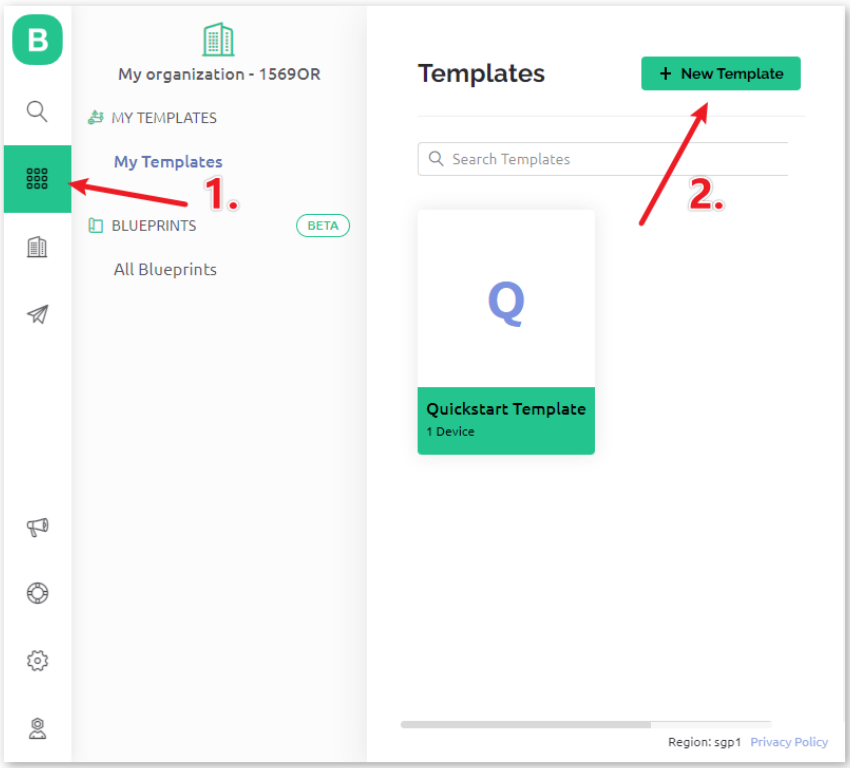
Wiring


LM393
A0
D0
GND
VCC
GND
IO2
IO1
RX
TX
EN
RST
3V3
5V
RESET
ICSP2
AREF
GND
13
12
11
10
9
8
7
6
5
4
3
2
TX0
RX0
DIGITAL (PWM=~)
L
TX
RX
SUNFOUNDER
UNO
ON
ICSP
POWER
ANALOG IN
IOREF
RESET
3V3
5V
GND
GND
VIN
A0
A1
A2
A3
A4
A5

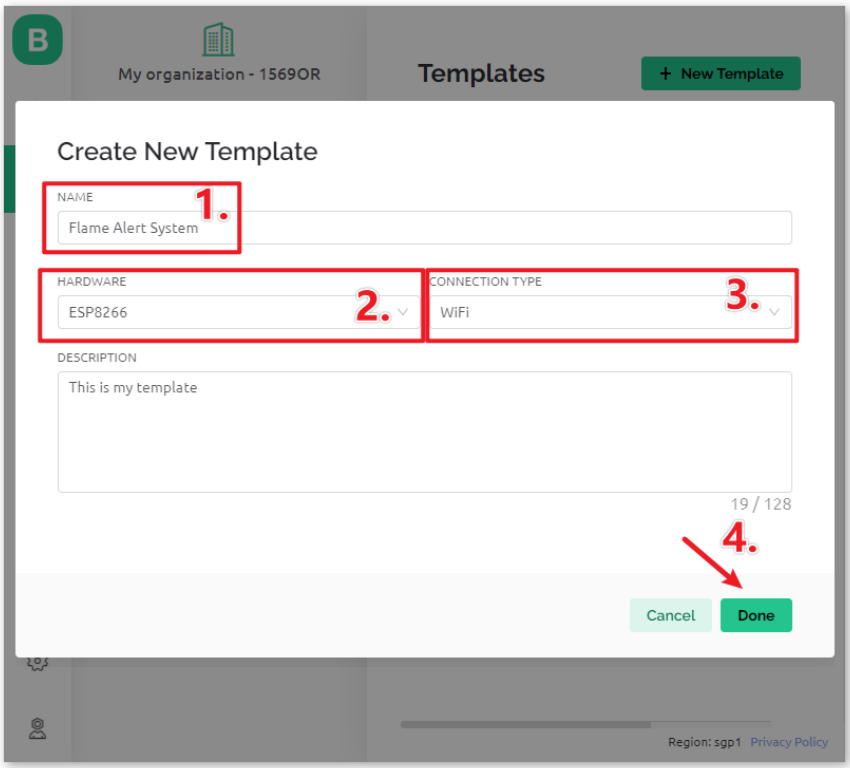
Configure Blynk

1 Create template

Firstly, we need to establish a template on Blynk. Follow the steps below to create a **"Flame Alert System"** template.

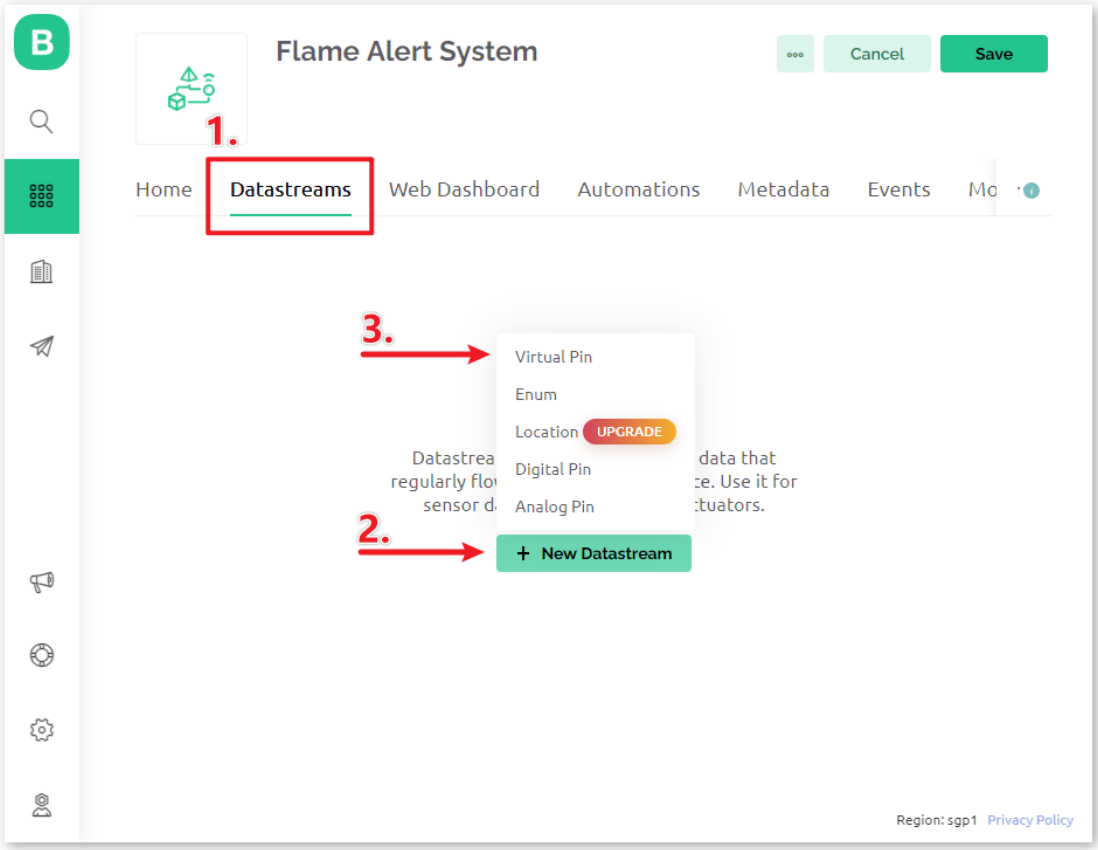


Ensure that the **HARDWARE** is configured as **ESP8266** and the **CONNECT TYPE** is set to **WiFi**.

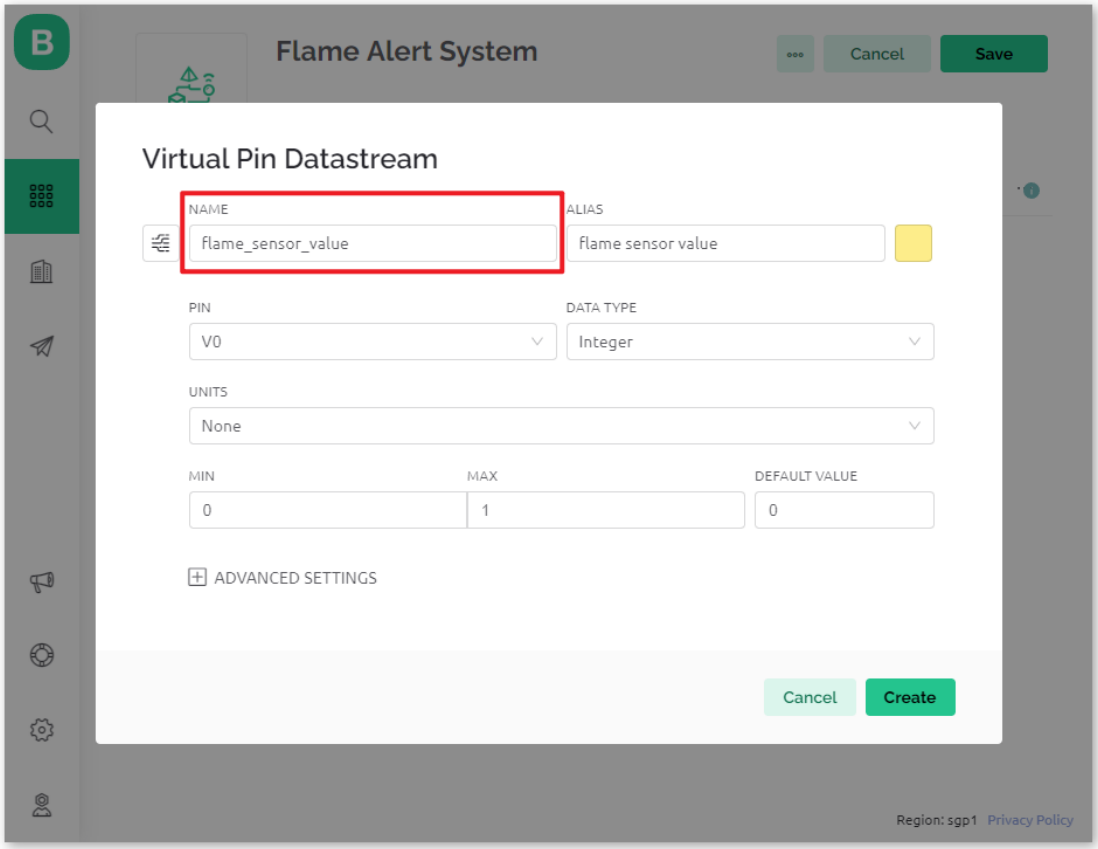


2 Datastream

Create a **Datastream** of type **Virtual Pin** in the **Datastream** page to get the value of Flame sensor module.

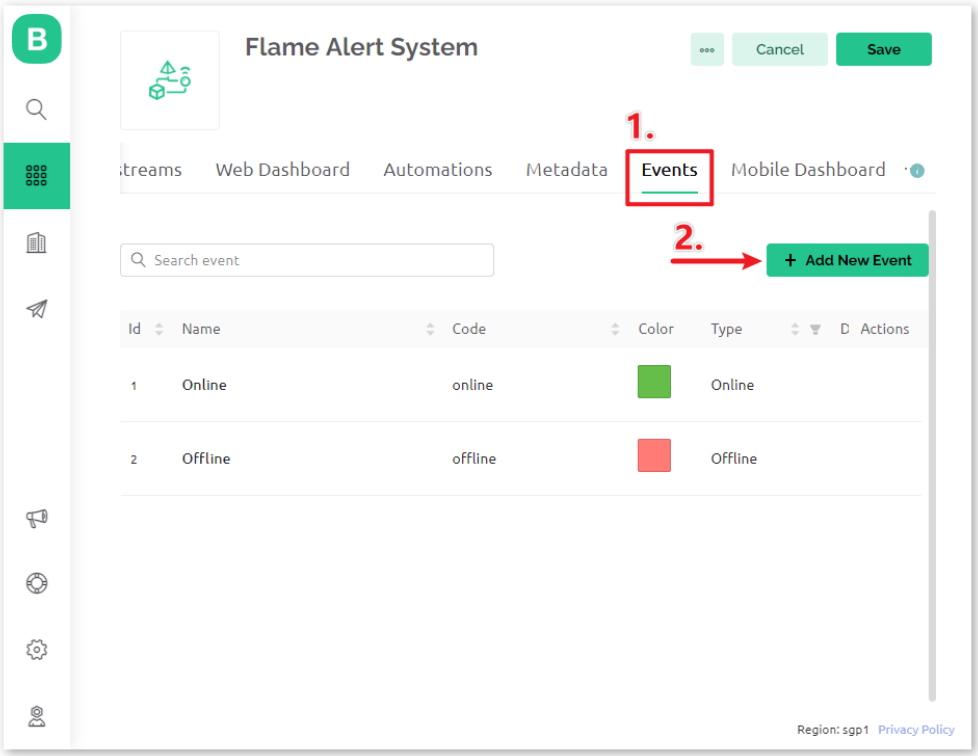


Set the name of the **Virtual Pin** to `flame_sensor_value`. Set the **DATA TYPE** to **Integer** and MIN and MAX to **0** and **1**.



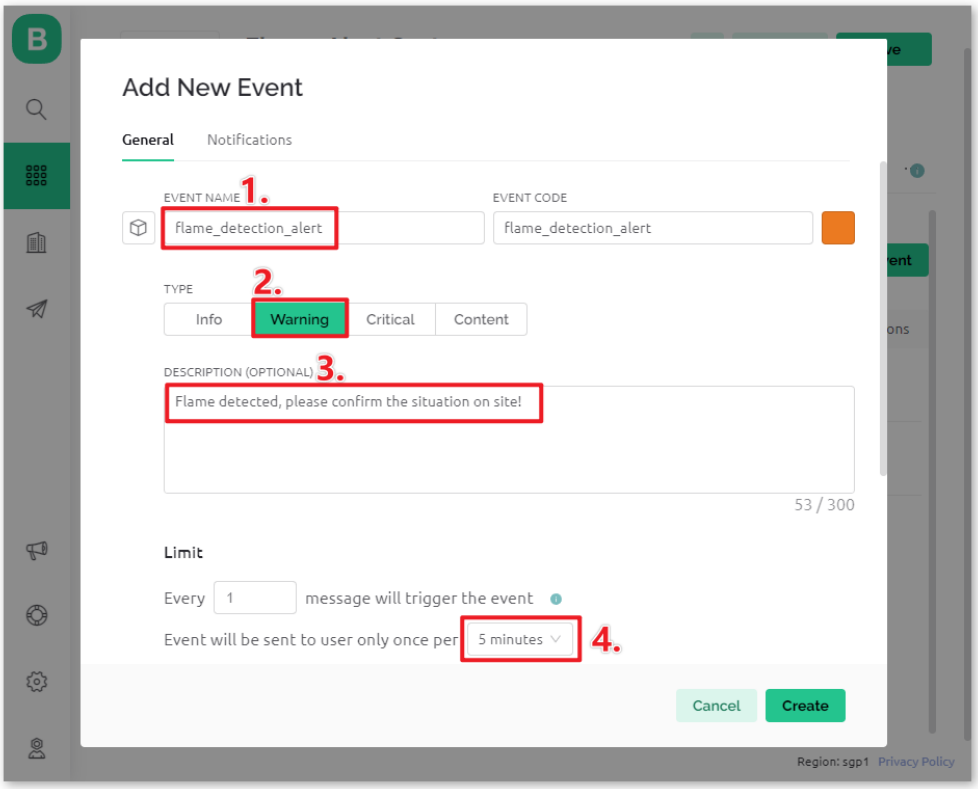
3 Event

Next, we will create an **event** that logs the detection of flames and sends an email notification.

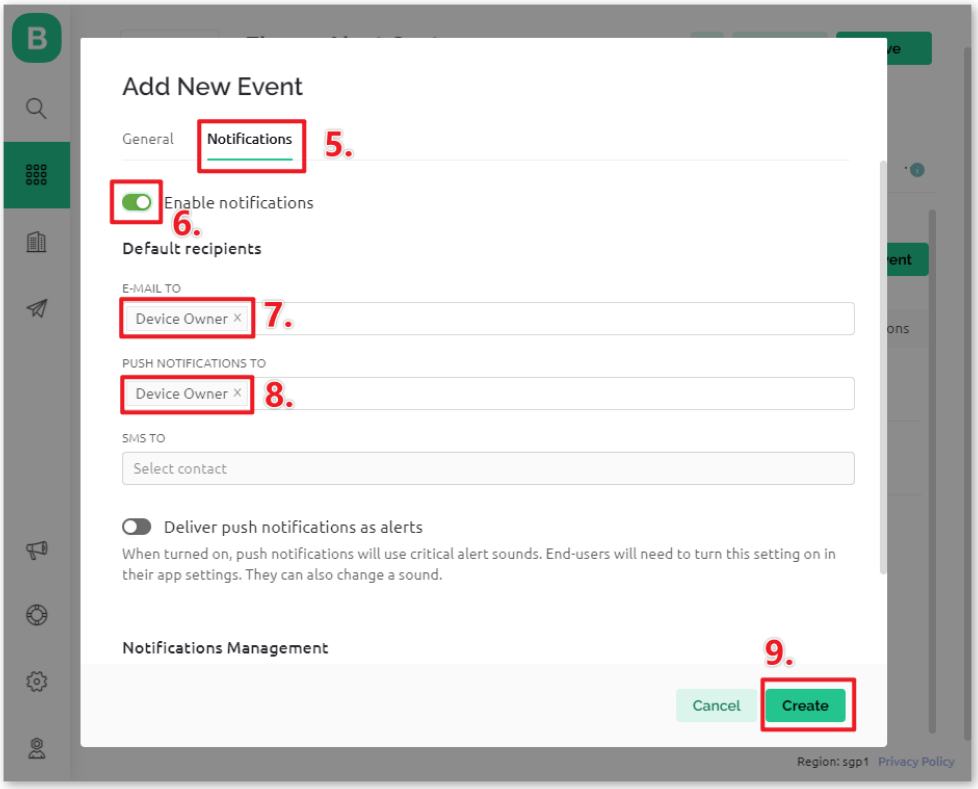


Note: It is recommended to keep it consistent with my settings, otherwise you may need to modify the code to run the project.

Set **EVENT NAME** to `flame_detection_alert`. At the same time, you can customize the content of email sent by setting **DESCRIPTION** for event triggering. You can also set frequency limits for event triggering below.



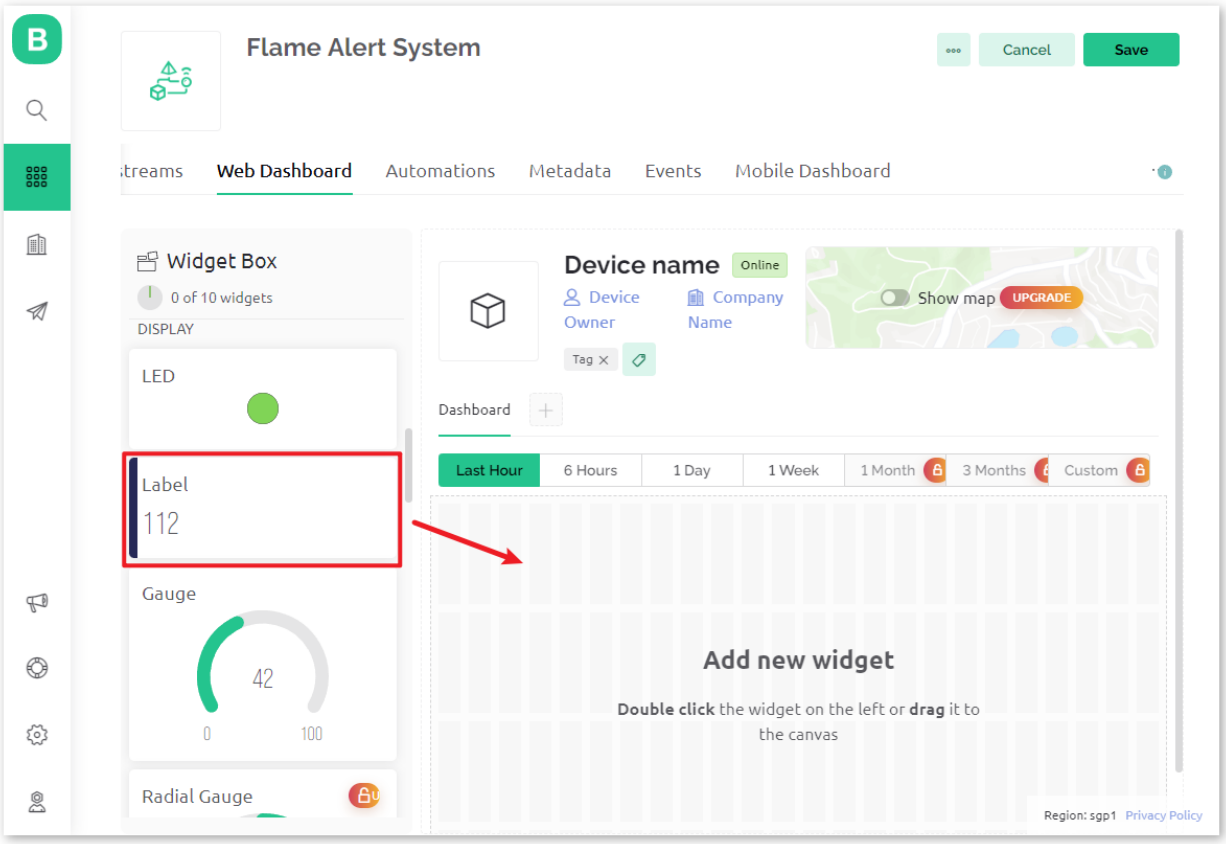
Go to the **Notifications** page and configure email settings.



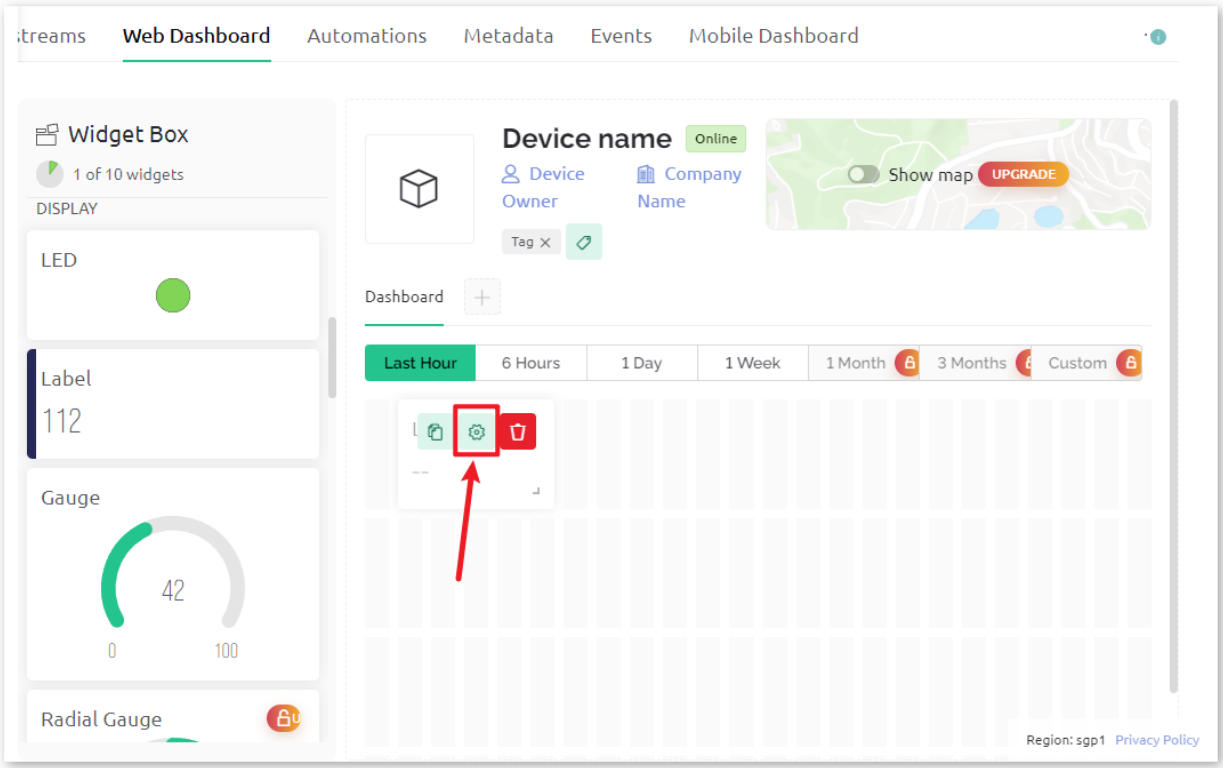
4 Web Dashboard

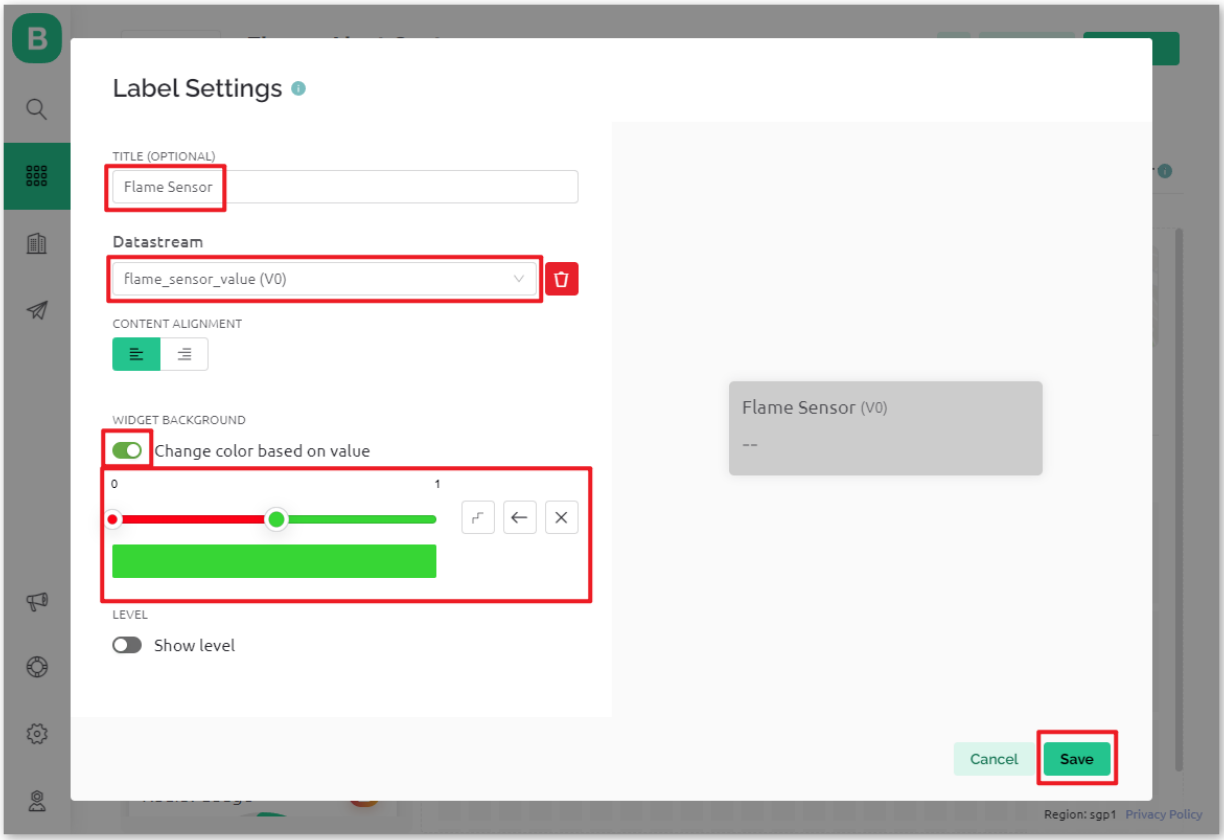
We also need to set up the **Web Dashboard** to display the sensor data sent from the Uno board.

Drag and drop an **Label widget** on the **Web Dashboard** page.



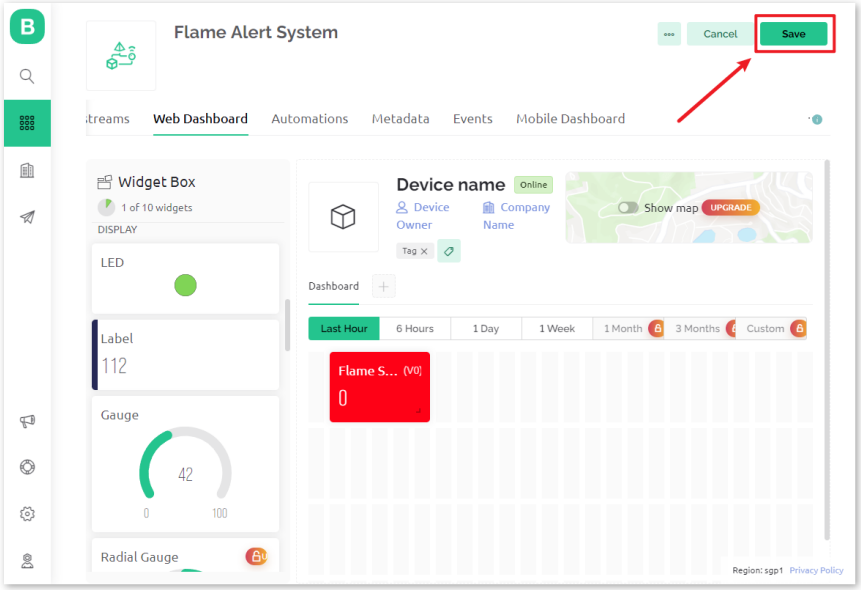
In the settings page of the **Label widget**, select **Datastream** as **flame_sensor_value(V0)**. Then set the color of **WIDGET BACKGROUND** to change with the value of data. When the displayed value is 1, it will be shown in green. When the value is 0, it will be shown in red.


Streams
Web Dashboard
Automations
Metadata
Events
Mobile Dashboard
Widget Box
1 of 10 widgets
DISPLAY
LED
Label
112
Gauge
42
0
100
Radial Gauge
Device name
Online
Device Owner
Company Name
Tag
Show map
UPGRADE
Dashboard
Last Hour
6 Hours
1 Day
1 Week
1 Month
3 Months
Custom
Region: sgp1
Privacy Policy

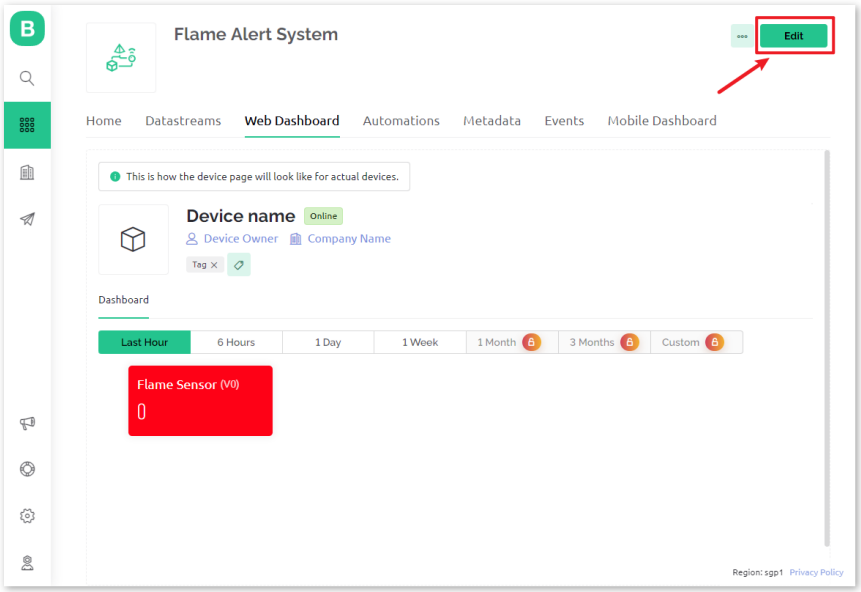

Label Settings
TITLE (OPTIONAL)
Flame Sensor
Datastream
flame_sensor_value (V0)
CONTENT ALIGNMENT
WIDGET BACKGROUND
Change color based on value
0
1
LEVEL
Show level
Flame Sensor (V0)
--
Cancel
Save
Region: sgp1
Privacy Policy

5 Save template

At last, remember to save the template.



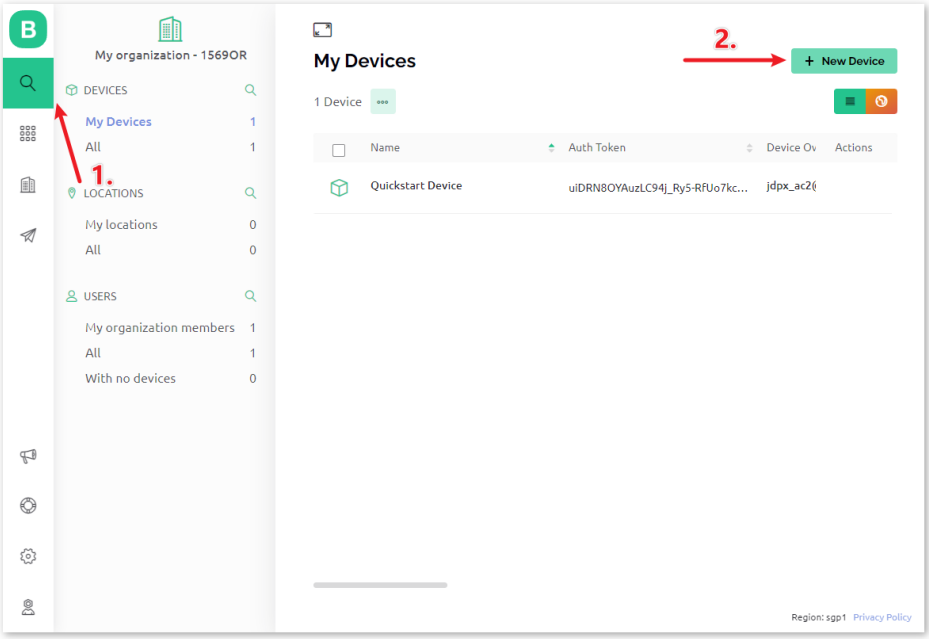
In case you need to edit the template, you can click on the edit button in the upper right corner.

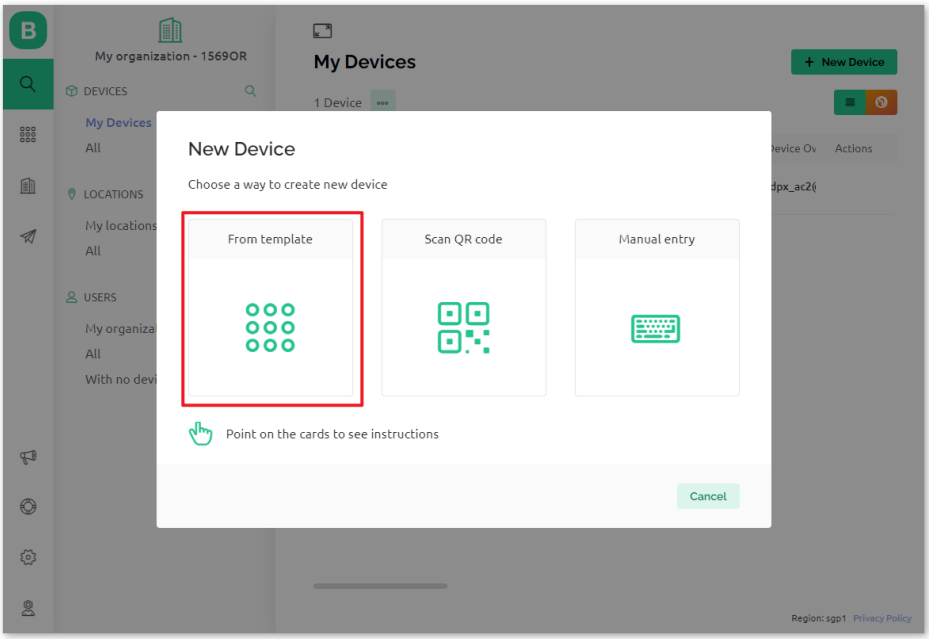


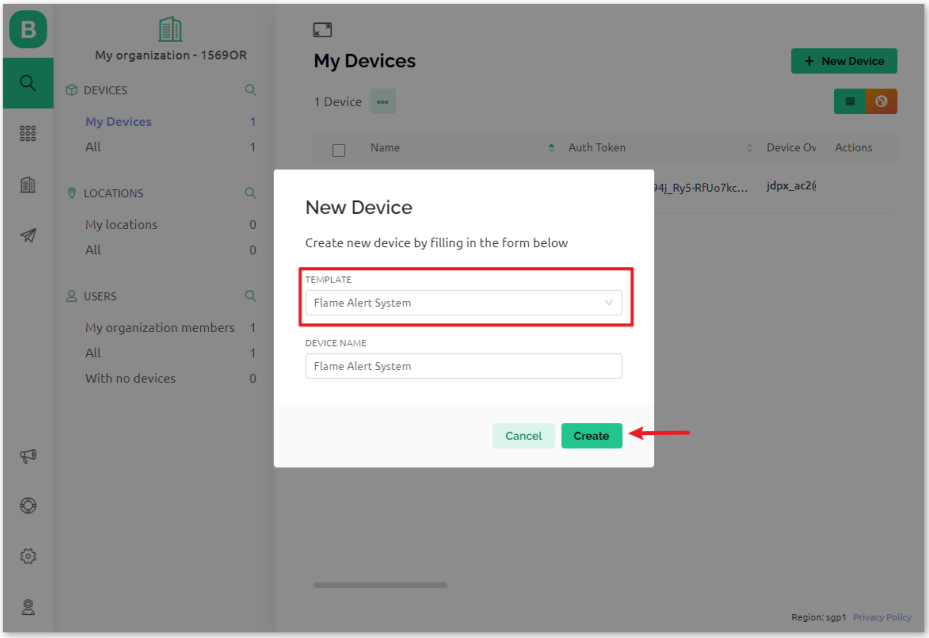
Code

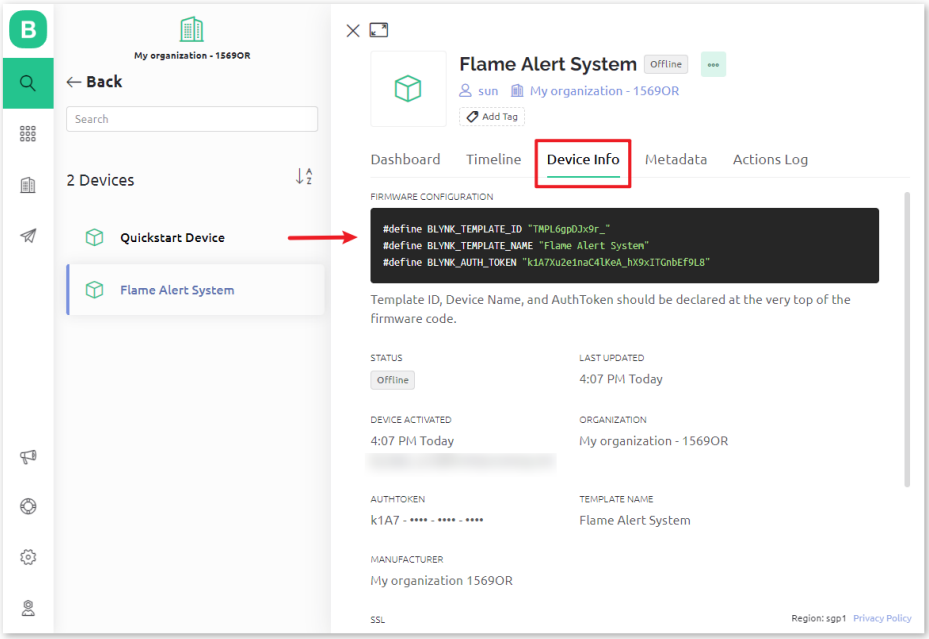
1. Open the `Lesson_50_Flame_alert_system.ino` file under the path of `universal-maker-sensor-kit\arduino_uno\Lesson_50_Flame_alert_system`, or copy this code into **Arduino IDE**.
2. Create a Blynk device using the Flame Detection Alert template. Then, replace the `BLYNK_TEMPLATE_ID`, `BLYNK_TEMPLATE_NAME`, and `BLYNK_AUTH_TOKEN` with your own.

```
#define BLYNK_TEMPLATE_ID "TMPxxxxxxx"
#define BLYNK_TEMPLATE_NAME "Flame Alert System"
#define BLYNK_AUTH_TOKEN "xxxxxxxxxxxx"
```





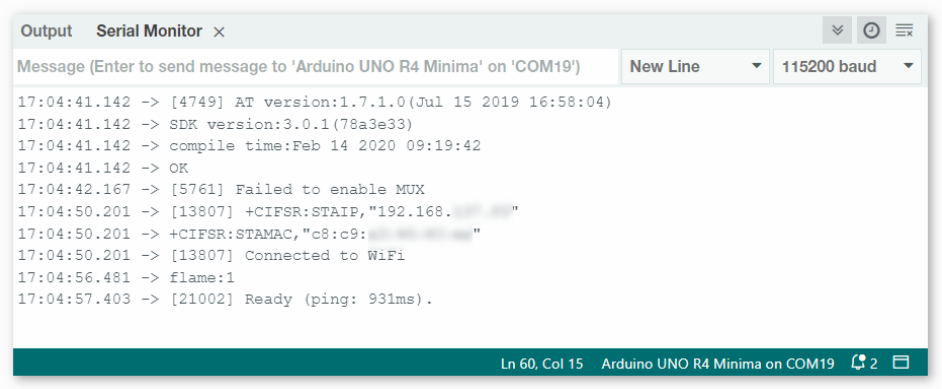




3. You also need to enter the `ssid` and `password` of the WiFi you are using.

```
char ssid[] = "your_ssid";
char pass[] = "your_password";
```

4. After selecting the correct board and port, click the **Upload** button.
5. Open the Serial monitor(set baudrate to 115200) and wait for a prompt such as a successful connection to appear.

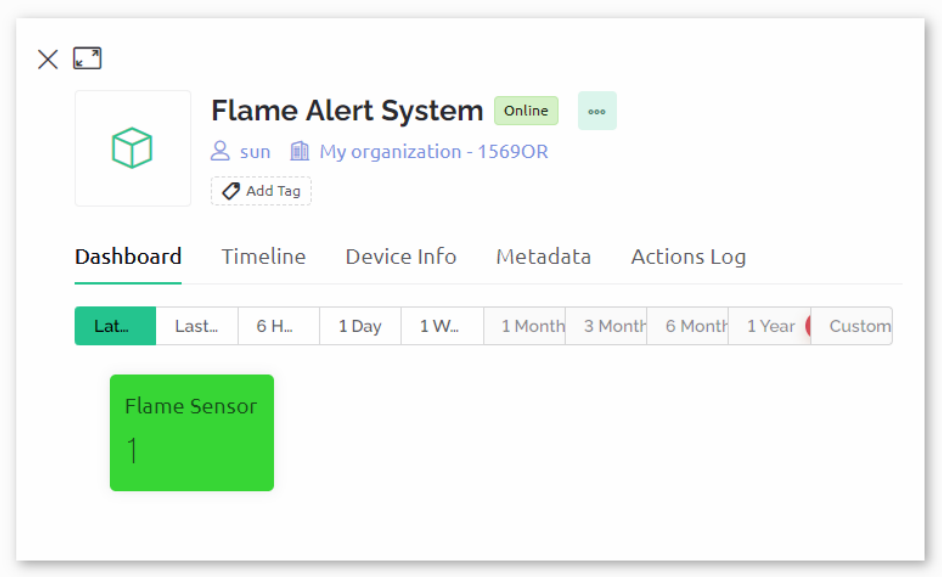


Note: If the message `ESP is not responding` appears when you connect, please follow these steps.

- Make sure the 9V battery is plugged in.
- Reset the ESP8266 module by connecting the pin RST to GND for 1 second, then unplug it.
- Press the reset button on the R4 board.

Sometimes, you may need to repeat the above operation 3-5 times, please be patient.

6. Now, Blynk will show the data read from flame sensor. In the label widget, you can see the value read by the flame sensor. When the displayed value is 1, the background of the label will be shown in green. When the value is 0, the background of the label will be shown in red and Blynk will send you an alert email.



7. If you want to use Blynk on mobile devices, please refer to *How to use Blynk on mobile device?*.

Code Analysis

1. **Library Initialization**

   Before we start, it's crucial to set up the necessary libraries and settings for communication between the Arduino, ESP8266 WiFi module, and Blynk app. This code sets up the required libraries and configures a software serial connection between the Arduino and ESP8266 module, with the appropriate baud rate for data transmission.

```
//Set debug prints on Serial Monitor
#define BLYNK_PRINT Serial

#include <ESP8266_Lib.h>               // Library for ESP8266
#include <BlynkSimpleShieldEsp8266.h>  // Library for Blynk

// Software Serial on Uno
#include <SoftwareSerial.h>
SoftwareSerial EspSerial(2, 3);  // RX, TX
#define ESP8266_BAUD 115200      // Set the ESP8266 baud rate
ESP8266 wifi(&EspSerial);
```

2. **Blynk and WiFi configuration**

   For the project to communicate with the Blynk app, it needs to connect to a Wi-Fi network. The credentials need to specified here.

```
// Template ID, Device Name and Auth Token are provided by the Blynk Cloud
// See the Device Info tab, or Template settings
#define BLYNK_TEMPLATE_ID "TMPxxxxxx"
#define BLYNK_TEMPLATE_NAME "Flame Alert System"
#define BLYNK_AUTH_TOKEN "xxxxxxxxxxxxxxx"

// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "your_ssid";
char pass[] = "your_password";
```

3. **Sensor Pin & Timer Declaration**

   Define the pin number for the flame. Blynk library provides a built-in timer, and we create a timer object. More about

```
const int sensorPin = 8;
BlynkTimer timer;
```

4. **setup() Function**

   Initial configurations such as setting the pin mode for the sensorPin, initiating serial communication, setting the BlynkTimer, and connecting to the Blynk app are done in this function.

   - We use `timer.setInterval(1000L, myTimerEvent)` to set the timer interval in setup(), here we set to execute the `myTimerEvent()` function every **1000ms**. You can modify the first parameter of `timer.setInterval(1000L, myTimerEvent)` to change the interval between `myTimerEvent` executions.

```
void setup() {
  pinMode(sensorPin, INPUT);
  Serial.begin(115200);
```

(continues on next page)

(continued from previous page)

```
  EspSerial.begin(ESP8266_BAUD);
  delay(1000);
  timer.setInterval(1000L, myTimerEvent);
  Blynk.config(wifi,BLYNK_AUTH_TOKEN);
  Blynk.connectWiFi(ssid, pass);
}
```

5. **loop() Function**

   The main loop runs the Blynk and Timer services continuously.

```
void loop() {
  Blynk.run();
  timer.run();
}
```

6. **myTimerEvent() & sendData() Function**

```
void myTimerEvent() {
  // Please don't send more that 10 values per second.
  sendData();  // Call function to send sensor data to Blynk app
}
```

   The `sendData()` function reads the value from the flame sensor and sends it to Blynk. If it detects a flame (value 0), it sends `flame_detection_alert` event to the Blynk app.

   - Use `Blynk.virtualWrite(vPin, value)` to send data to virtual pin V0 on Blynk. More about .
   - Use `Blynk.logEvent("event_code")` to log event to Blynk. More about .

```
void sendData() {
  int data = digitalRead(sensorPin);
  Blynk.virtualWrite(V0, data);  // send data to virtual pin V0 on Blynk
  Serial.print("flame:");
  Serial.println(data);  // Print flame status on Serial Monitor
  if (data == 0) {
    Blynk.logEvent("flame_alert");  // log flame alert event if sensor detects flame
  }
}
```

Reference

-
-
-
-
-

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.

- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.3.55 Lesson 51: Intrusion Alert System with Blynk

This project demonstrate a simple home intrusion detection system using a passive infrared (PIR) sensor (HC-SR501). When the system is set to 'Away' mode through the Blynk app, the PIR sensor monitors for motion. Any detected movement triggers a notification on the Blynk app, alerting the user of potential intrusion.

Required Components

In this project, we need the following components.

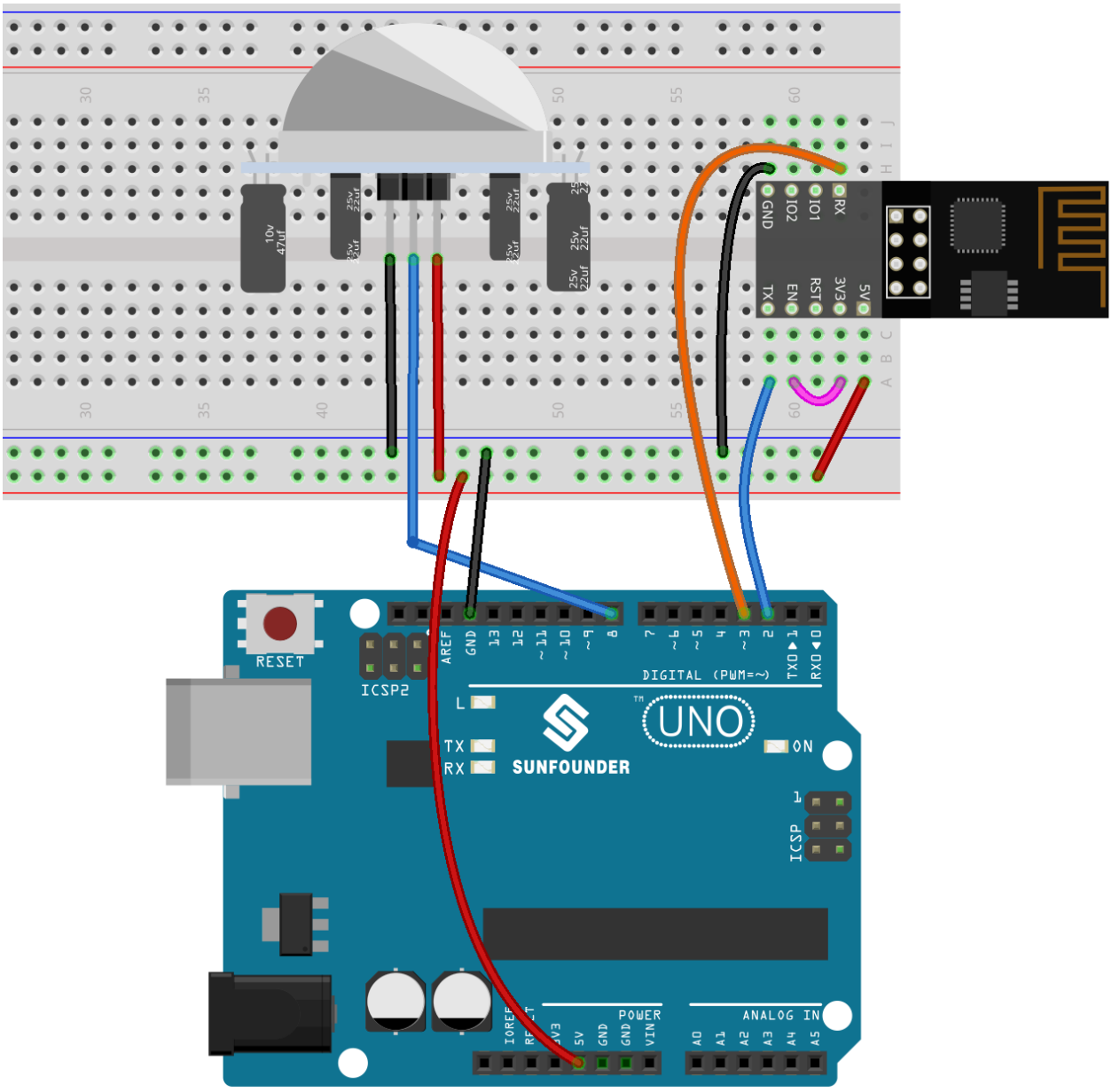
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Arduino UNO R3 or R4	
Breadboard	
ESP8266 Module	-
PIR Motion Module (HC-SR501)	-

Wiring

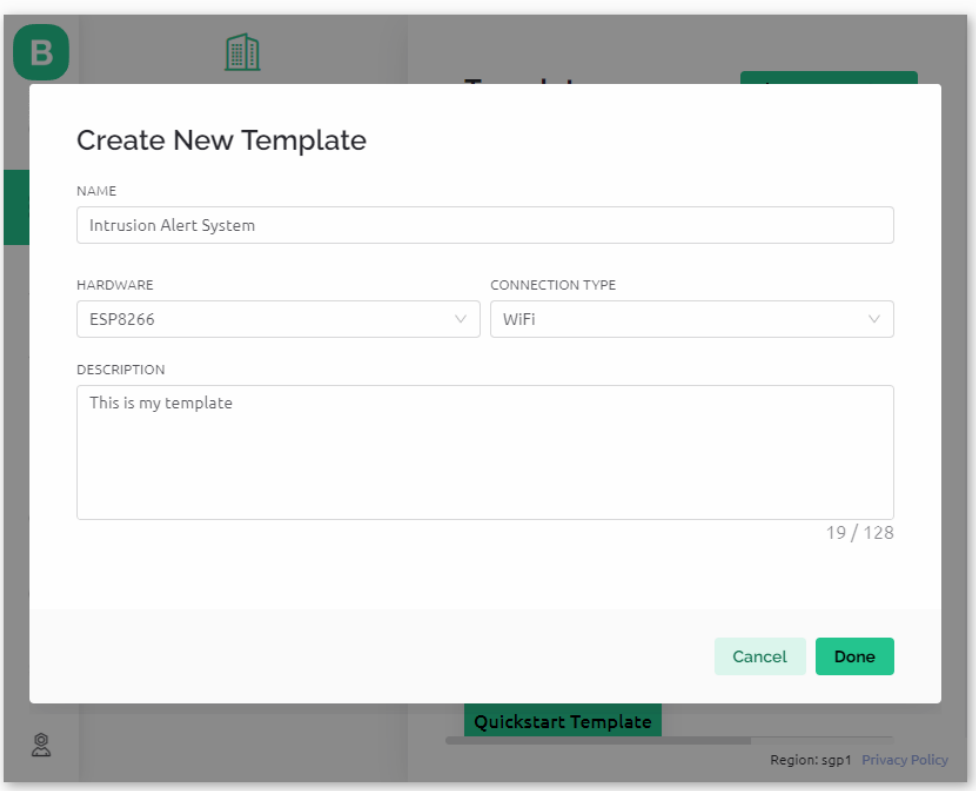

GND
IO2
IO1
RX
TX
EN
RST
3V3
5V
RESET
ICSP2
DIGITAL (PWM=~)
UNO
SUNFOUNDER
ON
POWER
ANALOG IN
ICSP

Configure Blynk

Note: If you are not familiar with Blynk, it is strongly recommended that you read these two tutorials first. *Get Started with Blynk* is a beginner's guide for Blynk, which includes how to configure ESP8266 and register with Blynk. And *Lesson 50: Flame Alert System with Blynk* is a simple example, but the description of the steps will be more detailed.

1 Create template

Firstly, we need to establish a template on Blynk. Follow the steps below to create a **"Intrusion Alert System"** template.

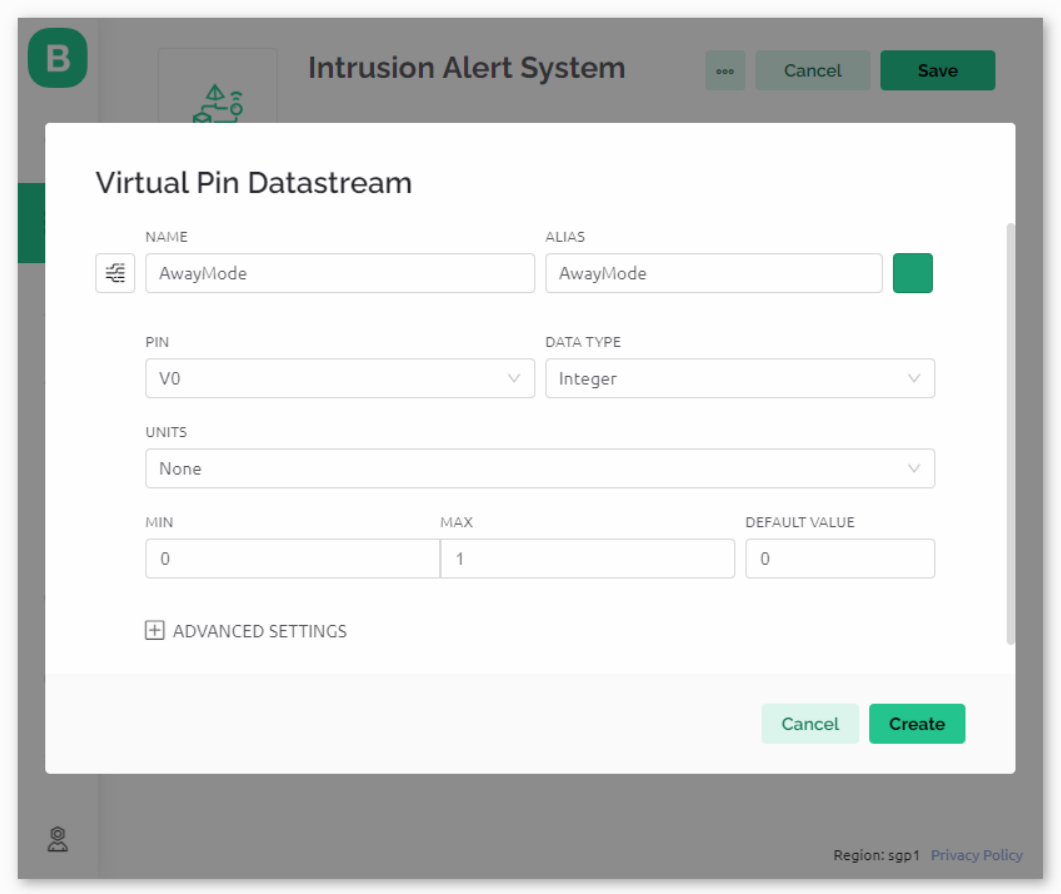


2 Datastream

Create **Datastreams** of type **Virtual Pin** in the **Datastream** page receive data from esp8266 and uno r4 board.

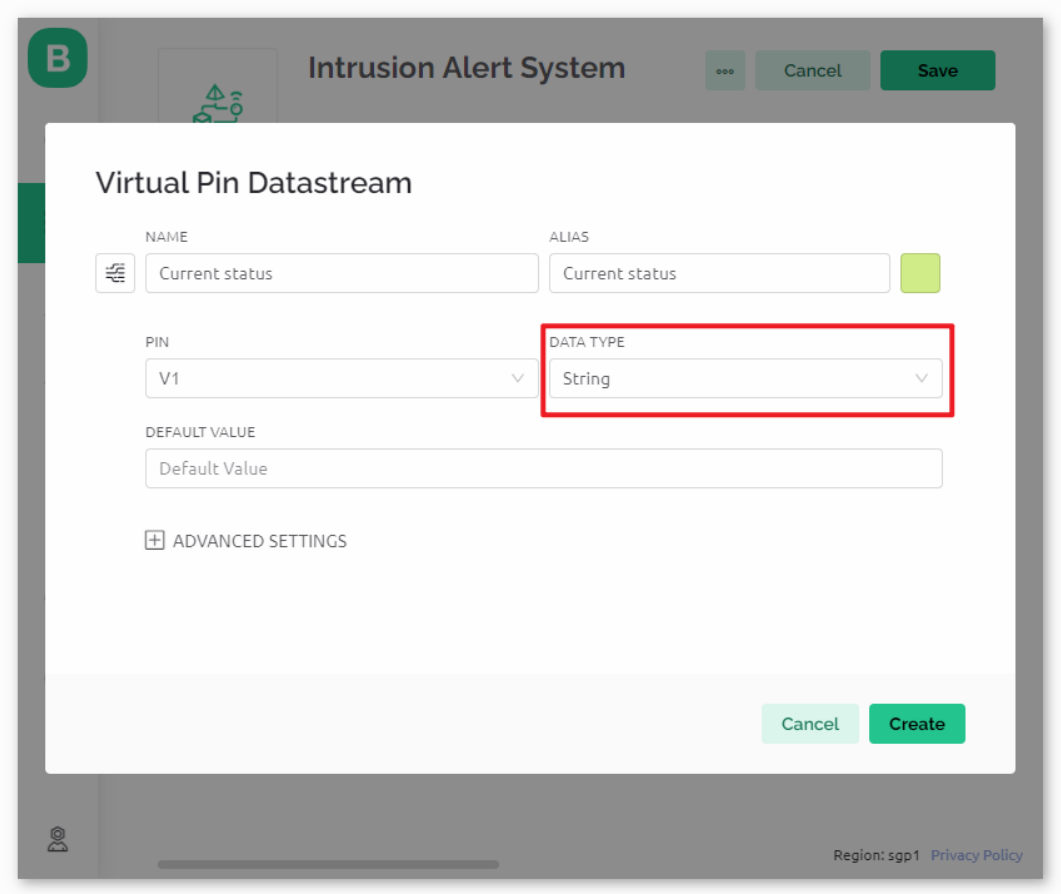
- Create Virtual Pin V0 according to the following diagram:

  Set the name of the **Virtual Pin V0** to **AwayMode**. Set the **DATA TYPE** to **Integer** and MIN and MAX to **0** and **1**.

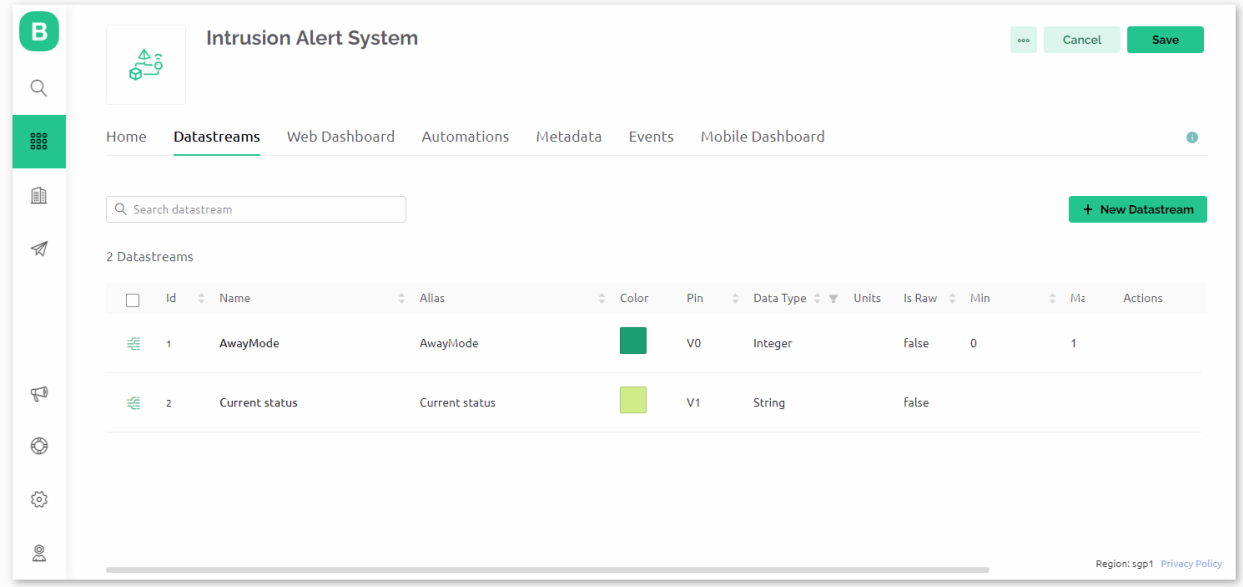


- Create Virtual Pin V1 according to the following diagram:

  Set the name of the **Virtual Pin V1** to **Current status**. Set the **DATA TYPE** to **String**.



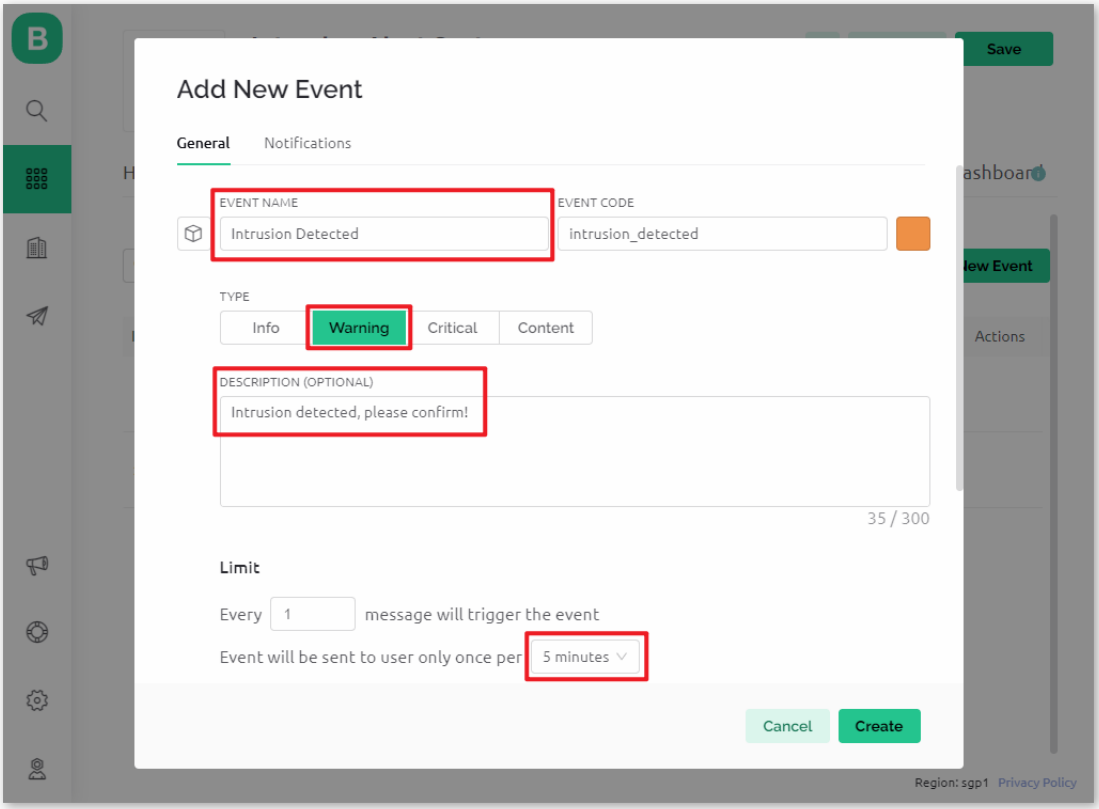
Make sure that you have set up two Virtual Pins according to the steps above.



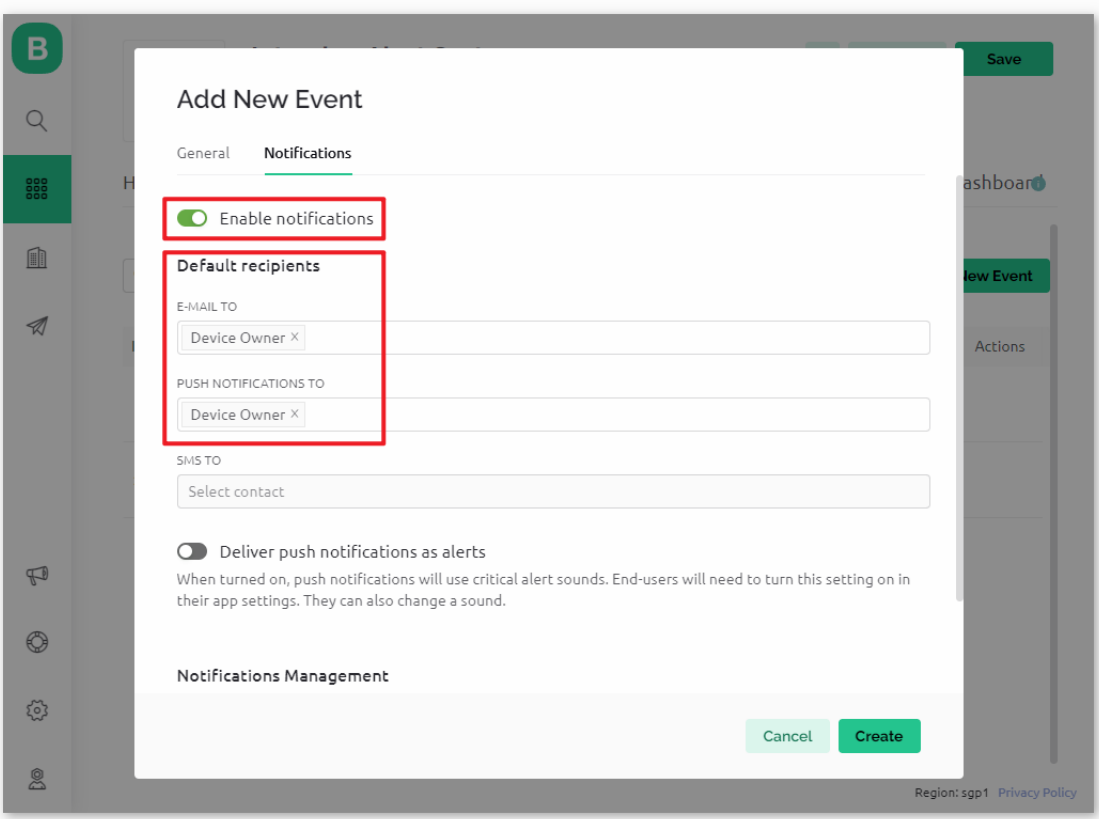
3 Event

Next, we will create an **event** that logs the detection of intrusion and sends an email notification.

Note: It is recommended to keep it consistent with my settings, otherwise you may need to modify the code to run the project. Make sure that the **EVENT CODE** is set as `intrusion_detected`.



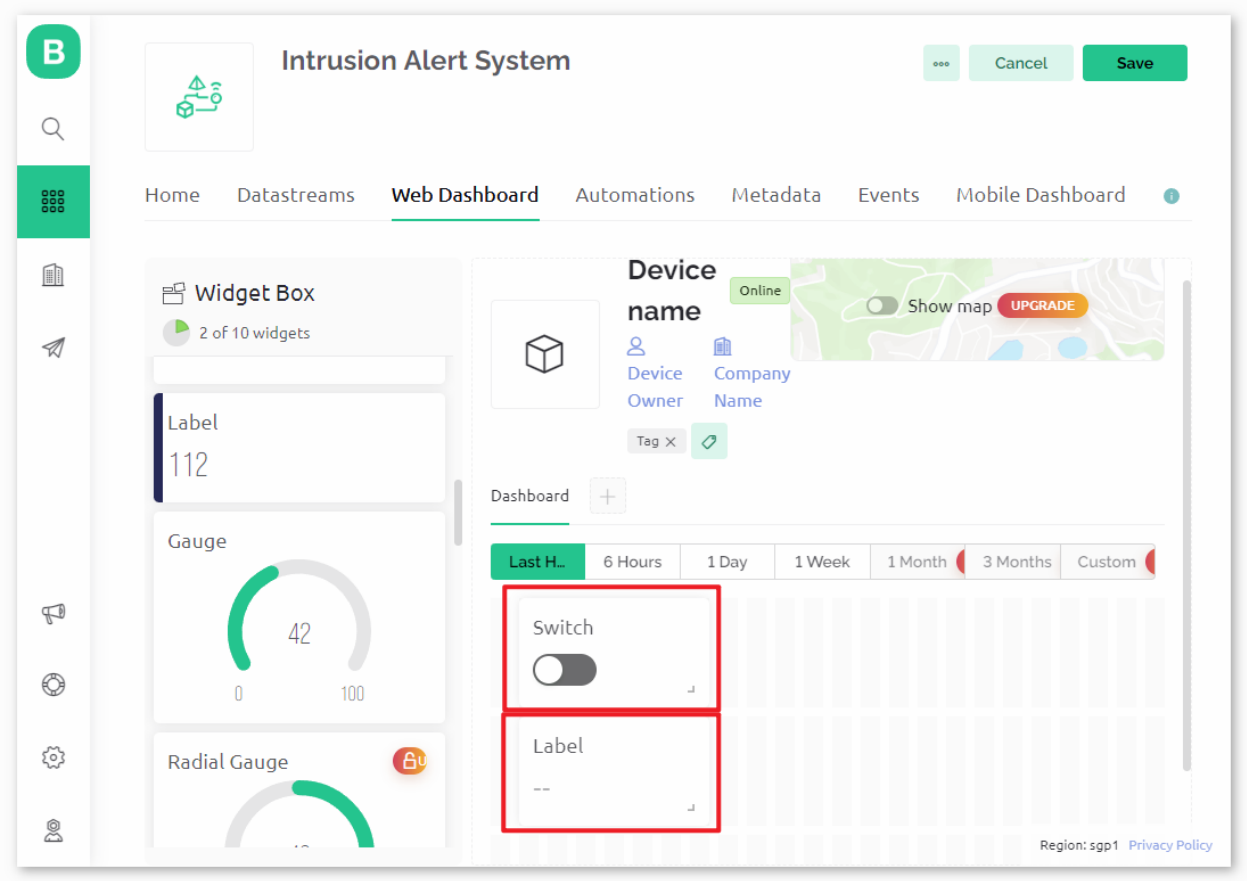
Go to the **Notifications** page and configure email settings.



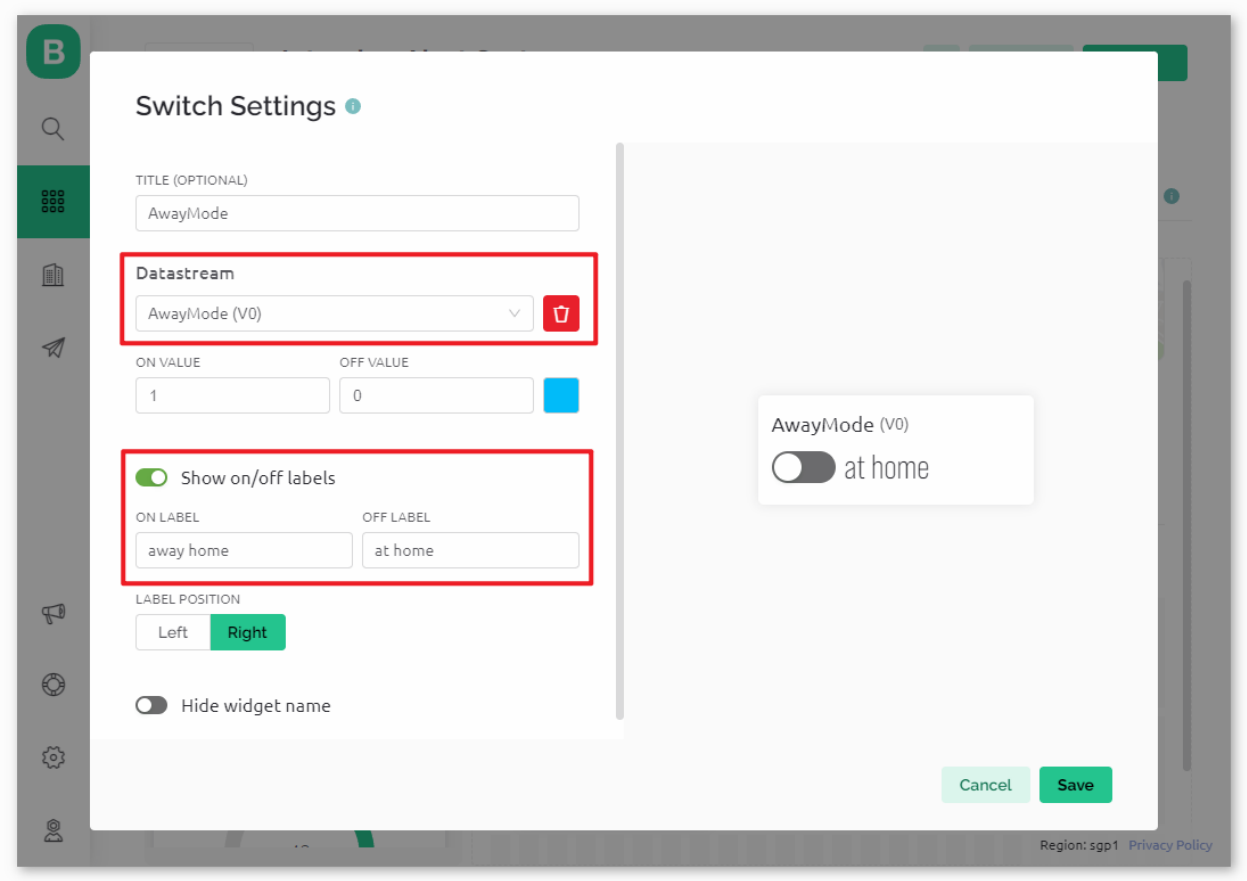
4 Web Dashboard

We also need to configure the **Web Dashboard** to interact with the Intrusion Alert System.

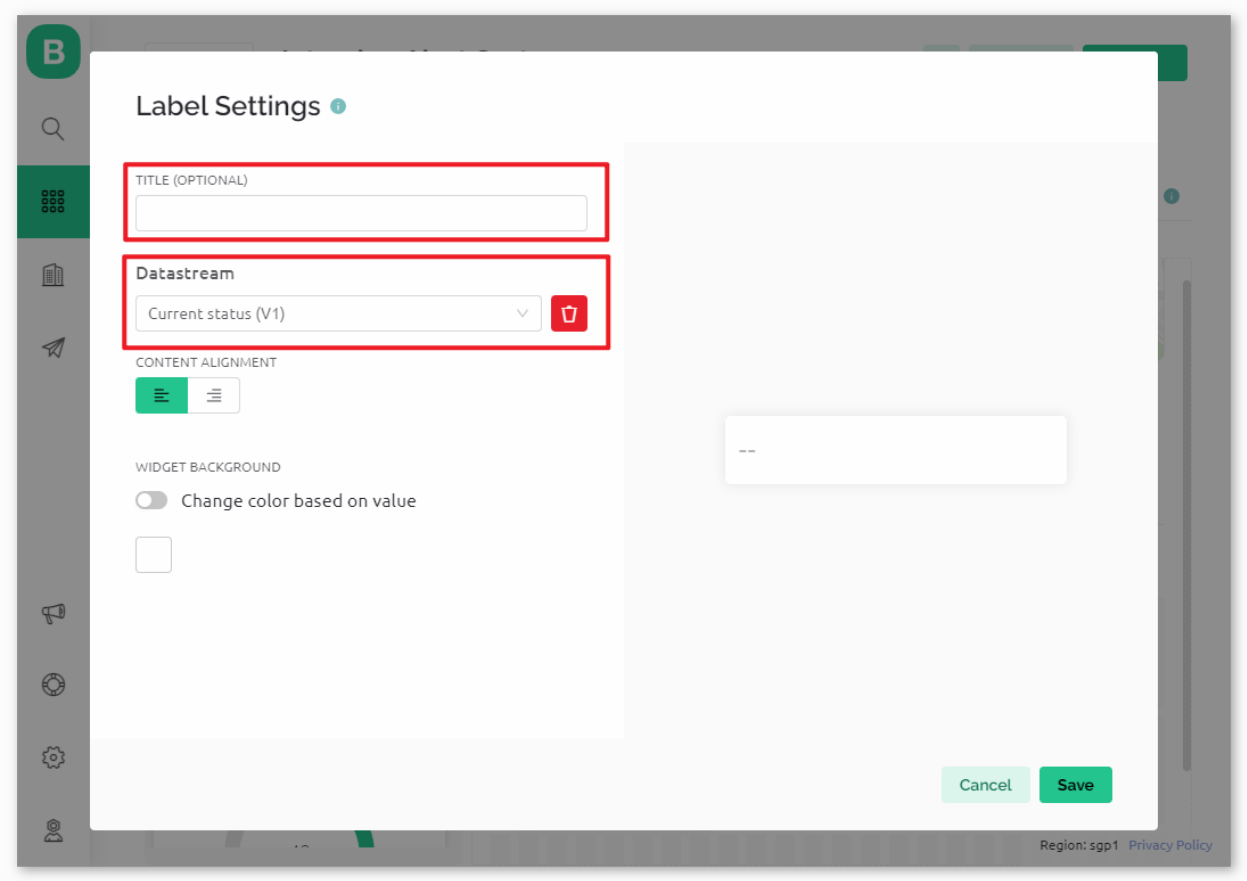
Drag and drop a **Switch widget** and a **Label widget** to the **Web Dashboard** page.



In the settings page of the **Switch widget**, select **Datastream** as **AwayMode(V0)**. Set **ONLABEL** and **OFFLABEL** to display “away home” when the switch is turned on, and “at home” when the switch is turned off.

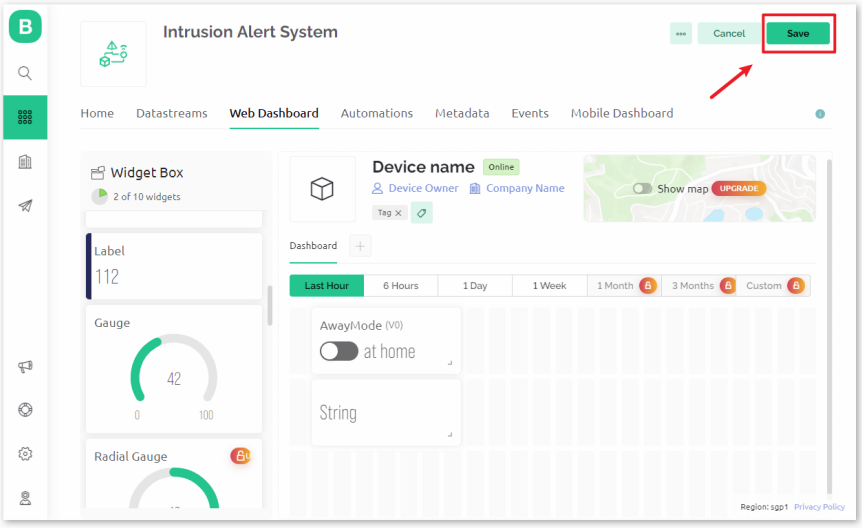


In the settings page of the **Label widget**, select **Datastream** as **Current status(V1)**.



5 Save template

At last, remember to save the template.



Code

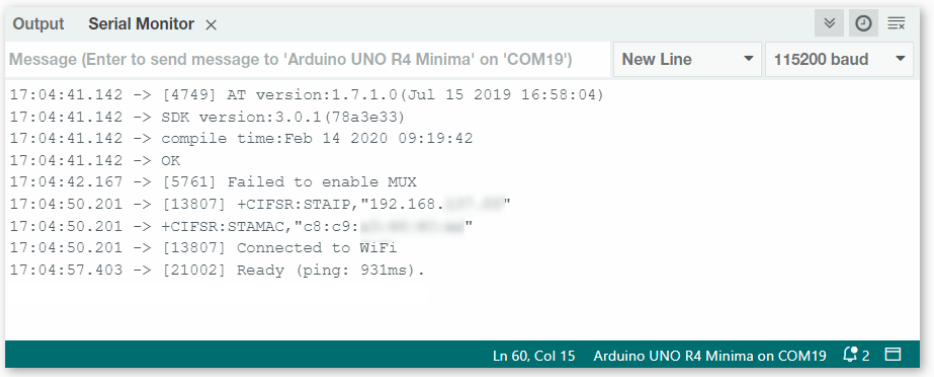
1. Open the `Lesson_51_Intrusion_alert_system.ino` file under the path of `universal-maker-sensor-kit\arduino_uno\Lesson_51_Intrusion_alert_system`, or copy this code into **Arduino IDE**.
2. Create a Blynk device using the "Intrusion Alert System" template. Then, replace the `BLYNK_TEMPLATE_ID`, `BLYNK_TEMPLATE_NAME`, and `BLYNK_AUTH_TOKEN` with your own.

```
#define BLYNK_TEMPLATE_ID "TMPxxxxxxx"
#define BLYNK_TEMPLATE_NAME "Intrusion Alert System"
#define BLYNK_AUTH_TOKEN "xxxxxxxxxxxxx"
```

3. You also need to enter the `ssid` and `password` of the WiFi you are using.

```
char ssid[] = "your_ssid";
char pass[] = "your_password";
```

4. After selecting the correct board and port, click the **Upload** button.
5. Open the Serial monitor(set baudrate to 115200) and wait for a prompt such as a successful connection to appear.



Note: If the message `ESP is not responding` appears when you connect, please follow these steps.

- Make sure the 9V battery is plugged in.
- Reset the ESP8266 module by connecting the pin RST to GND for 1 second, then unplug it.
- Press the reset button on the R4 board.

Sometimes, you may need to repeat the above operation 3-5 times, please be patient.

Code Analysis

1. **Configuration & Libraries**

   Here, constants and credentials for Blynk are set up. Necessary libraries for the ESP8266 WiFi module and Blynk are included.

```
#define BLYNK_TEMPLATE_ID "TMPxxxx"
#define BLYNK_TEMPLATE_NAME "Intrusion Alert System"
#define BLYNK_AUTH_TOKEN "xxxxxx-"
```

(continues on next page)

(continued from previous page)

```
#define BLYNK_PRINT Serial

#include <ESP8266_Lib.h>
#include <BlynkSimpleShieldEsp8266.h>
```

2. **WiFi Setup**

   Configure WiFi credentials and set up software serial communication with the ESP01 module.

```
char ssid[] = "your_ssid";
char pass[] = "your_password";

SoftwareSerial EspSerial(2, 3);
#define ESP8266_BAUD 115200
ESP8266 wifi(&EspSerial);
```

3. **PIR Sensor Configuration**

   Define the pin where the PIR sensor is connected and initialize state variables.

```
const int sensorPin = 8;
int state = 0;
int awayHomeMode = 0;
BlynkTimer timer;
```

4. **setup() Function**

   This initializes the PIR sensor as an input, sets up serial communication, connects to WiFi, and configures Blynk.

   - We use `timer.setInterval(1000L, myTimerEvent)` to set the timer interval in setup(), here we set to execute the `myTimerEvent()` function every **1000ms**. You can modify the first parameter of `timer.setInterval(1000L, myTimerEvent)` to change the interval between `myTimerEvent` executions.

```
void setup() {
   pinMode(sensorPin, INPUT);
   Serial.begin(115200);
   EspSerial.begin(ESP8266_BAUD);
   delay(10);
   Blynk.config(wifi, BLYNK_AUTH_TOKEN);
   Blynk.connectWiFi(ssid, pass);
   timer.setInterval(1000L, myTimerEvent);
}
```

5. **loop() Function**

   The loop function repeatedly runs Blynk and the Blynk timer functions.

```
void loop() {
   Blynk.run();
   timer.run();
}
```

6. **Blynk App Interaction**

   These functions are called when the device connects to Blynk and when there's a change in the state of the virtual pin V0 on the Blynk app.

- Every time the device connects to the Blynk server, or reconnects due to poor network conditions, the `BLYNK_CONNECTED()` function is called. The `Blynk.syncVirtual()` command request a single Virtual Pin value. The specified Virtual Pin will perform `BLYNK_WRITE()` call. Please refer to for more details.
- Whenever the value of a virtual pin on the BLYNK server changes, it will trigger `BLYNK_WRITE()`. More details at .

```
// This function is called every time the device is connected to the Blynk.Cloud
BLYNK_CONNECTED() {
   Blynk.syncVirtual(V0);
}

// This function is called every time the Virtual Pin 0 state changes
BLYNK_WRITE(V0) {
   awayHomeMode = param.asInt();
   // additional logic
}
```

7. **Data Handling**

   Every second, the `myTimerEvent()` function calls `sendData()`. If the away mode is enabled on Blynk, it checks the PIR sensor and sends a notification to Blynk if motion is detected.

   - We use `Blynk.virtualWrite(V1, "Somebody in your house! Please check!");` to change the text of a label.
   - Use `Blynk.logEvent("intrusion_detected");` to log event to Blynk.

```
void myTimerEvent() {
   sendData();
}

void sendData() {
   if (awayHomeMode == 1) {
      state = digitalRead(sensorPin);  // Read the state of the PIR sensor

      Serial.print("state:");
      Serial.println(state);

      // If the sensor detects movement, send an alert to the Blynk app
      if (state == HIGH) {
        Serial.println("Somebody here!");
        Blynk.virtualWrite(V1, "Somebody in your house! Please check!");
        Blynk.logEvent("intrusion_detected");
      }
   }
}
```

Reference

-
-
-
-
-

-
-

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4 For ESP32

ESP32 users, please refer to the following tutorial.

The following tutorial is based on the *ESP32 WROOM 32E* development board, but it is also applicable to other ESP32 development boards.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.1 Get Started with ESP32

We'll program the ESP32 development board using the Arduino IDE. This chapter offers a quick-start guide, with steps for installing the Arduino IDE, an overview of its features, and instructions for adding the ESP32 board package.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.

- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Install Arduino IDE(Important)

The Arduino IDE, known as Arduino Integrated Development Environment, provides all the software support needed to complete an Arduino project. It is a programming software specifically designed for Arduino, provided by the Arduino team, that allows us to write programs and upload them to the Arduino board.

The Arduino IDE 2.0 is an open-source project. It is a big step from its sturdy predecessor, Arduino IDE 1.x, and comes with revamped UI, improved board & library manager, debugger, autocomplete feature and much more.

In this tutorial, we will show how to download and install the Arduino IDE 2.0 on your Windows, Mac, or Linux computer.

Requirements

- Windows - Win 10 and newer, 64 bits
- Linux - 64 bits
- Mac OS X - Version 10.14: “Mojave” or newer, 64 bits

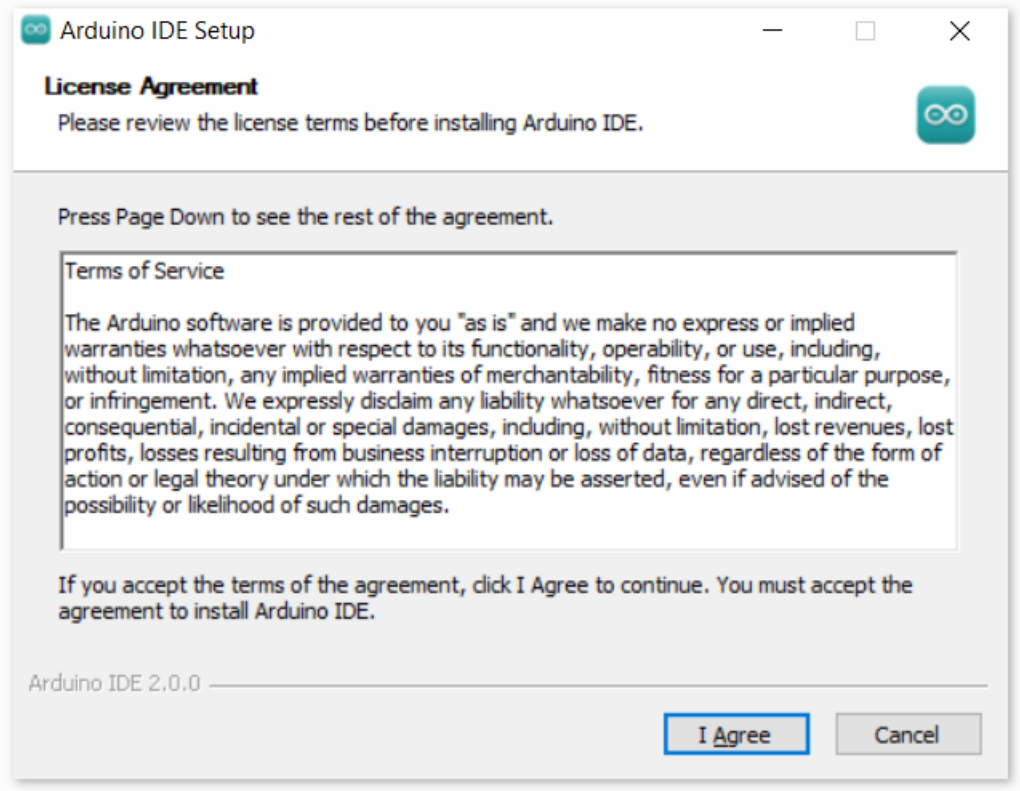
Download the Arduino IDE 2.0

1. Visit .
2. Download the IDE for your OS version.

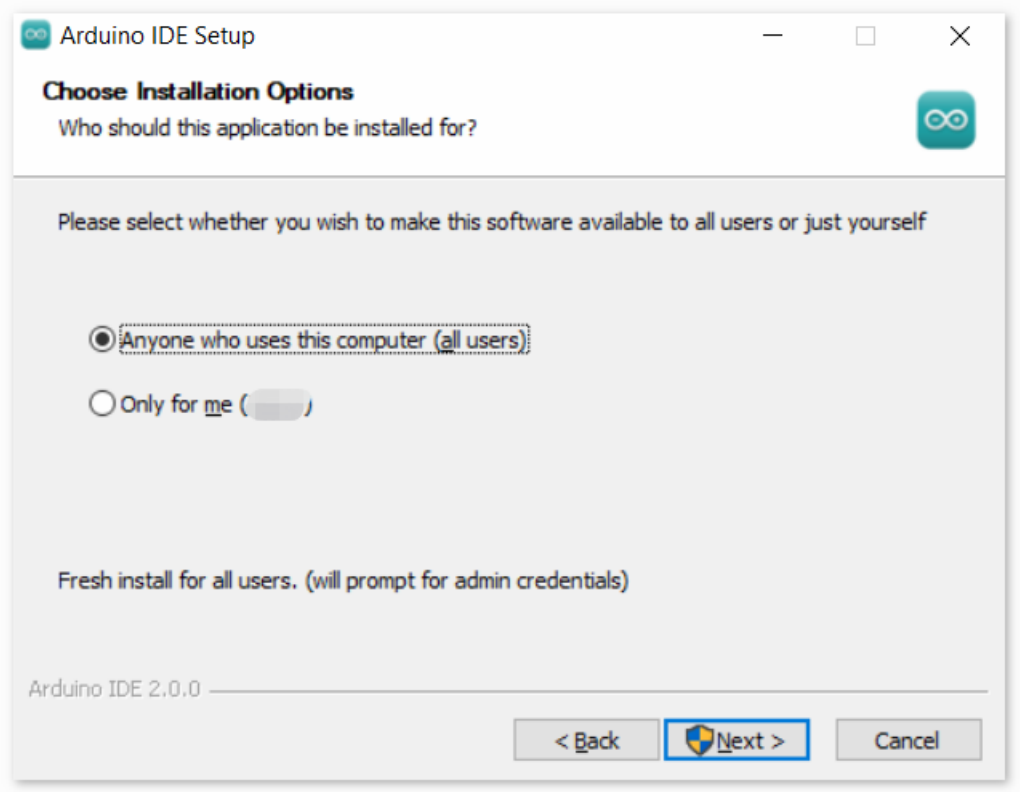
Installation

Windows

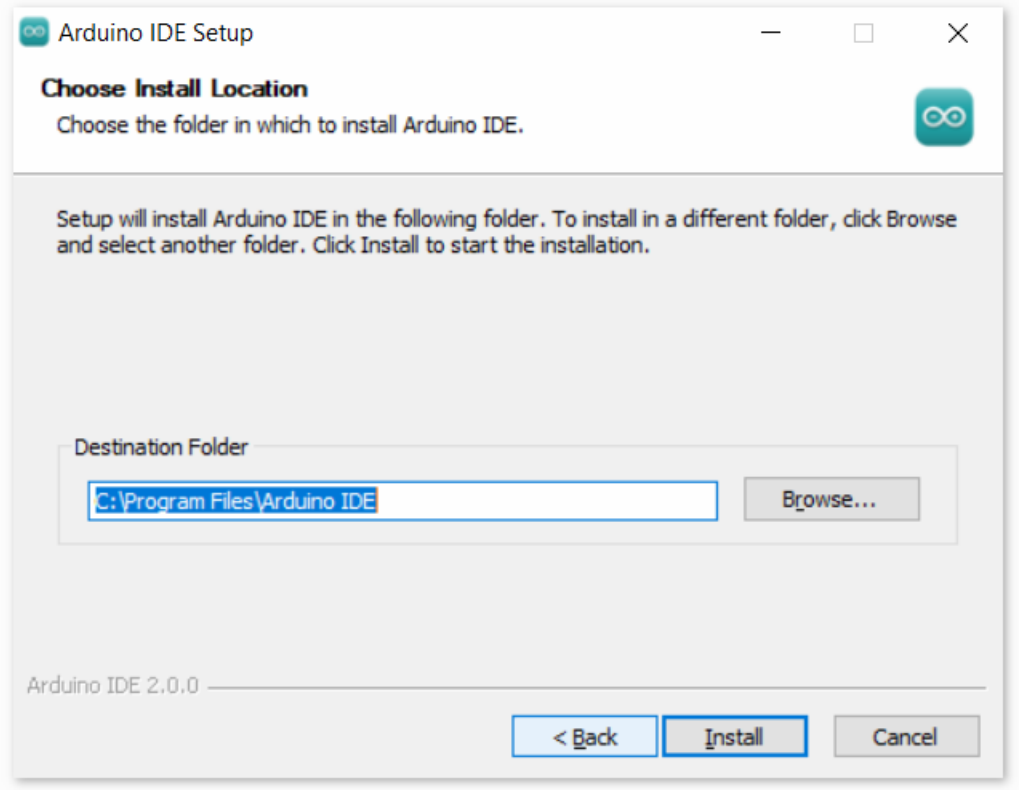
1. Double click the `arduino-ide_xxxx.exe` file to run the downloaded file.
2. Read the License Agreement and agree it.



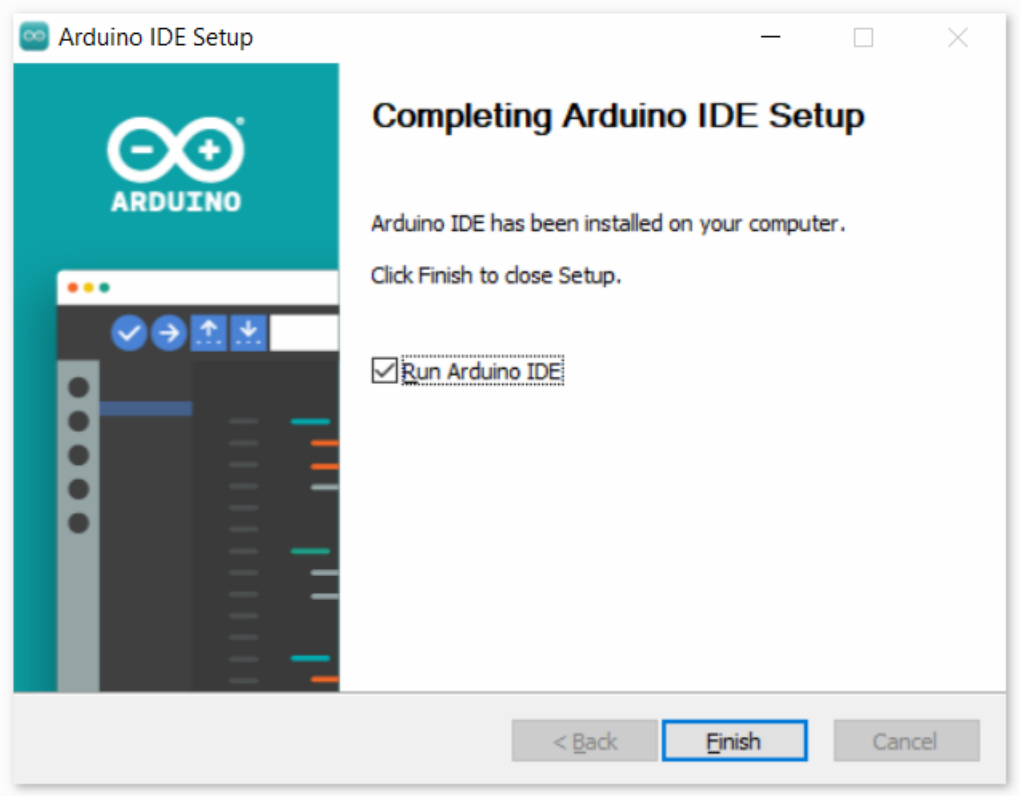
3. Choose installation options.



4. Choose install location. It is recommended that the software be installed on a drive other than the system drive.

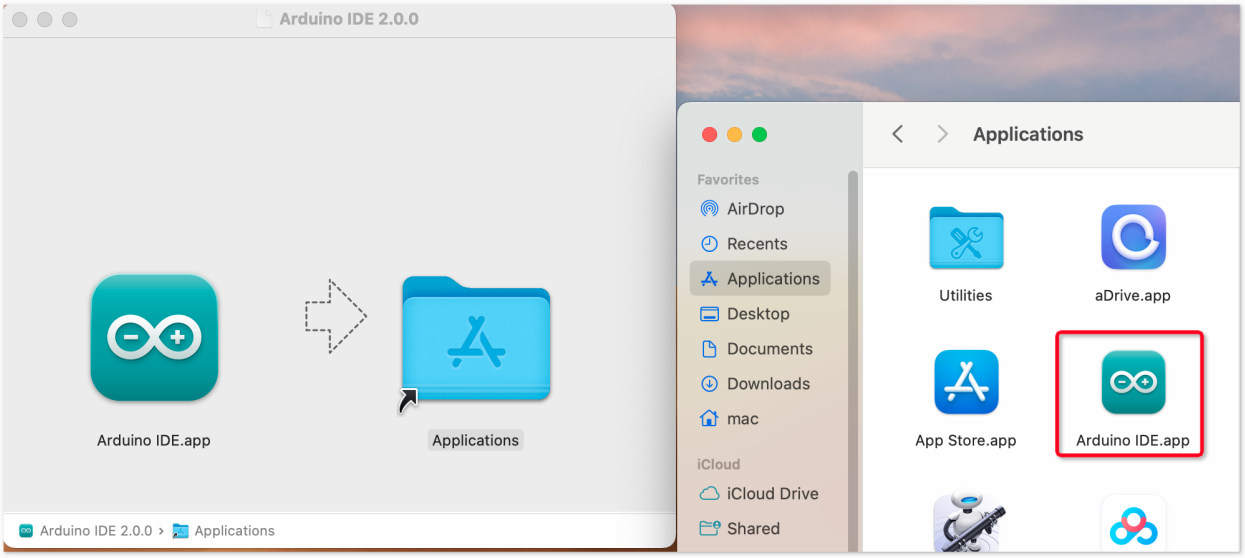


5. Then Finish.



macOS

Double click on the downloaded `arduino_ide_xxxx.dmg` file and follow the instructions to copy the **Arduino IDE.app** to the **Applications** folder, you will see the Arduino IDE installed successfully after a few seconds.

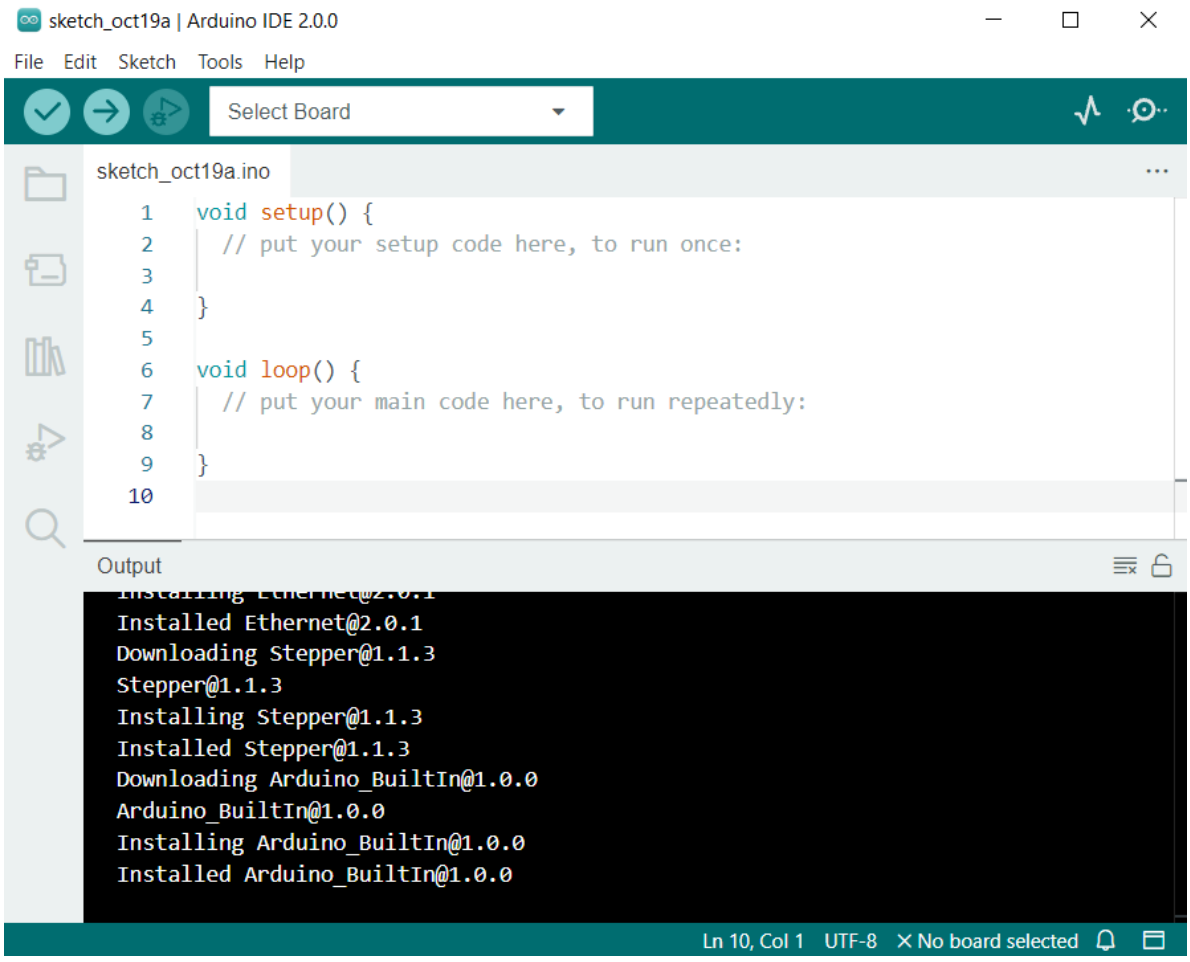


Linux

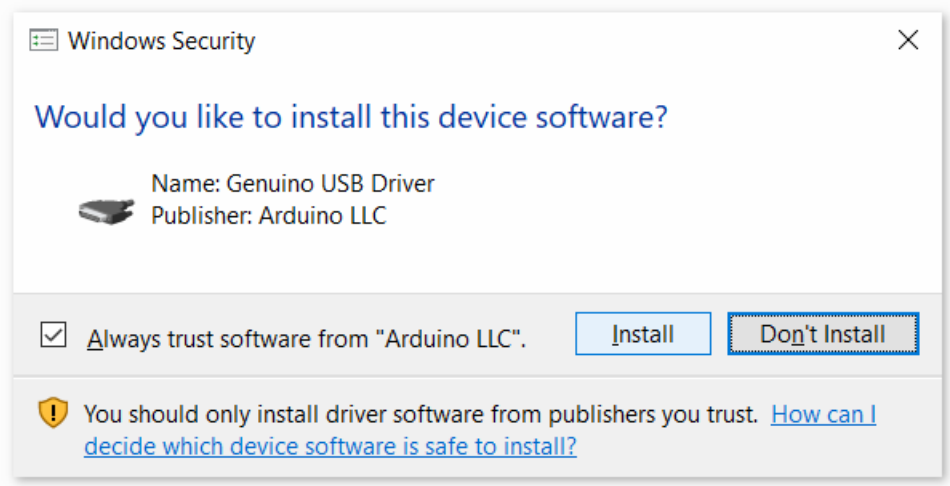
For the tutorial on installing the Arduino IDE 2.0 on a Linux system, please refer to: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started/ide-v2-downloading-and-installing#linux

Open the IDE

1. When you first open Arduino IDE 2.0, it automatically installs the Arduino AVR Boards, built-in libraries, and other required files.



2. In addition, your firewall or security center may pop up a few times asking you if you want to install some device driver. Please install all of them.



3. Now your Arduino IDE is ready!

Note: In the event that some installations didn't work due to network issues or other reasons, you can reopen the Arduino IDE and it will finish the rest of the installation. The Output window will not automatically open after all installations are complete unless you click Verify or Upload.

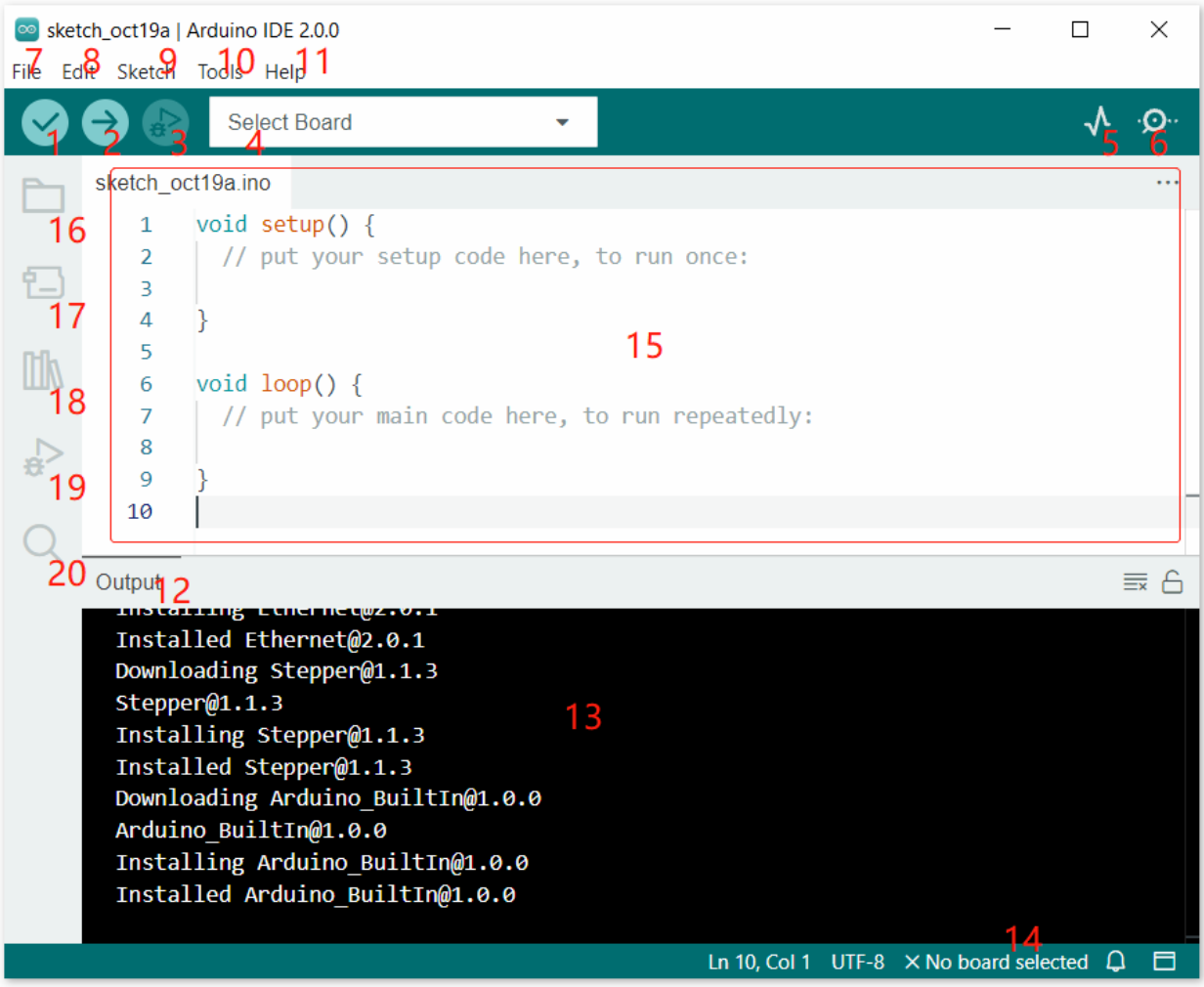
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Introduce of Arduino IDE



1. **Verify**: Compile your code. Any syntax problem will be prompted with errors.
2. **Upload**: Upload the code to your board. When you click the button, the RX and TX LEDs on the board will flicker fast and won't stop until the upload is done.
3. **Debug**: For line-by-line error checking.
4. **Select Board**: Quick setup board and port.
5. **Serial Plotter**: Check the change of reading value.
6. **Serial Monitor**: Click the button and a window will appear. It receives the data sent from your control board. It is very useful for debugging.
7. **File**: Click the menu and a drop-down list will appear, including file creating, opening, saving, closing, some parameter configuring, etc.
8. **Edit**: Click the menu. On the drop-down list, there are some editing operations like **Cut**, **Copy**, **Paste**, **Find**, and so on, with their corresponding shortcuts.
9. **Sketch**: Includes operations like **Verify**, **Upload**, **Add** files, etc. More important function is **Include Library** - where you can add libraries.

10. **Tool**: Includes some tools - the most frequently used Board (the board you use) and Port (the port your board is at). Every time you want to upload the code, you need to select or check them.
11. **Help**: If you're a beginner, you may check the options under the menu and get the help you need, including operations in IDE, introduction information, troubleshooting, code explanation, etc.
12. **Output Bar**: Switch the output tab here.
13. **Output Window**: Print information.
14. **Board and Port**: Here you can preview the board and port selected for code upload. You can select them again by **Tools** -> **Board / Port** if any is incorrect.
15. The editing area of the IDE. You can write code here.
16. **Sketchbook**: For managing sketch files.
17. **Board Manager**: For managing board driver.
18. **Library Manager**: For managing your library files.
19. **Debug**: Help debugging code.
20. **Search**: Search the codes from your sketches.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

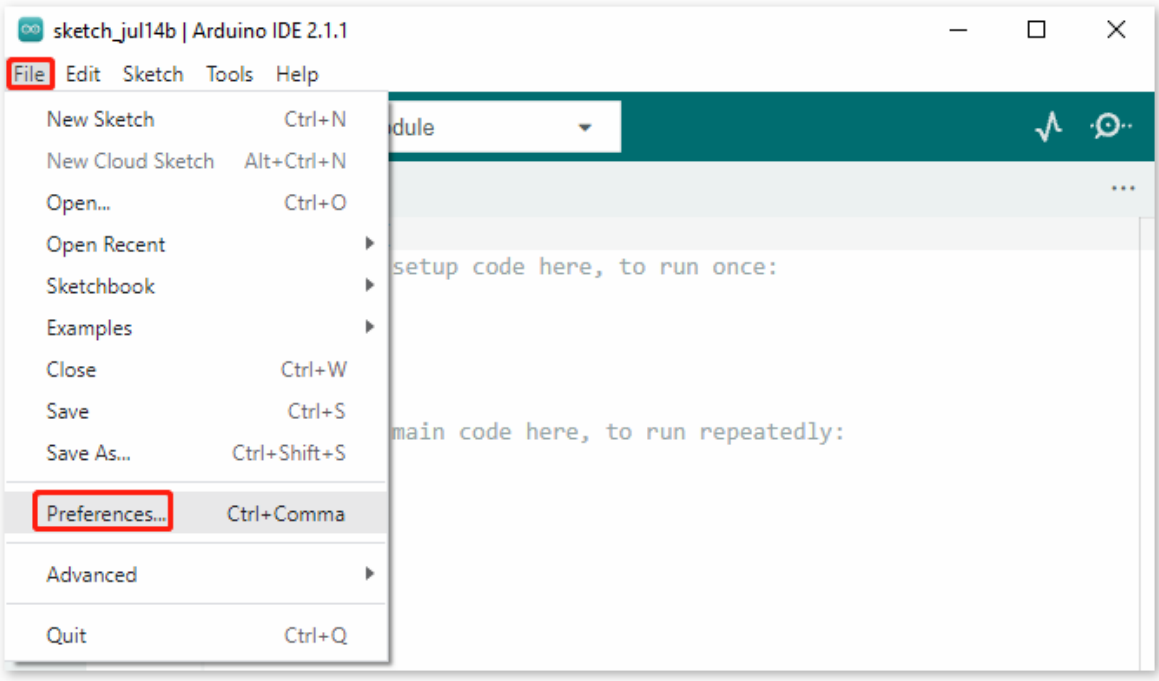
Ready to explore and create with us? Click [] and join today!

Install the ESP32 Board(Important)

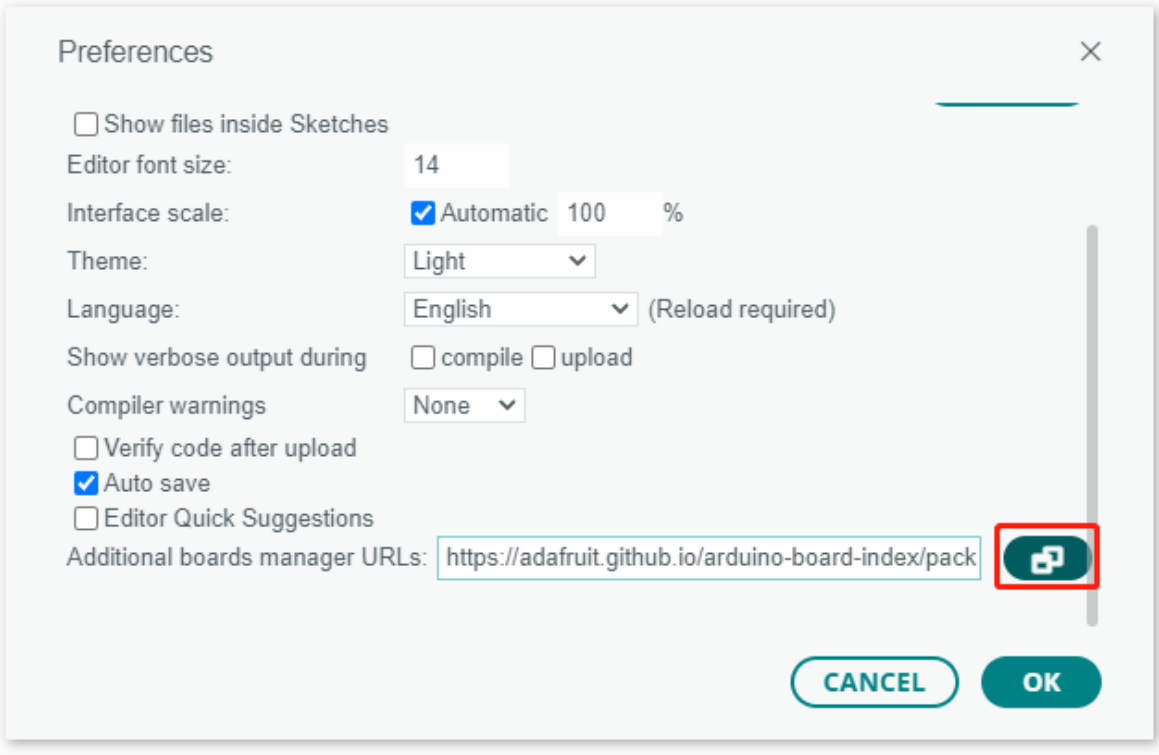
To program the ESP32 microcontroller, we need to install the ESP32 board package in the Arduino IDE. Follow the step-by-step guide below:

Install the ESP32 Board

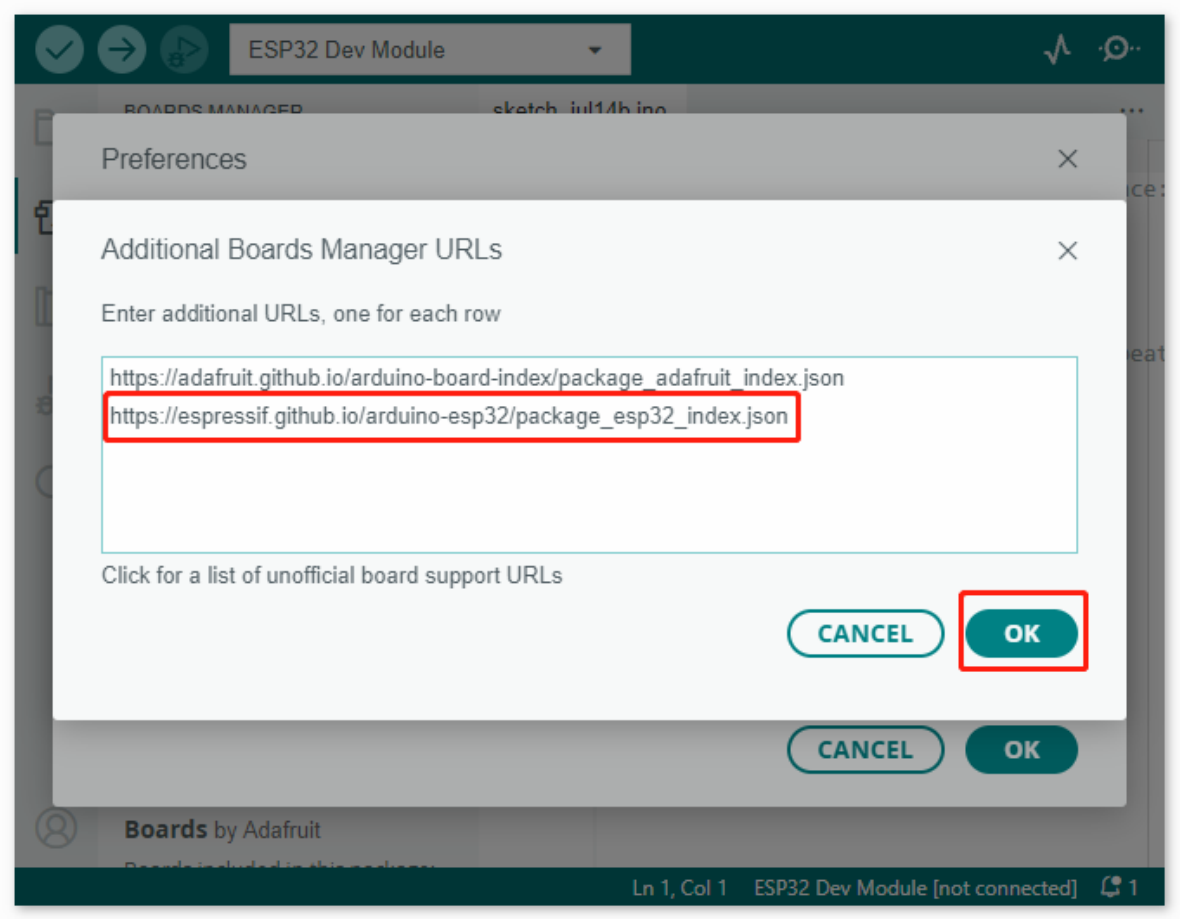
1. Open the Arduino IDE. Go to **File** and select **Preferences** from the drop-down menu.



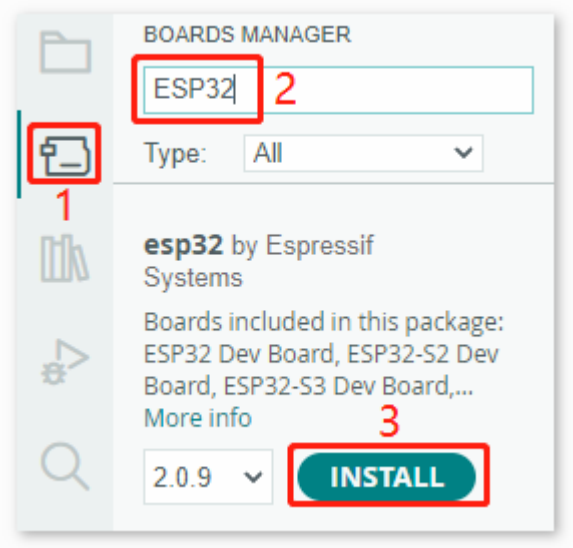
2. In the Preferences window, locate the **Additional Board Manager URLs** field. Click on it to activate the text box.



3. Add the following URL to the **Additional Board Manager URLs** field: https://espressif.github.io/arduino-esp32/package_esp32_index.json. This URL points to the package index file for the ESP32 boards. Click the **OK** button to save the changes.



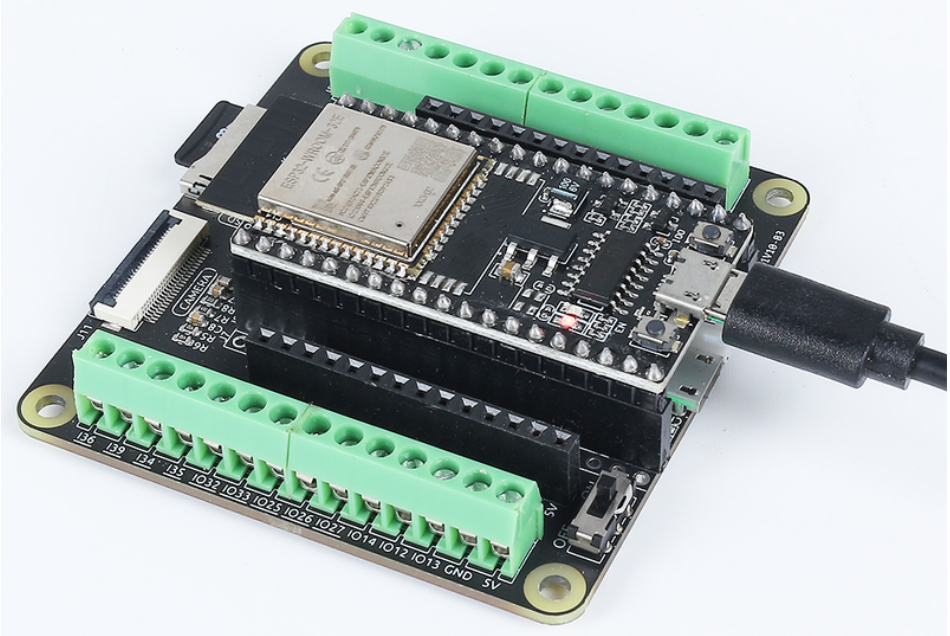
4. In the **Boards Manager** window, type **ESP32** in the search bar. Click the **Install** button to start the installation process. This will download and install the ESP32 board package.



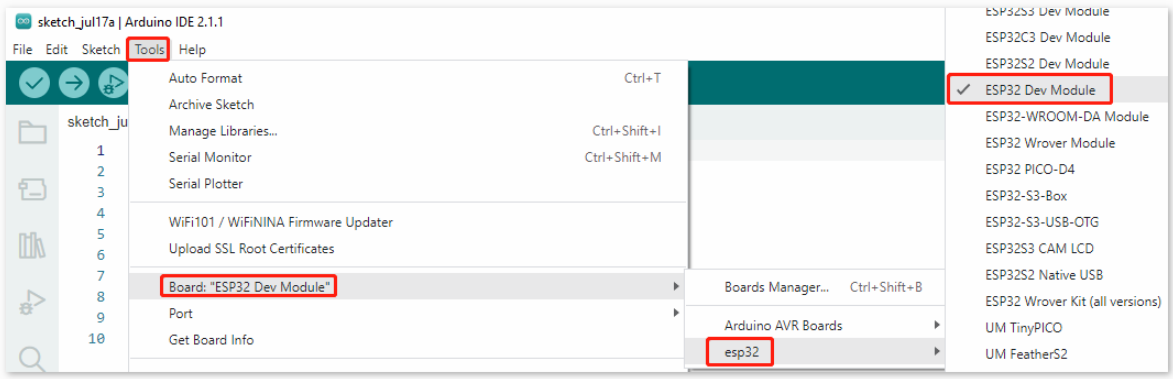
5. Congratulations! You have successfully installed the ESP32 board package in the Arduino IDE.

Upload the Code

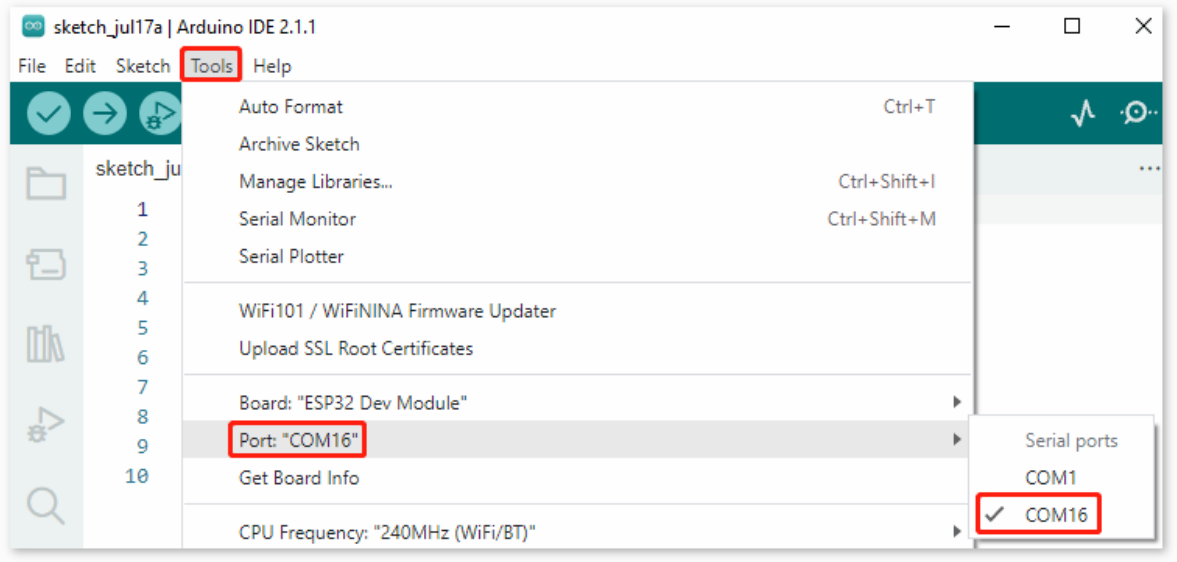
1. Now, connect the ESP32 WROOM 32E to your computer using a Micro USB cable.



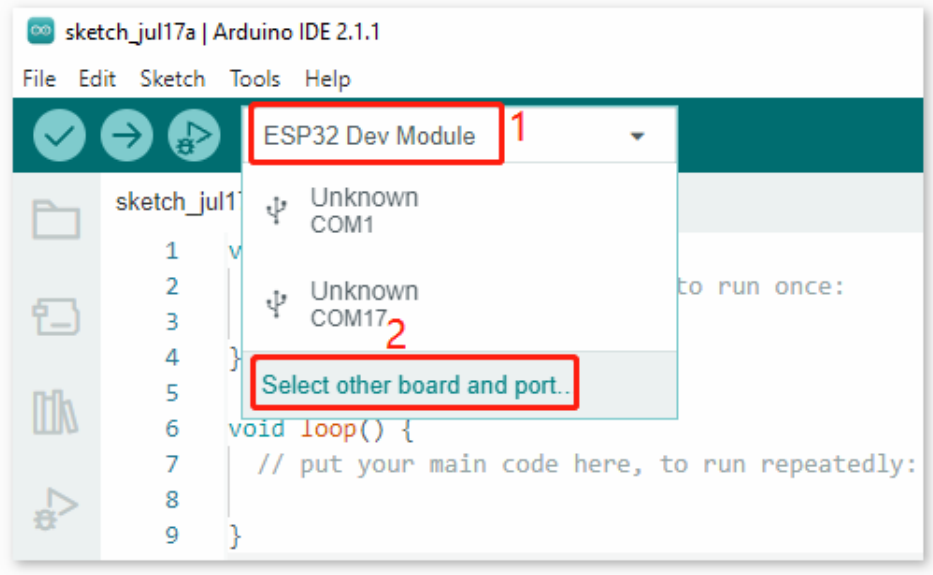
2. Then select the correct board, **ESP32 Dev Module**, by clicking on **Tools** -> **Board** -> **esp32**.



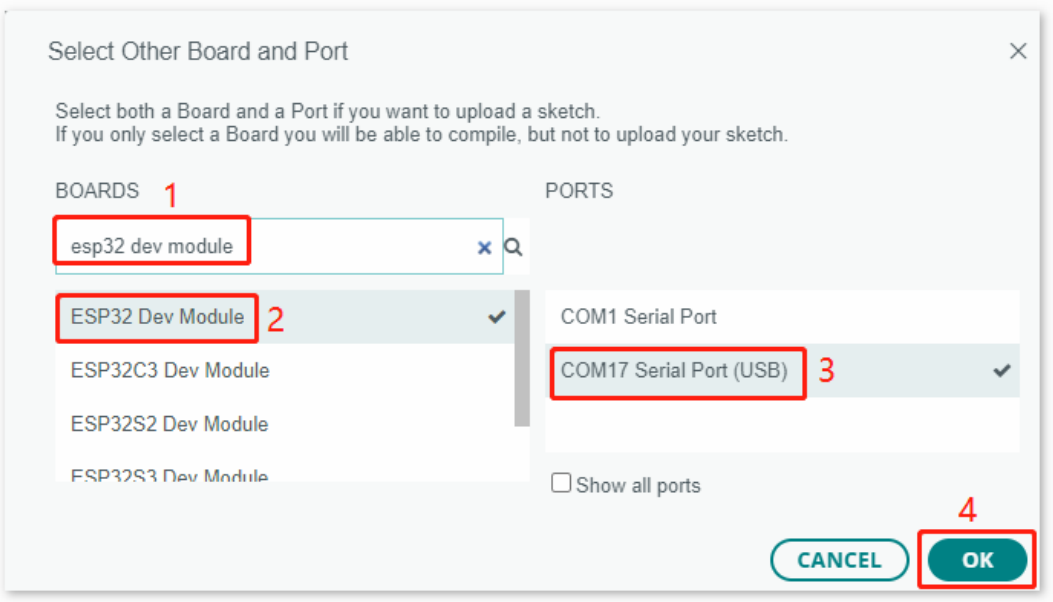
3. If your ESP32 is connected to the computer, you can choose the correct port by clicking on **Tools** -> **Port**.



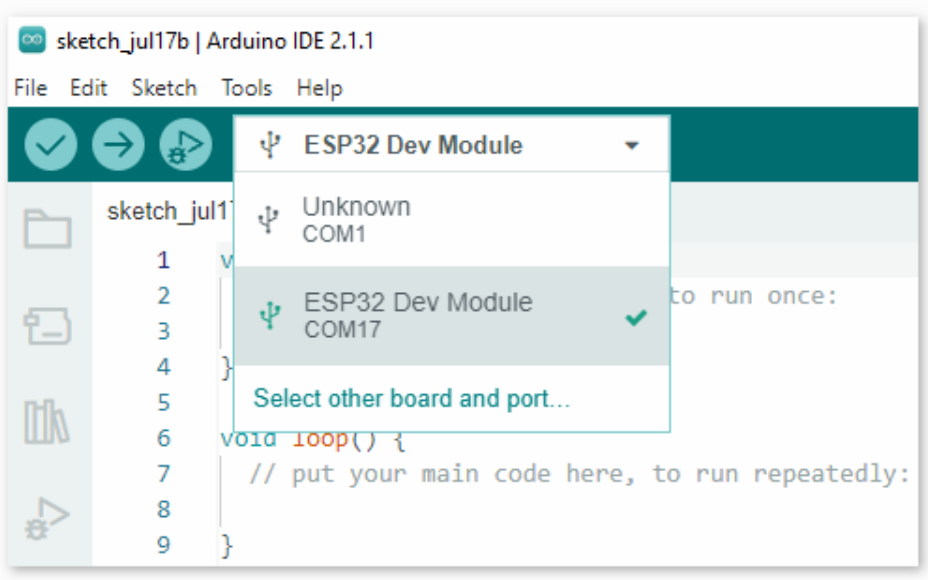
4. Additionally, Arduino 2.0 introduced a new way to quickly select the board and port. For ESP32, it is usually not automatically recognized, so you need to click on **Select other board and port**.



5. In the search box, type **ESP32 Dev Module** and select it when it appears. Then, choose the correct port and click **OK**.



6. Afterward, you can select it through this quick access window. Note that during subsequent use, there may be times when ESP32 is not available in the quick access window, and you will need to repeat the above two steps.



7. Both methods allow you to select the correct board and port, so choose the one that suits you best. Now, everything is ready to upload the code to the ESP32.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.

- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Install libraries (Important)

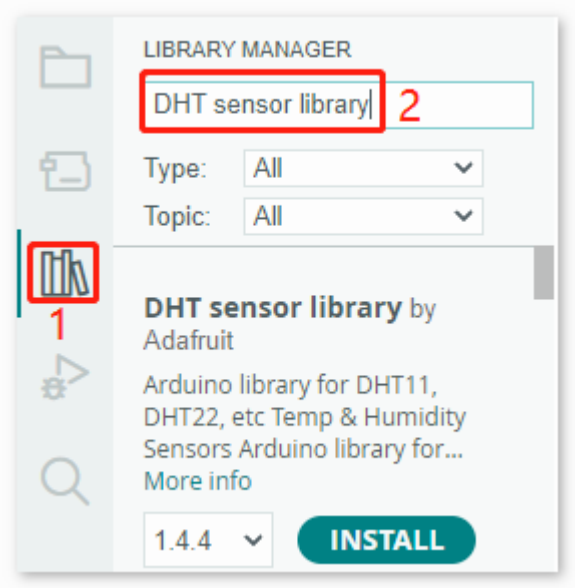
A library is a collection of pre-written code or functions that extend the capabilities of the Arduino IDE. Libraries provide ready-to-use code for various functionalities, allowing you to save time and effort in coding complex features.

Install from Library Manager

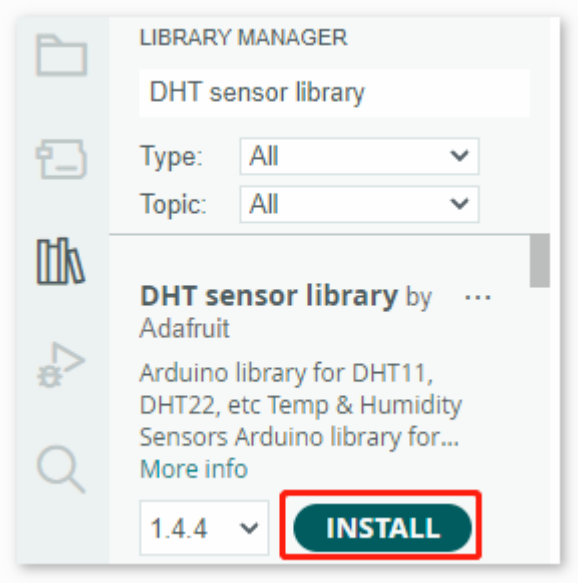
Many libraries are available directly through the Arduino Library Manager. You can access the Library Manager by following these steps:

1. In the **Library Manager**, you can search for the desired library by name or browse through different categories.

Note: In projects where library installation is required, there will be prompts indicating which libraries to install. Follow the instructions provided, such as "To install the library, use the Arduino Library Manager and search for "DHT sensor library" and install it." Simply search and install the recommended libraries as prompted.



2. Once you find the library you want to install, click on it and then click the **Install** button.



3. The Arduino IDE will automatically download and install the library for you.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.2 Lesson 01: Button Module

In this lesson, you will learn how a button interacts with an LED using ESP32 Development Board. We'll see how pressing the button lights up the LED and releasing it turns off the LED. This project is ideal for beginners as it provides a practical understanding of input and output operations on the ESP32 platform.

Required Components

In this project, we need the following components.

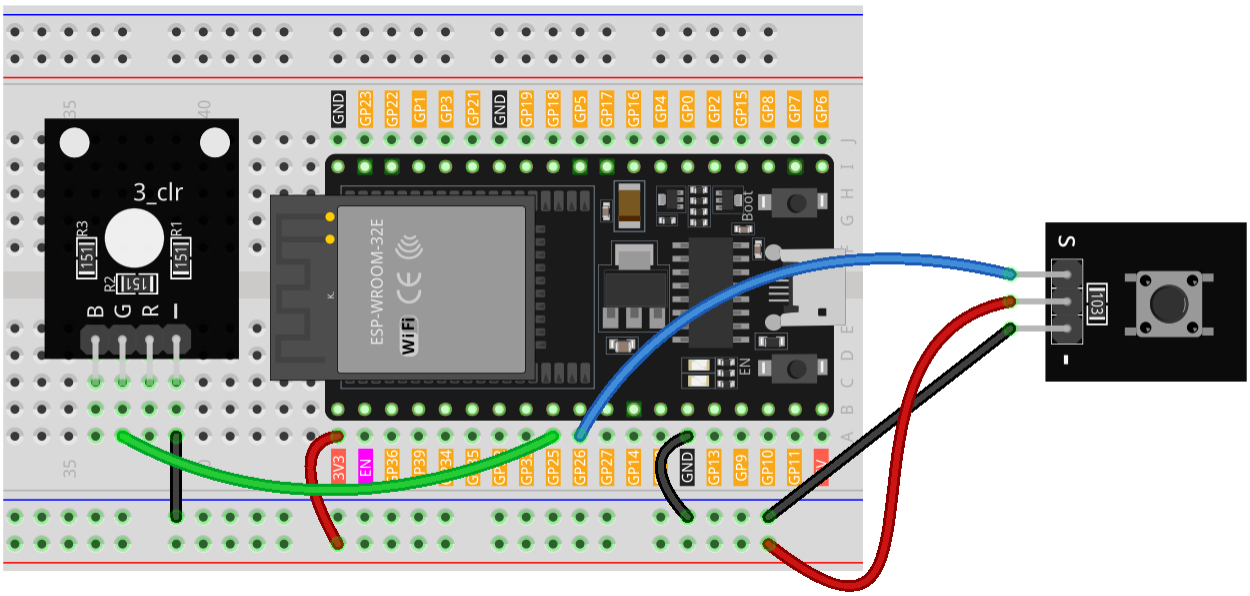
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Button Module	-
Breadboard	

Wiring



Code

Code Analysis

1. Initialization of Pins

   The pins for the button and LED are defined and initialized. The `buttonPin` is set as an input to read the button's state, and `ledPin` is set as an output to control the LED.

   ```
   const int buttonPin = 26;  // Pin number for the button
   const int ledPin = 25;     // Pin number for the LED
   int buttonState = 0;  // Variable to hold the current state of the button
   ```

2. Setup Function

   This function runs once and sets up the pin modes. `pinMode(buttonPin, INPUT)` configures the button pin as an input. `pinMode(ledPin, OUTPUT)` sets the LED pin as an output.

   ```
   void setup() {
     pinMode(buttonPin, INPUT);  // Initialize buttonPin as an input pin
   ```

(continues on next page)

(continued from previous page)

```
  pinMode(ledPin, OUTPUT);    // Initialize ledPin as an output pin
}
```

3. Main Loop Function

   This is the core of the program where the button state is continuously read and the LED state is controlled. `digitalRead(buttonPin)` reads the button's state. If the button is pressed (state is LOW), the LED is turned on by `digitalWrite(ledPin, HIGH)`. If not pressed, the LED is turned off (`digitalWrite(ledPin, LOW)`).

   The *button module* used in this project has an internal pull-up resistor (see its *schematic diagram*), causing the button to be at a low level when pressed and remain at a high level when released.

```
void loop() {
  // Read the current state of the button
  buttonState = digitalRead(buttonPin);

  // Check if the button is pressed (LOW)
  if (buttonState == LOW) {
    digitalWrite(ledPin, HIGH);  // Turn the LED on
  } else {
    digitalWrite(ledPin, LOW);  // Turn the LED off
  }
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.3 Lesson 02: Capacitive Soil Moisture Module

In this lesson, you will learn how to use a capacitive soil moisture sensor with an ESP32 Development Board to read the moisture level of soil. We'll cover connecting the sensor to pin 25, reading its analog value, and interpreting these readings to determine the soil's moisture level. This project is ideal for beginners as it provides hands-on experience in working with sensors and understanding analog input on the ESP32 platform.

Required Components

In this project, we need the following components.

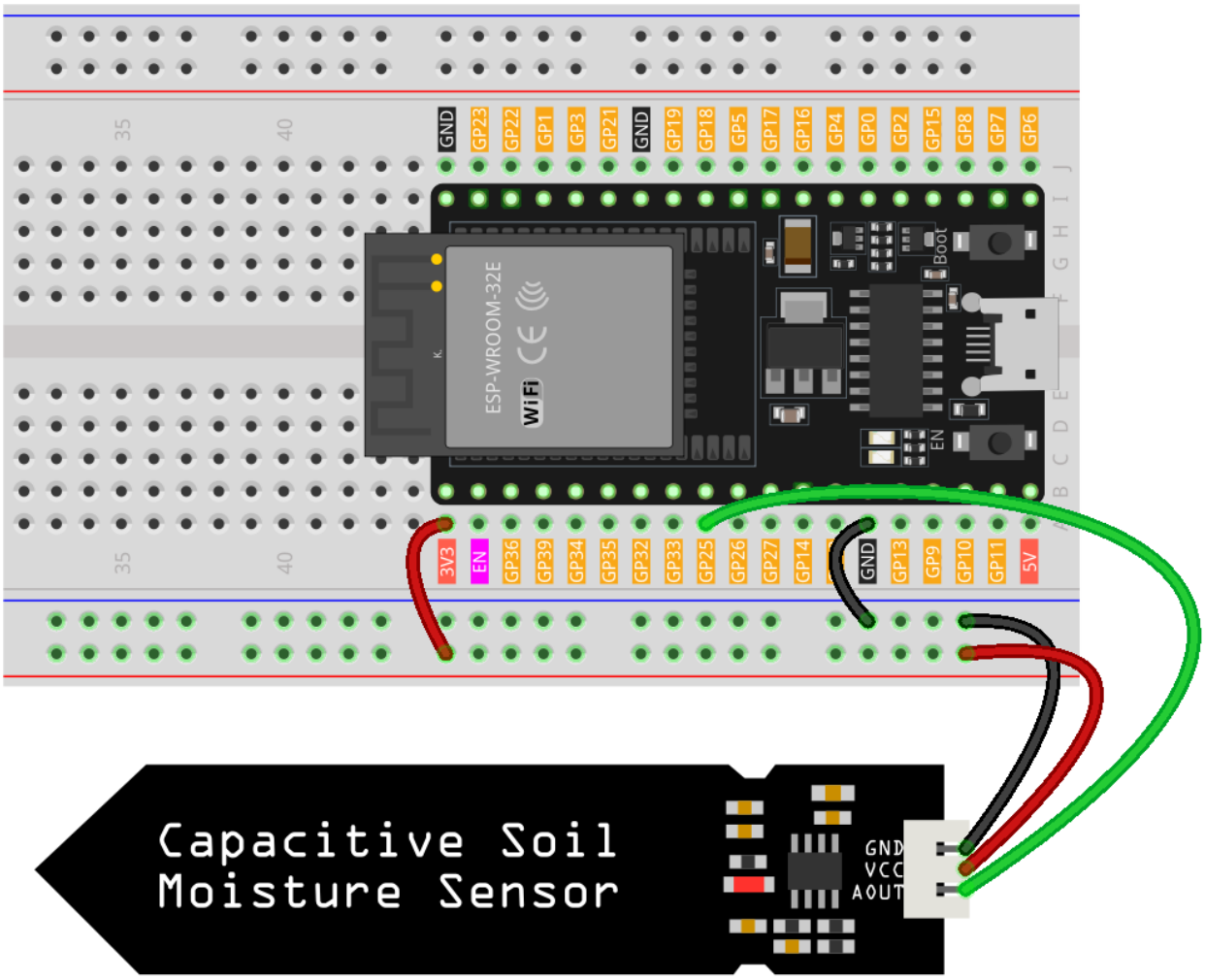
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Capacitive Soil Moisture Module	
Breadboard	

Wiring



Code

Code Analysis

1. Defining the sensor pin:

   This line of code declares a constant integer `sensorPin` and assigns it the value of `25`, which is the pin the sensor is connected to.

   ```
   const int sensorPin = 25;
   ```

2. Setup function:

   The `setup()` function is executed once when the program starts. It initializes serial communication at 9600 baud rate. This setup is necessary for sending data to the serial monitor.

   ```
   void setup() {
     Serial.begin(9600);
   }
   ```

3. Loop function:

   The `loop()` function runs continuously after `setup()`. It reads the sensor value from pin A0 using `analogRead()` and prints this value to the serial monitor. The `delay(500)` statement pauses the loop for 500 milliseconds before the next reading, thus controlling the rate of data acquisition.

```
void loop() {
  Serial.println(analogRead(sensorPin));
  delay(500);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.4 Lesson 03: Flame Sensor Module

In this lesson, you will learn how to connect a flame sensor to an ESP32 Development Board for fire detection. We'll examine the sensor's response to fire and how it triggers a warning message. This project is ideal for beginners working with sensors and ESP32, providing hands-on experience in monitoring environmental factors using basic electronic components.

Required Components

In this project, we need the following components.

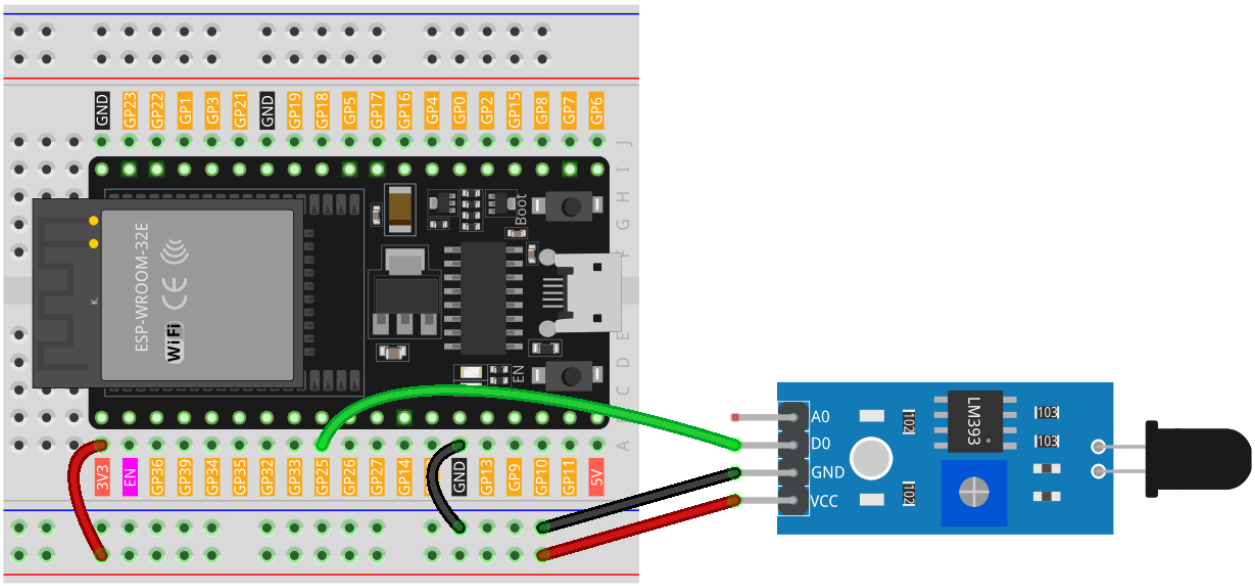
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Flame Sensor Module	
Breadboard	

Wiring



Code

Code Analysis

1. **Defining the Sensor Pin**:

   The pin to which the flame sensor is connected is defined as an integer constant.

   ```
   const int sensorPin = 25;
   ```

2. **Setup Function**:

   This function runs once when the ESP32 starts. It initializes the sensor pin as an input and begins serial communication at 9600 baud rate for output.

   ```
   void setup() {
     pinMode(sensorPin, INPUT);
     Serial.begin(9600);
   }
   ```

3. **Loop Function**:

The core of the program, it continuously checks the state of the flame sensor. If the sensor detects a flame (returns 0), it prints a fire alert message. Otherwise, it indicates no fire is detected. The check happens every 100 milliseconds.

```
void loop() {
  if (digitalRead(sensorPin) == 0) {
    Serial.println("** Fire detected!!! **");
  } else {
    Serial.println("No Fire detected");
  }
  delay(100);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.5 Lesson 04: Gas Sensor Module (MQ-2)

In this lesson, you will learn how to measure gas concentrations using an MQ-2 sensor with an ESP32 Development Board. We'll cover reading the analog output of the gas sensor and displaying it on the serial monitor. This project is ideal for beginners in electronics, providing hands-on experience with sensors and microcontrollers while teaching about analog signal processing and serial communication.

Required Components

In this project, we need the following components.

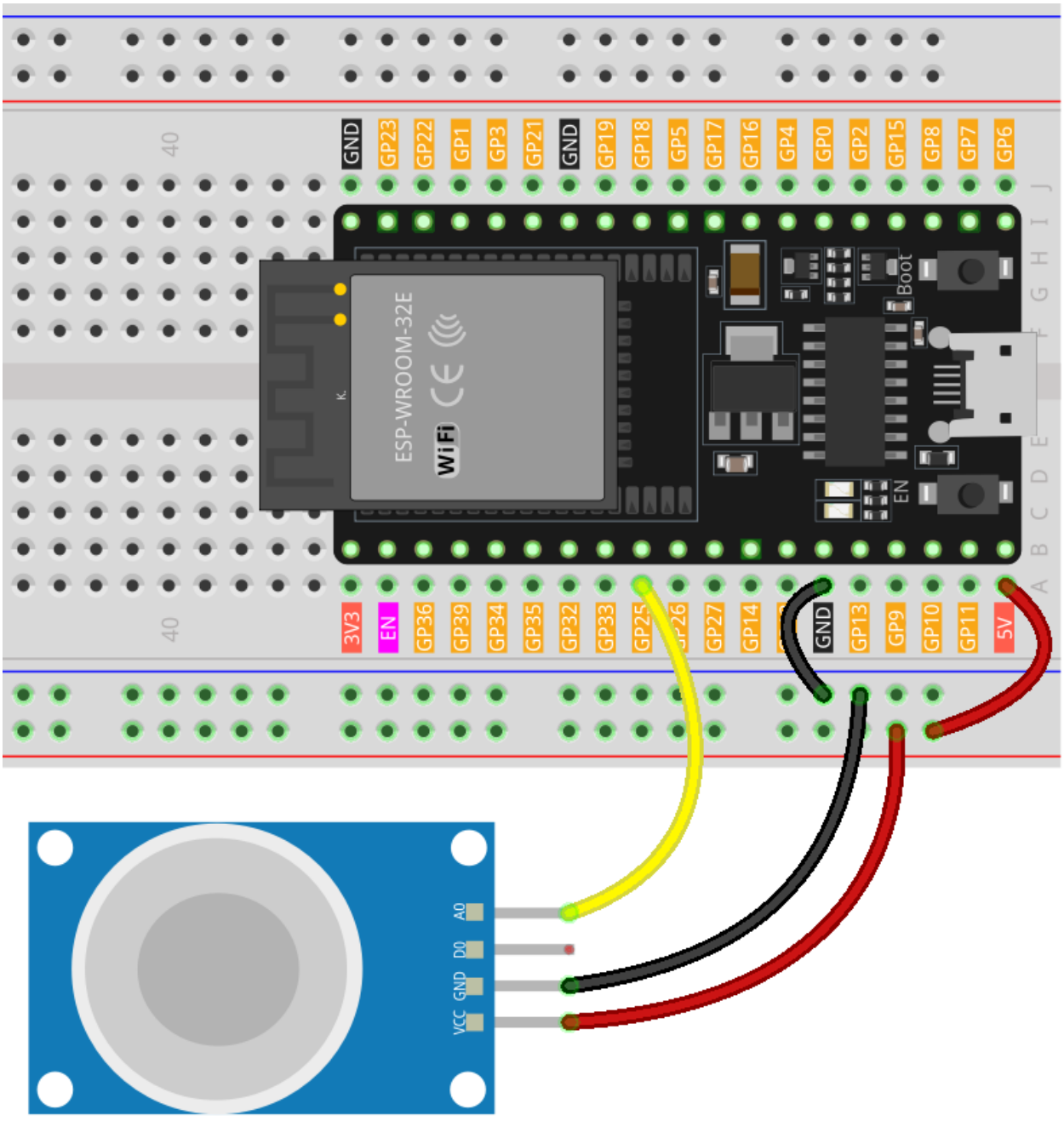
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Gas/Smoke Sensor Module (MQ2)	
Breadboard	

Wiring


GND
GP23
GP22
GP1
GP3
GP21
GND
GP19
GP18
GP5
GP17
GP16
GP4
GP0
GP2
GP15
GP8
GP7
GP6
ESP-WROOM-32E
WiFi
Boot
EN
3V3
EN
GP36
GP39
GP34
GP35
GP32
GP33
GP25
GP27
GP14
GND
GP13
GP9
GP10
GP11
5V
A0
D0
GND
VCC

Code

Code Analysis

1. The first line of code is a constant integer declaration for the gas sensor pin. We use the pin 25 to read the output from the gas sensor.

```
const int sensorPin = 25;
```

2. The `setup()` function is where we initialize our serial communication at a baud rate of 9600. This is necessary to print the readings from the gas sensor to the serial monitor.

```
void setup() {
  Serial.begin(9600);  // Start serial communication at 9600 baud rate
}
```

3. The `loop()` function is where we continuously read the analog value from the gas sensor and print it to the serial monitor. We use the `analogRead()` function to read the analog value from the sensor. We then wait for 50 milliseconds before the next reading. This delay gives some breathing space for the serial monitor to process the data.

Note: MQ2 is a heating-driven sensor that usually requires preheating before use. During the preheating period, the sensor typically reads high and gradually decreases until it stabilizes.

```
void loop() {
  Serial.print("Analog output: ");
  Serial.println(analogRead(sensorPin));  // Read the analog value of the gas␣
↪sensor and print it to the serial monitor
  delay(50);                              // Wait for 50 milliseconds
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.6 Lesson 05: Gyroscope & Accelerometer Module (MPU6050)

In this lesson, you will learn how to connect the MPU6050 accelerometer and gyroscope sensor to an ESP32 Development Board. We will go through setting up the Adafruit_MPU6050 library, initializing the sensor, and configuring its accelerometer and gyro ranges. You'll also learn how to read acceleration, rotation, and temperature data from the sensor and display these values on the serial monitor. This project is ideal for those interested in exploring motion tracking and orientation sensing in their projects, providing a practical experience in working with advanced sensors on the Arduino-compatible ESP32 platform.

Required Components

In this project, we need the following components.

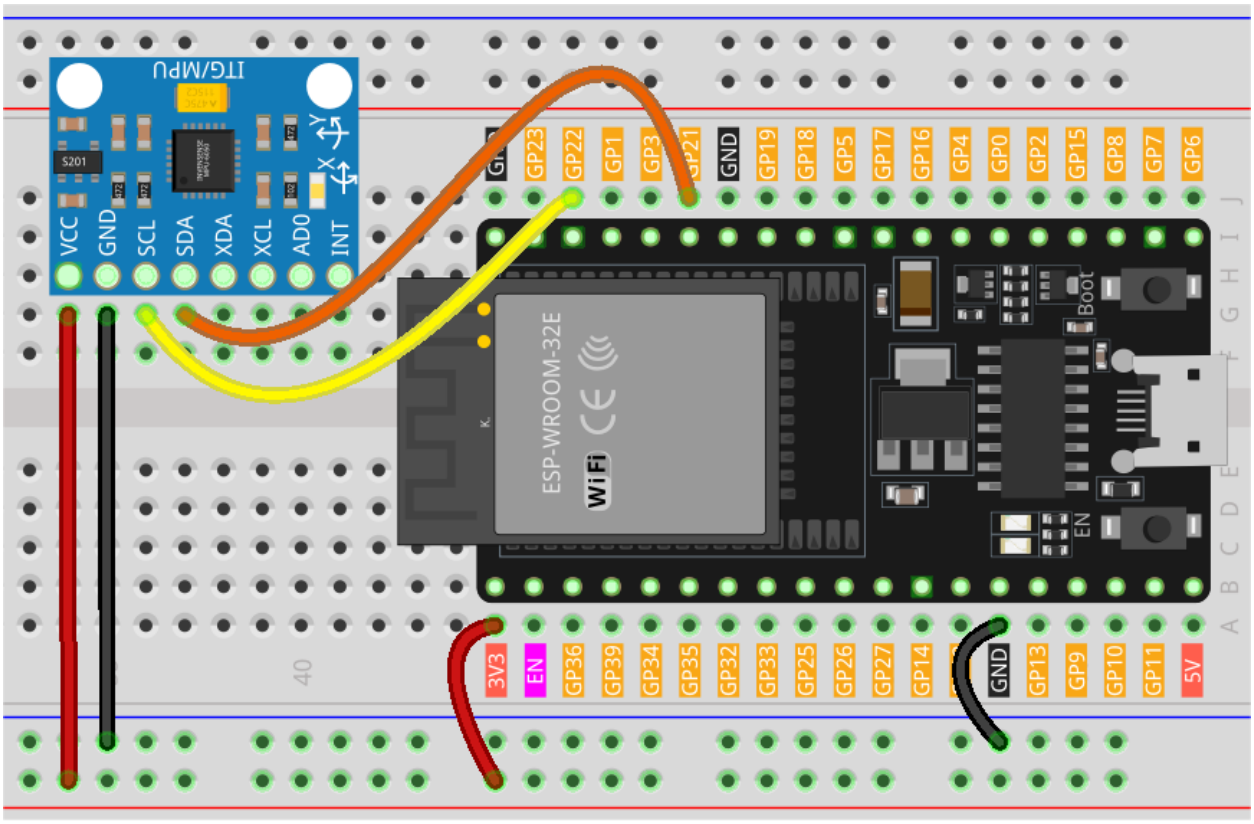
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Gyroscope & Accelerometer Module (MPU6050)	
Breadboard	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"Adafruit MPU6050"** and install it.

Code Analysis

1. The code starts by including the necessary libraries and creating an object for the MPU6050 sensor. This code uses the Adafruit_MPU6050 library, Adafruit_Sensor library, and Wire library. The `Adafruit_MPU6050` library is used to interact with the MPU6050 sensor and retrieve acceleration, rotation, and temperature data. The `Adafruit_Sensor` library provides a common interface for various types of sensors. The `Wire` library is used for I2C communication, which is necessary to communicate with the MPU6050 sensor.

   Note: To install the library, use the Arduino Library Manager and search for **"Adafruit MPU6050"** and install it.

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
Adafruit_MPU6050 mpu;
```

2. The `setup()` function initializes the serial communication and checks if the sensor is detected. If the sensor is not found, the Arduino enters an infinite loop with a "Failed to find MPU6050 chip" message. If found, the accelerometer range, gyro range, and filter bandwidth are set, and a delay is added for stability.

```
void setup(void) {
  // Initialize the serial communication
  Serial.begin(9600);

  // Check if the MPU6050 sensor is detected
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");

  // set accelerometer range to +-8G
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);

  // set gyro range to +- 500 deg/s
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);

  // set filter bandwidth to 21 Hz
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);

  // Add a delay for stability
  delay(100);
}
```

3. In the `loop()` function, the program creates events to store the sensor readings and then retrieves the readings. The acceleration, rotation, and temperature values are then printed to the serial monitor.

```
void loop() {
  // Get new sensor events with the readings
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  // Print out the acceleration, rotation, and temperature readings
  // ...

  // Add a delay to avoid flooding the serial monitor
  delay(1000);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.

- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.7 Lesson 06: Hall Sensor Module

In this lesson, you will learn how to use a Hall sensor with an ESP32 Development Board to detect the polarity of a magnetic field. We'll cover reading analog signals from the sensor and interpreting them to differentiate between south and north poles. This project is ideal for beginners in electronics, providing practical experience with sensors and signal processing on the ESP32 platform.

Required Components

In this project, we need the following components.

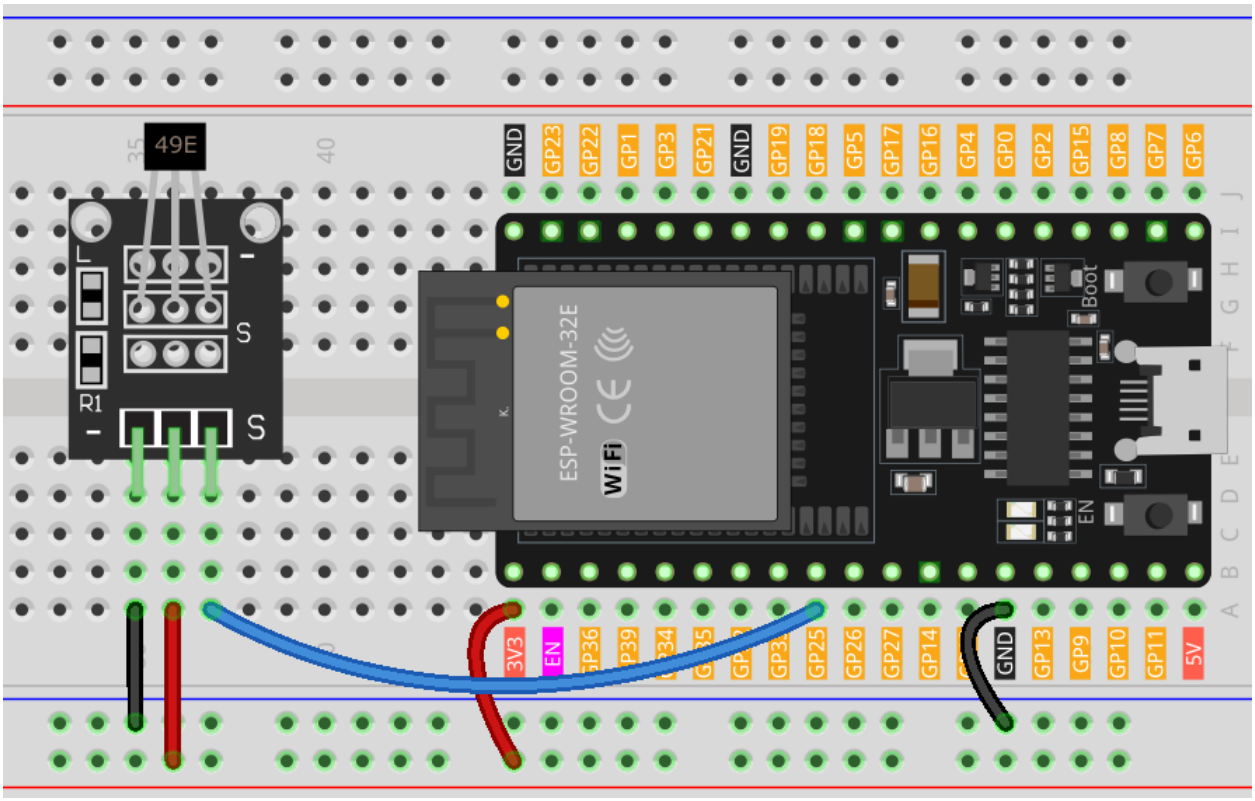
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Hall Sensor Module	-
Breadboard	

Wiring



Code

Code Analysis

1. Setting up the Hall Sensor

```
const int hallSensorPin = 25;  // Pin connected to the Hall sensor output
void setup() {
  Serial.begin(9600);             // Initialize serial communication at 9600 bps
  pinMode(hallSensorPin, INPUT);  // Set hall sensor pin as input
}
```

The hall sensor's output is connected to pin 25 on the ESP32 Development Board. The `setup()` function is used to initialize serial communication at 9600 bits per second (bps) for displaying data on the serial monitor. The `pinMode()` function is used to configure 25 as an input pin.

2. Reading from the Hall Sensor and Determining Polarity

The Hall sensor module is equipped with a 49E linear Hall effect sensor, which can measure the polarity of the magnetic field's north and south poles as well as the relative strength of the magnetic field. If you place a magnet's south pole near the side marked with 49E (the side with text engraved on it), the value read by the code will increase linearly in proportion to the applied magnetic field strength. Conversely, if you place a north pole near this side, the value read by the code will decrease linearly in proportion to that magnetic field strength. For more details, please refer to *Hall Sensor Module*.

```
void loop() {
  int sensorValue = analogRead(hallSensorPin);  // Read analog value from Hall␣
↪sensor
  Serial.print(sensorValue);                    // Output raw sensor value to␣
↪Serial Monitor
  delay(200);                                   // Delay for 200 milliseconds

  // Determine magnetic pole based on sensor value
  if (sensorValue >= 2600) {
    Serial.print(" - South pole detected");  // South pole detected if value >= 2600
  } else if (sensorValue <= 1200) {
    Serial.print(" - North pole detected");  // North pole detected if value <= 1200
  }

  Serial.println();  // New line for next output
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.8 Lesson 07: Infrared Speed Sensor Module

In this lesson, you'll learn how to use an ESP32 Development Board with a Speed Sensor Module to detect obstructions. We'll see how the sensor sends a high signal when there's an obstruction and a low signal when the path is clear. This project is ideal for those looking to grasp sensor integration and basic input/output operations in a practical setting using the ESP32 platform.

Required Components

In this project, we need the following components.

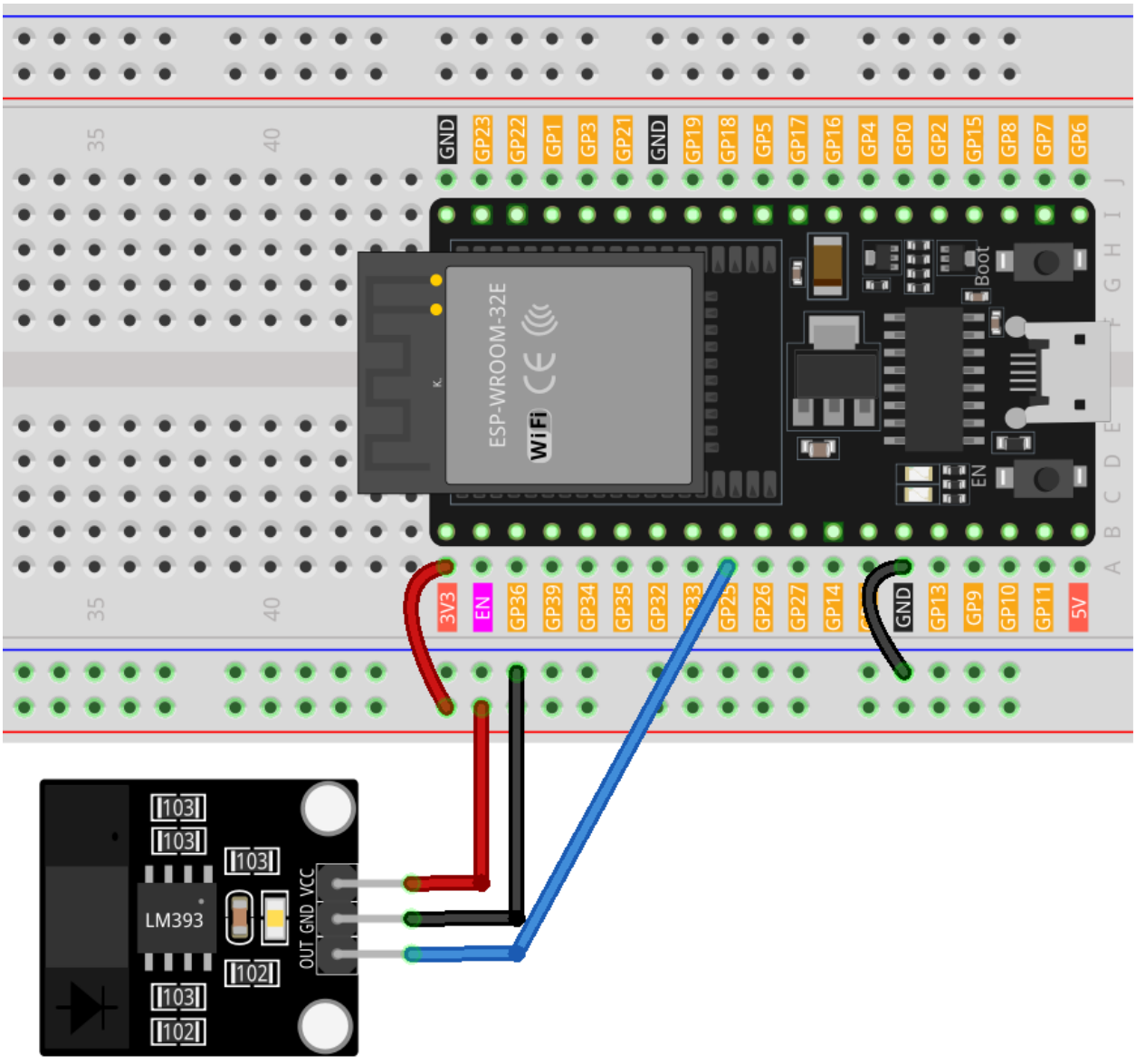
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Breadboard	
Infrared Speed Sensor Module	

Wiring



Code

Code Analysis

1. Define the sensor pin

   The sensor pin is declared as a constant integer and is assigned pin number 25 of the ESP32.

   ```
   const int sensorPin = 25;
   ```

2. Setup function

   This function initializes the serial communication at 9600 baud rate and sets the sensorPin as an input.

   ```
   void setup() {
     Serial.begin(9600);
     pinMode(sensorPin, INPUT);
   }
   ```

3. Loop function

   The loop function continuously checks the sensor pin's status. If the sensor pin reads HIGH, it prints "Obstruction detected" to the Serial Monitor. If the sensor pin is LOW, it prints "Unobstructed".

   ```
   void loop() {
     if (digitalRead(sensorPin) == HIGH) {
       Serial.println("Obstruction detected");
     } else {
       Serial.println("Unobstructed");
     }
   }
   ```

4. More

   If an encoder is mounted on the motor, the rotational speed of the motor can be calculated by counting the number of times an obstruction passes the sensor within a specific period.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.9 Lesson 08: IR Obstacle Avoidance Sensor Module

In this lesson, you'll learn how to use an Infrared obstacle avoidance sensor with an ESP32 Development Board. We'll explore how the sensor detects obstacles and alters its output signal. You'll also learn how to read these signals using the ESP32 and display them on the serial monitor. This project provides a great opportunity for beginners to gain hands-on experience with sensors and digital input processing on the ESP32 platform, making it perfect for those interested in building interactive projects.

Required Components

In this project, we need the following components.

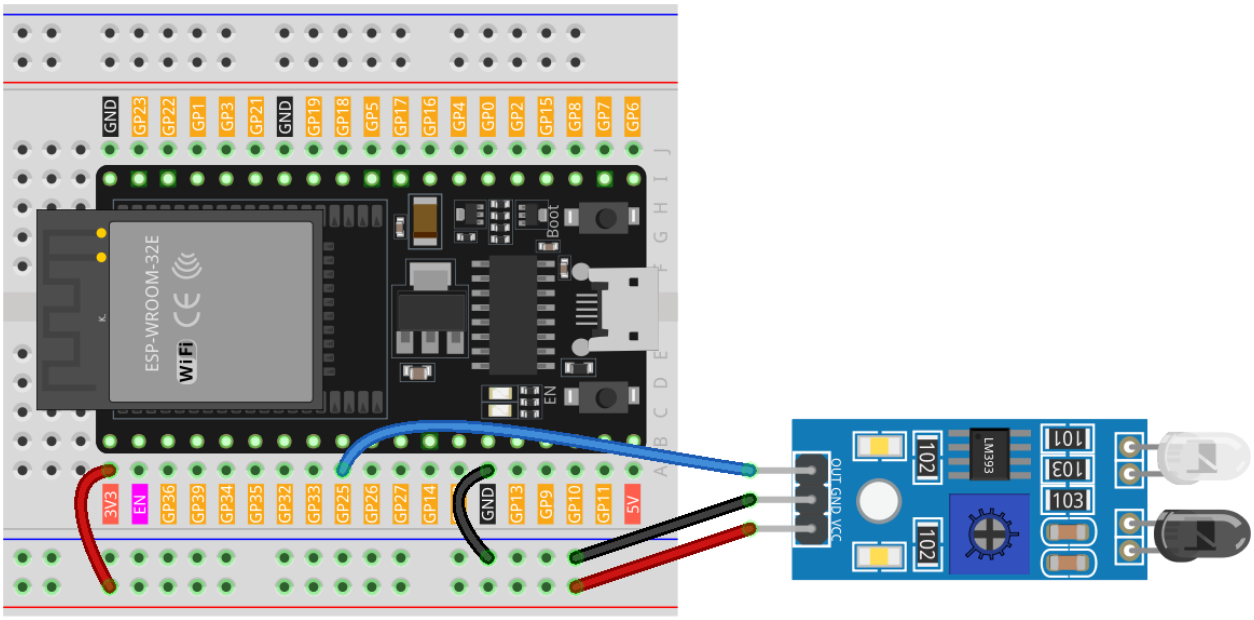
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
IR Obstacle Avoidance Sensor Module	
Breadboard	

Wiring



Code

Code Analysis

1. Define pin number for sensor connection:

   ```
   const int sensorPin = 25;
   ```

   Connect the sensor's output pin to pin 25.

2. Setup serial communication and define sensor pin as input:

   ```
   void setup() {
     pinMode(sensorPin, INPUT);
     Serial.begin(9600);
   }
   ```

   Initialize serial communication at 9600 baud rate to print to serial monitor. Set sensor pin as input to read input signal.

3. Read sensor value and print to serial monitor:

   ```
   void loop() {
     Serial.println(digitalRead(sensorPin));
     delay(50);
   }
   ```

   Continuously read digital value from sensor pin using `digitalRead()` and print value to serial monitor using `Serial.println()`. Add 50ms delay between prints for better viewing.

Note: If the sensor is not working properly, adjust the IR transmitter and receiver to make them parallel. Additionally, you can adjust the detection range using the built-in potentiometer.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.10 Lesson 09: Joystick Module

In this lesson, you will learn how to read values from a joystick module using the ESP32 Development Board. We'll cover measuring the X and Y axis movements of the joystick and interpreting the switch position. By integrating these inputs with the ESP32, you'll gain insights into handling analog and digital signals. This project is perfect for beginners, providing hands-on experience in reading and processing data from interactive hardware components.

Required Components

In this project, we need the following components.

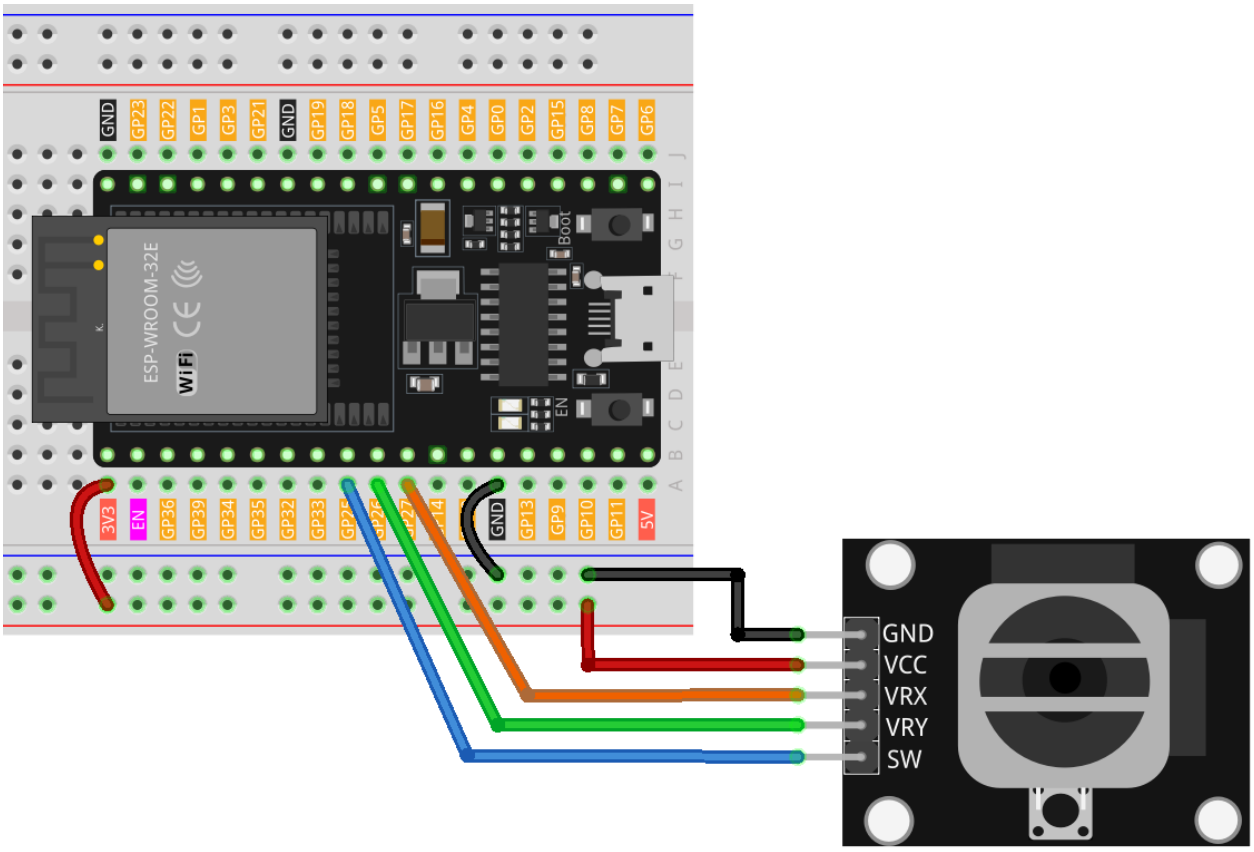
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Joystick Module	
Breadboard	

Wiring



Code

Code Analysis

1. Pin Definitions:

```
const int xPin = 27;  //the VRX attach to
const int yPin = 26;  //the VRY attach to
const int swPin = 25;  //the SW attach to
```

Constants for the joystick pins are defined. `xPin` and `yPin` are analog pins for the joystick's X and Y axes. `swPin` is a digital pin for the joystick's switch.

2. Setup Function:

```
void setup() {
  pinMode(swPin, INPUT_PULLUP);
  Serial.begin(9600);
}
```

Initializes `swPin` as an input with a pull-up resistor, essential for the switch's functionality. Starts serial communication at 9600 baud.

3. Main Loop:

```
void loop() {
  Serial.print("X: ");
  Serial.print(analogRead(xPin));  // print the value of VRX
  Serial.print("|Y: ");
  Serial.print(analogRead(yPin));  // print the value of VRY
  Serial.print("|Z: ");
  Serial.println(digitalRead(swPin));  // print the value of SW
  delay(50);
}
```

Continuously reads and prints the values from the joystick's axes and switch to the Serial Monitor, with a delay of 50 ms between readings.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.11 Lesson 10: PCF8591 ADC DAC Converter Module

In this lesson, you'll learn how to connect the ESP32 Development Board with a PCF8591 ADC DAC Converter Module. We'll cover reading analog values from input AIN0, sending these values to the DAC(AOUT), and displaying both the raw and voltage-converted readings on the serial monitor. The module's potentiometer is connected to AIN0 using jumper caps, and the D2 LED on the module is connected to AOUT, so you can see that the brightness of D2 LED changes as you rotate the potentiometer.

Required Components

In this project, we need the following components.

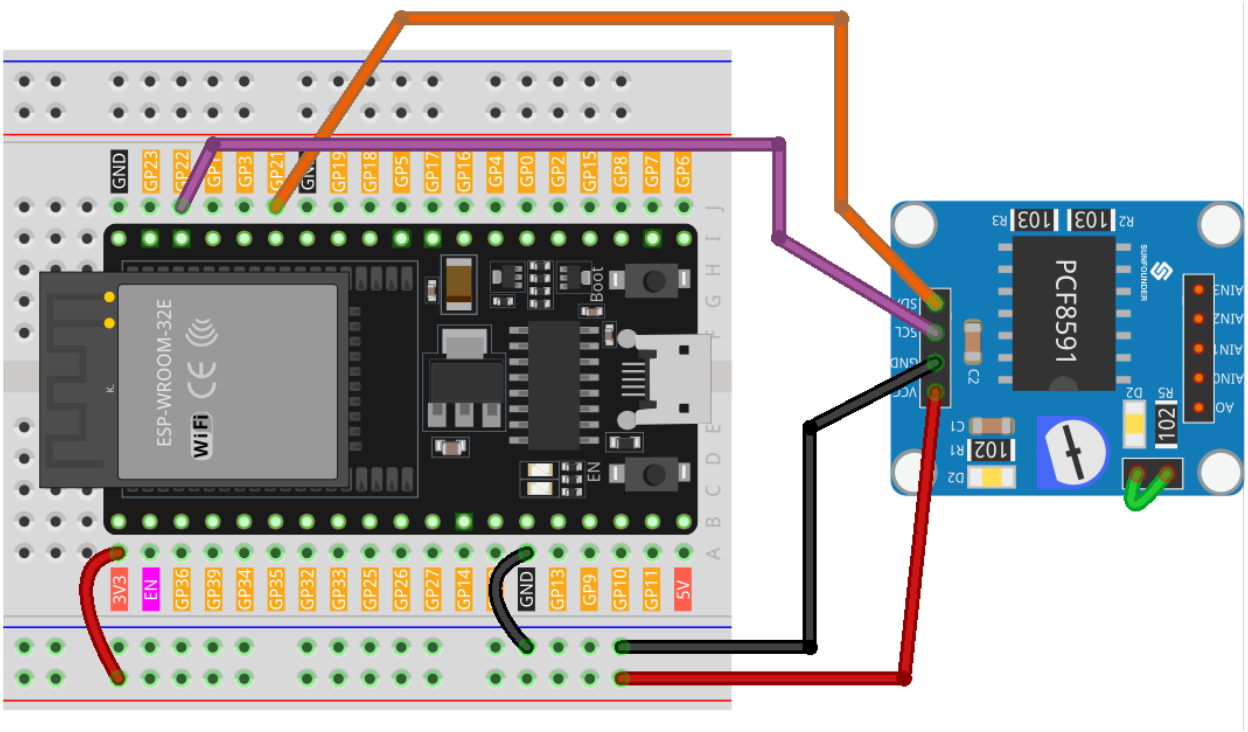
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
PCF8591 ADC DAC Converter Module	
Breadboard	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"Adafruit PCF8591"** and install it.

Code Analysis

1. **Including the Library and Defining Constants**

   Note: To install the library, use the Arduino Library Manager and search for **"Adafruit PCF8591"** and install it.

```
// Include Adafruit PCF8591 library
#include <Adafruit_PCF8591.h>
// Define the reference voltage for ADC conversion
#define ADC_REFERENCE_VOLTAGE 3.3
```

This section includes the Adafruit PCF8591 library, which provides functions for interacting with the PCF8591 module. The ADC reference voltage is set to 3.3 volts, which is the maximum voltage that the ADC can measure.

2. **Setting Up the PCF8591 Module**

```
// Create an instance of the PCF8591 module
Adafruit_PCF8591 pcf = Adafruit_PCF8591();
void setup() {
  Serial.begin(9600);
  Serial.println("# Adafruit PCF8591 demo");
  if (!pcf.begin()) {
    Serial.println("# PCF8591 not found!");
    while (1) delay(10);
  }
  Serial.println("# PCF8591 found");
  pcf.enableDAC(true);
}
```

In the setup function, serial communication is started, and an instance of the PCF8591 module is created. The `pcf.begin()` function checks if the module is connected properly. If not, it prints an error message and halts the program. If the module is found, it enables the DAC.

3. **Reading from ADC and Writing to DAC**

```
void loop() {
  AIN0 = pcf.analogRead(0);
  pcf.analogWrite(AIN0);
  Serial.print("AIN0: ");
  Serial.print(AIN0);
  Serial.print(", ");
  Serial.print(int_to_volts(AIN0, 8, ADC_REFERENCE_VOLTAGE));
  Serial.println("V");
  delay(500);
}
```

The loop function continuously reads the analog value from AIN0 (analog input 0) of the PCF8591 module, then writes this value back to the DAC. It also prints the raw value and the voltage-converted value of AIN0 to the Serial Monitor.

Jumper caps link the module's potentiometer to AIN0, and the D2 LED is connected to AOUT; please refer to the PCF8591 module *schematic* for details. The brightness of the LED changes as the potentiometer is rotated.

4. **Digital to Voltage Conversion Function**

```
float int_to_volts(uint16_t dac_value, uint8_t bits, float logic_level) {
  return (((float)dac_value / ((1 << bits) - 1)) * logic_level);
}
```

This function converts the digital value back to its corresponding voltage. It takes the digital value (`dac_value`), the number of bits of resolution (`bits`), and the logic level voltage (`logic_level`) as arguments. The formula used is a standard approach to convert a digital value to its equivalent voltage.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.12 Lesson 11: Photoresistor Module

In this lesson, you will learn how to use a photoresistance sensor with an ESP32 Development Board to measure light intensity. We'll explore how the sensor detects different light levels and processes and displays these readings on the serial monitor. This project is ideal for beginners as it provides hands-on experience with analog sensors and real-time data handling in Arduino programming.

Required Components

In this project, we need the following components.

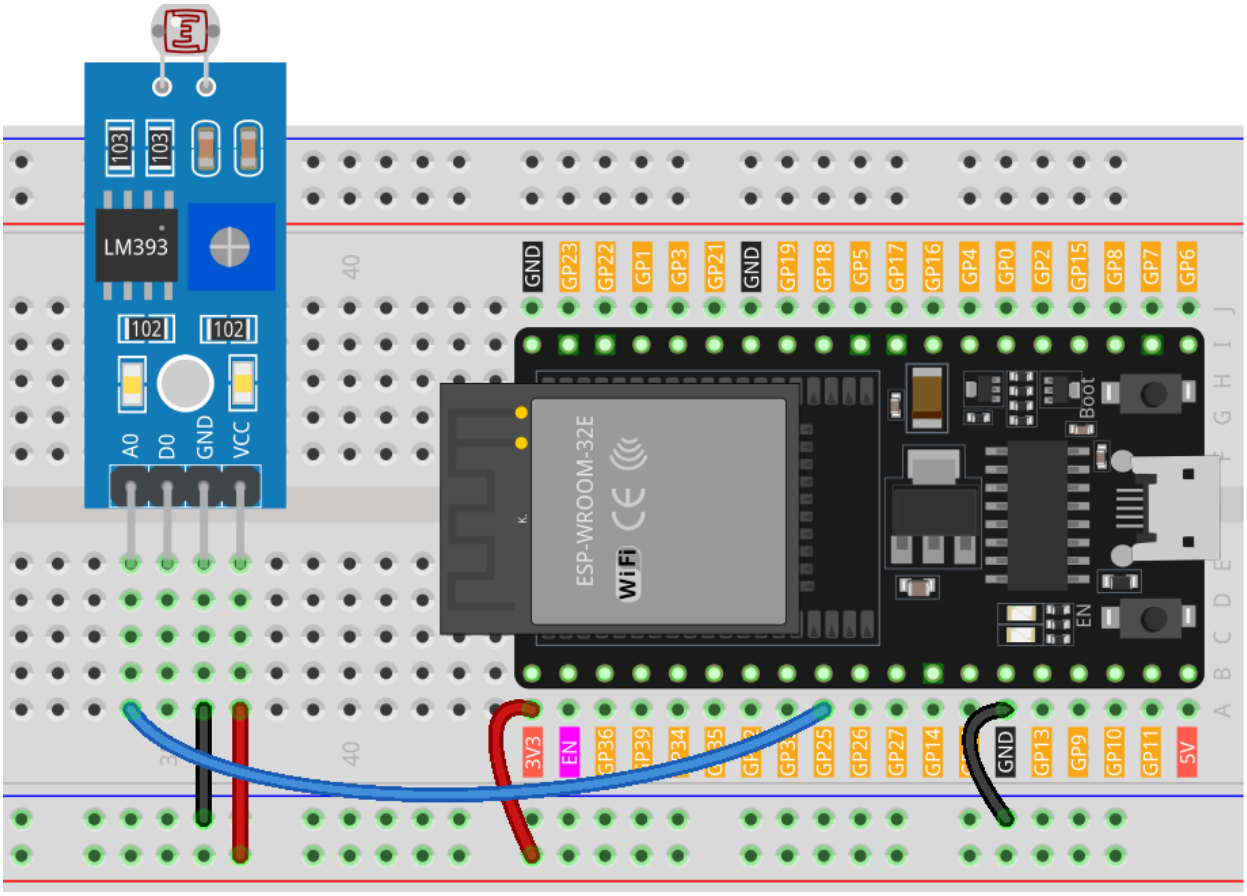
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Photoresistor Module	
Breadboard	

Wiring



Code

Code Analysis

1. **Setting Up the Sensor Pin and Serial Communication**

   We start by defining the sensor pin and initializing serial communication in the setup function. The photoresistor is connected to the pin 25.

```
const int sensorPin = 25;  // Pin connected to the photoresistor

void setup() {
  Serial.begin(9600);  // Start serial communication at 9600 baud rate
}
```

2. **Reading and Displaying Sensor Data**

   In the loop function, we continuously read the analog value from the sensor and print it to the Serial Monitor. We also add a short delay to stabilize the readings.

```
void loop() {
  Serial.println(analogRead(sensorPin));  // Read and print the analog value
```

(continues on next page)

(continued from previous page)

```
  delay(50);                                    // Short delay to stabilize readings
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.13 Lesson 12: PIR Motion Module (HC-SR501)

In this lesson, you will learn how to use a PIR (Passive Infrared) motion sensor with an ESP32 Development Board. You'll learn how to read digital inputs from the sensor to detect motion and output a corresponding message to the serial monitor. We'll cover the setup and programming required for the ESP32 board to respond when the sensor detects someone's presence by displaying "Somebody here!"

Required Components

In this project, we need the following components.

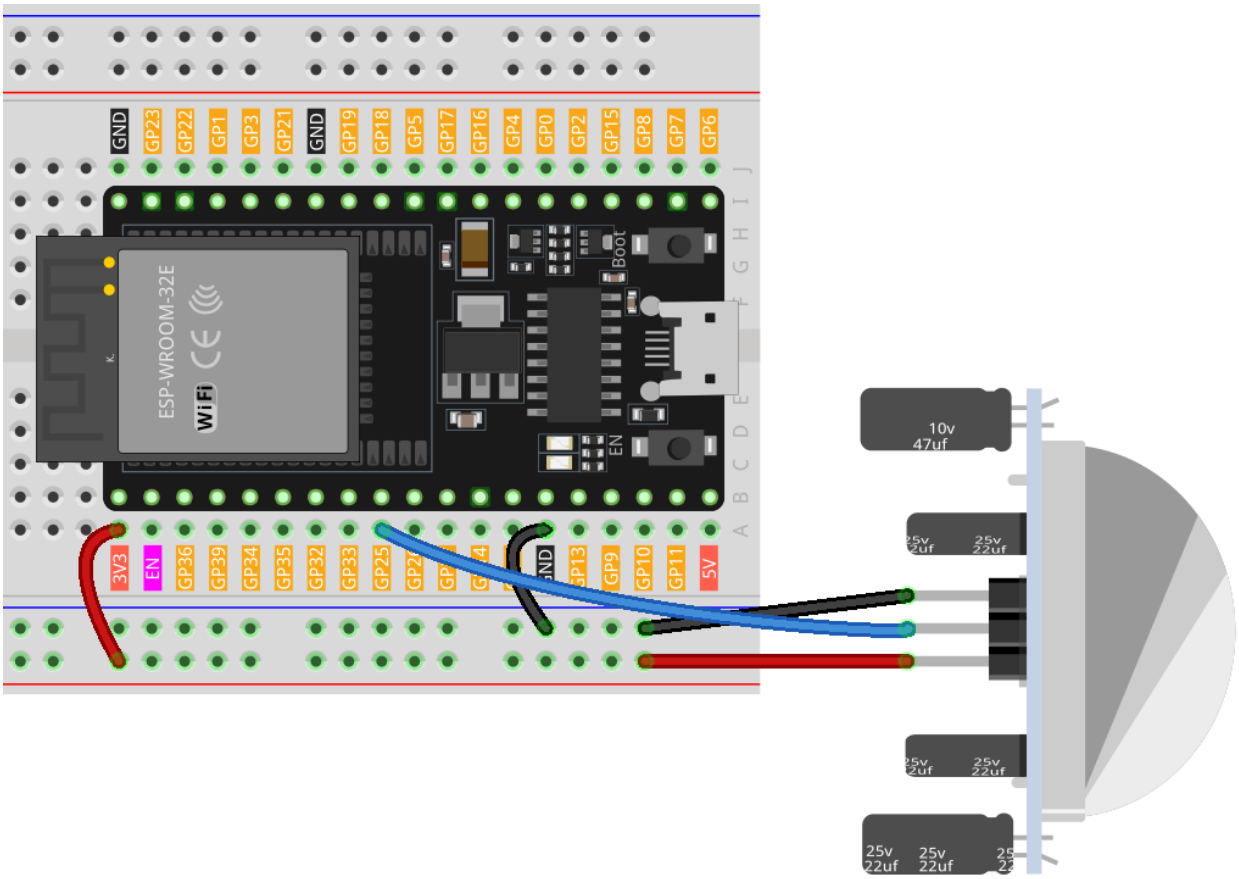
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
PIR Motion Module (HC-SR501)	-
Breadboard	

Wiring



Code

Code Analysis

1. Setting up the PIR Sensor Pin. The pin for the PIR sensor is defined as pin 25.

```
const int pirPin = 25;
int state = 0;
```

2. Initializing the PIR Sensor. In the `setup()` function, the PIR sensor pin is set as an input. This allows the Arduino to read the state of the PIR sensor.

```
void setup() {
  pinMode(pirPin, INPUT);
  Serial.begin(9600);
}
```

3. Reading from the PIR Sensor and Displaying the Results. In the `loop()` function, the state of the PIR sensor is continuously read.

```
void loop() {
  state = digitalRead(pirPin);
```

(continues on next page)

(continued from previous page)

```
  if (state == HIGH) {
    Serial.println("Somebody here!");
  } else {
    Serial.println("Monitoring...");
    delay(100);
  }
}
```

If the state is `HIGH`, meaning motion is detected, a message "Somebody here!" is printed to the serial monitor. Otherwise, "Monitoring…" is printed.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.14 Lesson 13: Potentiometer Module

In this lesson, you'll learn how to read the analog value of a potentiometer with the ESP32 development board. We'll connect a potentiometer module to pin 25 and observe the changing analog values (0-4095) in the serial monitor. This project provides hands-on experience in understanding analog inputs and serial communication, making it an excellent exercise for beginners to explore the capabilities of the ESP32 board.

Required Components

In this project, we need the following components.

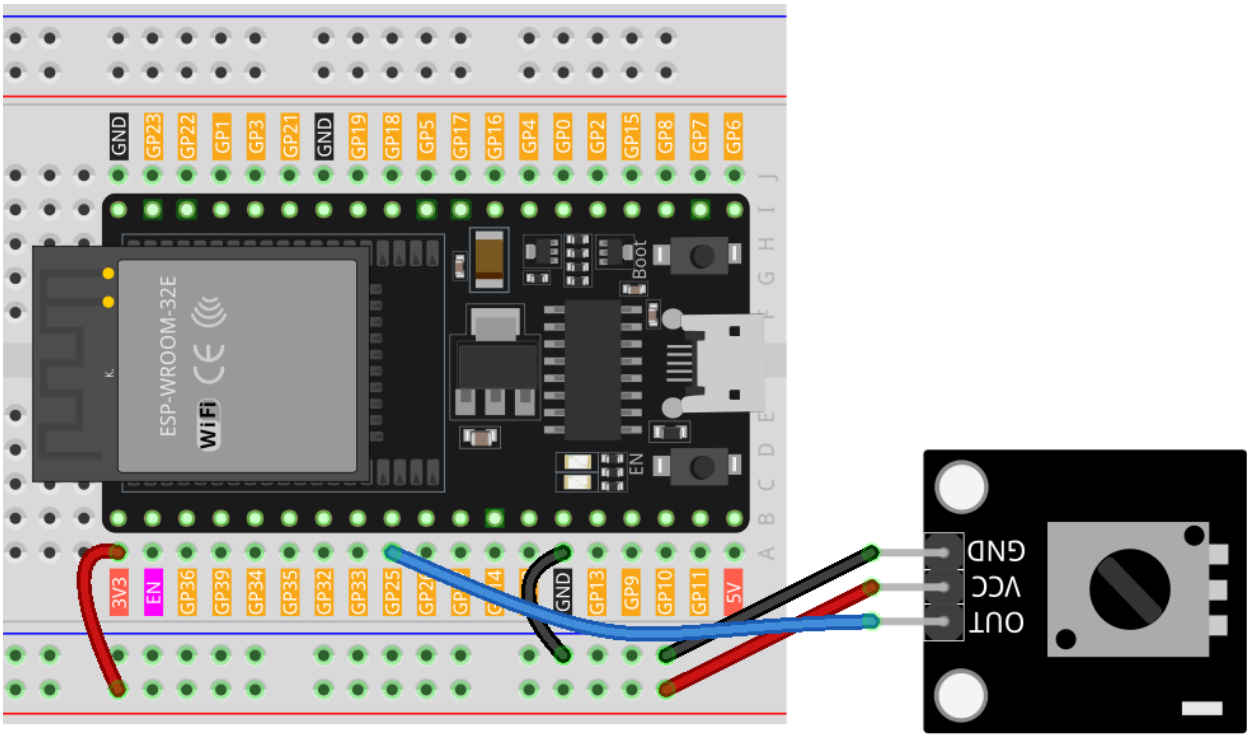
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Potentiometer Module	
Breadboard	

Wiring



Code

Code Analysis

1. This line of code defines the pin number to which the potentiometer is connected on the ESP32 Development Board.

```
const int sensorPin = 25;
```

2. The `setup()` function is a special function in Arduino that is executed only once when the ESP32 Development Board is powered on or reset. In this project, the `Serial.begin(9600)` command initiates serial communication at a baud rate of 9600.

```
void setup() {
  Serial.begin(9600);
}
```

3. The `loop()` function is the main function where the program runs repeatedly. In this function, the `analogRead()` function reads the analog value from the potentiometer and prints it to the serial monitor using `Serial.println()`. The `delay(50)` command makes the program wait for 50 milliseconds before taking the next reading.

```
void loop() {
  Serial.println(analogRead(sensorPin));
  delay(50);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.15 Lesson 14: Pulse Oximeter and Heart Rate Sensor Module (MAX30102)

In this lesson, you will learn how to measure heart rate using an ESP32 Development Board and MAX30102 Pulse Oximeter and Heart Rate Sensor. We'll cover setting up the sensor, reading infrared values, and accurately calculating beats per minute (BPM). This project is ideal for those interested in health monitoring systems and provides a valuable introduction to working with biomedical sensors using the ESP32.

Warning: This project detects heart-rate optically. This method is tricky and prone to give false readings. So please **DO NOT** use it for actual medical diagnosis.

Required Components

In this project, we need the following components.

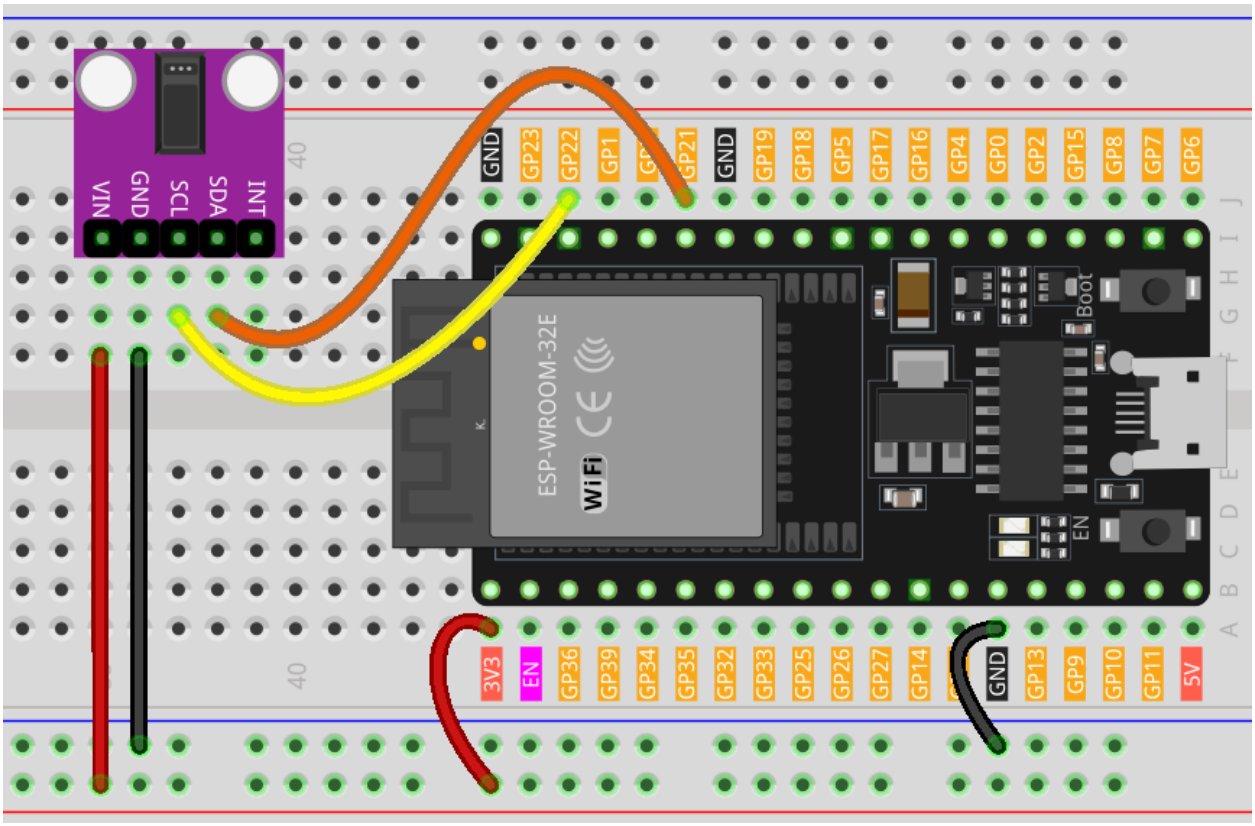
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Pulse Oximeter and Heart Rate Sensor Module (MAX30102)	
Breadboard	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"SparkFun MAX3010x"** and install it.

Code Analysis

1. **Including Libraries & Initializing Global Variables**:

   The essential libraries are imported, the sensor object is instantiated, and global variables for data management are set.

   Note: To install the library, use the Arduino Library Manager and search for **"SparkFun MAX3010x"** and install it.

```
#include <Wire.h>
#include "MAX30105.h"
#include "heartRate.h"
MAX30105 particleSensor;
// ... (other global variables)
```

2. **Setup Function & Sensor Initialization**:

   The Serial communication is initialized at a baud rate of 9600. The sensor's connection is checked, and if successful, an initialization sequence is run. An error message is displayed if the sensor isn't detected.

```
void setup() {
  Serial.begin(9600);
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) {
    Serial.println("MAX30102 not found.");
    while (1) ;  // Infinite loop if sensor not detected.
  }
  // ... (further setup)
```

3. **Reading IR Value & Checking for Heartbeat**:

   The IR value, which is indicative of the blood flow, is fetched from the sensor. The `checkForBeat()` function assesses if a heartbeat is detected based on this value.

```
long irValue = particleSensor.getIR();
if (checkForBeat(irValue) == true) {
    // ... (heartbeat detected actions)
}
```

4. **Calculating Beats Per Minute (BPM)**:

   Upon detecting a heartbeat, the BPM is calculated based on the time difference since the last detected heartbeat. The code also ensures the BPM falls within a realistic range before updating the average.

```
long delta = millis() - lastBeat;
beatsPerMinute = 60 / (delta / 1000.0);
if (beatsPerMinute < 255 && beatsPerMinute > 20) {
```

(continues on next page)

(continued from previous page)

```
    // ... (store and average BPM)
}
```

5. **Printing Values to the Serial Monitor**:

   The IR value, current BPM, and average BPM are printed to the Serial Monitor. Additionally, the code checks if the IR value is too low, suggesting the absence of a finger.

```
//Print the IR value, current BPM value, and average BPM value to the serial monitor
Serial.print("IR=");
Serial.print(irValue);
Serial.print(", BPM=");
Serial.print(beatsPerMinute);
Serial.print(", Avg BPM=");
Serial.print(beatAvg);

if (irValue < 50000)
  Serial.print(" No finger?");
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.16 Lesson 15: Raindrop Detection Module

In this lesson, you will learn how to use a raindrop detection sensor with an ESP32 Development Board. We'll cover reading digital signals from the sensor when it detects rainwater and displaying this information on the serial monitor. This project provides an engaging way to grasp digital input and output in microcontroller programming, making it ideal for beginners in electronics and coding with the ESP32 platform.

Required Components

In this project, we need the following components.

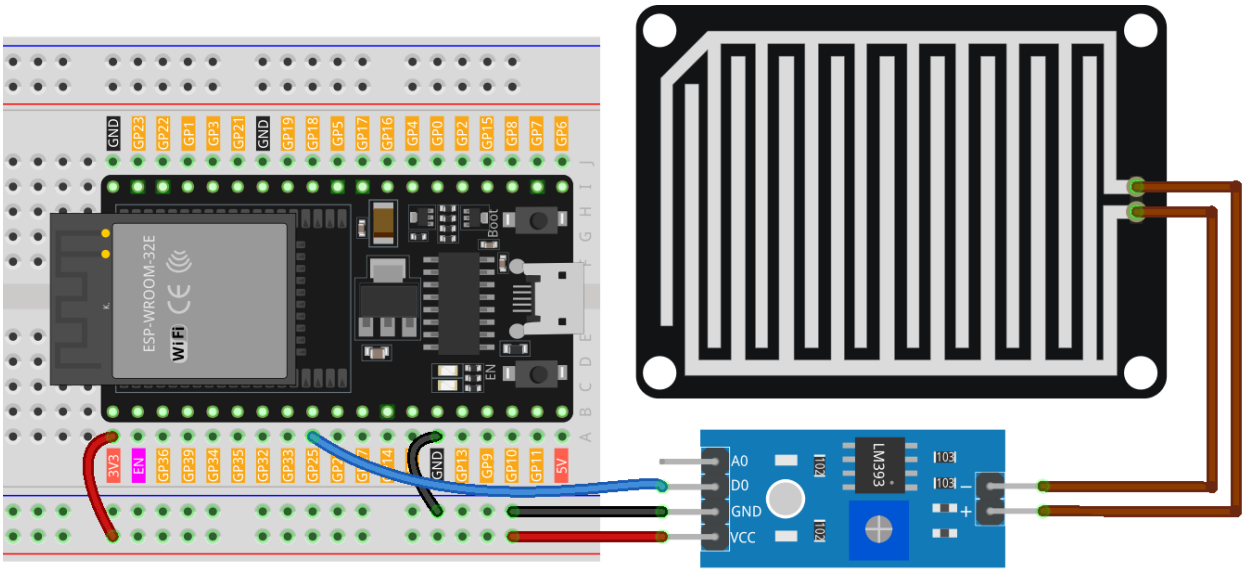
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Raindrop Detection Module	
Breadboard	

Wiring



Code

Code Analysis

1. Defining sensor pin

   Here, a constant integer named `sensorPin` is defined and assigned the value 25. This corresponds to the digital pin on the ESP32 Development Board where the raindrops detection sensor is connected.

   ```
   const int sensorPin = 25;
   ```

2. Setting up the pin mode and initiating serial communication.

   In the `setup()` function, two essential steps are performed. Firstly, `pinMode()` is used to set the `sensorPin` as an input, enabling us to read digital values from the raindrops sensor. Secondly, serial communication is initialized with a baud rate of 9600.

   ```
   void setup() {
     pinMode(sensorPin, INPUT);
     Serial.begin(9600);
   }
   ```

3. Reading the digital value and sending it to the serial monitor.

   The `loop()` function reads the digital value from the raindrops sensor using `digitalRead()`. This value (either HIGH or LOW) is printed to the Serial Monitor. When raindrops are detected, the serial monitor will display 0; when no raindrops are detected, it will display 1. The program then waits for 50 milliseconds before the next reading.

```
void loop() {
  Serial.println(digitalRead(sensorPin));
  delay(50);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.17 Lesson 16: Real Time Clock Module (DS1302)

In this lesson, you will learn how to set up and use a Real Time Clock (RTC) module with an ESP32 development board. We'll cover integrating the DS1302 RTC module, understanding its functions, and programming the ESP32 to display the current date and time. You'll also learn how to handle situations where the RTC has lost its date and time settings and automatically set it to the compile time of your sketch. This project is ideal for those seeking to enhance their comprehension of time-related functions in microcontroller projects.

Required Components

In this project, we need the following components.

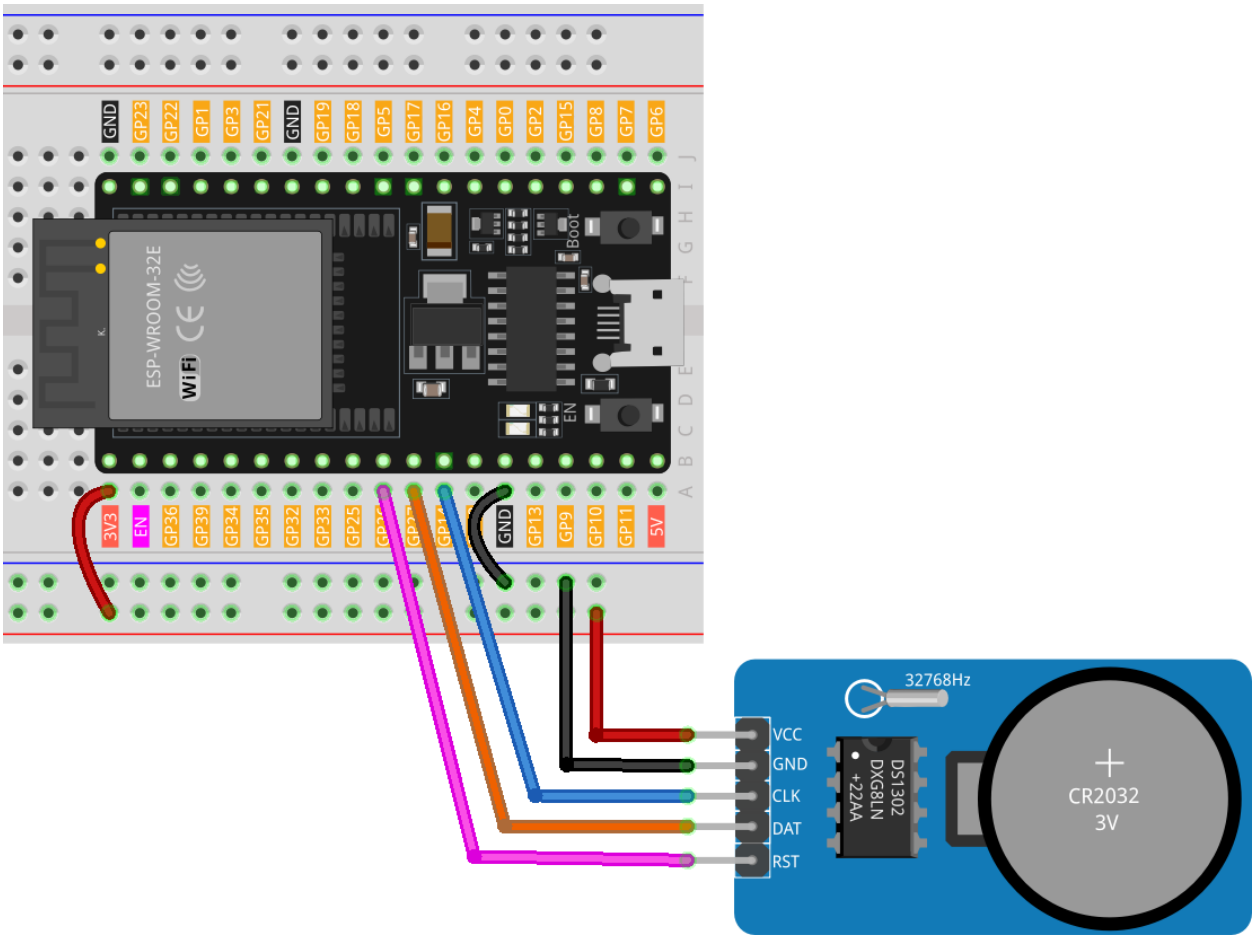
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Real Time Clock Module (DS1302)	
Breadboard	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"Rtc by Makuna"** and install it.

Code Analysis

1. Initialization and library inclusion

   Note: To install the library, use the Arduino Library Manager and search for **"Rtc by Makuna"** and install it.

   Here, necessary libraries are included for the DS1302 RTC module.

```
#include <ThreeWire.h>
#include <RtcDS1302.h>
```

2. Define pins and create RTC instance

   Pins for communication are defined and an instance of the RTC is created.

```
const int IO = 27;    // DAT
const int SCLK = 14;  // CLK
const int CE = 26;    // RST

ThreeWire myWire(IO, SCLK, CE));
RtcDS1302<ThreeWire> Rtc(myWire);
```

3. `setup()` function

   This function initializes the serial communication and sets up the RTC module. Various checks are made to ensure the RTC is running correctly.

```
void setup() {
  Serial.begin(9600);

  Serial.print("compiled: ");
  Serial.print(__DATE__);
  Serial.println(__TIME__);

  Rtc.Begin();

  RtcDateTime compiled = RtcDateTime(__DATE__, __TIME__);
  printDateTime(compiled);
  Serial.println();

  if (!Rtc.IsDateTimeValid()) {
    // Common Causes:
    //    1) first time you ran and the device wasn't running yet
    //    2) the battery on the device is low or even missing

    Serial.println("RTC lost confidence in the DateTime!");
    Rtc.SetDateTime(compiled);
  }

  if (Rtc.GetIsWriteProtected()) {
    Serial.println("RTC was write protected, enabling writing now");
    Rtc.SetIsWriteProtected(false);
  }
```

(continues on next page)

(continued from previous page)

```
  if (!Rtc.GetIsRunning()) {
    Serial.println("RTC was not actively running, starting now");
    Rtc.SetIsRunning(true);
  }

  RtcDateTime now = Rtc.GetDateTime();
  if (now < compiled) {
    Serial.println("RTC is older than compile time!  (Updating DateTime)");
    Rtc.SetDateTime(compiled);
  } else if (now > compiled) {
    Serial.println("RTC is newer than compile time. (this is expected)");
  } else if (now == compiled) {
    Serial.println("RTC is the same as compile time! (not expected but all is fine)
↪");
  }
}
```

4. `loop()` function

   This function periodically fetches the current date and time from the RTC and prints it on the serial monitor. It also checks if the RTC is still maintaining a valid date and time.

```
void loop() {
  RtcDateTime now = Rtc.GetDateTime();

  printDateTime(now);
  Serial.println();

  if (!now.IsValid()) {
    // Common Causes:
    //    1) the battery on the device is low or even missing and the power line␣
↪was disconnected
    Serial.println("RTC lost confidence in the DateTime!");
  }

  delay(5000);  // five seconds
}
```

5. Date and time printing function

   A helper function that takes a `RtcDateTime` object and prints the formatted date and time to the serial monitor.

```
void printDateTime(const RtcDateTime& dt) {
  char datestring[20];

  snprintf_P(datestring,
             countof(datestring),
             PSTR("%02u/%02u/%04u %02u:%02u:%02u"),
             dt.Month(),
             dt.Day(),
             dt.Year(),
             dt.Hour(),
             dt.Minute(),
```

(continues on next page)

(continued from previous page)

```
            dt.Second());
  Serial.print(datestring);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.18 Lesson 17: Rotary Encoder Module

In this lesson, you will learn how to use an ESP32 Development Board and a rotary encoder module to detect rotation direction and count, as well as button presses. We'll explore how the encoder signals clockwise and counterclockwise rotations and increments or decrements a counter accordingly. Additionally, you'll understand how to detect button presses on the encoder module. This project offers hands-on experience in managing rotary encoders and reading digital inputs, enhancing your skills in working with the ESP32 and Arduino programming.

Required Components

In this project, we need the following components.

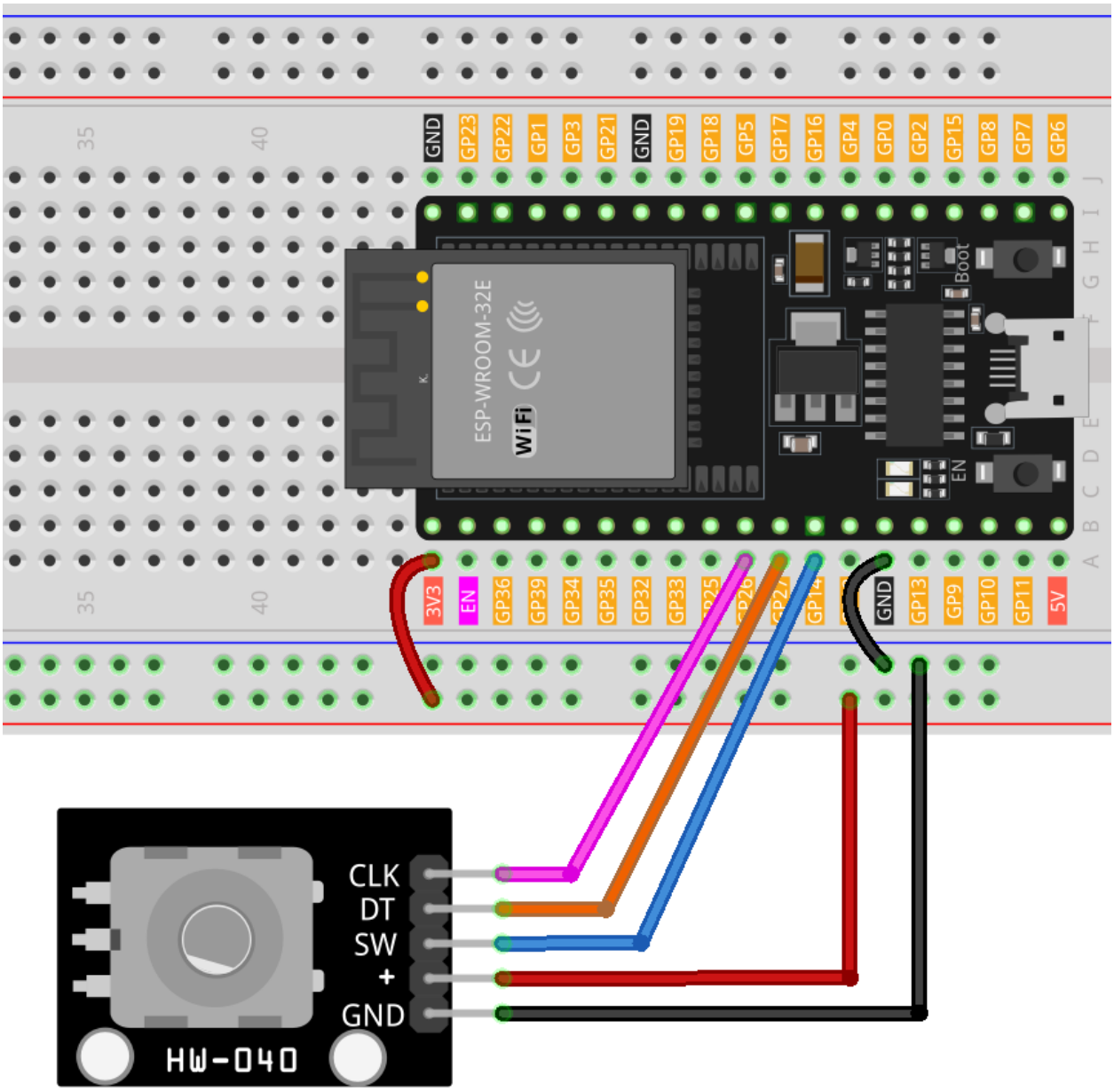
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Rotary Encoder Module	-
Breadboard	

Wiring



Code

Code Analysis

1. **Setup and Initialization**

```
void setup() {
  pinMode(CLK, INPUT);
  pinMode(DT, INPUT);
  pinMode(SW, INPUT_PULLUP);
  Serial.begin(9600);
```

(continues on next page)

(continued from previous page)

```
  lastStateCLK = digitalRead(CLK);
}
```

In the setup function, the digital pins connected to the encoder's CLK and DT are set as inputs. The SW pin, which is connected to the button, is set as an input with an internal pull-up resistor. This setup prevents the need for an external pull-up resistor. The Serial communication is started at a baud rate of 9600 to enable data visualization on the Serial Monitor. The initial state of the CLK pin is read and stored.

2. **Main Loop: Reading Encoder and Button State**

```
void loop() {
  currentStateCLK = digitalRead(CLK);
  if (currentStateCLK != lastStateCLK && currentStateCLK == 1) {
    if (digitalRead(DT) != currentStateCLK) {
      counter--;
      currentDir = "CCW";
    } else {
      counter++;
      currentDir = "CW";
    }
    Serial.print("Direction: ");
    Serial.print(currentDir);
    Serial.print(" | Counter: ");
    Serial.println(counter);
  }
  lastStateCLK = currentStateCLK;
  int btnState = digitalRead(SW);
  if (btnState == LOW) {
    if (millis() - lastButtonPress > 50) {
      Serial.println("Button pressed!");
    }
    lastButtonPress = millis();
  }
  delay(1);
}
```

In the loop function, the program continually reads the current state of the CLK pin. If there's a change in the state, it implies a rotation has occurred. The direction of rotation is determined by comparing the states of CLK and DT pins. If they are different, it indicates counterclockwise (CCW) rotation; otherwise, it's clockwise (CW). The encoder's count is incremented or decremented accordingly. This information is then sent to the Serial Monitor.

The button state is read from the SW pin. If it's LOW (pressed), a debounce mechanism is implemented by checking the time elapsed since the last button press. If more than 50 milliseconds have passed, it's considered a valid press, and a message is sent to the Serial Monitor. The *delay(1)* at the end helps in debouncing.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.19 Lesson 18: Temperature Sensor Module (DS18B20)

In this lesson, you will learn how to read temperature data from a DS18B20 temperature sensor module using an ESP32 Development Board. We'll use the DallasTemperature library to interface with the sensor and display temperature readings in both Celsius and Fahrenheit units on the Serial Monitor.

Required Components

In this project, we need the following components.

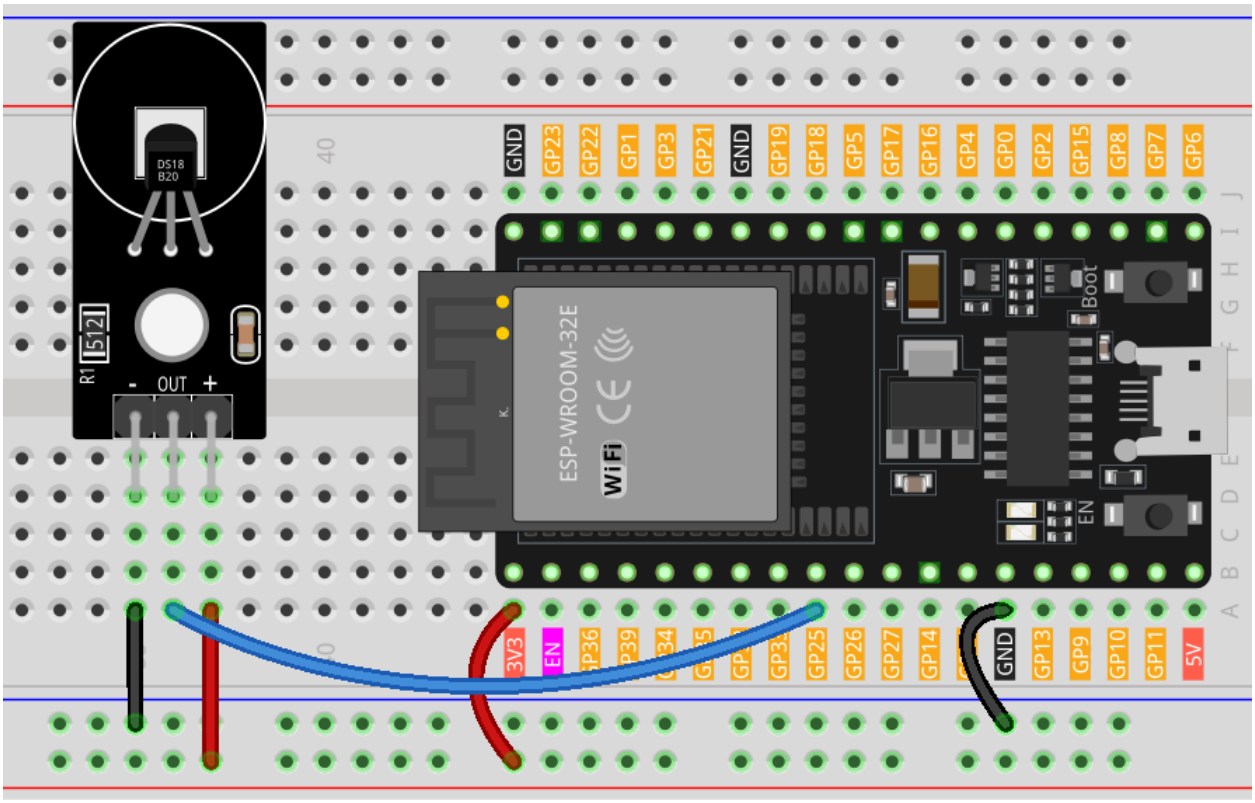
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Temperature Sensor Module (DS18B20)	-
Breadboard	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"DallasTemperature"** and install it.

Code Analysis

1. Library inclusion

   The inclusion of the OneWire and DallasTemperature libraries allows communication with the DS18B20 sensor.

   Note: To install the library, use the Arduino Library Manager and search for **"DallasTemperature"** and install it.

   ```
   #include <OneWire.h>
   #include <DallasTemperature.h>
   ```

2. Defining the sensor data pin

   The DS18B20 is connected to digital pin 25 of the Arduino.

   ```
   #define ONE_WIRE_BUS 25
   ```

3. Initializing the sensor

   The OneWire instance and DallasTemperature object are created and initialized.

```
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
```

4. Setup function

   The `setup()` function initializes the sensor and sets up serial communication.

```
void setup(void)
{
   sensors.begin();       // Start up the library
   Serial.begin(9600);
}
```

5. Main loop

   In the `loop()` function, the program requests temperature readings and prints them in both Celsius and Fahrenheit.

```
void loop(void)
{
   sensors.requestTemperatures();
   Serial.print("Temperature: ");
   Serial.print(sensors.getTempCByIndex(0));
   Serial.print("℃ | ");
   Serial.print((sensors.getTempCByIndex(0) * 9.0) / 5.0 + 32.0);
   Serial.println("");
   delay(500);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.20 Lesson 19: Temperature and Humidity Sensor Module (DHT11)

In this lesson, you will learn how to read temperature and humidity from a DHT11 sensor using an ESP32 Development Board. We'll also cover interpreting these readings and calculating the heat index in both Celsius and Fahrenheit. This project is ideal for beginners in environmental sensing, providing hands-on experience with sensor data acquisition and basic concepts of climate monitoring on the ESP32 platform.

Required Components

In this project, we need the following components.

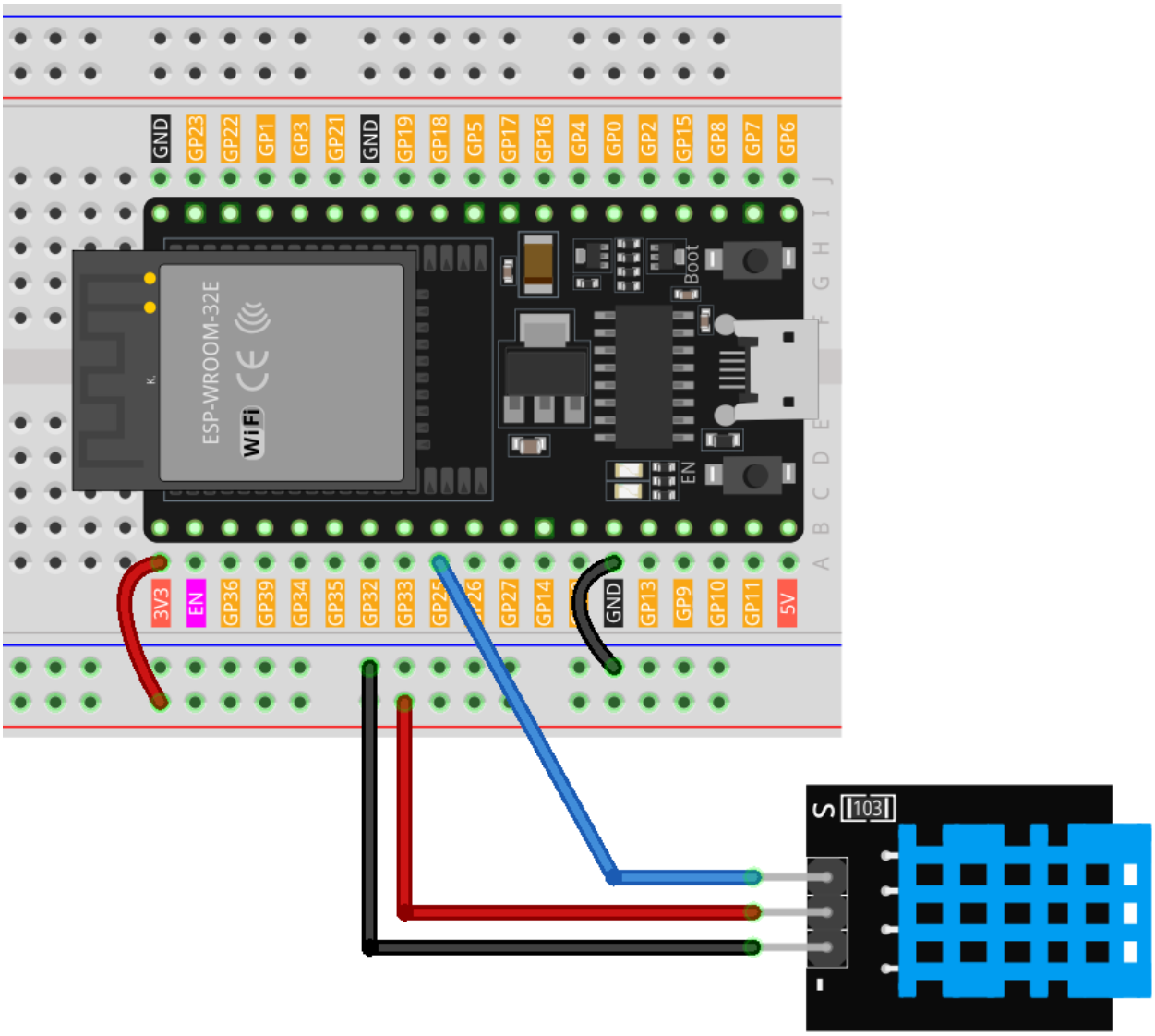
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Temperature and Humidity Sensor Module (DHT11)	
Breadboard	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"DHT sensor library"** and install it.

Code Analysis

1. Inclusion of necessary libraries and definition of constants. This part of the code includes the DHT sensor library and defines the pin number and sensor type used in this project.

Note: To install the library, use the Arduino Library Manager and search for **"DHT sensor library"** and install it.

```
#include <DHT.h>
#define DHTPIN 25       // Define the pin used to connect the sensor
#define DHTTYPE DHT11  // Define the sensor type
```

2. Creation of DHT object. Here we create a DHT object using the defined pin number and sensor type.

```
DHT dht(DHTPIN, DHTTYPE);  // Create a DHT object
```

3. This function is executed once when the ESP32 Development Board starts. We initialize the serial communication and the DHT sensor in this function.

```
void setup() {
  Serial.begin(9600);
  Serial.println(F("DHT11 test!"));
  dht.begin();  // Initialize the DHT sensor
}
```

4. Main loop. The `loop()` function runs continuously after the setup function. Here, we read the humidity and temperature values, calculate the heat index, and print these values to the serial monitor. If the sensor read fails (returns NaN), it prints an error message.

Note: The is a way to measure how hot it feels outside by combining the air temperature and the humidity. It is also called the "felt air temperature" or "apparent temperature".

```
void loop() {
  delay(2000);
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  float f = dht.readTemperature(true);
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }
  float hif = dht.computeHeatIndex(f, h);
  float hic = dht.computeHeatIndex(t, h, false);
  Serial.print(F("Humidity: "));
  Serial.print(h);
  Serial.print(F("%  Temperature: "));
  Serial.print(t);
  Serial.print(F("°C "));
  Serial.print(f);
  Serial.print(F("°F  Heat index: "));
  Serial.print(hic);
```

(continues on next page)

(continued from previous page)

```
  Serial.print(F("°C "));
  Serial.print(hif);
  Serial.println(F("°F"));
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.21 Lesson 20: Temperature, Humidity & Pressure Sensor (BMP280)

In this lesson, you will learn how to measure atmospheric pressure, temperature, and approximate altitude using the BMP280 sensor with an ESP32 Development Board. We will cover interfacing the sensor with the Adafruit BMP280 library and displaying readings on the Serial Monitor. This tutorial is ideal for those seeking to enhance their understanding of environmental sensing and data logging on the ESP32 platform.

Required Components

In this project, we need the following components.

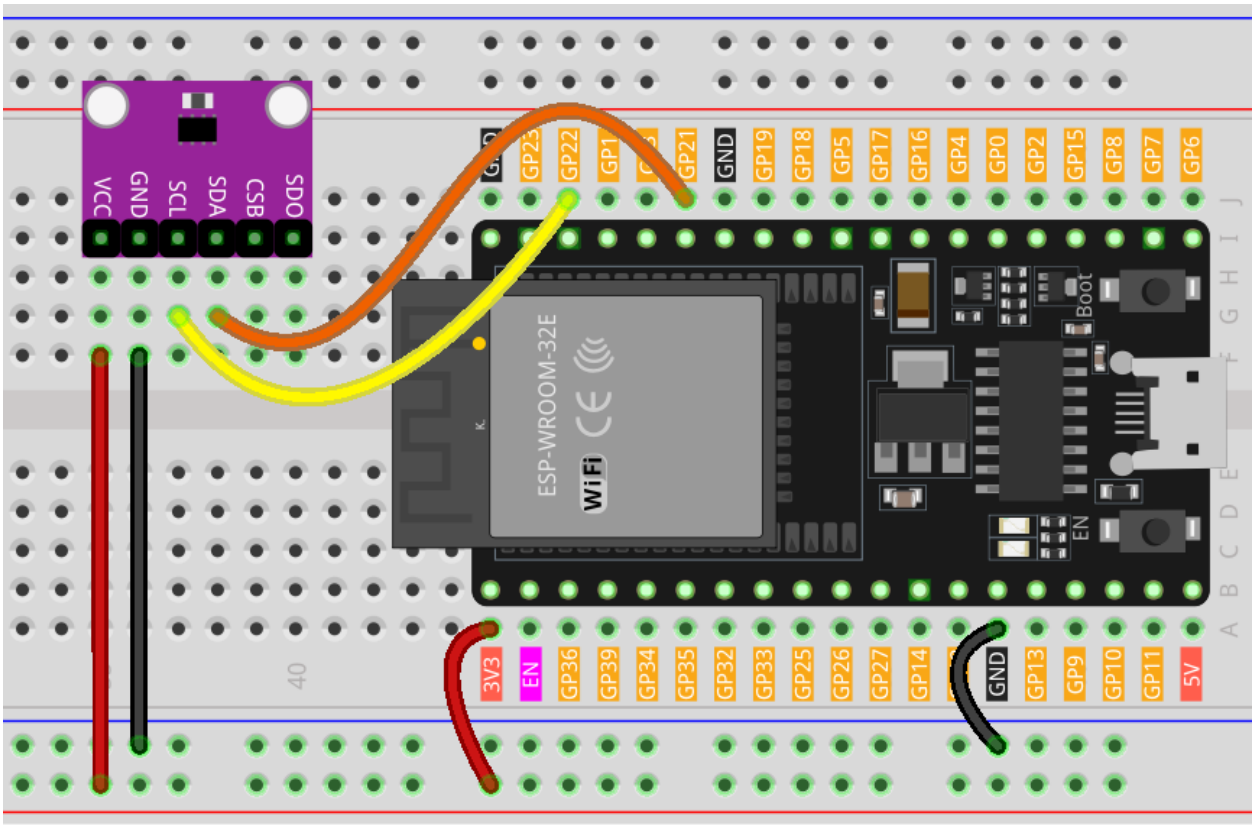
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Temperature, Humidity & Pressure Sensor (BMP280)	
Breadboard	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"Adafruit BMP280"** and install it.

Code Analysis

1. Including Libraries and Initialization. Necessary libraries are included and the BMP280 sensor is initialized for communication using the I2C interface.

   Note: To install the library, use the Arduino Library Manager and search for **"Adafruit BMP280"** and install it.

   - Adafruit BMP280 Library: This library provides an easy-to-use interface for the BMP280 sensor, allowing the user to read temperature, pressure, and altitude.
   - Wire.h: Used for I2C communication.

```
#include <Wire.h>
#include <Adafruit_BMP280.h>
#define BMP280_ADDRESS 0x76
Adafruit_BMP280 bmp;  // use I2C interface
```

2. The `setup()` function initializes the Serial communication, checks for the BMP280 sensor, and sets up the sensor with default settings.

```
void setup() {
  Serial.begin(9600);
  while (!Serial) delay(100);
  Serial.println(F("BMP280 test"));
  unsigned status;
  status = bmp.begin(BMP280_ADDRESS);
  // ... (rest of the setup code)
```

3. The `loop()` function reads data from the BMP280 sensor for temperature, pressure, and altitude. This data is printed to the Serial Monitor.

```
void loop() {
  // ... (read and print temperature, pressure, and altitude data)
  delay(2000);  // 2-second delay between readings.
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.22 Lesson 21: Time of Flight Micro-LIDAR Distance Sensor (VL53L0X)

In this lesson, you will learn how to use the Adafruit VL53L0X Time of Flight Distance Sensor with an ESP32 Development Board. We'll cover initializing the sensor, reading distance measurements, and displaying them in millimeters on the serial monitor.

Required Components

In this project, we need the following components.

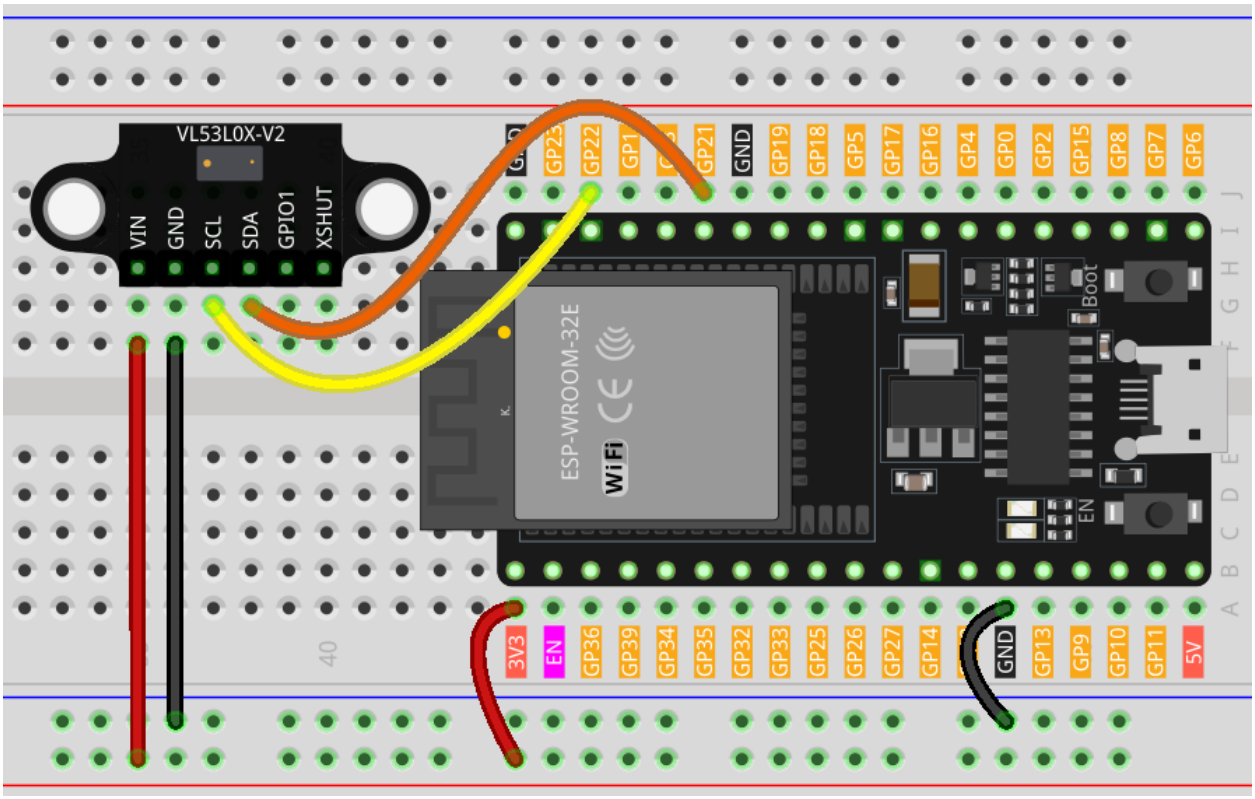
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Time of Flight Micro-LIDAR Distance Sensor (VL53L0X)	
Breadboard	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"Adafruit_VL53L0X"** and install it.

Code Analysis

1. Including the necessary library and initializing the sensor object. We start by including the library for the VL53L0X sensor and creating an instance of the Adafruit_VL53L0X class.

   Note: To install the library, use the Arduino Library Manager and search for **"Adafruit_VL53L0X"** and install it.

```
#include <Adafruit_VL53L0X.h>
Adafruit_VL53L0X lox = Adafruit_VL53L0X();
```

2. Initialization in the `setup()` function. Here, we set up serial communication and initialize the distance sensor. If the sensor can't be initialized, the program halts.

```
void setup() {
  Serial.begin(115200);
  while (!Serial) {
    delay(1);
  }
  Serial.println("Adafruit VL53L0X test");
  if (!lox.begin()) {
    Serial.println(F("Failed to boot VL53L0X"));
    while (1)
      ;
  }
  Serial.println(F("VL53L0X API Simple Ranging example\n\n"));
}
```

3. Capturing and displaying the measurements in the `loop()` function. Continuously, the ESP32 Development Board captures a distance measurement using the `rangingTest()` method. If the measurement is valid, it's printed to the serial monitor.

```
void loop() {
  VL53L0X_RangingMeasurementData_t measure;
  Serial.print("Reading a measurement... ");
  lox.rangingTest(&measure, false);
  if (measure.RangeStatus != 4) {
    Serial.print("Distance (mm): ");
    Serial.println(measure.RangeMilliMeter);
  } else {
    Serial.println(" out of range ");
  }
  delay(100);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.23 Lesson 22: Touch Sensor Module

In this lesson, you'll learn how to use a touch sensor with an ESP32 Development Board. We'll see how touching the sensor sends a signal to the ESP32, triggering a response displayed through serial communication. This project is ideal for beginners and provides hands-on experience with digital inputs and serial output on the ESP32 platform. You'll develop a foundational understanding of how sensors interact with microcontrollers, which is essential for building interactive hardware projects.

Required Components

In this project, we need the following components.

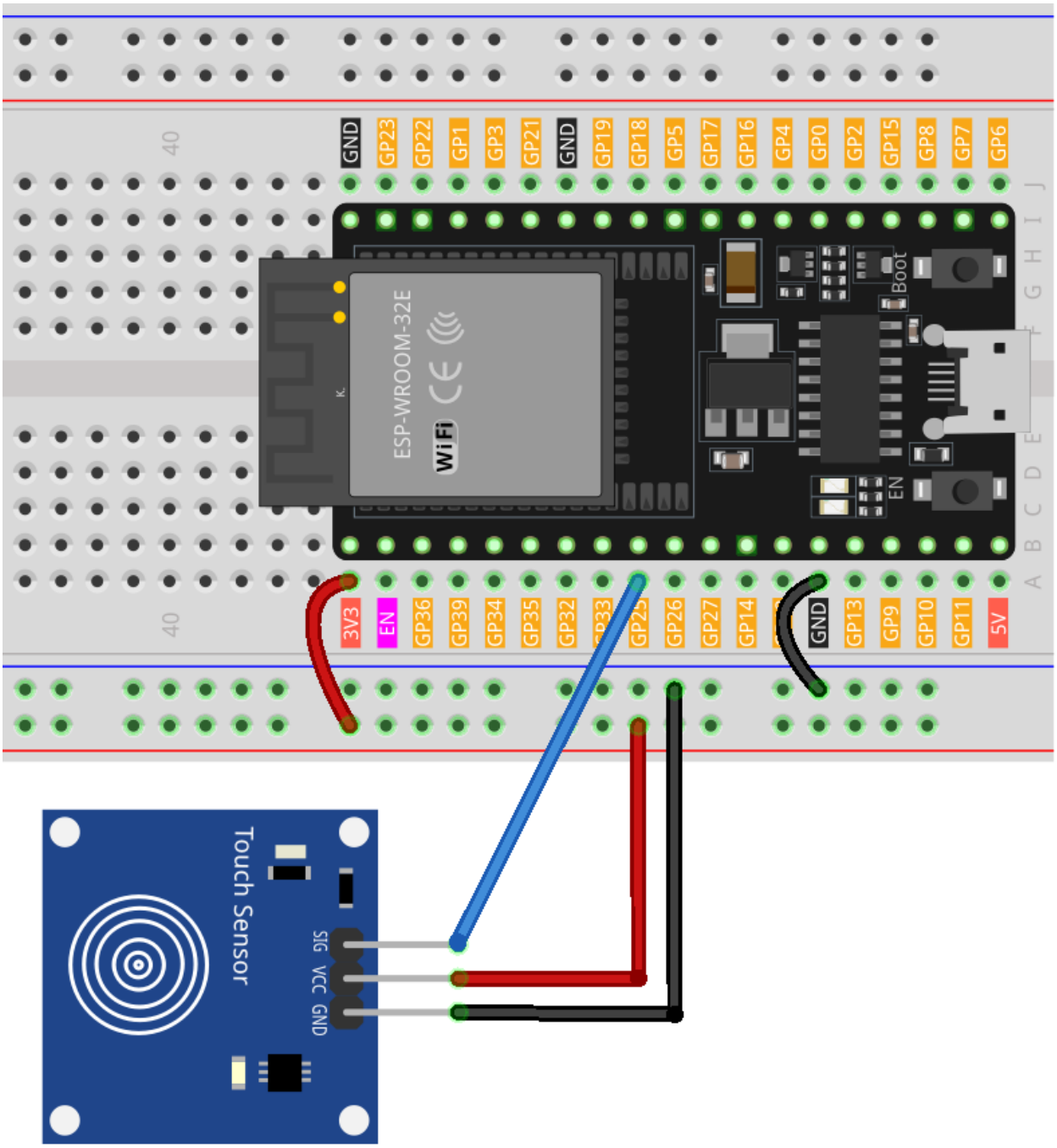
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Touch Sensor Module	
Breadboard	

Wiring


GND
GP23
GP22
GP1
GP3
GP21
GND
GP19
GP18
GP5
GP17
GP16
GP4
GP0
GP2
GP15
GP8
GP7
GP6
ESP-WROOM-32E
WiFi
Boot
EN
3V3
EN
GP36
GP39
GP34
GP35
GP32
GP33
GP25
GP26
GP27
GP14
GND
GP13
GP9
GP10
GP11
5V
Touch Sensor
SIG
VCC
GND

Code

Code Analysis

1. **Setting Up the Pin and Serial Communication**
    - The touch sensor is connected to pin 25 of the ESP32, and this pin is configured as an input.
    - The `Serial.begin(9600);` initializes serial communication at a baud rate of 9600 bits per second.

```
const int sensorPin = 25;

void setup() {
  pinMode(sensorPin, INPUT);     // Set the sensor pin as input
  Serial.begin(9600);            // Start the serial communication
}
```

2. **Reading the Sensor and Sending Data to Serial Monitor**
    - The `loop()` function continuously checks the state of the touch sensor.
    - `digitalRead(sensorPin)` reads the digital value (1 or 0) from the sensor pin.
    - If the sensor is touched (value 1), it prints "Touch detected!" to the Serial Monitor.
    - If not touched (value 0), it prints "No touch detected...".
    - The `delay(100);` helps in debouncing the sensor, preventing multiple rapid readings.

```
void loop() {
  if (digitalRead(sensorPin) == 1) {  // If the sensor is touched
    Serial.println("Touch detected!");
  } else {
    Serial.println("No touch detected...");
  }
  delay(100);  // Wait for a short period to avoid rapid reading of the sensor
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.24 Lesson 23: Ultrasonic Sensor Module (HC-SR04)

In this lesson, you'll learn how to use an ESP32 Development Board for measuring distance with an ultrasonic sensor (HC-SR04). We'll cover setting up the sensor, reading data, and calculating the distance in centimeters. This project is ideal for beginners working with microcontrollers and sensors, offering hands-on experience in data acquisition and serial communication. You'll develop practical skills in programming the ESP32 and grasp ultrasonic sensing technology.

Required Components

In this project, we need the following components.

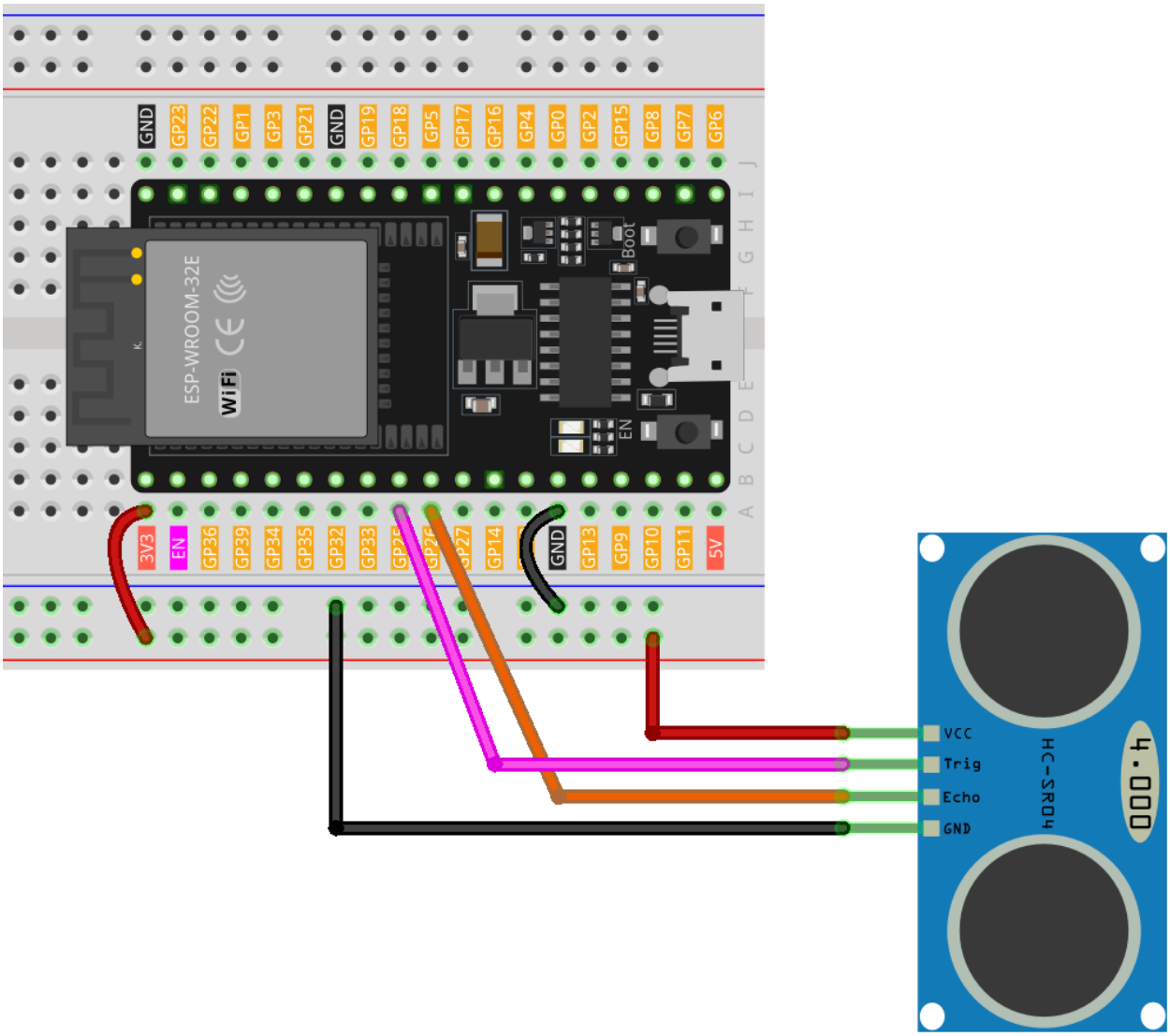
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Ultrasonic Sensor Module (HC-SR04)	
Breadboard	

Wiring



Code

Code Analysis

1. Pin declaration:

   Start by defining the pins for the ultrasonic sensor. `echoPin` and `trigPin` are declared as integers and their values are set to match the physical connection on the ESP32 Development Board.

   ```
   const int echoPin = 26;
   const int trigPin = 25;
   ```

2. `setup()` function:

   The `setup()` function initializes the serial communication, sets the pin modes, and prints a message to indicate the ultrasonic sensor is ready.

```
void setup() {
  Serial.begin(9600);
  pinMode(echoPin, INPUT);
  pinMode(trigPin, OUTPUT);
  Serial.println("Ultrasonic sensor:");
}
```

3. `loop()` function:

   The `loop()` function reads the distance from the sensor and prints it to the serial monitor, then delays for 400 milliseconds before repeating.

```
void loop() {
  float distance = readDistance();
  Serial.print(distance);
  Serial.println(" cm");
  delay(400);
}
```

4. `readDistance()` function :

   The `readDistance()` function triggers the ultrasonic sensor and calculates the distance based on the time it takes for the signal to bounce back.

   For more details, please refer to the working *principle* of the ultrasonic sensor module.

```
float readDistance() {
  digitalWrite(trigPin, LOW);   // Set trig pin to low to ensure a clean pulse
  delayMicroseconds(2);         // Delay for 2 microseconds
  digitalWrite(trigPin, HIGH);  // Send a 10 microsecond pulse by setting trig pin␣
↪to high
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);  // Set trig pin back to low
  float distance = pulseIn(echoPin, HIGH) / 58.00;  // Formula: (340m/s * 1us) / 2
  return distance;
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.25 Lesson 24: Vibration Sensor Module (SW-420)

In this lesson, you'll learn how to detect vibrations using an ESP32 Development Board and a Vibration Sensor (SW-420). We'll cover reading digital output from the sensor and using conditional statements to display messages on the serial monitor. When the sensor detects vibration, it will display "Detected vibration…"; otherwise, it will output "…". This project provides a practical way to grasp digital inputs and serial communication, making it ideal for electronics and programming beginners.

Required Components

In this project, we need the following components.

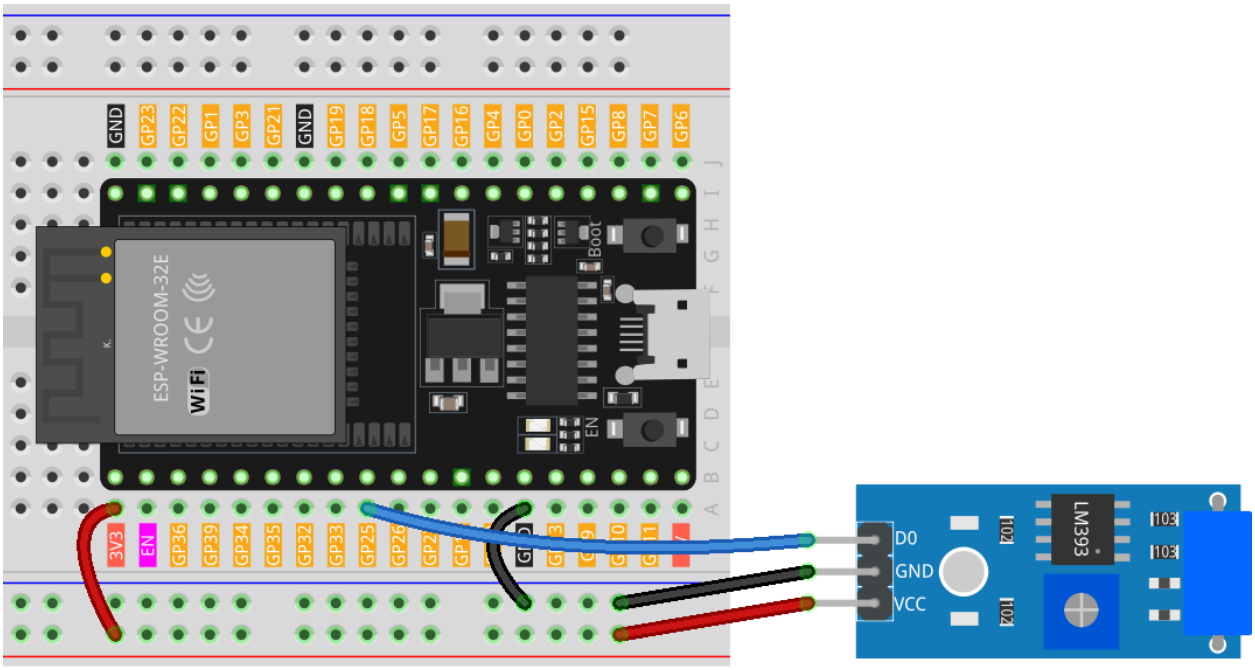
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Vibration Sensor Module (SW-420)	
Breadboard	

Wiring



Code

Code Analysis

1. The first line of code is a constant integer declaration for the vibration sensor pin. We use digital pin 25 to read the output from the vibration sensor.

```
const int sensorPin = 25;
```

2. In the `setup()` function, we initialize the serial communication at a baud rate of 9600 to print readings from the vibration sensor to the serial monitor. We also set the vibration sensor pin as an input.

```
void setup() {
  Serial.begin(9600);         // Start serial communication at 9600 baud rate
  pinMode(sensorPin, INPUT);  // Set the sensorPin as an input pin
}
```

3. The `loop()` function is where we continuously check for any vibrations detected by the sensor. If the sensor detects a vibration, it prints "Detected vibration…" to the serial monitor. If no vibration is detected, it prints "…". The loop repeats every 100 milliseconds.

```
void loop() {
  if (digitalRead(sensorPin)) {               // Check if there is any vibration␣
↪detected by the sensor
    Serial.println("Detected vibration...");  // Print "Detected vibration..." if␣
↪vibration detected
  }
  else {
    Serial.println("...");  // Print "..." otherwise
```

(continues on next page)

(continued from previous page)

```
  }
  // Add a delay to avoid flooding the serial monitor
  delay(100);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.26 Lesson 25: Water Level Sensor Module

In this lesson, you'll learn how to use an ESP32 Development Board for reading a water level sensor. We'll cover continuously monitoring the sensor's analog value and displaying it on the serial monitor. This project provides a great opportunity to grasp sensor integration and analog data reading with Arduino, making it ideal for beginners in electronics and microcontroller programming.

Required Components

In this project, we need the following components.

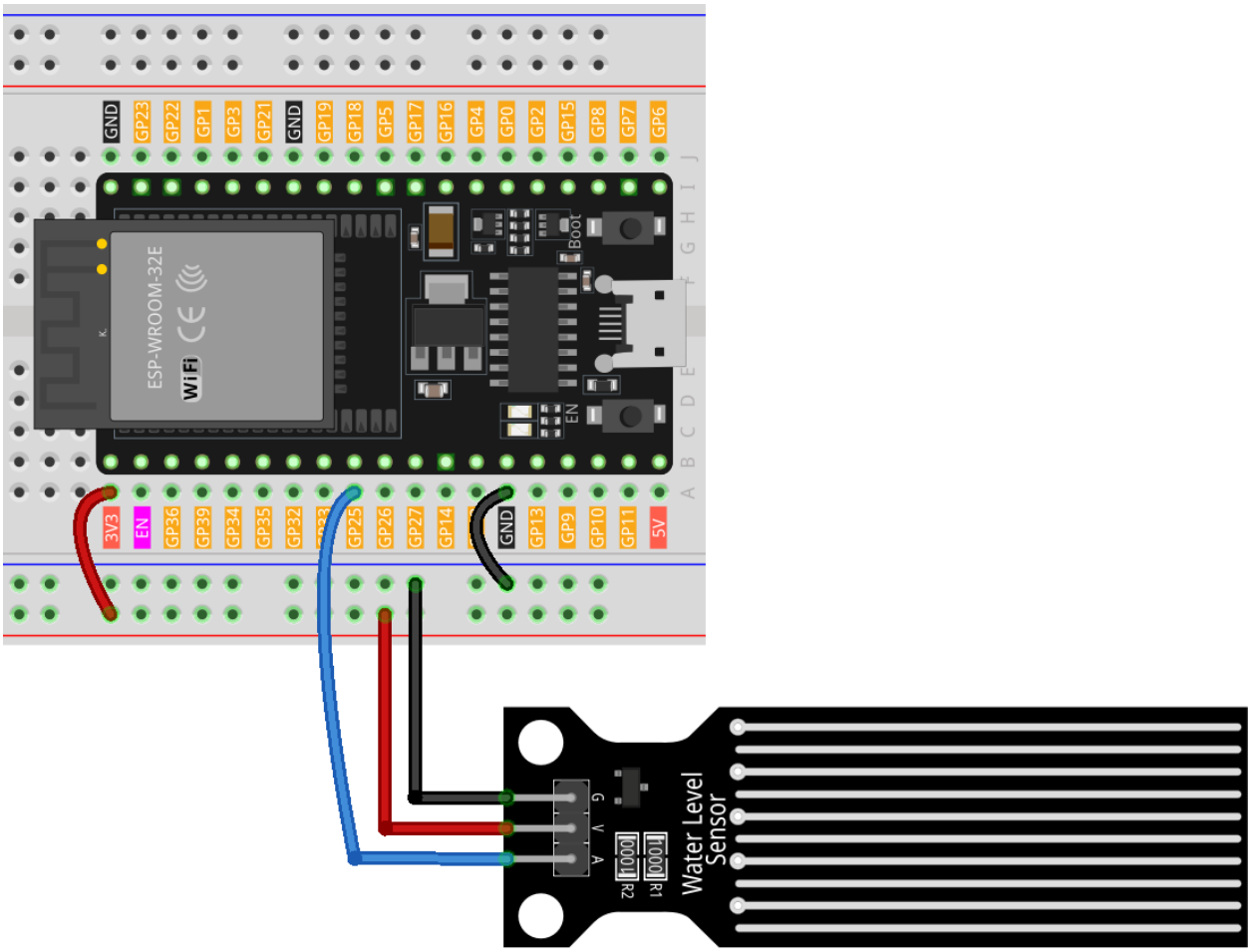
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Water Level Sensor Module	-
Breadboard	

Wiring



Code

Code Analysis

1. **Initializing the Sensor Pin**:

   Before using the water level sensor, it's pin number is defined using a constant variable. This makes the code more readable and easier to modify.

   ```
   const int sensorPin = 25;
   ```

2. **Setting Up Serial Communication**:

   In the `setup()` function, the baud rate for serial communication is set. This is crucial for the Arduino to communicate with the computer's serial monitor.

   ```
   void setup() {
     Serial.begin(9600);  // Start serial communication at 9600 baud rate
   }
   ```

3. **Reading Sensor Data and Outputting to Serial Monitor**:

The `loop()` function continuously reads the sensor's analog value using `analogRead()` and outputs it to the serial monitor using `Serial.println()`. The `delay(100)` function makes the Arduino wait for 100 milliseconds before repeating the loop, controlling the rate of data reading and transmission.

```
void loop() {
  Serial.println(analogRead(sensorPin));  // Read the analog value of the sensor␣
↪and print it to the serial monitor
  delay(100);                             // Wait for 100 milliseconds
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.27 Lesson 26: I2C LCD 1602

In this lesson, you'll learn how to set up and display messages on a 16x2 Liquid Crystal Display (LCD) with an I2C interface using an ESP32 Development Board. We'll cover initializing the LCD using the LiquidCrystal I2C library, then displaying "Hello world!" and "LCD Tutorial" on two separate lines of the screen. This tutorial is ideal for beginners, offering hands-on experience with LCD interfaces and improving your understanding of output operations in Arduino programming.

Required Components

In this project, we need the following components.

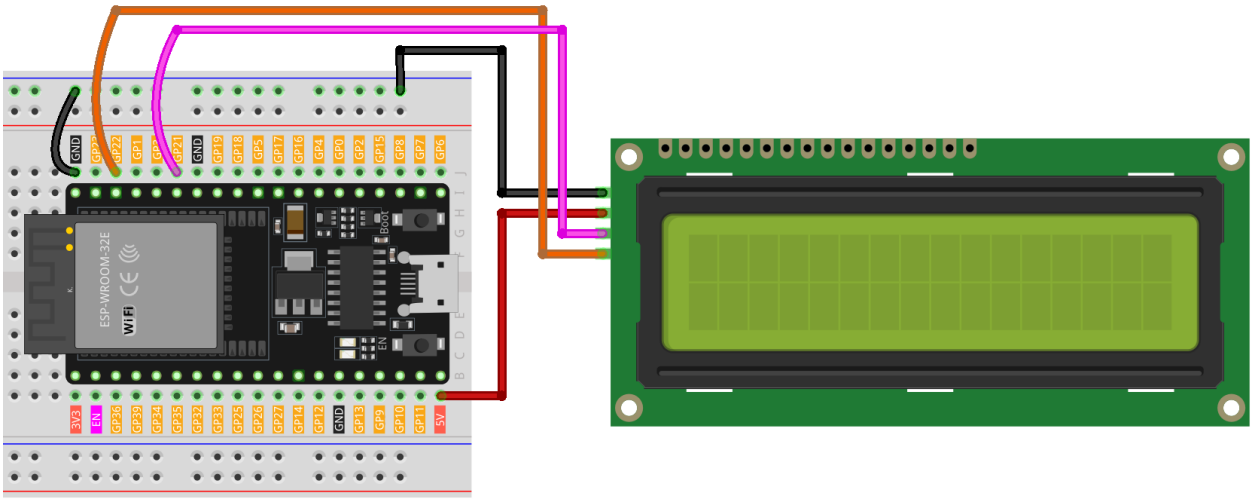
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
I2C LCD 1602	
Breadboard	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"LiquidCrystal I2C"** and install it.

Code Analysis

1. Library Inclusion and LCD Initialization: The LiquidCrystal I2C library is included to provide functions and methods for LCD interfacing. Following that, an LCD object is created using the LiquidCrystal_I2C class, specifying the I2C address, number of columns, and number of rows.

   Note: To install the library, use the Arduino Library Manager and search for **"LiquidCrystal I2C"** and install it.

   ```
   #include <LiquidCrystal_I2C.h>
   LiquidCrystal_I2C lcd(0x27, 16, 2);
   ```

2. Setup Function: The `setup()` function is executed once when the ESP32 Development Board starts. In this function, the LCD is initialized, cleared, and the backlight is turned on. Then, two messages are displayed on the LCD.

```
void setup() {
  lcd.init();       // initialize the LCD
  lcd.clear();      // clear the LCD display
  lcd.backlight();  // Make sure backlight is on

  // Print a message on both lines of the LCD.
  lcd.setCursor(2, 0);  //Set cursor to character 2 on line 0
  lcd.print("Hello world!");

  lcd.setCursor(2, 1);  //Move cursor to character 2 on line 1
  lcd.print("LCD Tutorial");
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.28 Lesson 27: OLED Display Module (SSD1306)

In this lesson, you will learn how to set up and utilize an OLED display with an ESP32 Development Board using the Adafruit SSD1306 and GFX libraries. We will cover displaying text in different sizes, inverting text colors, creating scrolling text animations, and rendering custom bitmap graphics. This project provides a thorough introduction to advanced display techniques, ideal for individuals seeking to improve their skills in developing interactive electronics with microcontrollers.

Required Components

In this project, we need the following components.

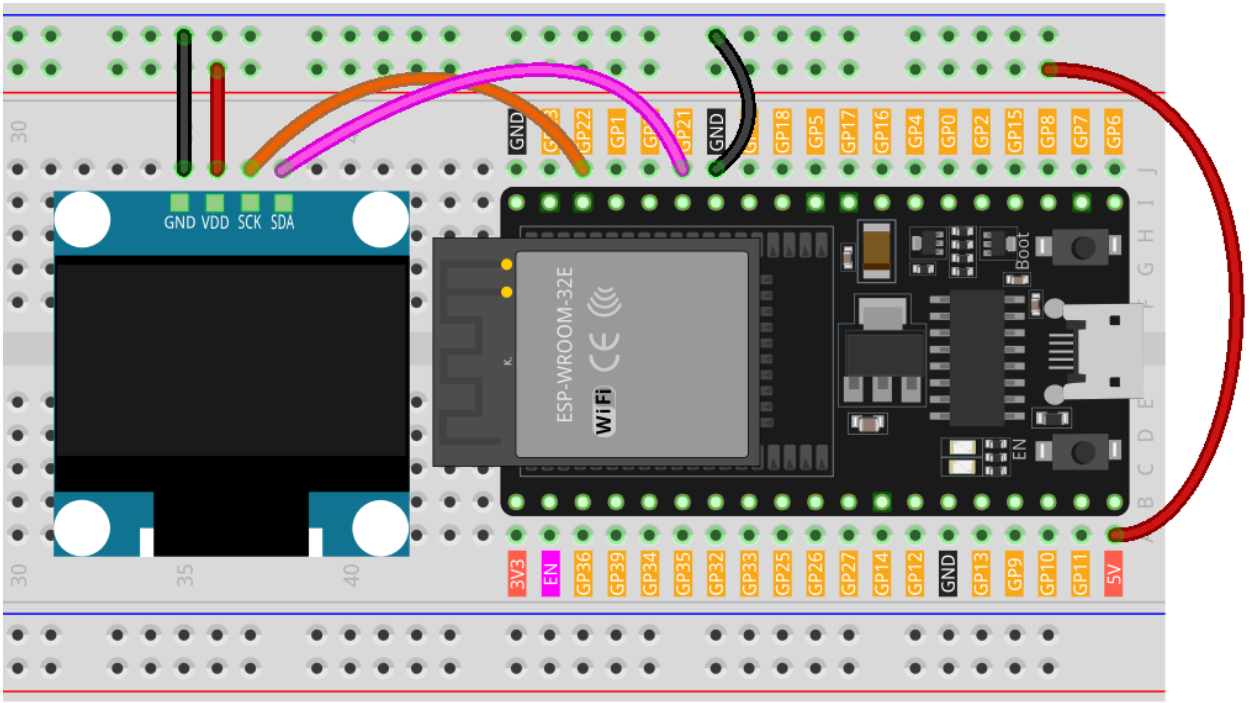
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
OLED Display Module (SSD1306)	-
Breadboard	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"Adafruit SSD1306"** and **"Adafruit GFX"** and install it.

Code Analysis

1. **Library Inclusion and Initial Definitions**: The necessary libraries for interfacing with the OLED are included. Following that, definitions regarding the OLED's dimensions and I2C address are provided.
    - **Adafruit SSD1306**: This library is designed to help with the interfacing of the SSD1306 OLED display. It provides methods to initialize the display, control its settings, and display content.
    - **Adafruit GFX Library**: This is a core graphics library for displaying text, producing colors, drawing shapes, etc., on various screens including OLEDs.

Note: To install the library, use the Arduino Library Manager and search for **"Adafruit SSD1306"** and **"Adafruit GFX"** and install it.

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128  // OLED display width, in pixels
#define SCREEN_HEIGHT 64  // OLED display height, in pixels

#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C
```

2. **Bitmap Data**: Bitmap data for displaying a custom icon on the OLED screen. This data represents an image in a format that the OLED can interpret.

   You can use this online tool called that can turn your image into an array.

   The `PROGMEM` keyword denotes that the array is stored in the program memory of the Arduino microcontroller. Storing data in program memory(PROGMEM) instead of RAM can be helpful for large amounts of data, which would otherwise take up too much space in RAM.

   ```
   static const unsigned char PROGMEM sunfounderIcon[] = {...};
   ```

3. **Setup Function (Initialization and Display)**: The `setup()` function initializes the OLED and displays a series of patterns, texts, and animations.

   ```
   void setup() {
      ...  // Serial initialization and OLED object initialization
      ...  // Displaying various text, numbers, and animations
   }
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.29 Lesson 28: RGB LED Module

In this lesson, you will learn how to control an RGB LED using an ESP32 Development Board. We'll cover using different color channels to display primary colors and creating a sequence of rainbow colors. This project is ideal for beginners in electronics and programming, providing hands-on experience with output operations and color mixing using the ESP32 and RGB LED module.

Required Components

In this project, we need the following components.

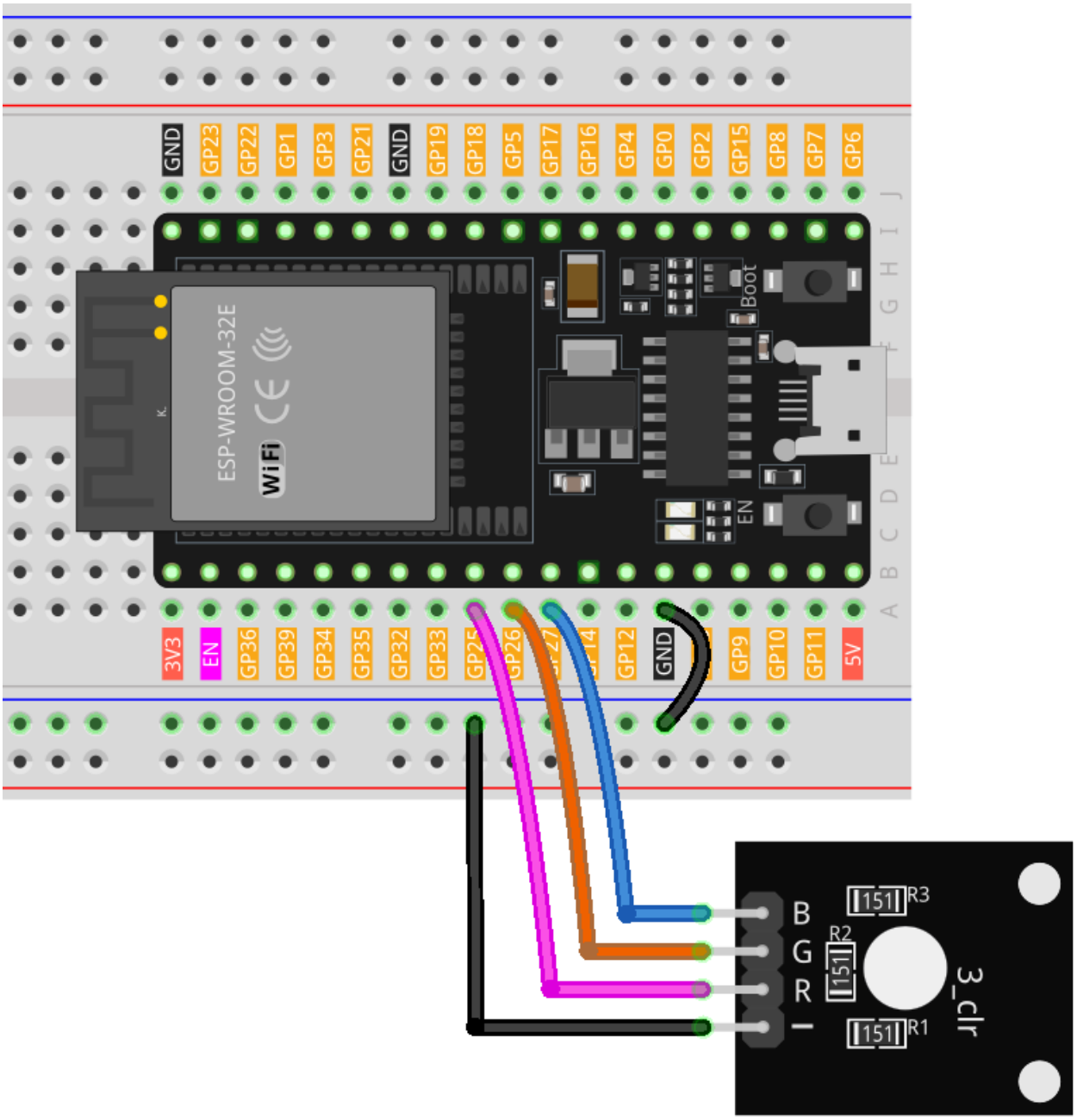
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
RGB LED Module	-
Breadboard	

Wiring


GND
GP23
GP22
GP1
GP3
GP21
GND
GP19
GP18
GP5
GP17
GP16
GP4
GP0
GP2
GP15
GP8
GP7
GP6
ESP-WROOM-32E
WiFi
Boot
EN
3V3
EN
GP36
GP39
GP34
GP35
GP32
GP33
GP25
GP26
GP12
GND
GP9
GP10
GP11
5V
B
G
R
I
151
R3
R2
R1
3_clr

Code

Code Analysis

1. The first segment of the code declares and initializes the pins to which each color channel of the RGB LED module is connected.

```
const int rledPin = 25;  // pin connected to the red color channel
const int gledPin = 26;   // pin connected to the green color channel
const int bledPin = 27;  // pin connected to the blue color channel
```

2. The `setup()` function initializes these pins as OUTPUT. This means we are sending signals OUT from these pins to the RGB LED module.

```
void setup() {
  pinMode(rledPin, OUTPUT);
  pinMode(gledPin, OUTPUT);
  pinMode(bledPin, OUTPUT);
}
```

3. In the `loop()` function, the `setColor()` function is called with different parameters to display different colors. The `delay()` function is used after setting each color to pause for 1000 milliseconds (or 1 second) before moving on to the next color.

```
void loop() {
  setColor(255, 0, 0);  // Set RGB LED color to red
  delay(1000);
  setColor(0, 255, 0);  // Set RGB LED color to green
  delay(1000);
  // The rest of the color sequence...
}
```

4. The `setColor()` function uses the `analogWrite()` function to adjust the brightness of each color channel on the RGB LED module. The `analogWrite()` function employs Pulse Width Modulation (PWM) to simulate varying voltage outputs. By controlling the PWM duty cycle (the percentage of time a signal is HIGH within a fixed period), the brightness of each color channel can be controlled, allowing the mixing of various colors.

```
void setColor(int R, int G, int B) {
  analogWrite(rledPin, R);  // Use PWM to control the brightness of the red color␣
↪channel
  analogWrite(gledPin, G);  // Use PWM to control the brightness of the green color␣
↪channel
  analogWrite(bledPin, B);  // Use PWM to control the brightness of the blue color␣
↪channel
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.

- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.30 Lesson 29: Traffic Light Module

In this lesson, you'll learn how to use an ESP32 Development Board to control a Mini Traffic Light Module. We'll cover setting up the board and writing code to create a traffic light sequence: 5 seconds of green light, blinking yellow light for 1.5 seconds, and 5 seconds of red light. This project is ideal for beginners in electronics and programming as it provides practical experience with output operations and basic timing control using the ESP32.

Required Components

In this project, we need the following components.

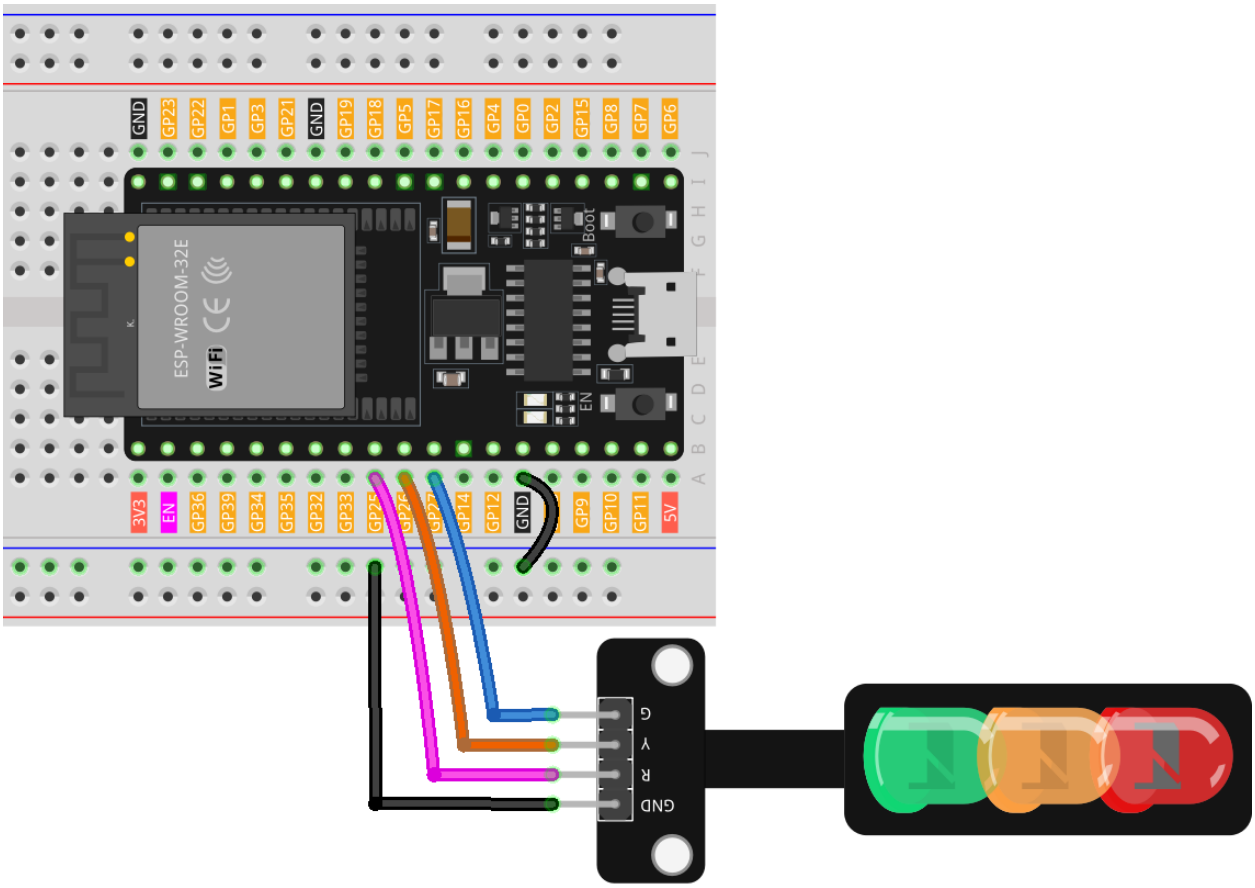
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Traffic Light Module	
Breadboard	

Wiring



Code

Code Analysis

1. Before any operations, we define constants for the pins where LEDs are connected. This makes our code easier to read and modify.

```
const int rledPin = 25;  //red
const int yledPin = 26;  //yellow
const int gledPin = 27;  //green
```

2. Here, we specify the pin modes for our LED pins. They are all set to `OUTPUT` because we intend to send voltage to them.

```
void setup() {
  pinMode(rledPin, OUTPUT);
  pinMode(yledPin, OUTPUT);
  pinMode(gledPin, OUTPUT);
}
```

3. This is where our traffic light cycle logic is implemented. The sequence of operations is:

    - Turn the green LED on for 5 seconds.

- Blink the yellow LED three times (each blink lasts for 0.5 seconds).
- Turn the red LED on for 5 seconds.

```
void loop() {
  digitalWrite(gledPin, HIGH);
  delay(5000);
  digitalWrite(gledPin, LOW);

  digitalWrite(yledPin, HIGH);
  delay(500);
  digitalWrite(yledPin, LOW);
  delay(500);
  digitalWrite(yledPin, HIGH);
  delay(500);
  digitalWrite(yledPin, LOW);
  delay(500);
  digitalWrite(yledPin, HIGH);
  delay(500);
  digitalWrite(yledPin, LOW);
  delay(500);

  digitalWrite(rledPin, HIGH);
  delay(5000);
  digitalWrite(rledPin, LOW);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.31 Lesson 30: Relay Module

In this lesson, you'll learn how to use an ESP32 Development Board to control a one-channel relay module. We'll cover turning the relay on and off in a loop, with a 3-second delay between each state change. This project provides hands-on experience with digital output operations in embedded systems, making it ideal for beginners entering the realm of ESP32 and relay modules.

Required Components

In this project, we need the following components.

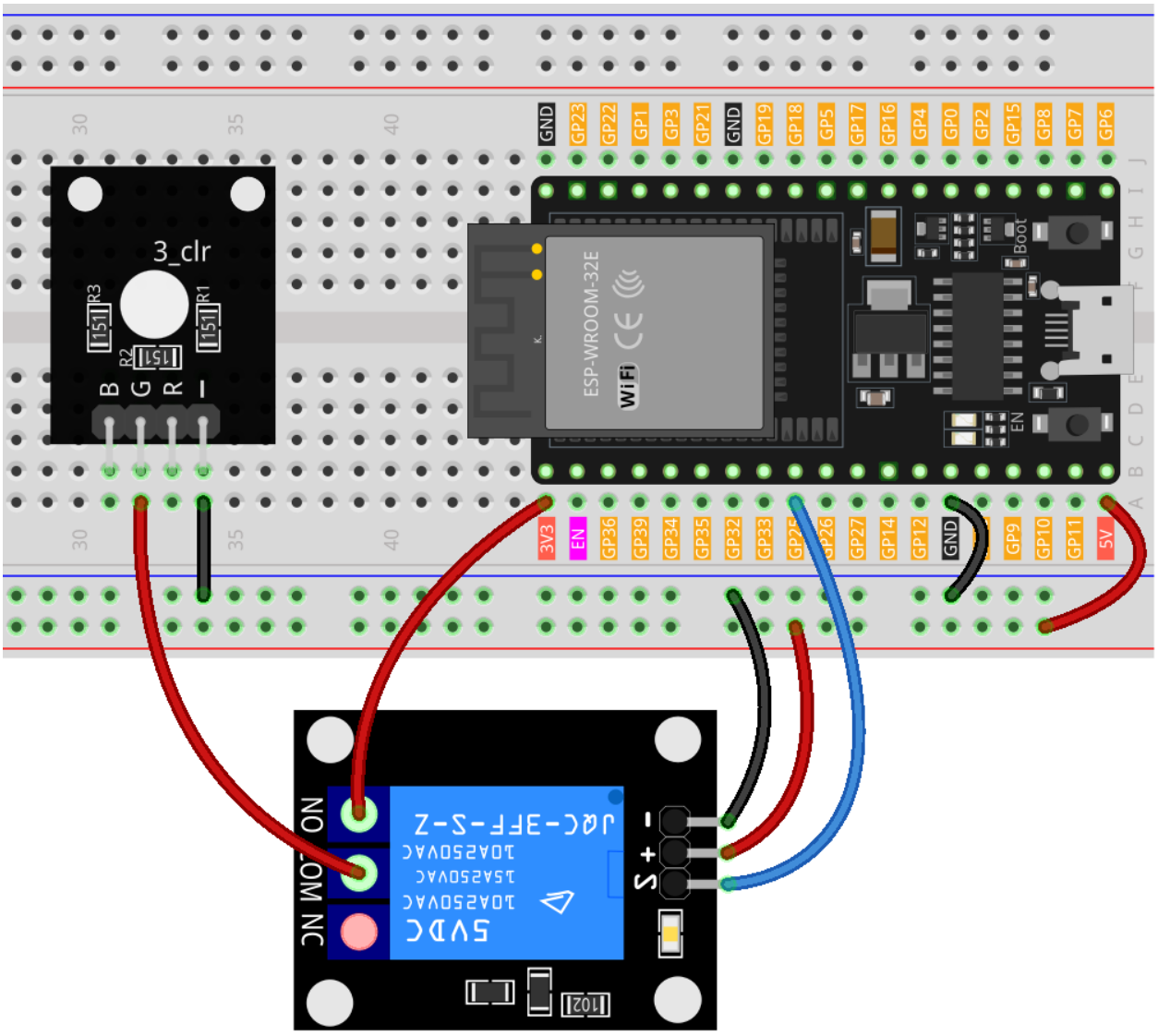
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Breadboard	
5V Relay Module	-
RGB LED Module	-

Wiring



Code

Code Analysis

1. Setting up the relay pin:
   - The relay module is connected to pin 25 of the ESP32 Development Board. This pin is defined as `relayPin` for ease of reference in the code.

```
const int relayPin = 25;
```

2. Configuring the relay pin as an output:
   - In the `setup()` function, the relay pin is set as an OUTPUT using the `pinMode()` function. This means the Arduino will send signals (either HIGH or LOW) to this pin.

```
void setup() {
  pinMode(relayPin, OUTPUT);
}
```

3. Toggling the relay ON and OFF:

   - In the `loop()` function, the relay is first set to the OFF state using `digitalWrite(relayPin, LOW)`. It remains in this state for 3 seconds (`delay(3000)`).
   - Then, the relay is set to the ON state using `digitalWrite(relayPin, HIGH)`. Again, it remains in this state for 3 seconds.
   - This cycle repeats indefinitely.

```
void loop() {
  digitalWrite(relayPin, LOW);
  delay(3000);

  digitalWrite(relayPin, HIGH);
  delay(3000);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.32 Lesson 31: Centrifugal Pump

In this lesson, you'll learn how to control a centrifugal pump with an ESP32 Development Board and an L9110 motor control board. We'll cover setting up and using two pins to operate the motor, causing the pump to spin in one direction for 5 seconds before shutting off. This project provides hands-on experience in managing motor operations and understanding digital signals in microcontroller programming, making it ideal for beginners in electronics and programming.

Required Components

In this project, we need the following components.

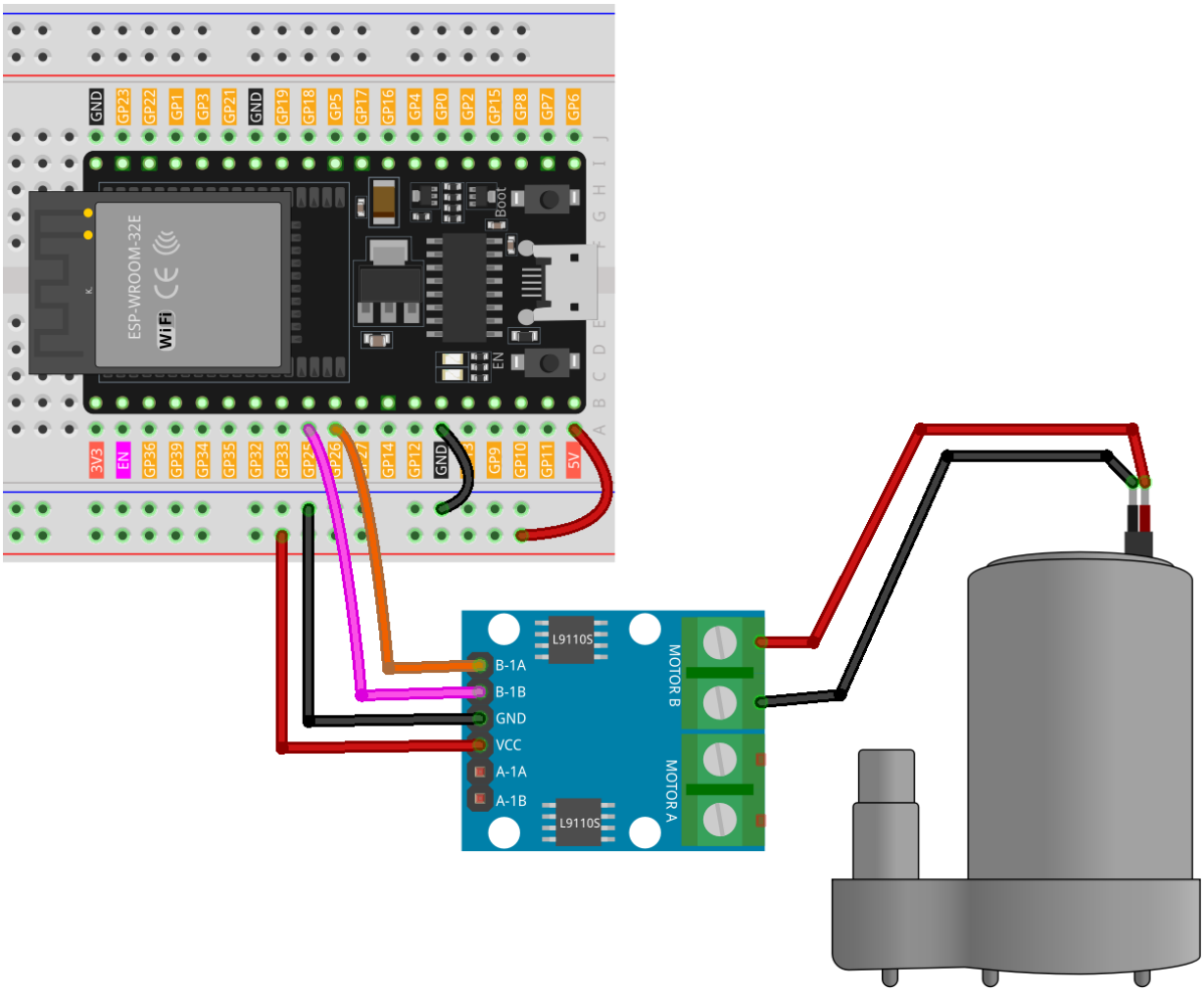
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Centrifugal Pump	-
L9110 Motor Driver Module	-
Breadboard	

Wiring



Code

Code Analysis

1. Two pins are defined for controlling the motor, specifically `motorB_1A` and `motorB_2A`. These pins will connect to the L9110 motor control board to control the direction and speed of the motor.

```
const int motorB_1A = 26;
const int motorB_2A = 25;
```

2. Configuring the pins and controlling the motor:

    - The `setup()` function initializes the pins as `OUTPUT` which means they can send signals to the motor control board.
    - The `analogWrite()` function is used to set the motor speed. Here, setting one pin to `HIGH` and the other to `LOW` makes the pump spin in one direction. After a 5-second delay, both pins are set to 0, turning off the motor.

```
void setup() {
   pinMode(motorB_1A, OUTPUT);  // set pump pin 1 as output
   pinMode(motorB_2A, OUTPUT);  // set pump pin 2 as output
   analogWrite(motorB_1A, HIGH);
   analogWrite(motorB_2A, LOW);
   delay(5000);// wait for 5 seconds
   analogWrite(motorB_1A, 0);  // turn off the pump
   analogWrite(motorB_2A, 0);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.33 Lesson 32: Passive Buzzer Module

In this lesson, you'll learn to play a melody on a passive buzzer module using an ESP32 Development Board. We'll cover programming the ESP32 to control the buzzer and create musical notes with varying durations. This project is ideal for beginners in electronics and programming, providing hands-on experience in sound generation and basic digital sound principles. You'll develop practical skills in utilizing the ESP32 board and integrating simple components like the passive buzzer.

Required Components

In this project, we need the following components.

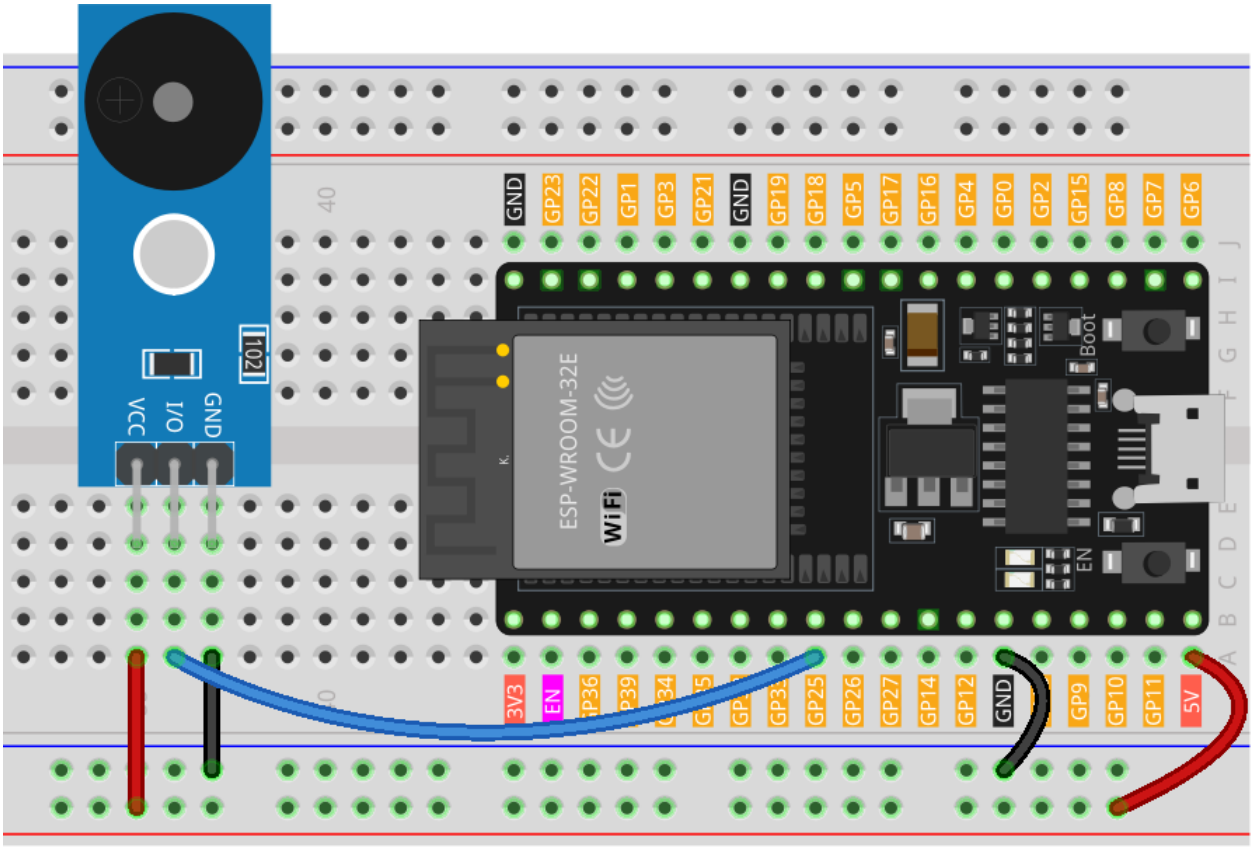
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Passive Buzzer Module	
Breadboard	

Wiring



Code

Code Analysis

1. Including the pitches library:

   This library provides the frequency values for various musical notes, allowing you to use musical notation in your code.

   ```
   #include "pitches.h"
   ```

2. Defining constants and arrays:

   - `buzzerPin` is the digital pin on the ESP32 Development Board where the buzzer is connected.

- `melody[]` is an array that stores the sequence of notes to be played.
- `noteDurations[]` is an array that stores the duration of each note in the melody.

```
const int buzzerPin = 25;
int melody[] = {
  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
};
int noteDurations[] = {
  4, 8, 8, 4, 4, 4, 4, 4
};
```

3. Playing the melody:
   - The `for` loop iterates over each note in the melody.
   - The `tone()` function plays a note on the buzzer for a specific duration.
   - A delay is added between notes to distinguish them.
   - The `noTone()` function stops the sound.

```
void setup() {
  for (int thisNote = 0; thisNote < 8; thisNote++) {
    int noteDuration = 1000 / noteDurations[thisNote];
    tone(buzzerPin, melody[thisNote], noteDuration);
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    noTone(buzzerPin);
  }
}
```

4. Empty loop function:

   Since the melody is played only once in the setup, there's no code in the loop function.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.34 Lesson 33: Servo Motor (SG90)

In this lesson, you'll learn how to control a servo motor with an ESP32 Development Board. We'll cover the process of making the servo motor scan from 0 to 180 degrees and back, giving you hands-on experience in managing servo movements. This project is ideal for those seeking to grasp motor control and the use of pulse width modulation (PWM) in robotics, utilizing the versatile ESP32 board.

Required Components

In this project, we need the following components.

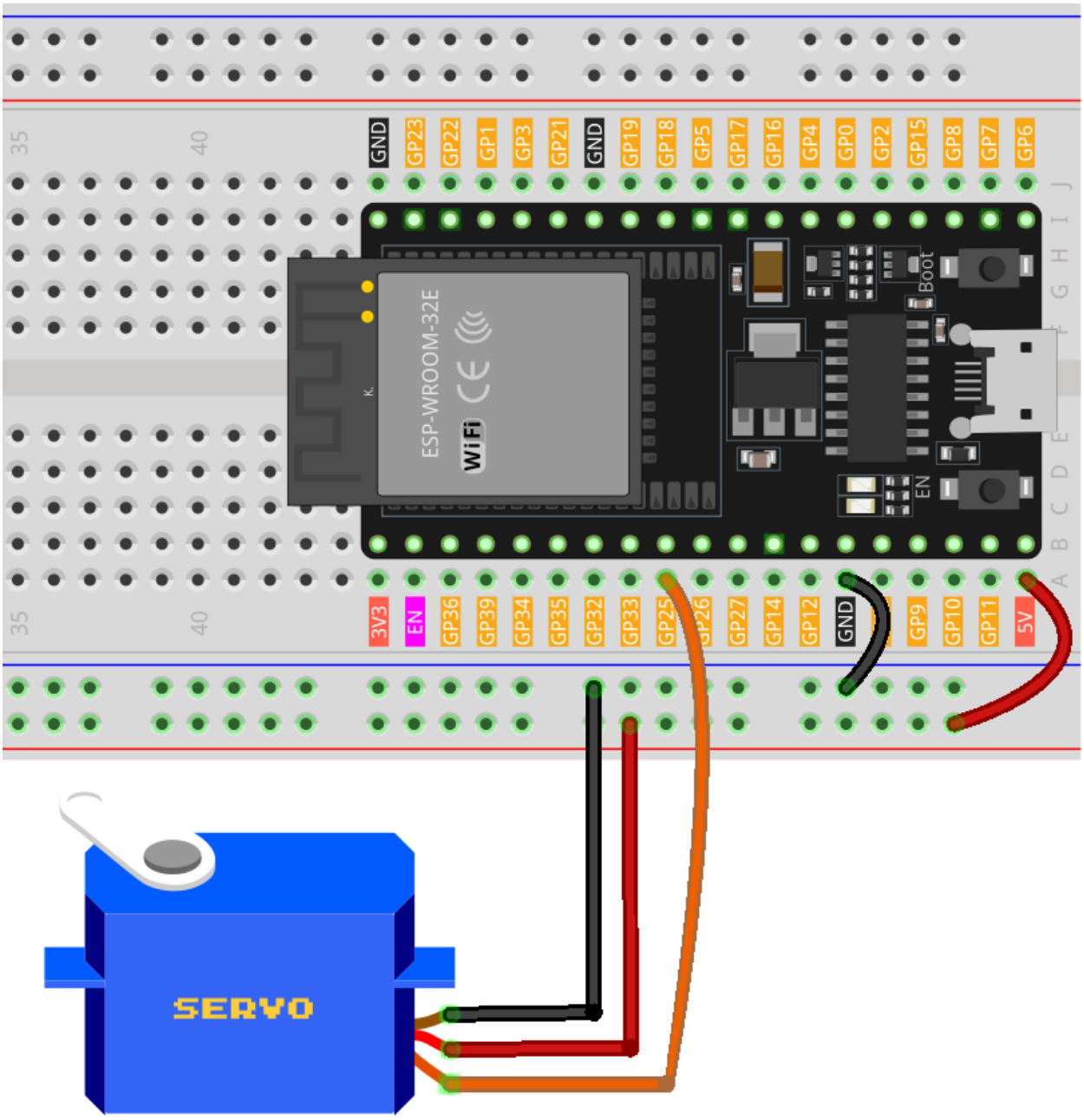
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Servo Motor (SG90)	
Breadboard	

Wiring


GND
GP23
GP22
GP1
GP3
GP21
GND
GP19
GP18
GP5
GP17
GP16
GP4
GP0
GP2
GP15
GP8
GP7
GP6
35
40
ESP-WROOM-32E
WiFi
Boot
EN
3V3
EN
GP36
GP39
GP34
GP35
GP32
GP33
GP25
GP26
GP27
GP14
GP12
GND
GP9
GP10
GP11
5V
SERVO

Code

Code Analysis

1. Including the Library

   The ESP32Servo library is included to manage servo motor operations.

   ```
   #include <ESP32Servo.h>
   ```

2. Defining Servo and Pin

   A Servo object is created, and a pin is defined for servo control.

   ```
   Servo myServo;
   const int servoPin = 25;
   ```

3. Setting Pulse Width Limits

   Minimum and maximum pulse widths are defined for servo motion limits.

   ```
   const int minPulseWidth = 500; // 0.5 ms
   const int maxPulseWidth = 2500; // 2.5 ms
   ```

4. Setup Function

   - The servo is attached to the defined pin and its pulse width range is set.
   - The PWM frequency is set to 50Hz, standard for servos.

   ```
   void setup() {
     myServo.attach(servoPin, minPulseWidth, maxPulseWidth);
     myServo.setPeriodHertz(50);
   }
   ```

5. Loop Function

   - Servo rotation is controlled in a loop, moving from 0 to 180 degrees, then back to 0 degrees.
   - `writeMicroseconds()` is used to set the servo position based on pulse width.

   ```
   void loop() {
     // Rotate the servo from 0 to 180 degrees
     for (int angle = 0; angle <= 180; angle++) {
       int pulseWidth = map(angle, 0, 180, minPulseWidth, maxPulseWidth);
       myServo.writeMicroseconds(pulseWidth);
       delay(15);
     }

     // Rotate the servo from 180 to 0 degrees
     for (int angle = 180; angle >= 0; angle--) {
       int pulseWidth = map(angle, 0, 180, minPulseWidth, maxPulseWidth);
       myServo.writeMicroseconds(pulseWidth);
       delay(15);
     }
   }
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.35 Lesson 34: TT Motor

In this lesson, you'll learn how to control a motor with the ESP32 Development Board and an L9110 motor control board. We'll cover defining and initializing motor pins, setting them as outputs, and adjusting the motor's speed using the analogWrite function. This project is ideal for those seeking to grasp motor control and pulse-width modulation (PWM) on the ESP32 platform, providing a hands-on demonstration of output operations in a microcontroller environment.

Required Components

In this project, we need the following components.

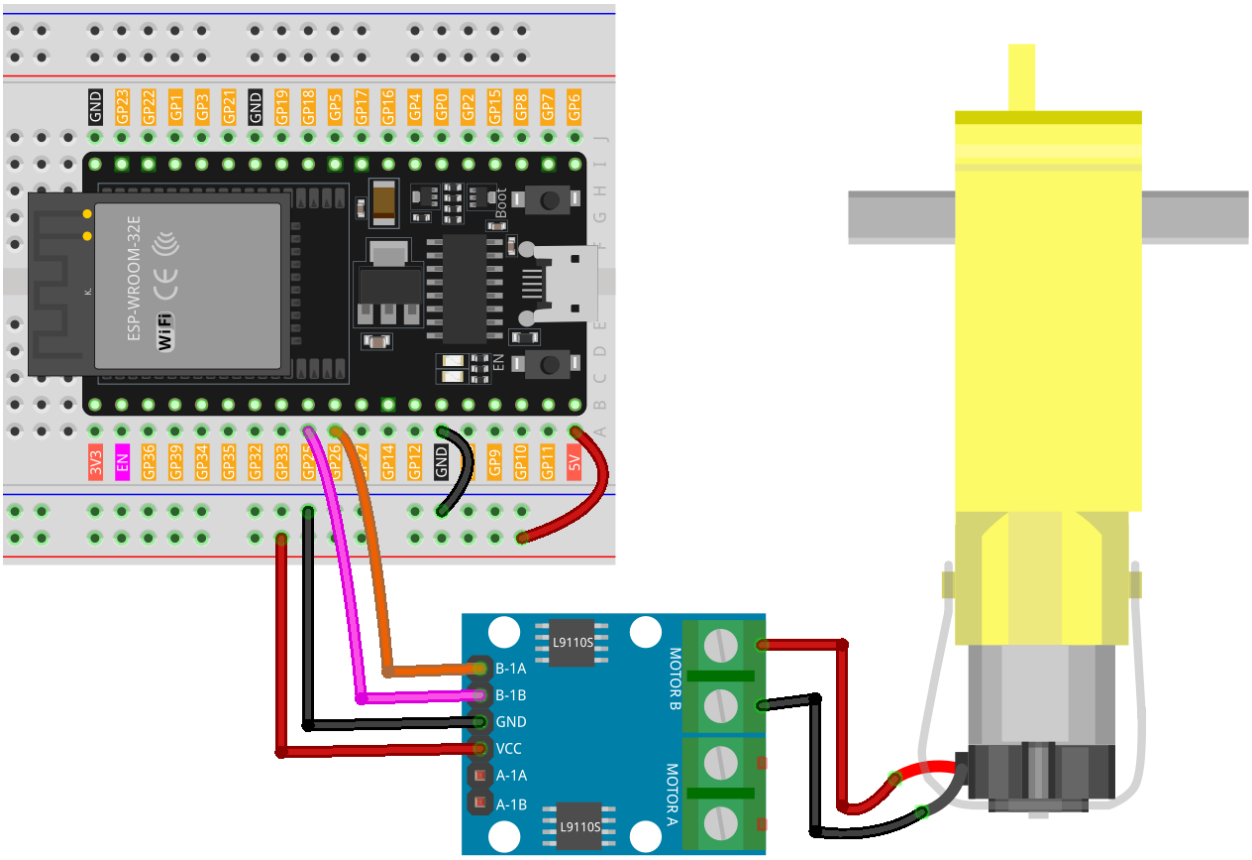
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
TT Motor	-
L9110 Motor Driver Module	-
Breadboard	

Wiring



Code

Code Analysis

1. The first part of the code defines the motor control pins. These are connected to the L9110 motor control board.

```
// Define the motor pins
const int motorB_1A = 26;
const int motorB_2A = 25;
```

2. The `setup()` function initializes the motor control pins as output using the `pinMode()` function. Then it uses `analogWrite()` to set the speed of the motor. The value passed to `analogWrite()` can range from 0 (off) to 255 (full speed). A `delay()` function is then used to pause the code for 5000 milliseconds (or 5 seconds), after which the motor speed is set to 0 (off).

```
void setup() {
  pinMode(motorB_1A, OUTPUT);  // set motor pin 1 as output
  pinMode(motorB_2A, OUTPUT);  // set motor pin 2 as output

  analogWrite(motorB_1A, 255);  // set motor speed (0-255)
  analogWrite(motorB_2A, 0);

  delay(5000);
```

(continues on next page)

(continued from previous page)

```
  analogWrite(motorB_1A, 0);
  analogWrite(motorB_2A, 0);
}
```

Fun Project

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.36 Lesson 35: Smart trashcan

This project revolves around the concept of a smart trash can. The primary aim is to have the trash can's lid automatically open when an object approaches within a set distance (20cm in this case). The functionality is achieved by using an ultrasonic distance sensor paired with a servo motor. The distance between the object and the sensor is continually measured. If the object is close enough, the servo motor is triggered to open the lid.

Required Components

In this project, we need the following components.

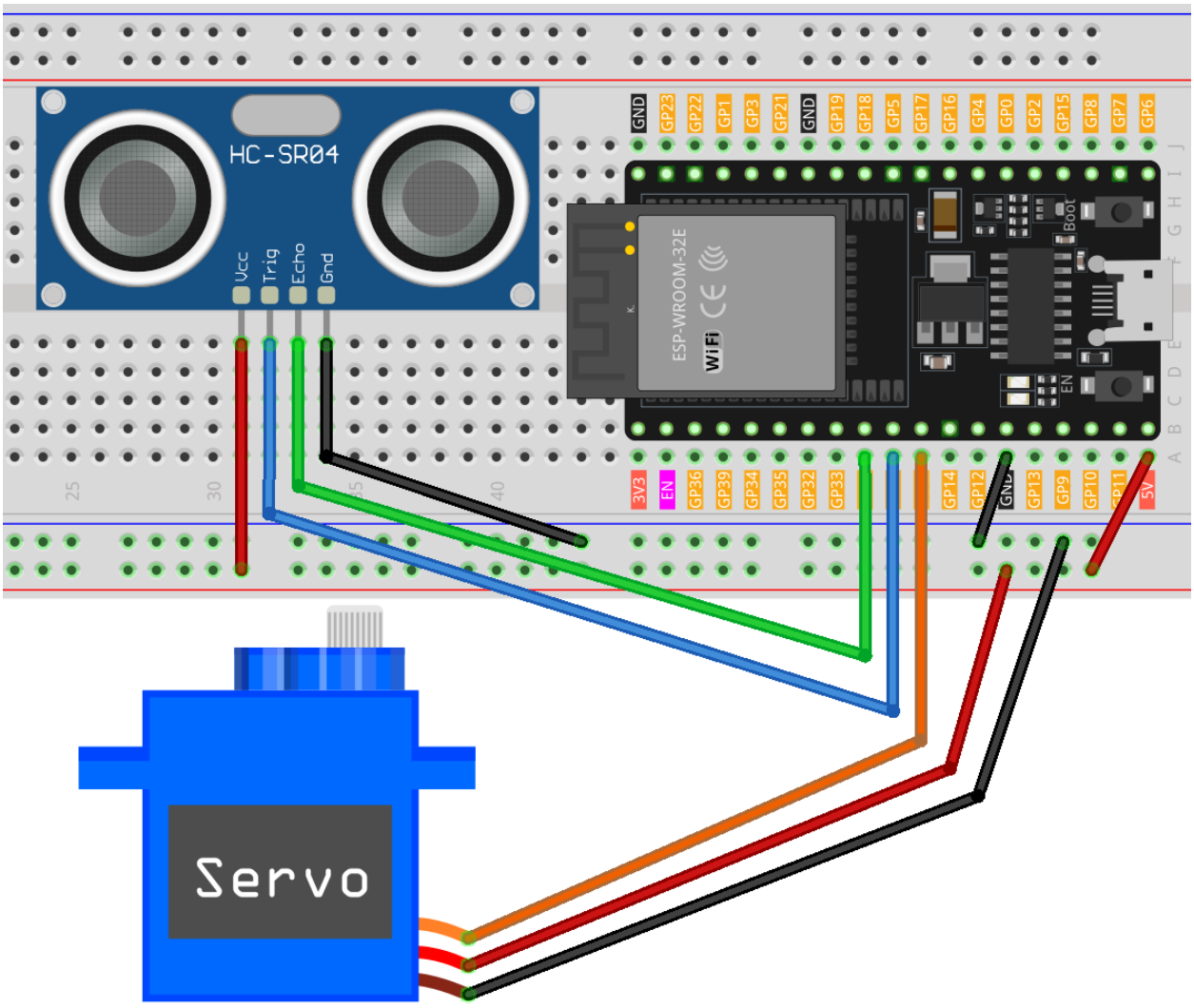
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Ultrasonic Sensor Module (HC-SR04)	
Servo Motor (SG90)	
Breadboard	

Wiring



Code

Code Analysis

The project is based on real-time monitoring of the distance between an object and a trash can. An ultrasonic sensor continuously measures this distance, and if an object approaches within 20cm, the trash can interprets it as an intention to dispose of waste and automatically opens its lid. This automation adds smartness and convenience to a regular trash can.

1. Initial Setup and Variable Declaration

   Here, we're including the `ESP32Servo` library and defining the constants and variables we'll use. The pins for the servo and the ultrasonic sensor are declared. We also have an array `averDist` to hold the three distance measurements.

```
#include <ESP32Servo.h>

// Set up the servo motor parameters
Servo servo;
const int servoPin = 27;
const int openAngle = 0;
const int closeAngle = 90;

// Define the minimum and maximum pulse widths for the servo
const int minPulseWidth = 500; // 0.5 ms
const int maxPulseWidth = 2500; // 2.5 ms


// Set up the ultrasonic sensor parameters
const int trigPin = 26;
const int echoPin = 25;
long distance, averageDistance;
long averDist[3];

// Distance threshold in centimeters
const int distanceThreshold = 20;
```

2. `setup()` Function

   The `setup()` function initializes serial communication, configures the ultrasonic sensor's pins, and sets the initial position of the servo to the closed position.

```
void setup() {
  Serial.begin(9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  servo.attach(servoPin);
  servo.write(closeAngle);
  delay(100);
}
```

3. `loop()` Function

   The `loop()` function is responsible for continuously measuring the distance, computing its average, and then making a decision whether to open or close the trash can's lid based on this averaged distance.

```
void loop() {
    // Measure the distance three times
    for (int i = 0; i <= 2; i++) {
        distance = readDistance();
        averDist[i] = distance;
        delay(10);
    }

    // Calculate the average distance
    averageDistance = (averDist[0] + averDist[1] + averDist[2]) / 3;
    Serial.println(averageDistance);

    // Control the servo based on the averaged distance
    if (averageDistance <= distanceThreshold) {
        servo.attach(servoPin);  // Reattach the servo before sending a command
        delay(1);
        servo.write(openAngle);  // Rotate the servo to the open position
        delay(3500);
    } else {
        servo.write(closeAngle);  // Rotate the servo back to the closed position
        delay(1000);
        servo.detach();  // Detach the servo to save power when not in use
    }
}
```

4. Distance Reading Function

   This function, `readDistance()`, is what actually interacts with the ultrasonic sensor. It sends a pulse and waits for an echo. The time taken for the echo is then used to calculate the distance between the sensor and any object in front of it.

   You can refer to the *Principle* of the ultrasonic sensor.

```
float readDistance() {
    // Send a pulse on the trigger pin of the ultrasonic sensor
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    // Measure the pulse width of the echo pin and calculate the distance value
    float distance = pulseIn(echoPin, HIGH) / 58.00;  // Formula: (340m/s * 1us) / 2
    return distance;
}
```

5. Servo Write Function

   This function maps the angle value to pulse width and calls the `writeMicroseconds(pulseWidth)` function to deflect the servo to a specific angle.

```
// Function to make the servo work
void servoWrite(int angle){
    int pulseWidth = map(angle, 0, 180, minPulseWidth, maxPulseWidth);
```

(continues on next page)

(continued from previous page)

```
    servo.writeMicroseconds(pulseWidth);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.37 Lesson 36: Gas leak alarm

This project revolves around simulating a gas leak detection scenario using an ESP32 board. By incorporating an MQ-2 gas sensor and an RGB LED, this demonstration continuously reads the gas concentration. If this concentration surpasses a predefined threshold, it activates an alarm (buzzer) and illuminates the RGB LED in red. Conversely, if the concentration remains below this threshold, the alarm remains inactive and the LED shines green. It's crucial to note that this demo is purely illustrative and shouldn't replace real gas leak detection systems.

Required Components

In this project, we need the following components.

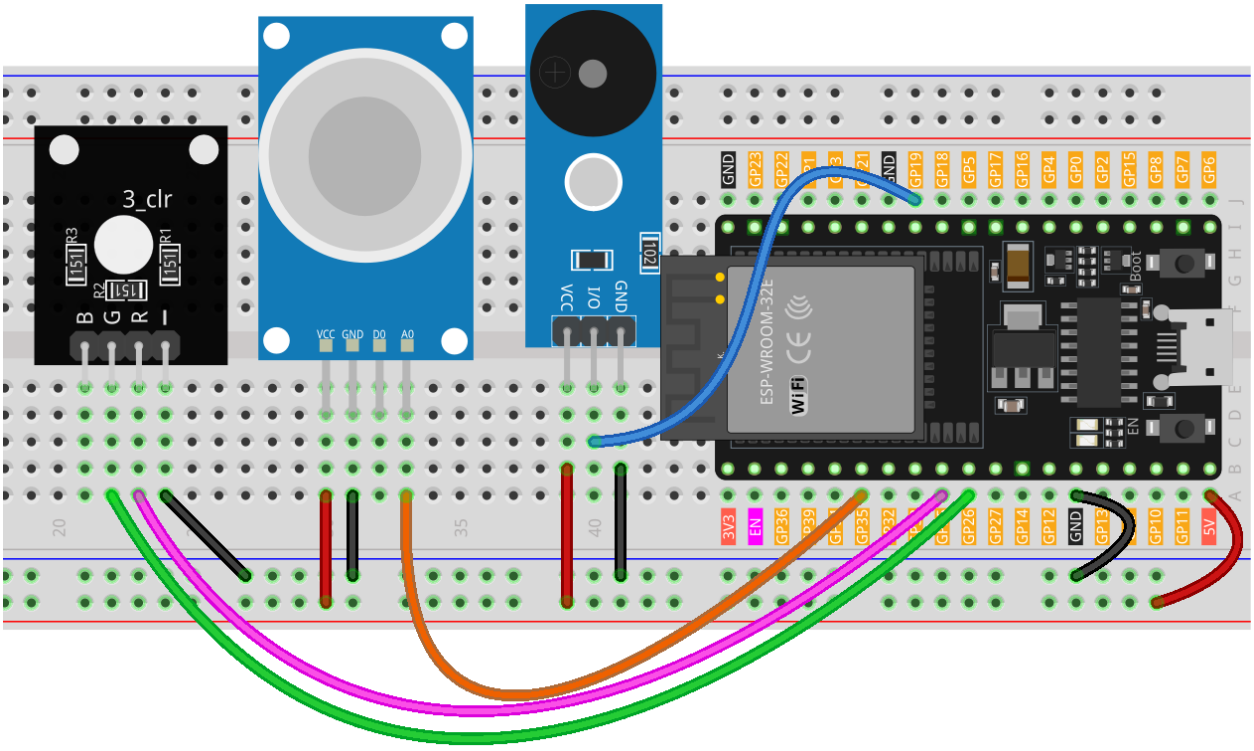
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Gas/Smoke Sensor Module (MQ2)	
Passive Buzzer Module	
RGB LED Module	-
Breadboard	

Wiring



Code

Code Analysis

The core principle of the project revolves around continuously monitoring the gas concentration. When the detected gas concentration surpasses a certain threshold, it sets off an alarm and changes the LED's color to red. This serves as a simulated warning mechanism, indicative of potentially hazardous conditions. If the concentration drops below the threshold, the alarm is deactivated and the LED switches to green, indicating a safe environment.

1. Defining Constants and Variables

   These lines declare and initialize the pin numbers for various components. The `sensorPin` denotes the analog pin where the MQ-2 gas sensor is connected. `sensorValue` is an integer variable storing the sensor's analog output. The `buzzerPin` indicates the digital pin to which the buzzer is connected. Finally, the `RPin` and `GPin` are the pins for the red and green channels of the RGB LED, respectively.

```
// Define the pin numbers for the Gas Sensor
const int sensorPin = 35;
int sensorValue;

// Define the pin number for the buzzer
const int buzzerPin = 19;

// Define pin numbers for the RGB LED
const int RPin = 25;  // R channel of RGB LED
const int GPin = 26;  // G channel of RGB LED
```

2. Initialization in `setup()`

   The `setup()` function initializes the required settings. Serial communication begins at a baud rate of 9600, allowing us to view sensor readings on the Serial Monitor. Pins for the buzzer and RGB LED are set as `OUTPUT`, meaning they'll send signals out to external components.

```
void setup() {
    Serial.begin(9600);  // Start serial communication at 9600 baud rate

    // Initialize the buzzer and RGB LED pins as output
    pinMode(buzzerPin, OUTPUT);
    pinMode(RPin, OUTPUT);
    pinMode(GPin, OUTPUT);
}
```

3. Main Loop: Reading Sensor and Triggering Alarm

   The `loop()` function continually reads the gas sensor's output. The reading is then displayed on the Serial Monitor for observation. Depending on the sensor value, two scenarios can occur:

   - If the value exceeds 300, the buzzer is activated using `tone()`, and the RGB LED turns red.
   - If the value is below 300, the buzzer is silenced using `noTone()`, and the LED turns green.

   Lastly, a delay of 50 milliseconds is introduced before the next loop iteration to manage the read frequency and reduce the CPU load.

```
void loop() {
    // Read the analog value of the gas sensor
    sensorValue = analogRead(sensorPin);

    // Print the sensor value to the serial monitor
    Serial.print("Analog output: ");
    Serial.println(sensorValue);

    // If the sensor value exceeds the threshold, trigger the alarm and␣
↪make the RGB LED red
    if (sensorValue > 3000) {
        tone(buzzerPin, 500, 300);
        digitalWrite(GPin, LOW);
        digitalWrite(RPin, HIGH);
        delay(500);
        // stop the tone playing:
        noTone(buzzerPin);
    } else {
        // If the sensor value is below the threshold, turn off the alarm␣
↪and make the RGB LED green
        noTone(buzzerPin);
        digitalWrite(RPin, LOW);
        digitalWrite(GPin, HIGH);
    }

    // Wait for 50 milliseconds before the next loop iteration
    delay(50);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.38 Lesson 37: Automatic soap dispenser

The Automatic Soap Dispenser project uses an Arduino Uno board along with an infrared obstacle avoidance sensor and a water pump. The sensor detects the presence of an object such as a hand, which activates the water pump to dispense soap.

Required Components

In this project, we need the following components.

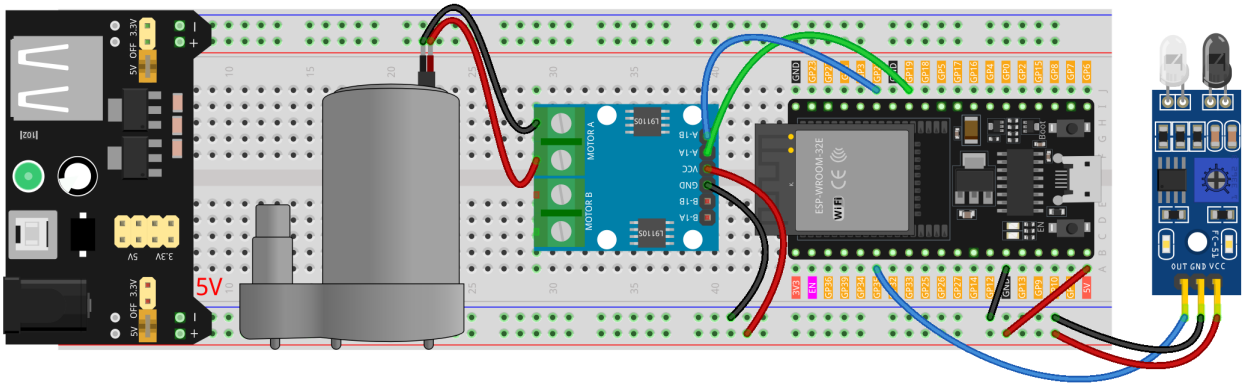
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
IR Obstacle Avoidance Sensor Module	
Centrifugal Pump	-
L9110 Motor Driver Module	-
Power Supply Module	-
Breadboard	

Wiring



Code

Code Analysis

The main idea behind this project is to create a hands-free soap dispensing system. The infrared obstacle avoidance sensor detects when an object (like a hand) is close. Upon detecting an object, the sensor sends a signal to the Arduino, which in turn triggers the water pump to dispense soap. The pump stays active for a brief period, dispensing soap, then turns off.

1. **Defining the pins for the sensor and the pump**

   In this code snippet, we define the Arduino pins that connect to the sensor and pump. We define pin 7 as the sensor pin and we will use the variable `sensorValue` to store the data read from this sensor. For the water pump, we use two pins, 9 and 10.

   ```
   // Define the pin numbers for the Infrared obstacle avoidance sensor
   const int sensorPin = 35;
   int sensorValue;

   // Define pin numbers for the water pump
   const int pump1A = 19;
   const int pump1B = 21;
   ```

2. **Setting up the sensor and pump**

   In the `setup()` function, we define the modes for the pins we're using. The sensor pin is set to `INPUT` as it will be used to receive data from the sensor. The pump pins are set to `OUTPUT` as they will send commands to the pump. We ensure that the pin `pump1B` starts in a `LOW` state (off), and we start the serial communication with a baud rate of 9600.

   ```
   void setup() {
       // Set the sensor pin as input
       pinMode(sensorPin, INPUT);

       // Initialize the pump pins as output
       pinMode(pump1A, OUTPUT);
       pinMode(pump1B, OUTPUT);

       // Keep pump1B low
   ```

(continues on next page)

(continued from previous page)

```
    digitalWrite(pump1A, LOW);
    digitalWrite(pump1B, LOW);

    Serial.begin(9600);
}
```

3. **Continuously checking the sensor and controlling the pump**

   In the `loop()` function, the Arduino constantly reads the value from the sensor using `digitalRead()` and assigns it to `sensorValue()`. It then prints this value to the serial monitor for debugging purposes. If the sensor detects an object, `sensorValue()` will be 0. When this happens, `pump1A` is set to `HIGH`, activating the pump, and a delay of 700 milliseconds allows the pump to dispense soap. The pump is then deactivated by setting `pump1A` to `LOW`, and a 1-second delay gives the user time to move their hand away before the cycle repeats.

Note: If the sensor is not working properly, adjust the IR transmitter and receiver to make them parallel. Additionally, you can adjust the detection range using the built-in potentiometer.

```
void loop() {
    sensorValue = digitalRead(sensorPin);
    Serial.println(sensorValue);

    // If an object is detected, turn on the pump for a brief period, then turn it␣
↪off
    if (sensorValue == 0) {
        digitalWrite(pump1A, HIGH);
        delay(700);
        digitalWrite(pump1A, LOW);
        delay(1000);
    }
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.39 Lesson 38: Motion triggered relay

This project aims to control a relay-operated light using a passive infrared (PIR) sensor. When the PIR sensor detects motion, the relay is activated, turning the light on. The light remains on for 5 seconds after the last detected motion.

> **Warning:** As a demonstration, we are using a relay to control an RGB LED module. However, in real-life scenarios, this may not be the most practical approach.
>
> **While you can connect the relay to other appliances in actual applications, extreme caution is required when dealing with HIGH AC voltage. Improper or incorrect use can lead to severe injury or even death. Therefore, it is intended for people who are familiar with and knowledgeable about HIGH AC voltage. Always prioritize safety.**

Required Components

In this project, we need the following components.

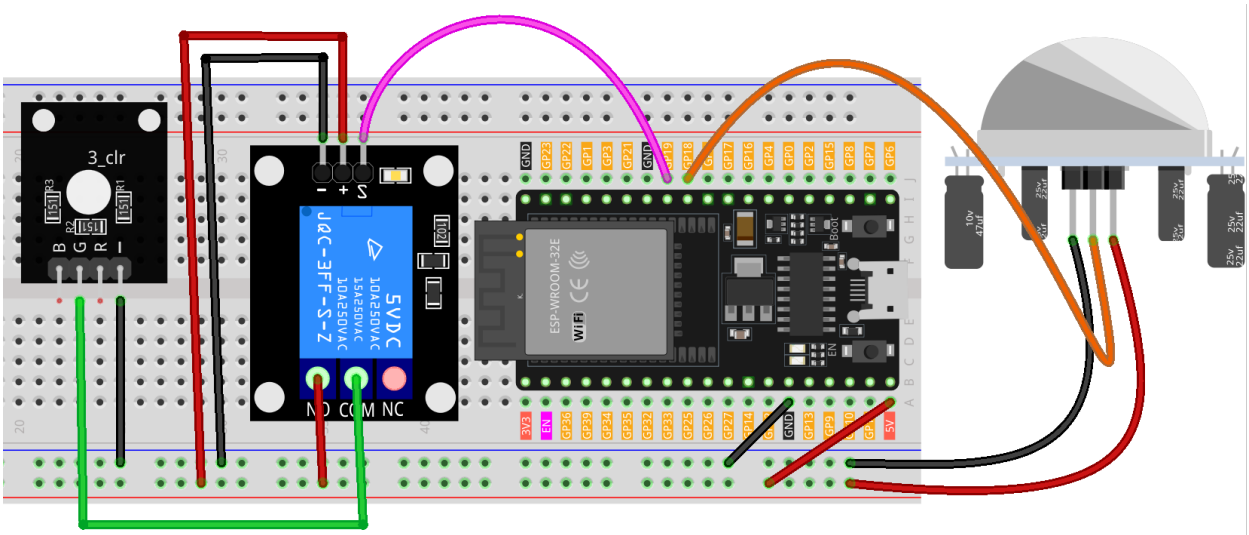
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
PIR Motion Module (HC-SR501)	-
5V Relay Module	-
RGB LED Module	-
Breadboard	

Wiring



Code

Code Analysis

The project revolves around the PIR motion sensor's capability to detect motion. When motion is detected, a signal is sent to the Arduino, triggering the relay module, which in turn activates a light. The light stays on for a specified duration (in this case, 5 seconds) after the last detected motion, ensuring the area remains illuminated for a short period even if motion ceases.

1. **Initial setup and variable declarations**

   This segment defines constants and variables that will be used throughout the code. We set up the relay and PIR pins and a delay constant for motion. We also have a variable to keep track of the last detected motion time and a flag to monitor if motion is detected.

   ```
   // Define the pin number for the relay
   const int relayPin = 19;

   // Define the pin number for the PIR sensor
   const int pirPin = 18;

   // Motion delay threshold in milliseconds
   const unsigned long MOTION_DELAY = 5000;

   unsigned long lastMotionTime = 0;  // Timestamp of the last motion detection
   bool motionDetected = false;       // Flag to track if motion is detected
   ```

2. **Configuration of pins in setup() function**

   In the `setup()` function, we configure the pin modes for both the relay and PIR sensor. We also initialize the relay to be off at the start.

   ```
   void setup() {
       pinMode(relayPin, OUTPUT);    // Set relayPin as an output pin
   ```

(continues on next page)

(continued from previous page)

```
    pinMode(pirPin, INPUT);       // Set the PIR pin as an input
    digitalWrite(relayPin, LOW);  // Turn off the relay initially
}
```

3. **Main logic in loop() function**

   The `loop()` function contains the primary logic. When the PIR sensor detects motion, it sends a `HIGH` signal, turning on the relay and updating the `lastMotionTime`. If there's no motion for the specified delay (5 seconds in this case), the relay is turned off.

   This approach ensures that even if motion is sporadic or brief, the light remains on for at least 5 seconds after the last detected motion, providing a consistent illumination duration.

```
void loop() {
    if (digitalRead(pirPin) == HIGH) {
        lastMotionTime = millis();     // Update the last motion time
        digitalWrite(relayPin, HIGH);  // Turn on the relay (and hence the␣
↪light)
        motionDetected = true;
    }

    // If motion was detected earlier and 5 seconds have elapsed, turn off␣
↪the relay
    if (motionDetected && (millis() - lastMotionTime >= MOTION_DELAY)) {
        digitalWrite(relayPin, LOW);  // Turn off the relay
        motionDetected = false;
    }
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.40 Lesson 39: Heart rate monitor

This Arduino project aims to build a simple Heart Rate Monitor using a MAX30102 pulse oximeter sensor and an SSD1306 OLED Display. The code takes measurements of the heart rate by determining the time between heartbeats. By taking four measurements, it computes their average and presents the resultant average heart rate on an OLED screen. If the sensor doesn't detect a finger, it sends a prompt to the user to position their finger correctly on the sensor.

Required Components

In this project, we need the following components.

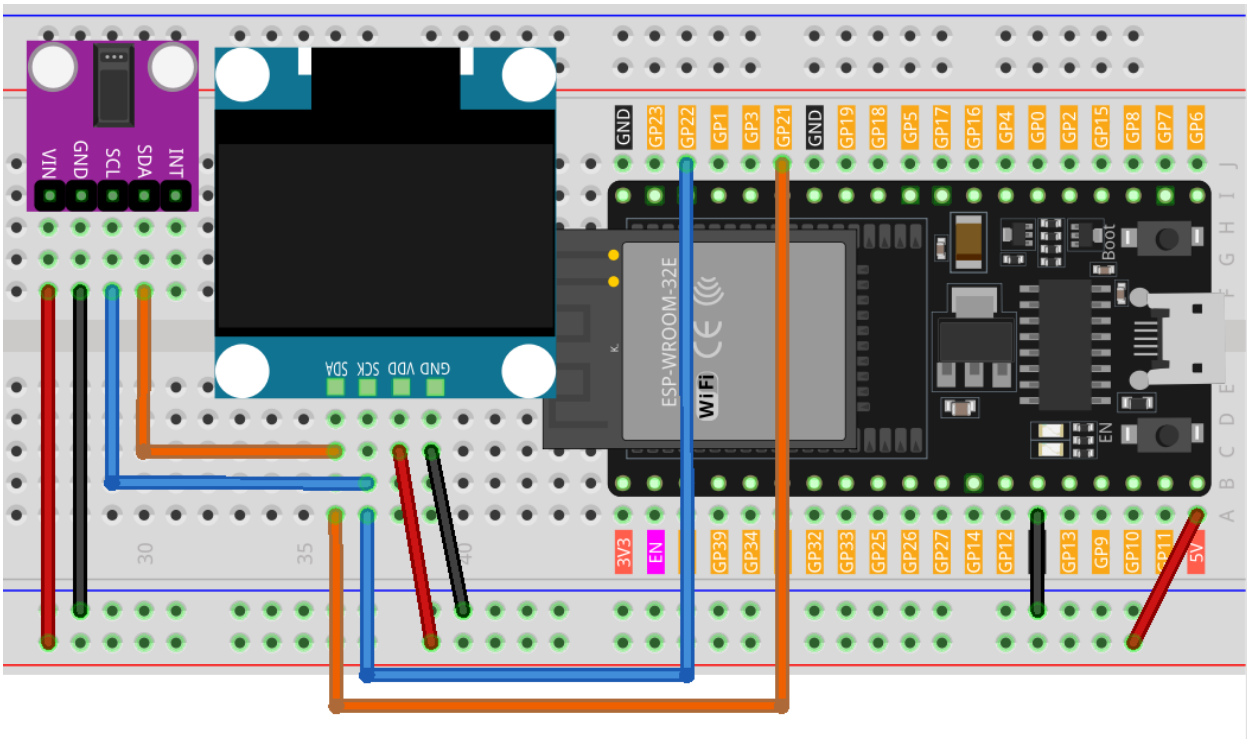
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Pulse Oximeter and Heart Rate Sensor Module (MAX30102)	
OLED Display Module (SSD1306)	-
Breadboard	

Wiring



Code

Note: To install the library, open the Arduino Library Manager, search for **"SparkFun MAX3010x"** , **"Adafruit SSD1306"** , and **"Adafruit GFX"**, then install them.

Code Analysis

The main principle behind this project is to capture the pulsation of blood flow through a finger using the MAX30102 sensor. As blood pumps through the body, it causes tiny changes in the volume of blood in the vessels of the fingertip. By shining light through the finger and measuring the amount of light that gets absorbed or reflected back, the sensor detects these minute volume changes. The time interval between subsequent pulses is then used to calculate the heart rate in beats per minute (BPM). This value is then averaged over four measurements and displayed on the OLED screen.

1. **Library Inclusions and Initial Declarations**:

   The code begins by including necessary libraries for the OLED display, MAX30102 sensor, and heart rate calculation. Additionally, the configuration for the OLED display and the variables for heart rate calculation are declared.

   Note: To install the library, open the Arduino Library Manager, search for **"SparkFun MAX3010x"** , **"Adafruit SSD1306"** , and **"Adafruit GFX"**, then install them.

```
#include <Adafruit_GFX.h>  // OLED  libraries
#include <Adafruit_SSD1306.h>
#include <Wire.h>
#include "MAX30105.h"   // MAX3010x library
#include "heartRate.h"  // Heart rate  calculating algorithm

// ... Variables and OLED configuration
```

In this project, we've also whipped up a couple of bitmaps. The `PROGMEM` keyword denotes that the array is stored in the program memory of the microcontroller. Storing data in program memory(PROGMEM) instead of RAM can be helpful for large amounts of data, which would otherwise take up too much space in RAM.

```
static const unsigned char PROGMEM beat1_bmp[] = {...}

static const unsigned char PROGMEM beat2_bmp[] = {...}
```

2. **Setup Function**:

   Initializes I2C communication, starts serial communication, initializes the OLED display, and sets up the MAX30102 sensor.

```
void setup() {
    Wire.setClock(400000);
    Serial.begin(9600);
    display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);
    // ... Rest of the setup code
```

3. **Main Loop**:

   The core functionality resides here. The IR value is read from the sensor. If a finger is detected (IR value greater than 50,000), the program checks if a heartbeat is sensed. When a heartbeat is detected, the OLED screen displays the BPM and the time between heartbeats is used to calculate BPM. Otherwise, it prompts the user to place their finger on the sensor.

   We have also prepared two bitmaps with heartbeats, and by switching between these two bitmaps, we can achieve a dynamic visual effect.

```
void loop() {
  // Get IR value from sensor
  long irValue = particleSensor.getIR();

  //If a finger is detected
  if (irValue > 50000) {

    // Check if a beat is detected
    if (checkForBeat(irValue) == true) {

      // Update OLED display
      // Calculate the BPM

      // Calculate the average BPM
      //Print the IR value, current BPM value, and average BPM value to the serial␣
↪monitor

      // Update OLED display
```

(continues on next page)

(continued from previous page)

```
    }
  }
  else {
    // ... Prompt to place the finger on the sensor
  }
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.41 Lesson 40: Touch toggle light

This project is a simple implementation of a traffic light control system utilizing a touch sensor and a traffic light LED module. Activating the touch sensor initiates a sequence where LEDs illuminate in the following order: Red -> Yellow -> Green.

Required Components

In this project, we need the following components.

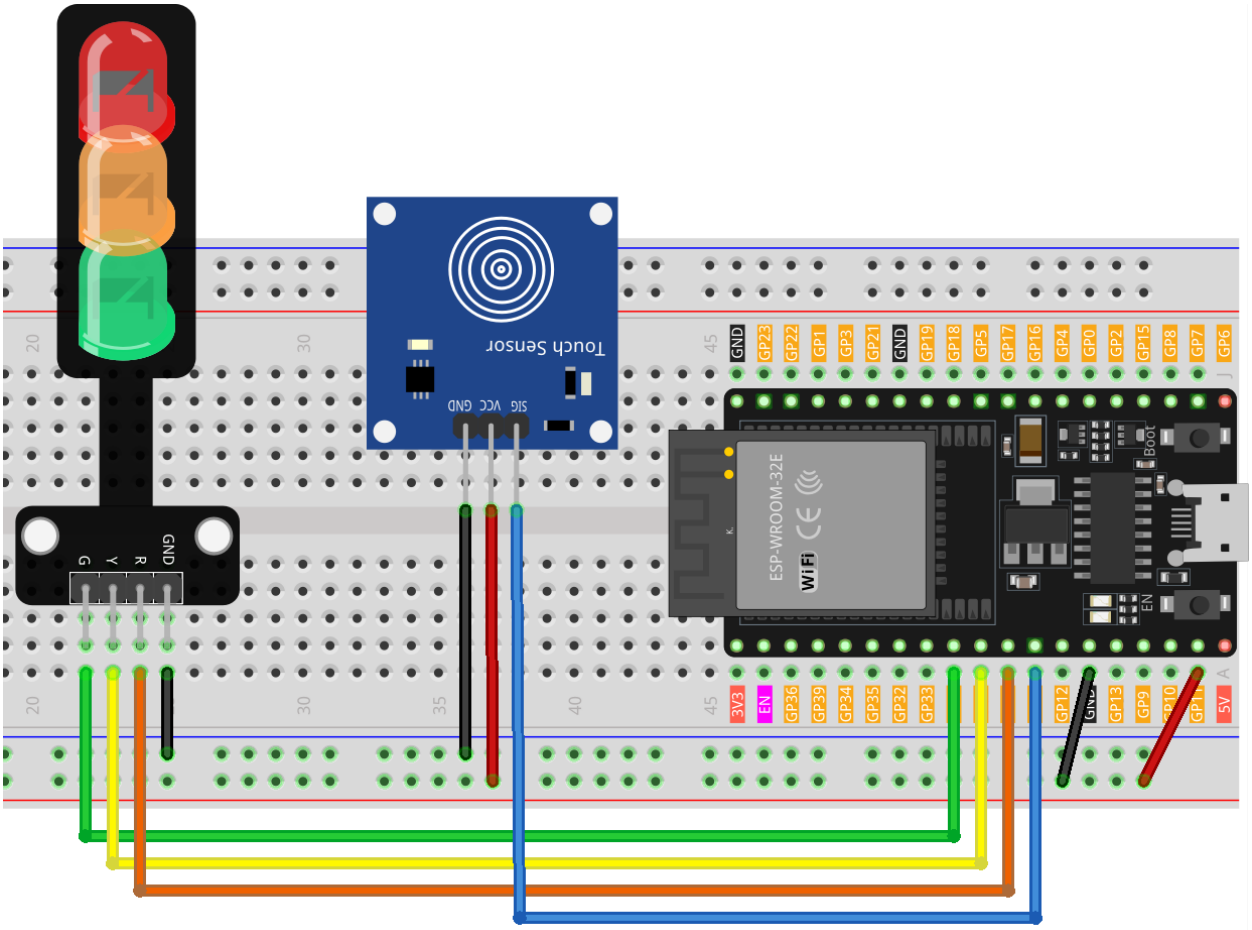
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Touch Sensor Module	-
Traffic Light Module	-
Breadboard	

Wiring



Code

Code Analysis

The operation of this project is straightforward: a touch detection on the sensor triggers the illumination of the next LED in the sequence (Red -> Yellow -> Green), controlled by the `currentLED` variable.

1. Define pins and initial values

```
// Define pins for touch sensor and LEDs
const int touchSensorPin = 14;  // touch sensor pin
const int rledPin = 27;         // red LED pin
const int yledPin = 26;         // yellow LED pin
const int gledPin = 25;         // green LED pin

int lastTouchState;     // the previous state of touch sensor
int currentTouchState;  // the current state of touch sensor
int currentLED = 0;     // current LED 0->Red, 1->Yellow, 2->Green
```

These lines establish the pin connections for the Arduino board components and initialize the touch sensor and LED states.

2. setup() function

```
void setup() {
  Serial.begin(9600);              // initialize serial
  pinMode(touchSensorPin, INPUT);  // configure touch sensor pin as input

  // set LED pins as outputs
  pinMode(rledPin, OUTPUT);
  pinMode(yledPin, OUTPUT);
  pinMode(gledPin, OUTPUT);

  currentTouchState = digitalRead(touchSensorPin);
}
```

This function configures the initial setup for the Arduino, defining input and output modes and starting serial communication for debugging.

3. loop() function

```
void loop() {
  lastTouchState = currentTouchState;               // save the last state
  currentTouchState = digitalRead(touchSensorPin);  // read new state

  // check if the touch sensor was just touched
  if (lastTouchState == LOW && currentTouchState == HIGH) {
    Serial.println("The sensor is touched");

    turnAllLEDsOff();  // Turn off all LEDs

    // switch on the next LED in sequence
    switch (currentLED) {
      case 0:
        digitalWrite(rledPin, HIGH);
        currentLED = 1;
        break;
      case 1:
        digitalWrite(yledPin, HIGH);
        currentLED = 2;
        break;
      case 2:
        digitalWrite(gledPin, HIGH);
        currentLED = 0;
        break;
    }
  }
}
```

The loop continuously monitors the touch sensor, cycling through the LEDs when a touch is detected, ensuring only one LED is on at any given time.

4. Turn off LEDs function

```
// function to turn off all LEDs
void turnAllLEDsOff() {
  digitalWrite(rledPin, LOW);
```

(continues on next page)

(continued from previous page)

```
  digitalWrite(yledPin, LOW);
  digitalWrite(gledPin, LOW);
}
```

This auxiliary function turns off all LEDs, aiding in the cycling process.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.42 Lesson 41: Potentiometer scale value

This project focuses on reading a potentiometer's value and displaying it on an LCD 1620 equipped with an I2C interface. Additionally, the value is transmitted to the serial monitor for live monitoring. A distinctive aspect of this project is the graphical representation of the potentiometer's value on the LCD, which is depicted as a variable-length bar proportional to the potentiometer's reading.

Required Components

In this project, we need the following components.

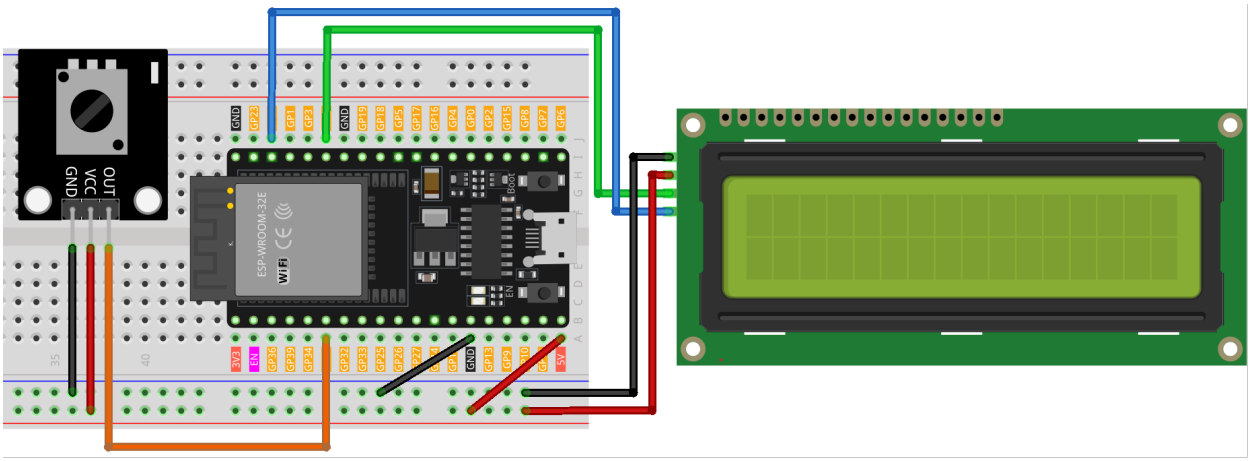
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Potentiometer Module	-
I2C LCD 1602	-
Breadboard	

Wiring



Code

Code Analysis

The core functionality of this project is to consistently read the potentiometer's value, map it to a scaled range (0-16), and display the result both numerically and graphically on the LCD. The implementation minimizes jitter by updating the display only when significant changes in the reading occur, thus maintaining a smooth visual experience.

1. **Library Inclusion and Initialization**:

```
// Required libraries for I2C and LCD operations
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Initialize LCD at I2C address 0x27 with 16 columns and 2 rows
LiquidCrystal_I2C lcd(0x27, 16, 2);
```

   This segment incorporates the necessary libraries for I2C communication and LCD control. It then initializes an LCD instance with the I2C address of `0x27`, specifying its dimensions as `16 columns` and `2 rows`.

2. **Variable Declaration**:

```
// Variables to hold the potentiometer readings
int lastRead = 0;     // Previous potentiometer value
int currentRead = 0;  // Current potentiometer value
```

   Variables `lastRead` and `currentRead` are used to keep track of the potentiometer's readings across different moments.

3. **setup() Function**:

```
void setup() {
  lcd.init();          // Initiates the LCD
  lcd.backlight();     // Activates the LCD's backlight
  Serial.begin(9600);  // Commences serial communication at 9600 baud
}
```

This function prepares the LCD and starts serial communication, setting up the environment for the project's operation.

4. **Main Loop**:

```
void loop() {
   // Read the current potentiometer value
   int currentRead = analogRead(35);

   // Map the read value from 0-4096 to 0-16
   int barLength = map(currentRead, 0, 4096, 0, 16);

   // Update LCD only if the difference between current and last reading is greater␣
↪than 2 to avoid jitter
   if (abs(lastRead - currentRead) > 2) {
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Value:");
      lcd.setCursor(7, 0);
      lcd.print(currentRead);
      Serial.println(currentRead);

      // Display a bar on the second row of LCD proportional to the potentiometer␣
↪value
      for (int i = 0; i < barLength; i++) {
         lcd.setCursor(i, 1);
         lcd.print(char(255));
      }
   }
   // Update the last read value for the next iteration
   lastRead = currentRead;

   // Introduce a delay for a stable reading
   delay(200);
}
```

- Reads the potentiometer and converts its value to a scale suitable for visual representation.
- Updates the LCD only when a meaningful change is detected, displaying the numeric value and a corresponding bar graph.
- Also sends the reading to the serial monitor for external observation.
- Ensures stability and responsiveness by introducing a brief delay between iterations.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.43 Lesson 42: Digital Dice

This program simulates a dice roll using an OLED display. The simulation is triggered by shaking the vibration switch, causing the display to cycle through numbers 1 to 6, akin to rolling a dice. The display halts after a short duration, revealing a randomly selected number that represents the dice roll outcome.

Required Components

In this project, we need the following components.

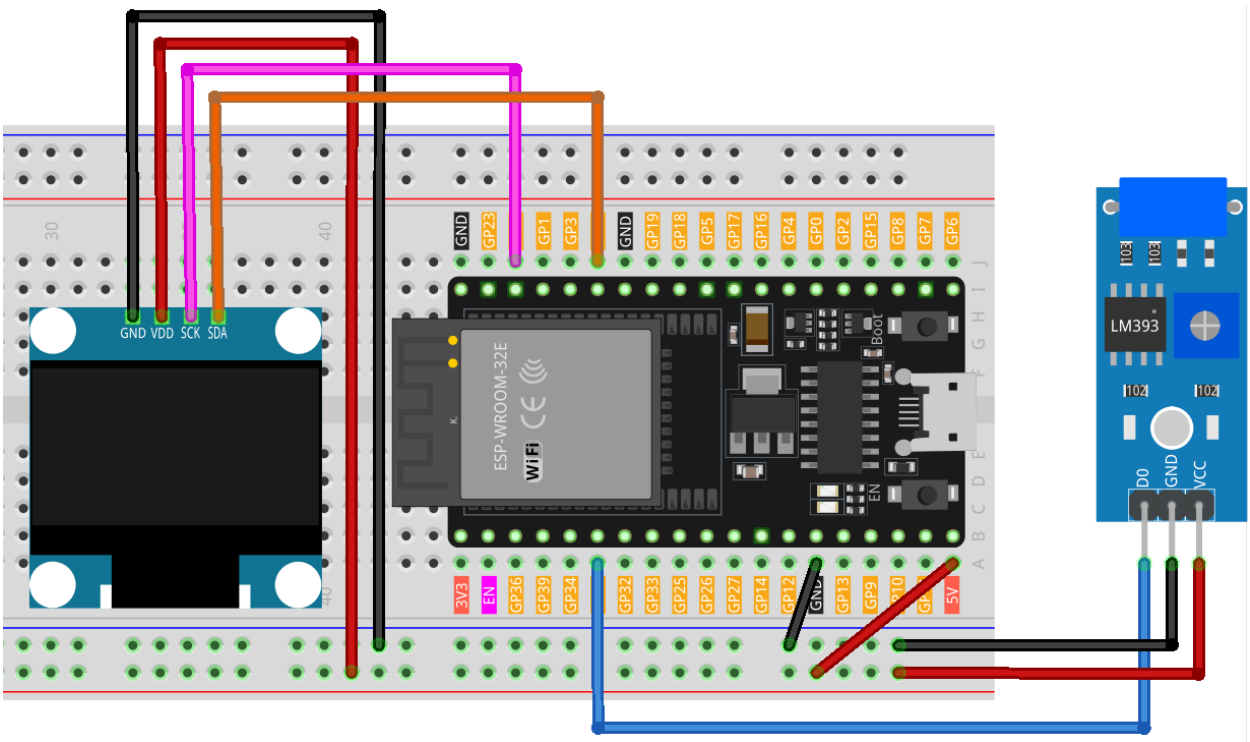
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Vibration Sensor Module (SW-420)	
OLED Display Module (SSD1306)	-
Breadboard	

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"Adafruit SSD1306"** and **"Adafruit GFX"** and install it.

Code Analysis

A comprehensive breakdown of the code:

1. Initialization of variables:

   `vibPin`: Digital pin connected to the vibration sensor.

   ```
   const int vibPin = 35;    // The pin where the vib switch is connected
   ```

2. Volatile variables:

   `rolling`: A volatile flag that indicates the dice's rolling status. It is volatile as it is accessed within both the interrupt service routine and the main program.

   ```
   volatile bool rolling = false;
   ```

3. `setup()`:

   Configures the vibration sensor's input mode. Assigns an interrupt to the sensor to trigger the rollDice function upon state change. Initializes the OLED display.

```
void setup() {
    // Initialize pins
    pinMode(vibPin, INPUT);

    // initialize the OLED object
    if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
        Serial.println(F("SSD1306 allocation failed"));
        for (;;)
        ;
    }

    // Attach an interrupt to the vibPin. When the vib switch is activated,␣
↪the shakeDetected function will be called
    attachInterrupt(digitalPinToInterrupt(vibPin), rollDice, CHANGE);
}
```

4. `loop()`:

   Continuously checks if `rolling` is true, displaying a random number between 1 and 6 during this state. The rolling ceases if the sensor has been shaken for over 500 milliseconds.

```
void loop() {
    // Check if it's rolling
    if (rolling) {
        byte number = random(1, 7);  // Generate a random number between 1␣
↪and 6
        displayNumber(number);
        delay(80);  // Delay to make the rolling effect visible

        // Stop rolling after 1 second
        if ((millis() - lastShakeTime) > 1000) {
            rolling = false;
        }
    }
}
```

5. `rollDice()`:

   The interrupt service routine for the vibration sensor. It initiates the dice roll when the sensor is shaken by recording the current time.

```
// Interrupt handler for shake detection
void rollDice() {
    if (digitalRead(vibPin) == LOW) {
        lastShakeTime = millis();  // Record the time of shake
        rolling = true;            // Start rolling
    }
}
```

6. `displayNumber()`:

   Displays a selected number on the OLED screen.

```
// Function to display a number on the 7-segment display
void displayNumber(byte number) {
```

(continues on next page)

(continued from previous page)

```
    display.clearDisplay();  // Clear the screen

    // Display Text
    display.setTextSize(4);       // Set text size
    display.setTextColor(WHITE);  // Set text color
    display.setCursor(54, 20);     // Set cursor position
    display.println(number);
    display.display();  // Display the content on the screen

}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.44 Lesson 43: Plant Monitor

This project intelligently automates plant watering by triggering a water pump whenever the soil's moisture level dips below a predetermined threshold. It also features an LCD display that showcases the temperature, humidity, and soil moisture levels, offering users valuable insights into the plant's environmental conditions.

Required Components

In this project, we need the following components.

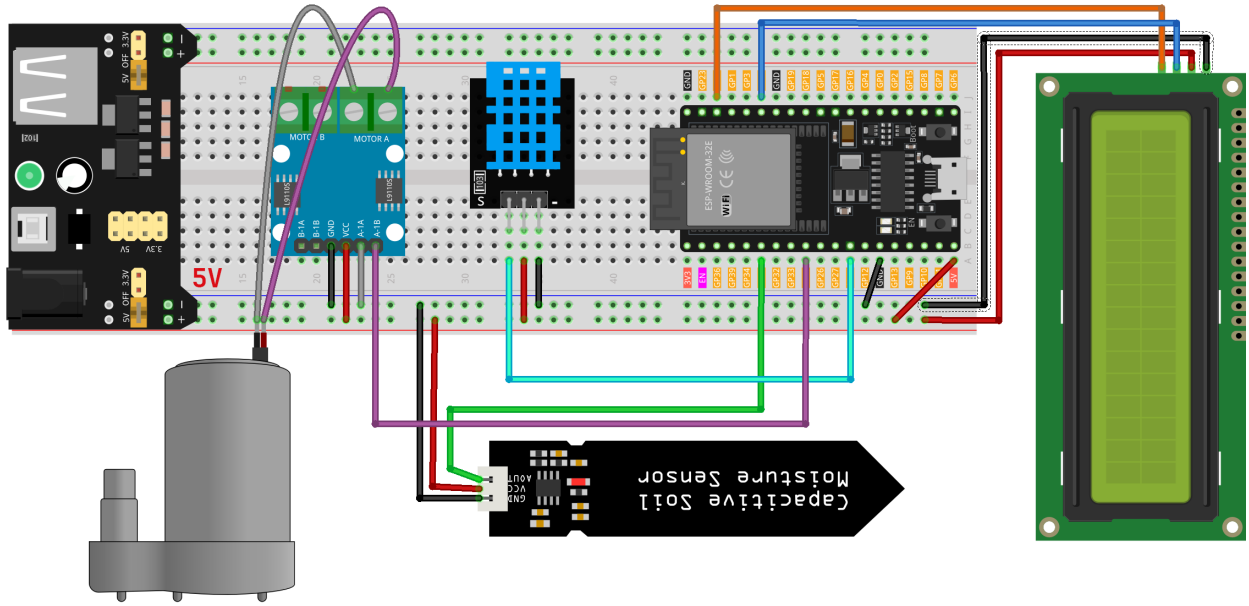
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
ESP32 & Development Board	
Breadboard	
Power Supply Module	-
I2C LCD 1602	
Centrifugal Pump	-
L9110 Motor Driver Module	-
Capacitive Soil Moisture Module	
Temperature and Humidity Sensor Module (DHT11)	-

Wiring



Code

Note: To install the library, use the Arduino Library Manager and search for **"LiquidCrystal I2C"** and **"DHT sensor library"** and install it.

Code Analysis

The code is structured to seamlessly manage plant watering by monitoring environmental parameters:

1. Library Inclusions and Constants/Variables:

   Incorporate `Wire.h`, `LiquidCrystal_I2C.h`, and `DHT.h` libraries for functionality. Specify pin assignments and settings for the DHT11 sensor, soil moisture sensor, and water pump.

   ```
   #include <Wire.h>
   #include <LiquidCrystal_I2C.h>
   #include <DHT.h>

   #define DHTPIN 14              // Digital pin for DHT11 sensor
   #define DHTTYPE DHT11          // DHT11 sensor type
   #define SOIL_MOISTURE_PIN 35  // Analog pin for soil moisture sensor
   #define WATER_PUMP_PIN 25      // Digital pin for water pump


   // Initialize sensor and LCD objects
   DHT dht(DHTPIN, DHTTYPE);
   LiquidCrystal_I2C lcd(0x27, 16, 2);
   ```

2. `setup()`:

   Configure pin modes for the moisture sensor and pump. Initially deactivate the pump. Initialize and backlight the LCD. Activate the DHT sensor.

   ```
   void setup() {
       // Set pin modes
       pinMode(SOIL_MOISTURE_PIN, INPUT);
       pinMode(WATER_PUMP_PIN, OUTPUT);

       // Initialize water pump as off
       digitalWrite(WATER_PUMP_PIN, LOW);

       // Initialize LCD and backlight
       lcd.init();
       lcd.backlight();

       // Start DHT sensor
       dht.begin();
   }
   ```

3. `loop()`:

   Measure humidity and temperature via the DHT sensor. Gauge soil moisture through the soil moisture sensor. Display the temperature and humidity on the LCD, then show soil moisture levels. Assess

soil moisture to decide on water pump activation; if soil moisture is under 500 (adjustable threshold), run the pump for 1 second.

```
void loop() {
    // Read humidity and temperature from DHT11
    float humidity = dht.readHumidity();
    float temperature = dht.readTemperature();

    // Read soil moisture level
    int soilMoisture = analogRead(SOIL_MOISTURE_PIN);

    // Display temperature and humidity on LCD
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Temp: " + String(temperature) + "C");
    lcd.setCursor(0, 1);
    lcd.print("Humidity: " + String(humidity) + "%");

    delay(2000);

    // Display soil moisture on LCD
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Soil Moisture: ");
    lcd.setCursor(0, 1);
    lcd.print(String(soilMoisture));

    // Activate water pump if soil is dry
    if (soilMoisture > 650) {
        digitalWrite(WATER_PUMP_PIN, HIGH);  // Turn on water pump
        delay(1000);                         // Pump water for 1 second
        digitalWrite(WATER_PUMP_PIN, LOW);   // Turn off water pump
    }

    delay(2000);  // Wait before next loop iteration
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.45 Lesson 44: Bluetooth

This project provides a guide to develop a simple Bluetooth Low Energy (BLE) serial communication application using the ESP32 microcontroller. The ESP32 is a powerful microcontroller that integrates Wi-Fi and Bluetooth connectivity, making it an ideal candidate for developing wireless applications. BLE is a low-power wireless communication protocol that is designed for short-range communication. This document will cover the steps to set up the ESP32 to act as a BLE server and communicate with a BLE client over a serial connection.

About the Bluetooth Function

The ESP32 WROOM 32E is a module that integrates Wi-Fi and Bluetooth connectivity into a single chip. It supports Bluetooth Low Energy (BLE) and Classic Bluetooth protocols.

The module can be used as a Bluetooth client or server. As a Bluetooth client, the module can connect to other Bluetooth devices and exchange data with them. As a Bluetooth server, the module can provide services to other Bluetooth devices. s The ESP32 WROOM 32E supports various Bluetooth profiles, including the Generic Access Profile (GAP), Generic Attribute Profile (GATT), and Serial Port Profile (SPP). The SPP profile allows the module to emulate a serial port over Bluetooth, enabling serial communication with other Bluetooth devices.

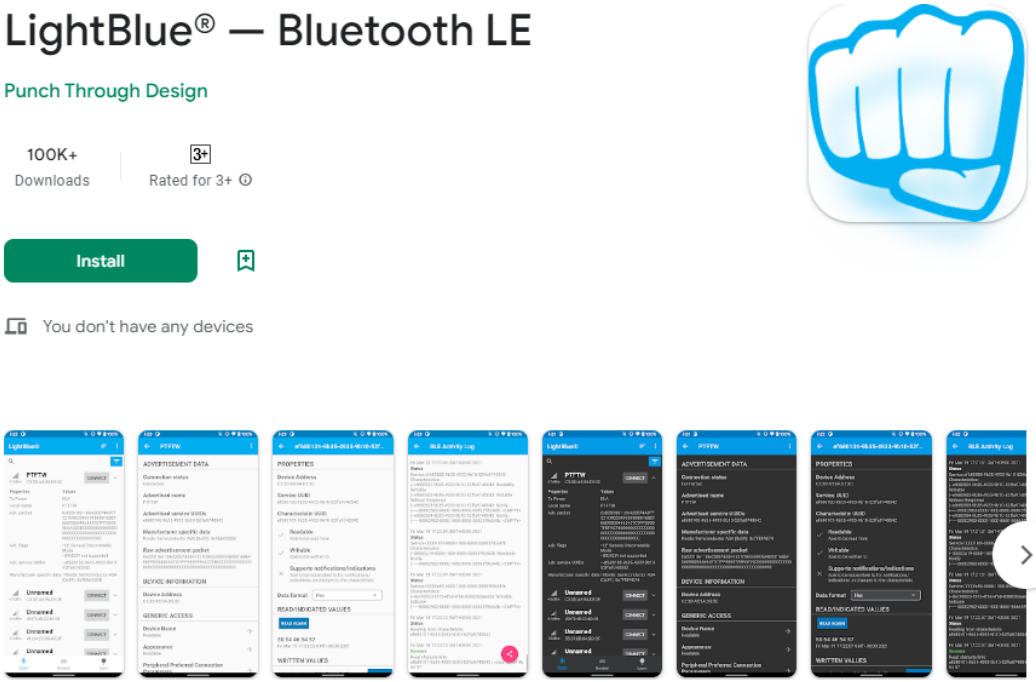
To use the Bluetooth function of the ESP32 WROOM 32E, you need to program it using an appropriate software development kit (SDK) or using the Arduino IDE with the ESP32 BLE library. The ESP32 BLE library provides a high-level interface for working with BLE. It includes examples that demonstrate how to use the module as a BLE client and server.

Overall, the Bluetooth function of the ESP32 WROOM 32E provides a convenient and low-power way to enable wireless communication in your projects.

Operation Steps

Here are the step-by-step instructions to set up Bluetooth communication between your ESP32 and mobile device using the LightBlue app:

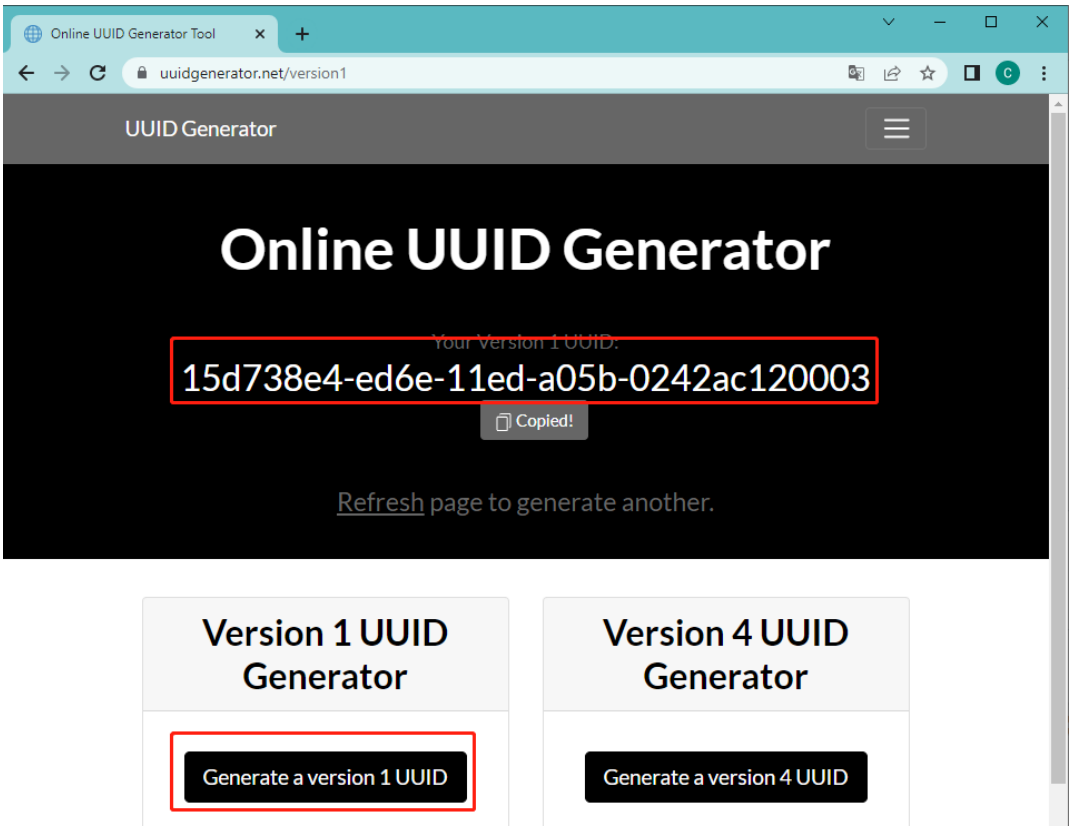
1. Download the LightBlue app from the **App Store** (for iOS) or **Google Play** (for Android).



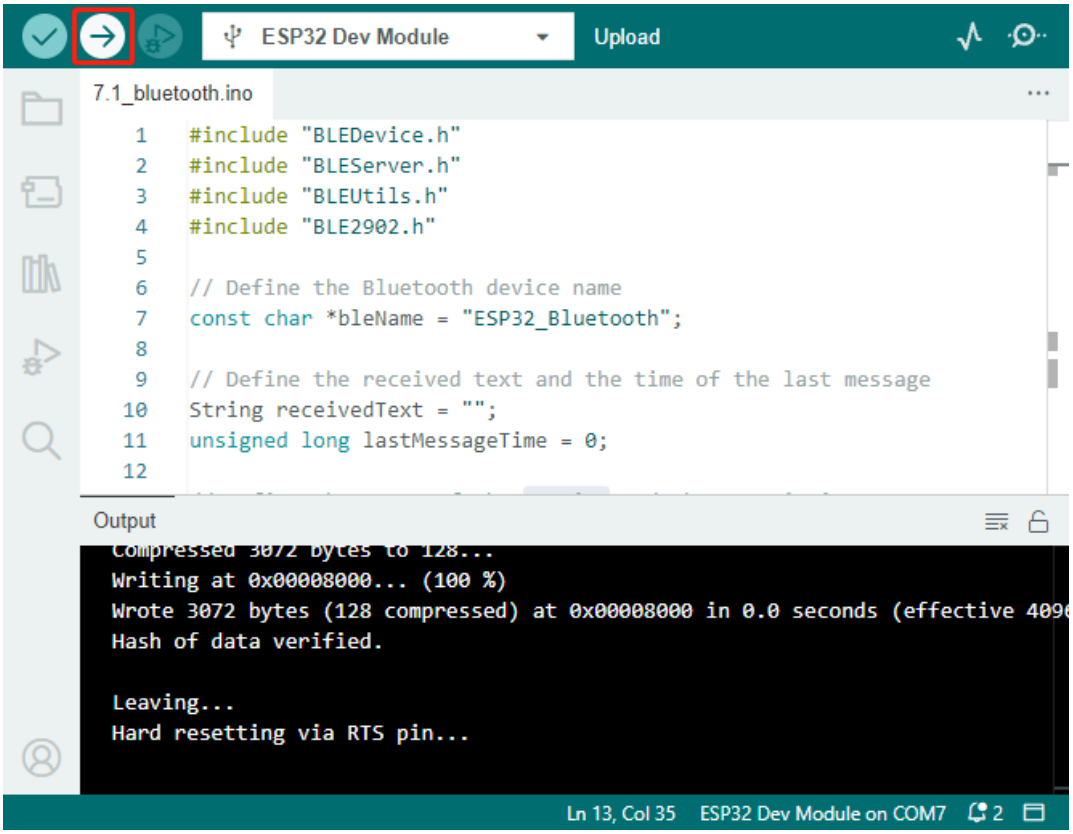
2. Open the `Lesson_44_Bluetooth.ino` file located in the `universal-maker-sensor-kit\esp32\Lesson_44_Bluetooth` directory, or copy the code into the Arduino IDE.

3. To avoid UUID conflicts, it is recommended to randomly generate three new UUIDs using the , and fill them in the following lines of code.

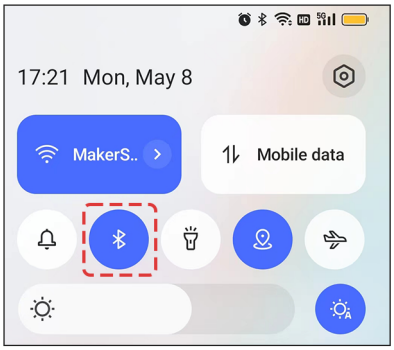
```
#define SERVICE_UUID           "your_service_uuid_here"
#define CHARACTERISTIC_UUID_RX "your_rx_characteristic_uuid_here"
#define CHARACTERISTIC_UUID_TX "your_tx_characteristic_uuid_here"
```



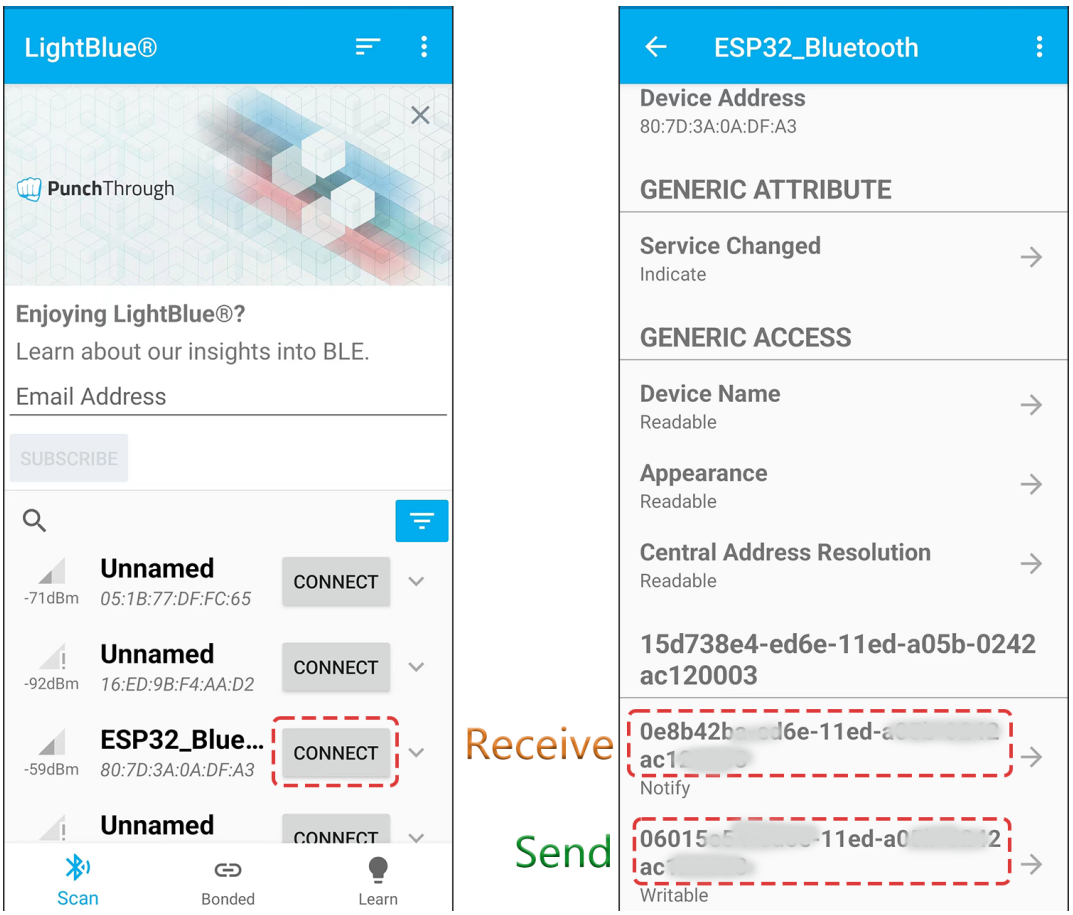
4. Select the correct board and port, then click the **Upload** button.



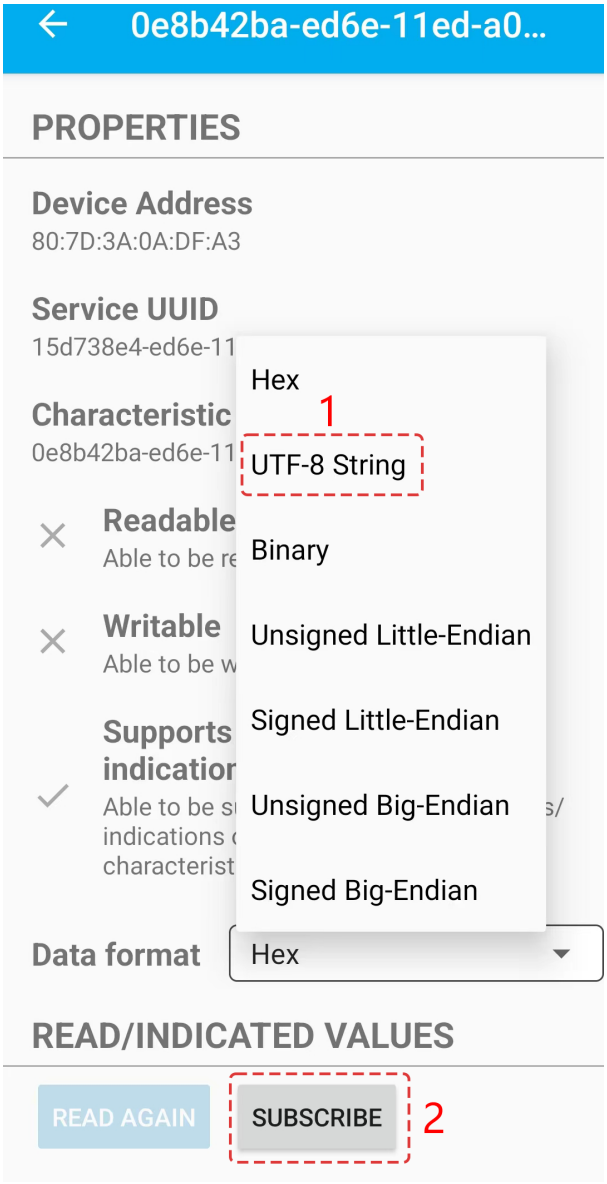
5. After the code has been successfully uploaded, turn on **Bluetooth** on your mobile device and open the **LightBlue** app.



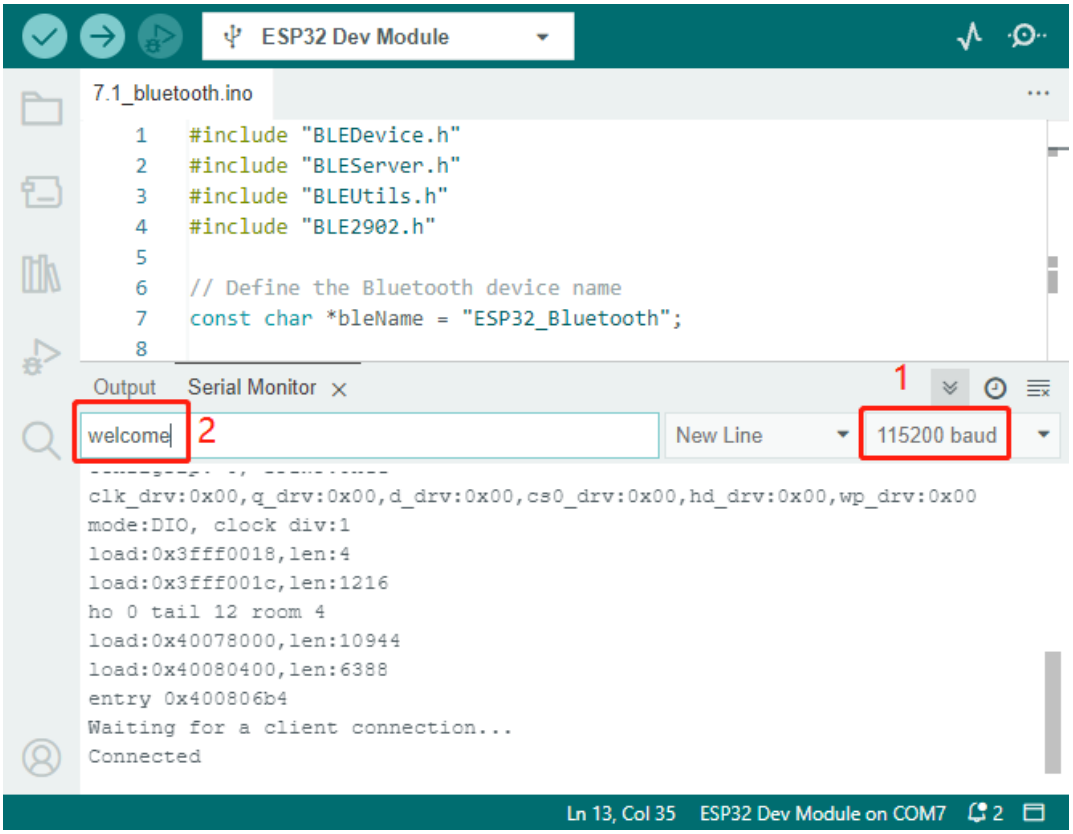
6. On the **Scan** page, find **ESP32-Bluetooth** and click **CONNECT**. If you don't see it, try refreshing the page a few times. When **"Connected to device!"** appears, the Bluetooth connection is successful. Scroll down to see the three UUIDs set in the code.



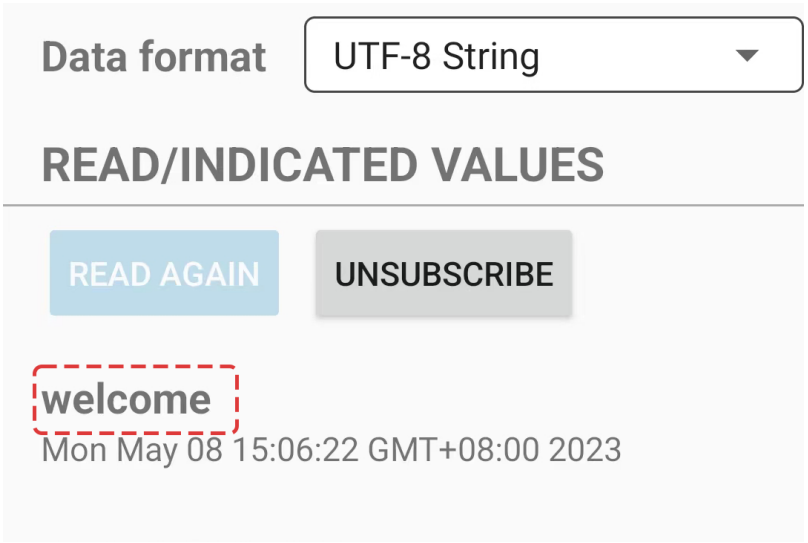
7. Click the **Receive** UUID. Select the appropriate data format in the box to the right of **Data Format**, such as "HEX" for hexadecimal, "UTF-8 String" for character, or "Binary" for binary, etc. Then click **SUBSCRIBE**.



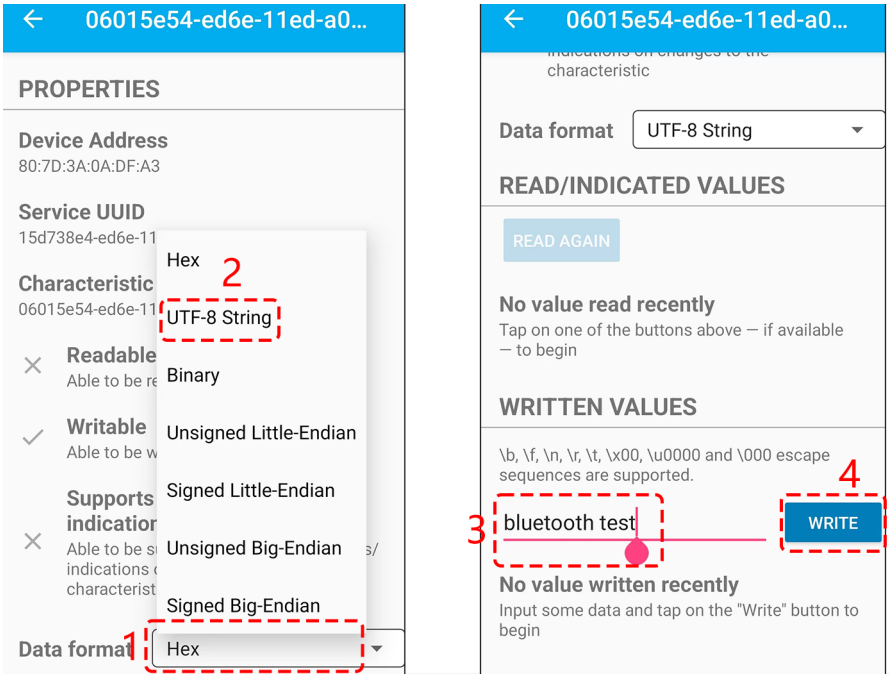
8. Go back to the Arduino IDE, open the Serial Monitor, set the baud rate to 115200, then type “welcome” and press Enter.



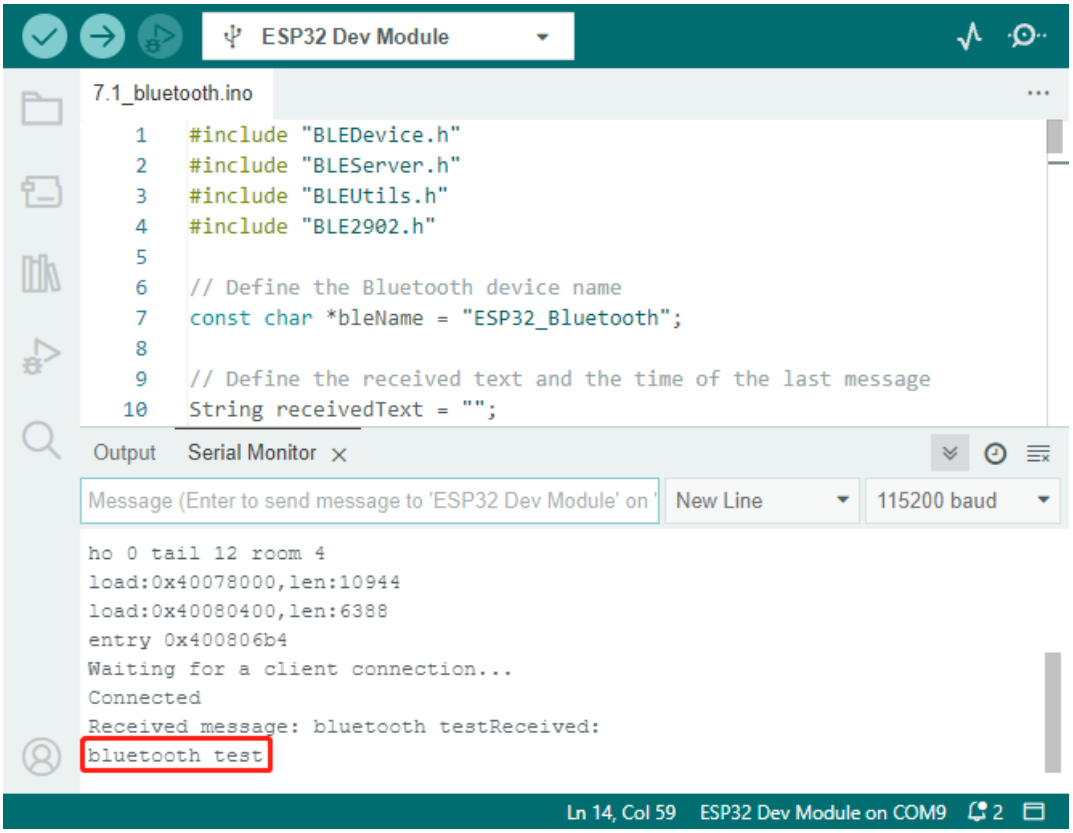
9. You should now see the “welcome” message in the LightBlue app.



10. To send information from the mobile device to the Serial Monitor, click the Send UUID, set the data format to “UTF-8 String”, and write a message.



11. You should see the message in the Serial Monitor.



How it works?

This Arduino code is written for the ESP32 microcontroller and sets it up to communicate with a Bluetooth Low Energy (BLE) device.

The following is a brief summary of the code:

- **Include necessary libraries**: The code begins by including necessary libraries for working with Bluetooth Low Energy (BLE) on the ESP32.

```
#include "BLEDevice.h"
#include "BLEServer.h"
#include "BLEUtils.h"
#include "BLE2902.h"
```

- **Global Variables**: The code defines a set of global variables including the Bluetooth device name (`bleName`), variables to keep track of received text and the time of the last message, UUIDs for the service and characteristics, and a `BLECharacteristic` object (`pCharacteristic`).

```
// Define the Bluetooth device name
const char *bleName = "ESP32_Bluetooth";

// Define the received text and the time of the last message
String receivedText = "";
unsigned long lastMessageTime = 0;

// Define the UUIDs of the service and characteristics
#define SERVICE_UUID           "your_service_uuid_here"
#define CHARACTERISTIC_UUID_RX "your_rx_characteristic_uuid_here"
#define CHARACTERISTIC_UUID_TX "your_tx_characteristic_uuid_here"

// Define the Bluetooth characteristic
BLECharacteristic *pCharacteristic;
```

- **Setup**: In the `setup()` function, the serial port is initialized with a baud rate of 115200 and the `setupBLE()` function is called to set up the Bluetooth BLE.

```
void setup() {
    Serial.begin(115200);  // Initialize the serial port
    setupBLE();            // Initialize the Bluetooth BLE
}
```

- **Main Loop**: In the `loop()` function, if a string was received over BLE (i.e., `receivedText` is not empty) and at least 1 second has passed since the last message, the code prints the received string to the serial monitor, sets the characteristic value to the received string, sends a notification, and then clears the received string. If data is available on the serial port, it reads the string until a newline character is encountered, sets the characteristic value to this string, and sends a notification.

```
void loop() {
    // When the received text is not empty and the time since the last␣
↪message is over 1 second
    // Send a notification and print the received text
    if (receivedText.length() > 0 && millis() - lastMessageTime > 1000) {
        Serial.print("Received message: ");
        Serial.println(receivedText);
        pCharacteristic->setValue(receivedText.c_str());
        pCharacteristic->notify();
        receivedText = "";
    }
```

(continues on next page)

(continued from previous page)

```
    // Read data from the serial port and send it to BLE characteristic
    if (Serial.available() > 0) {
        String str = Serial.readStringUntil('\n');
        const char *newValue = str.c_str();
        pCharacteristic->setValue(newValue);
        pCharacteristic->notify();
    }
}
```

- **Callbacks**: Two callback classes (`MyServerCallbacks` and `MyCharacteristicCallbacks`) are defined to handle events related to Bluetooth communication. `MyServerCallbacks` is used to handle events related to the connection state (connected or disconnected) of the BLE server. `MyCharacteristicCallbacks` is used to handle write events on the BLE characteristic, i.e., when a connected device sends a string to the ESP32 over BLE, it's captured and stored in `receivedText`, and the current time is recorded in `lastMessageTime`.

```
// Define the BLE server callbacks
class MyServerCallbacks : public BLEServerCallbacks {
    // Print the connection message when a client is connected
    void onConnect(BLEServer *pServer) {
    Serial.println("Connected");
    }
    // Print the disconnection message when a client is disconnected
    void onDisconnect(BLEServer *pServer) {
    Serial.println("Disconnected");
    }
};

// Define the BLE characteristic callbacks
class MyCharacteristicCallbacks : public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
        // When data is received, get the data and save it to receivedText,␣
↪and record the time
        std::string value = pCharacteristic->getValue();
        receivedText = String(value.c_str());
        lastMessageTime = millis();
        Serial.print("Received: ");
        Serial.println(receivedText);
    }
};
```

- **Setup BLE**: In the `setupBLE()` function, the BLE device and server are initialized, the server callbacks are set, the BLE service is created using the defined UUID, characteristics for sending notifications and receiving data are created and added to the service, and the characteristic callbacks are set. Finally, the service is started and the server begins advertising.

```
// Initialize the Bluetooth BLE
void setupBLE() {
    BLEDevice::init(bleName);                          // Initialize the BLE␣
↪device
    BLEServer *pServer = BLEDevice::createServer();  // Create the BLE␣
↪server
    // Print the error message if the BLE server creation fails
```

(continues on next page)

(continued from previous page)

```
    if (pServer == nullptr) {
        Serial.println("Error creating BLE server");
        return;
    }
    pServer->setCallbacks(new MyServerCallbacks());  // Set the BLE server␣
↪callbacks

    // Create the BLE service
    BLEService *pService = pServer->createService(SERVICE_UUID);
    // Print the error message if the BLE service creation fails
    if (pService == nullptr) {
        Serial.println("Error creating BLE service");
        return;
    }
    // Create the BLE characteristic for sending notifications
    pCharacteristic = pService->createCharacteristic(CHARACTERISTIC_UUID_TX,
↪ BLECharacteristic::PROPERTY_NOTIFY);
    pCharacteristic->addDecodeor(new BLE2902());  // Add the decodeor
    // Create the BLE characteristic for receiving data
    BLECharacteristic *pCharacteristicRX = pService->
↪createCharacteristic(CHARACTERISTIC_UUID_RX, BLECharacteristic::PROPERTY_
↪WRITE);
    pCharacteristicRX->setCallbacks(new MyCharacteristicCallbacks());  //␣
↪Set the BLE characteristic callbacks
    pService->start();                                                 //␣
↪Start the BLE service
    pServer->getAdvertising()->start();                                //␣
↪Start advertising
    Serial.println("Waiting for a client connection...");              //␣
↪Wait for a client connection
}
```

Please note that this code allows for bidirectional communication - it can send and receive data via BLE. However, to interact with specific hardware like turning on/off an LED, additional code should be added to process the received strings and act accordingly.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.46 Lesson 45: Bluetooth Control RGB LED

This project is an extension of a previous project(*Lesson 44: Bluetooth*), adding RGB LED configurations and custom commands such as "led_off", "red", "green", etc. These commands allow the RGB LED to be controlled by sending commands from a mobile device using LightBlue.

Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

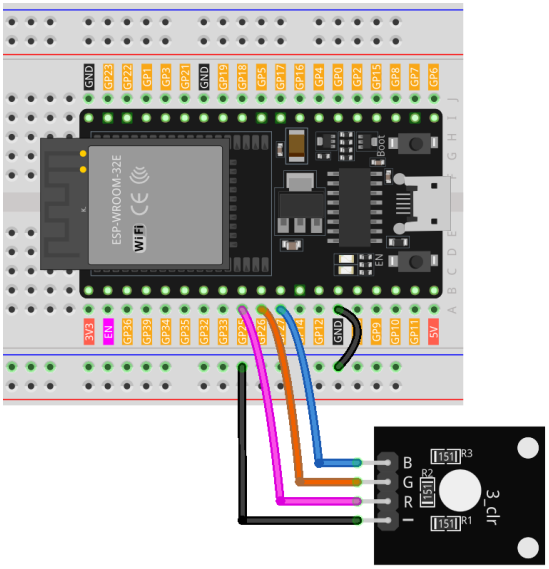
Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

COMPONENT INTRODUCTION	PURCHASE LINK
ESP32 & Development Board	
Breadboard	
RGB LED Module	-

Operation Steps

1. Build the circuit.

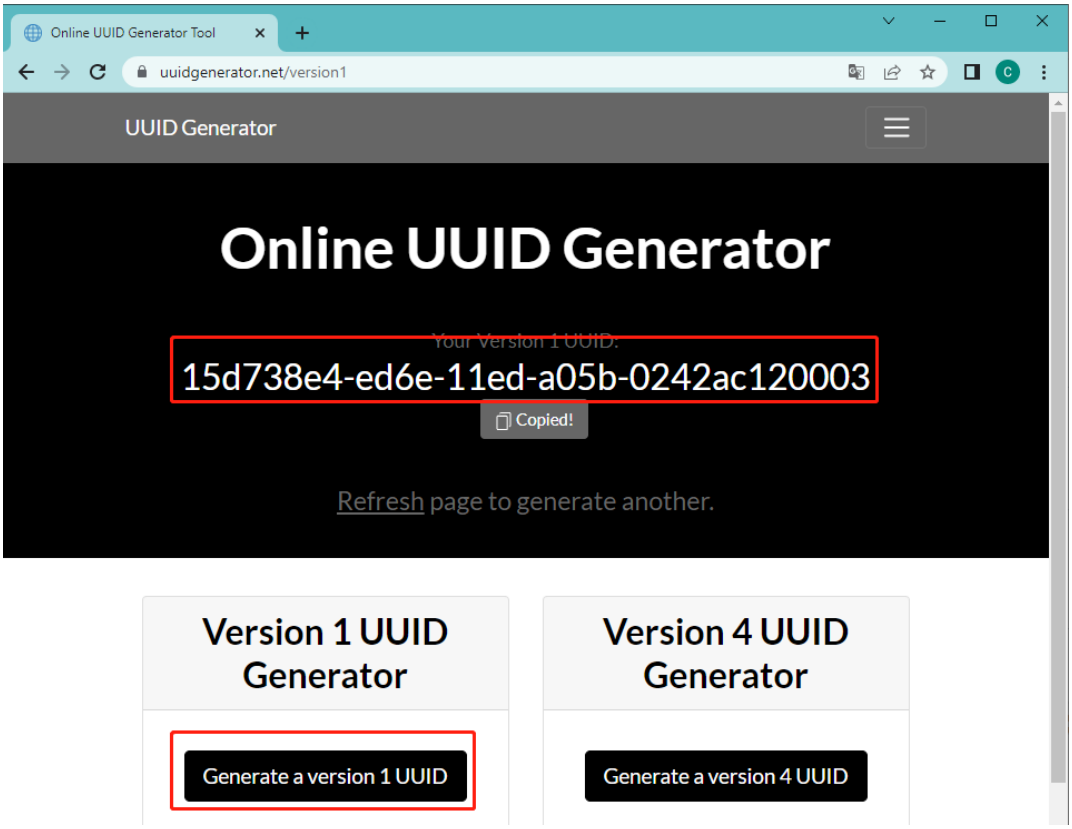


2. Open the `Lesson_45_Bluetooth_RGB.ino` file located in the `universal-maker-sensor-kit\esp32\Lesson_45_Bluetooth_RGB` directory, or copy the code into the Arduino IDE.

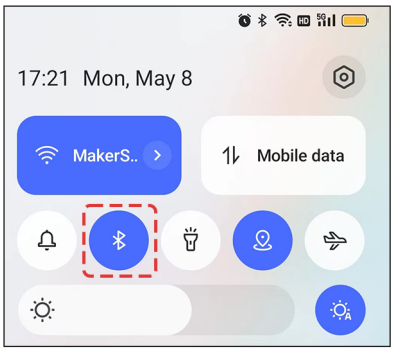
3. To avoid UUID conflicts, it is recommended to randomly generate three new UUIDs using the provided by the Bluetooth SIG, and fill them in the following lines of code.

Note: If you have already generated three new UUIDs in the *Lesson 44: Bluetooth* project, then you can continue using them.

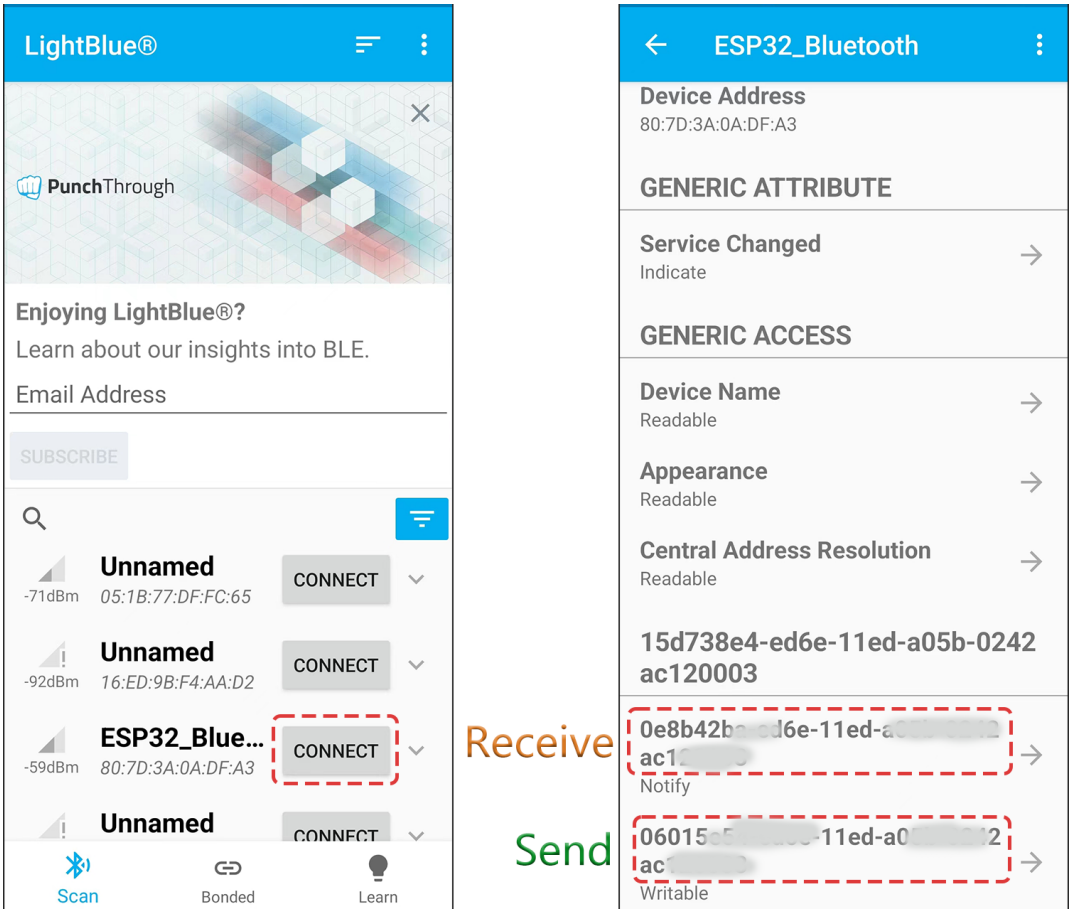
```
#define SERVICE_UUID           "your_service_uuid_here"
#define CHARACTERISTIC_UUID_RX "your_rx_characteristic_uuid_here"
#define CHARACTERISTIC_UUID_TX "your_tx_characteristic_uuid_here"
```



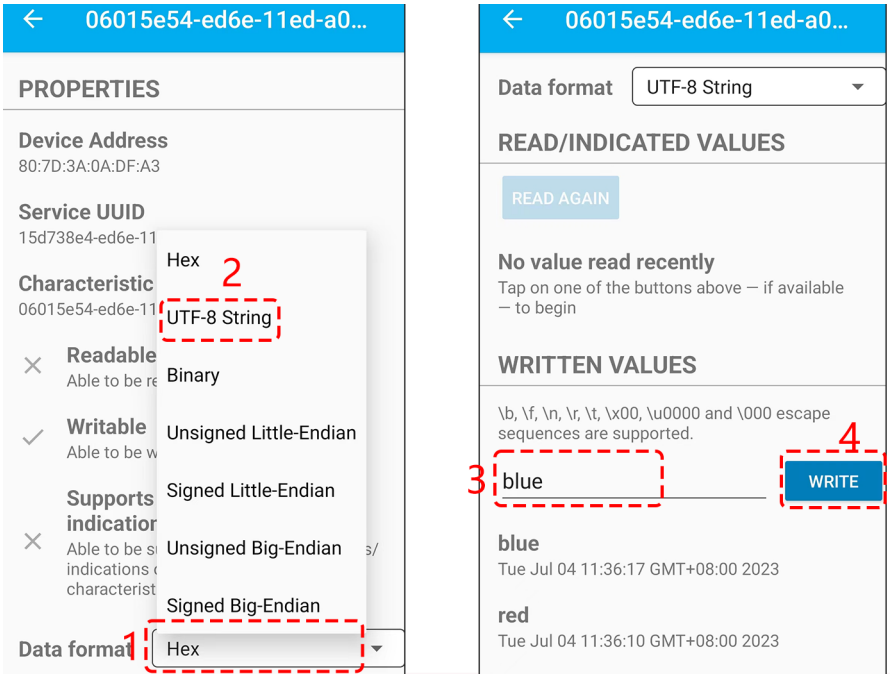
4. Select the correct board and port, then click the **Upload** button.
5. After the code has been successfully uploaded, turn on **Bluetooth** on your mobile device and open the **LightBlue** app.



6. On the **Scan** page, find **ESP32-Bluetooth** and click **CONNECT**. If you don't see it, try refreshing the page a few times. When **"Connected to device!"** appears, the Bluetooth connection is successful. Scroll down to see the three UUIDs set in the code.



7. Tap the Send UUID, then set the data format to “UTF-8 String”. Now you can write these commands: “led_off”, “red”, “green”, “blue”, “yellow”, and “purple” to see if the RGB LED responds to these instructions.



How it works?

This code is an extension of a previous project(*Lesson 44: Bluetooth*), adding RGB LED configurations and custom commands such as "led_off", "red", "green", etc. These commands allow the RGB LED to be controlled by sending commands from a mobile device using LightBlue.

Let's break down the code step by step:

- Add new global variables for the RGB LED pins, PWM channels, frequency, and resolution.

```
...

// Define RGB LED pins
const int redPin = 27;
const int greenPin = 26;
const int bluePin = 25;

// Define PWM channels
const int redChannel = 0;
const int greenChannel = 1;
const int blueChannel = 2;

...
```

- Within the `setup()` function, the PWM channels are initialized with the predefined frequency and resolution. The RGB LED pins are then attached to their respective PWM channels.

```
void setup() {
    ...

    // Set up PWM channels
    ledcSetup(redChannel, freq, resolution);
    ledcSetup(greenChannel, freq, resolution);
    ledcSetup(blueChannel, freq, resolution);

    // Attach pins to corresponding PWM channels
    ledcAttachPin(redPin, redChannel);
    ledcAttachPin(greenPin, greenChannel);
    ledcAttachPin(bluePin, blueChannel);

}
```

- Modify the `onWrite` method in the `MyCharacteristicCallbacks` class. This function listens for data coming from the Bluetooth connection. Based on the received string (like `"led_off"`, `"red"`, `"green"`, etc.), it controls the RGB LED.

```
// Define the BLE characteristic callbacks
class MyCharacteristicCallbacks : public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
        std::string value = pCharacteristic->getValue();
        if (value == "led_off") {
            setColor(0, 0, 0); // turn the RGB LED off
            Serial.println("RGB LED turned off");
        } else if (value == "red") {
            setColor(255, 0, 0); // Red
            Serial.println("red");
        }
```

(continues on next page)

(continued from previous page)

```
        else if (value == "green") {
            setColor(0, 255, 0); // green
            Serial.println("green");
        }
        else if (value == "blue") {
            setColor(0, 0, 255); // blue
            Serial.println("blue");
        }
        else if (value == "yellow") {
            setColor(255, 150, 0); // yellow
            Serial.println("yellow");
        }
        else if (value == "purple") {
            setColor(80, 0, 80); // purple
            Serial.println("purple");
        }
    }
};
```

- Finally, a function is added to set the RGB LED color.

```
void setColor(int red, int green, int blue) {
    // For common-anode RGB LEDs, use 255 minus the color value
    ledcWrite(redChannel, red);
    ledcWrite(greenChannel, green);
    ledcWrite(blueChannel, blue);
}
```

In summary, this script enables a remote control interaction model, where the ESP32 operates as a Bluetooth Low Energy (BLE) server.

The connected BLE client (like a smartphone) can send string commands to change the color of an RGB LED. The ESP32 also gives feedback to the client by sending back the string received, allowing the client to know what operation was performed.

IoT Project

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.47 Lesson 46: Real-time Weather From @OpenWeatherMap

The IoT Open Weather Display project utilizes the ESP32 board and an I2C LCD1602 module to create a weather information display that retrieves data from the OpenWeatherMap API.

This project serves as an excellent introduction to working with APIs, Wi-Fi connectivity, and data display on an LCD module using the ESP32 board. With the IoT Open Weather Display, you can conveniently access real-time weather updates at a glance, making it an ideal solution for home or office environments.

Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

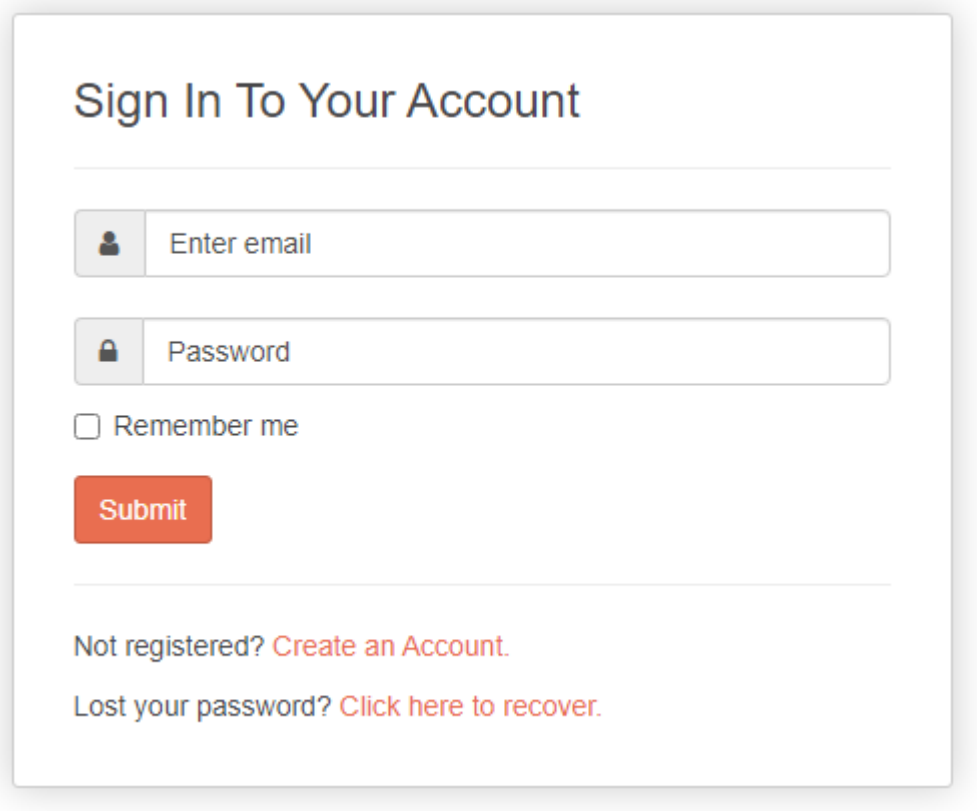
You can also buy them separately from the links below. .. list-table:

```
:widths: 30 20
:header-rows: 1

*   - COMPONENT INTRODUCTION
    - PURCHASE LINK

*   - ESP32 & Development Board
    - |link_esp32_camera_pro_kit_buy|
*   - :ref:`cpn_i2c_lcd1602`
    - |link_i2clcd1602_buy|
```

Get OpenWeather API keys

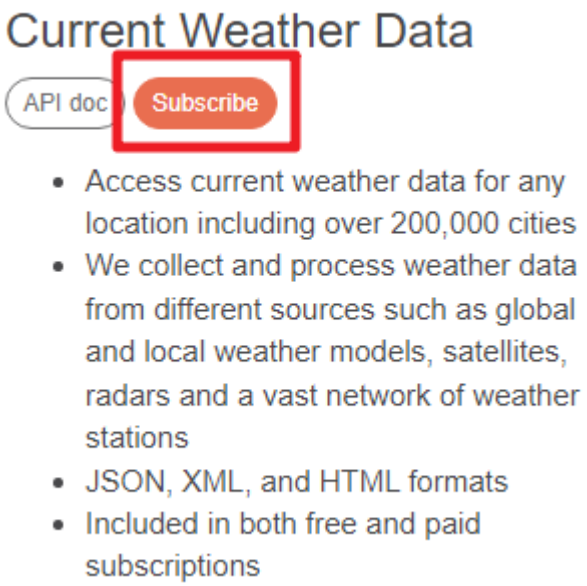
is an online service, owned by OpenWeather Ltd, that provides global weather data via API, including current weather data, forecasts, nowcasts and historical weather data for any geographical location.

1. Visit to log in/create an account.



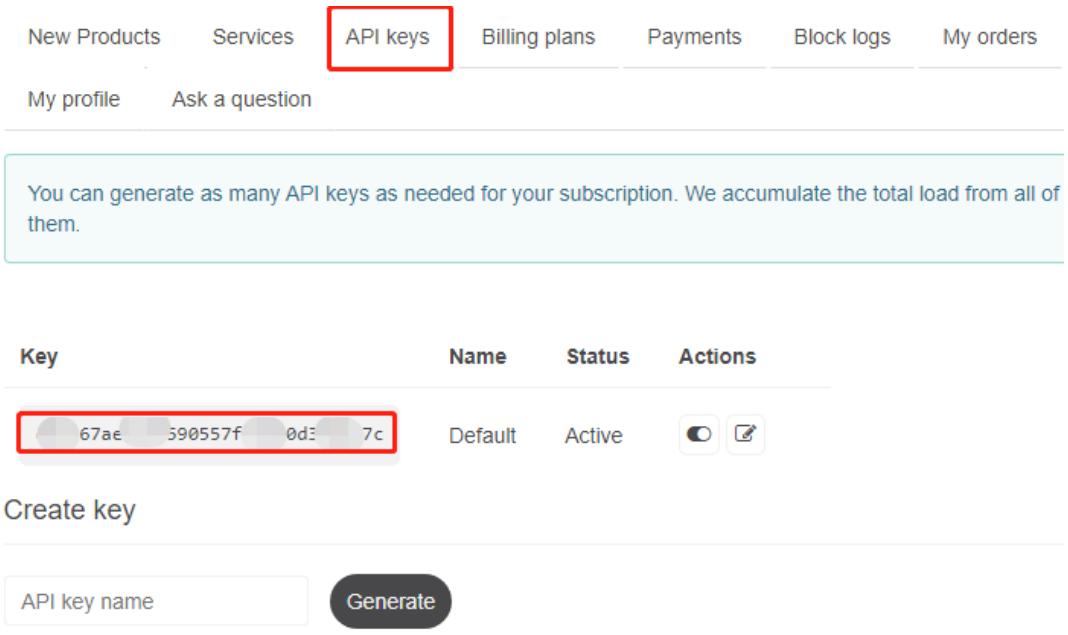
2. Click into the API page from the navigation bar.

3. Find **Current Weather Data** and click Subscribe.



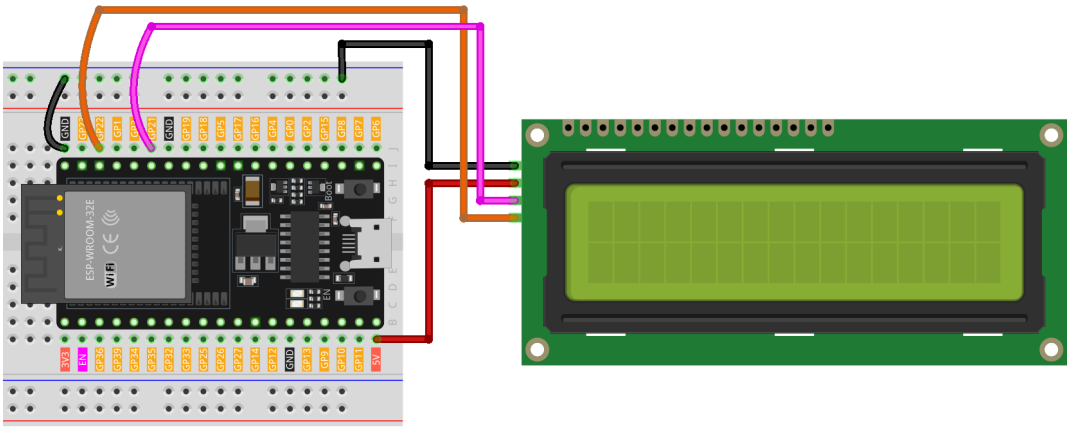
4. Under **Current weather and forecasts collection** , subscribe to the appropriate service. In our project, Free is good enough.

5. Copy the Key from the **API keys** page.



Complete Your Device

1. Build the circuit.



2. Open the code.

    - Open the `Lesson_46_OpenWeatherMap.ino` file located in the `universal-maker-sensor-kit\esp32\Lesson_46_OpenWeatherMap` directory, or copy the code into the Arduino IDE.
    - After selecting the board (ESP32 Dev Module) and the appropriate port, click the **Upload** button.
    - *Always displaying "Unknown COMxx"?*
    - The `LiquidCrystal I2C` and `Arduino_JSON` libraries are used here, you can install them from the **Library Manager**.

3. Locate the following lines and modify them with your <SSID> and <PASSWORD>.

```
// Replace the next variables with your SSID/Password combination
const char* ssid = "<SSID>";
const char* password = "<PASSWORD>";
```

4. Fill in the API keys you copied earlier into `openWeatherMapApiKey`.

```
// Your Domain name with URL path or IP address with path
String openWeatherMapApiKey = "<openWeatherMapApiKey>";
```

5. Replace with your country code and city.

```
// Replace with your country code and city
// Fine the country code by https://openweathermap.org/find
String city = "<CITY>";
String countryCode = "<COUNTRY CODE>";
```

6. After the code runs, you will see the time and weather information of your location on the I2C LCD1602.

Note: When the code is running, if the screen is blank, you can turn the potentiometer on the back of the module to increase the contrast.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.

- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.48 Lesson 47: IoT Communication with MQTT

This project focuses on utilizing MQTT, a popular communication protocol in the Internet of Things (IoT) domain. MQTT enables IoT devices to exchange data using a publish/subscribe model, where devices communicate through topics.

In this project, we explore the implementation of MQTT by building a circuit that includes an LED, a button, and a thermistor. The ESP32-WROOM-32E microcontroller is used to establish a connection to WiFi and communicate with an MQTT broker. The code allows the microcontroller to subscribe to specific topics, receive messages, and control the LED based on the received information. Additionally, the project demonstrates publishing temperature data from the thermistor to a designated topic when the button is pressed.

Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

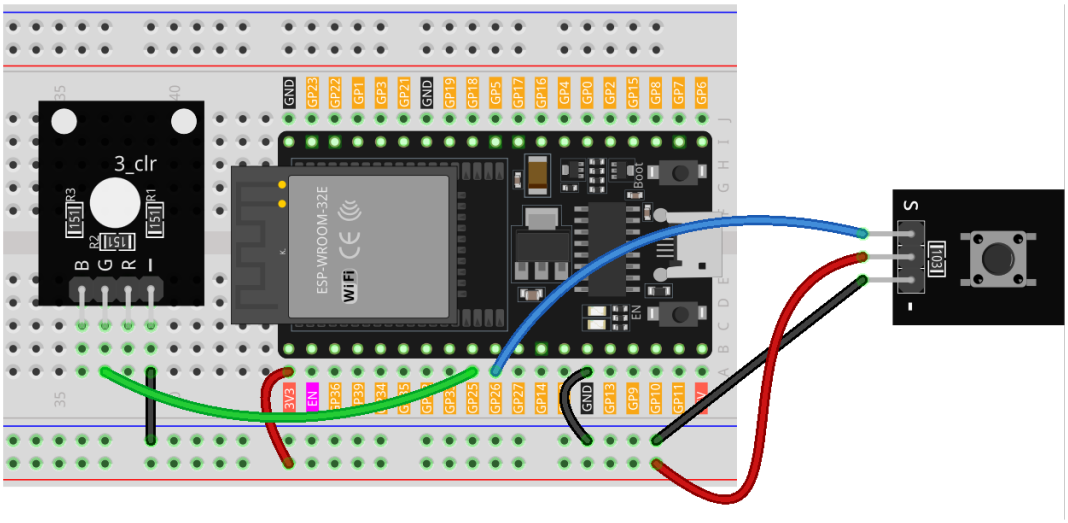
You can also buy them separately from the links below.

COMPONENT INTRODUCTION	PURCHASE LINK
ESP32 & Development Board	
Button Module	-
Breadboard	
RGB LED Module	-

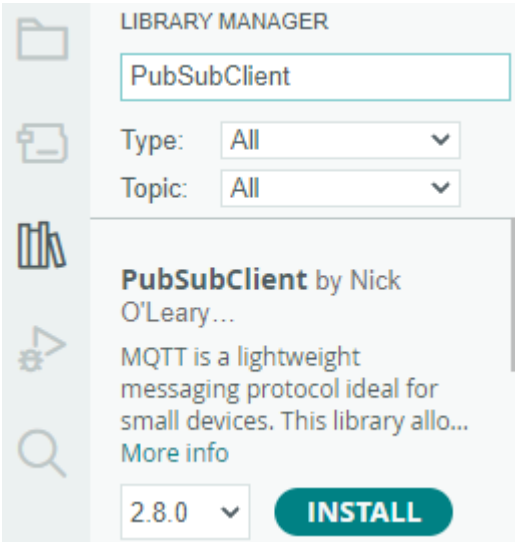
Code Upload

1. Build the circuit.

> **Note:** When establishing a connection to WiFi, only the 36, 39, 34, 35, 32, 33 pins can be employed for analog reading. Please ensure the thermistor is connected to these designated pins.



2. Then, connect ESP32-WROOM-32E to the computer using the USB cable.
3. Open the code.
    - Open the `Lesson_47_MQTT.ino` file located in the `universal-maker-sensor-kit\esp32\Lesson_47_MQTT` directory, or copy the code into the Arduino IDE.
    - After selecting the board (ESP32 Dev Module) and the appropriate port, click the **Upload** button.
    - *Always displaying "Unknown COMxx"?*
    - The `PubSubClient` library is used here, you can install it from the **Library Manager**.



4. Locate the following lines and modify them with your <SSID> and <PASSWORD>.

```
// Replace the next variables with your SSID/Password combination
const char* ssid = "<SSID>";
const char* password = "<PASSWORD>";
```

5. Find the next line and modify your `unique_identifier`. Guarantee that your `unique_identifier` is truly unique as any IDs that are identical trying to log in to the same MQTT Broker may result in a login failure.

```
// Add your MQTT Broker address, example:
const char* mqtt_server = "broker.hivemq.com";
const char* unique_identifier = "sunfounder-client-sdgvsda";
```

Topic Subscription

1. To avoid interference from messages sent by other participants, you can set it as an obscure or uncommon string. Simply replace the current topic `SF/LED` with your desired topic name.

> **Note:** You have the freedom to set the Topic as any character you desire. Any MQTT device that has subscribed to the identical Topic will be able to receive the same message. You can also simultaneously subscribe to multiple Topics.

```
void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect(unique_identifier)) {
            Serial.println("connected");
            // Subscribe
            client.subscribe("SF/LED");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}
```

2. Modify the functionality to respond to the subscribed topic. In the provided code, if a message is received on the topic `SF/LED`, it checks whether the message is `on` or `off`. Depending on the received message, it changes the output state to control the LED's on or off status.

> **Note:** You can modify it for any topic you are subscribed to, and you can write multiple if statements to respond to multiple topics.

```
void callback(char* topic, byte* message, unsigned int length) {
    Serial.print("Message arrived on topic: ");
    Serial.print(topic);
    Serial.print(". Message: ");
    String messageTemp;

    for (int i = 0; i < length; i++) {
        Serial.print((char)message[i]);
        messageTemp += (char)message[i];
    }
    Serial.println();
```

(continues on next page)

(continued from previous page)

```
    // If a message is received on the topic "SF/LED", you check if the␣
↪message is either "on" or "off".
    // Changes the output state according to the message
    if (String(topic) == "SF/LED") {
        Serial.print("Changing state to ");
        if (messageTemp == "on") {
            Serial.println("on");
            digitalWrite(ledPin, HIGH);
        } else if (messageTemp == "off") {
            Serial.println("off");
            digitalWrite(ledPin, LOW);
        }
    }
}
```

3. After selecting the correct board (ESP32 Dev Module) and port, click the **Upload** button.
4. Open the serial monitor and if the following information is printed, it indicates a successful connection to the MQTT server.

```
WiFi connected
IP address:
192.168.18.77
Attempting MQTT connection...connected
```

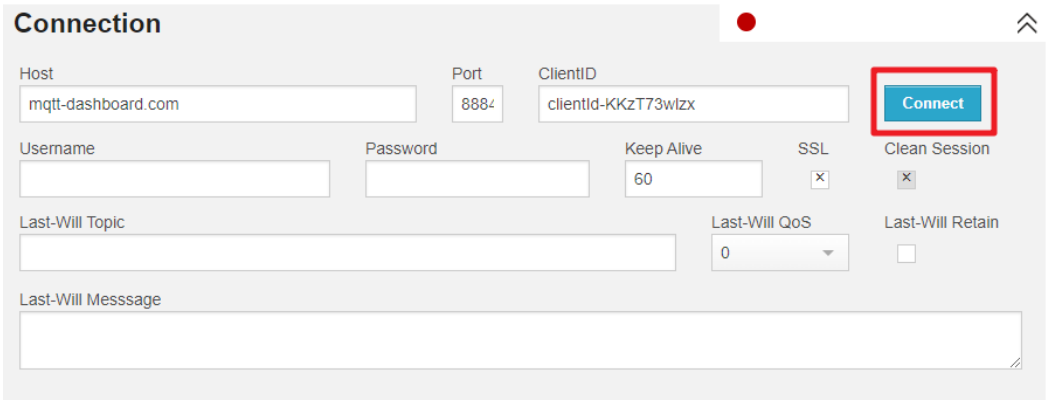
Message Publication via HiveMQ

HiveMQ is a messaging platform that functions as an MQTT broker, facilitating fast, efficient, and reliable data transfer to IoT devices.

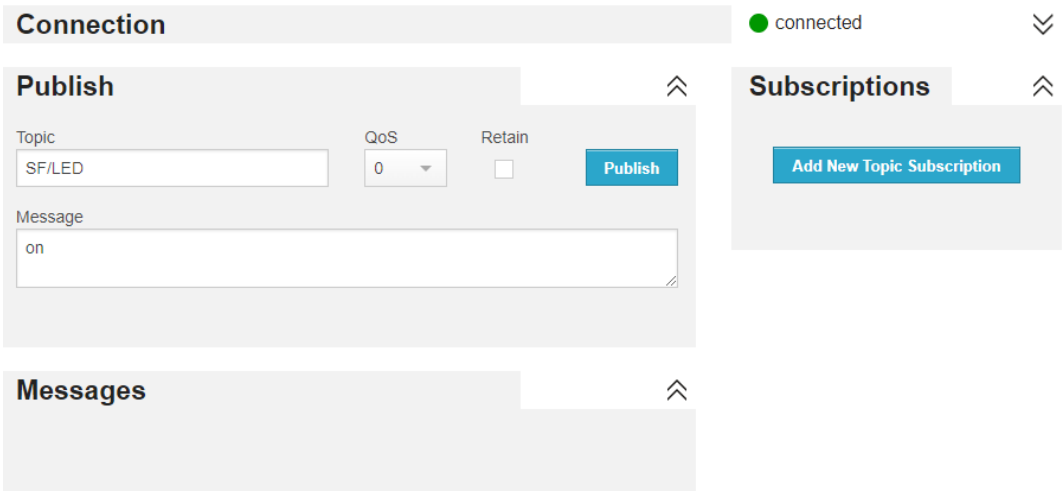
Our code specifically utilizes the MQTT broker provided by HiveMQ. We have included the address of the HiveMQ MQTT broker in the code as follows:

```
// Add your MQTT Broker address, example:
const char* mqtt_server = "broker.hivemq.com";
```

1. At present, open the in your web browser.
2. Connect the client to the default public proxy.



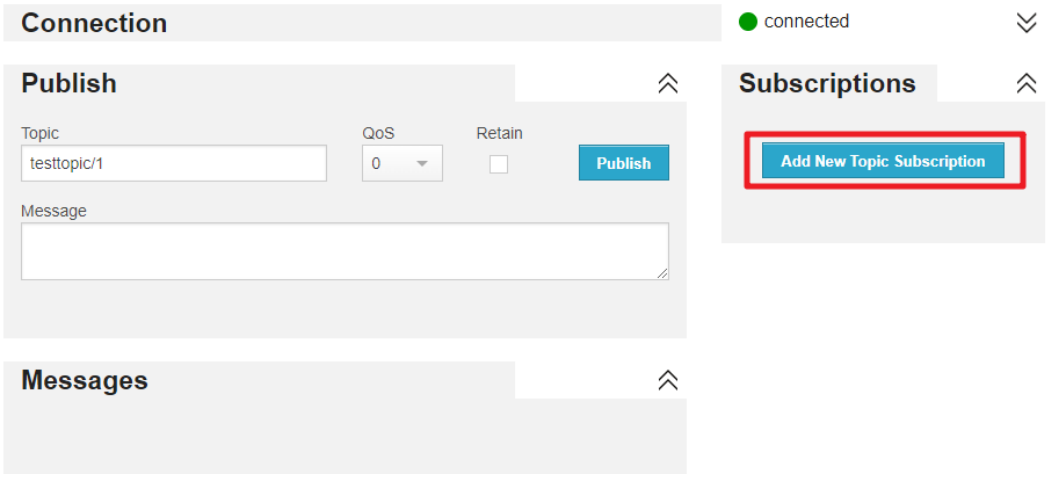
3. Publish a message in the Topic you have subscribed to. In this project, you can publish `on` or `off` to control your LED.



Message Publication to MQTT

We can also utilize the code to publish information to the Topic. In this demonstration, we have coded a feature that sends the simple message to the Topic when you press the button.

1. Click on **Add New Topic Subscription**.



2. Fill in the topics you desire to follow and click **Subscribe**. In the code, we send message to the topic `SF/TEMP`.

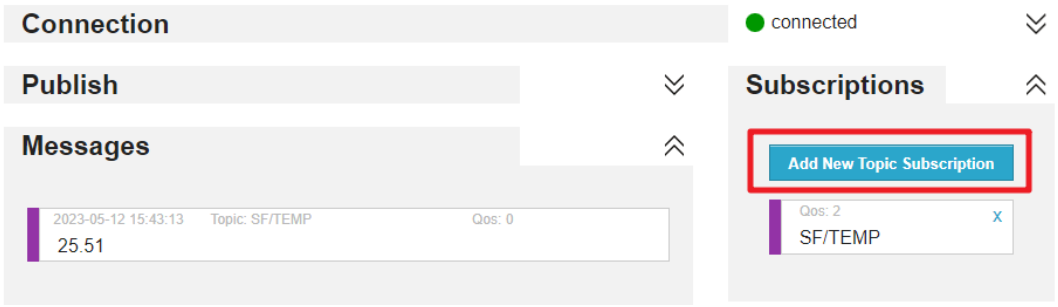
```
void loop() {
    if (!client.connected()) {
        reconnect();
    }
    client.loop();

    // if the button pressed, publish the temperature to topic "SF/TEMP"
    if (digitalRead(buttonPin)) {
            long now = millis();
            if (now - lastMsg > 5000) {
            lastMsg = now;
            char tempString[8];
            strcpy(tempString,"hello");
            client.publish("SF/TEMP", tempString);
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```

3. Hence, we can monitor this Topic on HiveMQ, allowing us to view the information you have published.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.49 Lesson 48: Temperature and Humidity Monitoring with Adafruit IO

In this project, we will guide you on how to use a popular IoT platform. There are many free (or low-cost) platforms available online for programming enthusiasts. Some examples are Adafruit IO, Blynk, Arduino Cloud, ThingSpeak, and so on. The usage of these platforms is quite similar. Here, we will be focusing on Adafruit IO.

We will write an Arduino program that uses the DHT11 sensor to send temperature and humidity readings to Adafruit IO's dashboard. You can also control an LED on the circuit through a switch on the dashboard.

Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

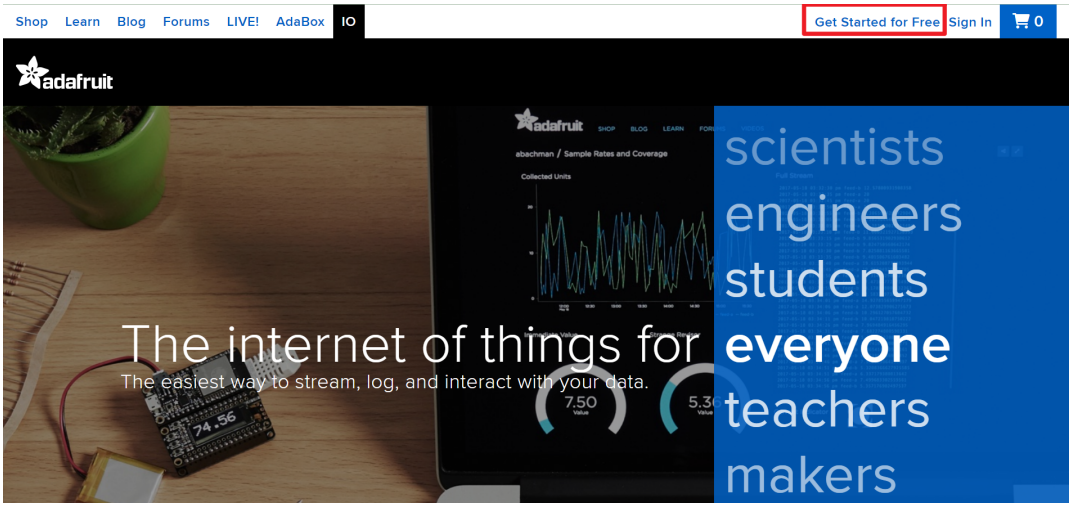
Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

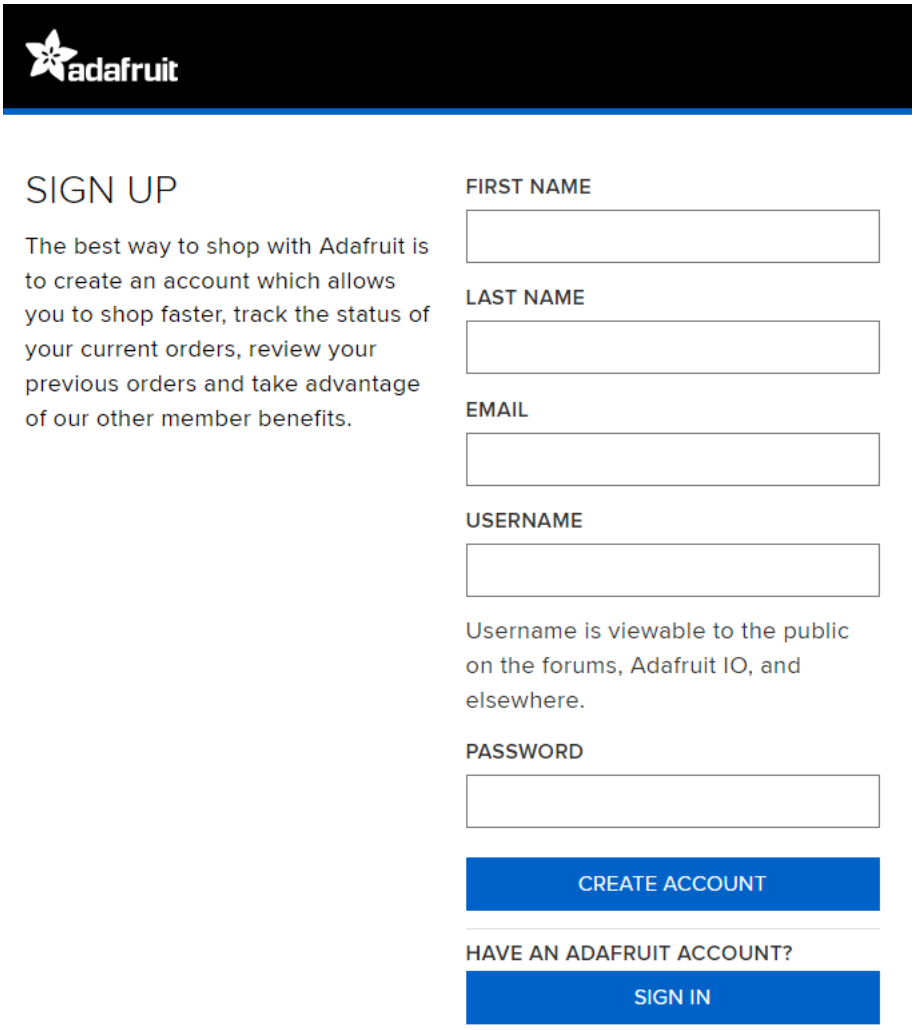
COMPONENT INTRODUCTION	PURCHASE LINK
ESP32 & Development Board	
RGB LED Module	-
Temperature and Humidity Sensor Module (DHT11)	

Setting up the Dashboard

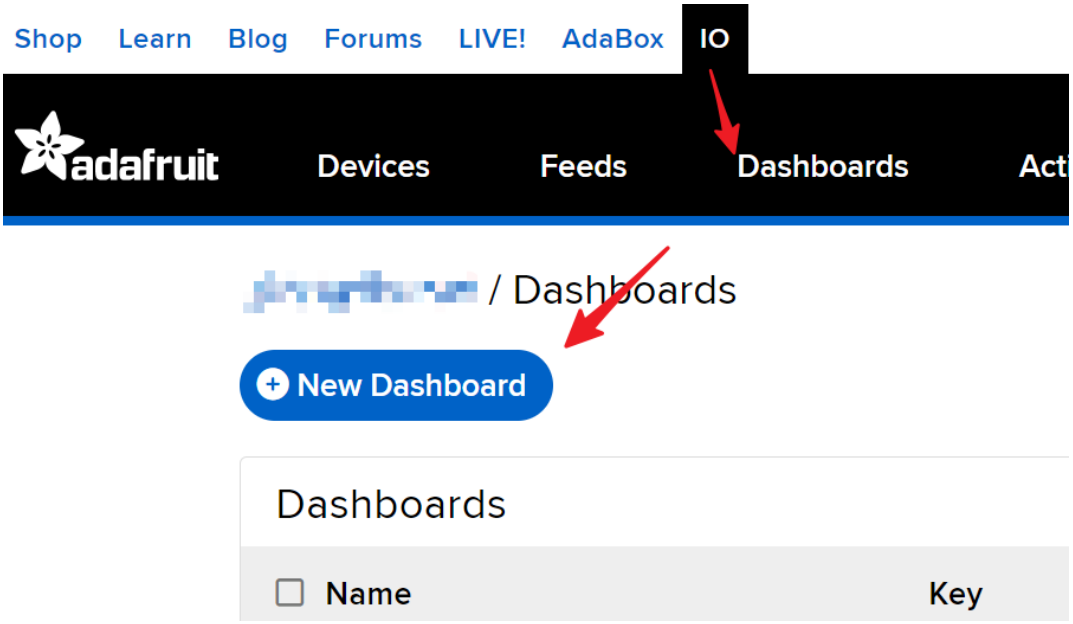
1. Visit , then click on **Start for free** to create a free account.



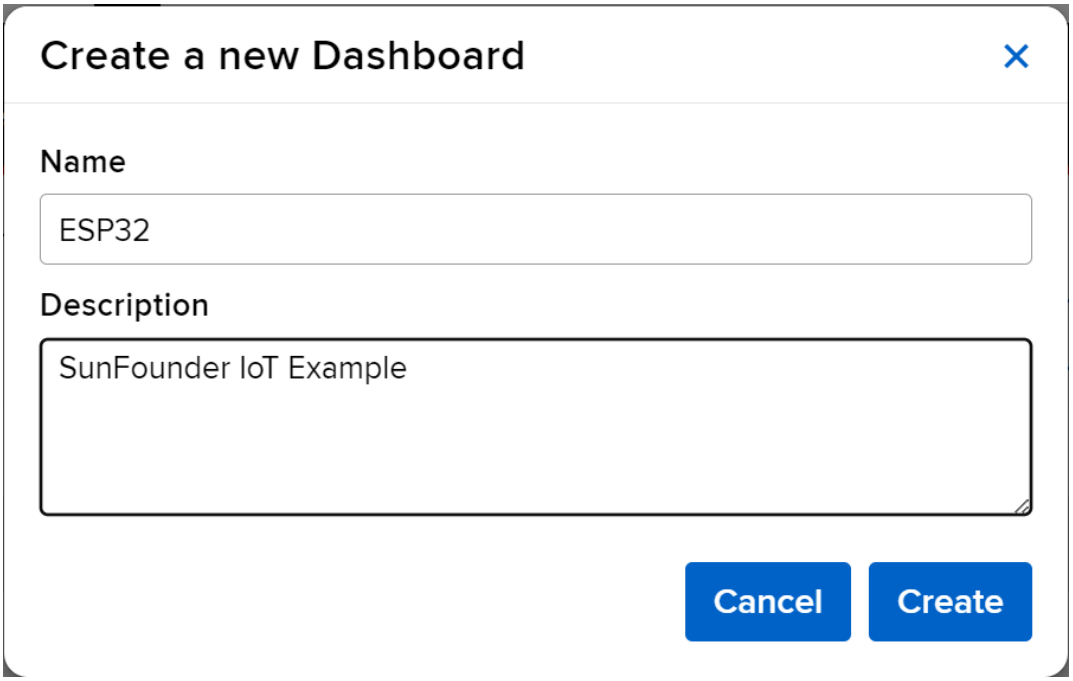
2. Fill out the form to create an account.



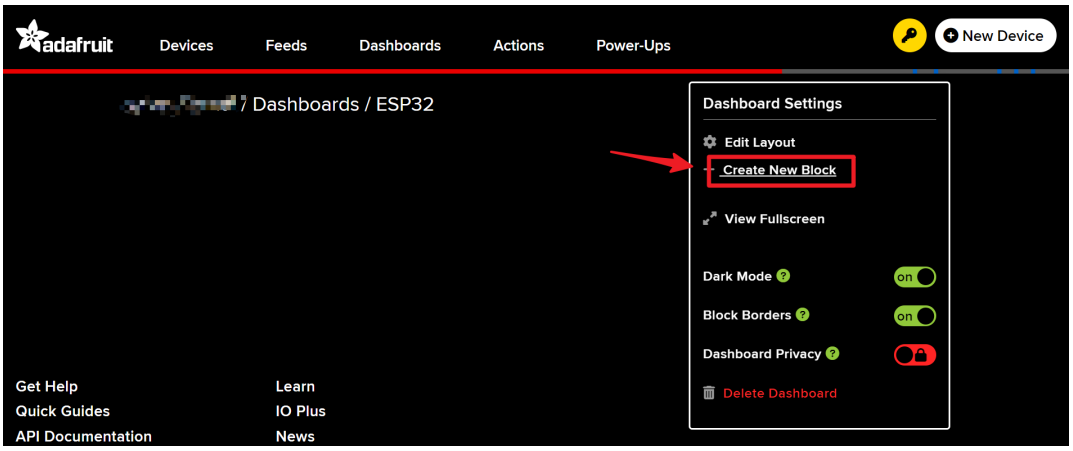
3. After creating an Adafruit account, you'll need to reopen Adafruit io. Click on the **Dashboards**, then click on **New Dashboard**.



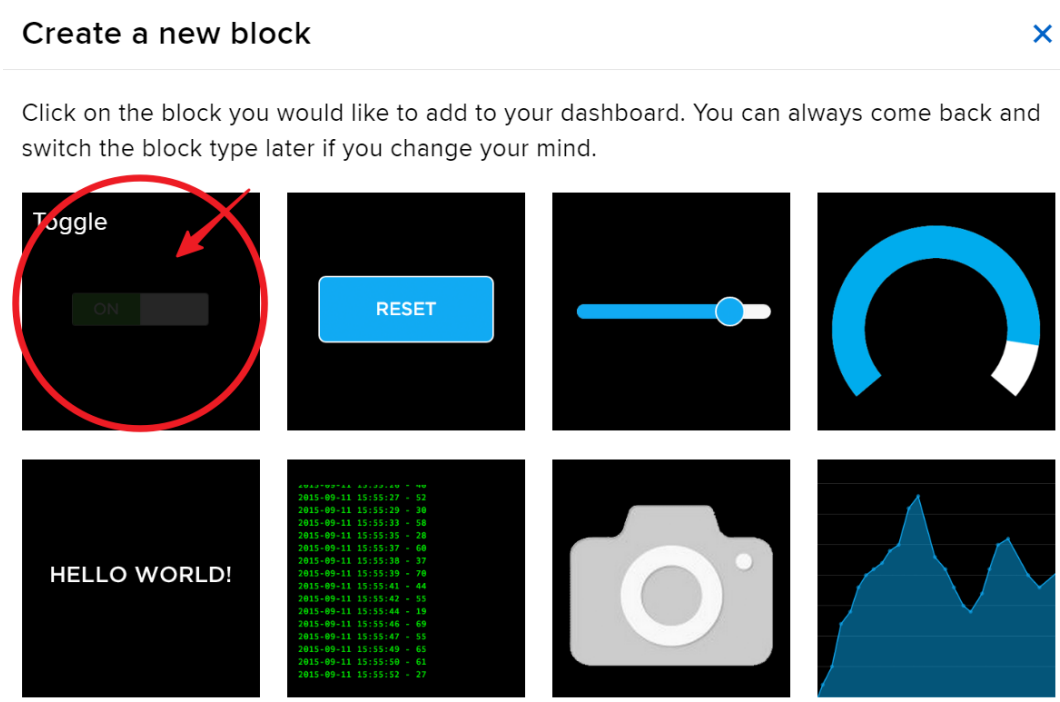
4. Create a **New Dashboard**.



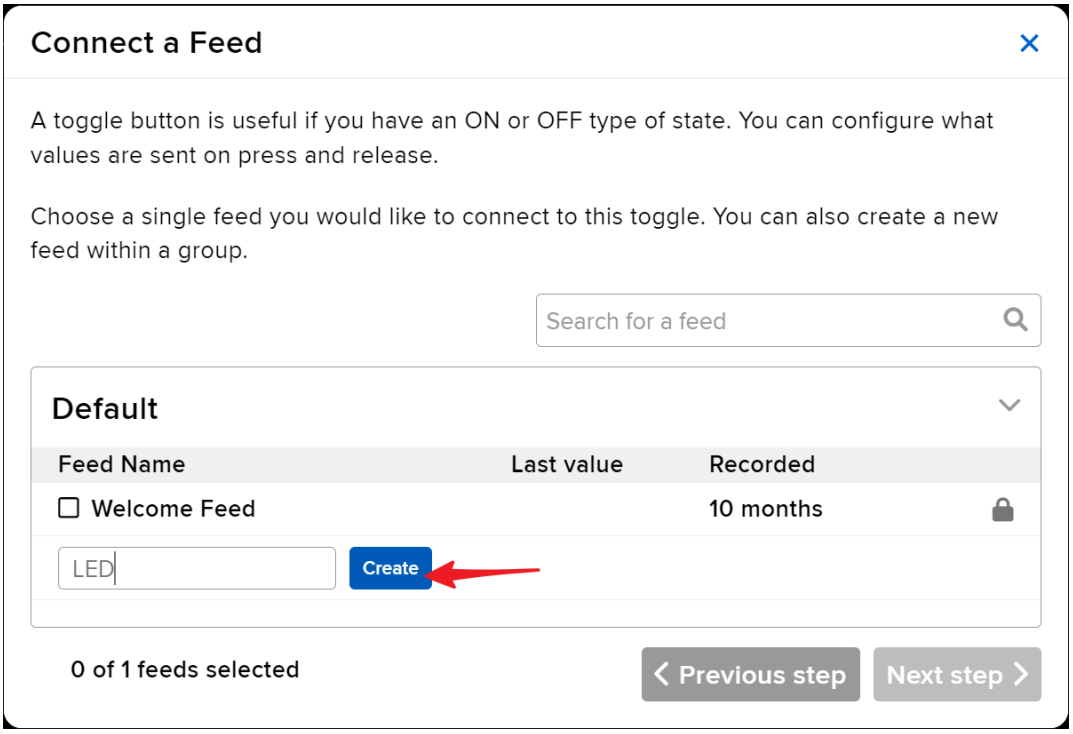
5. Enter the newly created **Dashboard** and create a new block.



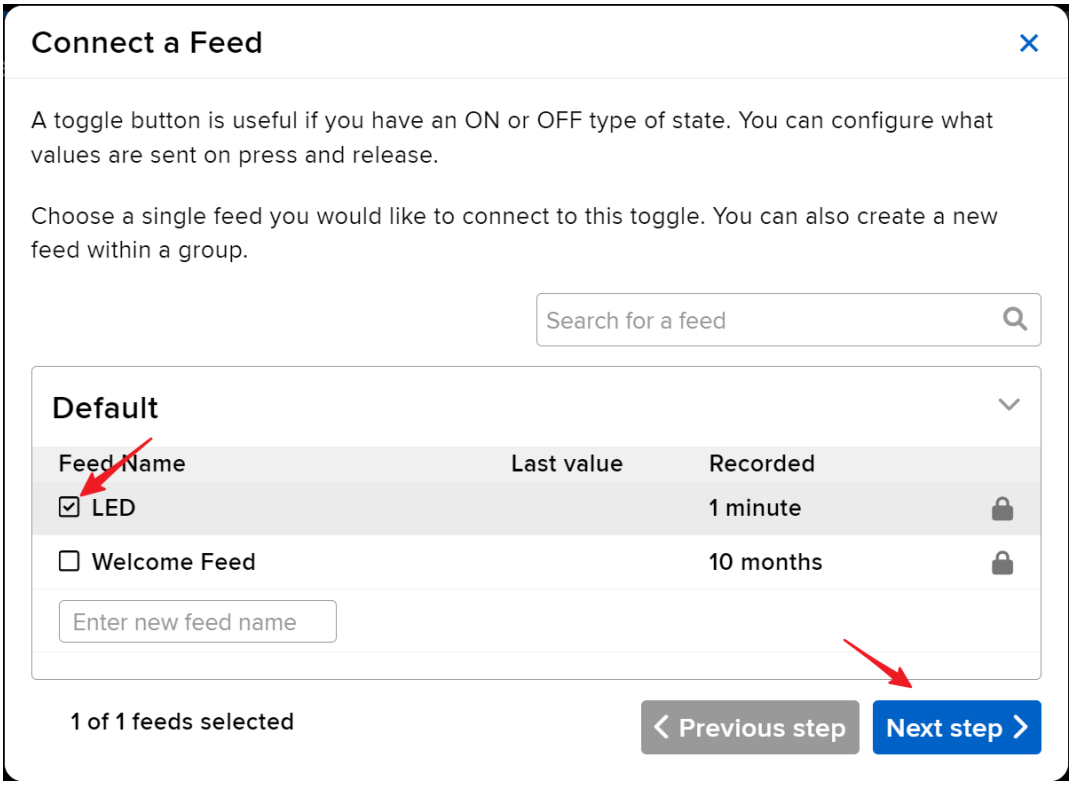
6. Create 1 **Toggle** block.



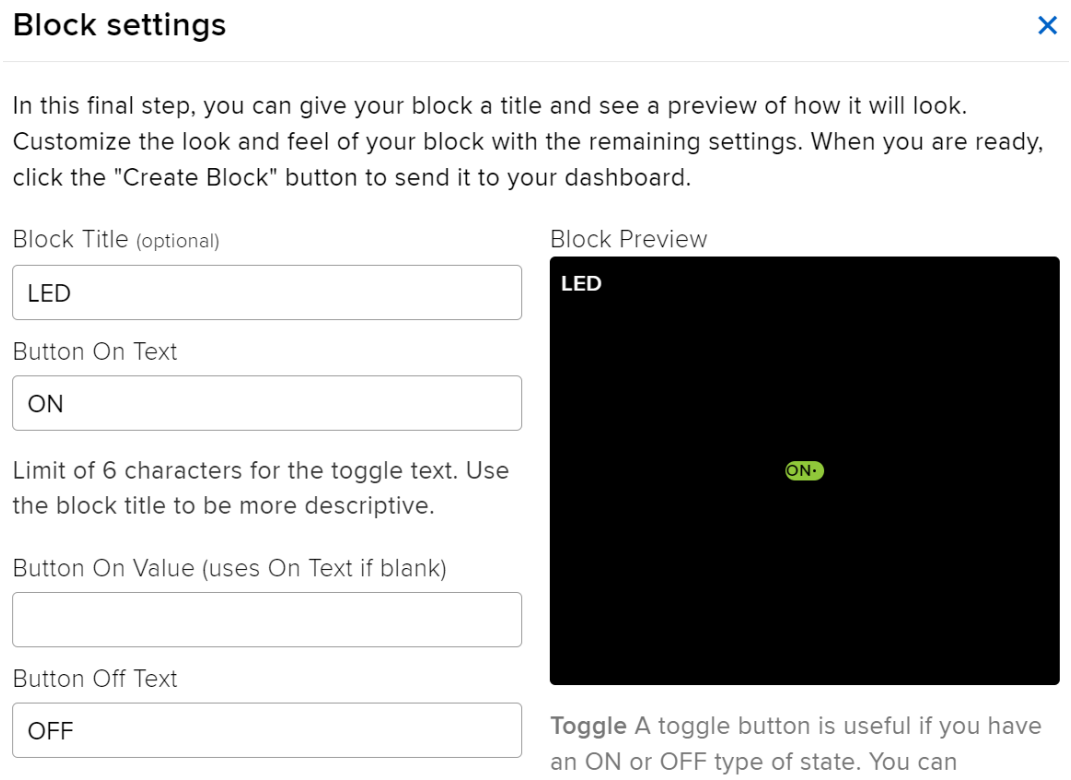
7. Next, you'll need to create a new feed here. This toggle will be used to control the LED, and we'll name this feed "LED".



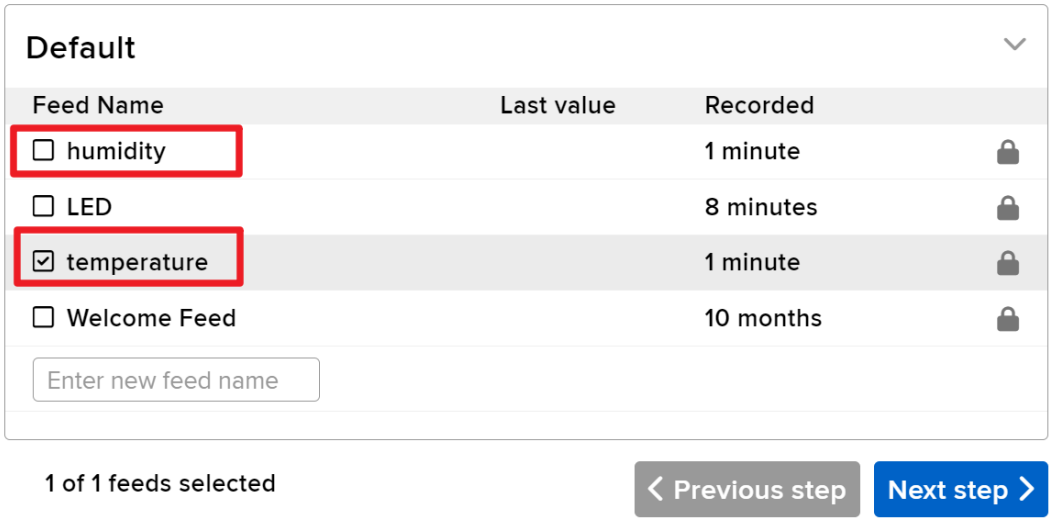
8. Check the **LED** feed, then move to the next step.



9. Complete the block settings (mainly Block Title, On Text, and Off Text), then click on the **Create block** button at the bottom right to finish.

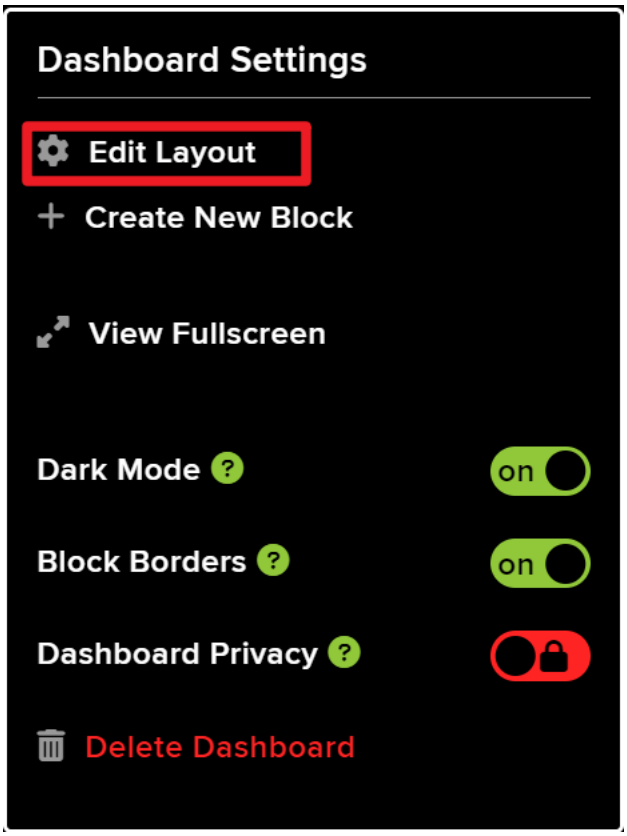


10. We also need to create two **Text Blocks** next. They will be used to display temperature and humidity. So, create two feeds named **temperature** and **humidity**.

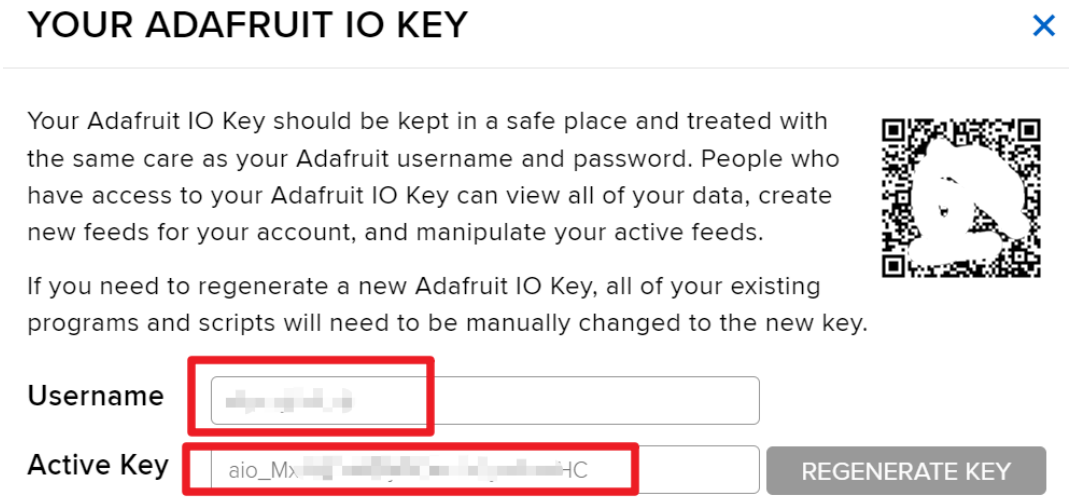


11. After creation, your Dashboard should look something like this:

12. You can adjust the layout by using the **Edit Layout** option on the Dashboard.

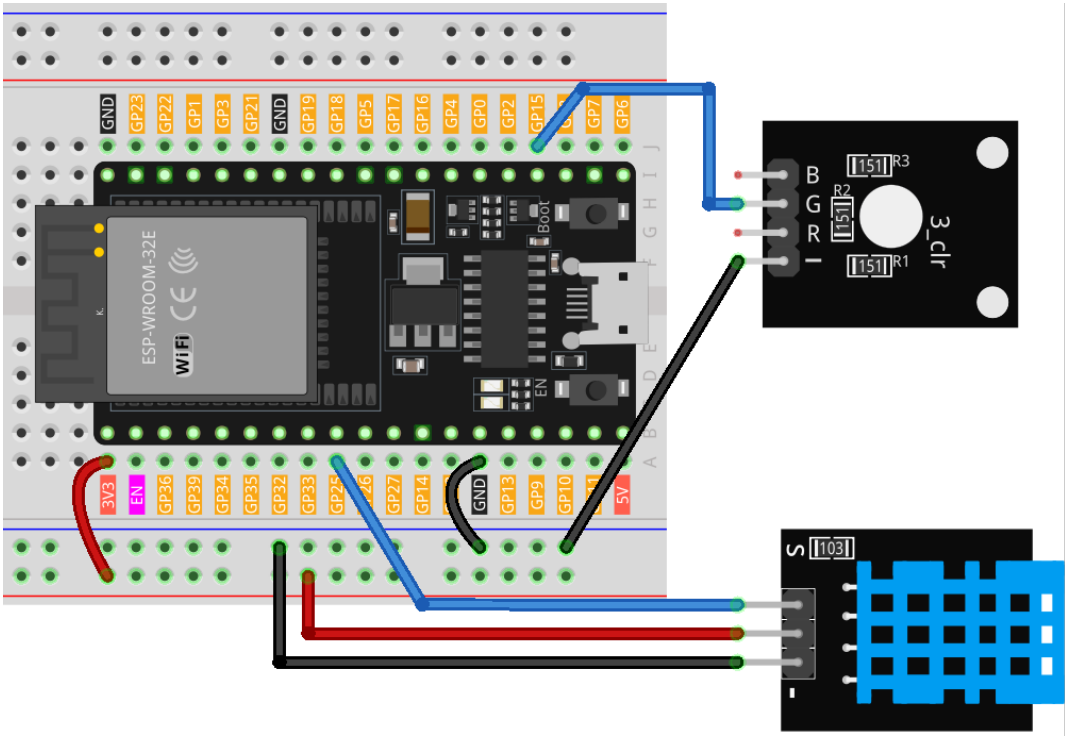


13. Click on **API KEY**, and you will see your username and **API KEY** displayed. Note these down as you'll need them for your code.



Running the Code

1. Build the circuit.



2. Then, connect ESP32 to the computer using the USB cable.
3. Open the code.
    - Open the `Lesson_48_Adafruit_IO.ino` file located in the `universal-maker-sensor-kit\esp32\Lesson_48_Adafruit_IO` directory, or copy the code into the Arduino IDE.
    - After selecting the board (ESP32 Dev Module) and the appropriate port, click the **Upload** button.
    - *Always displaying "Unknown COMxx"?*
    - The `Adafruit_MQTT Library` and `DHT sensor library` are used here, you can install them from the **Library Manager**.

4. Find the following lines and replace <SSID> and <PASSWORD> with the specific details of your WiFi network.

```
/************************* WiFi Access Point␣
↪*********************************/

#define WLAN_SSID "<SSID>"
#define WLAN_PASS "<PASSWORD>"
```

5. Then replace <YOUR_ADAFRUIT_IO_USERNAME> with your Adafruit IO username and <YOUR_ADAFRUIT_IO_KEY> with the **API KEY** you just copied.

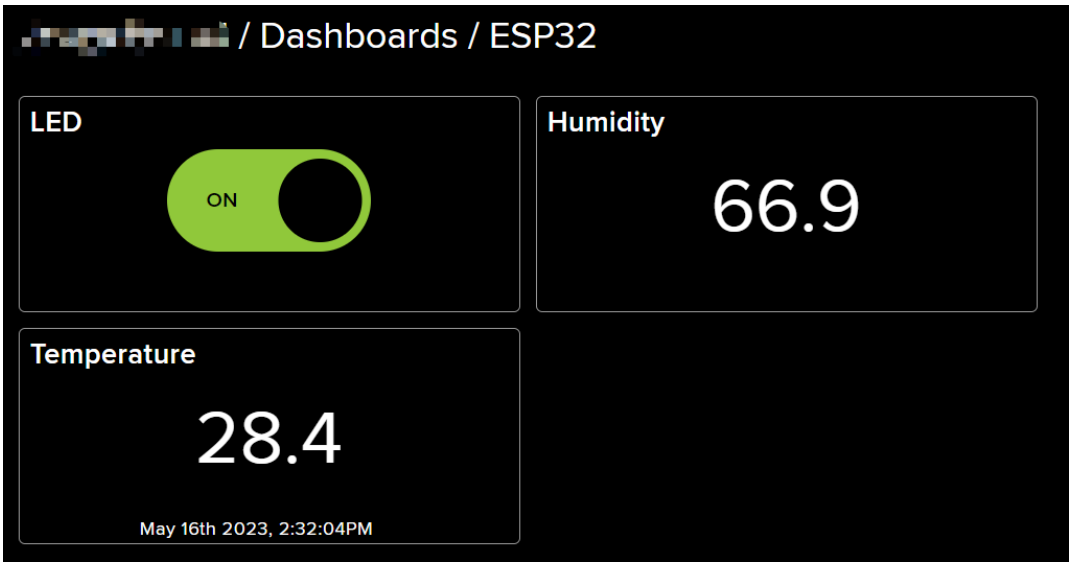
```
// Adafruit IO Account Configuration
// (to obtain these values, visit https://io.adafruit.com and click on␣
↪Active Key)
#define AIO_USERNAME "<YOUR_ADAFRUIT_IO_USERNAME>"
#define AIO_KEY      "<YOUR_ADAFRUIT_IO_KEY>"
```

6. After selecting the correct board (ESP32 Dev Module) and port, click the **Upload** button.

7. Once the code is successfully uploaded, you will observe the following message in the serial monitor, indicating successful communication with Adafruit IO.

```
Adafruit IO MQTTS (SSL/TLS) Example

Connecting to xxxxx
WiFi connected
IP address:
192.168.18.76
Connecting to MQTT... MQTT Connected!
Temperature: 27.10
Humidity: 61.00
```

8. Navigate back to Adafruit IO. Now you can observe the temperature and humidity readings on the dashboard, or utilize the LED toggle switch to control the on/off state of the external LED connected to the circuit.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook!

Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.50 Lesson 49: Blynk-based Intrusion Notification System

This project demonstrate a simple home intrusion detection system using a PIR motion sensor (HC-SR501). When the system is set to "Away" mode through the Blynk app, the PIR sensor monitors for motion. Any detected movement triggers a notification on the Blynk app, alerting the user of potential intrusion.

Required Components

In this project, we need the following components.

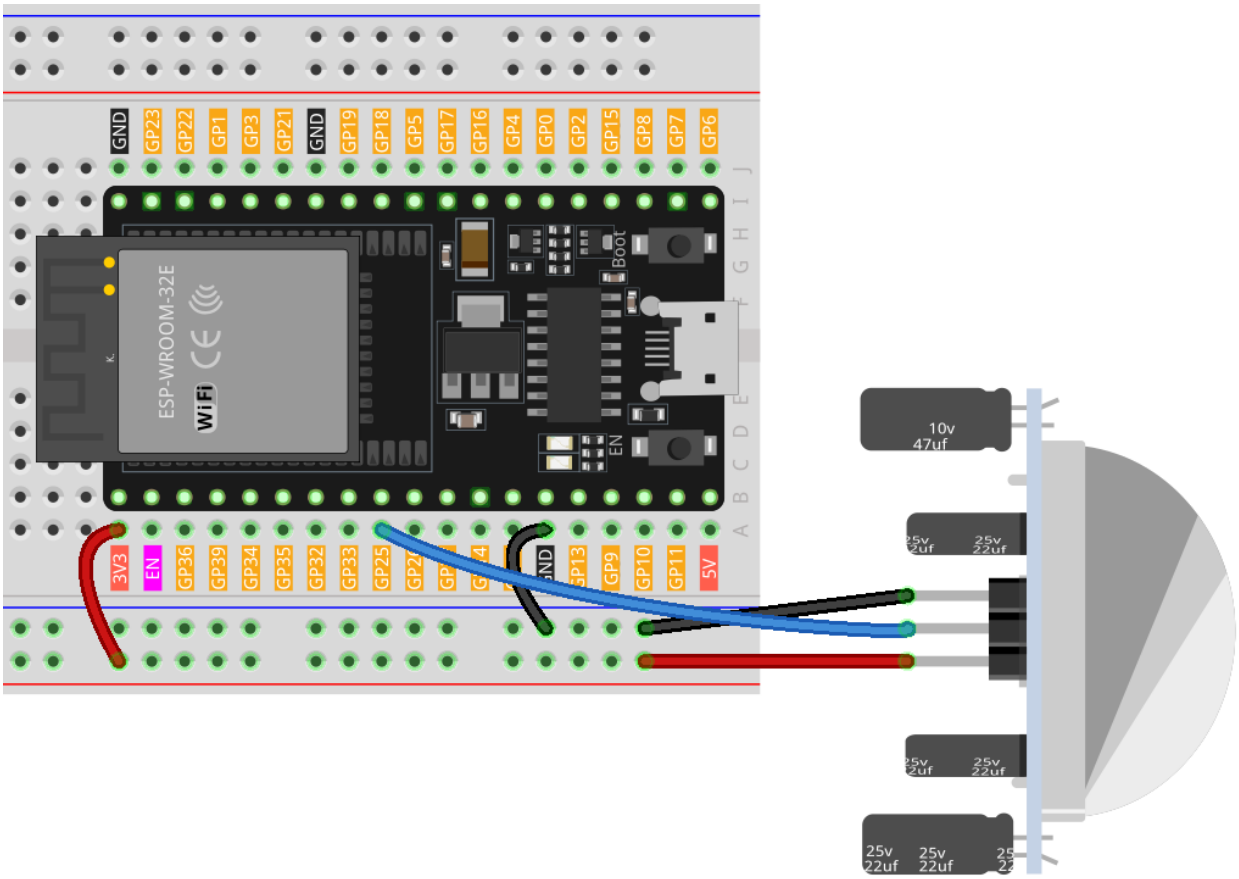
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

COMPONENT INTRODUCTION	PURCHASE LINK
ESP32 & Development Board	
PIR Motion Module (HC-SR501)	-

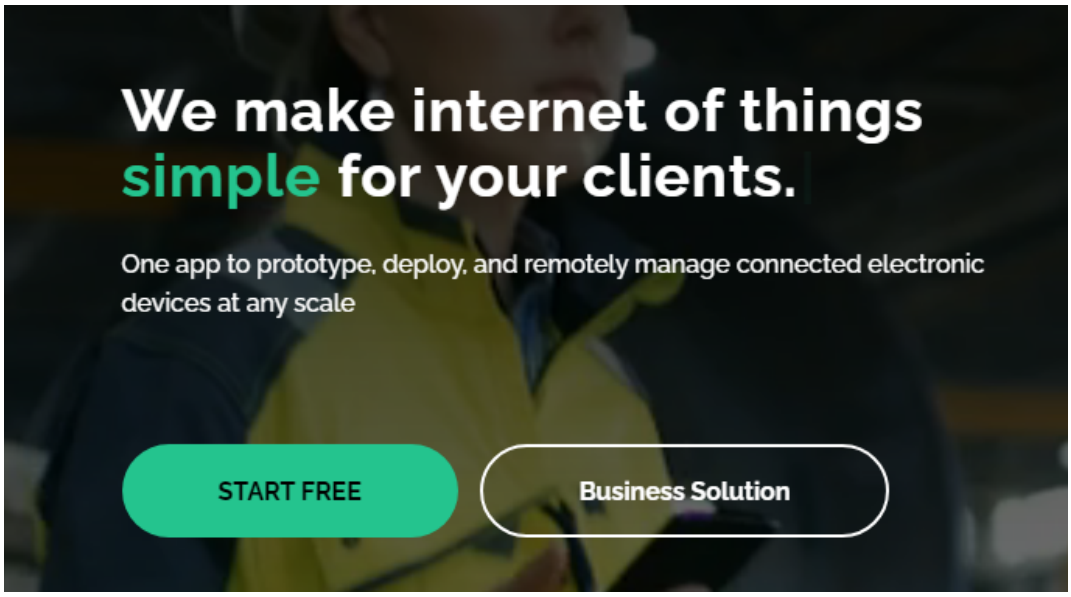
1. Circuit Assembly



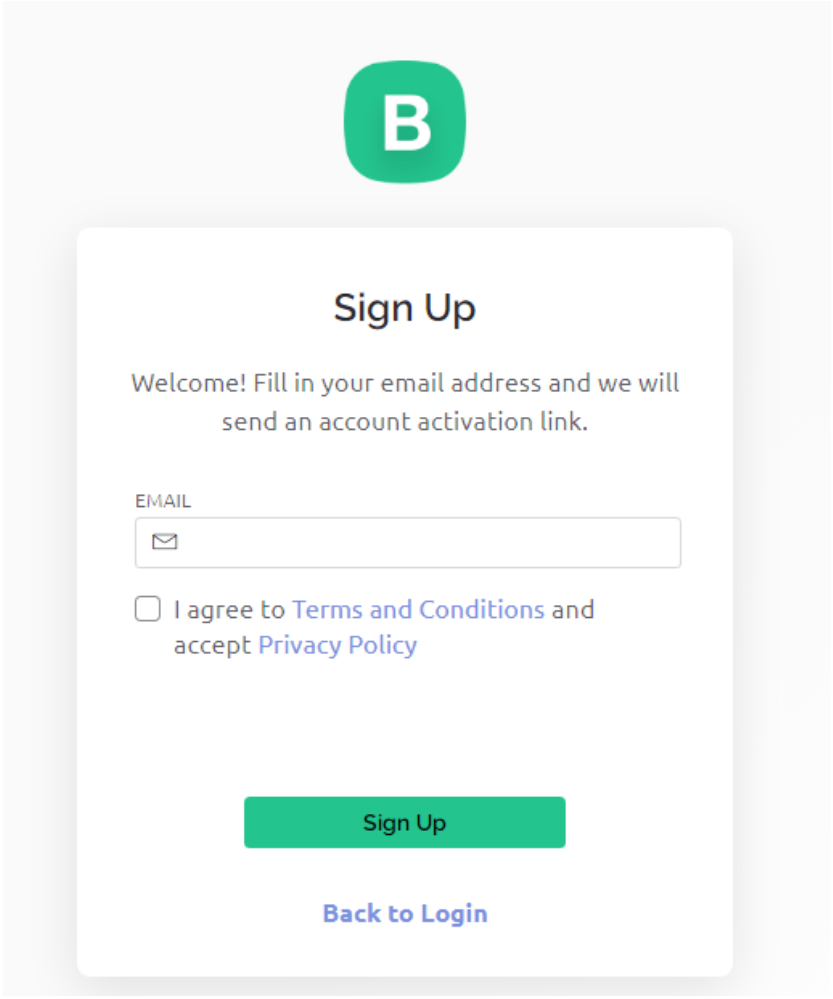
2. Blynk Configuration

2.1 Initializing Blynk

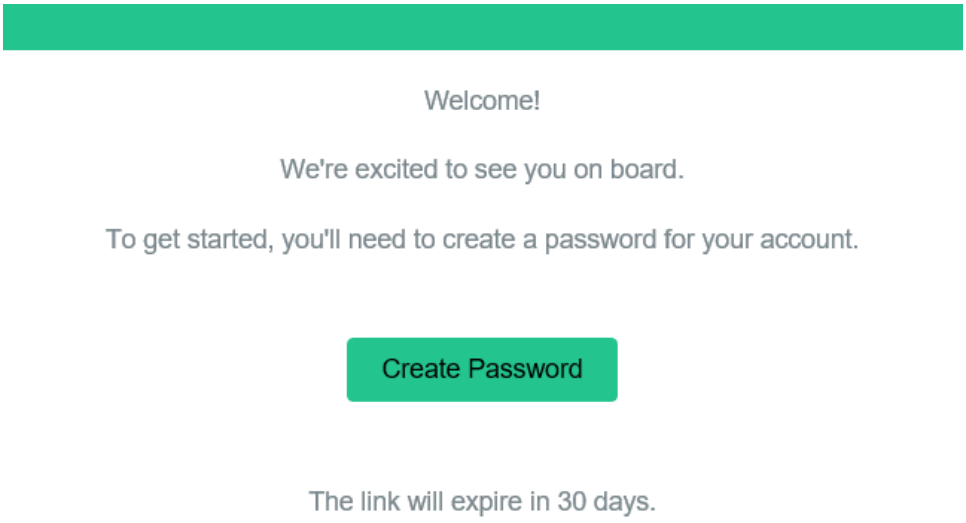
1. Navigate to the and select **START FREE**.



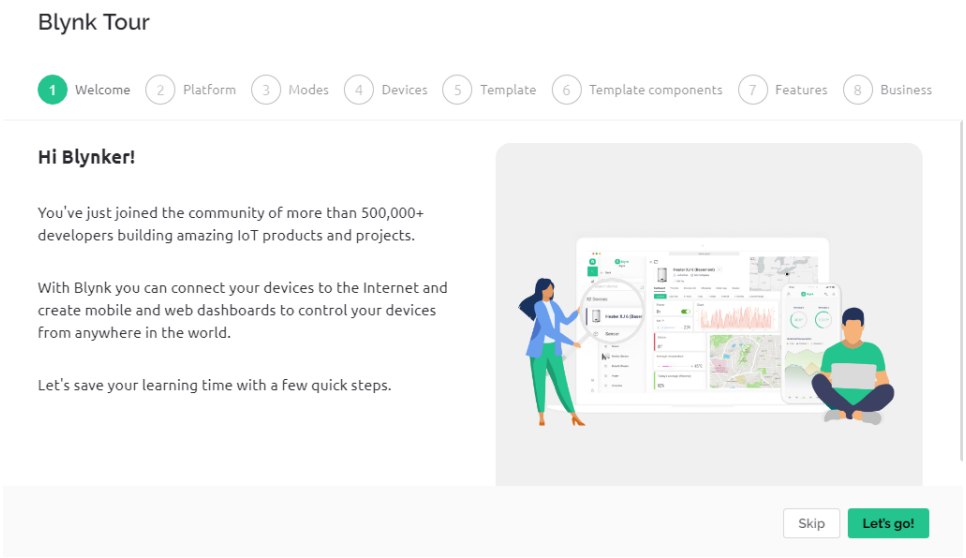
2. Enter your email to initiate the registration process.



3. Confirm your registration through your email.

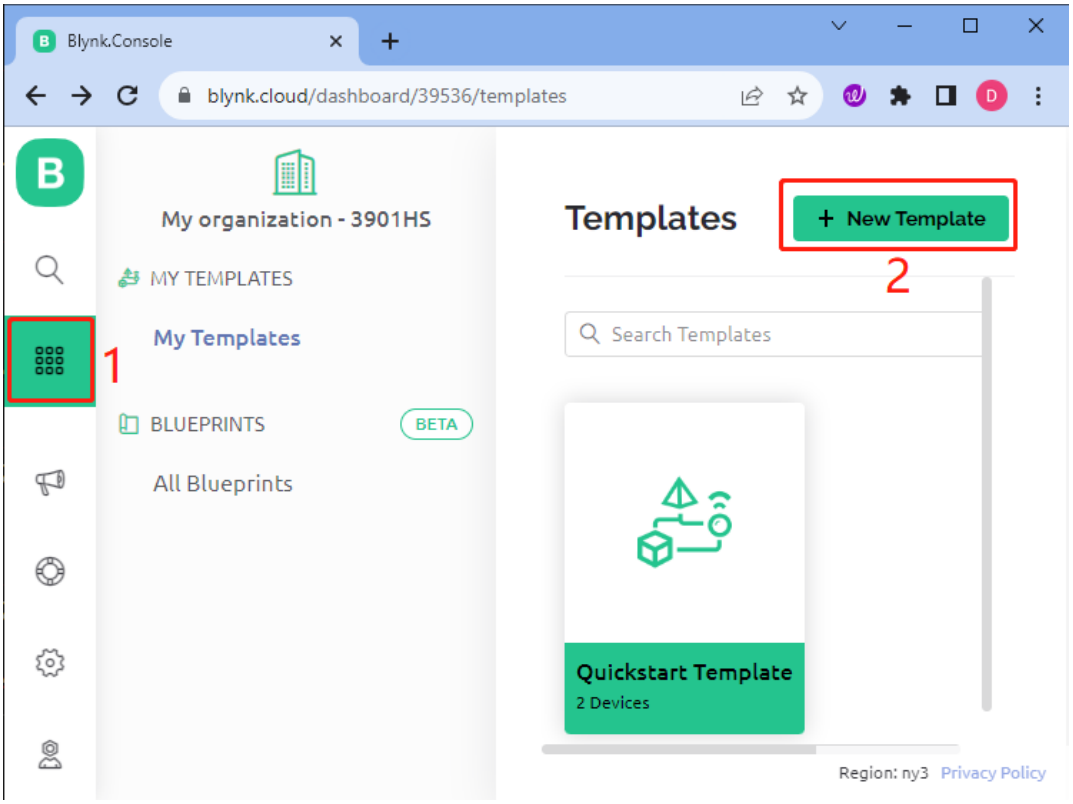


4. After confirmation, **Blynk Tour** will appear. It is recommended to select "Skip". If **Quick Start** also appears, consider skipping it as well.

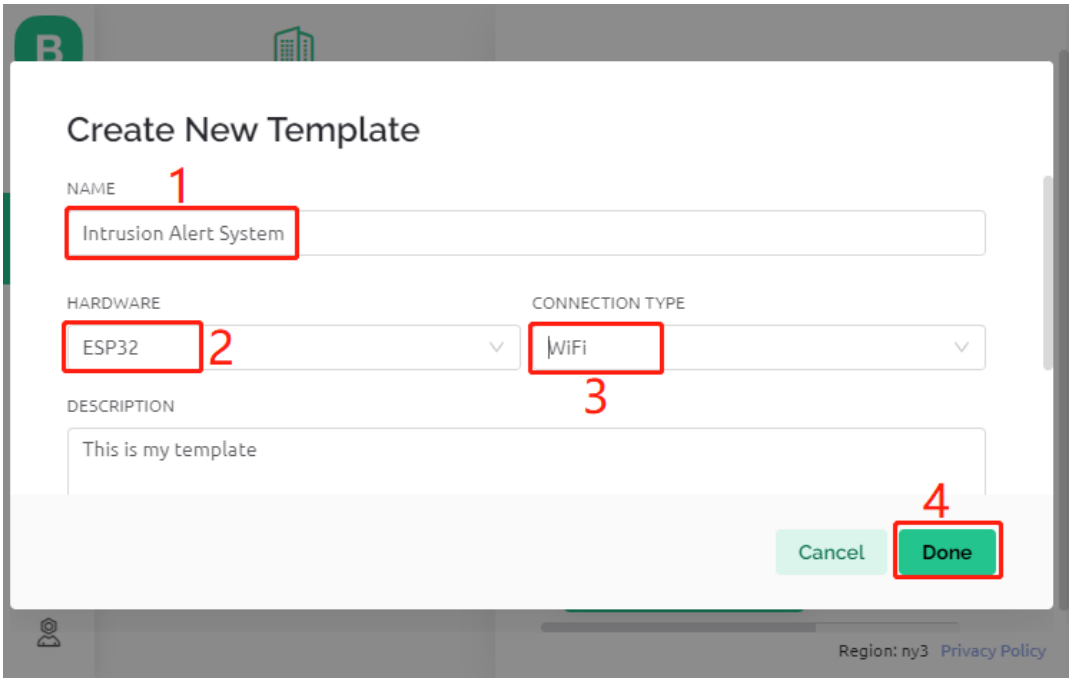


2.2 Template Creation

1. First, create a template in Blynk. Follow the subsequent instructions to create the **Intrusion Alert System** template.



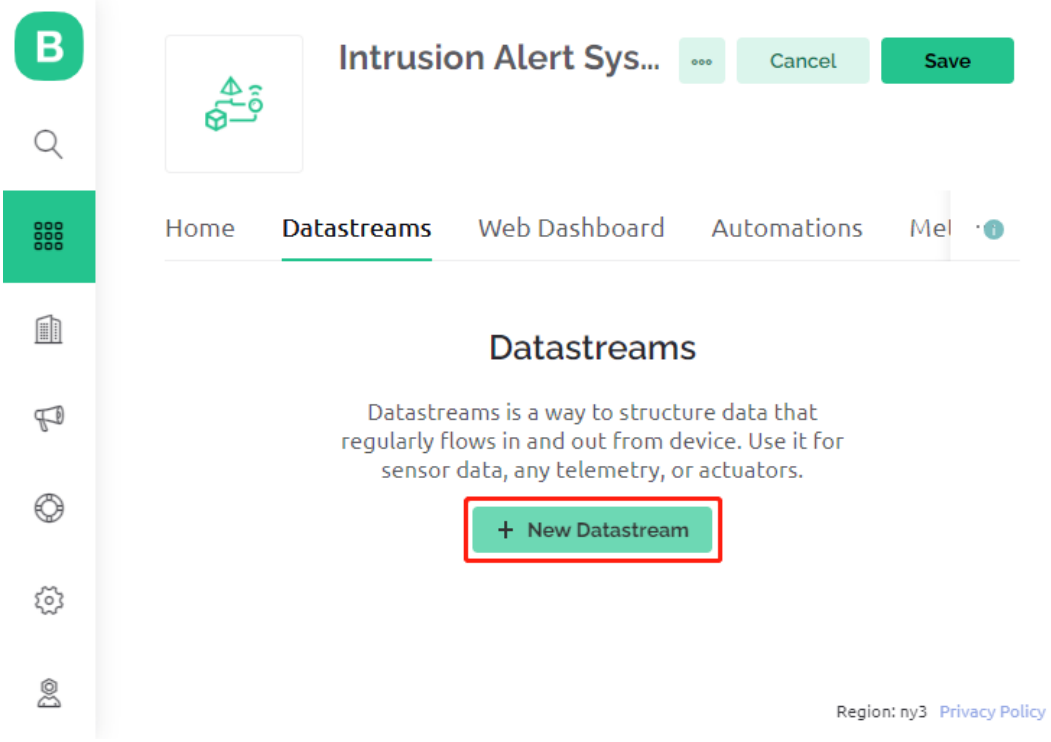
2. Assign a name to the template, select **ESP32** Hardware, and select **Connection Type** as **WiFi**, then select **Done**.



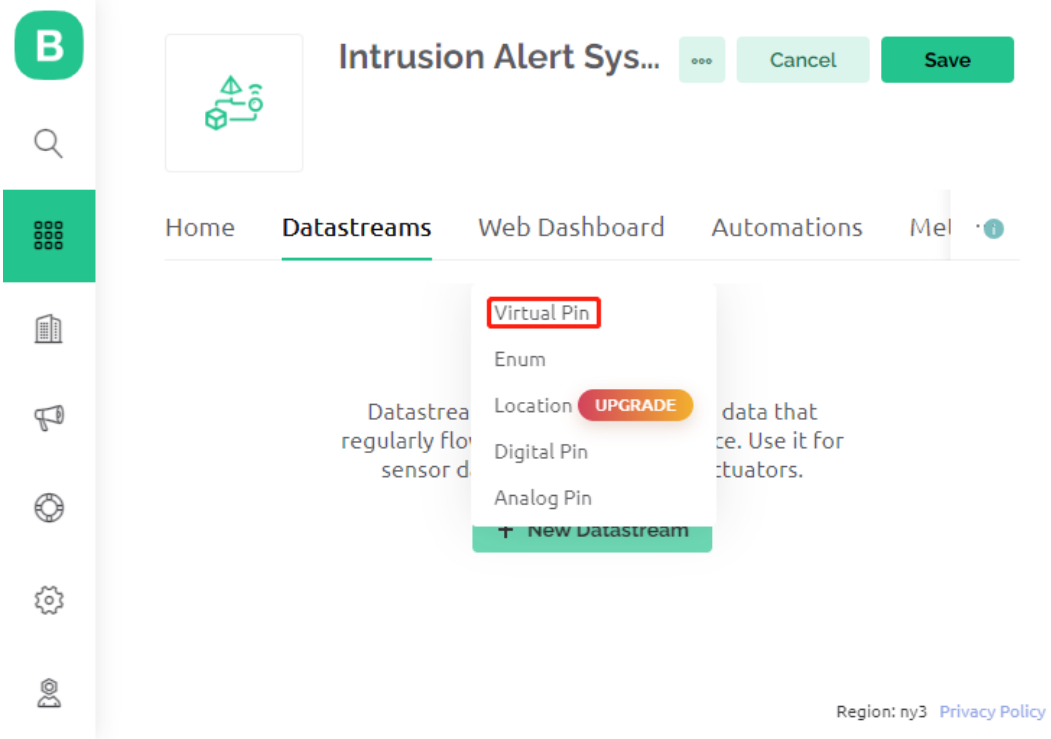
2.3 Datastream Generation

Open the template you just set up, let's create two datastreams.

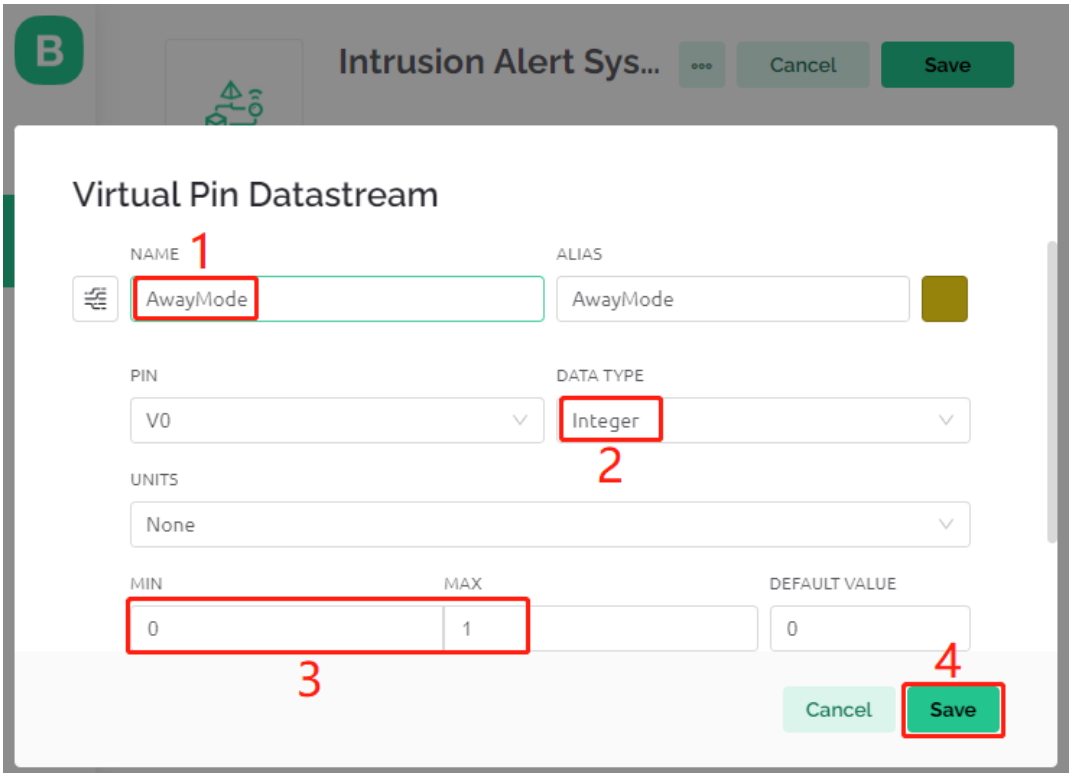
1. Click **New Datastream**.



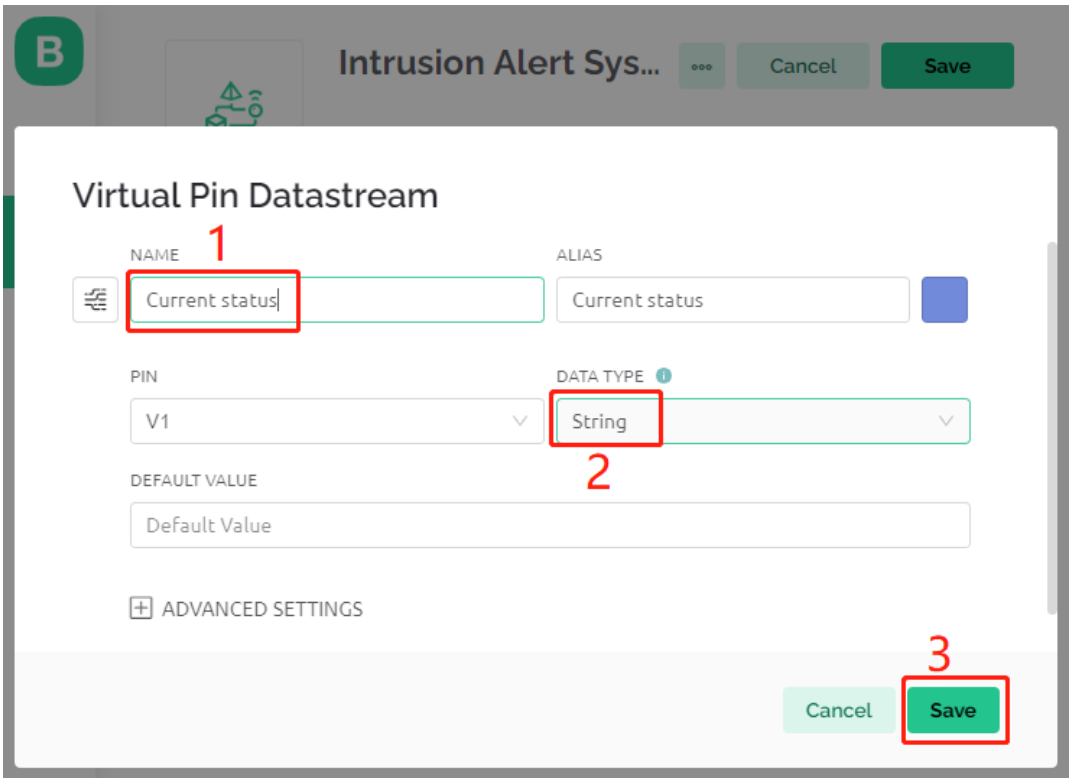
2. In the popup, choose **Virtual Pin**.



3. Name the **Virtual Pin V0** as **AwayMode**. Set the **DATA TYPE** as **Integer** with **MIN** and **MAX** values as **0** and **1**.



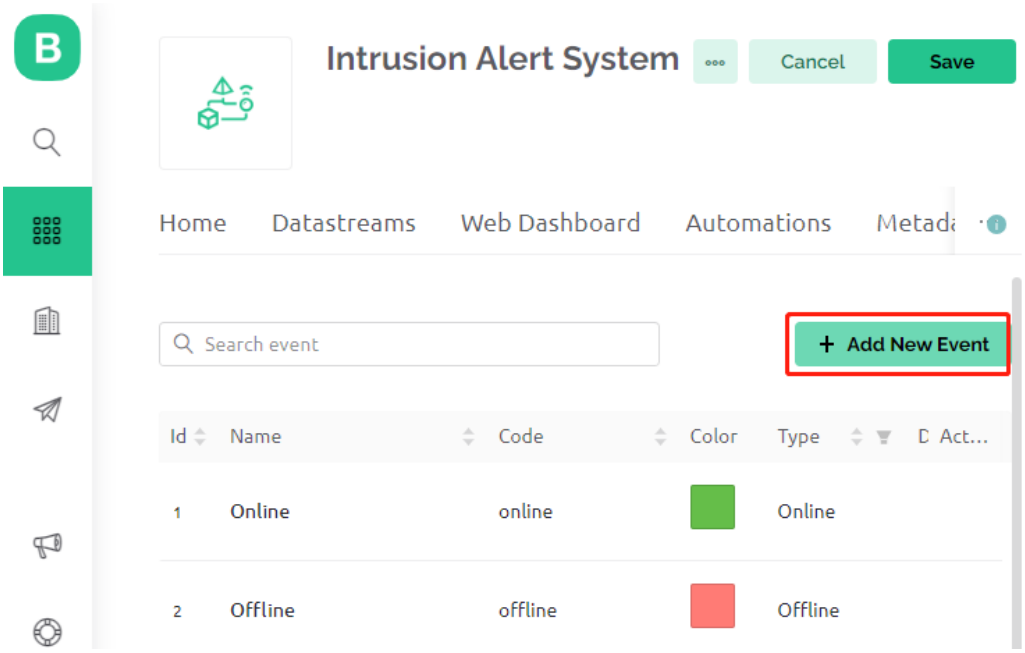
4. Similarly, create another **Virtual Pin** datastream. Name it **Current Status** and set the **DATA TYPE** to **String**.



2.4 Setting Up an Event

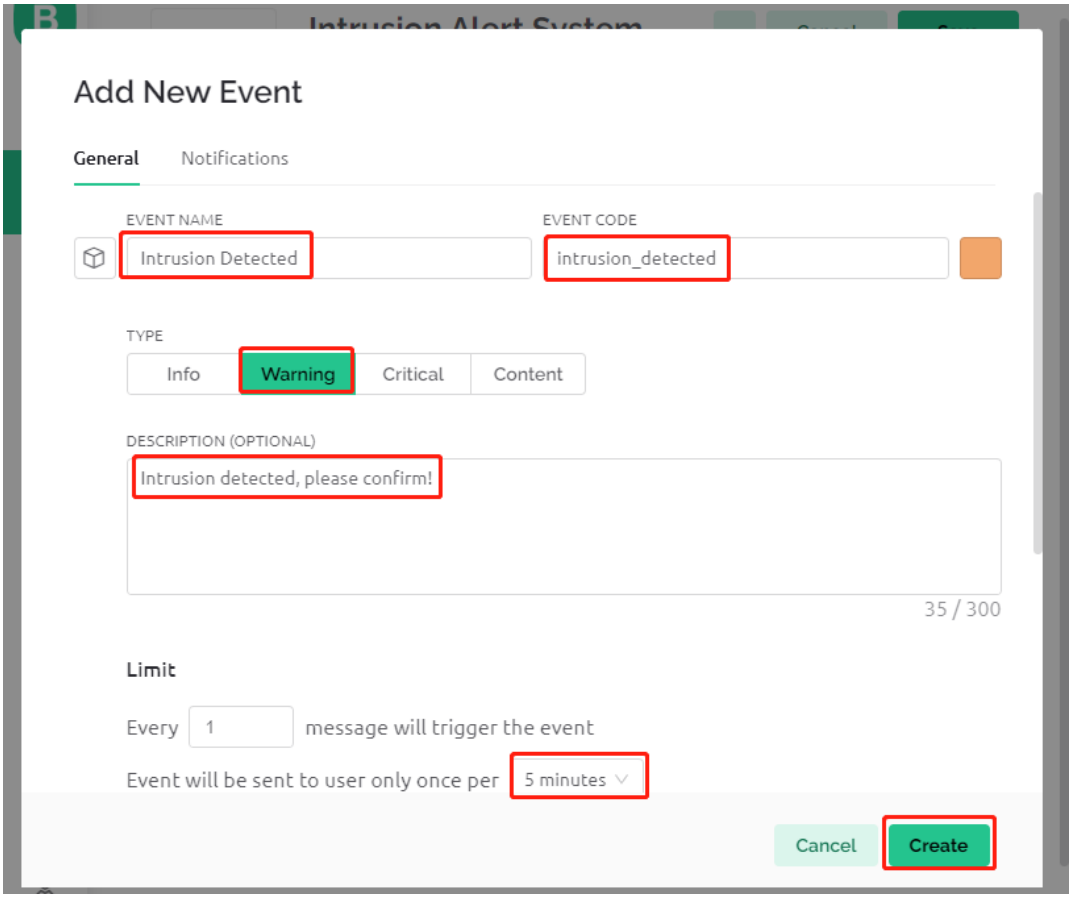
Next, we'll set up an event that sends an email notification if an intrusion is detected.

1. Click **Add New Event**.

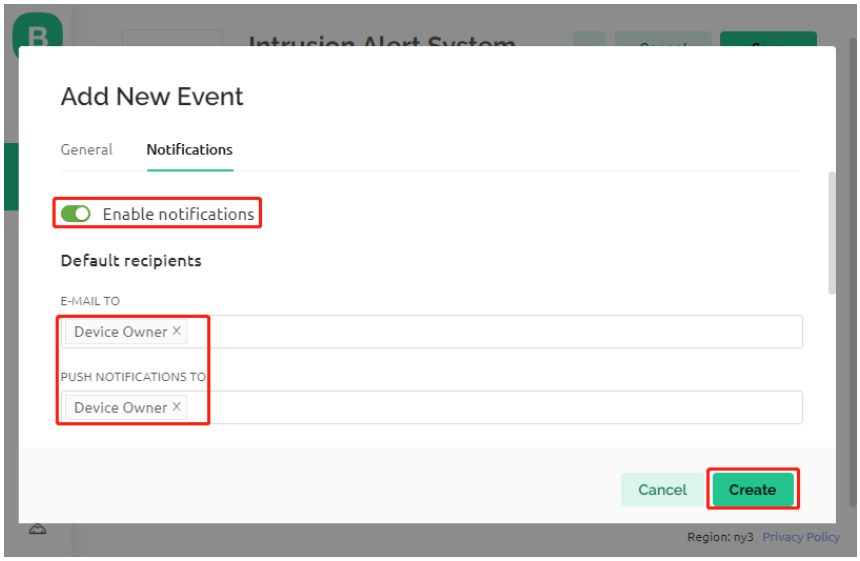


2. Define the event's name and its specific code. For **TYPE**, choose **Warning** and write a short description for the email to be sent when the event happens. You can also adjust how often you get notified.

Note: Make sure the **EVENT CODE** is set as `intrusion_detected`. This is predefined in the code, so any changes would mean you need to adjust the code as well.



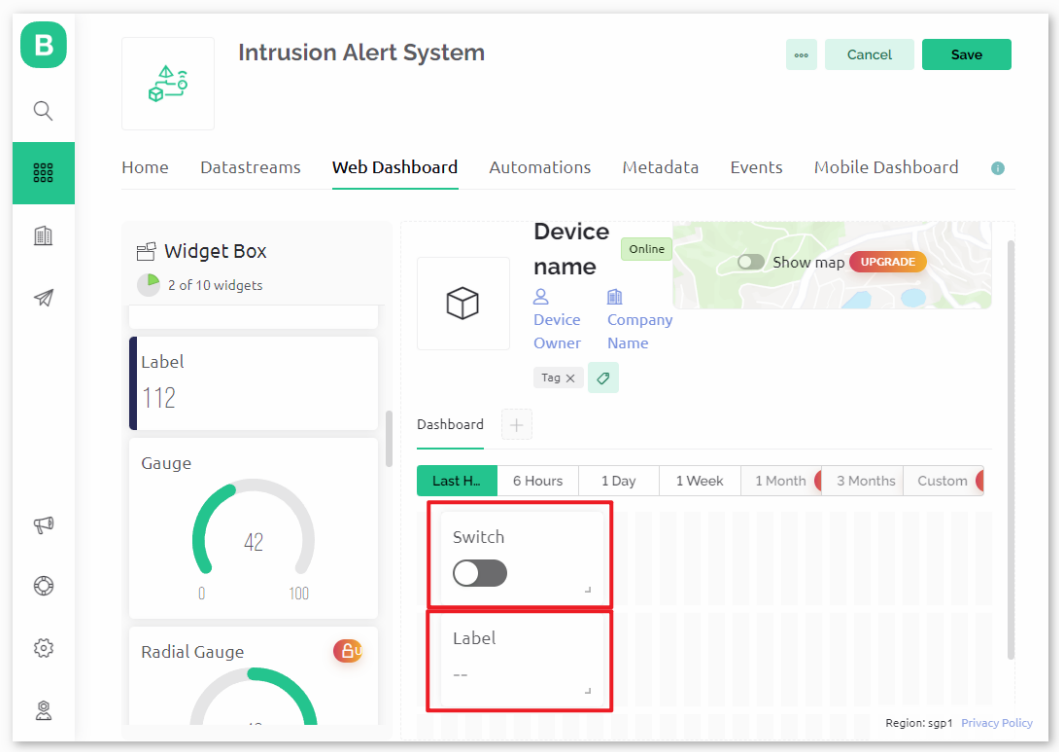
3. Go to the **Notifications** section to turn on notifications and set up email details.



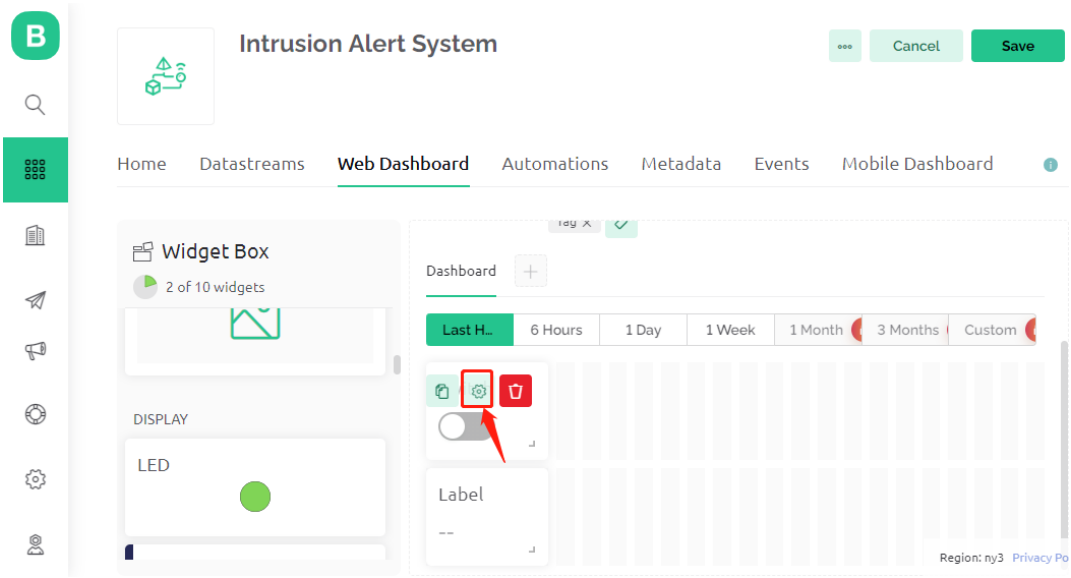
2.5 Fine-Tuning the Web Dashboard

Making sure the **Web Dashboard** interacts perfectly with the Intrusion Alert System is vital.

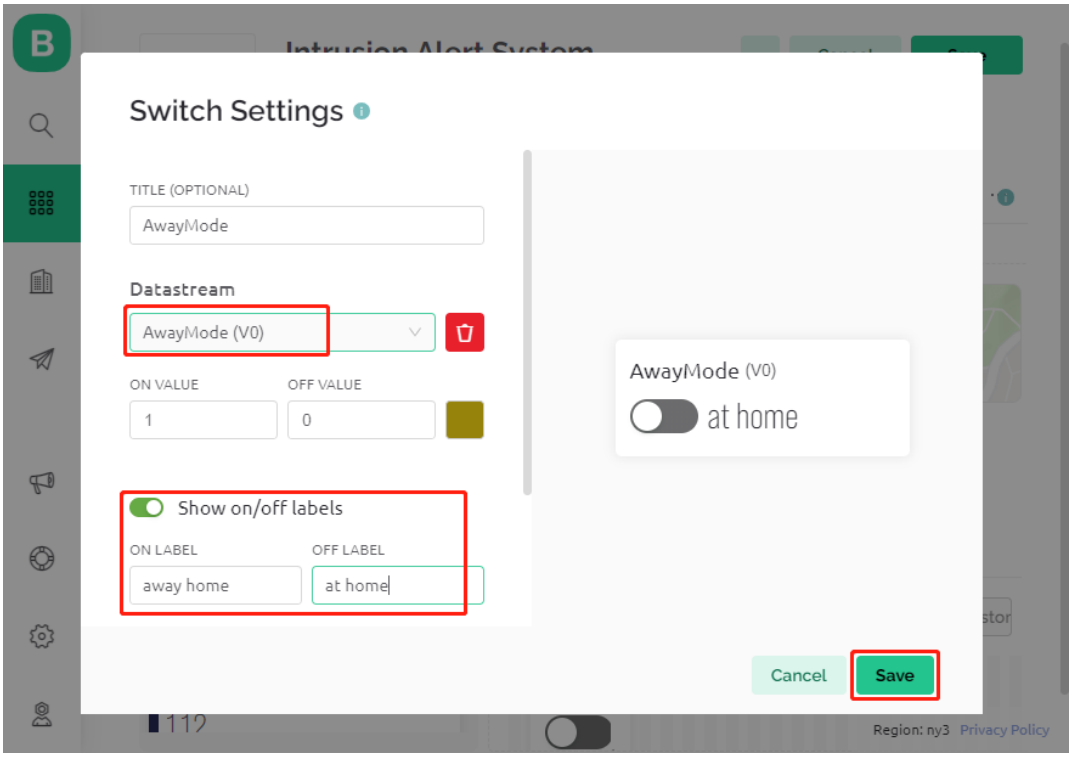
1. Simply drag and place both the **Switch widget** and the **Label widget** onto the **Web Dashboard**.



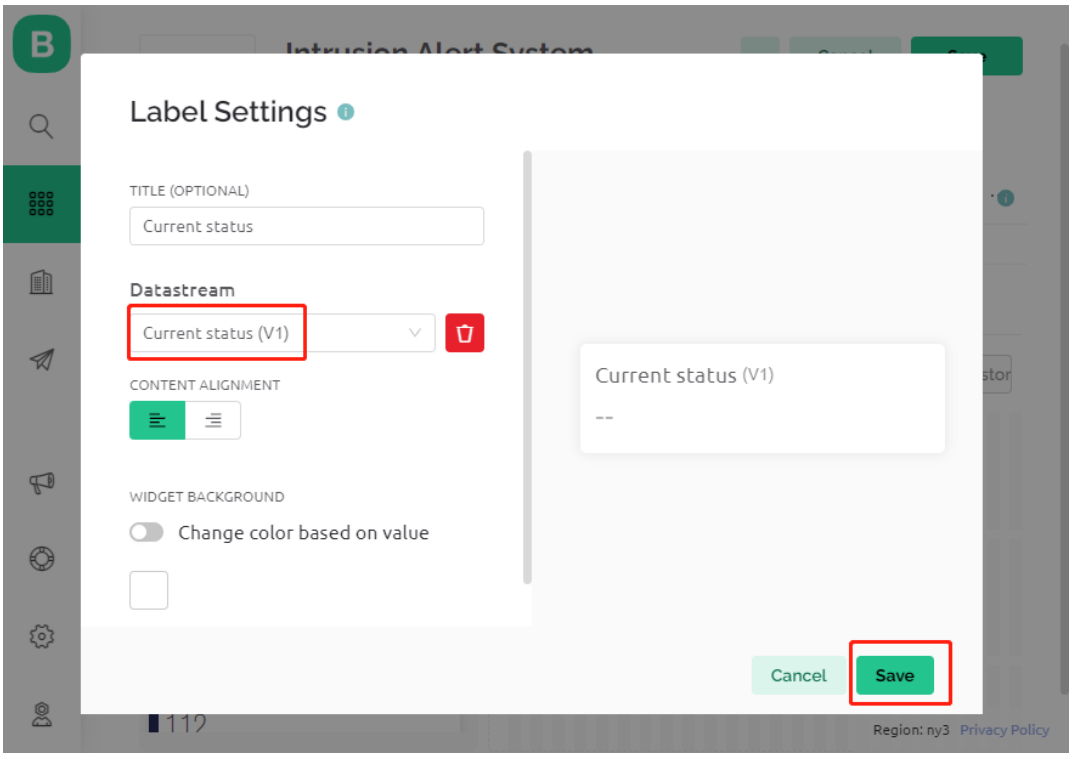
2. When you hover over a widget, three icons will appear. Use the settings icon to adjust the widget's properties.



3. In the **Switch widget** settings, select **Datastream** as **AwayMode(V0)**. Set **ONLABEL** and **OFFLABEL** to display **"away"** and **"home"**, respectively.

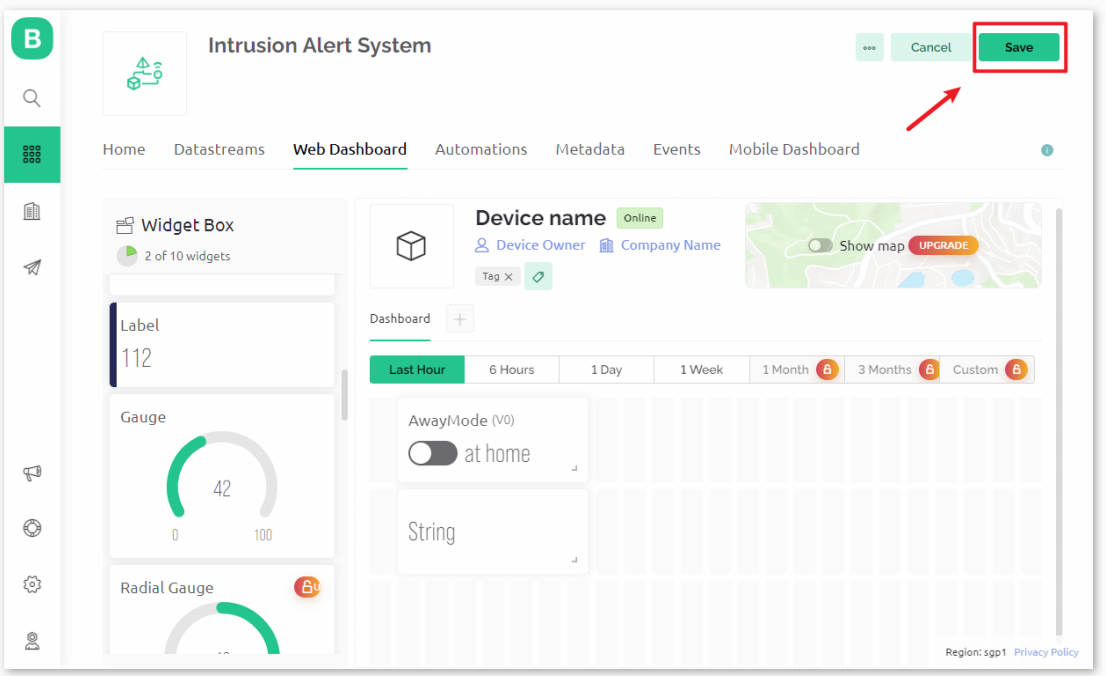


4. In the **Label widget** settings, select **Datastream** as **Current Status(V1)**.



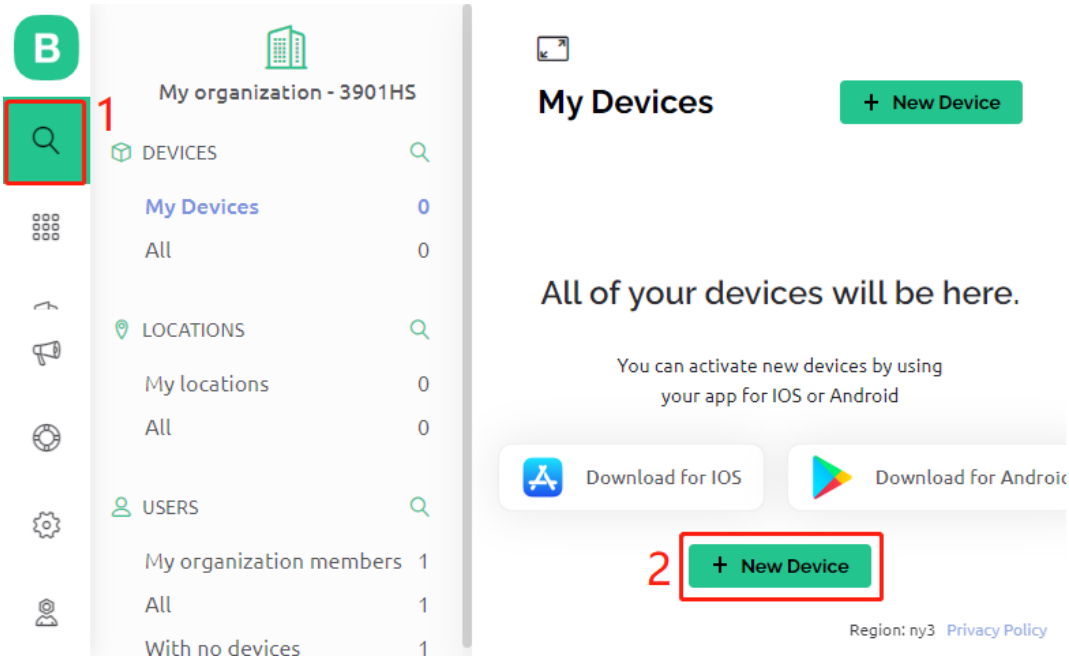
2.6 Saving the Template

Lastly, don't forget to save your template.

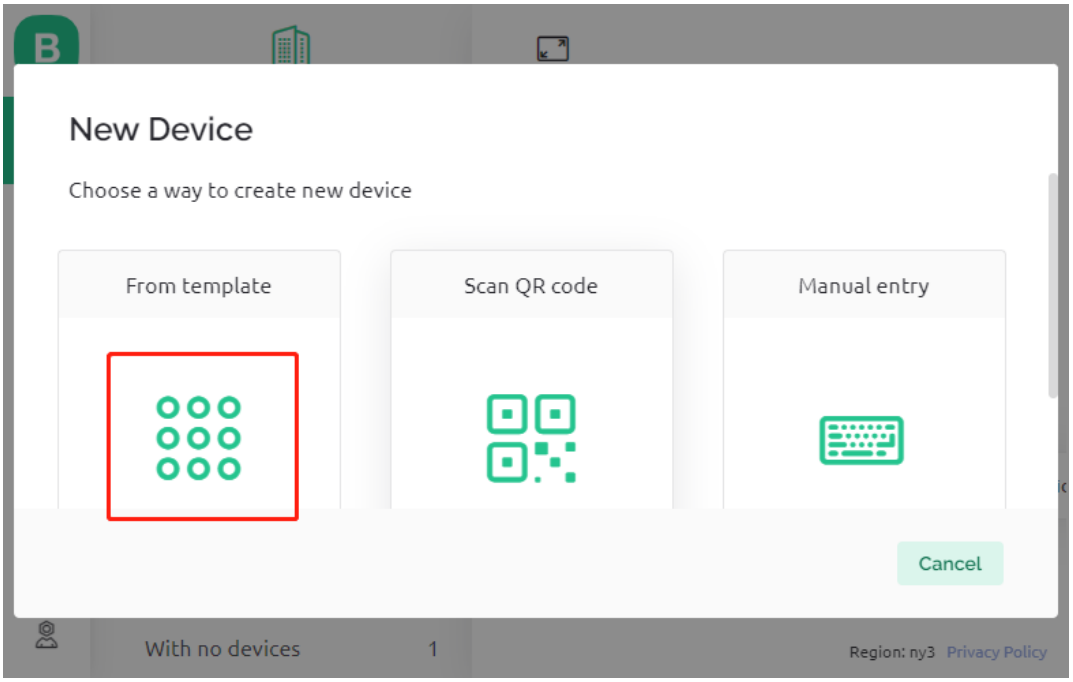


2.7 Making a Device

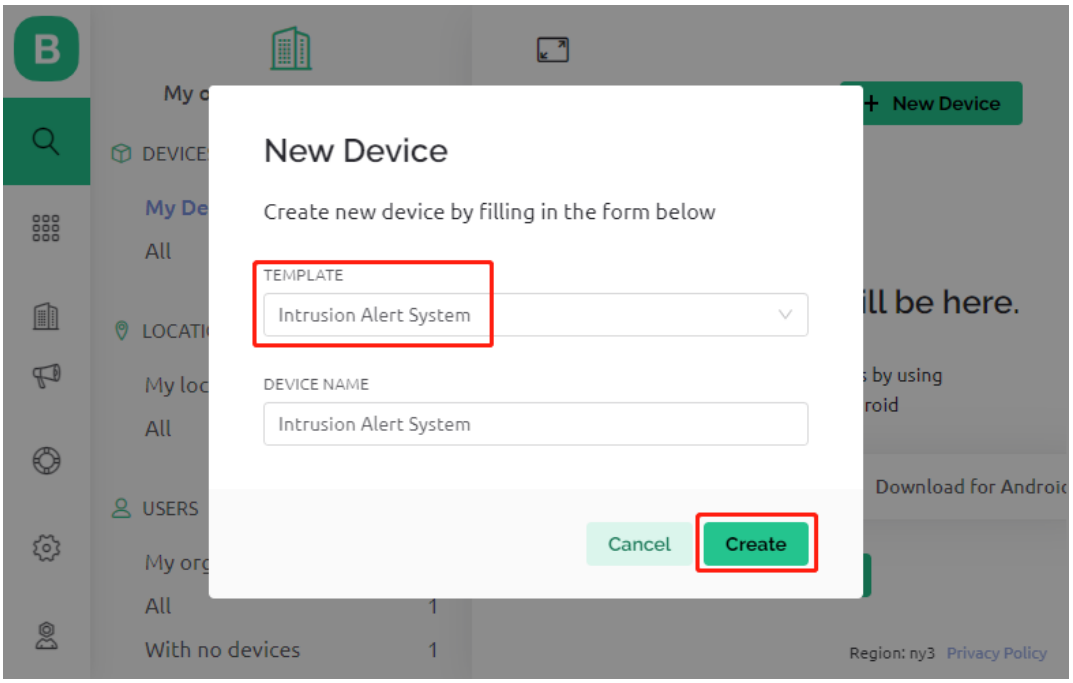
1. It's time to create a new device.



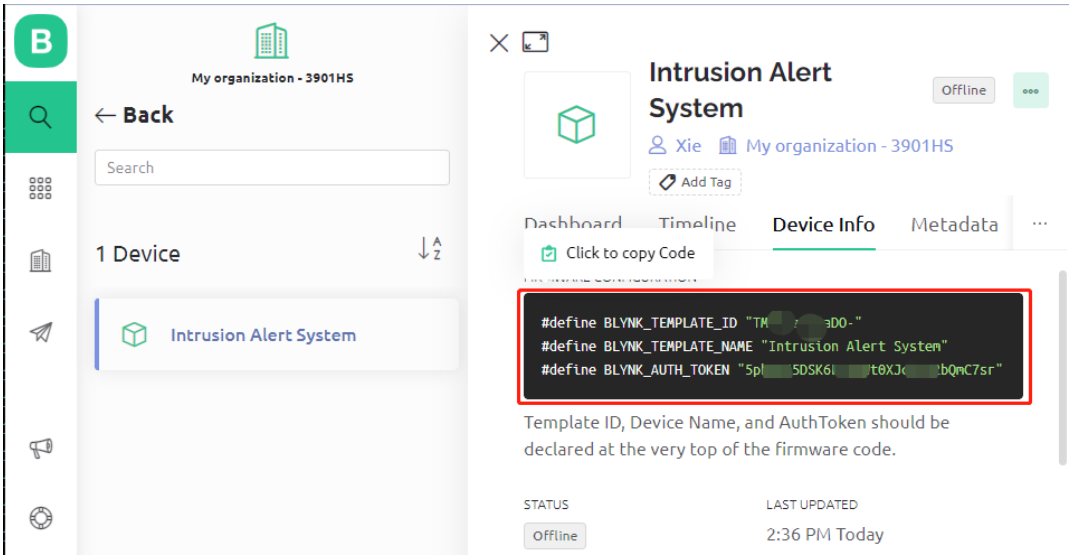
2. Click on **From template** to start with a new setup.



3. Then, pick the **Intrusion Alert System** template and click on **Create**.

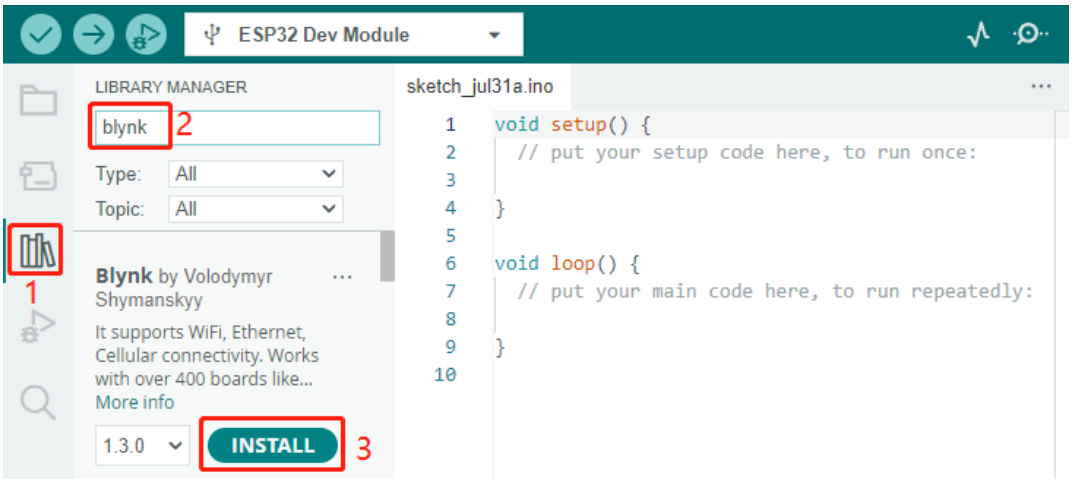


4. Here, you'll see the `Template ID`, `Device Name`, and `AuthToken`. You need to copy these into your code so the ESP32 can work with Blynk.



3. Code Execution

1. Before running the code, make sure to install the `Blynk` library from the **Library Manager** on the Arduino IDE.



2. Open the `Lesson_49_Blynk_based_intrusion_system.ino` file, which is located in the `universal-maker-sensor-kit\esp32\Lesson_49_Blynk_based_intrusion_system` directory. You can also copy its content into the Arduino IDE.

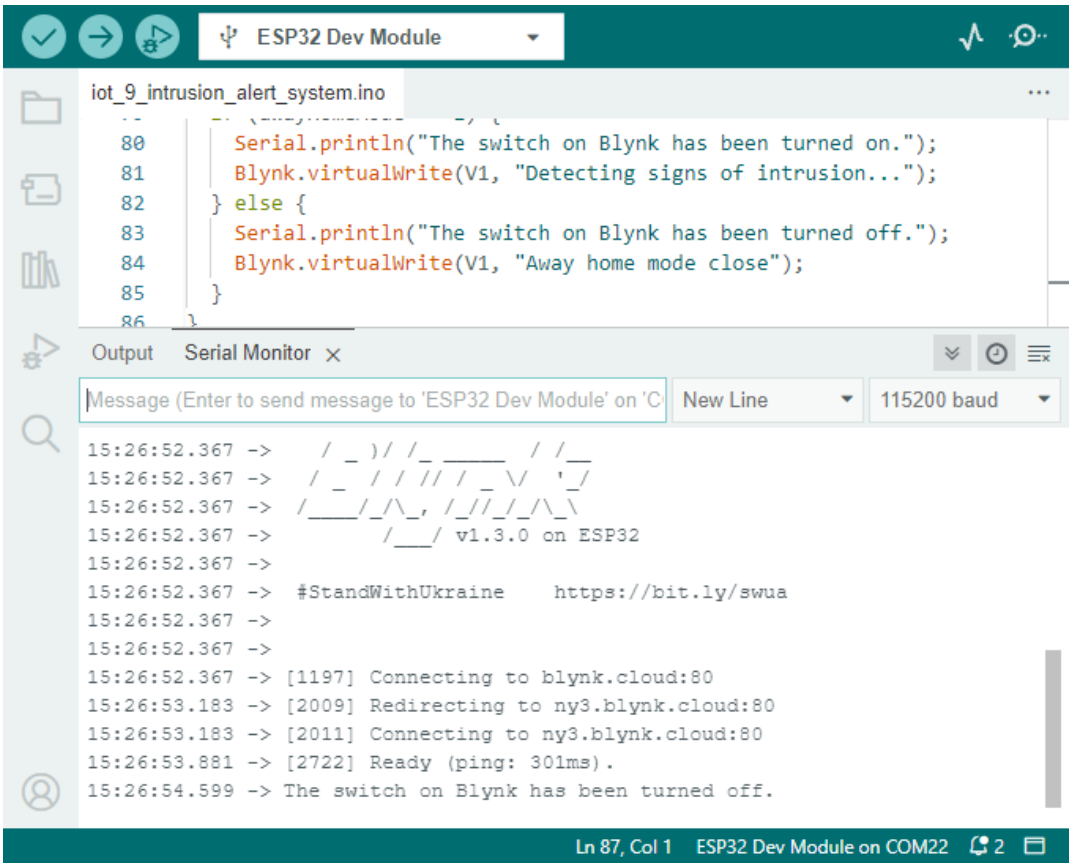
3. Replace the placeholders for `BLYNK_TEMPLATE_ID`, `BLYNK_TEMPLATE_NAME`, and `BLYNK_AUTH_TOKEN` with your own unique IDs.

```
#define BLYNK_TEMPLATE_ID "TMPxxxxxxx"
#define BLYNK_TEMPLATE_NAME "Intrusion Alert System"
#define BLYNK_AUTH_TOKEN "xxxxxxxxxxxx"
```

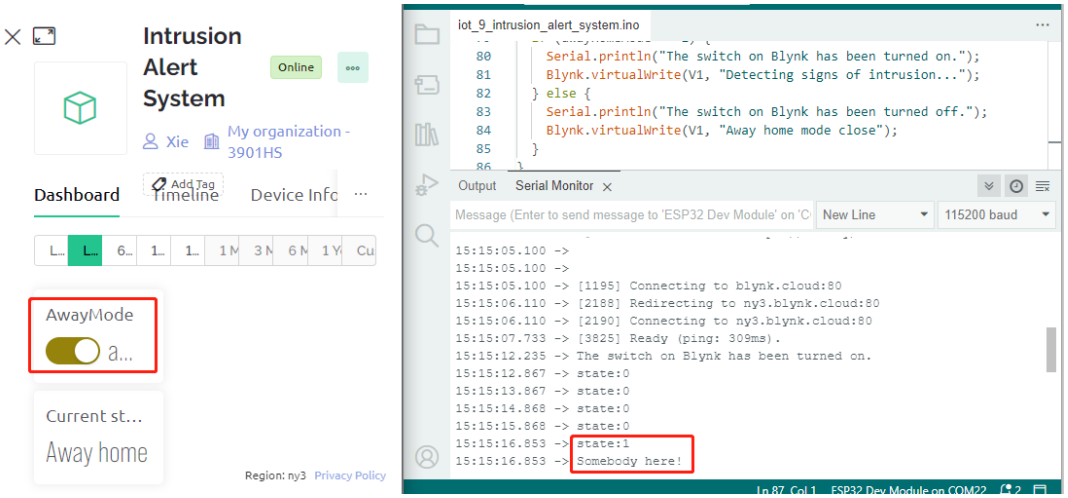
4. You also need to enter your WiFi network's `ssid` and `password`.

```
char ssid[] = "your_ssid";
char pass[] = "your_password";
```

5. Choose the correct board (**ESP32 Dev Module**) and port, then click the **Upload** button.
6. Open the Serial monitor (set baud rate to 115200) and wait for a successful connection message.



7. After a successful connection, activating the switch in Blynk will start the PIR module's surveillance. When motion is detected (state of 1), it will say, "Somebody here!" and send an alert to your email.



4. Code explanation

1. **Configuration & Libraries**

   Here, you set up the Blynk constants and credentials. You also include the necessary libraries for the ESP32 and Blynk.

   ```
   /* Comment this out to disable prints and save space */
   #define BLYNK_PRINT Serial

   #define BLYNK_TEMPLATE_ID "xxxxxxxxxx"
   #define BLYNK_TEMPLATE_NAME "Intrusion Alert System"
   #define BLYNK_AUTH_TOKEN "xxxxxxxxxxxxxxxxxxxxxxxxxx"

   #include <WiFi.h>
   #include <WiFiClient.h>
   #include <BlynkSimpleEsp32.h>
   ```

2. **WiFi Setup**

   Enter your WiFi credentials.

   ```
   char ssid[] = "your_ssid";
   char pass[] = "your_password";
   ```

3. **PIR Sensor Configuration**

   Set the pin where the PIR sensor is connected and initialize the state variables.

   ```
   const int sensorPin = 14;
   int state = 0;
   int awayHomeMode = 0;
   BlynkTimer timer;
   ```

4. **setup() Function**

   This function initializes the PIR sensor as an input, sets up serial communication, connects to WiFi, and configures Blynk.

   - We use `timer.setInterval(1000L, myTimerEvent)` to set the timer interval in `setup()`, here we set to execute the `myTimerEvent()` function every **1000ms**. You can modify the first parameter of `timer.setInterval(1000L, myTimerEvent)` to change the interval between `myTimerEvent` executions.

   ```
   void setup() {

       pinMode(sensorPin, INPUT);  // Set PIR sensor pin as input
       Serial.begin(115200);           // Start serial communication at 115200 baud␣
   ↪rate for debugging

       // Configure Blynk and connect to WiFi
       Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);

       timer.setInterval(1000L, myTimerEvent);  // Setup a function to be called every␣
   ↪second
   }
   ```

5. **loop() Function**

The loop function continuously runs Blynk and the Blynk timer functions.

```
void loop() {
   Blynk.run();
   timer.run();
}
```

6. **Blynk App Interaction**

   These functions are called when the device connects to Blynk and when there's a change in the state of the virtual pin V0 on the Blynk app.

   - Every time the device connects to the Blynk server, or reconnects due to poor network conditions, the `BLYNK_CONNECTED()` function is called. The `Blynk.syncVirtual()` command request a single Virtual Pin value. The specified Virtual Pin will perform `BLYNK_WRITE()` call.
   - Whenever the value of a virtual pin on the BLYNK server changes, it will trigger `BLYNK_WRITE()`.

```
// This function is called every time the device is connected to the Blynk.Cloud
BLYNK_CONNECTED() {
    Blynk.syncVirtual(V0);
}

// This function is called every time the Virtual Pin 0 state changes
BLYNK_WRITE(V0) {
    awayHomeMode = param.asInt();
    // additional logic
}
```

7. **Data Handling**

   Every second, the `myTimerEvent()` function calls `sendData()`. If the away mode is enabled on Blynk, it checks the PIR sensor and sends a notification to Blynk if motion is detected.

   - We use `Blynk.virtualWrite(V1, "Somebody in your house! Please check!");` to change the text of a label.
   - Use `Blynk.logEvent("intrusion_detected");` to log event to Blynk.

```
void myTimerEvent() {
   sendData();
}

void sendData() {
   if (awayHomeMode == 1) {
      state = digitalRead(sensorPin);  // Read the state of the PIR sensor

      Serial.print("state:");
      Serial.println(state);

      // If the sensor detects movement, send an alert to the Blynk app
      if (state == HIGH) {
        Serial.println("Somebody here!");
        Blynk.virtualWrite(V1, "Somebody in your house! Please check!");
        Blynk.logEvent("intrusion_detected");
      }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

Reference

-
-
-
-
-
-
-

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.4.51 Lesson 50: Android Application - RGB LED Operation via Arduino and Bluetooth

The objective of this project is to develop an Android application capable of manipulating the hue of an RGB LED through a smartphone using Bluetooth technology.

This Android application will be constructed utilizing a complimentary web-based platform known as MIT App Inventor 2. The project presents an excellent opportunity to gain familiarity with the interfacing of an Arduino with a smartphone.

Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

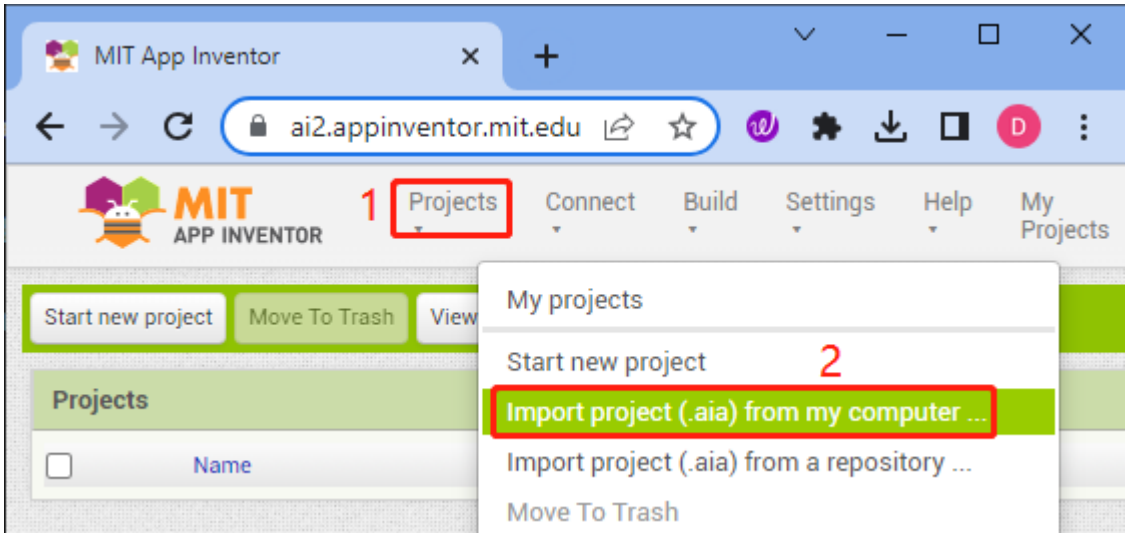
COMPONENT INTRODUCTION	PURCHASE LINK
ESP32 & Development Board	
Breadboard	
RGB LED Module	-

1. Creation of the Android Application

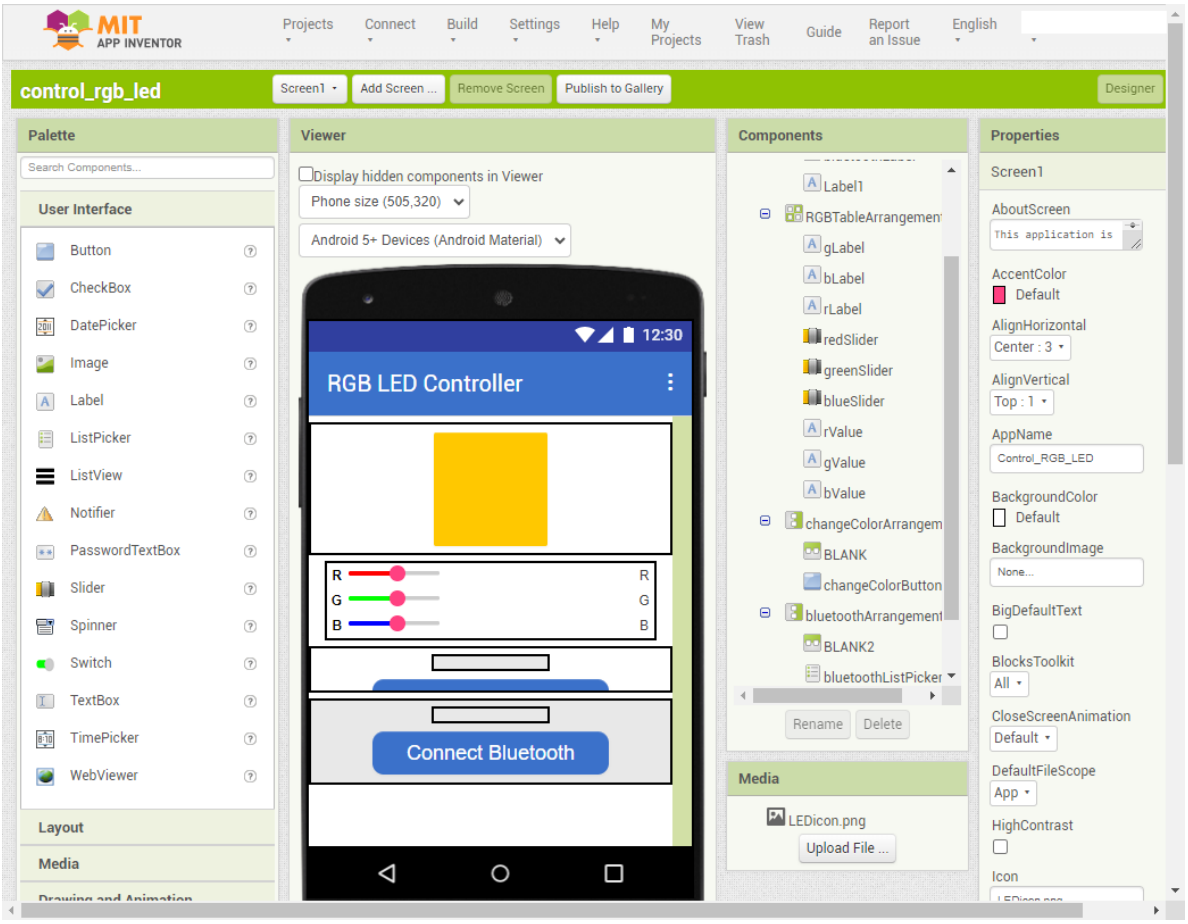
The Android application will be fashioned using a free web application known as . MIT App Inventor serves as an excellent starting point for Android development, owing to its intuitive drag-and-drop features allowing for the creation of simplistic applications.

Now, let's begin.

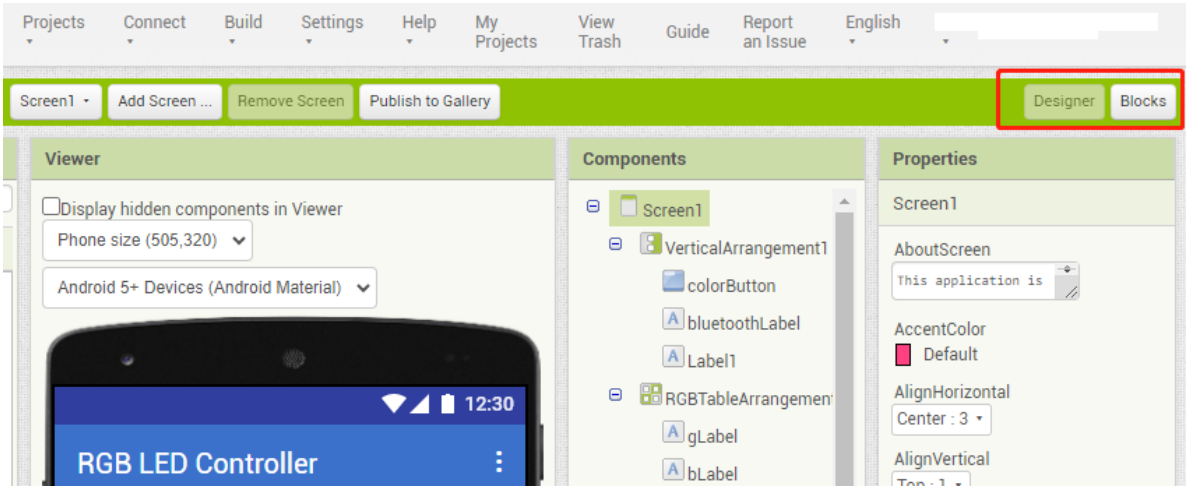
1. Here is the login page: http://ai2.appinventor.mit.edu. You will require a Google account to register with MIT App Inventor.
2. After logging in, navigate to **Projects** -> **Import project (.aia) from my computer**. Subsequently, upload the `control_rgb_led.aia` file located in the path `esp32-starter-kit-main\c\codes\iot_10_bluetooth_app_inventor`.



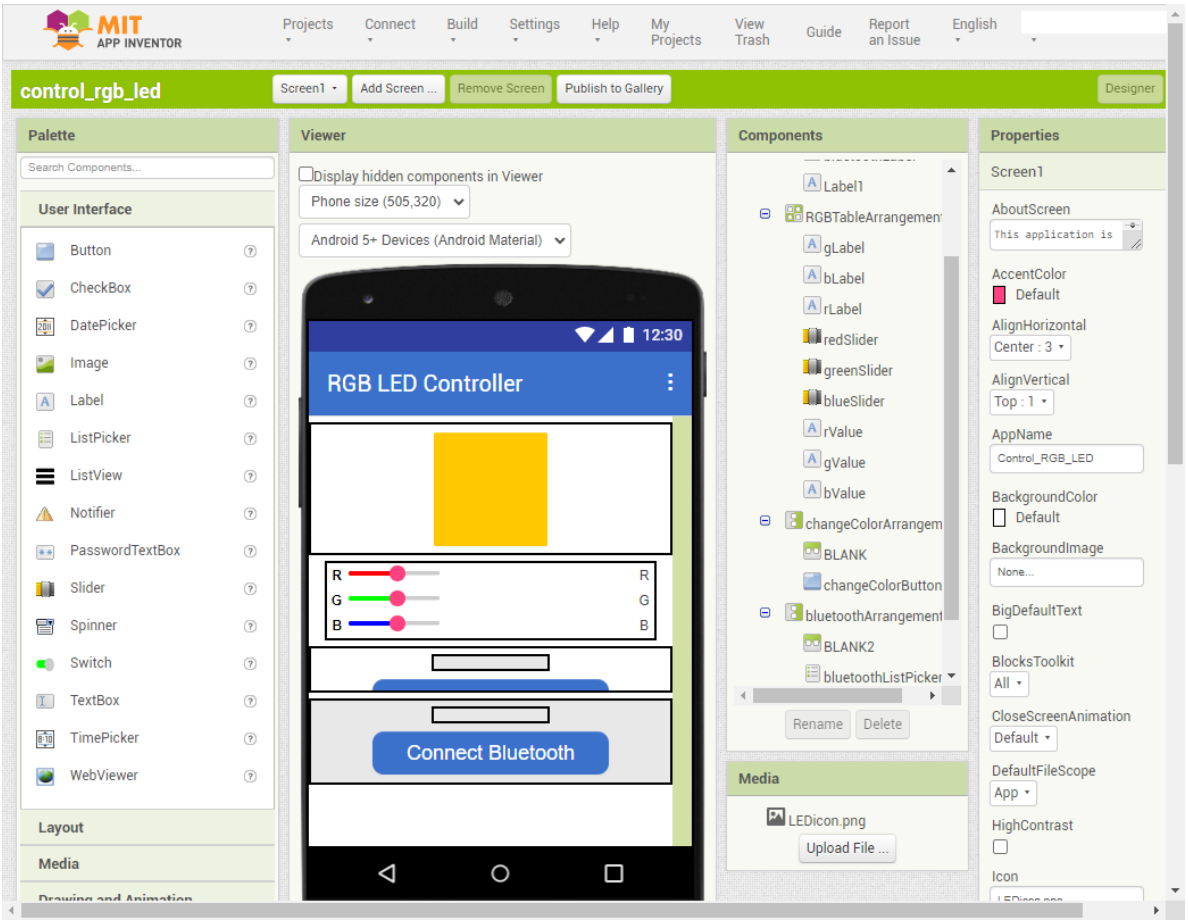
3. Upon uploading the `.aia` file, you will see the application on the **MIT App Inventor** software. This is a pre-configured template. You can modify this template after you have familiarized yourself with **MIT App Inventor** through the following steps.



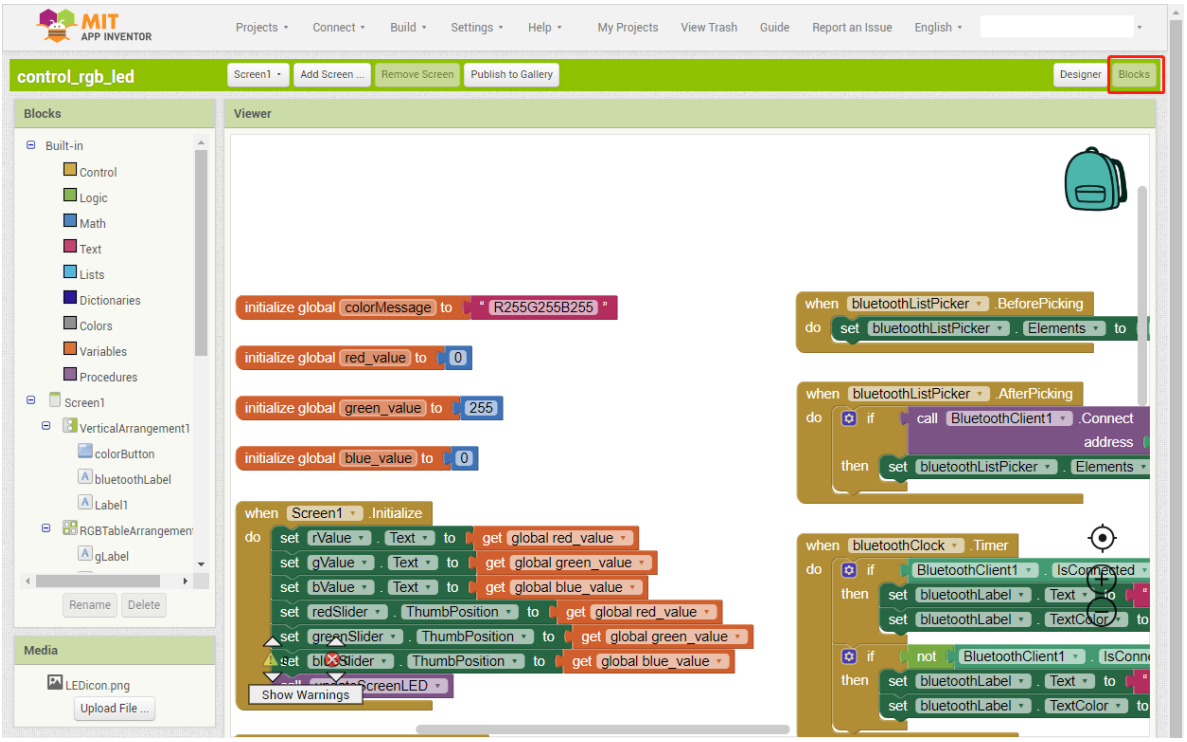
4. In **MIT App Inventor**, you have 2 primary sections: the **Designer** and the **Blocks**.



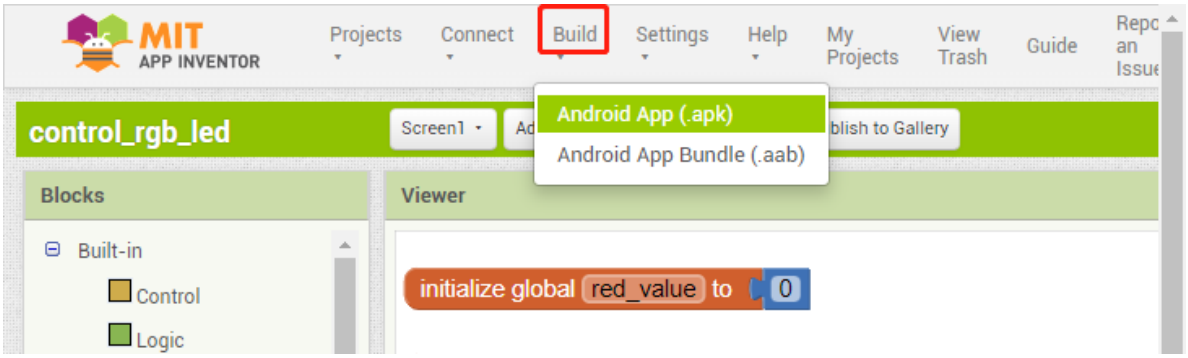
5. The **Designer** allows you to add buttons, text, screens, and modify the overall aesthetic of your application.



6. Subsequently, you have the **Blocks** section. The **Blocks** section facilitates the creation of bespoke functions for your application.



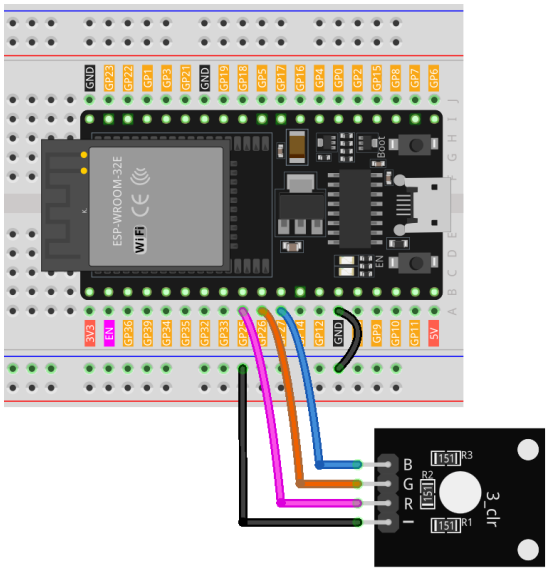
7. To install the application on a smartphone, navigate to the **Build** tab.



- You can generate a `.apk` file. After selecting this option, a page will appear allowing you to choose between downloading a `.apk` file or scanning a QR code for installation. Follow the installation guide to complete the application installation.
- If you wish to upload this app to **Google Play** or another app marketplace, you can generate a `.aab` file.

2. Upload the code

1. Build the circuit.

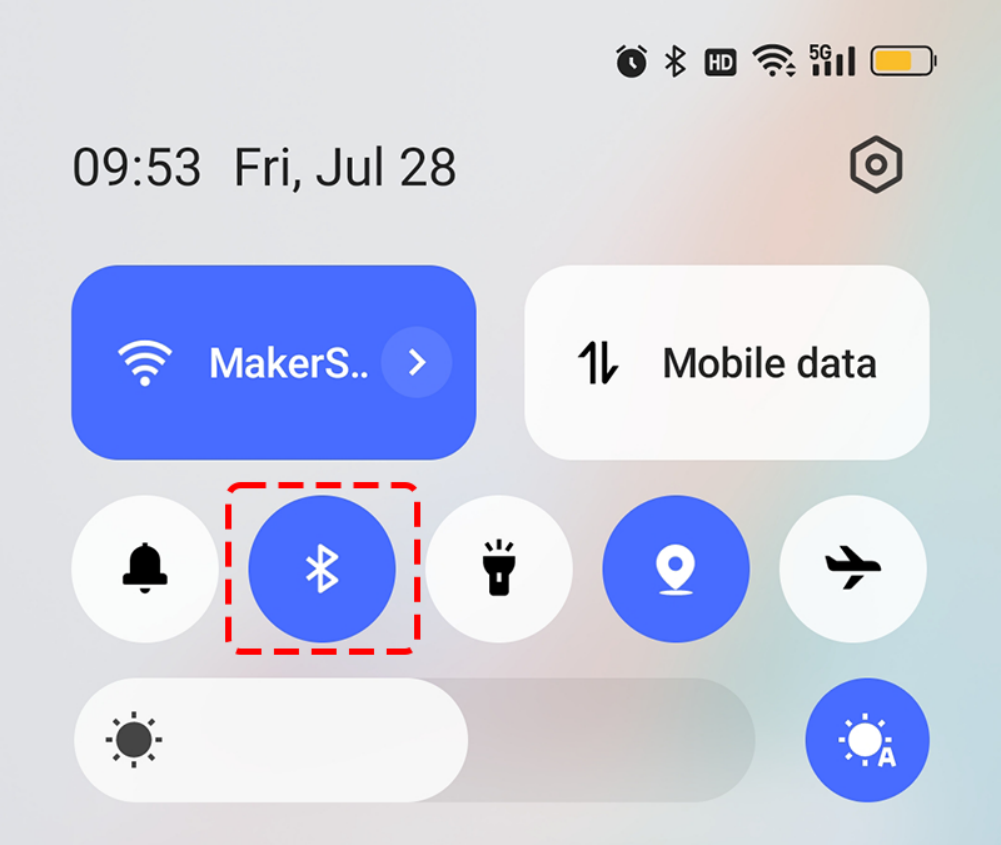


2. Subsequently, connect the ESP32 to your computer using a USB cable.
3. Open the `Lesson_50_Bluetooth_app_inventor.ino` file situated in the `universal-maker-sensor-kit\esp32\Lesson_50_Bluetooth_app_inventor` directory, or copy the code into the Arduino IDE.
4. Upon selecting the appropriate board (**ESP32 Dev Module**) and port, click the **Upload** button.

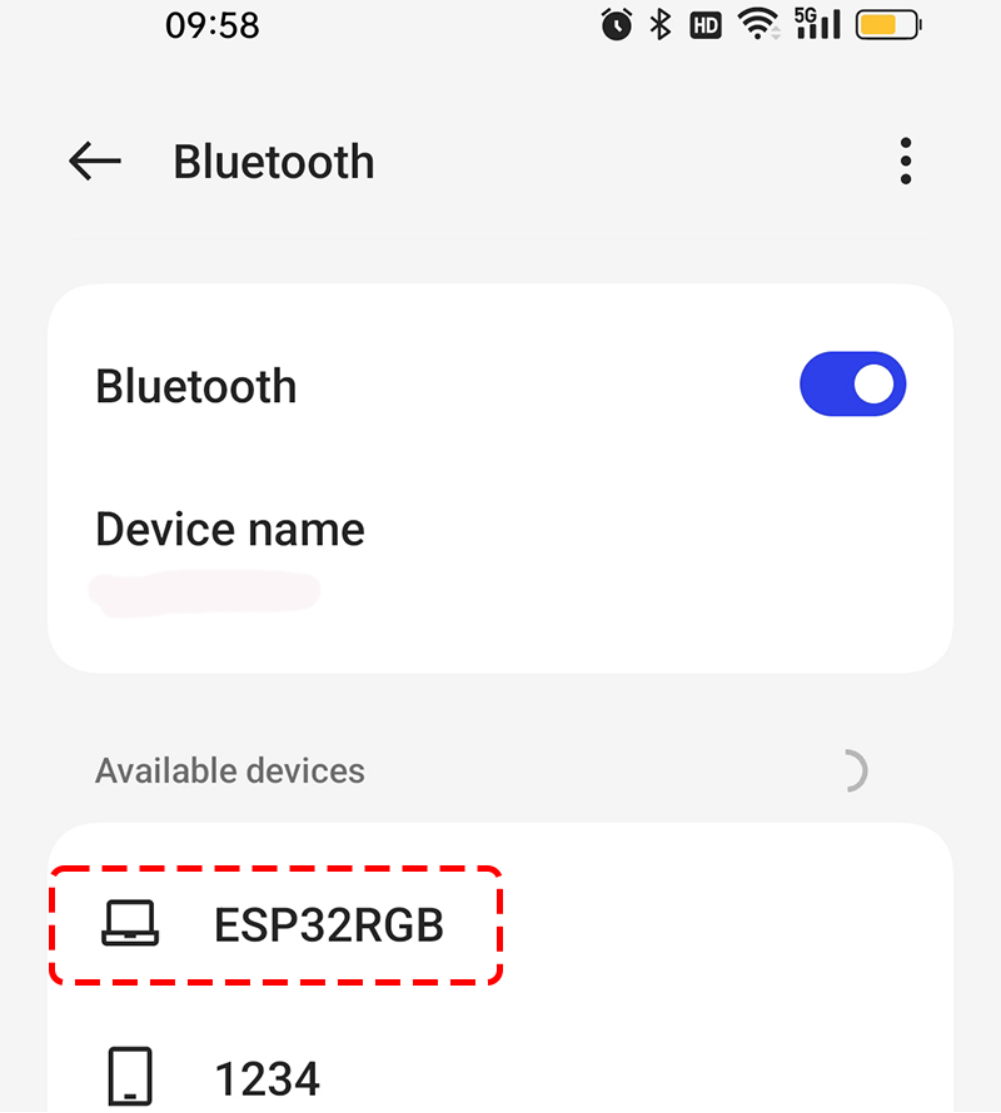
3. App and ESP32 Connection

Ensure that the application created earlier is installed on your smartphone.

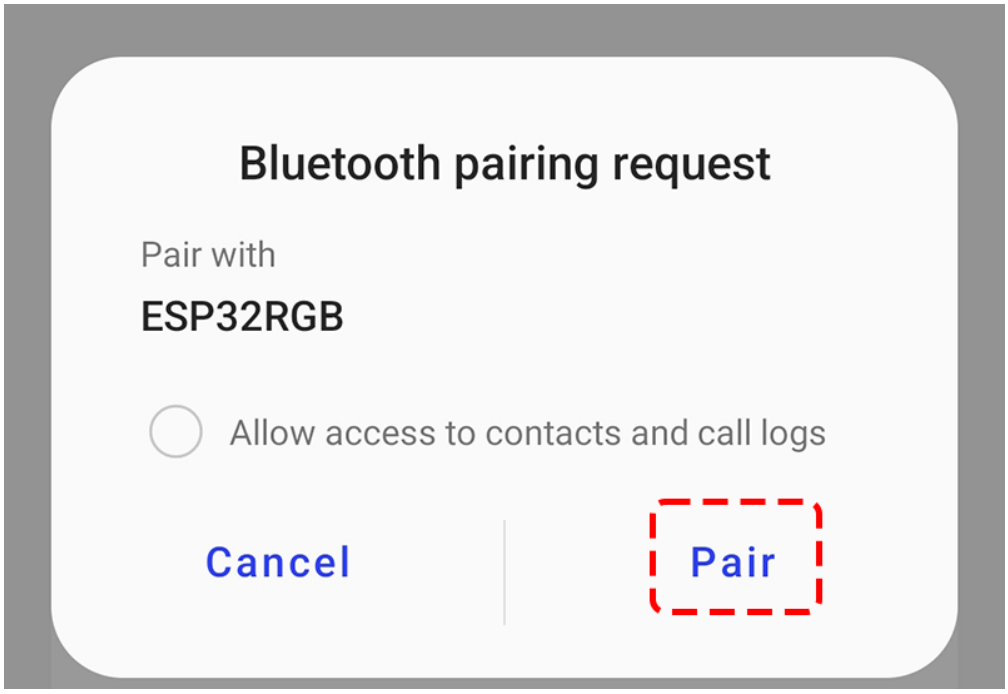
1. Initially, activate **Bluetooth** on your smartphone.



2. Navigate to the **Bluetooth settings** on your smartphone and find **ESP32RGB**.

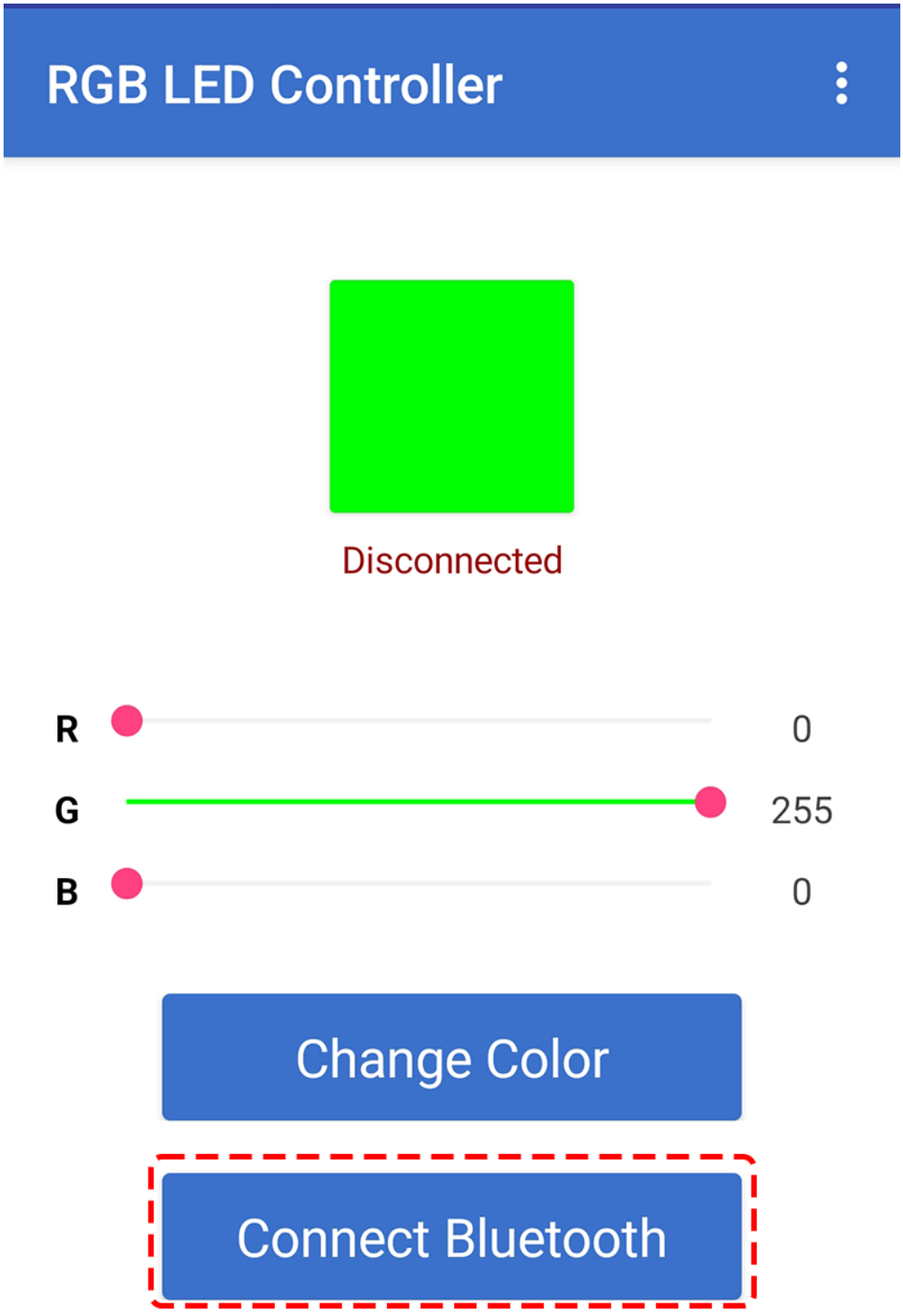


3. After clicking it, agree to the **Pair** request in the pop-up window.

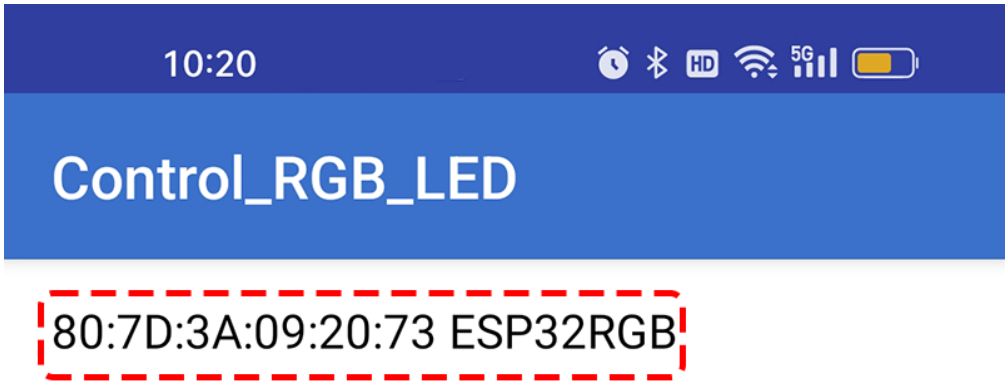


4. Now open the recently installed **Control_RGB_LED** APP.

5. In the APP, click on **Connect Bluetooth** to establish a connection between the APP and ESP32.

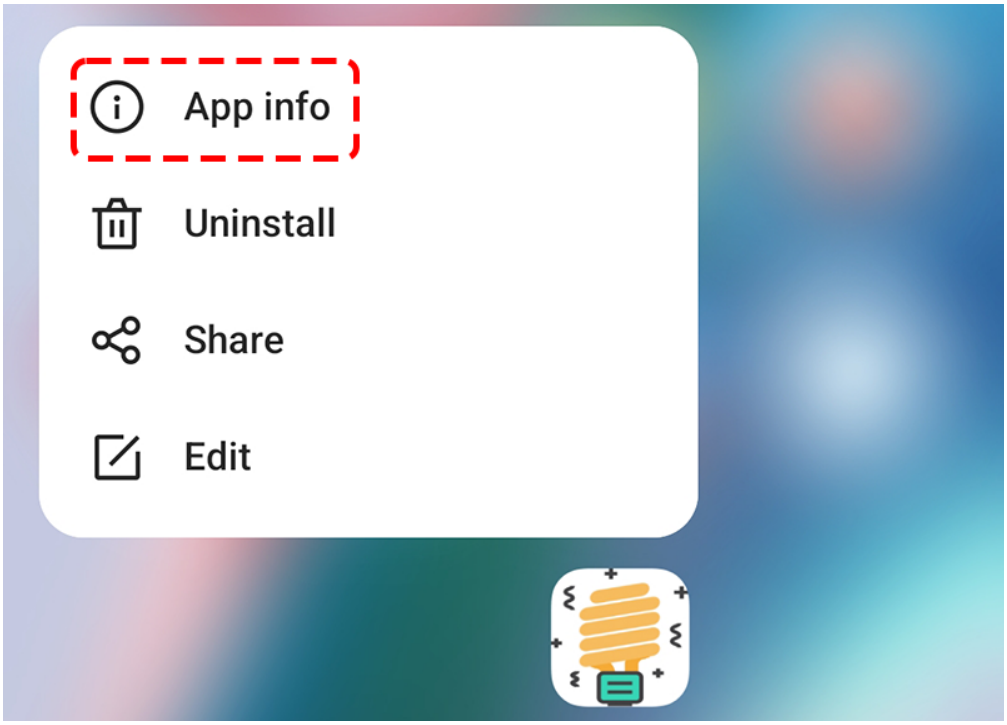


6. Select the `xx.xx.xx.xx.xx.xx ESP32RGB` that comes up. if you changed `SerialBT.begin("ESP32RGB");` in the code, then just select the name of your setting.

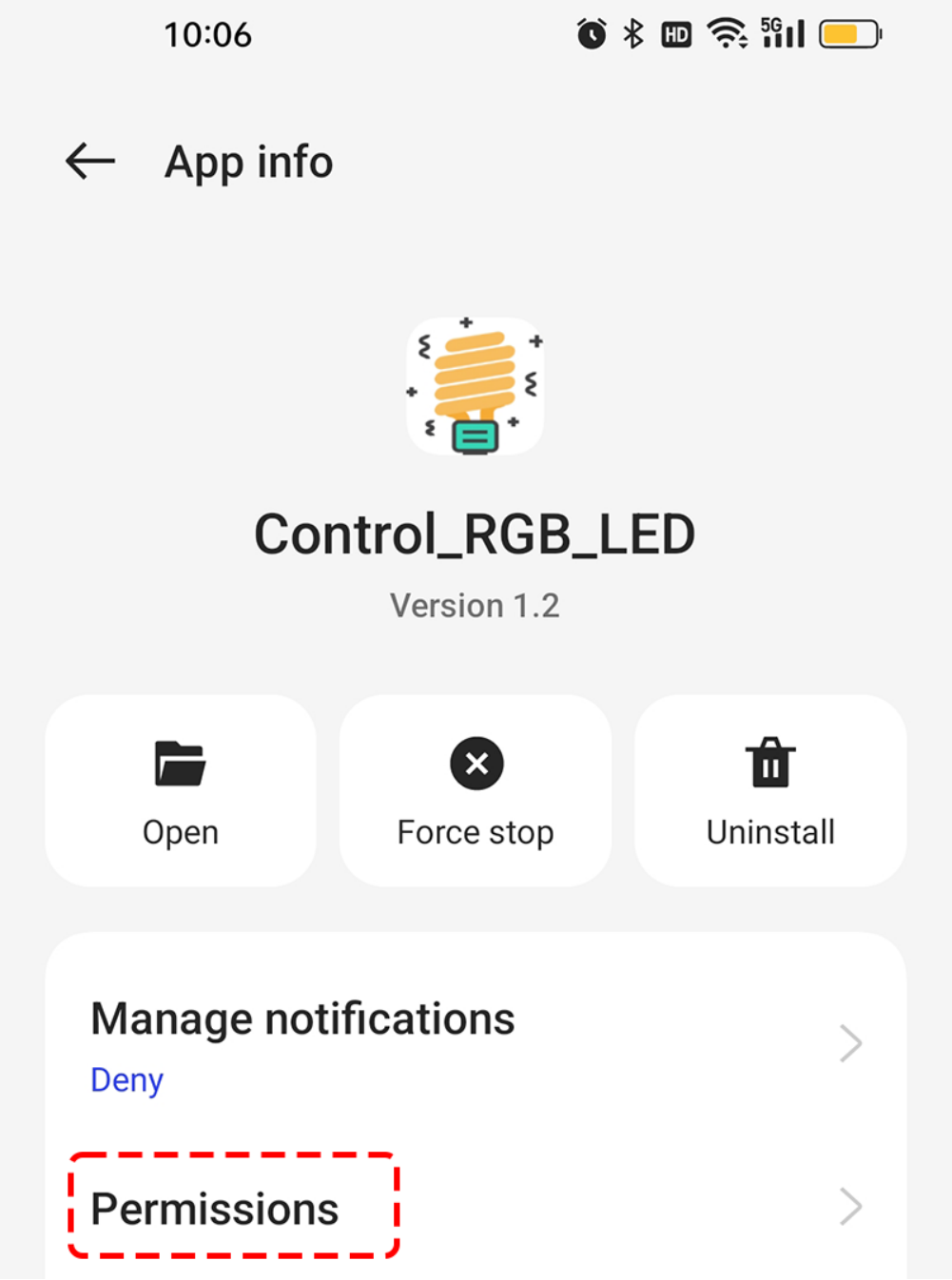


7. If you have been waiting for a while and still can't see any device names, it may be that this APP is not allowed to scan surrounding devices. In this case, you need to adjust the settings manually.

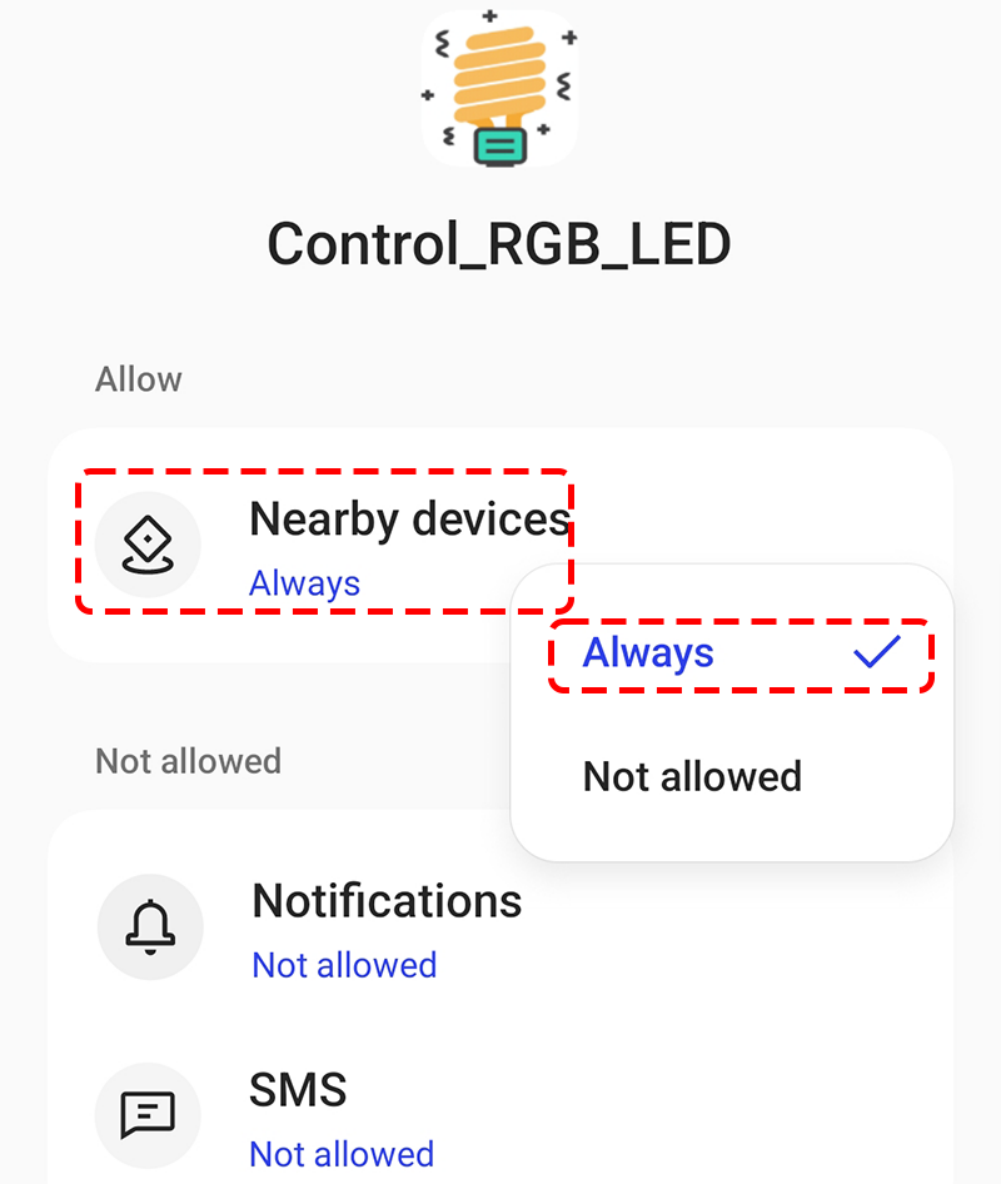
    - Long press the APP icon and click on the resulting **APP Info**. If you have another method to access this page, follow that.



    - Navigate to the **Permissions** page.

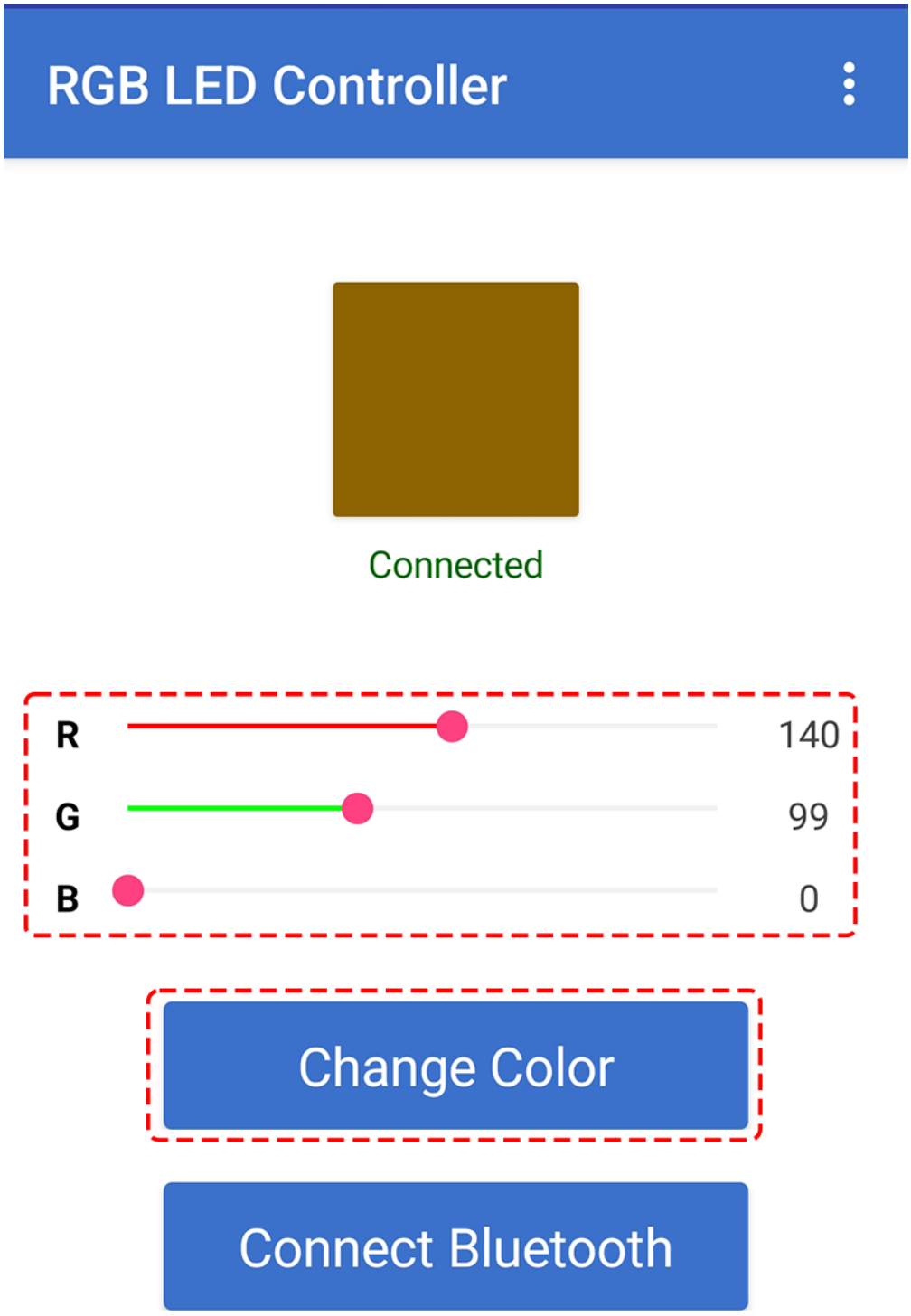


- Locate **Nearby devices**, and select **Always** to allow this APP to scan for nearby devices.



- Now, restart the APP and repeat steps 5 and 6 to successfully connect to Bluetooth.

8. Upon successful connection, you will automatically return to the main page, where it will display connected. Now you can adjust the RGB values and change the color of the RGB display by pressing the **Change Color** button.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5 For Raspberry Pi Pico W

Raspberry Pi Pico W users, please check out the following tutorial. This tutorial is suitable for We will use MicroPython to program the Pi Pico W and write a simple code example for each component to help you get started quickly.

MicroPython is a version of the Python programming language for microcontrollers, such as your Raspberry Pi Pico W. MicroPython lets you use your Python knowledge to write code to interact with electronics components.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.1 Getting Started with MicroPython

In this section, you will learn about the history of MicroPython, how to install MicroPython on Pico W, its basic syntax, and a dozen interesting and practical projects to help you quickly grasp MicroPython.

We recommend that you read the chapters in order.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Introduction of MicroPython

MicroPython is a software implementation of a programming language largely compatible with Python 3, written in C, that is optimized to run on a microcontroller.

MicroPython consists of a Python compiler to bytecode and a runtime interpreter of that bytecode. The user is presented with an interactive prompt (the REPL) to execute supported commands immediately. Included are a selection of core Python libraries; MicroPython includes modules which give the programmer access to low-level hardware.

- Reference: MicroPython - Wikipedia

The Story Starts Here

Things changed in 2013 when Damien George launched a crowdfunding campaign (Kickstarter).

Damien was an undergraduate student at Cambridge University and an avid robotics programmer. He wanted to reduce the world of Python from a gigabyte machine to a kilobyte. His Kickstarter campaign was to support his development while he turned his proof of concept into a finished implementation.

MicroPython is supported by a diverse Pythonista community that has a keen interest in seeing the project succeed.

Apart from testing and supporting the code base, the developers provided tutorials, code libraries, and hardware porting, so Damien was able to focus on other aspects of the project.

- Reference: realpython

Why MicroPython

Although the original Kickstarter campaign released MicroPython as a development board "pyboard" with STM32F4, MicroPython supports many ARM-based product architectures. The mainline supported ports are ARM Cortex-M (many STM32 boards, TI CC3200/WiPy, Teensy boards, Nordic nRF series, SAMD21 and SAMD51), ESP8266, ESP32, 16bit PIC, Unix, Windows, Zephyr and JavaScript. Second, MicroPython allows for fast feedback. This is because you can use REPL to enter commands interactively and get responses. You can even tweak code and run it immediately instead of traversing the code-compile-upload-execute cycle.

While Python has the same advantages, for some Microcontroller boards like the Raspberry Pi Pico, they are small, simple and have little memory to run the Python language at all. That's why MicroPython has evolved, keeping the main Python features and adding a bunch of new ones to work with these Microcontroller boards.

Next you will learn to install MicroPython into the Raspberry Pi Pico.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

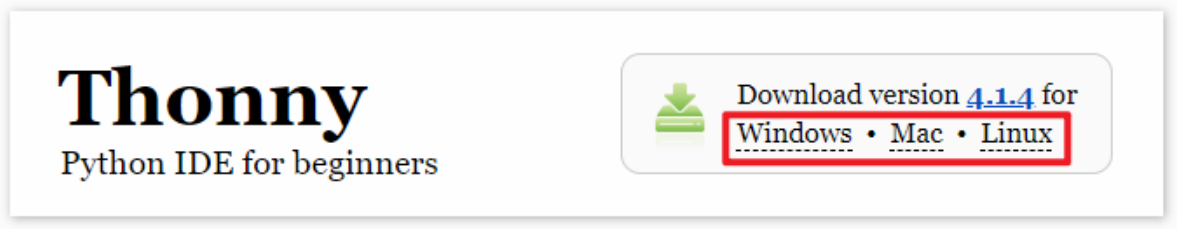
Install Thonny IDE

Before you can start to program Pico with MicroPython, you need an integrated development environment (IDE), here we recommend Thonny. Thonny comes with Python 3.10 built in, just one simple installer is needed and you're ready to learn programming.

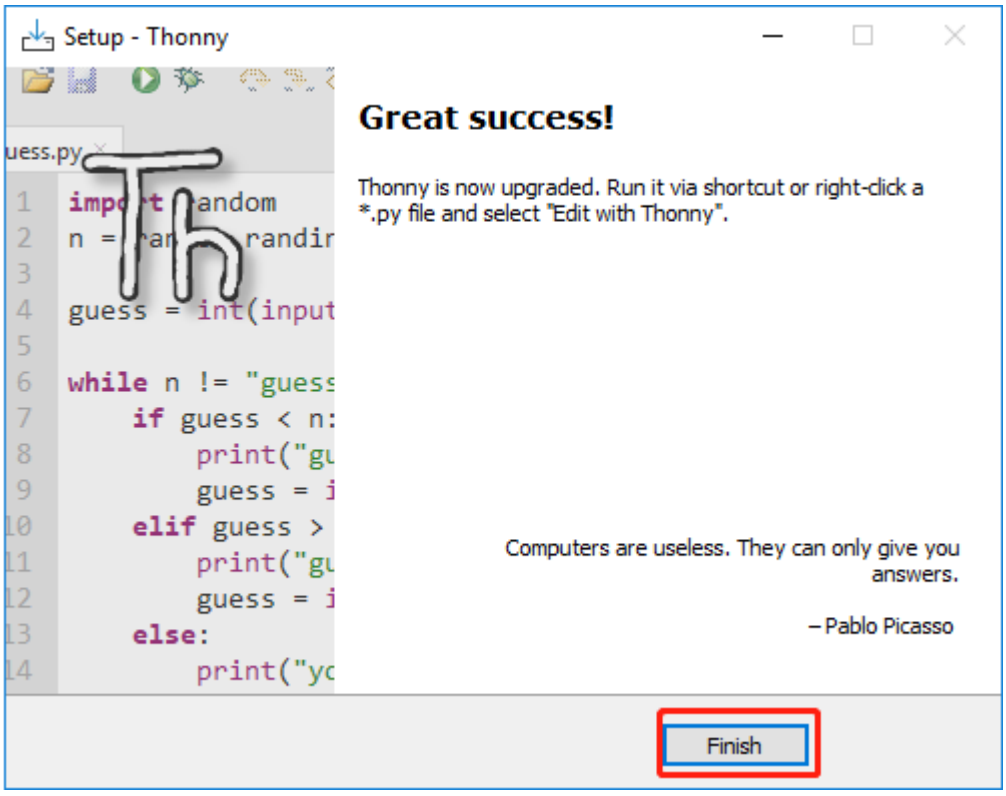
Note: If you already have Thonny installed, make sure it is version 3.3.3 or later, as the Raspberry Pi Pico interpreter is only compatible with these versions; if not, please update or install the appropriate version.

---

1. You can download it by visiting the website. Once open the page, you will see a light gray box in the upper right corner, click on the link that applies to your operating system.



2. After downloading, run the installer to begin the installation process. Then click "Next" followed by "Install" to complete the Thonny installation.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

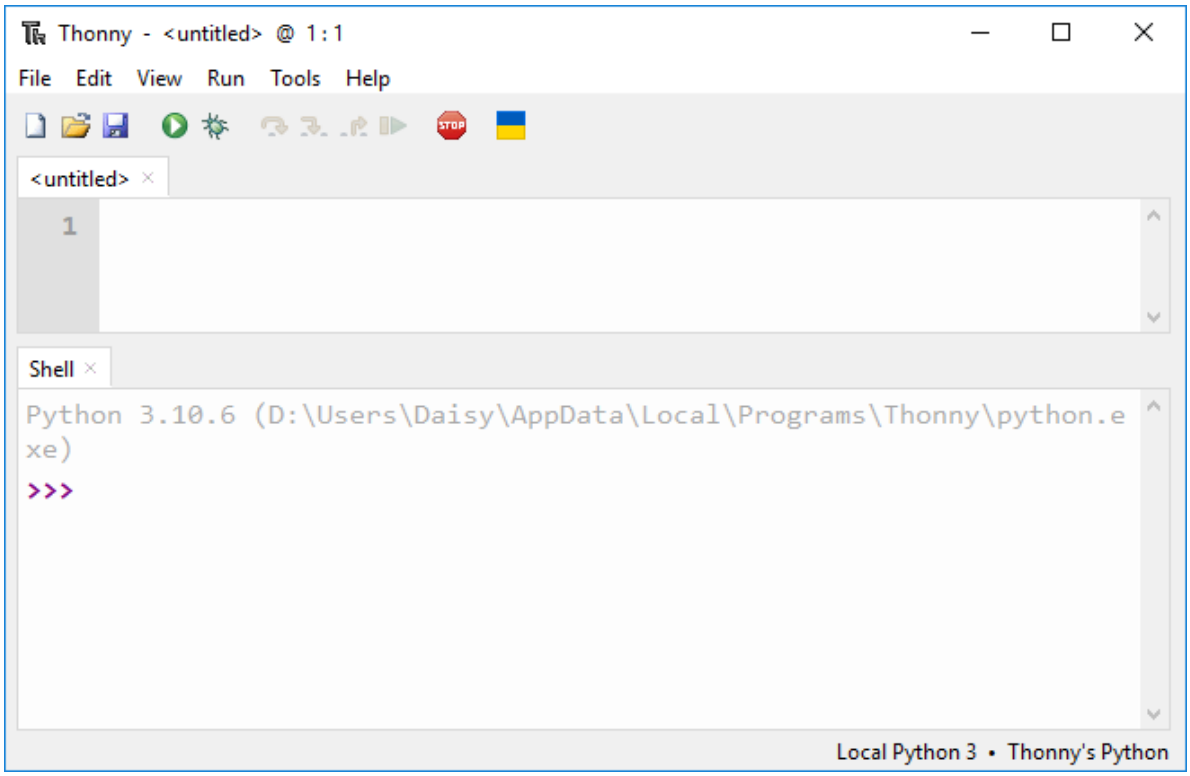
Install MicroPython on Your Pico

Now come to install MicroPython into Raspberry Pi Pico, Thonny IDE provides a very convenient way for you to install it with one click.

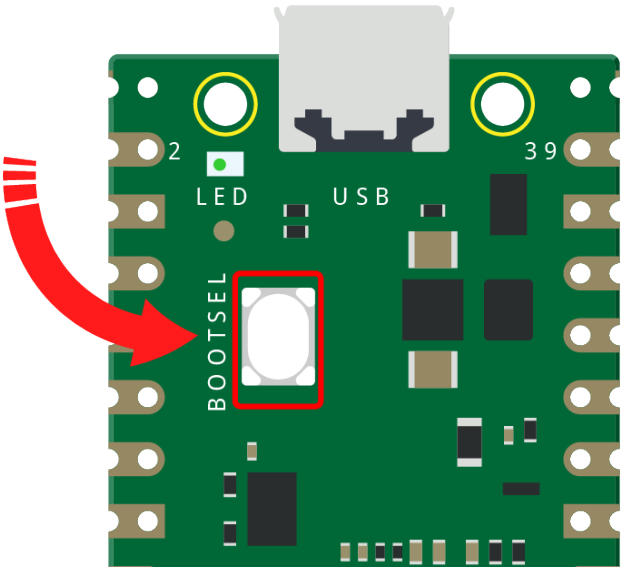
Note: You can also use the provided by Raspberry Pi official by dragging and dropping an `rp2_pico_xxxx.uf2` file into Raspberry Pi Pico.

---

1. Open Thonny IDE.

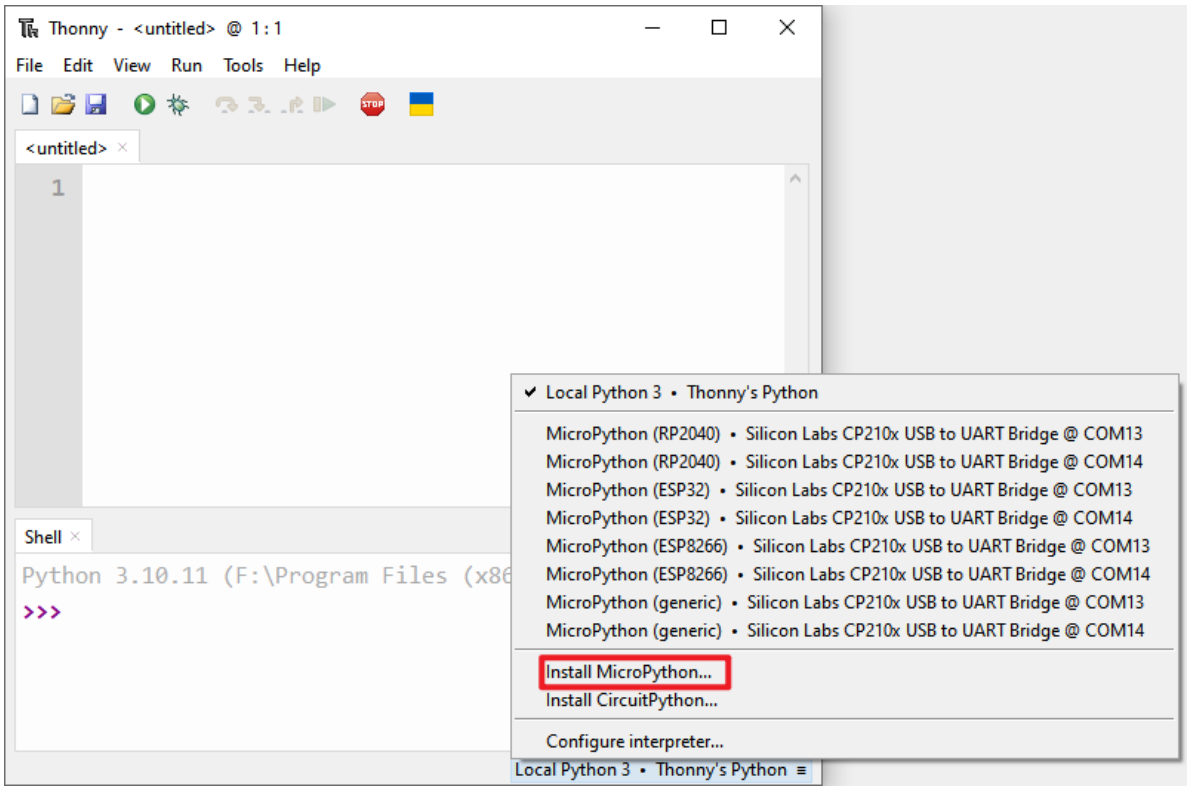


2. Press and hold the **BOOTSEL** button and then connect the Pico to computer via a Micro USB cable. Release the **BOOTSEL** button after your Pico is mount as a Mass Storage Device called **RPI-RP2**.

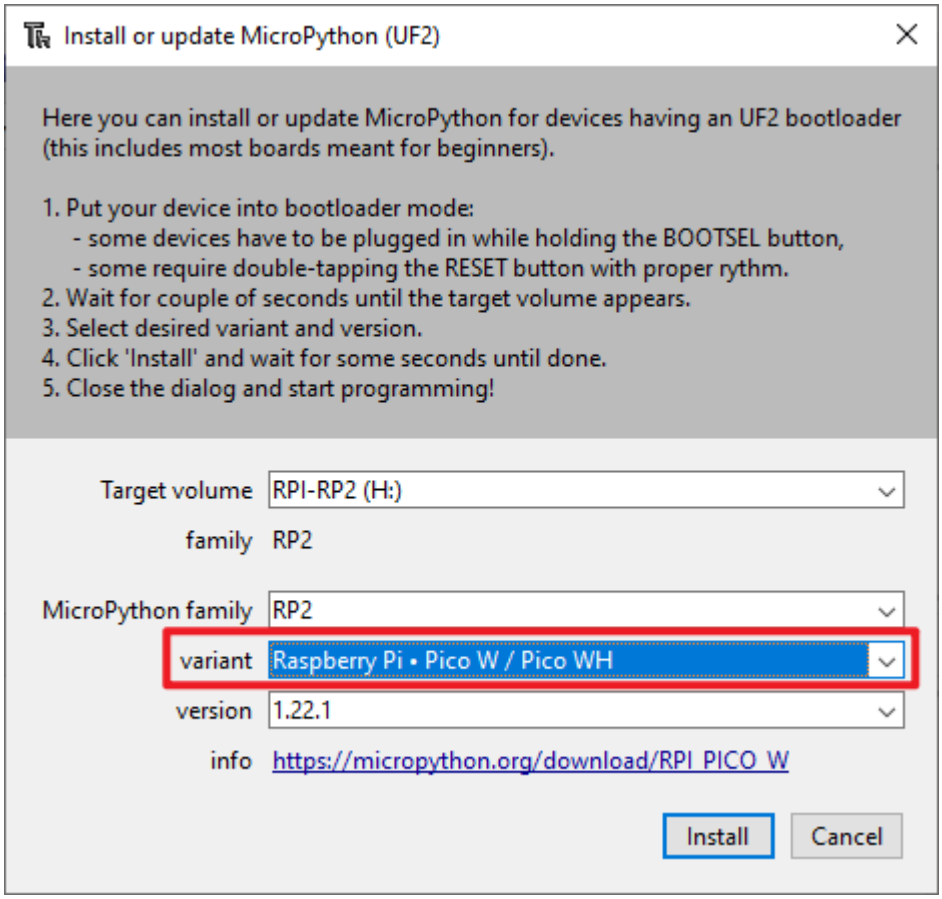


3. In the bottom right corner, click the interpreter selection button and select **Install Micropython**.

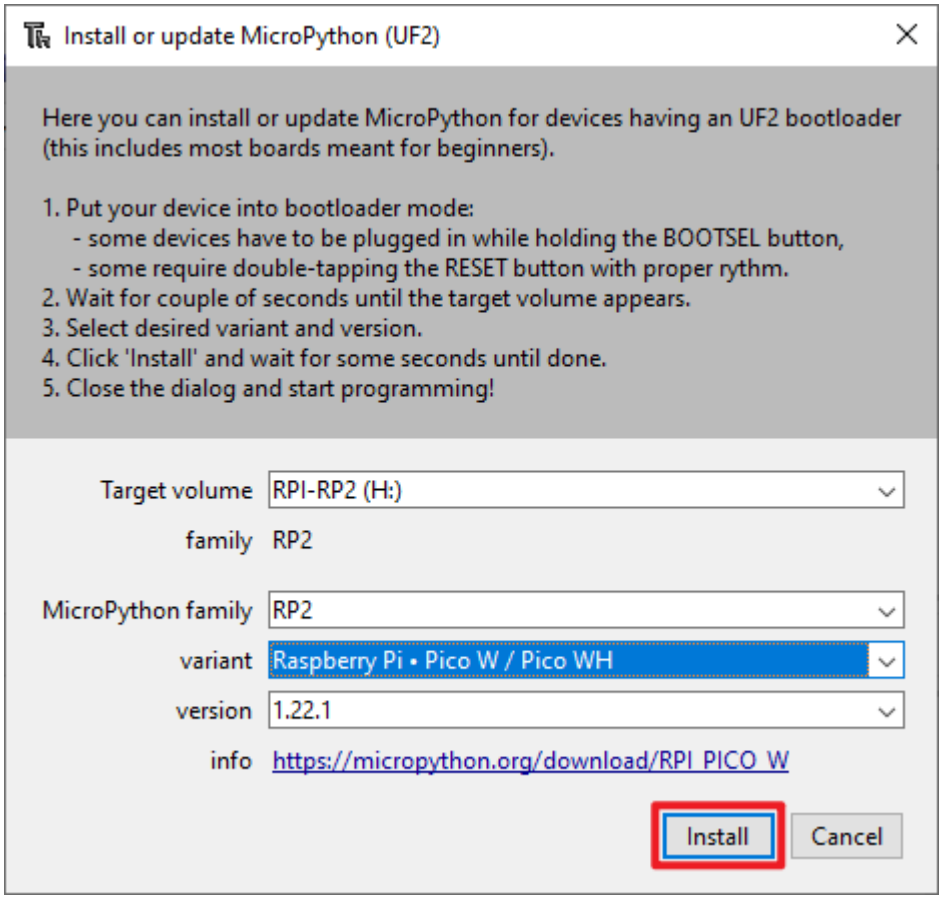
Note: If your Thonny does not have this option, please update to the latest version.



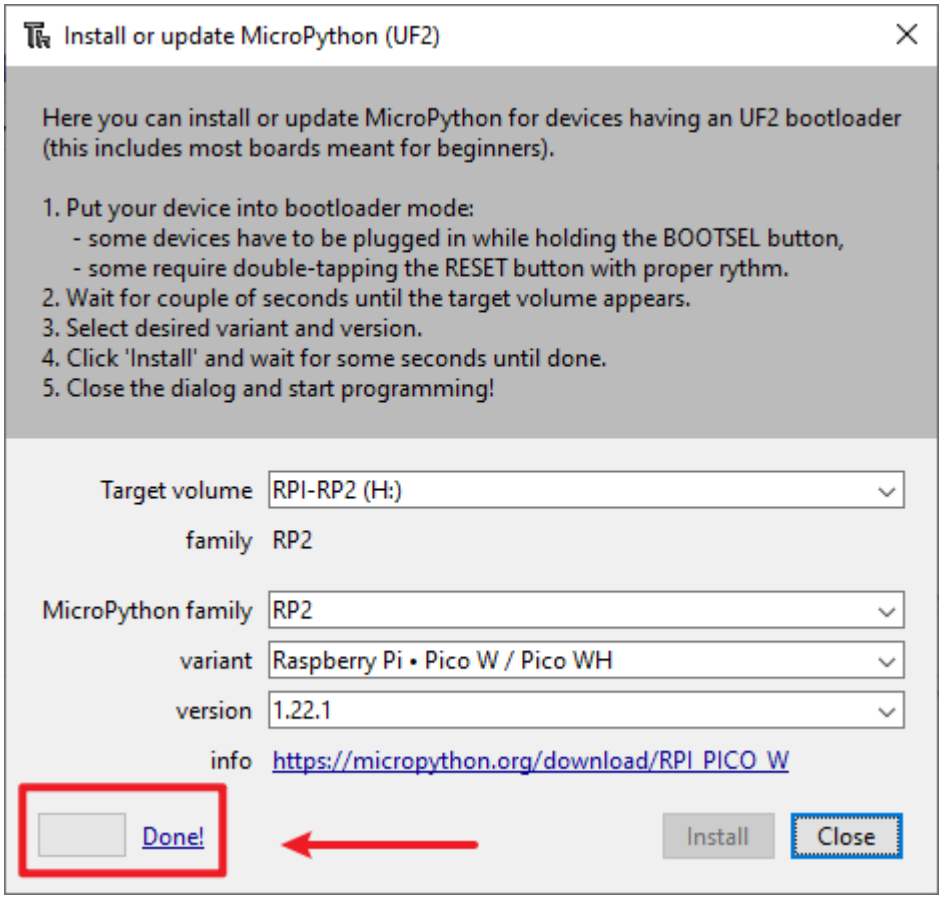
4. In the **Target volume** section, the volume of the Pico you just plugged in will automatically appear. In the **variant** section, select **Raspberry Pi.Pico/Pico H**. Select the latest version in the version dropdown menu.



5. Click the **Install** button, wait for the installation to complete.



6. Congratulations, now your Raspberry Pi Pico is ready to go.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

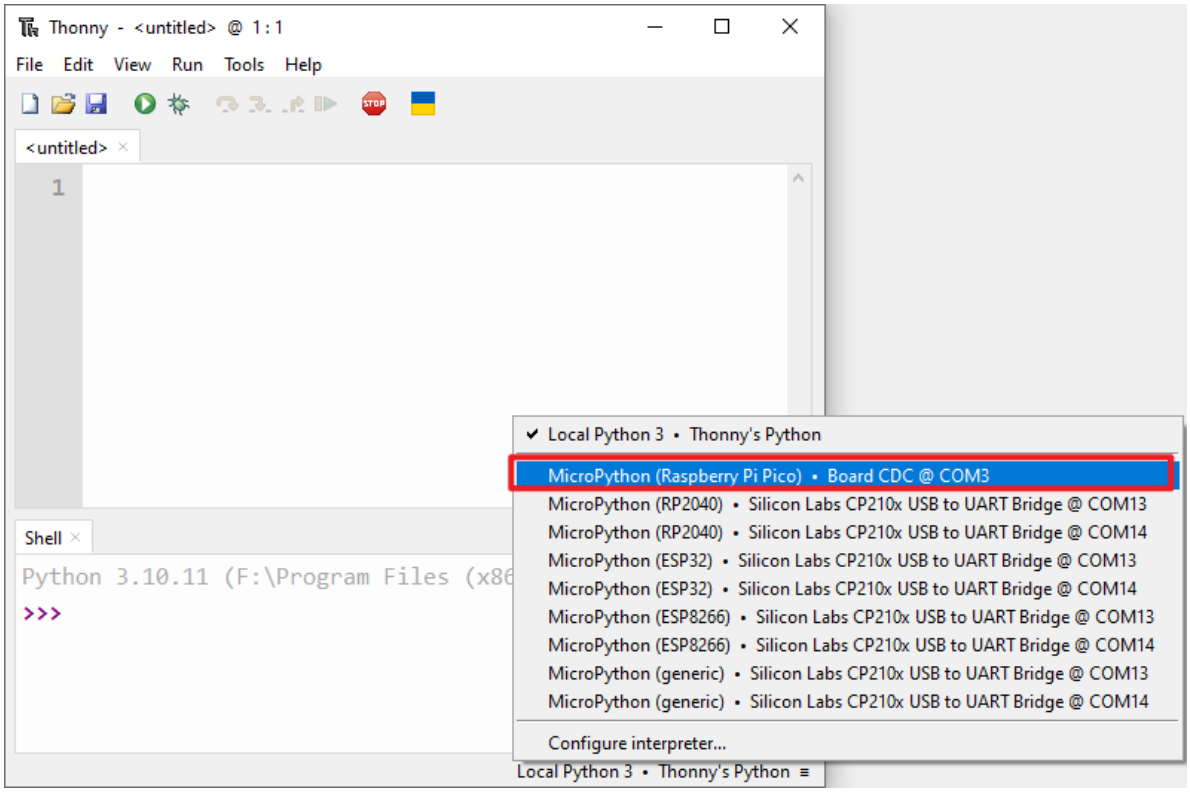
Upload the Libraries to Pico

In some projects, you will need additional libraries. So here we upload these libraries to Raspberry Pi Pico W first, and then we can run the code directly later.

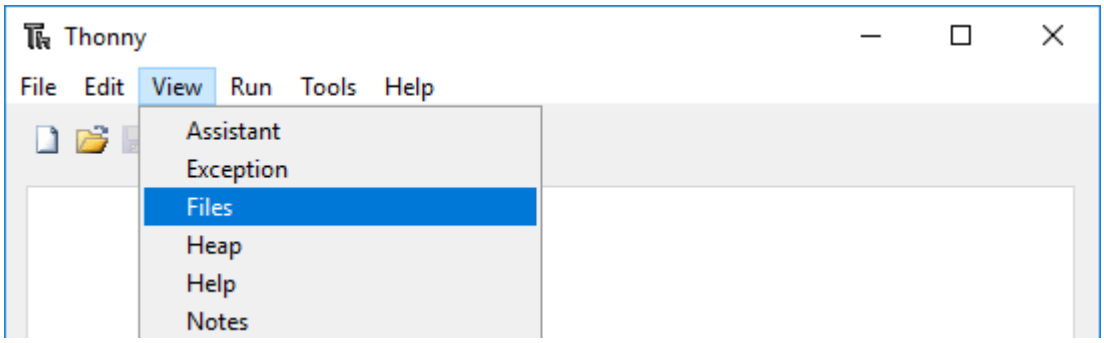
1. Download the relevant code from the link below.

    - `SunFounder Universal Maker Sensor Kit`

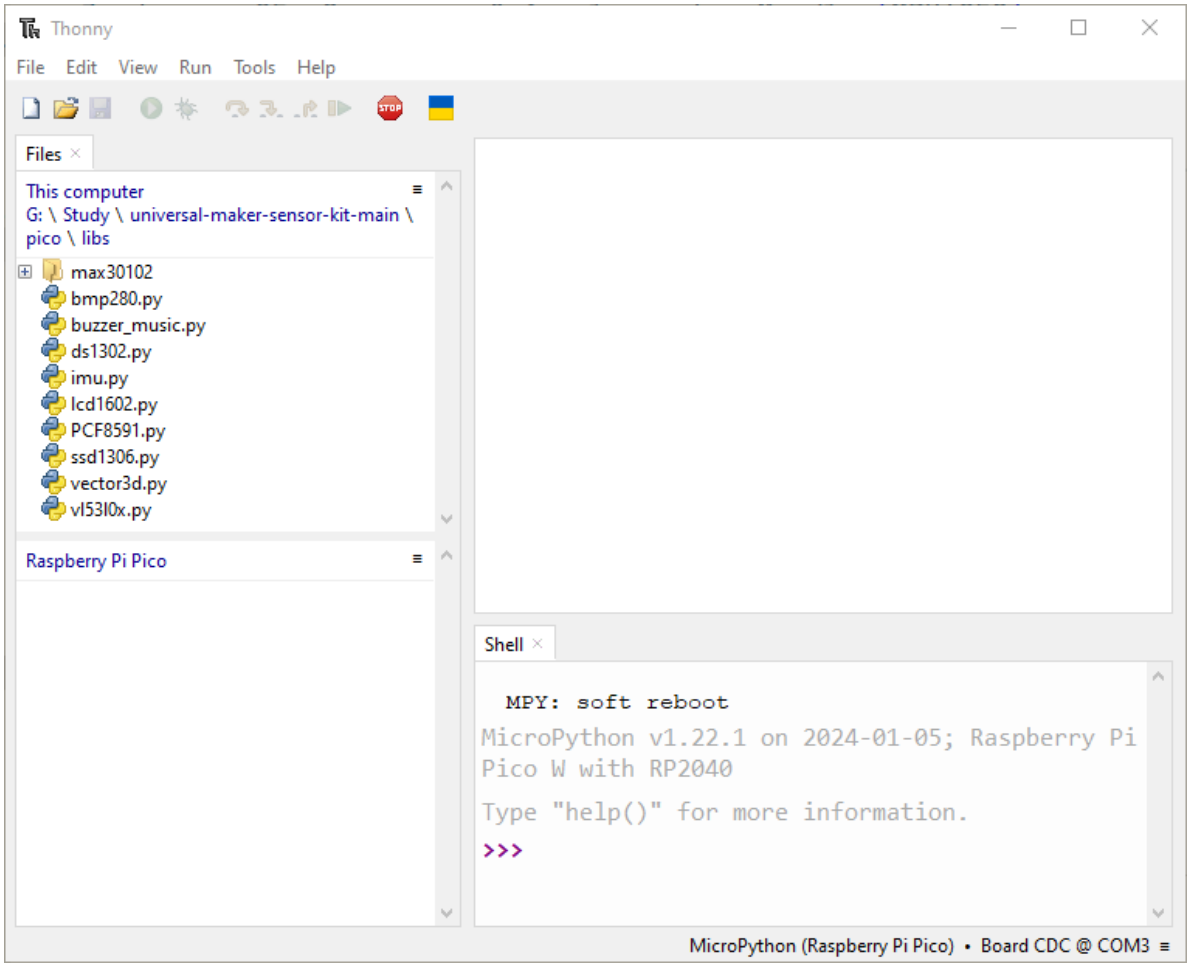
2. Open Thonny IDE and plug the Pico into your computer with a micro USB cable and click on the "MicroPython (Raspberry Pi Pico).COMXX" interpreter in the bottom right corner.



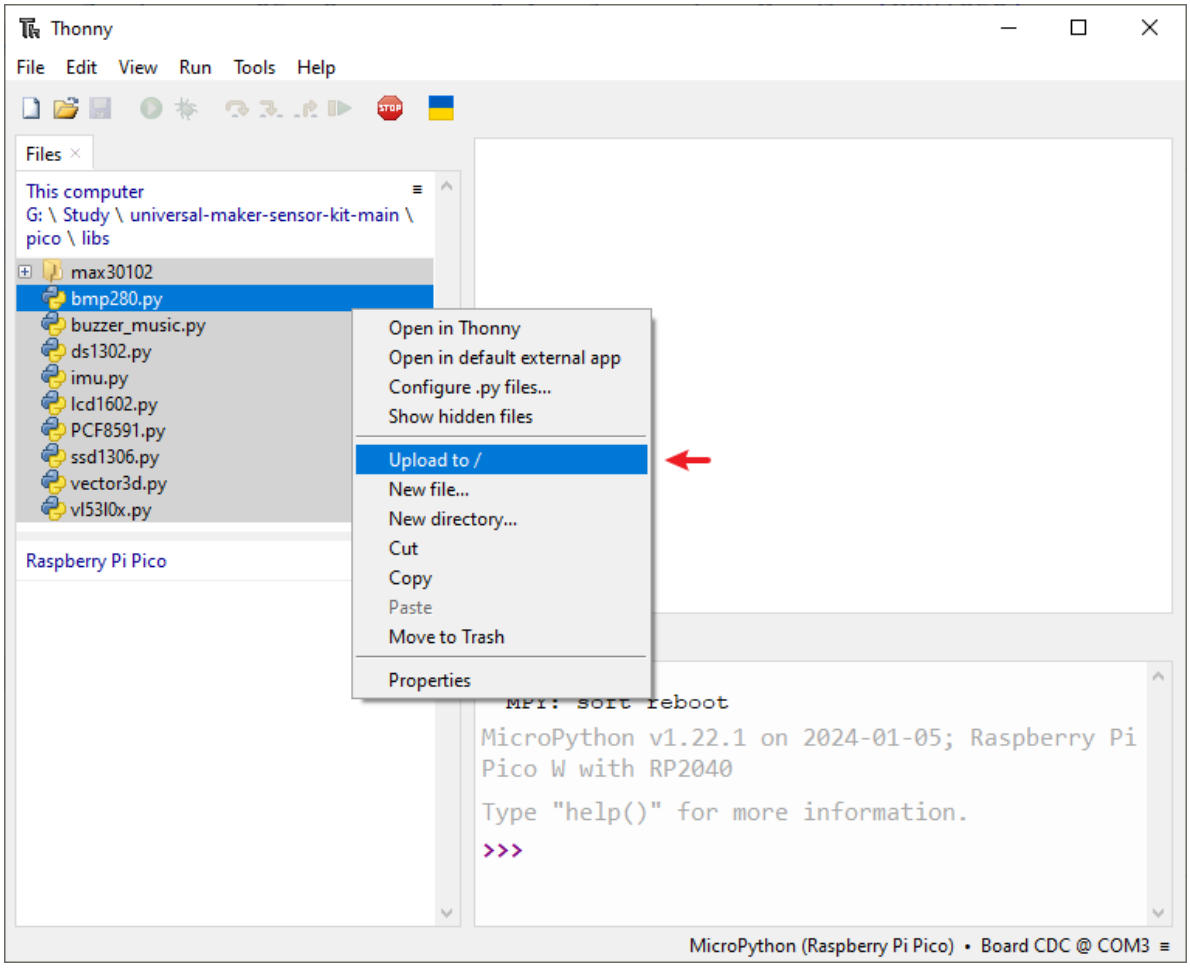
3. In the top navigation bar, click **View** -> **Files**.



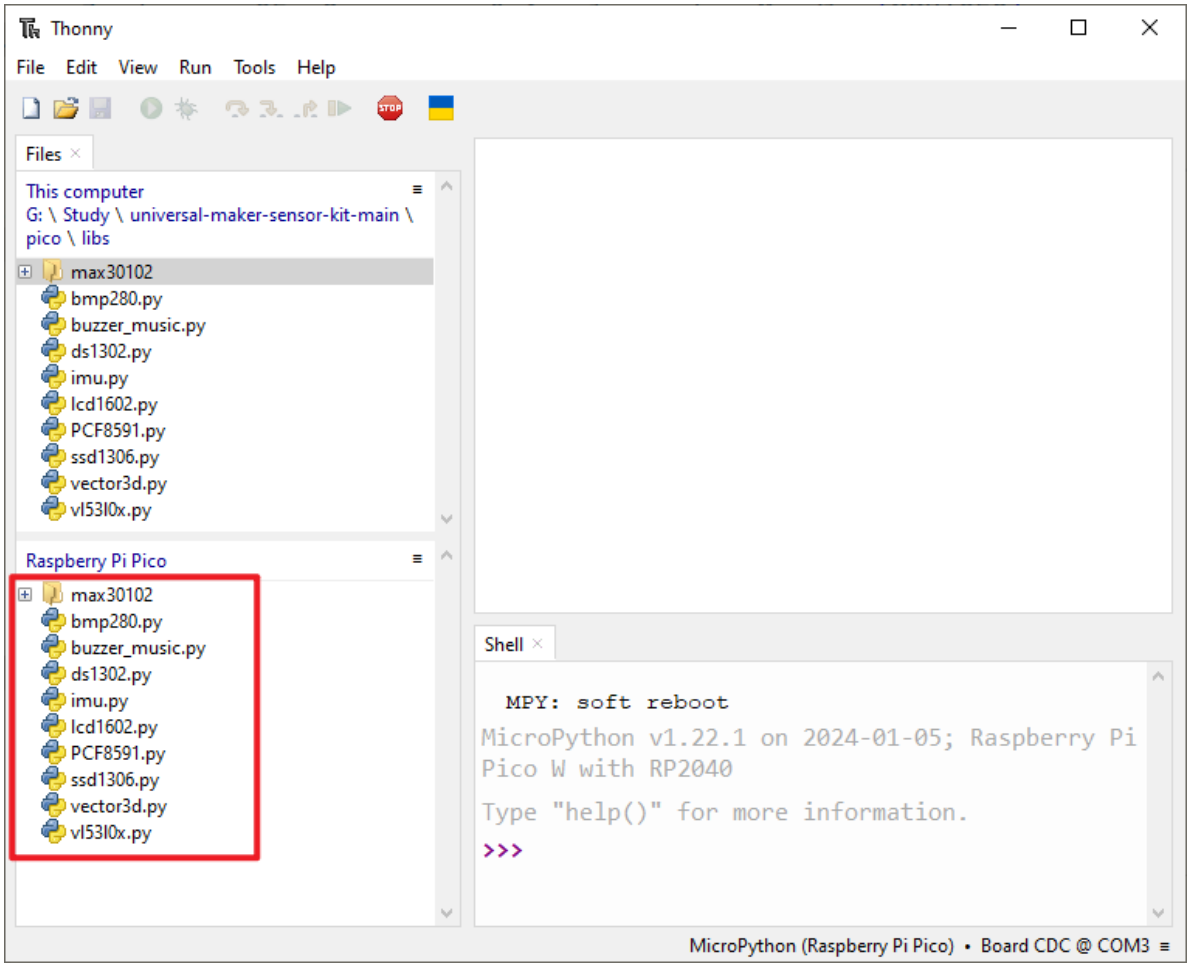
4. Switch the path to the folder where you downloaded the code package before, and then go to the `universal-maker-sensor-kit-main/pico/libs` folder.



5. Select all the files or folders in the "libs/" folder (by holding down Shift and clicking on the first and last file in the folder), then right-click and select **Upload to /**, it will take a while to upload.



6. Now you will see the files you just uploaded inside your drive `Raspberry Pi Pico`.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Quick Guide on Thonny

Open and Run Code Directly

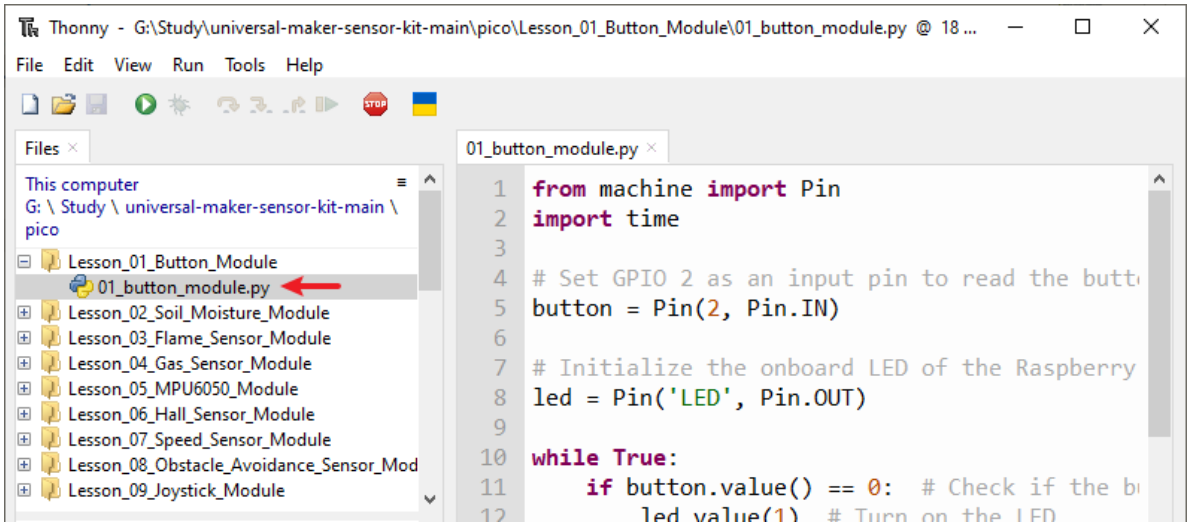
In each project we provide, the specific code used is clearly identified. You can find the corresponding code for each project in the `universal-maker-sensor-kit-main/pico/` directory.

However, you must first download the package and upload the library, as described in *Upload the Libraries to Pico*.

1. Open code.

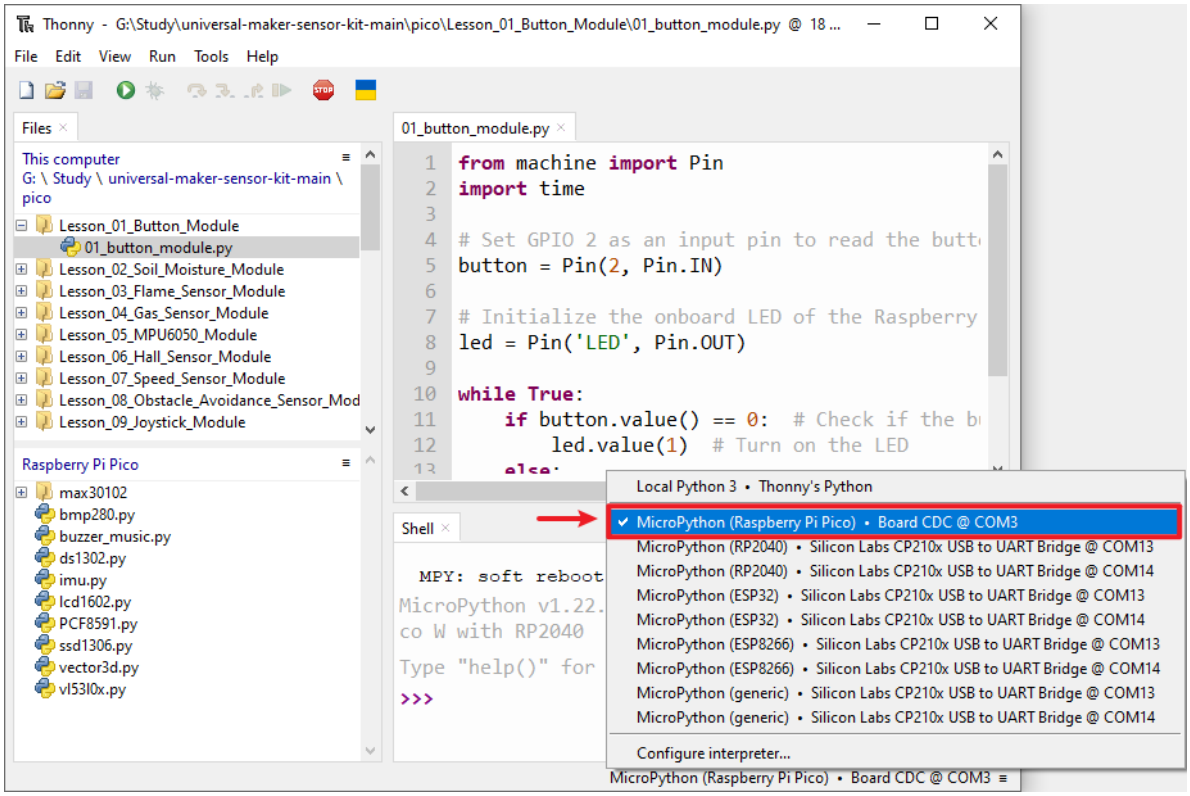
   For example, `Lesson_01_Button_Module\01_button_module.py`.

   If you double click on it, a new window will open on the right. You can open more than one code at the same time.



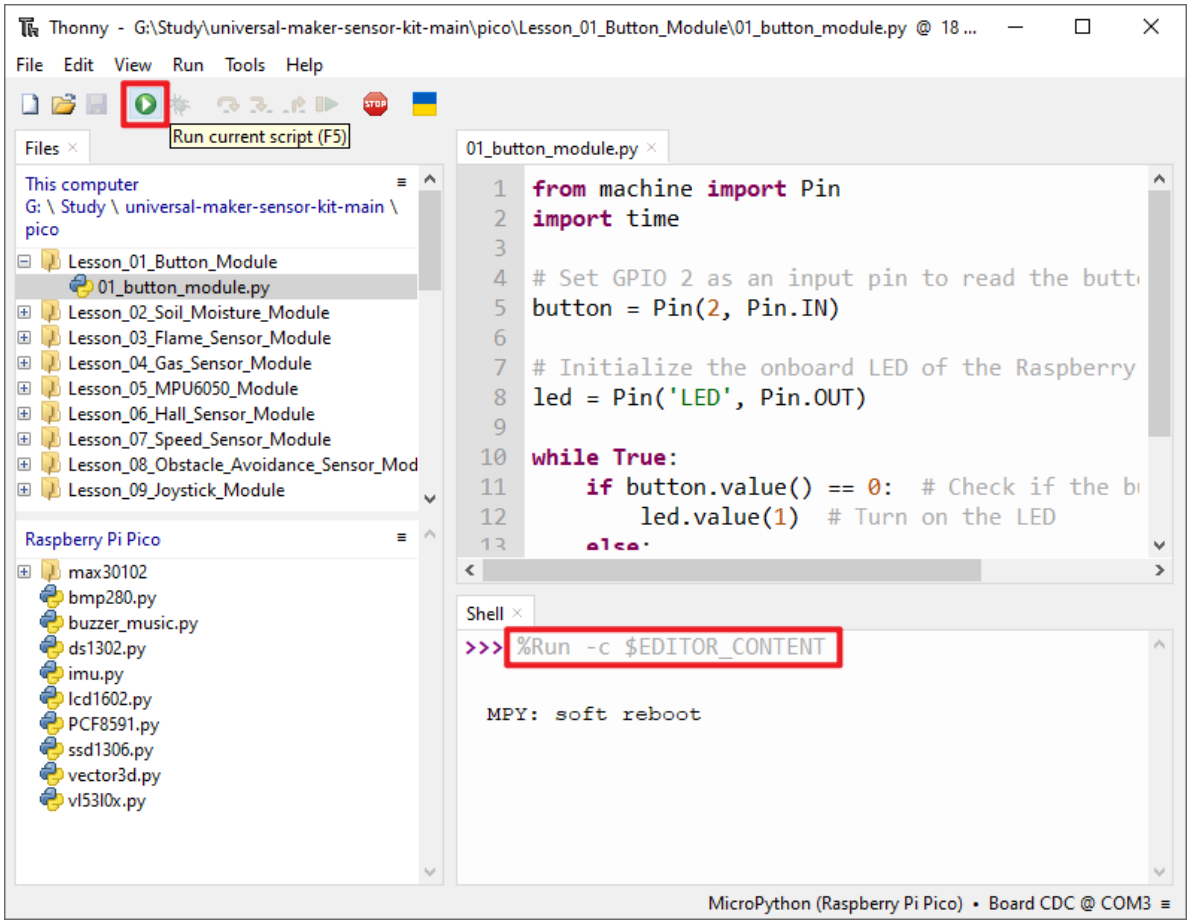
2. Select correct interpreter

   Use a micro USB cable to connect the Pico W to your computer and select the “MicroPython (Raspberry Pi Pico)” interpreter.



3. Run the code

   To run the script, click the **Run current script** button or press F5.



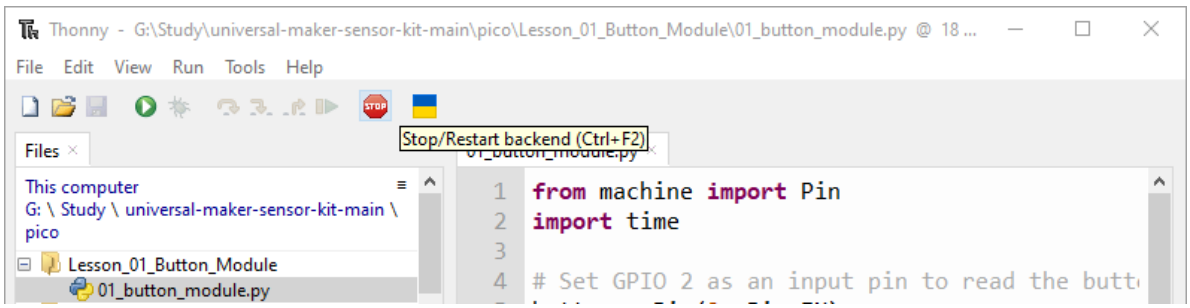
If the code contains any information that needs to be printed, it will appear in the Shell; otherwise, only the following information will appear.

```
>>> %Run -c $EDITOR_CONTENT
```

Click **View** -> **Shell** to open the Shell window if it doesn't appear on your Thonny.

- `%Run -c $EDITOR_CONTENT` is a command from Thonny telling the MicroPython interpreter on your Pico W to run the contents of the script area - "EDITOR_CONTENT".
- If there is any message after that, it is usually the message that you tell MicroPython to print, or an error message for the code.

4. Stop running

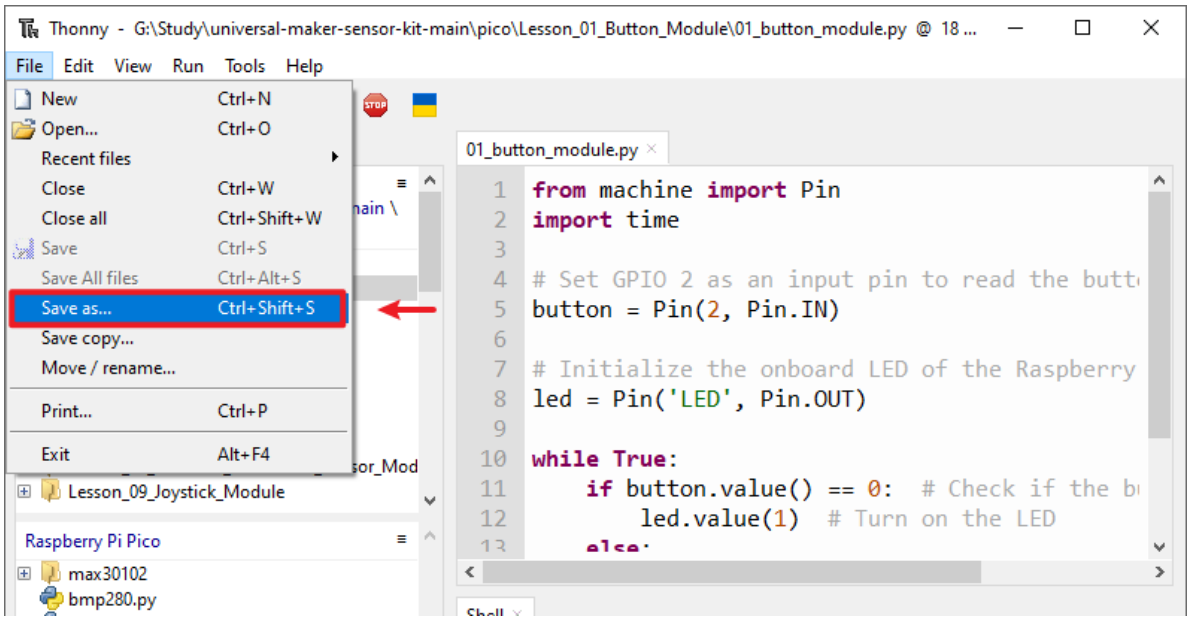


To stop the running code, click the **Stop/Restart backend** button. The **%RUN -c $EDITOR_CONTENT** command will disappear after stopping.

5. Save or save as

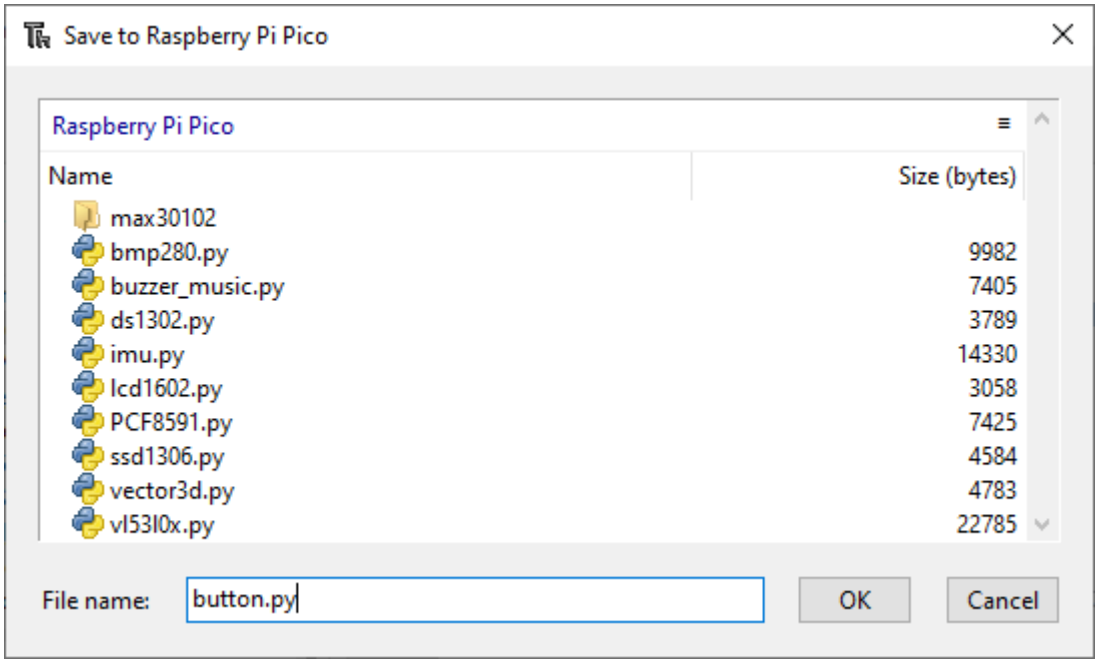
   You can save changes made to the open example by pressing **Ctrl+S** or clicking the **Save** button on Thonny.

   The code can be saved as a separate file within the Raspberry Pi Pico W by clicking on **File** -> **Save As**.



Select **Raspberry Pi Pico**.

Then click **OK** after entering the file name and extension **.py**. On the Raspberry Pi Pico W drive, you will see your saved file.



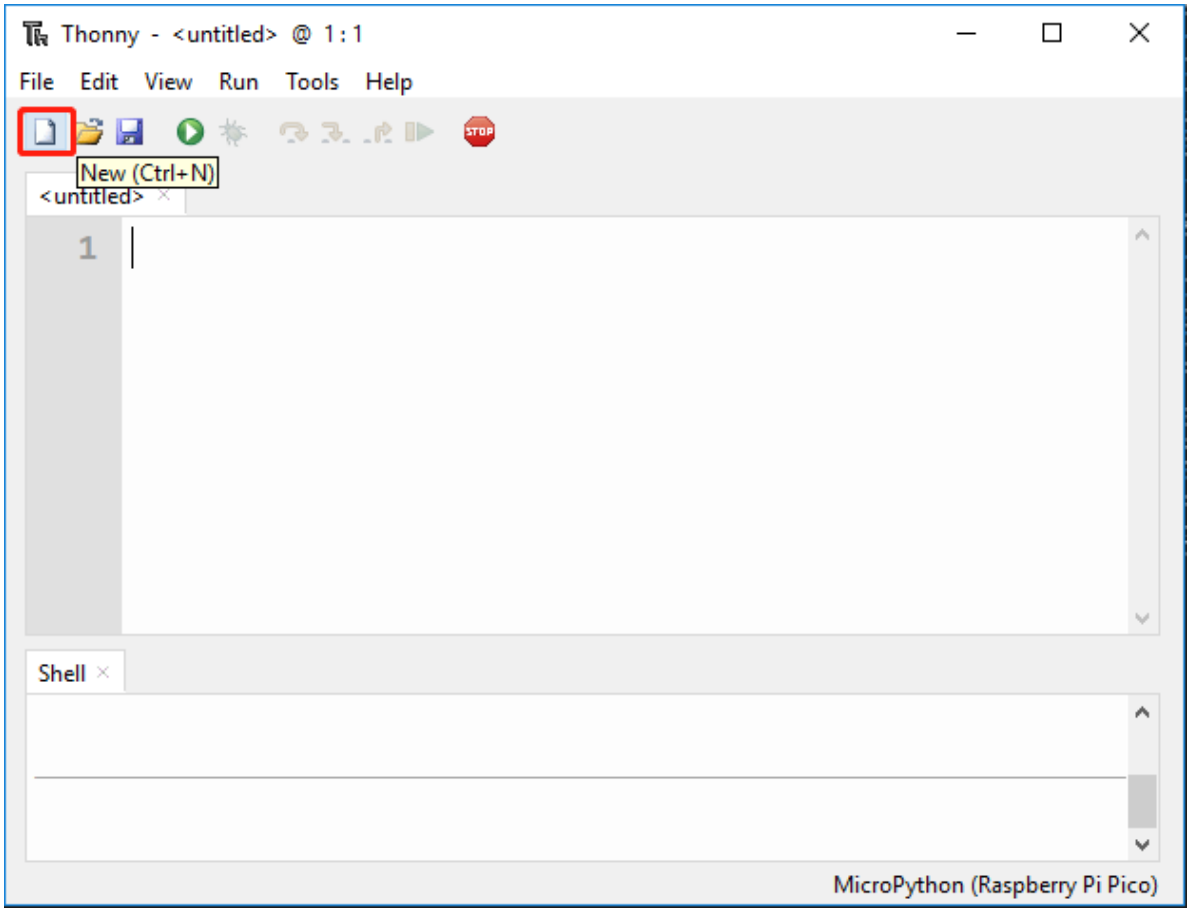
Note: Regardless of what name you give your code, it's best to describe what type of code it is, and not give it a meaningless name like `abc.py`. When you save the code as `main.py`, it will run automatically when the power is turned on.

Create File and Run it

The code is shown directly in the code section. You can copy it to Thonny and run it as follows.

1. Create a new file

   Open Thonny IDE, click **New** button to create a new blank file.

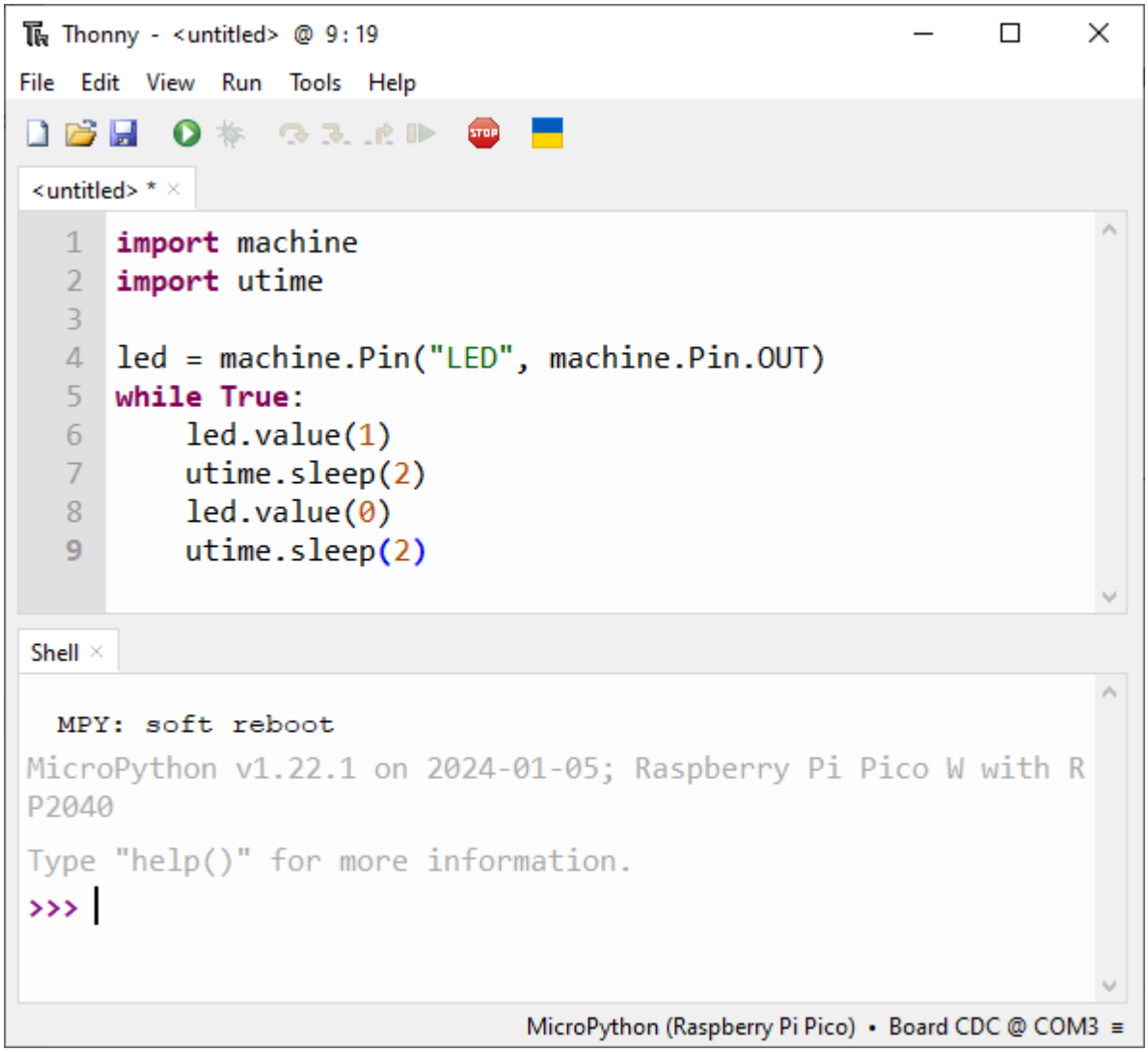


2. Copy code

   Copy the code from the project to the Thonny IDE.

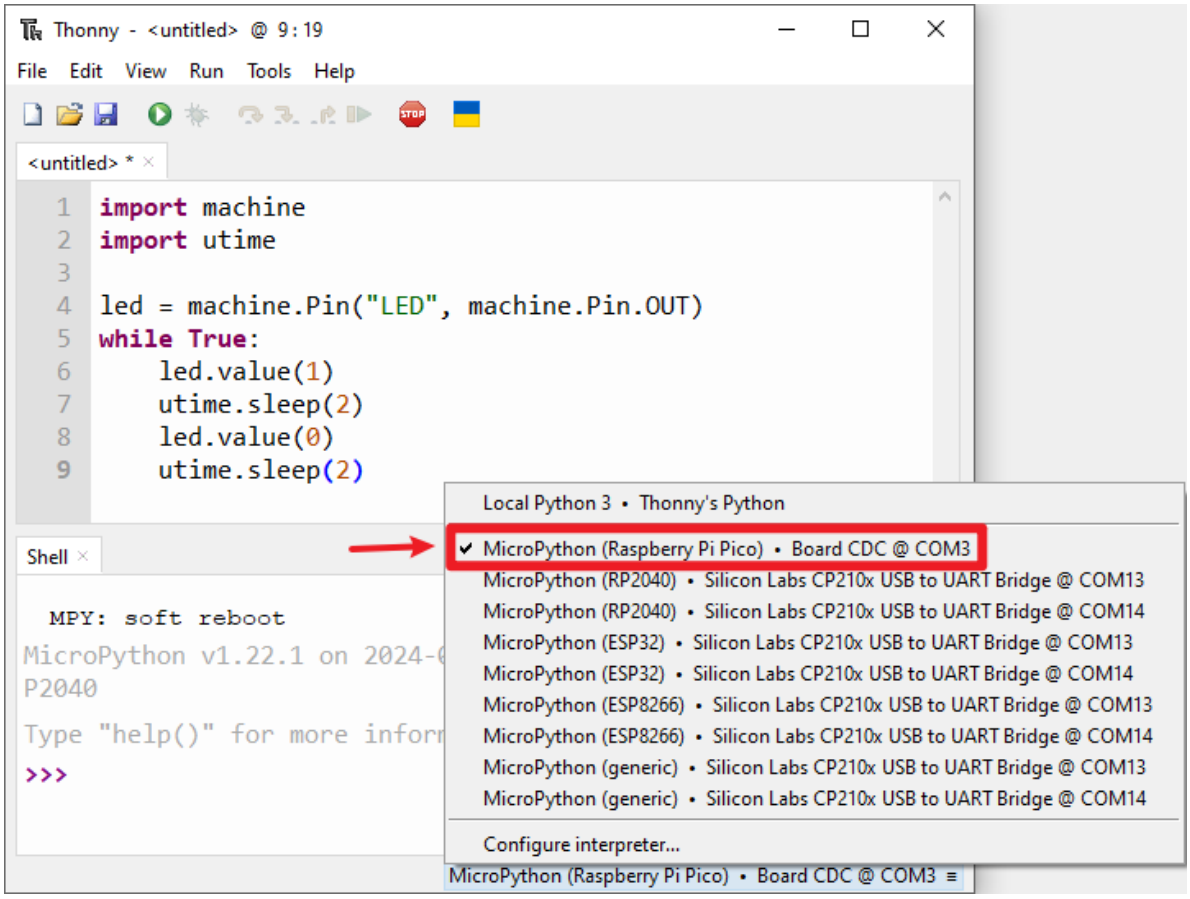
   For example:

```
import machine
import utime

led = machine.Pin("LED", machine.Pin.OUT)
while True:
    led.value(1)
    utime.sleep(2)
    led.value(0)
    utime.sleep(2)
```



3. Select correct interpreter

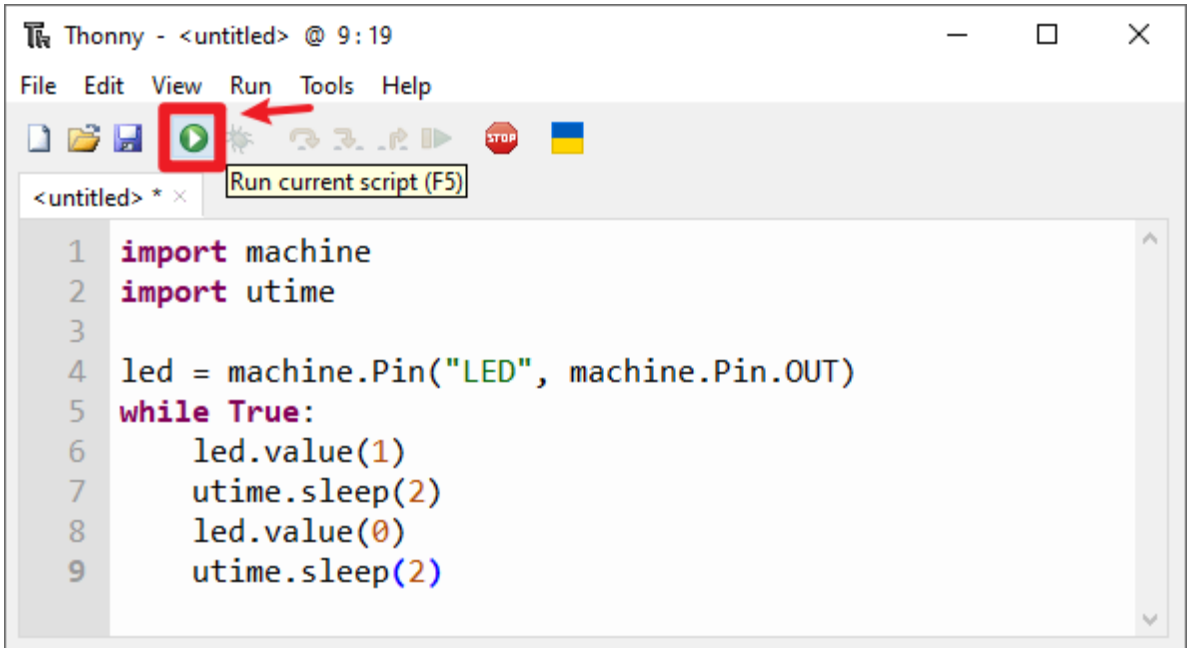
   Plug the Pico W into your computer with a micro USB cable and select the “MicroPython (Raspberry Pi Pico)” interpreter in the bottom right corner.



4. Run the code

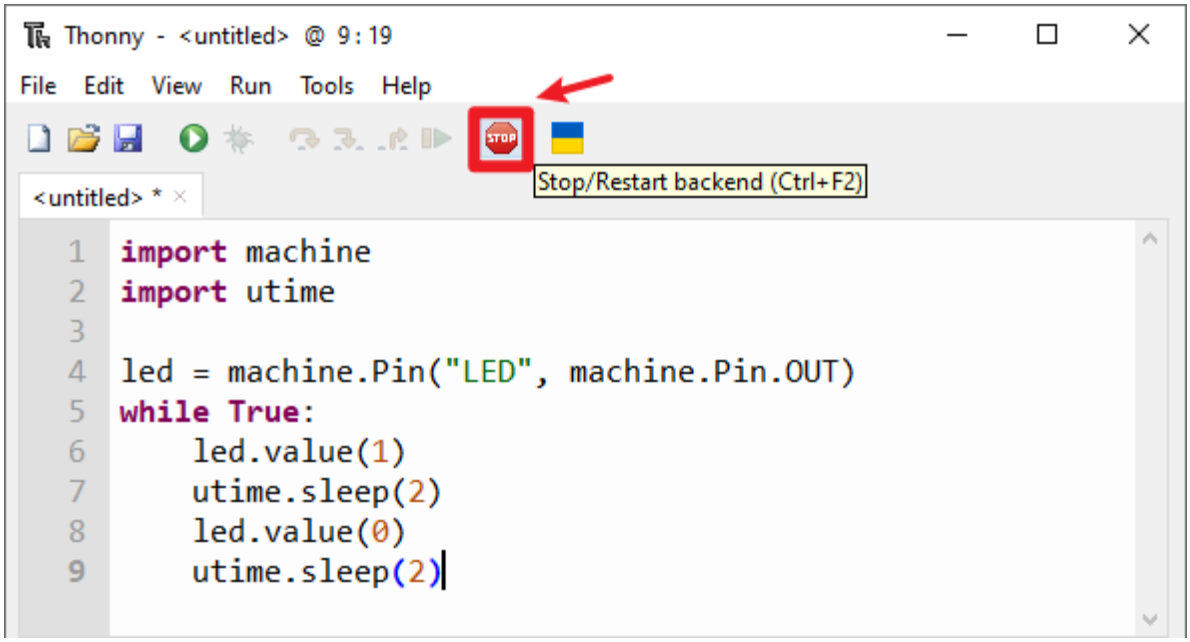
   You can click **Run Current Script** or simply press F5 to run it.

   This code is designed to toggle the onboard LED of the Pico on and off every two seconds, creating a blinking effect. Once the code is executed, you will observe the corresponding blinking phenomenon.



5. Stop running

   To stop the code, click the **Stop/Restart backend** button.

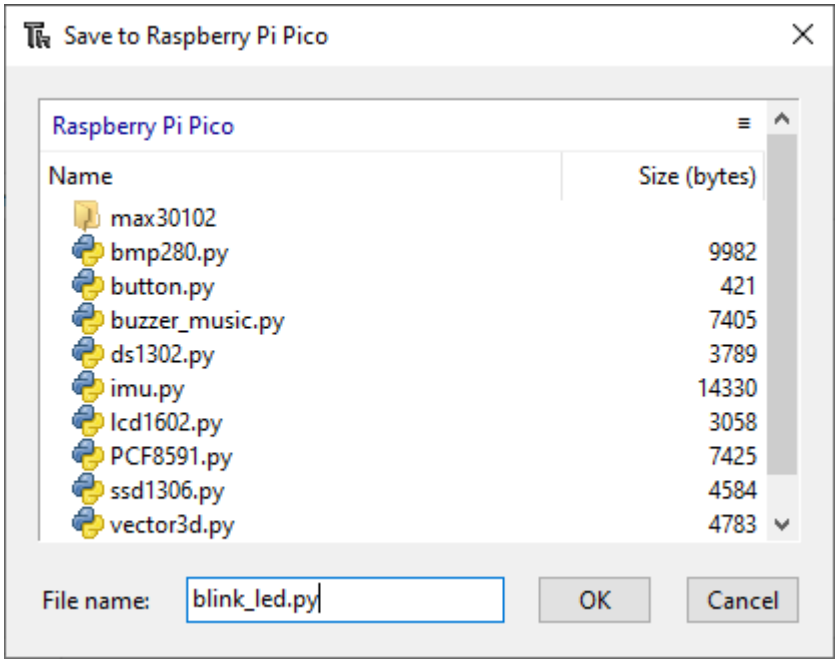


6. Save the code

   You can click the **Save** button to save the code.

Next, Thonny will ask you where to save the code. You can choose to save the code directly to Pico.

Then click OK after entering the file name and extension .py.



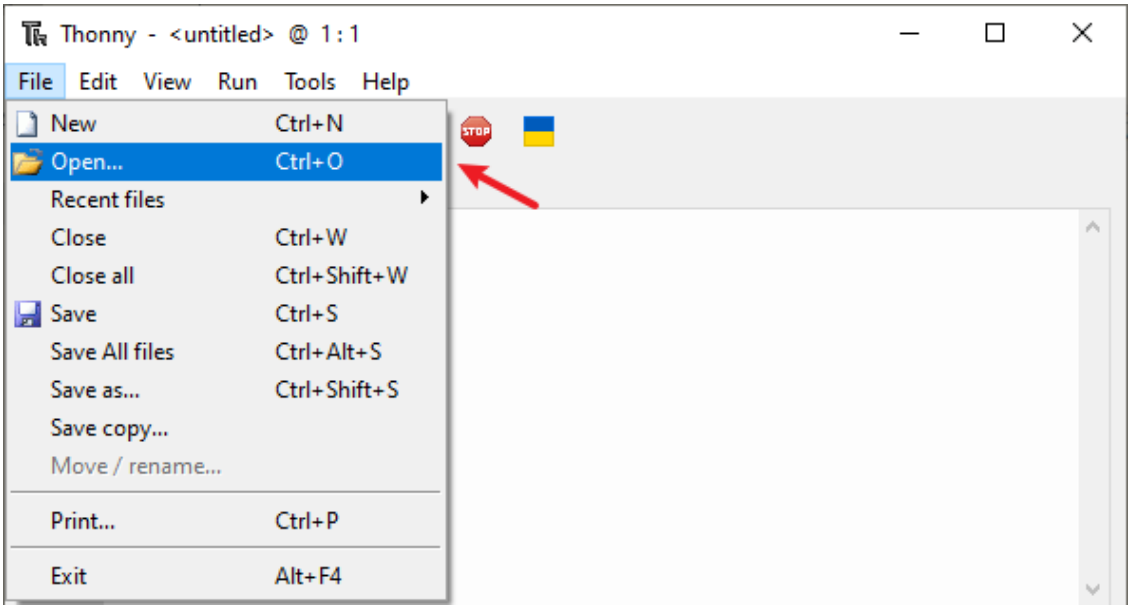
Note: Regardless of what name you give your code, it's best to describe what type of code it is, and not give it a meaningless name like `abc.py`. When you save the code as `main.py`, it will run automatically when the power is turned on.

---

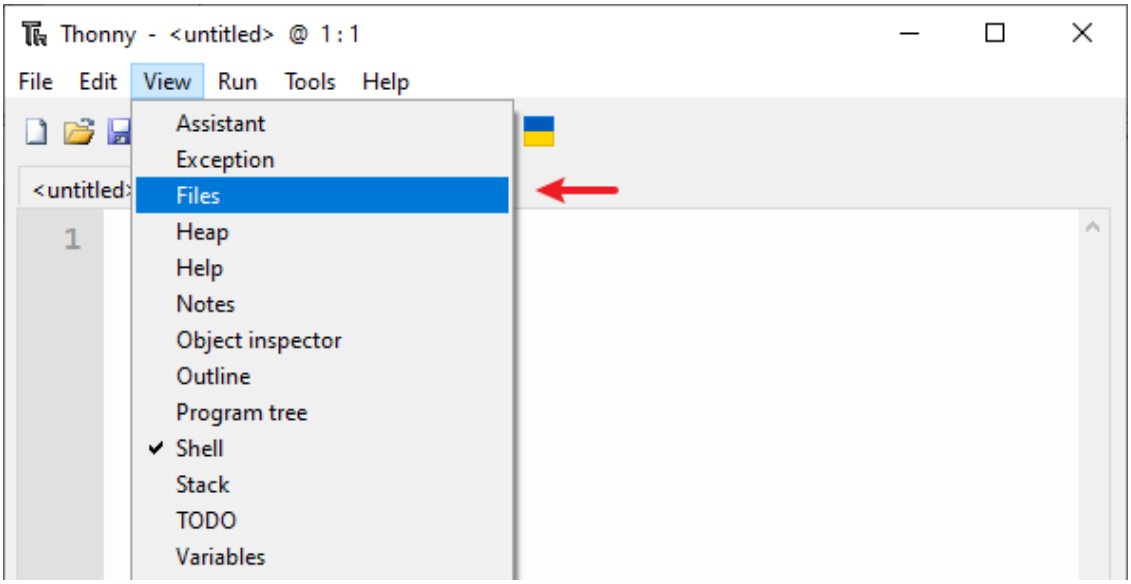
7. Open file

   Here are two ways to open a saved code file.

   - The first method is to click the open icon on the Thonny toolbar, just like when you save a program, you will be asked if you want to open it from **this computer** or **Raspberry Pi Pico**, for example, click **Raspberry Pi Pico** and you will see a list of all the programs you have saved on the Pico W.



- The second is to open the file preview directly by clicking **View**-> **File**-> and then double-clicking on the corresponding `.py` file to open it.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

MicroPython Basic Syntax(Optional)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Indentation

Indentation refers to the spaces at the beginning of a code line. Like standard Python programs, MicroPython programs usually run from top to bottom: It traverses each line in turn, runs it in the interpreter, and then continues to the next line, Just like you type them line by line in the Shell. A program that just browses the instruction list line by line is not very smart, though – so MicroPython, just like Python, has its own method to control the sequence of its program execution: indentation.

You must put at least one space before print(), otherwise an error message “Invalid syntax” will appear. It is usually recommended to standardise spaces by pressing the Tab key uniformly.

```
if 8 > 5:
print("Eight is greater than Five!")
```

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 2
SyntaxError: invalid syntax
```

You must use the same number of spaces in the same block of code, or Python will give you an error.

```
if 8 > 5:
print("Eight is greater than Five!")
        print("Eight is greater than Five")
```

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 2
SyntaxError: invalid syntax
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Comments

The comments in the code help us understand the code, make the entire code more readable and comment out part of the code during testing, so that this part of the code does not run.

Single-line Comment

Single-line comments in MicroPython begin with #, and the following text is considered a comment until the end of the line. Comments can be placed before or after the code.

```
print("hello world") #This is a annotationhello world
```

```
>>> %Run -c $EDITOR_CONTENT
hello world
```

Comments are not necessarily text used to explain the code. You can also comment out part of the code to prevent micropython from running the code.

```
#print("Can't run it")
print("hello world") #This is a annotationhello world
```

```
>>> %Run -c $EDITOR_CONTENT
hello world
```

Multi-line comment

If you want to comment on multiple lines, you can use multiple # signs.

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

```
>>> %Run -c $EDITOR_CONTENT
Hello, World!
```

Or, you can use multi-line strings instead of expected.

Since MicroPython ignores string literals that are not assigned to variables, you can add multiple lines of strings (triple quotes) to the code and put comments in them:

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

```
>>> %Run -c $EDITOR_CONTENT
Hello, World!
```

As long as the string is not assigned to a variable, MicroPython will ignore it after reading the code and treat it as if you made a multi-line comment.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Print()

The `print()` function prints the specified message to the screen, or other standard output device. The message can be a string, or any other object, the object will be converted into a string before written to the screen.

Print multiple objects:

```
print("Welcome!", "Enjoy yourself!")
```

```
>>> %Run -c $EDITOR_CONTENT
Welcome! Enjoy yourself!
```

Print tuples:

```
x = ("pear", "apple", "grape")
print(x)
```

```
>>> %Run -c $EDITOR_CONTENT
('pear', 'apple', 'grape')
```

Print two messages and specify the separator:

```
print("Hello", "how are you?", sep="---")
```

```
>>> %Run -c $EDITOR_CONTENT
Hello---how are you?
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Variables

Variables are containers used to store data values.

Creating a variable is very simple. You only need to name it and assign it a value. You don't need to specify the data type of the variable when assigning it, because the variable is a reference, and it accesses objects of different data types through assignment.

Naming variables must follow the following rules:

- Variable names can only contain numbers, letters, and underscores
- The first character of the variable name must be a letter or underscore
- Variable names are case sensitive

Create Variable

There is no command for declaring variables in MicroPython. Variables are created when you assign a value to it for the first time. It does not need to use any specific type declaration, and you can even change the type after setting the variable.

```
x = 8       # x is of type int
x = "lily" # x is now of type str
print(x)
```

```
>>> %Run -c $EDITOR_CONTENT
lily
```

Casting

If you want to specify the data type for the variable, you can do it by casting.

```
x = int(5)    # y will be 5
y = str(5)    # x will be '5'
z = float(5)  # z will be 5.0
print(x,y,z)
```

```
>>> %Run -c $EDITOR_CONTENT
5 5 5.0
```

Get the Type

You can get the data type of a variable with the *type()* function.

```
x = 5
y = "hello"
z = 5.0
print(type(x),type(y),type(z))
```

```
>>> %Run -c $EDITOR_CONTENT
<class 'int'> <class 'str'> <class 'float'>
```

Single or Double Quotes?

In MicroPython, single quotes or double quotes can be used to define string variables.

```
x = "hello"
# is the same as
x = 'hello'
```

Case-Sensitive

Variable names are case-sensitive.

```
a = 5
A = "lily"
#A will not overwrite a
print(a, A)
```

```
>>> %Run -c $EDITOR_CONTENT
5 lily
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

If Else

Decision making is required when we want to execute a code only if a certain condition is satisfied.

if

```
if test expression:
    statement(s)
```

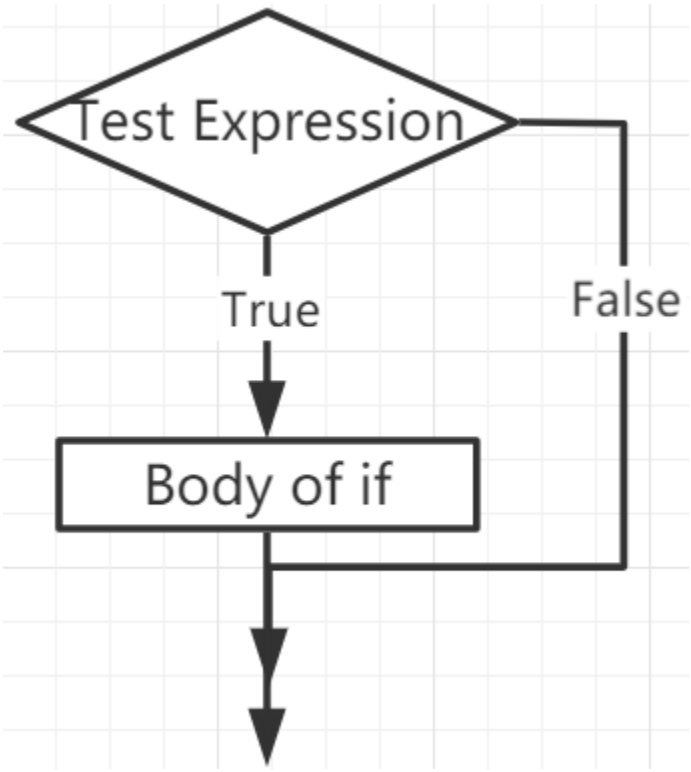
Here, the program evaluates the `test expression` and executes the `statement` only when the `test expression` is True.

If `test expression` is False, then `statement(s)` will not be executed.

In MicroPython, indentation means the body of the `if` statement. The body starts with an indentation and ends with the first unindented line.

Python interprets non-zero values as “True”. None and 0 are interpreted as “False”.

if Statement Flowchart



Example

```
num = 8
if num > 0:
    print(num, "is a positive number.")
print("End with this line")
```

```
>>> %Run -c $EDITOR_CONTENT
8 is a positive number.
End with this line
```

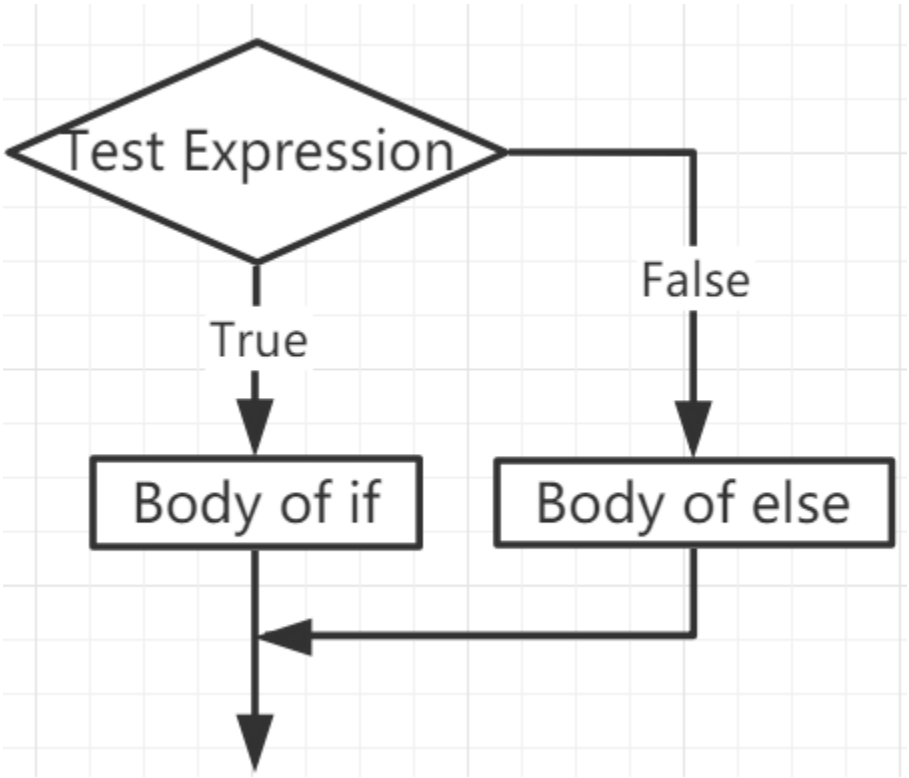
if...else

```
if test expression:
    Body of if
else:
    Body of else
```

The `if..else` statement evaluates `test expression` and will execute the body of `if` only when the test condition is `True`.

If the condition is `False`, the body of `else` is executed. Indentation is used to separate the blocks.

if...else Statement Flowchart



Example

```
num = -8
if num > 0:
    print(num, "is a positive number.")
else:
    print(num, "is a negative number.")
```

```
>>> %Run -c $EDITOR_CONTENT
-8 is a negative number.
```

if...elif...else

```
if test expression:
    Body of if
elif test expression:
    Body of elif
else:
    Body of else
```

`Elif` is short for `else if`. It allows us to check multiple expressions.

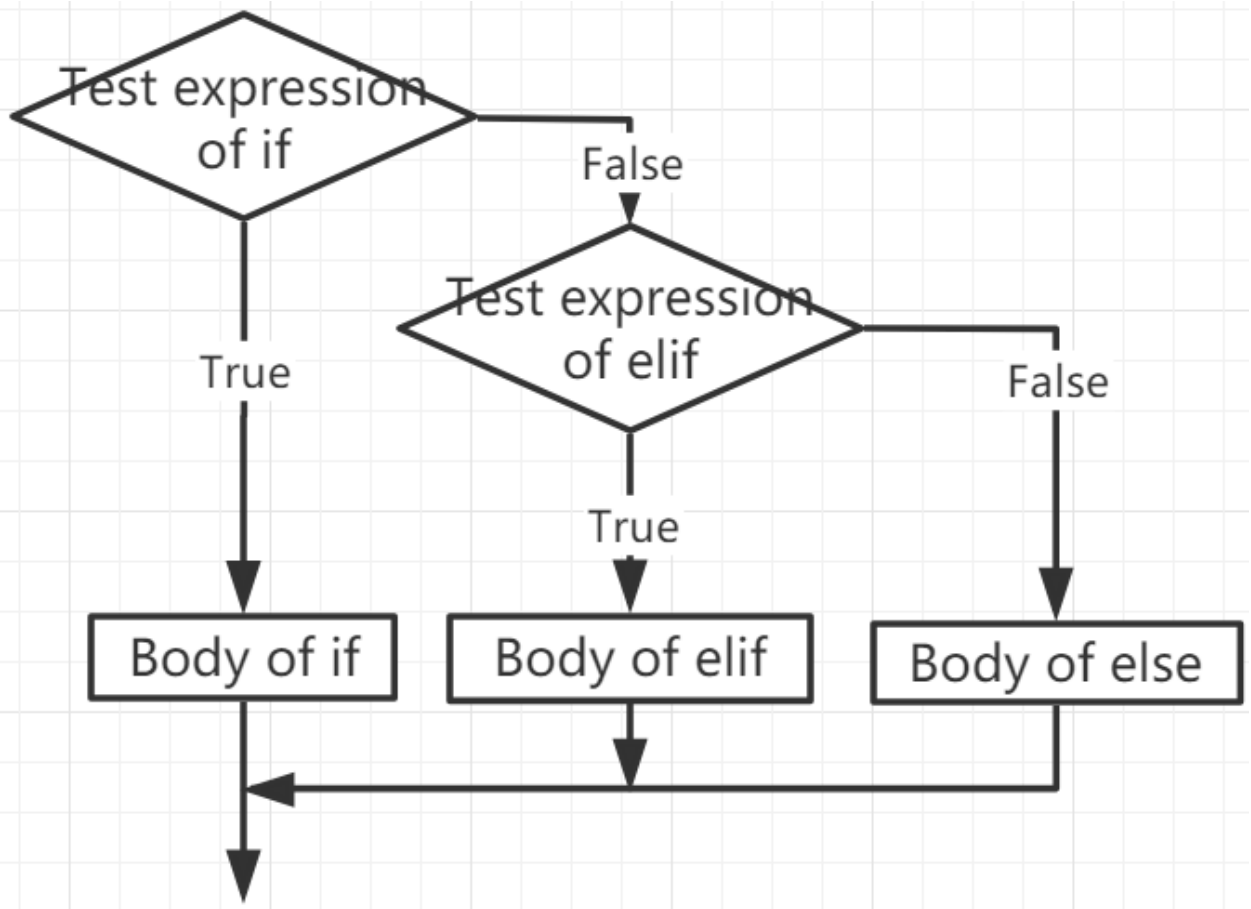
If the condition of the `if` is False, the condition of the next elif block is checked, and so on.

If all conditions are `False`, the body of `else` is executed.

Only one of several `if...elif...else` blocks is executed according to the conditions.

The `if` block can only have one `else` block. But it can have multiple `elif` blocks.

if…elif…else Statement Flowchart



Example

```
x = 10
y = 9

if x > y:
    print("x is greater than y")
elif x == y:
    print("x and y are equal")
else:
    print("x is greater than y")
```

```
>>> %Run -c $EDITOR_CONTENT
x is greater than y
```

Nested if

We can embed an if statement into another if statement, and then call it a nested if statement.

Example

```
x = 67

if x > 10:
    print("Above ten,")
    if x > 20:
        print("and also above 20!")
    else:
        print("but not above 20.")
```

```
>>> %Run -c $EDITOR_CONTENT
Above ten,
and also above 20!
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

While Loops

The `while` statement is used to execute a program in a loop, that is, to execute a program in a loop under certain conditions to handle the same task that needs to be processed repeatedly.

Its basic form is:

```
while test expression:
    Body of while
```

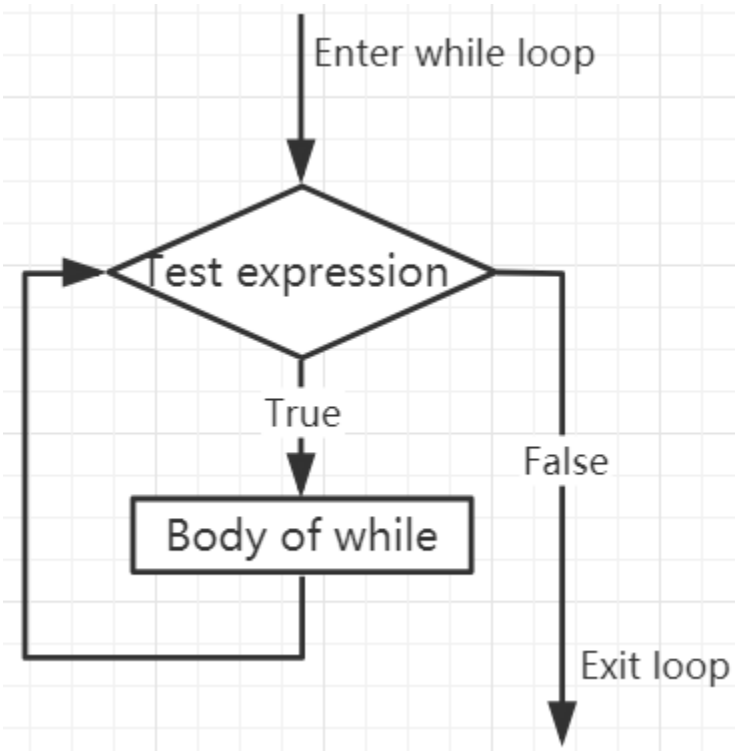
In the `while` loop, first check the `test expression`. Only when `test expression` evaluates to `True`, enter the body of the while. After one iteration, check the `test expression` again. This process continues until `test expression` evaluates to `False`.

In MicroPython, the body of the `while` loop is determined by indentation.

The body starts with an indentation and ends with the first unindented line.

Python interprets any non-zero value as `True`. None and 0 are interpreted as `False`.

while Loop Flowchart




```
x = 10

while x > 0:
    print(x)
    x -= 1
```

```
>>> %Run -c $EDITOR_CONTENT
10
9
8
7
6
5
4
3
2
1
```

Break Statement

With the break statement we can stop the loop even if the while condition is true:

```
x = 10

while x > 0:
    print(x)
    if x == 6:
```

(continues on next page)

(continued from previous page)

```
        break
    x -= 1
```

```
>>> %Run -c $EDITOR_CONTENT
10
9
8
7
6
```

While Loop with Else

Like the `if` loop, the `while` loop can also have an optional `else` block.

If the condition in the `while` loop is evaluated as `False`, the `else` part is executed.

```
x = 10

while x > 0:
    print(x)
    x -= 1
else:
    print("Game Over")
```

```
>>> %Run -c $EDITOR_CONTENT
10
9
8
7
6
5
4
3
2
1
Game Over
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

For Loops

The `for` loop can traverse any sequence of items, such as a list or a string.

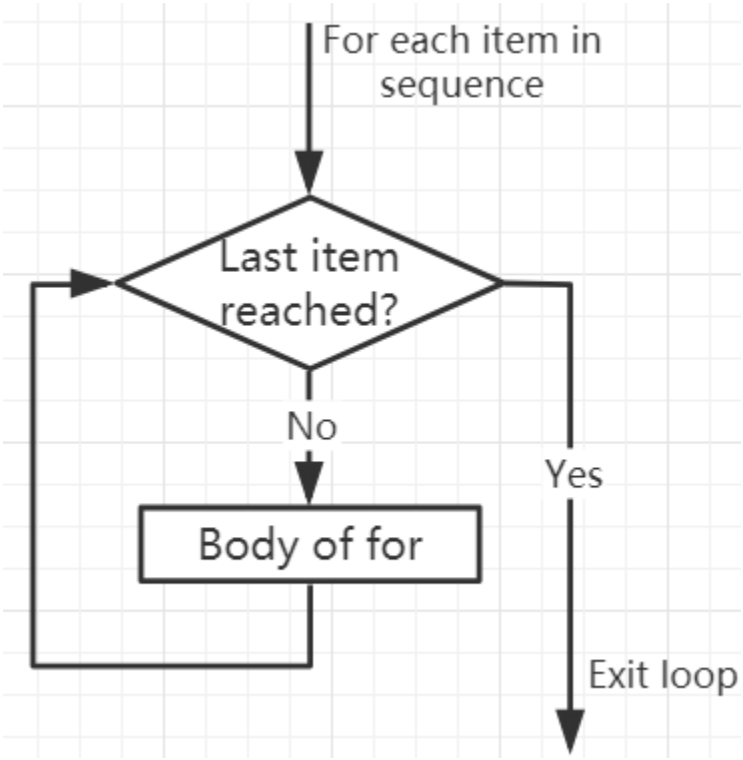
The syntax format of for loop is as follows:

```
for val in sequence:
    Body of for
```

Here, `val` is a variable that gets the value of the item in the sequence in each iteration.

The loop continues until we reach the last item in the sequence. Use indentation to separate the body of the `for` loop from the rest of the code.

Flowchart of for Loop




```
numbers = [1, 2, 3, 4]
sum = 0

for val in numbers:
    sum = sum+val

print("The sum is", sum)
```

```
>>> %Run -c $EDITOR_CONTENT
The sum is 10
```

The break Statement

With the break statement we can stop the loop before it has looped through all the items:

```
numbers = [1, 2, 3, 4]
sum = 0

for val in numbers:
    sum = sum+val
    if sum == 6:
        break
print("The sum is", sum)
```

```
>>> %Run -c $EDITOR_CONTENT
The sum is 6
```

The continue Statement

With the `continue` statement we can stop the current iteration of the loop, and continue with the next:

```
numbers = [1, 2, 3, 4]

for val in numbers:
    if val == 3:
        continue
    print(val)
```

```
>>> %Run -c $EDITOR_CONTENT
1
2
4
```

The range() function

We can use the range() function to generate a sequence of numbers. range(6) will produce numbers between 0 and 5 (6 numbers).

We can also define start, stop and step size as range(start, stop, step_size). If not provided, step_size defaults to 1.

In a sense of range, the object is "lazy" because when we create the object, it does not generate every number it "contains". However, this is not an iterator because it supports in, len and `__getitem__` operations.

This function will not store all values in memory; it will be inefficient. So it will remember the start, stop, step size and generate the next number during the journey.

To force this function to output all items, we can use the function list().

```
print(range(6))

print(list(range(6)))

print(list(range(2, 6)))
```

(continues on next page)

(continued from previous page)

```
print(list(range(2, 10, 2)))
```

```
>>> %Run -c $EDITOR_CONTENT
range(0, 6)
[0, 1, 2, 3, 4, 5]
[2, 3, 4, 5]
[2, 4, 6, 8]
```

We can use `range()` in a `for` loop to iterate over a sequence of numbers. It can be combined with the len() function to use the index to traverse the sequence.

```
fruits = ['pear', 'apple', 'grape']

for i in range(len(fruits)):
    print("I like", fruits[i])
```

```
>>> %Run -c $EDITOR_CONTENT
I like pear
I like apple
I like grape
```

Else in For Loop

The `for` loop can also have an optional `else` block. If the items in the sequence used for the loop are exhausted, the `else` part is executed.

The `break` keyword can be used to stop the `for` loop. In this case, the `else` part will be ignored.

Therefore, if no interruption occurs, the `else` part of the `for` loop will run.

```
for val in range(5):
    print(val)
else:
    print("Finished")
```

```
>>> %Run -c $EDITOR_CONTENT
0
1
2
3
4
Finished
```

The else block will NOT be executed if the loop is stopped by a break statement.

```
for val in range(5):
    if val == 2: break
    print(val)
else:
    print("Finished")
```

```
>>> %Run -c $EDITOR_CONTENT
0
1
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Functions

In MicroPython, a function is a group of related statements that perform a specific task.

Functions help break our program into smaller modular blocks. As our plan becomes larger and larger, functions make it more organized and manageable.

In addition, it avoids duplication and makes the code reusable.

Create a Function

```
def function_name(parameters):
    """docstring"""
    statement(s)
```

- A function is defined using the `def` keyword
- A function name to uniquely identify the function. Function naming is the same as variable naming, and both follow the following rules.
  - Can only contain numbers, letters, and underscores.
  - The first character must be a letter or underscore.
  - Case sensitive.
- Parameters (arguments) through which we pass values to a function. They are optional.
- The colon (:) marks the end of the function header.
- Optional docstring, used to describe the function of the function, we usually use triple quotes so that the docstring can be expanded to multiple lines.
- One or more valid Micropython statements that make up the function body. Statements must have the same indentation level (usually 4 spaces).

- Each function needs at least one statement, but if for some reason there is a function that does not contain any statement, please put in the pass statement to avoid errors.
- An optional `return` statement to return a value from the function.

Calling a Function

To call a function, add parentheses after the function name.

```
def my_function():
    print("Your first function")

my_function()
```

```
>>> %Run -c $EDITOR_CONTENT
Your first function
```

The return Statement

The return statement is used to exit a function and return to the place where it was called.

Syntax of return

```
return [expression_list]
```

The statement can contain an expression that is evaluated and returns a value. If there is no expression in the statement, or the `return` statement itself does not exist in the function, the function will return a `None` object.

```
def my_function():
        print("Your first function")

print(my_function())
```

```
>>> %Run -c $EDITOR_CONTENT
Your first function
None
```

Here, `None` is the return value, because the `return` statement is not used.

Arguments

Information can be passed to the function as arguments.

Specify arguments in parentheses after the function name. You can add as many arguments as you need, just separate them with commas.

```
def welcome(name, msg):
    """This is a welcome function for
    the person with the provided message"""
    print("Hello", name + ', ' + msg)

welcome("Lily", "Welcome to China!")
```

```
>>> %Run -c $EDITOR_CONTENT
Hello Lily, Welcome to China!
```

Number of Arguments

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 parameters, you have to call the function with 2 arguments, not more, and not less.

```
def welcome(name, msg):
    """This is a welcome function for
    the person with the provided message"""
    print("Hello", name + ', ' + msg)

welcome("Lily", "Welcome to China!")
```

Here, the function welcome() has 2 parameters.

Since we called this function with two arguments, the function runs smoothly without any errors.

If it is called with a different number of arguments, the interpreter will display an error message.

The following is the call to this function, which contains one and one no arguments and their respective error messages.

```
welcome("Lily")Only one argument
```

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 6, in <module>
TypeError: function takes 2 positional arguments but 1 were given
```

```
welcome()No arguments
```

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 6, in <module>
TypeError: function takes 2 positional arguments but 0 were given
```

Default Arguments

In MicroPython, we can use the assignment operator (=) to provide a default value for the parameter.

If we call the function without argument, it uses the default value.

```
def welcome(name, msg = "Welcome to China!"):
    """This is a welcome function for
    the person with the provided message"""
    print("Hello", name + ', ' + msg)
welcome("Lily")
```

```
>>> %Run -c $EDITOR_CONTENT
Hello Lily, Welcome to China!
```

In this function, the parameter `name` has no default value and is required (mandatory) during the call.

On the other hand, the default value of the parameter `msg` is "Welcome to China!". Therefore, it is optional during the call. If a value is provided, it will overwrite the default value.

Any number of arguments in the function can have a default value. However, once there is a default argument, all arguments on its right must also have default values.

This means that non-default arguments cannot follow default arguments.

For example, if we define the above function header as:

```
def welcome(name = "Lily", msg):
```

We will receive the following error message:

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
SyntaxError: non-default argument follows default argument
```

Keyword Arguments

When we call a function with certain values, these values will be assigned to arguments based on their position.

For example, in the above function welcome(), when we called it as welcome("Lily", "Welcome to China"), the value "Lily" gets assigned to the `name` and similarly "Welcome to China" to parameter `msg`.

MicroPython allows calling functions with keyword arguments. When we call the function in this way, the order (position) of the arguments can be changed.

```
# keyword arguments
welcome(name = "Lily",msg = "Welcome to China!")

# keyword arguments (out of order)
welcome(msg = "Welcome to China",name = "Lily")

#1 positional, 1 keyword argument
welcome("Lily", msg = "Welcome to China!")
```

As we can see, we can mix positional arguments and keyword arguments during function calls. But we must remember that the keyword arguments must come after the positional arguments.

Having a positional argument after a keyword argument will result in an error.

For example, if the function call as follows:

```
welcome(name="Lily","Welcome to China!")
```

Will result in an error:

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 5, in <module>
SyntaxError: non-keyword arg after keyword arg
```

Arbitrary Arguments

Sometimes, if you do not know the number of arguments that will be passed to the function in advance.

In the function definition, we can add an asterisk (*) before the parameter name.

```
def welcome(*names):
    """This function welcomes all the person
    in the name tuple"""
    #names is a tuple with arguments
    for name in names:
        print("Welcome to China!", name)

welcome("Lily","John","Wendy")
```

```
>>> %Run -c $EDITOR_CONTENT
Welcome to China! Lily
Welcome to China! John
Welcome to China! Wendy
```

Here, we have called the function with multiple arguments. These arguments are packed into a tuple before being passed into the function.

Inside the function, we use a for loop to retrieve all the arguments.

Recursion

In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.

This has the benefit of meaning that you can loop through data to reach a result.

The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

```
def rec_func(i):
    if(i > 0):
        result = i + rec_func(i - 1)
        print(result)
    else:
        result = 0
    return result

rec_func(6)
```

```
>>> %Run -c $EDITOR_CONTENT
1
3
6
10
15
21
```

In this example, rec_func() is a function that we have defined to call itself ("recursion"). We use the `i` variable as the data, and it will decrement (-1) every time we recurse. When the condition is not greater than 0 (that is, 0), the recursion ends.

For new developers, it may take some time to determine how it works, and the best way to test it is to test and modify it.

Advantages of Recursion

- Recursive functions make the code look clean and elegant.
- A complex task can be broken down into simpler sub-problems using recursion.
- Sequence generation is easier with recursion than using some nested iteration.

Disadvantages of Recursion

- Sometimes the logic behind recursion is hard to follow through.
- Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
- Recursive functions are hard to debug.

Data Types

Built-in Data Types

MicroPython has the following data types:

- Text Type: str
- Numeric Types: int, float, complex
- Sequence Types: list, tuple, range
- Mapping Type: dict
- Set Types: set, frozenset
- Boolean Type: bool
- Binary Types: bytes, bytearray, memoryview

Getting the Data Type

You can get the data type of any object by using the `type()` function:

```
a = 6.8
print(type(a))
```

```
>>> %Run -c $EDITOR_CONTENT
<class 'float'>
```

Setting the Data Type

MicroPython does not need to set the data type specifically, it has been determined when you assign a value to the variable.

```
x = "welcome"
y = 45
z = ["apple", "banana", "cherry"]

print(type(x))
print(type(y))
print(type(z))
```

```
>>> %Run -c $EDITOR_CONTENT
<class 'str'>
<class 'int'>
<class 'list'>
>>>
```

Setting the Specific Data Type

If you want to specify the data type, you can use the following constructor functions:

Example	Date Type
x = int(20)	int
x = float(20.5)	float
x = complex(1j)	complex
x = str(“Hello World”)	str
x = list((“apple”, “banana”, “cherry”))	list
x = tuple((“apple”, “banana”, “cherry”))	tuple
x = range(6)	range
x = dict(name=”John”, age=36)	dict
x = set((“apple”, “banana”, “cherry”))	set
x = frozenset((“apple”, “banana”, “cherry”))	frozenset
x = bool(5)	bool
x = bytes(5)	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

You can print some of them to see the result.

```
a = float(20.5)
b = list(("apple", "banana", "cherry"))
c = bool(5)

print(a)
print(b)
print(c)
```

```
>>> %Run -c $EDITOR_CONTENT
20.5
['apple', 'banana', 'cherry']
True
>>>
```

Type Conversion

You can convert from one type to another with the int(), float(), and complex() methods: Casting in python is therefore done using constructor functions:

- int() - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
- float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

```
a = float("5")
b = int(3.7)
c = str(6.0)

print(a)
print(b)
print(c)
```

Note: You cannot convert complex numbers into another number type.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Operators

Operators are used to perform operations on variables and values.

- *Arithmetic Operators*
- *Assignment operators*
- *Comparison Operators*
- *Logical Operators*
- *Identity Operators*
- *Membership Operators*
- *Bitwise Operators*

Arithmetic Operators

You can use arithmetic operators to do some common mathematical operations.

Operator	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
**	Exponentiation
//	Floor division

```
x = 5
y = 3

a = x + y
b = x - y
c = x * y
d = x / y
e = x % y
f = x ** y
g = x // y

print(a)
print(b)
print(c)
print(d)
print(e)
print(f)
print(g)
```

```
>>> %Run -c $EDITOR_CONTENT
8
2
```

(continues on next page)

(continued from previous page)

```
15
1.666667
2
125
1
8
2
15
>>>
```

Assignment operators

Assignment operators can used to assign values to variables.

Operator	Example	Same As
=	a = 6	a =6
+=	a += 6	a = a + 6
-=	a -= 6	a = a - 6
*=	a *= 6	a = a * 6
/=	a /= 6	a = a / 6
%=	a %= 6	a = a % 6
**=	a **= 6	a = a ** 6
//=	a //= 6	a = a // 6
&=	a &= 6	a = a & 6
|=	a |= 6	a = a | 6
^=	a ^= 6	a = a ^ 6
>>=	a >>= 6	a = a >> 6
<<=	a <<= 6	a = a << 6

```
a = 6

a *= 6
print(a)
```

```
>>> %Run test.py
36
>>>
```

Comparison Operators

Comparison operators are used to compare two values.

Operator	Name
==	Equal
!=	Not equal
<	Less than
>	Greater than
>=	Greater than or equal to
<=	Less than or equal to

```
a = 6
b = 8

print(a>b)
```

```
>>> %Run test.py
False
>>>
```

Return **False**, because the **a** is less than the **b**.

Logical Operators

Logical operators are used to combine conditional statements.

Operator	Description
and	Returns True if both statements are true
or	Returns True if one of the statements is true
not	Reverse the result, returns False if the result is true

```
a = 6
print(a > 2 and a < 8)
```

```
>>> %Run -c $EDITOR_CONTENT
True
>>>
```

Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location.

Operator	Description
is	Returns True if both variables are the same object
is not	Returns True if both variables are not the same object

```
a = ["hello", "welcome"]
b = ["hello", "welcome"]
c = a

print(a is c)
# returns True because z is the same object as x

print(a is b)
# returns False because x is not the same object as y, even if they have the same content

print(a == b)
# returns True because x is equal to y
```

```
>>> %Run -c $EDITOR_CONTENT
True
False
True
>>>
```

Membership Operators

Membership operators are used to test if a sequence is presented in an object.

Operator	Description
in	Returns True if a sequence with the specified value is present in the object
not in	Returns True if a sequence with the specified value is not present in the object

```
a = ["hello", "welcome", "Goodmorning"]

print("welcome" in a)
```

```
>>> %Run -c $EDITOR_CONTENT
True
>>>
```

Bitwise Operators

Bitwise operators are used to compare (binary) numbers.

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

```
num = 2

print(num & 1)
print(num | 1)
print(num << 1)
```

```
>>> %Run -c $EDITOR_CONTENT
0
3
4
>>>
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Lists

Lists are used to store multiple items in a single variable, and are created using square brackets:

```
B_list = ["Blossom", "Bubbles","Buttercup"]
print(B_list)
```

List items are changeable, ordered, and allow duplicate values. The list items are indexed, with the first item having index [0], the second item having index [1], and so on.

```
C_list = ["Red", "Blue", "Green", "Blue"]
print(C_list)             # duplicate
print(C_list[0])
print(C_list[1])          # ordered
C_list[2] = "Purple"      # changeable
print(C_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Red', 'Blue', 'Green', 'Blue']
Red
Blue
['Red', 'Blue', 'Purple', 'Blue']
```

A list can contain different data types:

```
A_list = ["Banana", 255, False, 3.14]
print(A_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Banana', 255, False, 3.14]
```

List Length

To determine how many items are in the list, use the len() function.

```
A_list = ["Banana", 255, False, 3.14]
print(len(A_list))
```

```
>>> %Run -c $EDITOR_CONTENT
4
```

Check List items

Print the second item of the list:

```
A_list = ["Banana", 255, False, 3.14]
print(A_list[1])
```

```
>>> %Run -c $EDITOR_CONTENT
[255]
```

Print the last one item of the list:

```
A_list = ["Banana", 255, False, 3.14]
print(A_list[-1])
```

```
>>> %Run -c $EDITOR_CONTENT
[3.14]
```

Print the second, third item:

```
A_list = ["Banana", 255, False, 3.14]
print(A_list[1:3])
```

```
>>> %Run -c $EDITOR_CONTENT
[255, False]
```

Change List Items

Change the second, third item:

```
A_list = ["Banana", 255, False, 3.14]
A_list[1:3] = [True,"Orange"]
print(A_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Banana', True, 'Orange', 3.14]
```

Change the second value by replacing it with two values:

```
A_list = ["Banana", 255, False, 3.14]
A_list[1:2] = [True,"Orange"]
print(A_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Banana', True, 'Orange', False, 3.14]
```

Add List Items

Using the append() method to add an item:

```
C_list = ["Red", "Blue", "Green"]
C_list.append("Orange")
print(C_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Red', 'Blue', 'Green', 'Orange']
```

Insert an item as the second position:

```
C_list = ["Red", "Blue", "Green"]
C_list.insert(1, "Orange")
print(C_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Red', 'Orange', 'Blue', 'Green']
```

Remove List Items

The remove() method removes the specified item.

```
C_list = ["Red", "Blue", "Green"]
C_list.remove("Blue")
print(C_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Red', 'Green']
```

The pop() method removes the specified index. If you do not specify the index, the pop() method removes the last item.

```
A_list = ["Banana", 255, False, 3.14, True,"Orange"]
A_list.pop(1)
print(A_list)
A_list.pop()
print(A_list)
```

```
>>> %Run -c $EDITOR_CONTENT
255
['Banana', False, 3.14, True, 'Orange']
'Orange'
['Banana', False, 3.14, True]
```

The `del` keyword also removes the specified index:

```
C_list = ["Red", "Blue", "Green"]
del C_list[1]
print(C_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Red', 'Green']
```

The clear() method empties the list. The list still remains, but it has no content.

```
C_list = ["Red", "Blue", "Green"]
C_list.clear()
print(C_list)
```

```
>>> %Run -c $EDITOR_CONTENT
[]
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.2 Lesson 01: Button Module

In this lesson, you will learn how to use Raspberry Pi Pico W to interact with the onboard LED using a button. Pressing the button will light up the LED, and releasing the button will turn it off. This project is ideal for beginners as it offers hands-on experience with input and output operations on Raspberry Pi Pico W using MicroPython.

Required Components

In this project, we need the following components.

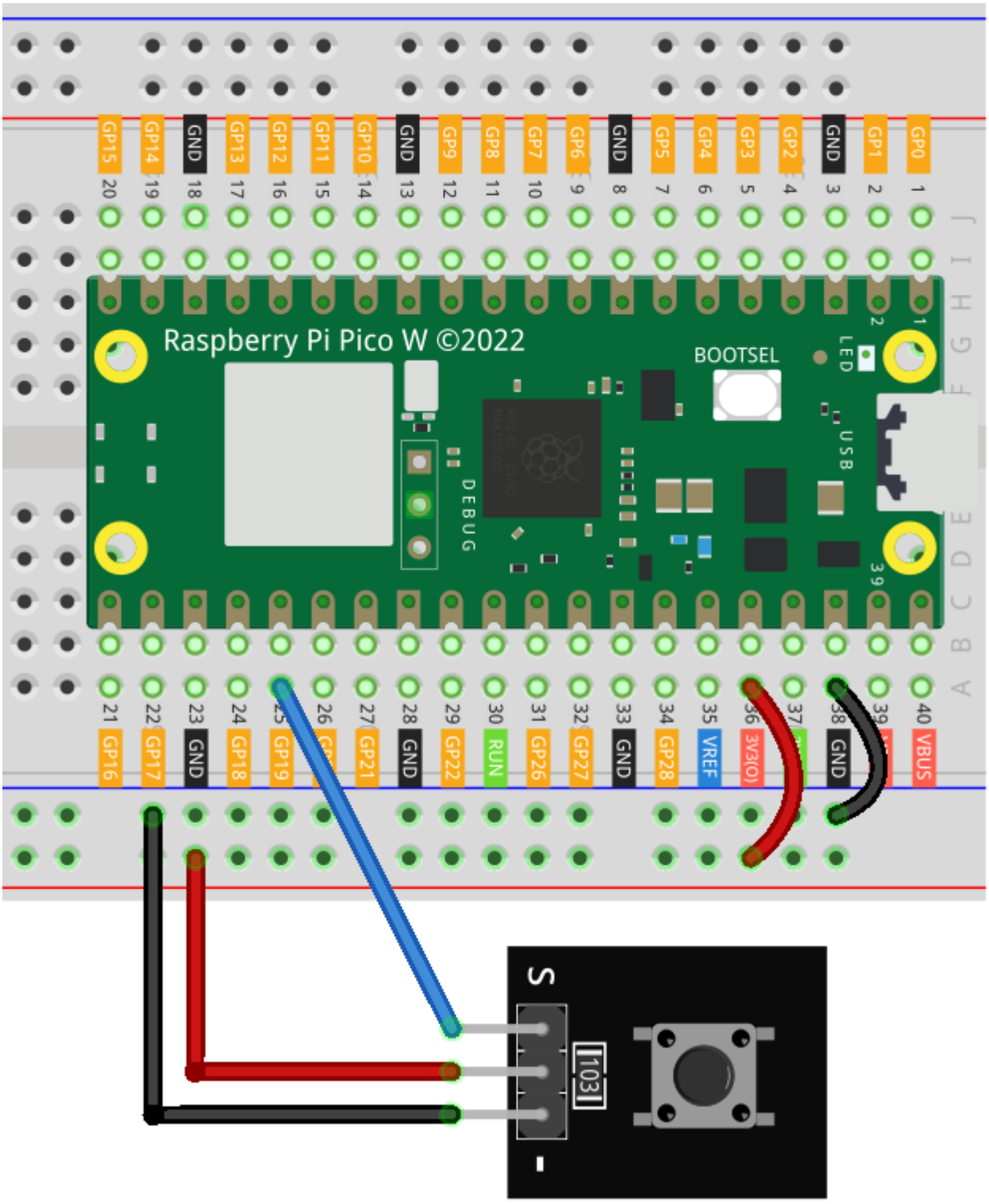
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Button Module	-
Breadboard	

Wiring


Raspberry Pi Pico W ©2022
BOOTSEL
LED
USB
DEBUG
GP15
GP14
GND
GP13
GP12
GP11
GP10
GND
GP9
GP8
GP7
GP6
GND
GP5
GP4
GP3
GP2
GND
GP1
GP0
GP16
GP17
GND
GP18
GP19
GP21
GND
GP22
RUN
GP26
GP27
GND
GP28
VREF
3V3(O)
GND
VBUS
S
103

Code

```
from machine import Pin
import time

# Set GPIO 19 as an input pin to read the button state
button = Pin(19, Pin.IN)

# Initialize the onboard LED of the Raspberry Pi Pico W
led = Pin('LED', Pin.OUT)

while True:
    if button.value() == 0:  # Check if the button is pressed
        led.value(1)  # Turn on the LED
    else:
        led.value(0)  # Turn off the LED

    time.sleep(0.1)  # Short delay to reduce CPU usage
```

Code Analysis

1. Importing Modules

   The `machine` module is imported to interact with the GPIO pins, and the `time` module is for handling timing.

   ```
   from machine import Pin
   import time
   ```

2. Setting up the Button

   GPIO 19 is configured as an input pin. This will read the state of the push button connected to it.

   ```
   button = Pin(19, Pin.IN)
   ```

3. Setting up the LED

   The onboard LED is set up as an output pin, enabling us to turn it on or off programmatically.

   ```
   led = Pin('LED', Pin.OUT)
   ```

4. Main Loop

   - An infinite loop is used to continuously check the state of the button.
   - If the button is pressed (`button.value() == 0`), the LED is turned on. Otherwise, it's turned off.
   - A short delay of 0.1 seconds is added to reduce CPU usage.

   The *button module* used in this project has an internal pull-up resistor (see its *schematic diagram*), causing the button to be at a low level when pressed and remain at a high level when released.

   ```
   while True:
       if button.value() == 0:  # Check if the button is pressed
           led.value(1)  # Turn on the LED
       else:
   ```

(continues on next page)

(continued from previous page)

```
        led.value(0)  # Turn off the LED
    time.sleep(0.1)  # Short delay to reduce CPU usage
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.3 Lesson 02: Capacitive Soil Moisture Module

In this lesson, you'll learn how to use the Raspberry Pi Pico W to measure soil moisture levels using a capacitive sensor and an ADC (Analog to Digital Converter). This beginner-friendly project will introduce you to handling analog signals in MicroPython.

Required Components

In this project, we need the following components.

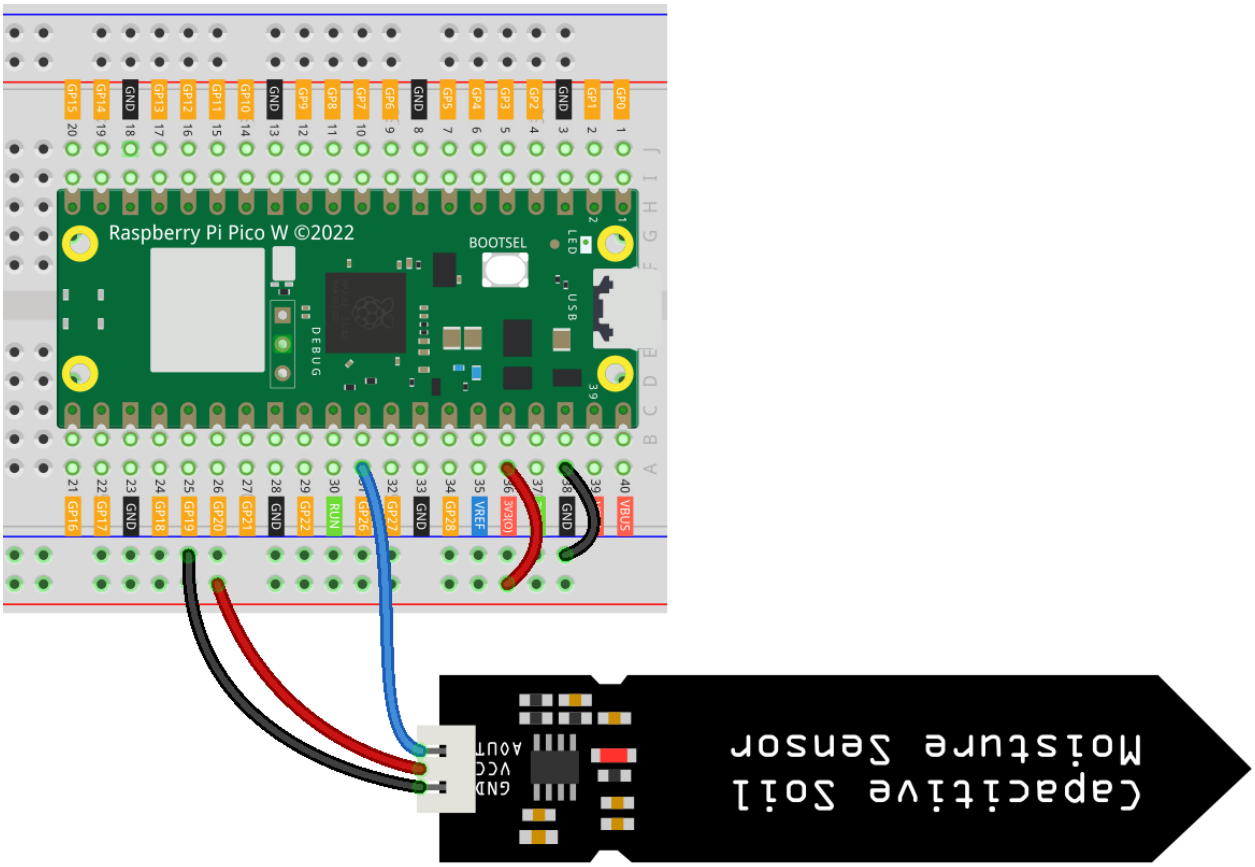
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Capacitive Soil Moisture Module	
Breadboard	

Wiring



Code

```
from machine import ADC
import time

# Initialize an ADC object on GPIO pin 26.
# This is typically used for reading analog signals.
sensor_AO = ADC(26)

# Continuously read and print sensor data.
while True:
    value = sensor_AO.read_u16()  # Read and convert analog value to 16-bit integer
    print("AO:", value)  # Print the analog value

    time.sleep_ms(200)  # Wait for 200 milliseconds before the next read
```

Code Analysis

1. Importing Libraries:

```
from machine import ADC
import time
```

2. ADC Setup:

```
sensor_AO = ADC(26)
```

This code initializes an ADC object on GPIO pin 26. ADC is used to convert analog signals (from analog sensors) to digital data that the microcontroller can process.

3. Reading Sensor Data in a Loop:

```
while True:
    value = sensor_AO.read_u16()
    print("AO:", value)
    time.sleep_ms(200)
```

The `while True` loop runs indefinitely, constantly reading data from the sensor. The `read_u16()` method reads the analog value and converts it to a 16-bit unsigned integer. The `print` statement displays this value. The `time.sleep_ms(200)` causes the loop to wait for 200 milliseconds before reading the sensor value again, preventing excessive data readings and console output.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.4 Lesson 03: Flame Sensor Module

In this lesson, you will learn how to use the Raspberry Pi Pico W to detect fire using a flame sensor. When the sensor detects a flame, the onboard LED of the Raspberry Pi Pico W will turn on and display a message indicating fire detection. If no fire is detected, the LED remains off and shows a different message. This project introduces working with external sensors and provides practical experience in handling digital inputs and outputs on the Raspberry Pi Pico W using MicroPython.

Required Components

In this project, we need the following components.

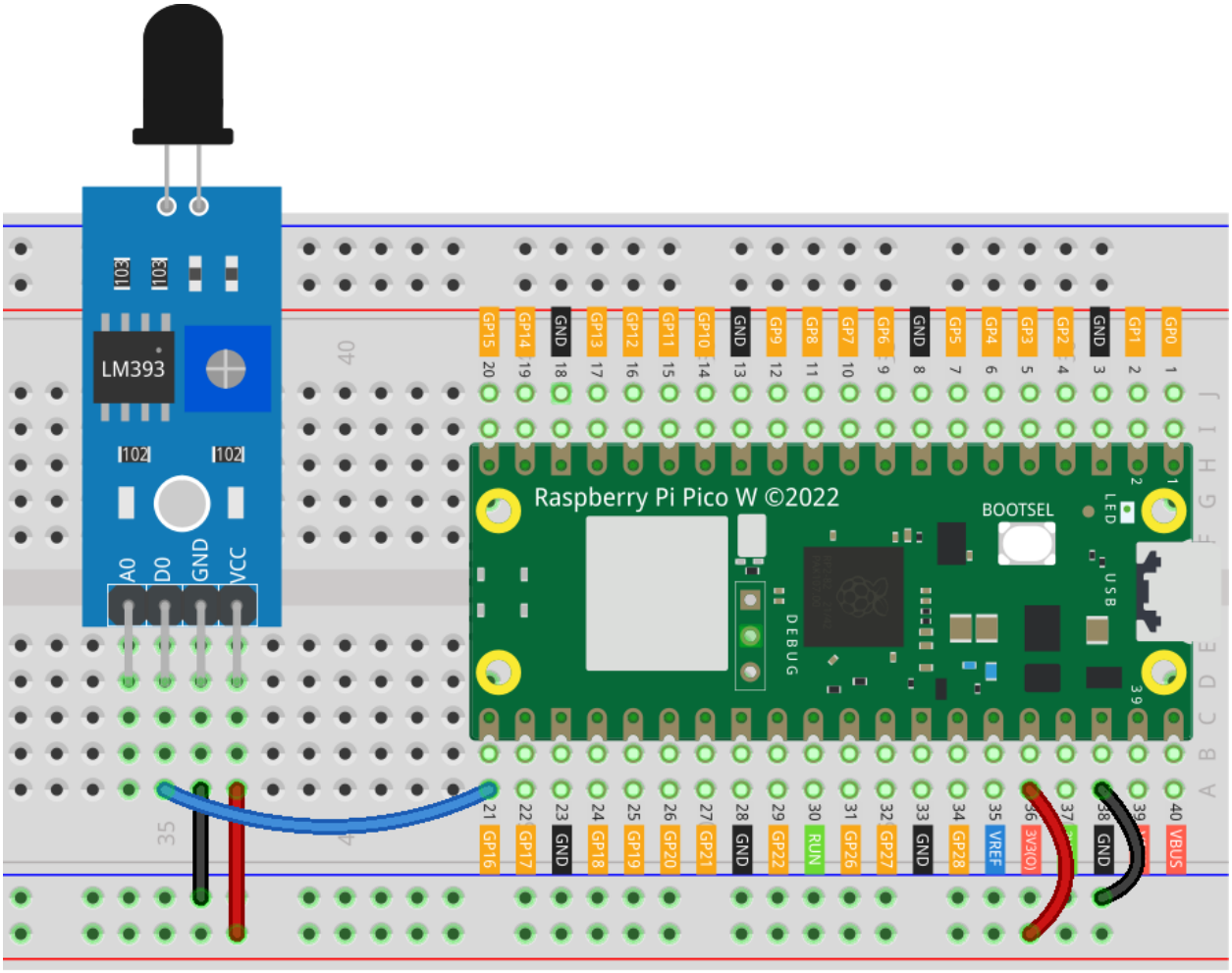
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Flame Sensor Module	
Breadboard	

Wiring



Code

```
from machine import Pin
import time

# Set GPIO 16 as an input pin to read the flame sensor state
flame_sensor = Pin(16, Pin.IN)

# Initialize the onboard LED of the Raspberry Pi Pico W
led = Pin("LED", Pin.OUT)

while True:
    if flame_sensor.value() == 0:
        led.value(1)  # Turn on the LED
        print("** Fire detected!!! **")
    else:
        led.value(0)  # Turn off the LED
        print("No Fire detected")
```

(continues on next page)

(continued from previous page)

```
    time.sleep(0.1)  # Short delay to reduce CPU usage
```

Code Analysis

1. Importing Required Modules

   This part of the code imports necessary modules. `machine` is used for interacting with GPIO pins, and `time` provides functionality for delays.

   ```
   from machine import Pin
   import time
   ```

2. Initializing the Flame Sensor and LED

   Sets up the flame sensor and onboard LED. Pin 16 is configured as an input to read the flame sensor, and the onboard LED is set as an output.

   ```
   flame_sensor = Pin(16, Pin.IN)
   led = Pin("LED", Pin.OUT)
   ```

3. The Main Loop

   - An infinite loop checks the state of the flame sensor. If the sensor detects a flame (value 0), it turns on the LED and prints a message. Otherwise, it turns off the LED and prints a different message.
   - A delay of 0.1 seconds reduces CPU usage.

   ```
   while True:
       if flame_sensor.value() == 0:
           led.value(1)
           print("** Fire detected!!! **")
       else:
           led.value(0)
           print("No Fire detected")
       time.sleep(0.1)
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.5 Lesson 04: Gas Sensor Module (MQ-2)

In this lesson, you'll learn how to use the Raspberry Pi Pico W to read data from a gas sensor module (MQ-2) using MicroPython. We'll guide you through setting up an ADC on GPIO pin 26 to process analog signals from the MQ-2 sensor. You'll gain practical experience in continuously monitoring and printing sensor data to understand the presence of gases in the environment.

Required Components

In this project, we need the following components.

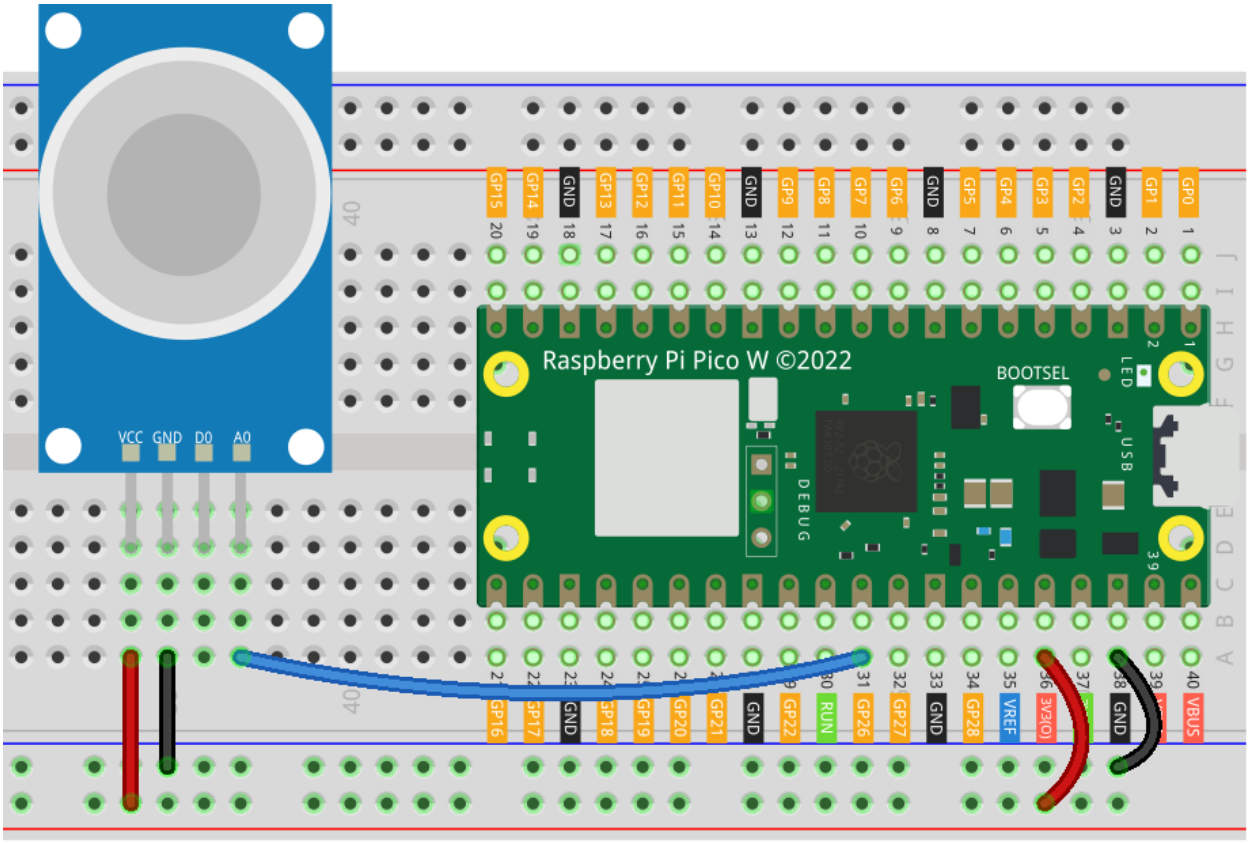
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Gas/Smoke Sensor Module (MQ2)	
Breadboard	

Wiring



Code

```
import machine
import utime

# Initialize an ADC object on GPIO pin 26.
# This is typically used for reading analog signals.
mq2_AO = machine.ADC(26)

# Continuously read and print sensor data.
while True:
    value = mq2_AO.read_u16()  # Read and convert analog value to 16-bit integer
    print("AO:", value)  # Print the analog value

    utime.sleep_ms(200)  # Wait for 200 milliseconds before the next read
```

Code Analysis

1. Importing Libraries:

   The code begins by importing necessary libraries: `machine` for hardware interactions, and `utime` for handling time-related tasks.

```
import machine
import utime
```

2. Initializing the MQ-2 Sensor:

   An ADC object is created on GPIO pin 26 to read analog signals from the MQ-2 sensor. The MQ-2 sensor outputs an analog signal which varies with the concentration of gas in the air.

```
mq2_AO = machine.ADC(26)
```

3. Reading Sensor Data in a Loop:

   The main loop of the program continuously reads the analog value from the sensor. The `read_u16` method is used to read the analog value and convert it to a 16-bit integer. This value is then printed out. The loop includes a delay (`utime.sleep_ms(200)`) to wait for 200 milliseconds before reading the sensor value again. This delay is crucial to prevent overwhelming the sensor and the microcontroller with rapid readings.

   Note: MQ2 is a heating-driven sensor that usually requires preheating before use. During the preheating period, the sensor typically reads high and gradually decreases until it stabilizes.

```
while True:
    value = mq2_AO.read_u16()  # Read and convert analog value to 16-bit integer
    print("AO:", value)  # Print the analog value
    utime.sleep_ms(200)  # Wait for 200 milliseconds before the next read
```

1.5.6 Lesson 05: Gyroscope & Accelerometer Module (MPU6050)

In this lesson, you will learn how to use the Raspberry Pi Pico W with the MPU6050 module, which combines a gyroscope and accelerometer. You'll discover how to connect the MPU6050 to the Raspberry Pi Pico W and read its acceleration and gyroscopic data using MicroPython. The lesson will guide you through writing a script to continuously display the X, Y, and Z values of both the accelerometer and gyroscope.

Required Components

In this project, we need the following components.

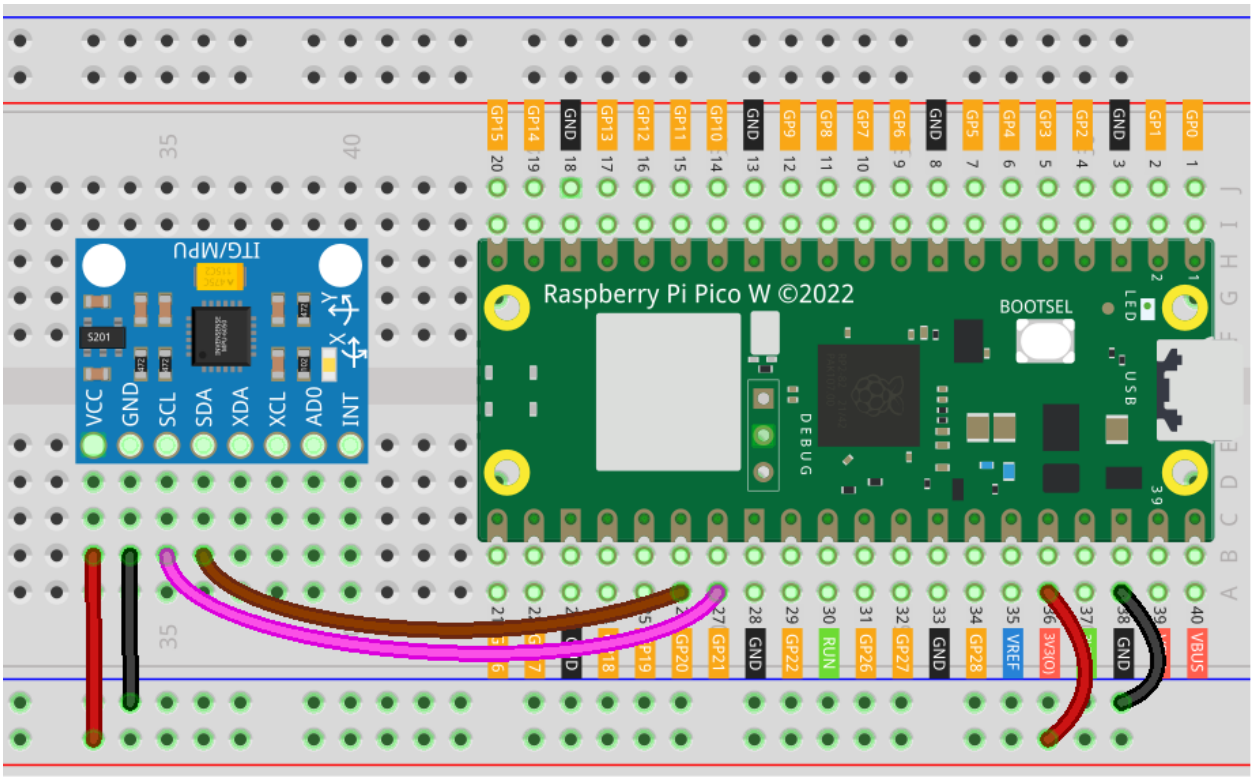
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Gyroscope & Accelerometer Module (MPU6050)	
Breadboard	

Wiring



Code

Note:

- Open the `05_mpu6050_module.py` file under the path of `universal-maker-sensor-kit-main/pico/Lesson_05_MPU6050_Module` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it. For detailed tutorials, please refer to *Open and Run Code Directly*.
- Here you need to use the `imu.py` and `vector3d.py`, please check if it has been uploaded to Pico W, for a detailed tutorial refer to *Upload the Libraries to Pico*.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.

```
# Import libraries
from imu import MPU6050
from machine import I2C, Pin
import time

# Initialize I2C for MPU6050
i2c = I2C(1, sda=Pin(20), scl=Pin(21), freq=400000)  # I2C bus 1, SDA pin 20, SCL pin 21,
↪ 400kHz

# Create MPU6050 object
mpu = MPU6050(i2c)
```

(continues on next page)

(continued from previous page)

```
# Main loop to read and print sensor data
while True:
    # Print accelerometer data (x, y, z)
    print("-" * 50)
    print("x: %s, y: %s, z: %s" % (mpu.accel.x, mpu.accel.y, mpu.accel.z))
    time.sleep(0.1)

    # Print gyroscope data (x, y, z)
    print("X: %s, Y: %s, Y: %s" % (mpu.gyro.x, mpu.gyro.y, mpu.gyro.z))
    time.sleep(0.1)

    # Delay between readings
    time.sleep(0.5)
```

Code Analysis

1. Importing Libraries and Initializing I2C

   The code starts by importing necessary libraries. The `imu` library is used to read the values of the MPU6050 sensor, and `machine` allows controlling the hardware features of the Raspberry Pi Pico W. I2C is initialized using specific pins (SDA and SCL) for data communication.

   For more information about the `imu` library, please visit .

   ```
   from imu import MPU6050
   from machine import I2C, Pin
   import time

   i2c = I2C(1, sda=Pin(20), scl=Pin(21), freq=400000)
   ```

2. Creating MPU6050 Object

   An object of the MPU6050 sensor is created by passing the initialized I2C. This object will be used to access sensor data.

   ```
   mpu = MPU6050(i2c)
   ```

3. Reading and Printing Sensor Data in a Loop

   The code then enters an infinite loop where it continually reads and prints accelerometer and gyroscope data. `time.sleep` is used to create a delay between successive readings.

   ```
   while True:
       print("-" * 50)
       print("x: %s, y: %s, z: %s" % (mpu.accel.x, mpu.accel.y, mpu.accel.z))
       time.sleep(0.1)
       print("X: %s, Y: %s, Y: %s" % (mpu.gyro.x, mpu.gyro.y, mpu.gyro.z))
       time.sleep(0.1)
       time.sleep(0.5)
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.7 Lesson 06: Hall Sensor Module

In this lesson, you will learn how to use the Raspberry Pi Pico W to measure magnetic fields with a Hall effect sensor. By connecting the sensor to the Pico W, you'll discover how to read analog values and interpret them to detect the presence and type of magnetic poles. This engaging project is perfect for those starting out, as it provides practical experience with analog input processing on the Raspberry Pi Pico W using MicroPython. You'll understand how to set up the sensor, read its data, and apply conditional logic to determine the magnetic pole type, enhancing your skills in electronics and programming.

Required Components

In this project, we need the following components.

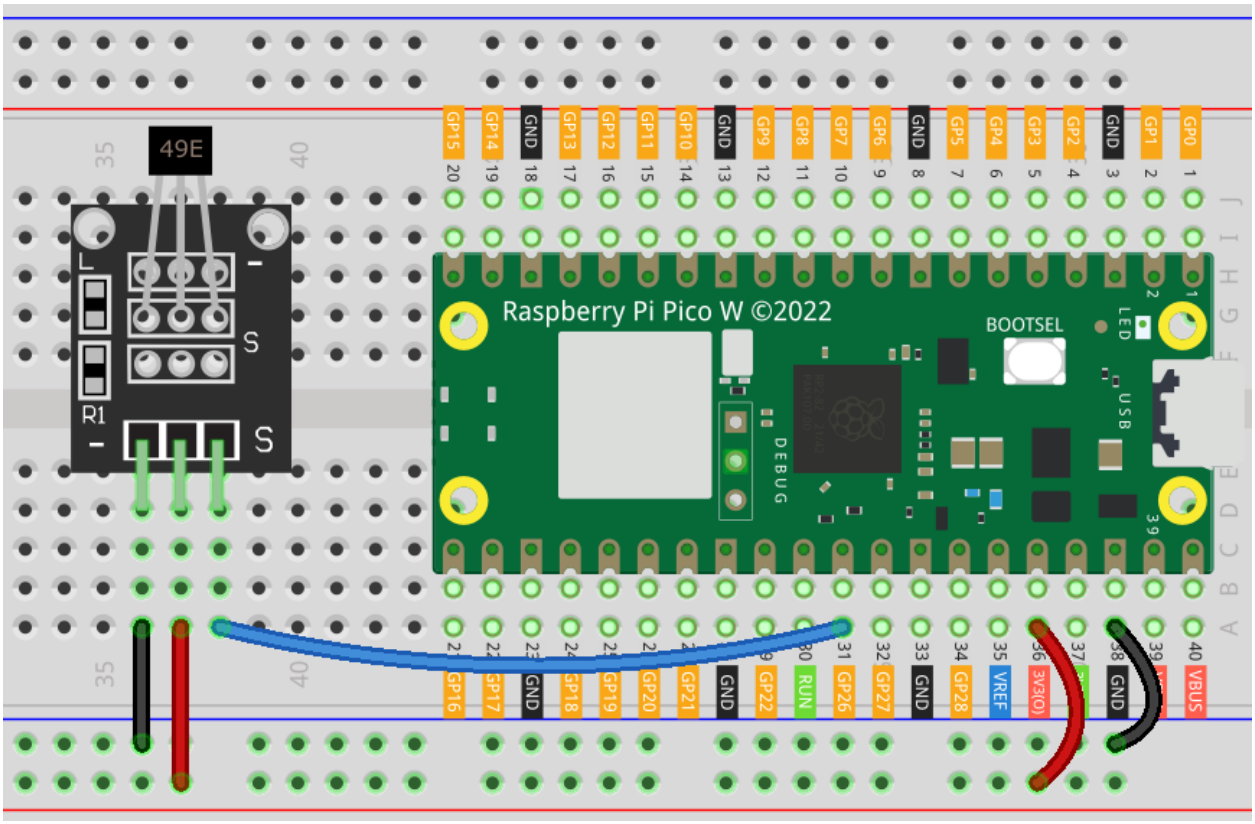
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Hall Sensor Module	-
Breadboard	

Wiring



Code

```
import machine
import utime

# Initialize an ADC on GPIO pin 26 for Hall effect sensor readings.
hall_sensor = machine.ADC(26)

# Continuously monitor and process Hall sensor data.
while True:
    # Read the analog value from the sensor and convert to a 16-bit integer.
    value = hall_sensor.read_u16()
    print(value, end="")  # Output the raw sensor value.

    # Detect and print the type of magnetic pole based on the sensor reading.
    if value >= 48000:
        print(" - South pole detected", end="")
    elif value <= 18000:
        print(" - North pole detected", end="")

    print()

    # Wait 200 milliseconds before the next sensor reading
```

(continues on next page)

(continued from previous page)

```
    utime.sleep_ms(200)
```

Code Analysis

1. **Import Necessary Modules**:

   This section imports the required modules. `machine` is used for hardware interfaces, and `utime` provides timing functions.

   ```
   import machine
   import utime
   ```

2. **Initialize the Hall Sensor**:

   Here, we initialize an ADC (Analog-to-Digital Converter) on GPIO pin 26. This is where the Hall sensor is connected. The `machine.ADC` function is used to read analog values from the sensor.

   ```
   hall_sensor = machine.ADC(26)
   ```

3. **Main Loop for Sensor Reading**:

   In this loop, `hall_sensor.read_u16()` reads the sensor's analog value and converts it to a 16-bit integer. This loop will run indefinitely.

   ```
   while True:
       value = hall_sensor.read_u16()
   ```

4. **Processing Sensor Data**:

   After reading the value, the code checks whether it falls within certain thresholds to determine if a magnetic North or South pole is detected. The values `48000` and `18000` are threshold values that represent the presence of different magnetic poles. You can adjust the threshold values representing the South and North poles according to actual conditions.

   The Hall sensor module is equipped with a 49E linear Hall effect sensor, which can measure the polarity of the magnetic field's north and south poles as well as the relative strength of the magnetic field. If you place a magnet's south pole near the side marked with 49E (the side with text engraved on it), the value read by the code will increase linearly in proportion to the applied magnetic field strength. Conversely, if you place a north pole near this side, the value read by the code will decrease linearly in proportion to that magnetic field strength. For more details, please refer to *Hall Sensor Module*.

   ```
   print(value, end="")
   if value >= 48000:
       print(" - South pole detected", end="")
   elif value <= 18000:
       print(" - North pole detected", end="")
   print()
   ```

5. **Delay Between Readings**:

   This line introduces a 200-millisecond delay before the next reading, using `utime.sleep_ms`. This prevents the loop from running too quickly and flooding the output.

   ```
   utime.sleep_ms(200)
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.8 Lesson 07: Infrared Speed Sensor Module

In this lesson, you will learn how to use the Raspberry Pi Pico W to interface with an infrared speed sensor module. By connecting the sensor to GPIO 16, you will detect obstructions in real-time. The program monitors the sensor output, and when an obstruction is detected, it prints "Obstruction detected" to the console. If there's no obstruction, it prints "Unobstructed."

Required Components

In this project, we need the following components.

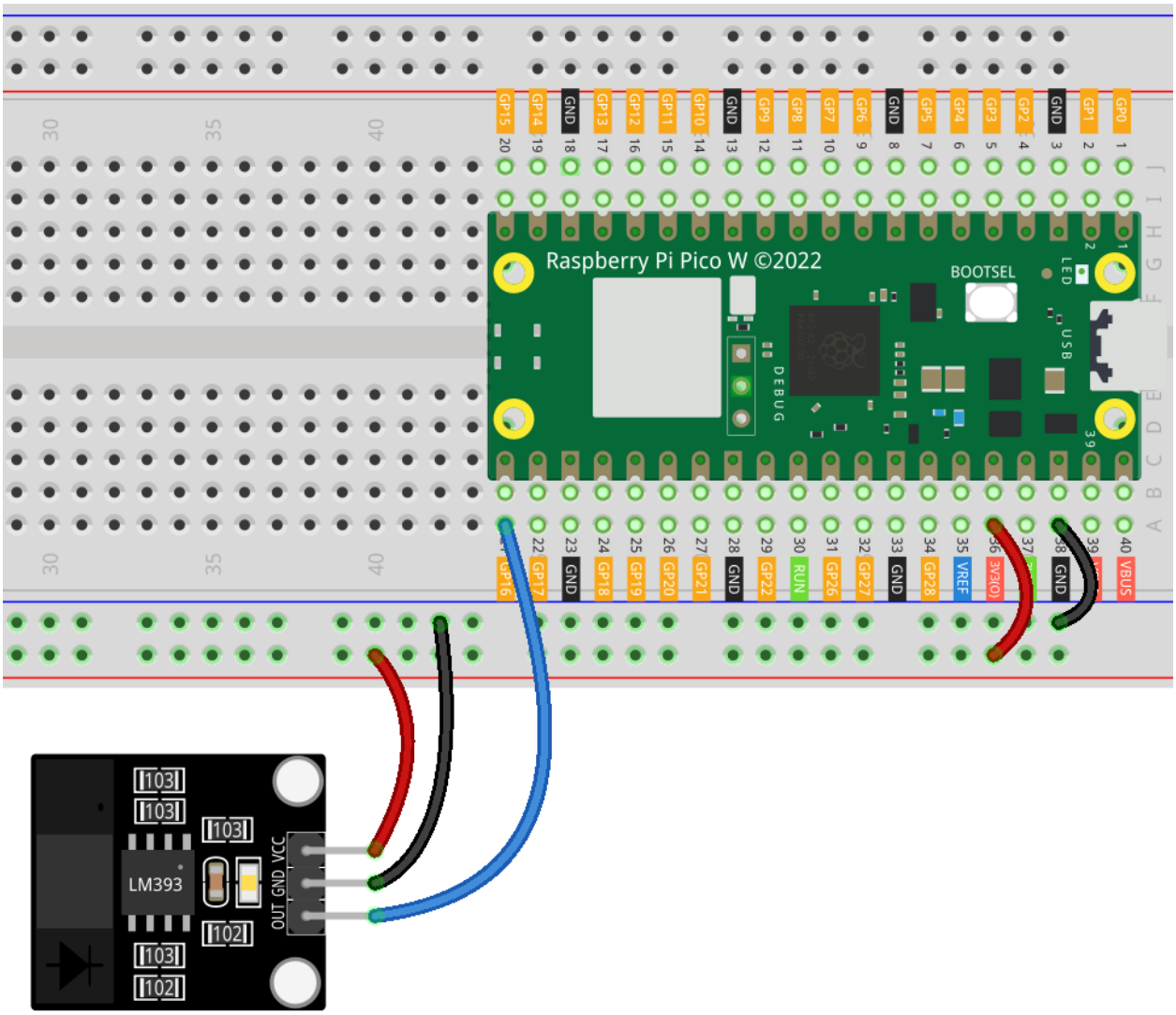
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Infrared Speed Sensor Module	
Breadboard	

Wiring



Code

```
from machine import Pin
import time

# Set GPIO 16 as an input pin to read the speed sensor
speed_sensor = Pin(16, Pin.IN)

while True:
    if speed_sensor.value() == 1:
        print("Obstruction detected")
    else:
        print("Unobstructed")

    time.sleep(0.1)  # Short delay to reduce CPU usage
```

Code Analysis

1. **Import Libraries**:

   This code begins by importing necessary libraries. The `machine` library is used to interact with the GPIO pins, and the `time` library is for adding delays in the program.

```
from machine import Pin
import time
```

2. **Sensor Configuration**:

   The infrared speed sensor is connected to GPIO 16. It's set as an input, meaning the Pi Pico W will read data from this pin.

```
speed_sensor = Pin(16, Pin.IN)
```

3. **Main Loop**:

   The `while True:` loop creates an infinite loop. Inside this loop, the program continuously checks the sensor's value.

   If `speed_sensor.value()` is 1, it means the sensor detects an obstruction. If it is 0, then there is no obstruction.

```
while True:
    if speed_sensor.value() == 1:
        print("Obstruction detected")
    else:
        print("Unobstructed")
```

4. **Delay to Reduce CPU Usage**:

   A short delay of 0.1 seconds is introduced in each iteration of the loop. This reduces the CPU usage by preventing the loop from running too rapidly.

```
time.sleep(0.1)
```

5. **More**

   If an encoder is mounted on the motor, the rotational speed of the motor can be calculated by counting the number of times an obstruction passes the sensor within a specific period.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.9 Lesson 08: IR Obstacle Avoidance Sensor Module

In this lesson, you'll learn how to use the Raspberry Pi Pico W with an IR Obstacle Avoidance Sensor Module. We'll walk you through setting up the sensor and writing a MicroPython script that continuously reads its value to detect obstacles. By monitoring changes in the sensor data, you'll grasp how to use it for basic obstacle detection.

Required Components

In this project, we need the following components.

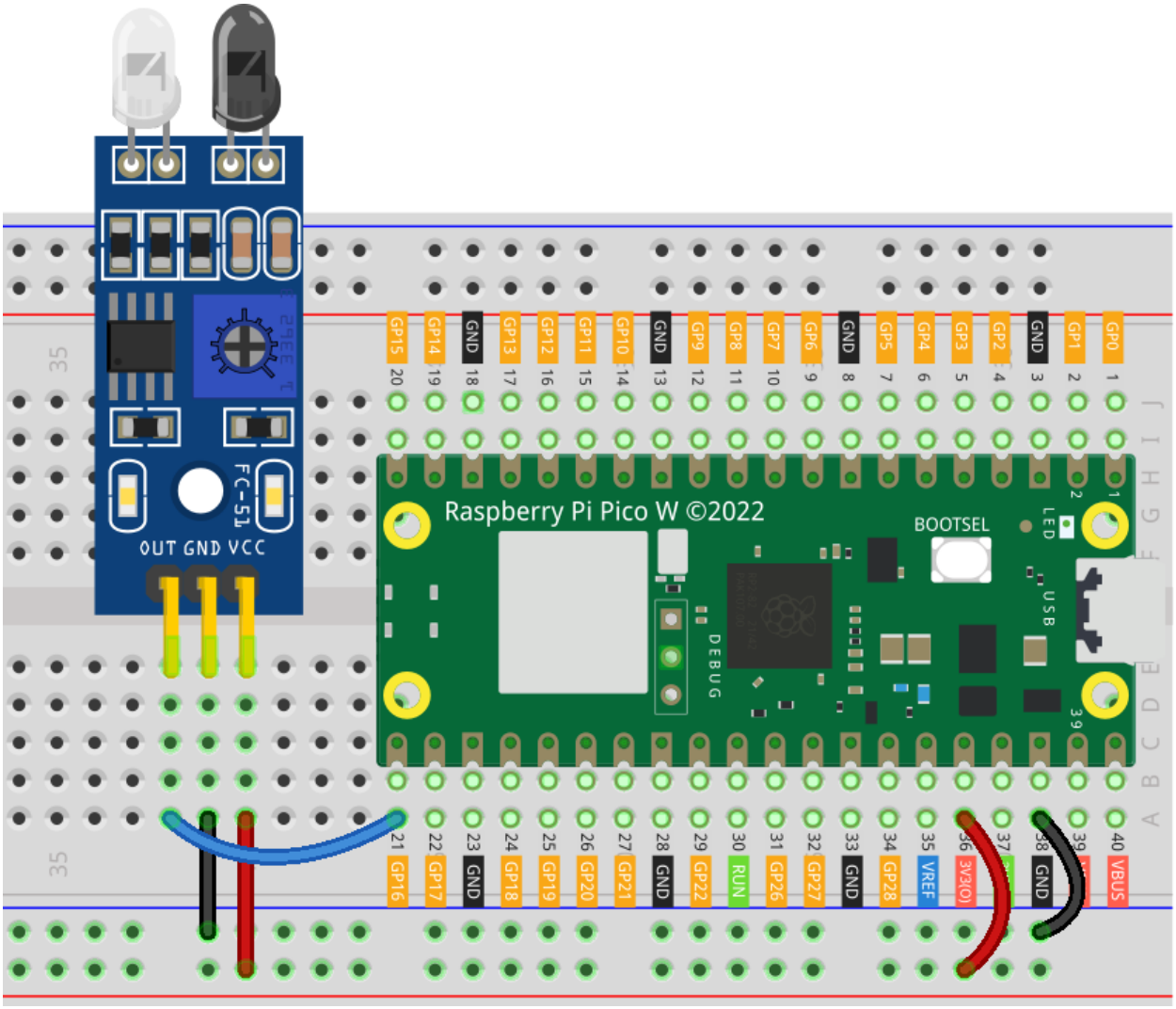
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
IR Obstacle Avoidance Sensor Module	
Breadboard	

Wiring



Code

```
from machine import Pin
import time

# Initialize obstacle avoidance sensor connected to pin 16 as input
obstacle_avoidance_sensor = Pin(16, Pin.IN)

while True:
    # Read and print the value of the obstacle avoidance sensor
    print(obstacle_avoidance_sensor.value())

    # Wait for 0.1 seconds before the next read
    time.sleep(0.1)
```

Code Analysis

1. Importing Libraries

   The `machine` module is imported to interact with the GPIO pins, and the `time` module is used for adding delays.

   ```
   from machine import Pin
   import time
   ```

2. Setting Up the Sensor

   The obstacle avoidance sensor is set up as an input device on GPIO pin 16. The `Pin.IN` parameter configures the pin as an input.

   ```
   obstacle_avoidance_sensor = Pin(16, Pin.IN)
   ```

3. Reading Sensor Data in a Loop

   The `while True:` loop continuously checks the sensor's output. If the sensor detects an obstacle, it returns `0`, which is printed out. The `time.sleep(0.1)` adds a small delay to make the readings more manageable.

   ```
   while True:
       print(obstacle_avoidance_sensor.value())
       time.sleep(0.1)
   ```

   Note: If the sensor is not working properly, adjust the IR transmitter and receiver to make them parallel. Additionally, you can adjust the detection range using the built-in potentiometer.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.10 Lesson 09: Joystick Module

In this lesson, you'll learn how to interface with and read data from a joystick module using the Raspberry Pi Pico W. You'll explore initializing and reading analog values from the X and Y axes of the joystick, as well as handling digital input from its switch using MicroPython. This lesson is ideal for beginners, offering practical experience in reading and interpreting analog and digital inputs on the Raspberry Pi Pico W.

Required Components

In this project, we need the following components.

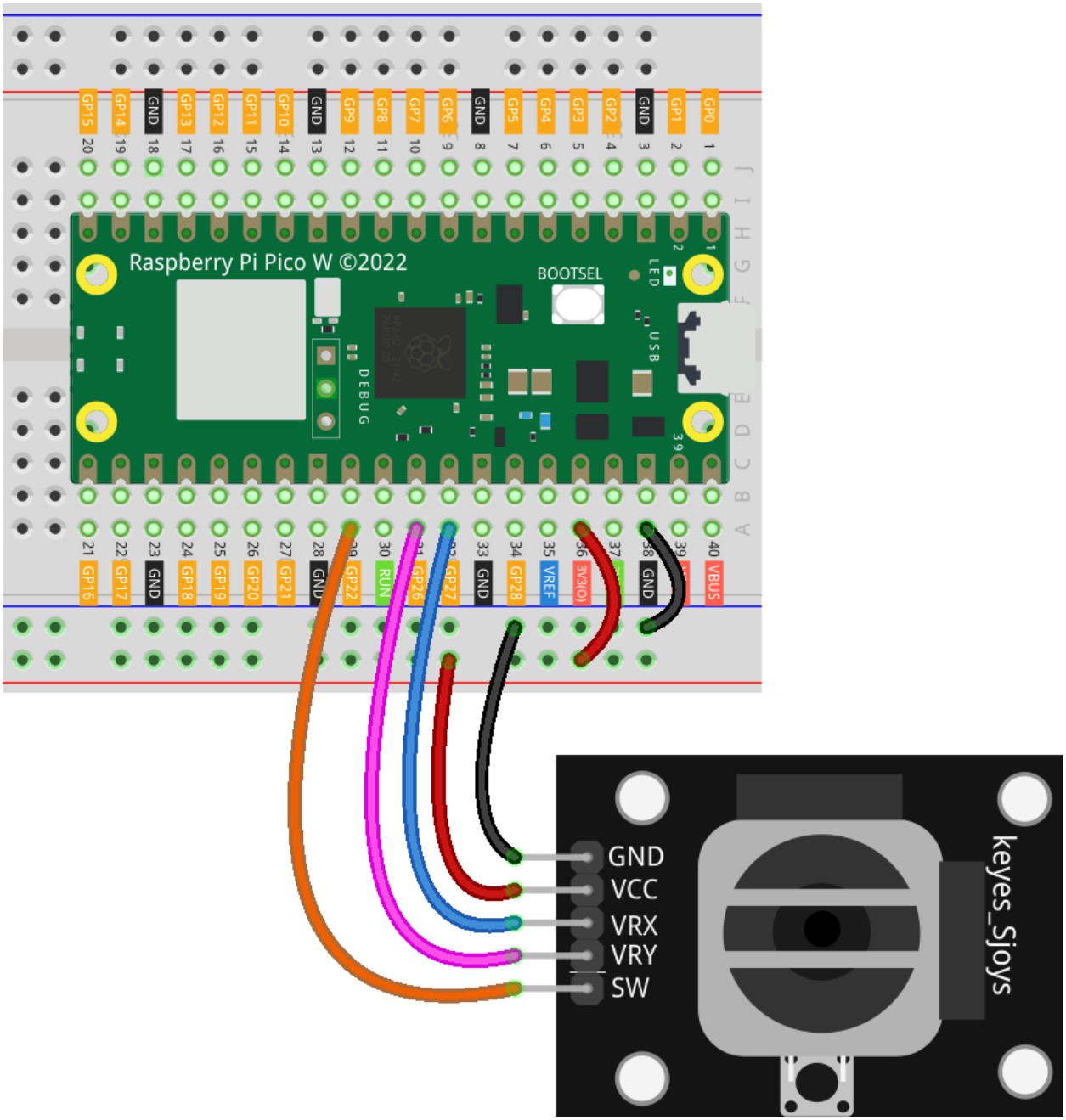
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Joystick Module	
Breadboard	

Wiring


Raspberry Pi Pico W ©2022
BOOTSEL
LED
USB
DEBUG
GND
VCC
VRX
VRY
SW
keyes_Sjoys

Code

```
import machine  # Import hardware control module
import time  # Import time module

# Initialize X and Y axes of the joystick
x_joystick = machine.ADC(27)
y_joystick = machine.ADC(26)

# Initialize joystick switch with pull-up resistor
z_switch = machine.Pin(22, machine.Pin.IN, machine.Pin.PULL_UP)

while True:  # Continuous reading loop
    x_value = x_joystick.read_u16()  # Read X-axis value
    y_value = y_joystick.read_u16()  # Read Y-axis value
    z_value = z_switch.value()  # Read switch state

    # Print joystick values and switch state
    print("X: ", x_value, " Y: ", y_value)
    print("SW: ", z_value)

    time.sleep_ms(200)  # Loop every 200 milliseconds
```

Code Analysis

1. Import Libraries

   The `machine` and `time` modules are imported for hardware control and time functions.

   ```
   import machine  # Import hardware control module
   import time  # Import time module
   ```

2. Initialize Joystick Axes

   The joystick's X and Y axes are connected to analog pins (27 and 26 respectively). These pins are initialized as ADC (Analog to Digital Converter) objects.

   ```
   x_joystick = machine.ADC(27)
   y_joystick = machine.ADC(26)
   ```

3. Initialize Joystick Switch

   The joystick's switch is connected to pin 22. It's set as an input with a pull-up resistor. When the button is not pressed, it reads high (1), and when pressed, it reads low (0).

   ```
   z_switch = machine.Pin(22, machine.Pin.IN, machine.Pin.PULL_UP)
   ```

4. Main Loop

   - An infinite loop continuously reads values from the joystick.
   - `read_u16` method is used to read 16-bit values from the X and Y axes.
   - `value()` method is used to read the state of the switch.
   - The values are then printed, and the loop pauses for 200 milliseconds.

```
while True:  # Continuous reading loop
    x_value = x_joystick.read_u16()  # Read X-axis value
    y_value = y_joystick.read_u16()  # Read Y-axis value
    z_value = z_switch.value()  # Read switch state

    # Print joystick values and switch state
    print("X: ", x_value, " Y: ", y_value)
    print("SW: ", z_value)

    time.sleep_ms(200)  # Loop every 200 milliseconds
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.11 Lesson 10: PCF8591 ADC DAC Converter Module

In this lesson, you'll learn how to connect the Raspberry Pi Pico W with the PCF8591 ADC DAC Converter Module using MicroPython. You'll establish an I2C connection, initialize the PCF8591 module, and read analog values from its channels. This hands-on session will deepen your grasp of analog-to-digital conversion and I2C communication on the Raspberry Pi Pico W. The module's potentiometer is connected to AIN0 using jumper caps, and the D2 LED on the module is connected to AOUT, so you can see that the brightness of D2 LED changes as you rotate the potentiometer.

Required Components

In this project, we need the following components.

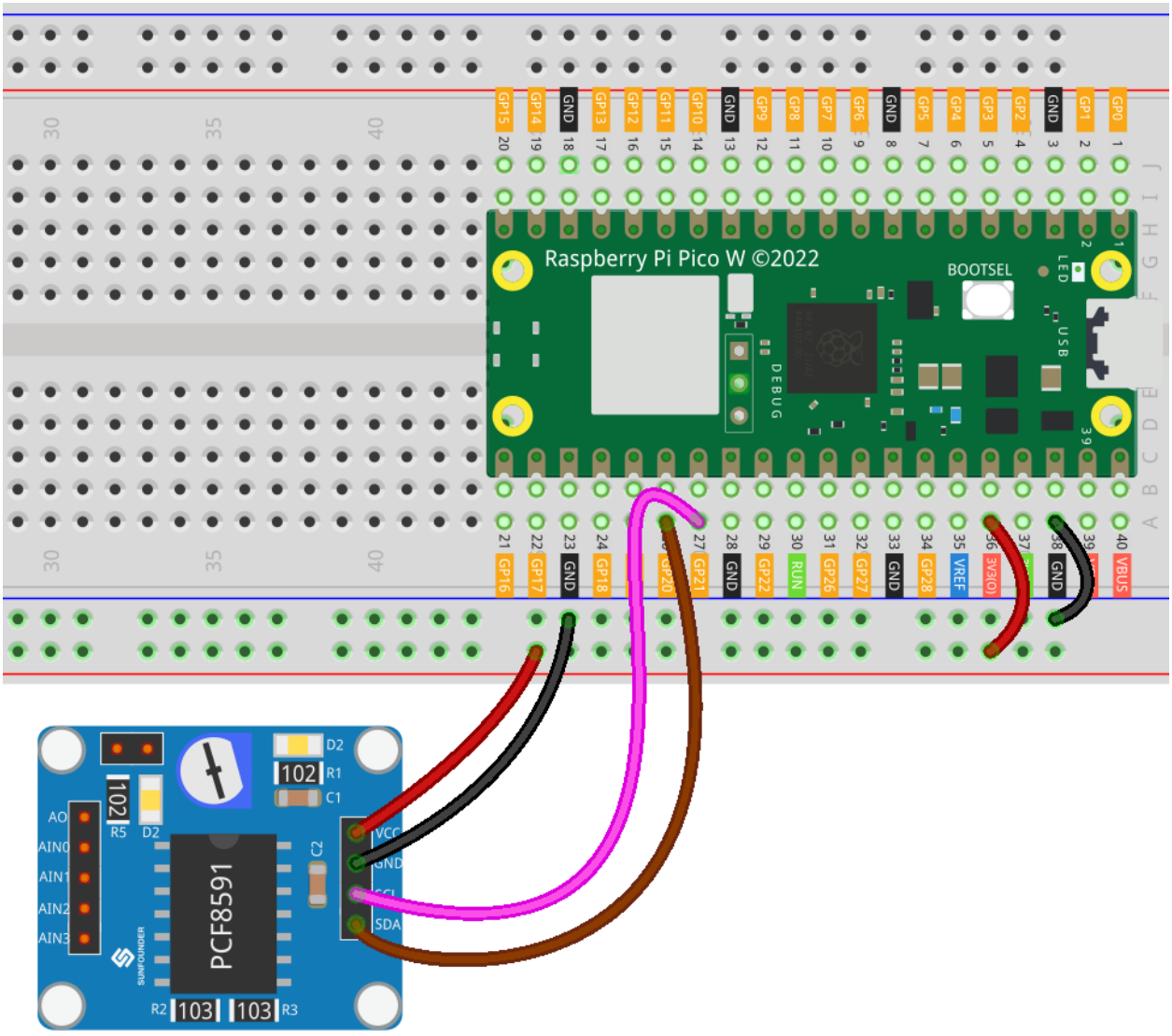
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
PCF8591 ADC DAC Converter Module	
Breadboard	

Wiring



Code

Note:

- Open the `10_pcf8591_module.py` file under the path of `universal-maker-sensor-kit-main/pico/Lesson_10_PCF8591_Module` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it. For detailed tutorials, please refer to *Open and Run Code Directly*.
- Here you need to use the `PCF8591.py`, please check if it has been uploaded to Pico W, for a detailed tutorial refer to *Upload the Libraries to Pico*.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.

```
from machine import I2C, Pin
import time
from PCF8591 import PCF8591

# Setup I2C connection on pins 20 (SDA) and 21 (SCL)
i2c = I2C(0, sda=Pin(20), scl=Pin(21))

# Initialize the PCF8591 module at address 0x48
pcf8591 = PCF8591(0x48, i2c)  # Adjust the address if needed

# Check if the PCF8591 module is connected
if pcf8591.begin():
    print("PCF8591 found")

# Main loop to read analog values
while True:
    # Read and print the analog value from channel AIN0
    AIN0 = pcf8591.analog_read(PCF8591.AIN0)
    print("AIN0 ", AIN0)  # PCF8591.CHANNEL_0 can also be used
    # Additional channels can be read by uncommenting the following lines
    # print("AIN1 ", pcf8591.analog_read(PCF8591.AIN1))
    # print("AIN2 ", pcf8591.analog_read(PCF8591.AIN2))
    # print("AIN3 ", pcf8591.analog_read(PCF8591.AIN3))

    # Write the value back to AOUT. This will change the brightness of the D2 LED on the␣
↪module.
    pcf8591.analog_write(AIN0)

    # Wait for 0.2 seconds before the next read
    time.sleep(0.2)
```

Code Analysis

1. Importing Libraries and Setting Up I2C

    - The `machine` module is imported to use I2C communication and `Pin` class.
    - The `time` module is imported for adding delays in the program.
    - The `PCF8591` library is imported for easy interaction with the PCF8591 module. For more information about the `PCF8591` library, please visit .

```
from machine import I2C, Pin
import time
from PCF8591 import PCF8591
```

2. Initializing I2C Connection

    I2C communication is initialized using SDA (Serial Data) and SCL (Serial Clock) pins. The Raspberry Pi Pico W uses GPIO 20 and 21 for this purpose.

```
i2c = I2C(0, sda=Pin(20), scl=Pin(21))
```

3. Initializing the PCF8591 Module

    The PCF8591 module is initialized with its I2C address (0x48). This address might need adjustment depending on the module's configuration.

```
pcf8591 = PCF8591(0x48, i2c)  # Adjust the address if needed
```

4. Checking Connection

    The program checks if the PCF8591 module is connected correctly.

```
if pcf8591.begin():
    print("PCF8591 found")
```

5. Main Loop for Reading Analog Values

    - The program enters an infinite loop, continuously reading the analog value from channel AIN0.
    - The `analog_read` function is used to read the value from a specified channel.
    - The `analog_write` function is used to write the value to AOUT.
    - Jumper caps link the module's potentiometer to AIN0, and the D2 LED is connected to AOUT. So the brightness of the LED changes as the potentiometer is rotated. Please refer to the PCF8591 module *schematic* for details.
    - A delay of 0.2 seconds is added between reads to stabilize the output.

```
while True:
    # Read and print the analog value from channel AIN0
    AIN0 = pcf8591.analog_read(PCF8591.AIN0)
    print("AIN0 ", AIN0)  # PCF8591.CHANNEL_0 can also be used
    # Additional channels can be read by uncommenting the following lines
    # print("AIN1 ", pcf8591.analog_read(PCF8591.AIN1))
    # print("AIN2 ", pcf8591.analog_read(PCF8591.AIN2))
    # print("AIN3 ", pcf8591.analog_read(PCF8591.AIN3))

    # Write the value back to AOUT. This will change the brightness of the D2 LED␣
```

(continues on next page)

(continued from previous page)

```
↪on the module.
    pcf8591.analog_write(AIN0)

    # Wait for 0.2 seconds before the next read
    time.sleep(0.2)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.12 Lesson 11: Photoresistor Module

In this lesson, you'll learn how to connect a photoresistor module to the Raspberry Pi Pico W in order to measure light intensity. By linking the photoresistor to the analog input, you can read different analog values that correspond to varying light levels. This project is ideal for beginners and provides hands-on experience in utilizing analog inputs on the Raspberry Pi Pico W with MicroPython.

Required Components

In this project, we need the following components.

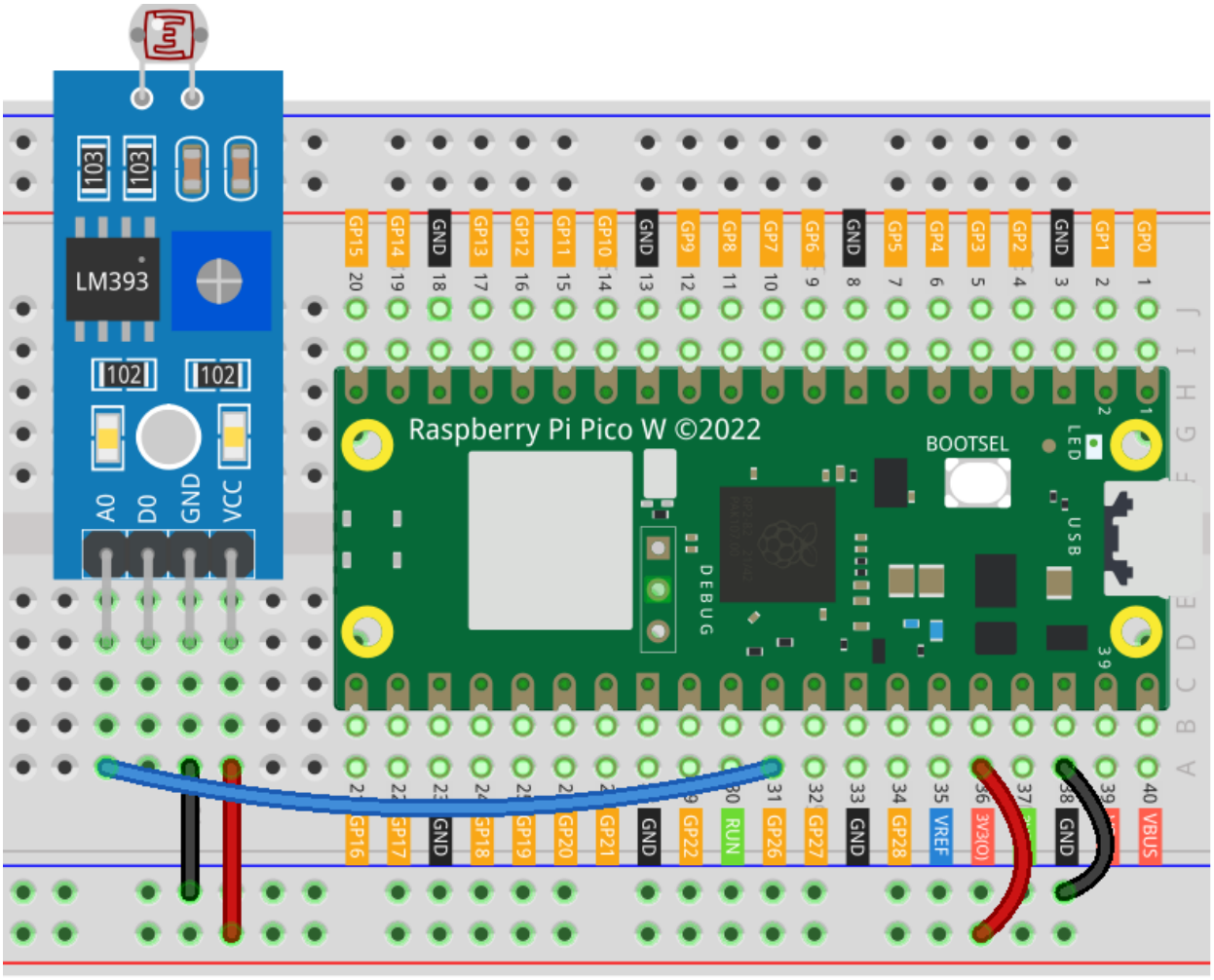
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Photoresistor Module	
Breadboard	

Wiring



Code

```
import machine  # Hardware control library
import time  # Time control library

photoresistor = machine.ADC(26)  # Initialize ADC on pin 26

while True:
    value = photoresistor.read_u16()  # Read analog value
    print(value)  # Print the value

    time.sleep_ms(200)  # Delay of 200 ms between reads
```

Code Analysis

1. **Importing Libraries**:

   The code begins by importing necessary libraries. The `machine` library is used for controlling hardware components, and the `time` library is used for managing time-related tasks such as delays.

```
import machine  # Hardware control library
import time  # Time control library
```

2. **Initializing the Photoresistor**:

   Here, we initialize the photoresistor. We use the `machine.ADC` class to create an ADC object on pin 26, where the photoresistor is connected. The ADC object will be used to read the analog values from the photoresistor.

```
photoresistor = machine.ADC(26)  # Initialize ADC on pin 26
```

3. **Reading from the Photoresistor**:

   In this loop, the code continuously reads the analog value from the photoresistor using `photoresistor.read_u16()`. This method reads the value as a 16-bit unsigned integer. The value is then printed to the console.

```
while True:
    value = photoresistor.read_u16()  # Read analog value
    print(value)  # Print the value
```

4. **Adding a Delay**:

   To prevent the code from running too quickly and flooding the console with data, a delay of 200 milliseconds is introduced after each read using `time.sleep_ms(200)`.

```
time.sleep_ms(200)  # Delay of 200 ms between reads
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.13 Lesson 12: PIR Motion Module (HC-SR501)

In this lesson, you'll learn how to connect a PIR Motion Sensor to the Raspberry Pi Pico W. You'll discover how to configure the sensor for motion detection and use basic MicroPython code to react to movement. By monitoring the PIR sensor, you'll gain experience in managing digital inputs and creating a simple security measure or automation trigger.

Required Components

In this project, we need the following components.

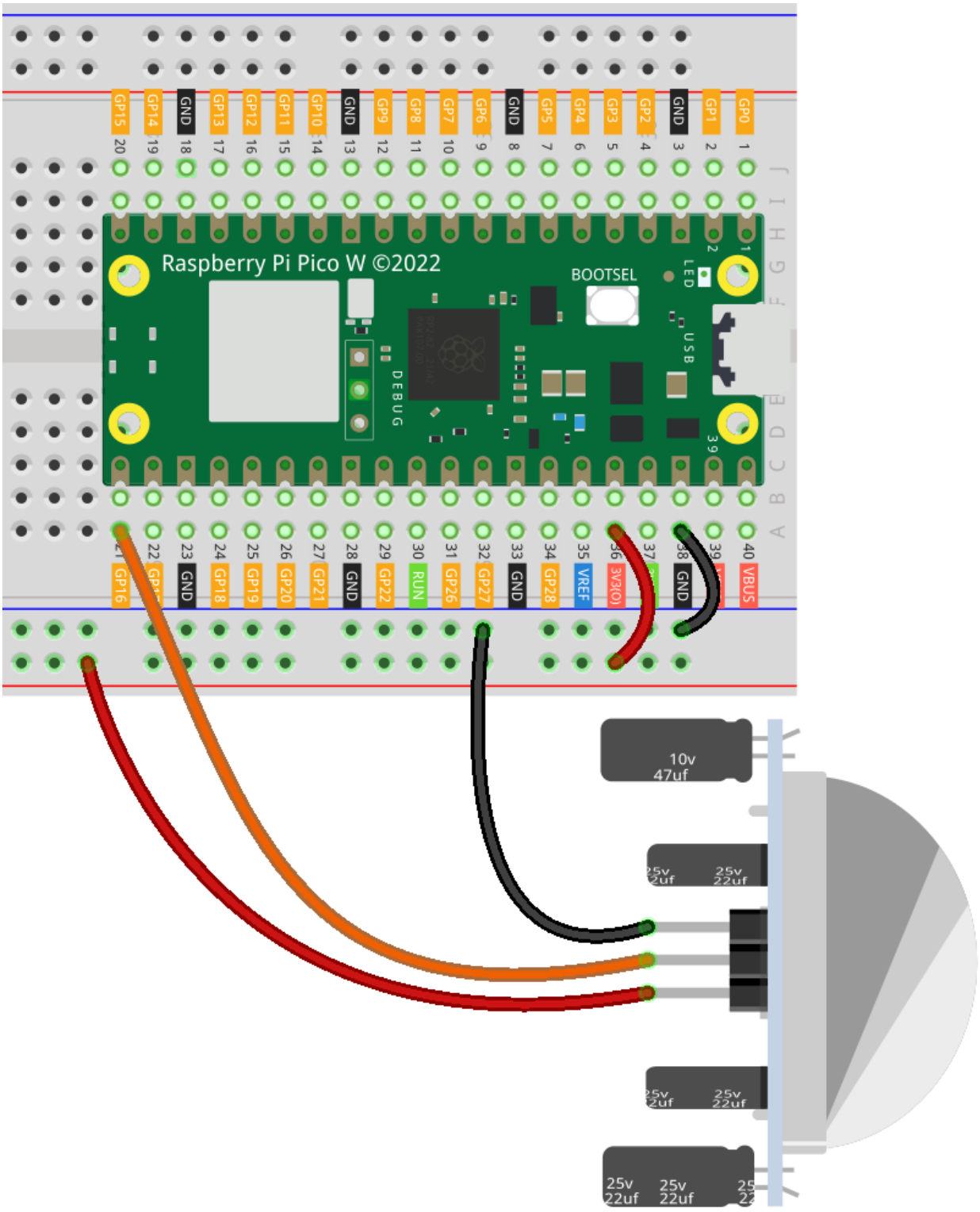
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
PIR Motion Module (HC-SR501)	-
Breadboard	

Wiring


Raspberry Pi Pico W ©2022
BOOTSEL
LED
USB
DEBUG
GP15
GP14
GND
GP13
GP12
GP11
GP10
GND
GP9
GP8
GP7
GP6
GND
GP5
GP4
GP3
GP2
GND
GP1
GP0
GP16
GND
GP18
GP19
GP20
GP21
GND
GP22
RUN
GP26
GP27
GND
GP28
VREF
3V3(O)
GND
VBUS
10v
47uf
25v
22uf

Code

```
from machine import Pin
import time

# Initialize PIR sensor connected to pin 16 as input
pir_sensor = Pin(16, Pin.IN)

while True:
    # Check the PIR sensor value
    if pir_sensor.value() == 0:
        print("Monitoring...")  # No motion detected
    else:
        print("Somebody here!")  # Motion detected

    time.sleep(0.1)  # Short delay of 0.1 seconds to reduce CPU usage
```

Code Analysis

1. Importing modules

   The `machine` module is imported to use the `Pin` class for GPIO pin control. The `time` module is imported for creating delays in the loop.

   ```
   from machine import Pin
   import time
   ```

2. Initializing the PIR sensor

   The PIR sensor is connected to GPIO pin 16 of the Raspberry Pi Pico W. It is set as an input device because it sends data to the microcontroller.

   ```
   # Initialize PIR sensor connected to pin 16 as input
   pir_sensor = Pin(16, Pin.IN)
   ```

3. Main loop

   The `while True` loop makes the code run continuously. Inside this loop, the PIR sensor's value is checked. If the value is `0`, it means no motion is detected. Otherwise, motion is detected. A delay of 0.1 seconds is added to reduce CPU usage and prevent the code from running too fast.

   ```
   while True:
       # Check the PIR sensor value
       if pir_sensor.value() == 0:
           print("Monitoring...")  # No motion detected
       else:
           print("Somebody here!")  # Motion detected

       time.sleep(0.1)  # Short delay of 0.1 seconds to reduce CPU usage
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.14 Lesson 13: Potentiometer Module

In this lesson, you'll learn how to use a potentiometer with the Raspberry Pi Pico W to measure analog values. The potentiometer, which is a variable resistor, lets you adjust the voltage that the Raspberry Pi Pico W reads on one of its analog input pins. By turning the knob of the potentiometer, you'll observe changes in the input value. This project offers a basic grasp of analog inputs and their application in electronic projects, making it an ideal entry point for beginners in electronics and MicroPython programming.

Required Components

In this project, we need the following components.

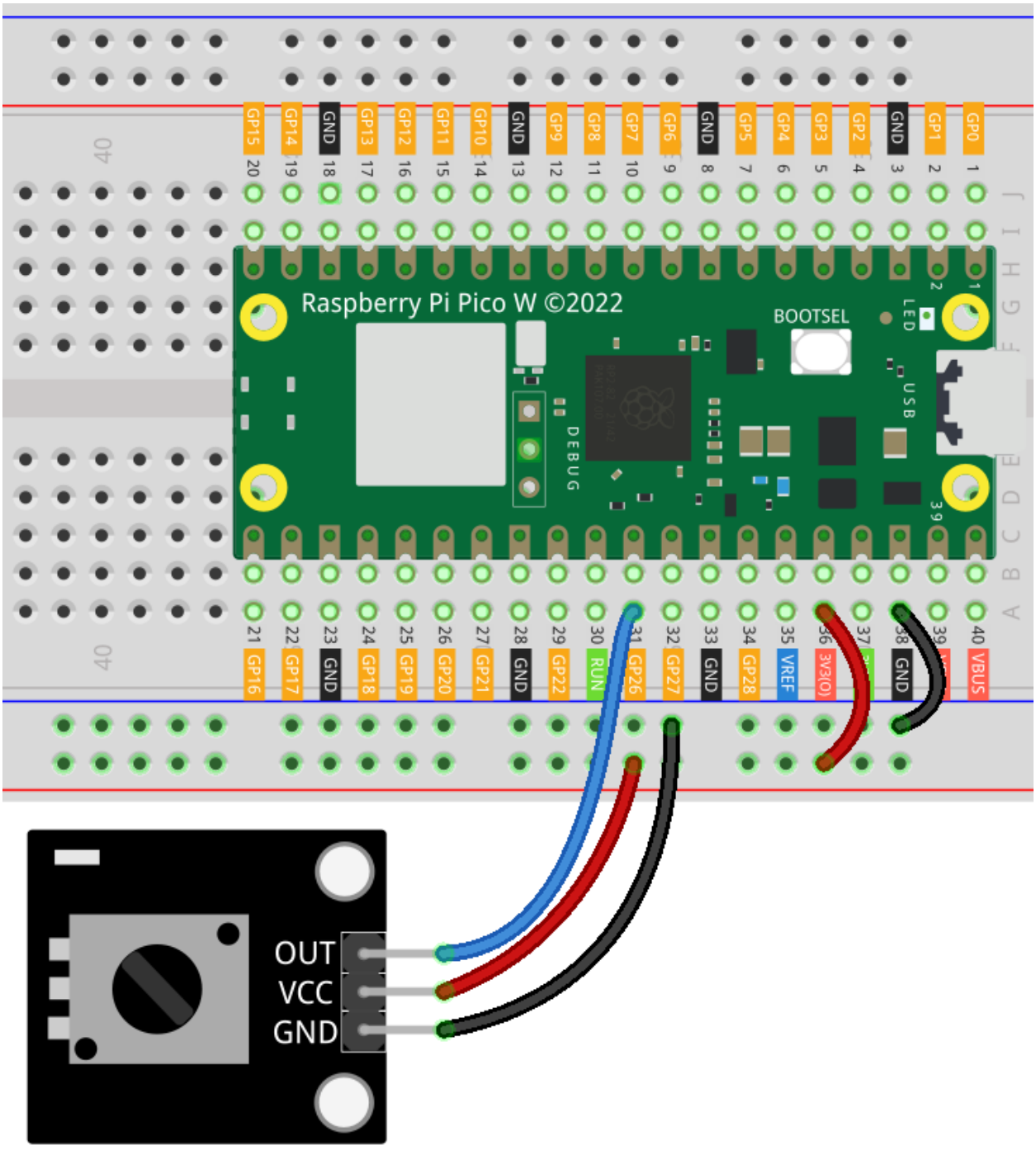
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Potentiometer Module	
Breadboard	

Wiring


Raspberry Pi Pico W ©2022
BOOTSEL
LED
USB
DEBUG
GP15 GP14 GND GP13 GP12 GP11 GP10 GND GP9 GP8 GP7 GP6 GND GP5 GP4 GP3 GP2 GND GP1 GP0
GP16 GP17 GND GP18 GP19 GP20 GP21 GND GP22 RUN GP26 GP27 GND GP28 VREF 3V3(O) GND VBUS
OUT
VCC
GND

Code

```
import machine  # Hardware control library
import time  # Time control library

potentiometer = machine.ADC(26)  # Initialize ADC on pin 26

while True:
    value = potentiometer.read_u16()  # Read analog value
    print(value)  # Print the value

    time.sleep_ms(200)  # Delay of 200 ms between reads
```

Code Analysis

1. Import Libraries

   First, the necessary libraries are imported. `machine` is for hardware control, and `time` is for managing delays.

   ```
   import machine  # Hardware control library
   import time     # Time control library
   ```

2. Initialize ADC (Analog to Digital Converter)

   The photoresistor is connected to pin 26 of the Pico W. This pin is initialized as an ADC pin to read analog values.

   ```
   potentiometer = machine.ADC(26)  # Initialize ADC on pin 26
   ```

3. Reading and Printing the Analog Value The code enters an infinite loop (`while True:`) where it continually reads the analog value from the photoresistor using `potentiometer.read_u16()` and prints it.

   ```
   while True:
       value = potentiometer.read_u16()  # Read analog value
       print(value)                      # Print the value
   ```

4. Adding a Delay

   To prevent the loop from running too fast, a delay of 200 milliseconds is introduced using `time.sleep_ms(200)`. This gives a readable output and reduces processor load.

   ```
   time.sleep_ms(200)               # Delay of 200 ms between reads
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.15 Lesson 14: Pulse Oximeter and Heart Rate Sensor Module (MAX30102)

In this lesson, you will learn how to use the Raspberry Pi Pico W to interface with the MAX30102 pulse oximeter and heart rate sensor. You will gain knowledge on setting up I2C communication, configuring the sensor, and reading raw data from the sensor. By observing changes in the data, you can obtain heartbeat information.

Required Components

In this project, we need the following components.

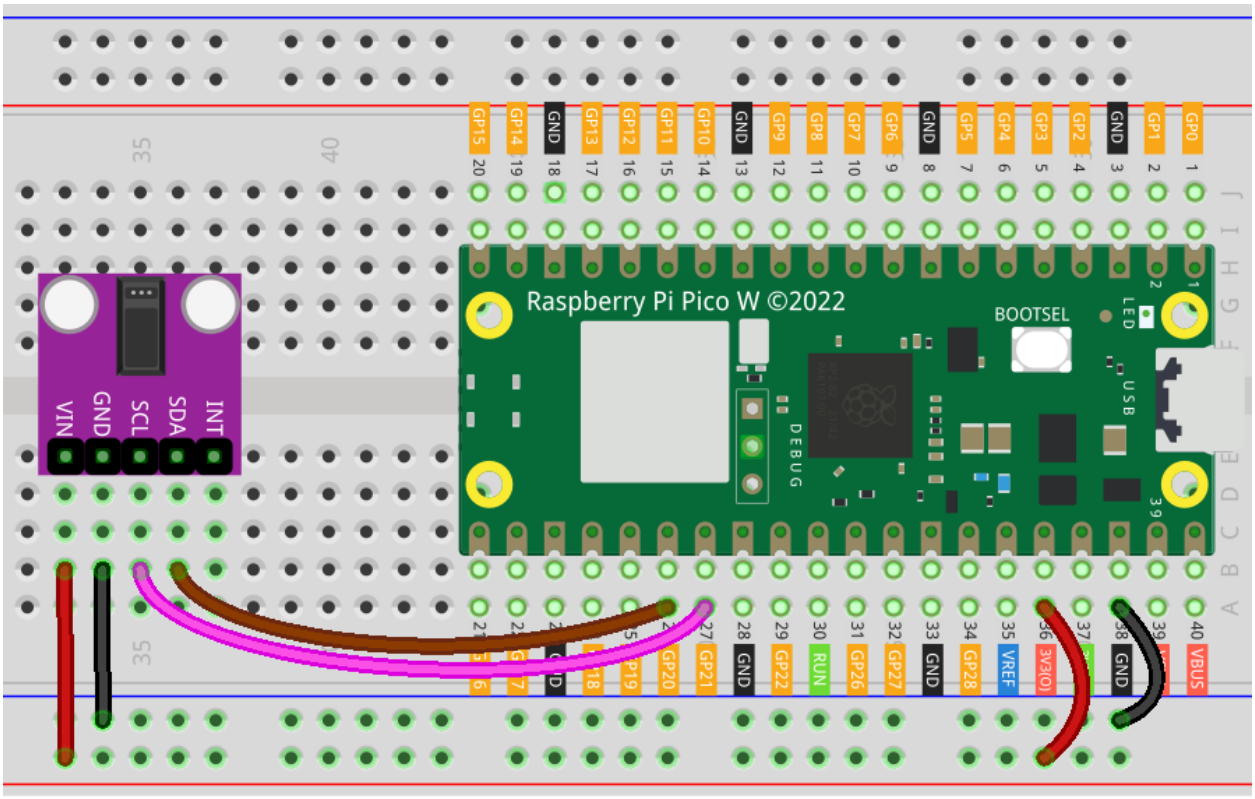
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Pulse Oximeter and Heart Rate Sensor Module (MAX30102)	
Breadboard	

Wiring



Code

Note:

- Open the `14_max30102_module.py` file under the path of `universal-maker-sensor-kit-main/pico/Lesson_14_MAX30102_Module` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it. For detailed tutorials, please refer to *Open and Run Code Directly*.
- Here you need to use the `max30102` folder, please check if it has been uploaded to Pico W, for a detailed tutorial refer to *Upload the Libraries to Pico*.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.

```
from machine import SoftI2C, Pin
from time import ticks_diff, ticks_us, sleep

from max30102 import MAX30102, MAX30105_PULSE_AMP_MEDIUM


def main():
    # I2C software instance
    i2c = SoftI2C(sda=Pin(20),  # Here, use your I2C SDA pin
                  scl=Pin(21),  # Here, use your I2C SCL pin
                  freq=400000)  # Fast: 400kHz, slow: 100kHz
```

(continues on next page)

(continued from previous page)

```
# Sensor instance
sensor = MAX30102(i2c=i2c)  # An I2C instance is required

# Scan I2C bus to ensure that the sensor is connected
if sensor.i2c_address not in i2c.scan():
    print("Sensor not found.")
    return
elif not (sensor.check_part_id()):
    # Check that the targeted sensor is compatible
    print("I2C device ID not corresponding to MAX30102 or MAX30105.")
    return
else:
    print("Sensor connected and recognized.")

# It's possible to set up the sensor at once with the setup_sensor() method.
# If no parameters are supplied, the default config is loaded:
# Led mode: 2 (RED + IR)
# ADC range: 16384
# Sample rate: 400 Hz
# Led power: maximum (50.0mA - Presence detection of ~12 inch)
# Averaged samples: 8
# pulse width: 411
print("Setting up sensor with default configuration.", '\n')
sensor.setup_sensor()

# It is also possible to tune the configuration parameters one by one.
# Set the sample rate to 400: 400 samples/s are collected by the sensor
sensor.set_sample_rate(400)
# Set the number of samples to be averaged per each reading
sensor.set_fifo_average(8)
# Set LED brightness to a medium value
sensor.set_active_leds_amplitude(MAX30105_PULSE_AMP_MEDIUM)

sleep(1)

# The readTemperature() method allows to extract the die temperature in °C
print("Reading temperature in °C.", '\n')
print(sensor.read_temperature())

print("Starting data acquisition from RED & IR registers...", '\n')
sleep(1)

while True:
    # The check() method has to be continuously polled, to check if
    # there are new readings into the sensor's FIFO queue. When new
    # readings are available, this function will put them into the storage.
    sensor.check()

    # Check if the storage contains available samples
    if sensor.available():
        # Access the storage FIFO and gather the readings (integers)
```

(continues on next page)

(continued from previous page)

```
            red_reading = sensor.pop_red_from_storage()
            ir_reading = sensor.pop_ir_from_storage()

            # Print the acquired data (so that it can be plotted with a Serial Plotter)
            print("red_reading",red_reading, "ir_reading", ir_reading)

if __name__ == '__main__':
    main()
```

Code Analysis

1. Setting up I2C Interface

   SoftI2C is initialized with SDA and SCL pins, and a frequency of 400kHz is set for the communication.

   ```
   from machine import SoftI2C, Pin
   i2c = SoftI2C(sda=Pin(20), scl=Pin(21), freq=400000)
   ```

2. Initializing the Sensor

   The MAX30102 sensor is initialized using the I2C interface. A scan of the I2C bus is performed to ensure the sensor is connected and recognized.

   For more information about the max30102 library, please visit .

   ```
   from max30102 import MAX30102
   sensor = MAX30102(i2c=i2c)
   ```

3. Sensor Configuration

   The sensor is configured with default settings for LED mode, ADC range, sample rate, LED power, averaged samples, and pulse width. Additional configurations like sample rate, FIFO average, and LED amplitude are set.

   ```
   sensor.setup_sensor()
   sensor.set_sample_rate(400)
   sensor.set_fifo_average(8)
   sensor.set_active_leds_amplitude(MAX30105_PULSE_AMP_MEDIUM)
   ```

4. Reading Temperature

   The temperature of the sensor is read and printed.

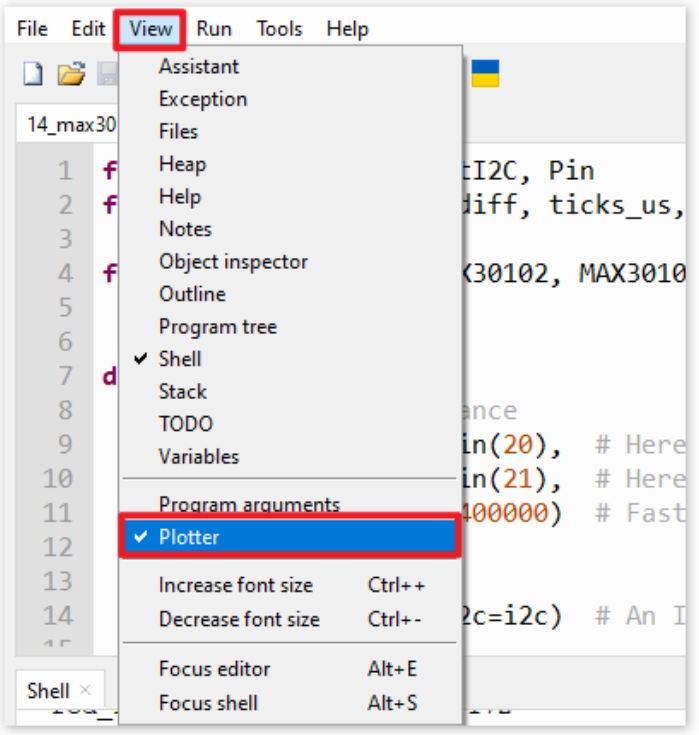
   ```
   print(sensor.read_temperature())
   ```

5. Data Acquisition

   A loop is set up to continuously acquire data from the sensor. The check() method is polled to see if new readings are available. Red and IR readings are retrieved from the sensor's storage and printed.

   ```
   while True:
       sensor.check()
       if sensor.available():
           red_reading = sensor.pop_red_from_storage()
           ir_reading = sensor.pop_ir_from_storage()
           print("red_reading",red_reading, "ir_reading", ir_reading)
   ```

Open Plotter in Thonny to observe the heartbeat data.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.16 Lesson 15: Raindrop Detection Module

In this lesson, you'll learn how to use the Raspberry Pi Pico W to detect raindrops using a raindrop sensor connected to pin 16. The script continuously monitors for any indication of raindrops and prints "Raindrop detected!" when one is detected; otherwise, it displays "Monitoring..." as it waits for raindrops. This session offers hands-on experience in handling digital inputs with the Raspberry Pi Pico W and understanding environmental sensing in MicroPython, making it ideal for beginners in electronics and programming.

Required Components

In this project, we need the following components.

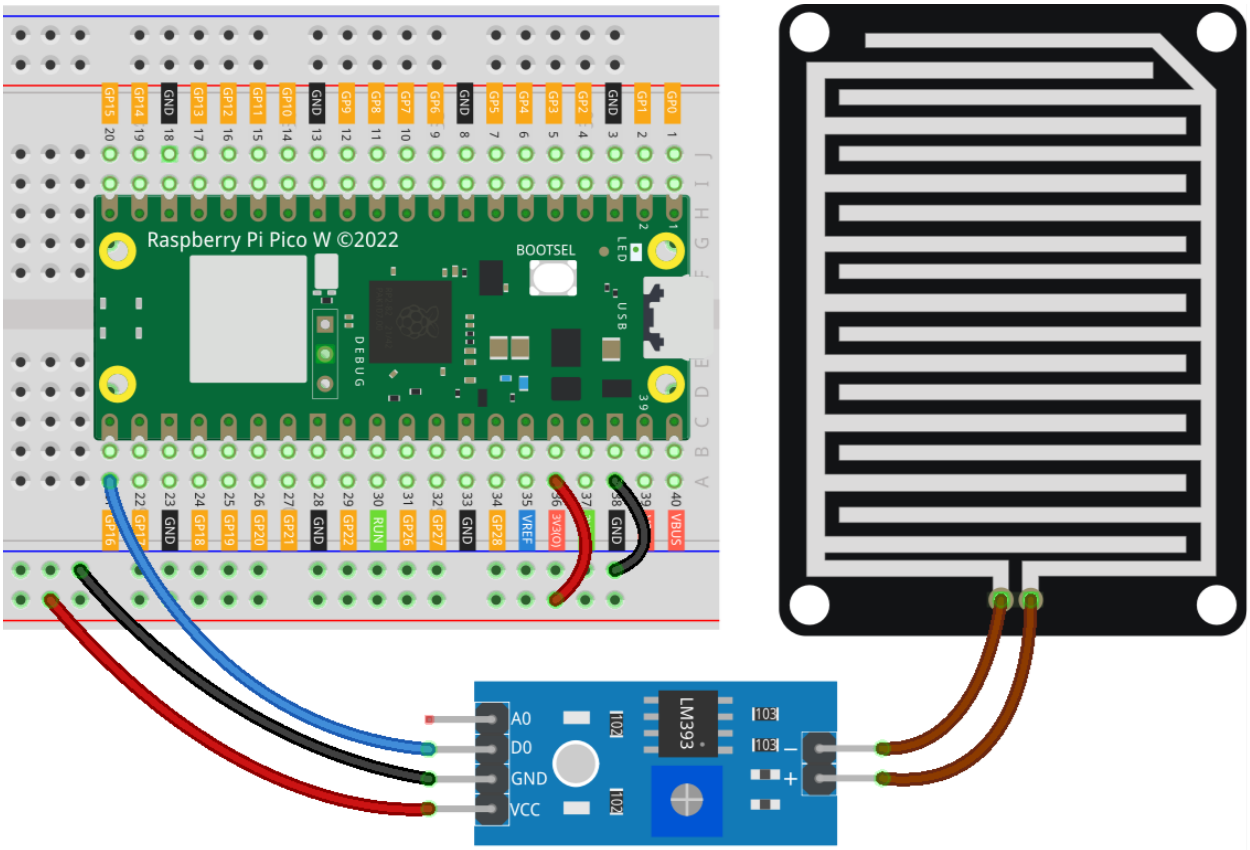
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Raindrop Detection Module	
Breadboard	

Wiring



Code

```
from machine import Pin
import time

# Initialize raindrop sensor connected to pin 16 as input
raindrop_sensor = Pin(16, Pin.IN)

while True:
    # Check the Raindrop sensor value
    if raindrop_sensor.value() == 0:
        print("Raindrop detected!")  # Raindrop detected
    else:
        print("Monitoring...")  # No raindrop detected

    time.sleep(0.1)  # Short delay of 0.1 seconds to reduce CPU usage
```

Code Analysis

1. Initializing the Raindrop Sensor:

   The raindrop sensor is initialized using the `Pin` class from the `machine` module, set to pin 16 in input mode. This allows the Raspberry Pi Pico W to read the sensor output.

   ```
   from machine import Pin
   raindrop_sensor = Pin(16, Pin.IN)
   ```

2. Continuous Monitoring Loop:

   A continuous while loop is used to monitor the sensor. Inside the loop, the sensor value is checked. If the value is 0, it indicates raindrops are detected and prints "Raindrop detected!" Otherwise, it prints "Monitoring..." to indicate the absence of raindrops.

   ```
   while True:
       if raindrop_sensor.value() == 0:
           print("Raindrop detected!")
       else:
           print("Monitoring...")
   ```

3. Introducing a Delay:

   To reduce CPU usage, a delay of 0.1 seconds is introduced in each iteration of the loop using `time.sleep(0.1)`. This prevents the loop from executing too rapidly.

   ```
   time.sleep(0.1)
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.

- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.17 Lesson 16: Real Time Clock Module (DS1302)

In this lesson, you'll learn how to use the Raspberry Pi Pico W to interact with a DS1302 Real-Time Clock module. We'll start by setting up the DS1302 and connecting it to the Pico W using specific GPIO pins. You'll also learn how to retrieve and set the current date and time on the DS1302. Additionally, we'll explore continuously displaying the current datetime on your console, updating every half second.

Required Components

In this project, we need the following components.

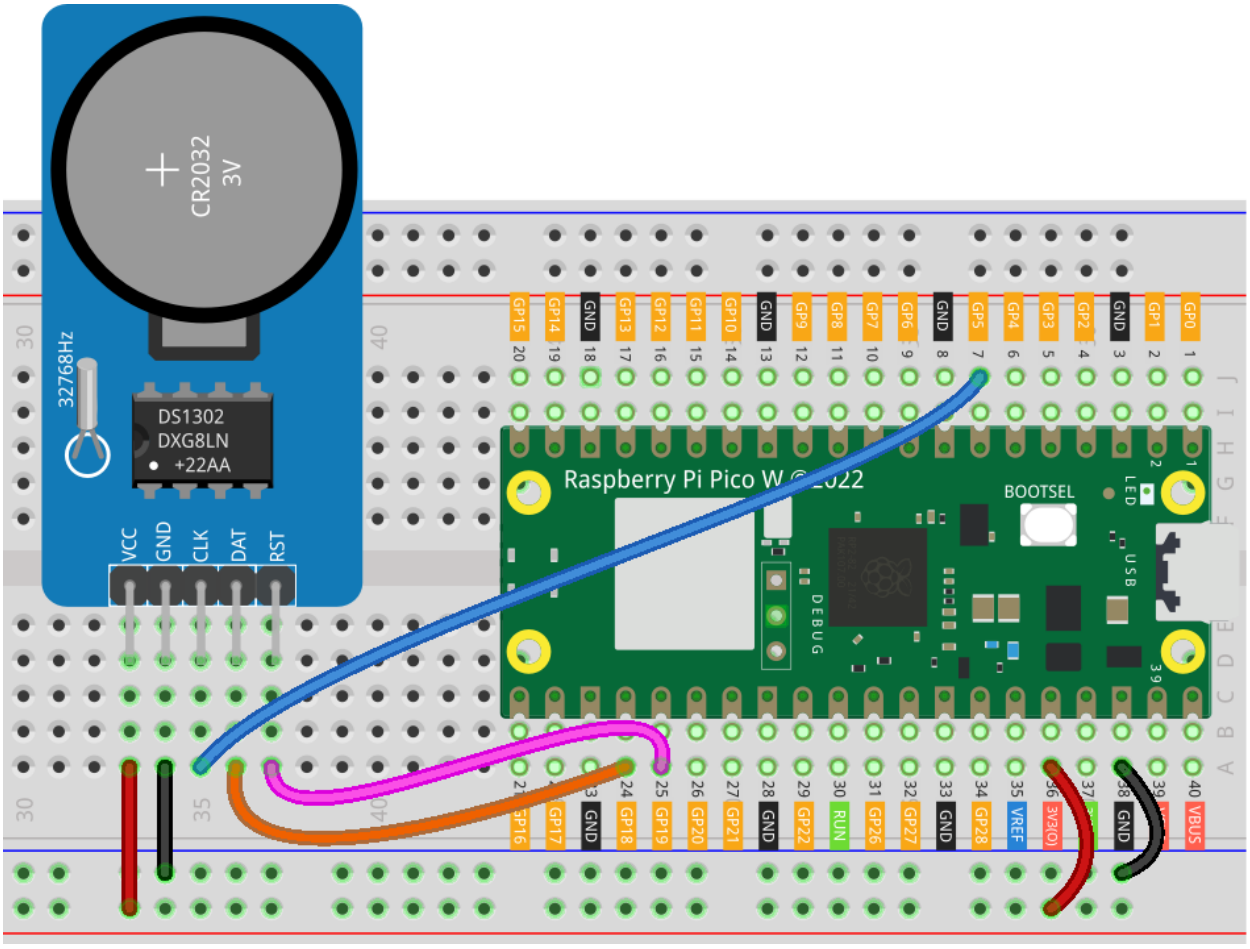
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Real Time Clock Module (DS1302)	
Breadboard	

Wiring



Code

Note:

- Open the `16_ds1302_module.py` file under the path of `universal-maker-sensor-kit-main/pico/Lesson_16_DS1302_Module` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it. For detailed tutorials, please refer to *Open and Run Code Directly*.
- Here you need to use the `ds1302.py`, please check if it has been uploaded to Pico W, for a detailed tutorial refer to *Upload the Libraries to Pico*.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.

```
from machine import Pin
import ds1302
import time

# Initialize DS1302 RTC with specific GPIO pins
ds = ds1302.DS1302(Pin(5), Pin(18), Pin(19))  # (clk, dio, cs)
```

(continues on next page)

(continued from previous page)

```
# Get current datetime from DS1302
ds.date_time()

# Set DS1302 datetime to 2024-01-01 Monday 00:00:00
ds.date_time([2024, 1, 1, 1, 0, 0, 0])  # (year,month,day,weekday,hour,minute,second)

# Set seconds to 10
ds.second(10)

# Continuously display current datetime every half second
while True:
    print(ds.date_time())
    time.sleep(0.5)
```

Code Analysis

1. **Import Libraries**

   This section imports necessary libraries: `machine` for GPIO control, `ds1302` for the RTC module, and `time` for implementing delays.

   For more detail about the `ds1302` library, please refer to `ds1302.py`.

   ```
   from machine import Pin
   import ds1302
   import time
   ```

2. **Initialize the DS1302 RTC**

   This code initializes the DS1302 module by defining which GPIO pins of the Raspberry Pi Pico W are connected to the clock (clk), data input/output (dio), and chip select (cs) pins of the DS1302.

   ```
   ds = ds1302.DS1302(Pin(5), Pin(18), Pin(19))  # (clk, dio, cs)
   ```

3. **Get Current DateTime**

   Retrieves the current date and time from the DS1302. The `date_time()` method returns a list containing year, month, day, weekday, hour, minute, and second.

   ```
   ds.date_time()
   ```

4. **Set DS1302 DateTime**

   Sets the DS1302's date and time to January 1, 2024, at 00:00:00. The day of the week (Monday) is represented by 1.

   ```
   ds.date_time([2024, 1, 1, 1, 0, 0, 0])  # (year,month,day,weekday,hour,minute,
   ↪second)
   ```

5. **Set Seconds**

   Sets the seconds value of the DS1302's time to 10.

```
ds.second(10)
```

6. **Display Current DateTime Continuously**

   This loop continuously displays the current date and time every half second. The `time.sleep(0.5)` function creates a half-second delay between each iteration.

```
while True:
    print(ds.date_time())
    time.sleep(0.5)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.18 Lesson 17: Rotary Encoder Module

In this lesson, you'll learn how to use the Raspberry Pi Pico W to control a rotary encoder. The rotary encoder is an advanced sensor that translates knob rotation into an output signal, indicating both the amount and direction of rotation. This project offers hands-on experience with digital input devices, enhancing your ability to work with more complex sensors. You'll configure the rotary encoder using specific GPIO pins, read its output to determine rotation direction and amount, and master using a button to trigger events.

Required Components

In this project, we need the following components.

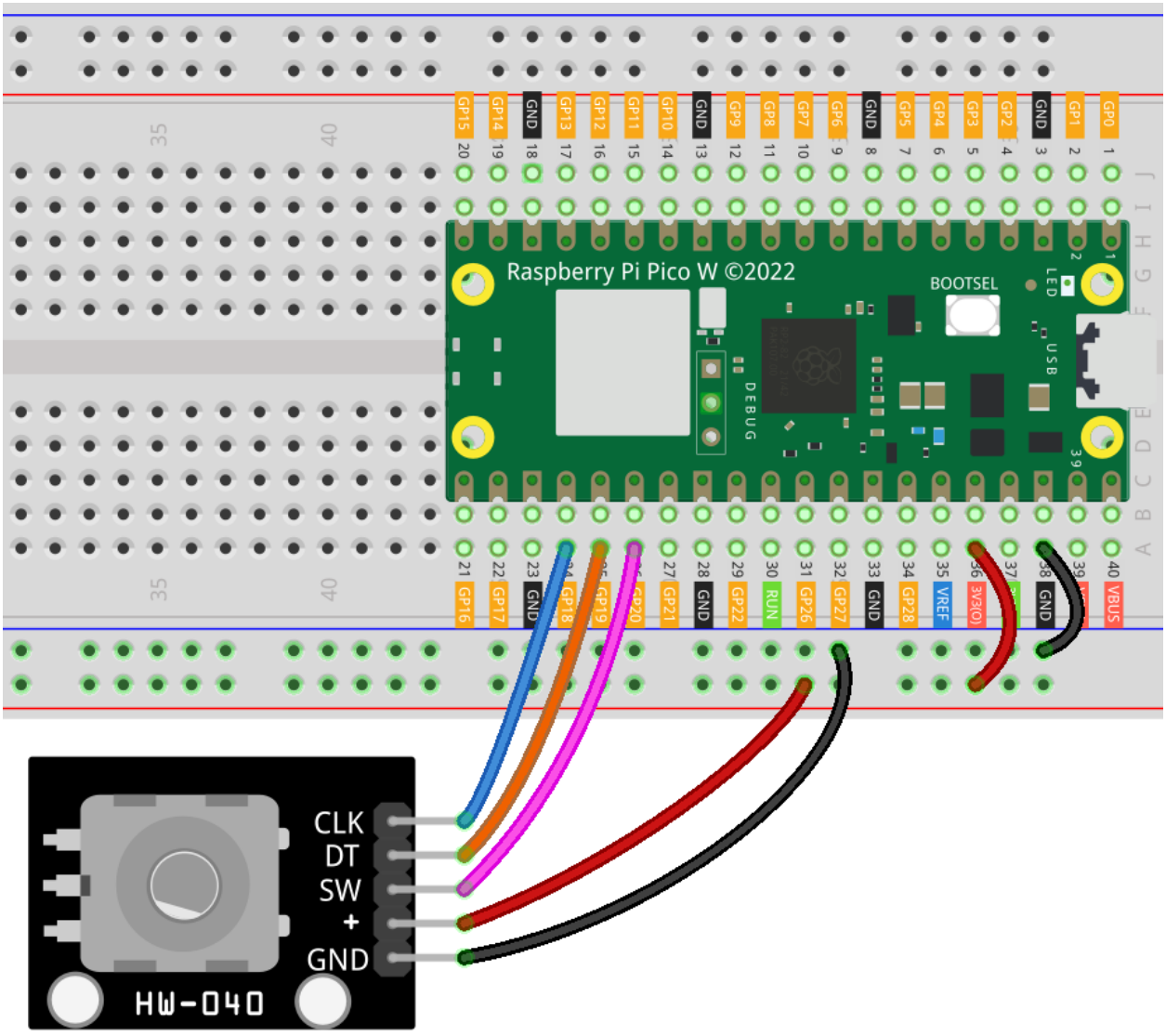
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Rotary Encoder Module	-
Breadboard	

Wiring



Code

Note:

- Open the `17_rotary_encoder_module.py` file under the path of `universal-maker-sensor-kit-main/pico/Lesson_17_Rotary_Encoder_Module` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it. For detailed tutorials, please refer to *Open and Run Code Directly*.
- Here you need to use the `rotary_irq_rp2.py`, please check if it has been uploaded to Pico W, for a detailed tutorial refer to *Upload the Libraries to Pico*.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.

```
from rotary_irq_rp2 import RotaryIRQ
import time
from machine import Pin

# Set GPIO 20 as an input pin for reading the button(sw)'s state
button_pin = Pin(20, Pin.IN, Pin.PULL_UP)

# Initialize the rotary encoder with specific GPIO pins and settings
rotary_encoder = RotaryIRQ(
    pin_num_clk=18,
    pin_num_dt=19,
    min_val=0,
    max_val=14,
    reverse=False,
    range_mode=RotaryIRQ.RANGE_WRAP,
)

# Store the initial value of the rotary encoder and button state
last_rotary_value = rotary_encoder.value()
last_button_state = button_pin.value()

# Main loop
while True:
    # Read the current value of the rotary encoder and button state
    current_rotary_value = rotary_encoder.value()
    current_button_state = button_pin.value()

    # Check if the rotary encoder's value has changed
    if last_rotary_value != current_rotary_value:
        last_rotary_value = current_rotary_value
        print("result =", current_rotary_value)

    # Check if the button's state changed from not pressed to pressed
    if last_button_state and not current_button_state:
        print("Button pressed!")

    # Update the previous state of the button for the next loop iteration
    last_button_state = current_button_state

    # Short delay to prevent debouncing issues
```

(continues on next page)

(continued from previous page)

```
    time.sleep_ms(50)
```

Code Analysis

1. **Importing Libraries**

   First, the necessary libraries are imported. `rotary_irq_rp2` is for the rotary encoder, `time` for delays, and `machine` for hardware control.

   For more information about the `rotary_irq_rp2` library, please visit .

   ```
   from rotary_irq_rp2 import RotaryIRQ
   import time
   from machine import Pin
   ```

2. **Setting up the Button Pin**

   The GPIO pin connected to the SW pin is configured as an input with a pull-up resistor. This ensures a stable HIGH signal when the button is not pressed.

   ```
   button_pin = Pin(20, Pin.IN, Pin.PULL_UP)
   ```

3. **Initializing the Rotary Encoder**

   The encoder is set up with specified GPIO pins for CLK and DT. `min_val` and `max_val` define the range of values, and `range_mode` sets how the value behaves at limits (wraps around in this case).

   ```
   rotary_encoder = RotaryIRQ(
       pin_num_clk=18,
       pin_num_dt=19,
       min_val=0,
       max_val=14,
       reverse=False,
       range_mode=RotaryIRQ.RANGE_WRAP,
   )
   ```

4. **Storing Initial Values**

   The initial values of the rotary encoder and the button are stored to detect changes in their states later.

   ```
   last_rotary_value = rotary_encoder.value()
   last_button_state = button_pin.value()
   ```

5. **Main Loop**

   The loop continuously checks for changes in the rotary encoder value and button state. If the rotary value changes, it prints the new value. If the button state changes from unpressed to pressed, it prints “Button pressed!”.

   ```
   while True:
       current_rotary_value = rotary_encoder.value()
       current_button_state = button_pin.value()

       if last_rotary_value != current_rotary_value:
           last_rotary_value = current_rotary_value
           print("result =", current_rotary_value)
   ```

(continues on next page)

(continued from previous page)

```
    if last_button_state and not current_button_state:
        print("Button pressed!")

    last_button_state = current_button_state
    time.sleep_ms(50)
```

The `time.sleep_ms(50)` at the end of the loop is to prevent debouncing issues, which can cause erratic readings.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.19 Lesson 18: Temperature Sensor Module (DS18B20)

In this lesson, you'll learn how to integrate and read temperature data from DS18B20 sensors using the Raspberry Pi Pico W. You'll begin by setting up a OneWire bus on the GPIO pin and scanning for DS18X20 devices. The main focus of the lesson is continuously reading and displaying temperature measurements from these sensors.

Required Components

In this project, we need the following components.

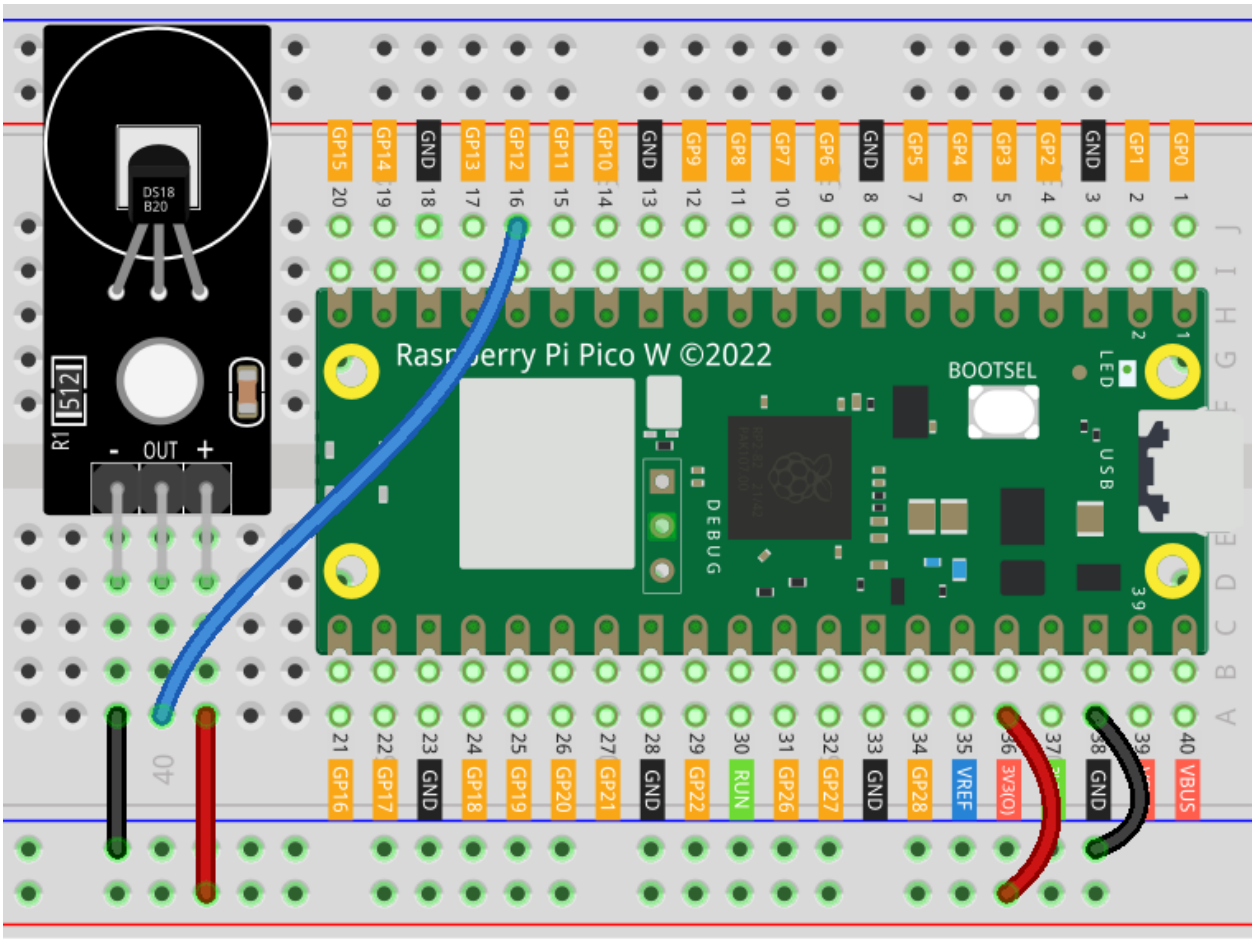
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Temperature Sensor Module (DS18B20)	-
Breadboard	

Wiring



Code

Note:

- Open the `18_ds18b20_module.py` file under the path of `universal-maker-sensor-kit-main/pico/Lesson_18_DS18B20_Module` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it. For detailed tutorials, please refer to *Open and Run Code Directly*.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.

```
from machine import Pin
import onewire
import time, ds18x20

# Initialize the OneWire bus on GPIO pin 12
ow = onewire.OneWire(Pin(12))

# Create a DS18X20 instance using the OneWire bus
ds = ds18x20.DS18X20(ow)
```

(continues on next page)

(continued from previous page)

```
# Scan for DS18X20 devices on the bus and print their addresses
roms = ds.scan()
print('found devices:', roms)

# Continuously read and print temperature data from the sensors
while True:
    # Start the temperature conversion process
    ds.convert_temp()
    # Wait for the conversion to complete (750 ms for DS18X20)
    time.sleep_ms(750)

    # Read and print the temperature from each sensor found on the bus
    for rom in roms:
        print(ds.read_temp(rom))

    # Wait for a short period before the next reading (1000 ms)
    time.sleep_ms(1000)
```

Code Analysis

1. Importing Libraries

   The code begins by importing necessary libraries. `machine` is used for controlling GPIO pins, `onewire` for the OneWire communication protocol, `ds18x20` for the specific temperature sensor, and `time` for delays.

   Regarding OneWire in MicroPython, you can refer to .

   ```
   from machine import Pin
   import onewire
   import time, ds18x20
   ```

2. Initializing OneWire Bus

   A OneWire bus is initialized on GPIO pin 12. This sets up the communication between the Raspberry Pi Pico W and the DS18B20 sensor.

   ```
   ow = onewire.OneWire(Pin(12))
   ```

3. Creating DS18X20 Instance

   A DS18X20 instance is created using the OneWire bus. This instance is used to interact with the temperature sensor.

   ```
   ds = ds18x20.DS18X20(ow)
   ```

4. Scanning for Devices

   The code scans for DS18X20 devices on the OneWire bus and prints their addresses. This is important for identifying the connected sensors.

   ```
   roms = ds.scan()
   print('found devices:', roms)
   ```

5. Reading Temperature Data

- The main loop of the program continuously reads temperature data from the sensor.
- It starts the temperature conversion process and waits for it to complete, which takes about 750 milliseconds.
- It then reads and prints the temperature from each sensor found on the bus.
- The loop pauses for 1000 milliseconds before repeating.

```
while True:
    ds.convert_temp()
    time.sleep_ms(750)
    for rom in roms:
        print(ds.read_temp(rom))
    time.sleep_ms(1000)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.20 Lesson 19: Temperature and Humidity Sensor Module (DHT11)

In this lesson, you'll learn how to use the Raspberry Pi Pico W to connect with a DHT11 temperature and humidity sensor. You'll explore accurate measurement of environmental conditions by recording temperature and humidity data. This tutorial offers practical guidance on using digital sensors with the Raspberry Pi Pico W, programming with MicroPython, and managing real-time data processing.

Required Components

In this project, we need the following components.

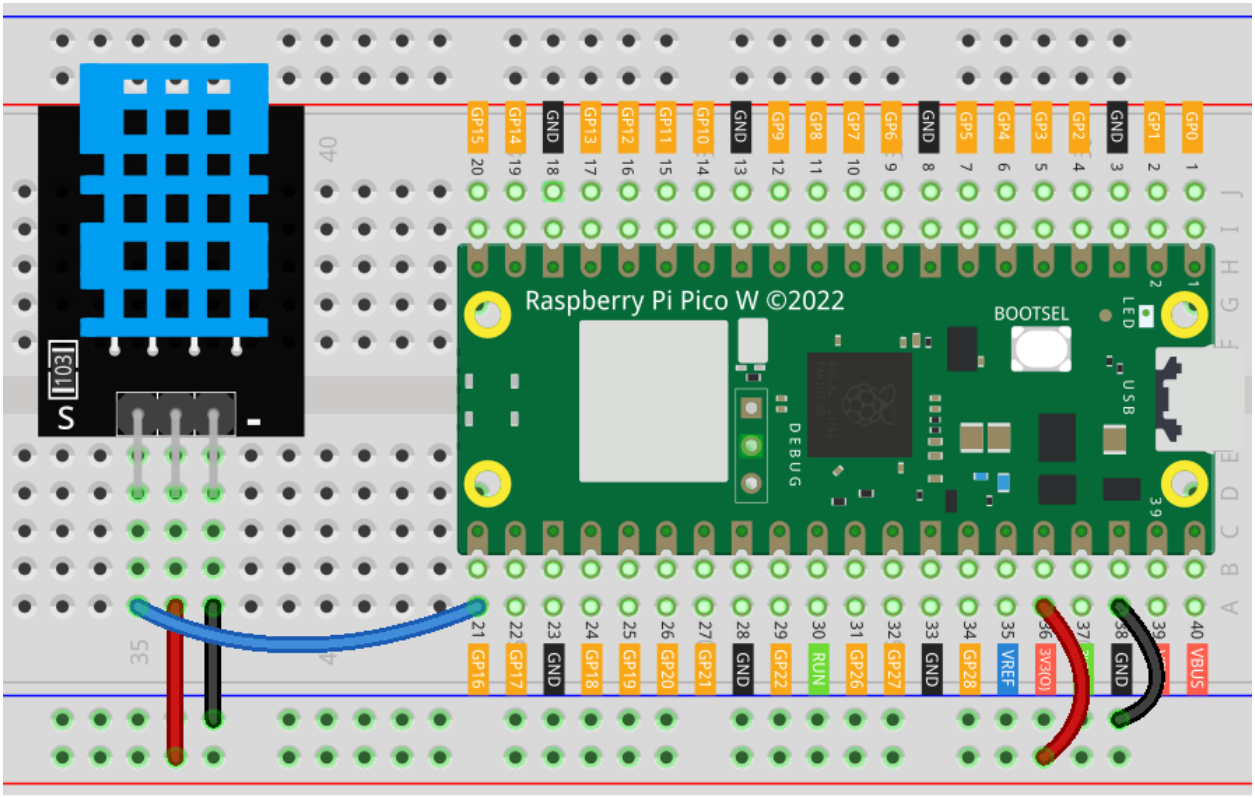
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Temperature and Humidity Sensor Module (DHT11)	
Breadboard	

Wiring



Code

```
import dht
import machine
import time

# Initialize DHT11 sensor on GPIO 16
d = dht.DHT11(machine.Pin(16))

# Continuously read and print temperature and humidity
while True:
    d.measure()
    print("Temperature:" ,d.temperature())  # Print temperature
    print("Humidity:" ,d.humidity())  # Print humidity
    time.sleep_ms(1000)  # Read every second
```

Code Analysis

1. Importing Libraries:

   The code begins by importing necessary libraries. `dht` is for the DHT11 sensor, `machine` is for interacting with the hardware, and `time` is for adding delays in the loop.

   ```
   import dht
   import machine
   import time
   ```

2. Initializing the DHT11 Sensor:

   The DHT11 sensor is initialized by specifying its connected GPIO pin. Here, it's connected to GPIO 16 on the Raspberry Pi Pico W. This is done using the `machine.Pin` function.

   ```
   d = dht.DHT11(machine.Pin(16))
   ```

3. Reading and Printing Data in a Loop:

   The `while True` loop enables the program to continuously read temperature and humidity data. Inside the loop, `d.measure()` is called to take a new measurement. `d.temperature()` and `d.humidity()` are used to retrieve the temperature and humidity data, respectively. These values are then printed. The loop pauses for one second (`1000` milliseconds) using `time.sleep_ms(1000)`, ensuring the data is read and printed every second.

   ```
   while True:
       d.measure()
       print("Temperature:" ,d.temperature())  # Print temperature
       print("Humidity:" ,d.humidity())  # Print humidity
       time.sleep_ms(1000)  # Read every second
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.21 Lesson 20: Temperature, Humidity & Pressure Sensor (BMP280)

In this lesson, you'll learn how to connect the BMP280 temperature, humidity, and pressure sensor to the Raspberry Pi Pico W using MicroPython. You'll get practical experience in setting up I2C communication, configuring the BMP280 sensor for weather monitoring, and obtaining temperature and pressure data. By the end of this tutorial, you'll be able to view real-time environmental data on your console.

Required Components

In this project, we need the following components.

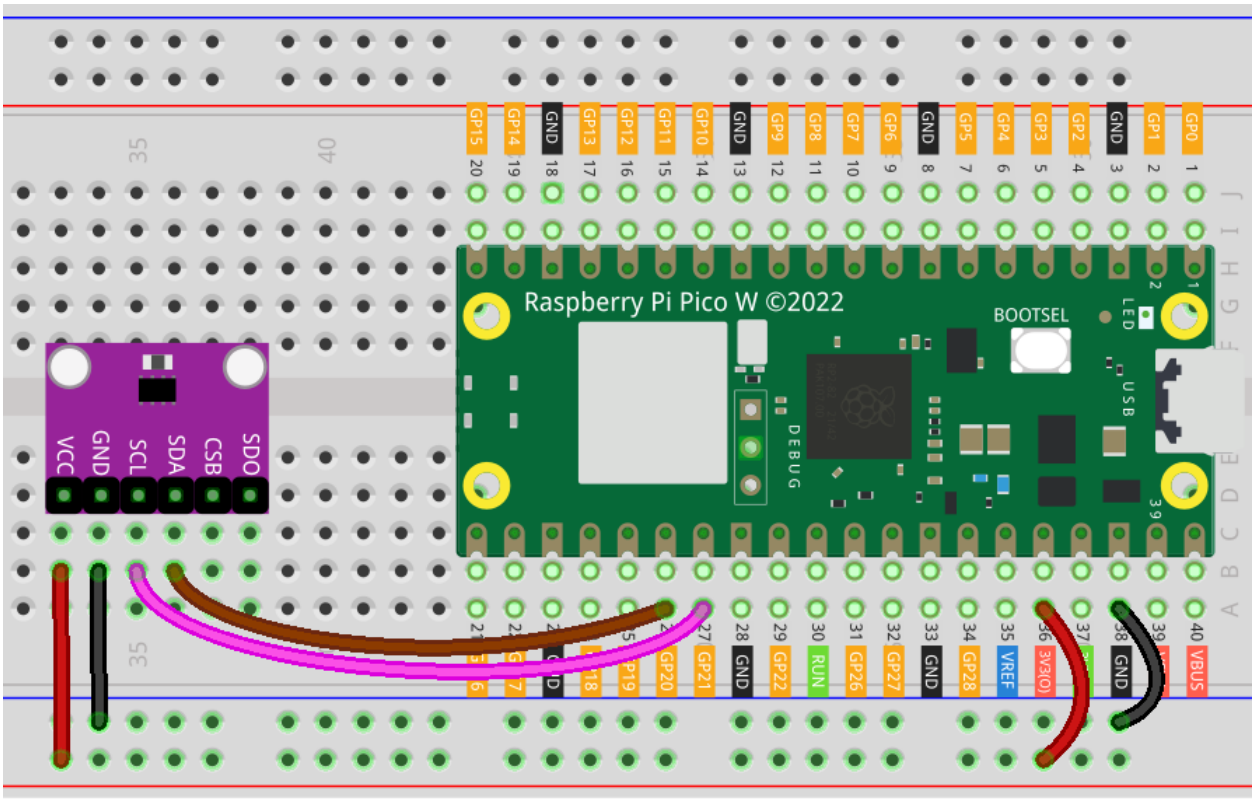
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Temperature, Humidity & Pressure Sensor (BMP280)	
Breadboard	

Wiring



Code

Note:

- Open the `20_bmp280_module.py` file under the path of `universal-maker-sensor-kit-main/pico/Lesson_20_BMP280_Module` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it. For detailed tutorials, please refer to *Open and Run Code Directly*.
- Here you need to use the `bmp280.py`, please check if it has been uploaded to Pico W, for a detailed tutorial refer to *Upload the Libraries to Pico*.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.

```
from machine import I2C, Pin
import bmp280
import time

# Initialize I2C communication
i2c = I2C(0, sda=Pin(20), scl=Pin(21), freq=100000)

# Configure BMP280 sensor
bmp = bmp280.BMP280(i2c)
bmp.oversample(bmp280.BMP280_OS_HIGH)
```

(continues on next page)

(continued from previous page)

```
while True:
    # Set sensor to weather monitoring mode
    bmp.use_case(bmp280.BMP280_CASE_WEATHER)

    # Print temperature and pressure data
    print("tempC: {}".format(bmp.temperature))
    print("pressure: {}Pa".format(bmp.pressure))

    # Read data every second
    time.sleep_ms(1000)
```

Code Analysis

1. **Importing Libraries and Initializing I2C Communication**:

   This code segment imports necessary libraries and initializes I2C communication. The `machine` module is used to interact with the hardware components like I2C and pins. The `bmp280` library is imported to interact with the BMP280 sensor.

   For more information about the `bmp280` library, please visit .

   ```
   from machine import I2C, Pin
   import bmp280
   import time

   # Initialize I2C communication
   i2c = I2C(0, sda=Pin(20), scl=Pin(21), freq=100000)
   ```

2. **Configuring the BMP280 Sensor**:

   Here, the BMP280 sensor is configured. An object `bmp` is created to interact with the sensor. The oversampling setting is adjusted for higher accuracy.

   ```
   # Configure BMP280 sensor
   bmp = bmp280.BMP280(i2c)
   bmp.oversample(bmp280.BMP280_OS_HIGH)
   ```

3. **Reading and Displaying Sensor Data in a Loop**:

   The sensor is continuously read in an infinite loop. Each iteration sets the sensor to weather monitoring mode, reads the temperature and pressure, and prints them. The `time.sleep_ms(1000)` ensures the loop runs once every second.

   ```
   while True:
       # Set sensor to weather monitoring mode
       bmp.use_case(bmp280.BMP280_CASE_WEATHER)

       # Print temperature and pressure data
       print("tempC: {}".format(bmp.temperature))
       print("pressure: {}Pa".format(bmp.pressure))

       # Read data every second
       time.sleep_ms(1000)
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.22 Lesson 21: Time of Flight Micro-LIDAR Distance Sensor (VL53L0X)

In this lesson, you'll learn how to use the Raspberry Pi Pico W for measuring distances with the VL53L0X Time of Flight Micro-LIDAR Distance Sensor. We'll walk you through setting up I2C communication between the Raspberry Pi Pico W and the sensor, and then we'll explore configuring the sensor's settings for optimal performance. You will also learn how to adjust the measurement timing budget and VCSEL pulse periods to improve accuracy and range.

Required Components

In this project, we need the following components.

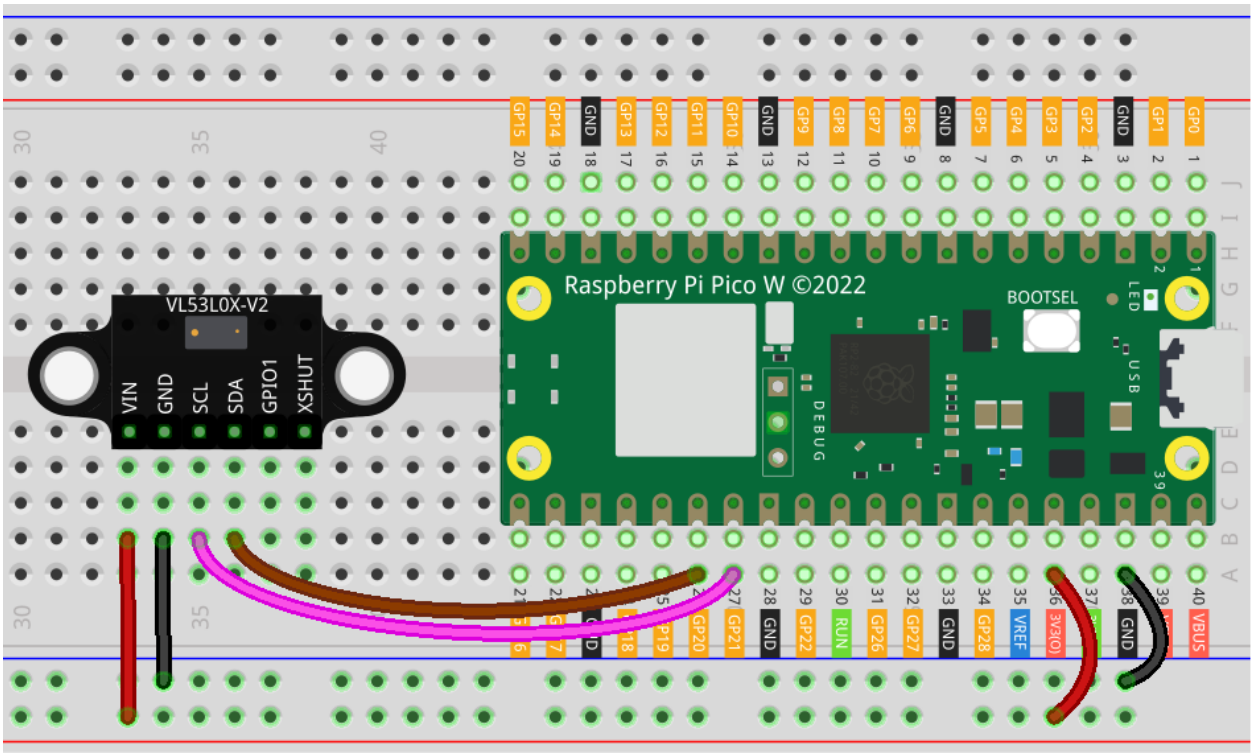
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Time of Flight Micro-LIDAR Distance Sensor (VL53L0X)	
Breadboard	

Wiring



Code

Note:

- Open the `21_vl53l0x_module.py` file under the path of `universal-maker-sensor-kit-main/pico/Lesson_21_VL53L0X_Module` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it. For detailed tutorials, please refer to *Open and Run Code Directly*.
- Here you need to use the `vl53l0x.py`, please check if it has been uploaded to Pico W, for a detailed tutorial refer to *Upload the Libraries to Pico*.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.

```
import time
from machine import Pin, I2C
from vl53l0x import VL53L0X

print("setting up i2c")
id = 0
sda = Pin(20)
scl = Pin(21)

i2c = I2C(id=id, sda=sda, scl=scl)

print(i2c.scan())
```

(continues on next page)

(continued from previous page)

```
# print("creating vl53lox object")
# Create a VL53L0X object
tof = VL53L0X(i2c)

# Pre: 12 to 18 (initialized to 14 by default)
# Final: 8 to 14 (initialized to 10 by default)

# the measuting_timing_budget is a value in ms, the longer the budget, the more accurate␣
↪the reading.
budget = tof.measurement_timing_budget_us
print("Budget was:", budget)
tof.set_measurement_timing_budget(40000)

# Sets the VCSEL (vertical cavity surface emitting laser) pulse period for the
# given period type (VL53L0X::VcselPeriodPreRange or VL53L0X::VcselPeriodFinalRange)
# to the given value (in PCLKs). Longer periods increase the potential range of the␣
↪sensor.
# Valid values are (even numbers only):

# tof.set_Vcsel_pulse_period(tof.vcsel_period_type[0], 18)
tof.set_Vcsel_pulse_period(tof.vcsel_period_type[0], 12)

# tof.set_Vcsel_pulse_period(tof.vcsel_period_type[1], 14)
tof.set_Vcsel_pulse_period(tof.vcsel_period_type[1], 8)

while True:
    # Start ranging
    print(tof.ping() - 50, "mm")

    time.sleep_ms(100)  # Short delay of 0.1 seconds to reduce CPU usage
```

Code Analysis

1. **Setting up the I2C Interface**:

   The code begins by importing necessary modules and initializing the I2C communication. The `machine` module is used to set up I2C with the correct pins of the Raspberry Pi Pico W.

   For more information about the `vl53l0x` library, please visit .

   ```
   import time
   from machine import Pin, I2C
   from vl53l0x import VL53L0X

   print("setting up i2c")
   id = 0
   sda = Pin(20)
   scl = Pin(21)
   i2c = I2C(id=id, sda=sda, scl=scl)
   print(i2c.scan())
   ```

2. **Creating VL53L0X Object**:

An object of `VL53L0X` class is created. This object will be used to interact with the VL53L0X sensor.

```
tof = VL53L0X(i2c)
```

3. **Configuring Measurement Timing Budget**:

   The measurement timing budget is set up. This determines how long the sensor takes to perform a measurement. A longer timing budget allows for more accurate readings.

```
budget = tof.measurement_timing_budget_us
print("Budget was:", budget)
tof.set_measurement_timing_budget(40000)
```

4. **Setting VCSEL Pulse Periods**:

   Here, the pulse periods for the VCSEL (Vertical Cavity Surface Emitting Laser) are set. This affects the range and accuracy of the sensor.

```
tof.set_Vcsel_pulse_period(tof.vcsel_period_type[0], 12)
tof.set_Vcsel_pulse_period(tof.vcsel_period_type[1], 8)
```

5. **Continuous Measurement Loop**:

   The sensor continuously measures the distance and prints it. The `ping()` method of `VL53L0X` class is used to get the distance in millimeters. A small delay is added to reduce CPU usage.

```
while True:
    print(tof.ping() - 50, "mm")
    time.sleep_ms(100)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.23 Lesson 22: Touch Sensor Module

In this lesson, you'll learn how to connect a touch sensor to the Raspberry Pi Pico W in order to control an onboard LED. By using straightforward Python code, you'll configure the touch sensor as an input device. When the sensor detects a touch, it will send a signal to turn on the LED, providing a visual indication that a touch has been detected. Conversely, when there's no touch, the LED stays off.

Required Components

In this project, we need the following components.

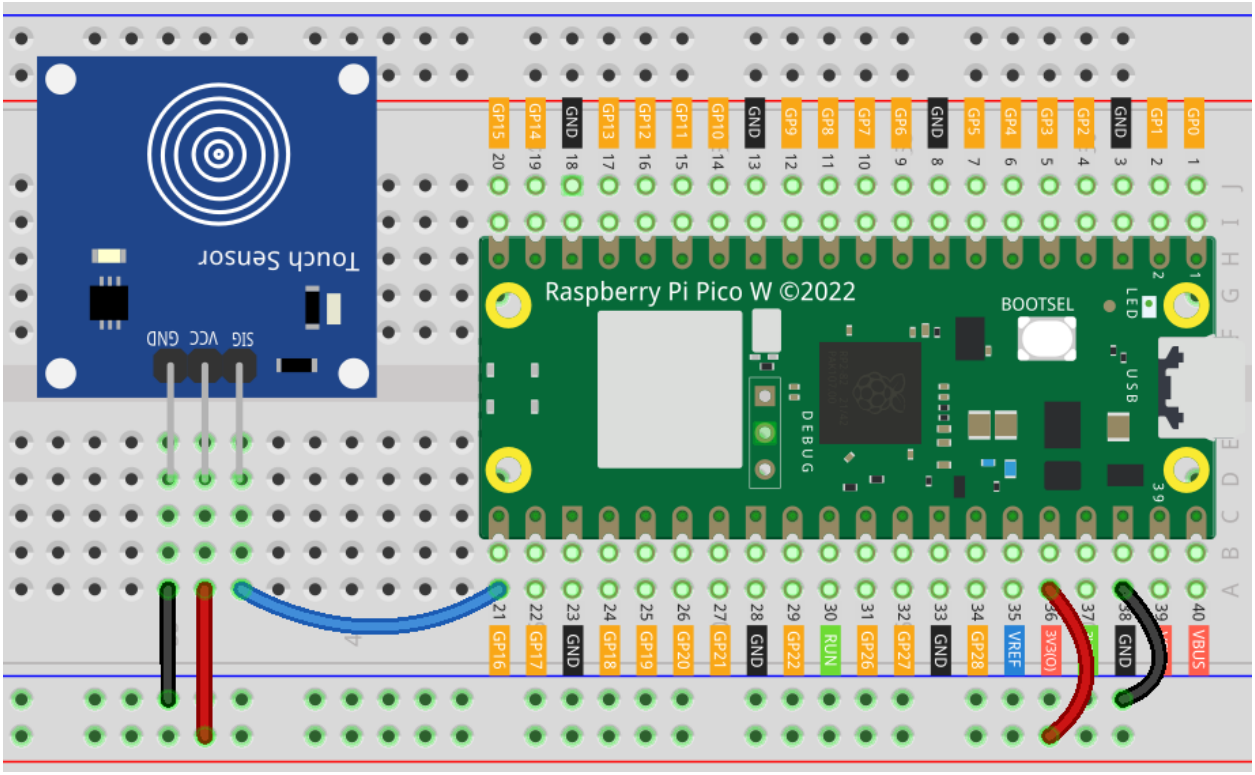
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Touch Sensor Module	
Breadboard	

Wiring



Code

```
from machine import Pin
import time

# Set GPIO 16 as an input pin to read the touch sensor state
touch_sensor = Pin(16, Pin.IN)

# Initialize the onboard LED of the Raspberry Pi Pico W
led = Pin("LED", Pin.OUT)

while True:
    if touch_sensor.value() == 1:
        led.value(1)  # Turn on the LED
        print("Touch detected!")
    else:
        led.value(0)  # Turn off the LED
        print("No touch detected")

    time.sleep(0.1)  # Short delay to reduce CPU usage
```

Code Analysis

1. **Setting up the pins**:

   Here, we import necessary libraries and set up GPIO pins. The touch sensor is connected to GPIO 16 as an input, and the onboard LED is configured as an output.

   ```
   from machine import Pin
   import time

   touch_sensor = Pin(16, Pin.IN)
   led = Pin("LED", Pin.OUT)
   ```

2. **Main loop and touch detection**:

   In an infinite loop, the code constantly checks the state of the touch sensor. If a touch is detected (value equals 1), the LED is turned on and a message is printed. Otherwise, the LED remains off, and a different message is printed. A short delay is added to reduce CPU usage.

   ```
   while True:
       if touch_sensor.value() == 1:
           led.value(1)  # Turn on the LED
           print("Touch detected!")
       else:
           led.value(0)  # Turn off the LED
           print("No touch detected")

       time.sleep(0.1)  # Short delay to reduce CPU usage
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.24 Lesson 23: Ultrasonic Sensor Module (HC-SR04)

In this lesson, you will learn how to measure distances using the Raspberry Pi Pico W and an HC-SR04 ultrasonic sensor. You'll find out how to connect the sensor to the Pico W and write a MicroPython script to control it. The lesson will cover calculating distances based on the time it takes for ultrasonic waves to reflect back from objects. This practical project provides insights into working with sensors, handling digital signals, and basic calculations in MicroPython, suitable for those interested in hardware interfacing with the Raspberry Pi Pico W.

Required Components

In this project, we need the following components.

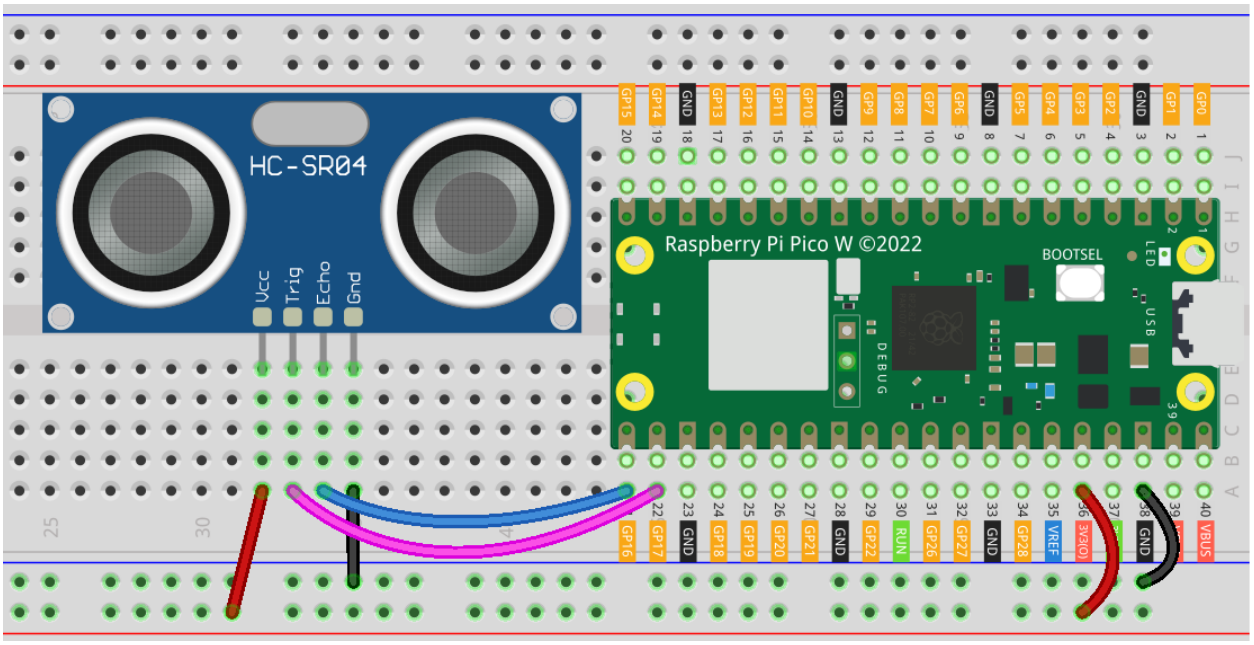
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Ultrasonic Sensor Module (HC-SR04)	
Breadboard	

Wiring



Code

```
import machine  # Import machine module for hardware control
import time  # Import time module for delays

# Define pin numbers for ultrasonic sensor's TRIG and ECHO pins
TRIG = machine.Pin(17, machine.Pin.OUT)  # TRIG pin set as output
ECHO = machine.Pin(16, machine.Pin.IN)  # ECHO pin set as input


def distance():
    # Function to calculate distance in centimeters
    TRIG.low()  # Set TRIG low
    time.sleep_us(2)  # Wait for 2 microseconds
    TRIG.high()  # Set TRIG high
    time.sleep_us(10)  # Wait for 10 microseconds
    TRIG.low()  # Set TRIG low again

    # Wait for ECHO pin to go high
    while not ECHO.value():
        pass

    time1 = time.ticks_us()  # Record time when ECHO goes high

    # Wait for ECHO pin to go low
    while ECHO.value():
        pass

    time2 = time.ticks_us()  # Record time when ECHO goes low
```

(continues on next page)

(continued from previous page)

```
    # Calculate the duration of the ECHO pin being high
    during = time.ticks_diff(time2, time1)

    # Return the calculated distance (using speed of sound)
    return during * 340 / 2 / 10000  # Distance in centimeters


# Main loop
while True:
    dis = distance()  # Get distance from sensor
    print("Distance: %.2f cm" % dis)  # Print distance
    time.sleep_ms(300)  # Wait for 300 milliseconds before next measurement
```

Code Analysis

1. **Importing libraries**

   The `machine` and `time` modules are imported for accessing hardware-specific functions and time-related functions, respectively.

   ```
   import machine
   import time
   ```

2. **Pin setup for HC-SR04**

   Two GPIO pins are defined for the HC-SR04 sensor: `TRIG` is an output pin to trigger the ultrasonic pulse, and `ECHO` is an input pin to receive the reflected pulse.

   ```
   TRIG = machine.Pin(17, machine.Pin.OUT)
   ECHO = machine.Pin(16, machine.Pin.IN)
   ```

3. **Distance measurement function**

   The `distance` function triggers the ultrasonic pulse and calculates the distance based on the time taken for the echo to return. It uses time-based functions to measure the duration of the echo.

   For more details, please refer to the working *principle* of the ultrasonic sensor module.

   ```
   def distance():
       TRIG.low()
       time.sleep_us(2)
       TRIG.high()
       time.sleep_us(10)
       TRIG.low()

       while not ECHO.value():
           pass

       time1 = time.ticks_us()

       while ECHO.value():
           pass
   ```

(continues on next page)

(continued from previous page)

```
    time2 = time.ticks_us()
    during = time.ticks_diff(time2, time1)
    return during * 340 / 2 / 10000
```

4. **Main loop**

   The main loop continuously calls the `distance` function and prints the measured distance. It waits for 300 milliseconds between each measurement to prevent sensor saturation.

```
while True:
    dis = distance()
    print("Distance: %.2f cm" % dis)
    time.sleep_ms(300)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.25 Lesson 24: Vibration Sensor Module (SW-420)

In this lesson, you will learn to connect and use a SW-420 Vibration Sensor Module with a Raspberry Pi Pico W. The course guides you through setting up the vibration sensor on GPIO 16 and writing a MicroPython script to monitor vibrations. You will write a loop to continually check the sensor's output, displaying a message when vibrations are detected. This practical exercise introduces you to working with external sensors on the Raspberry Pi Pico W, enhancing your understanding of hardware interfacing and programming in MicroPython.

Required Components

In this project, we need the following components.

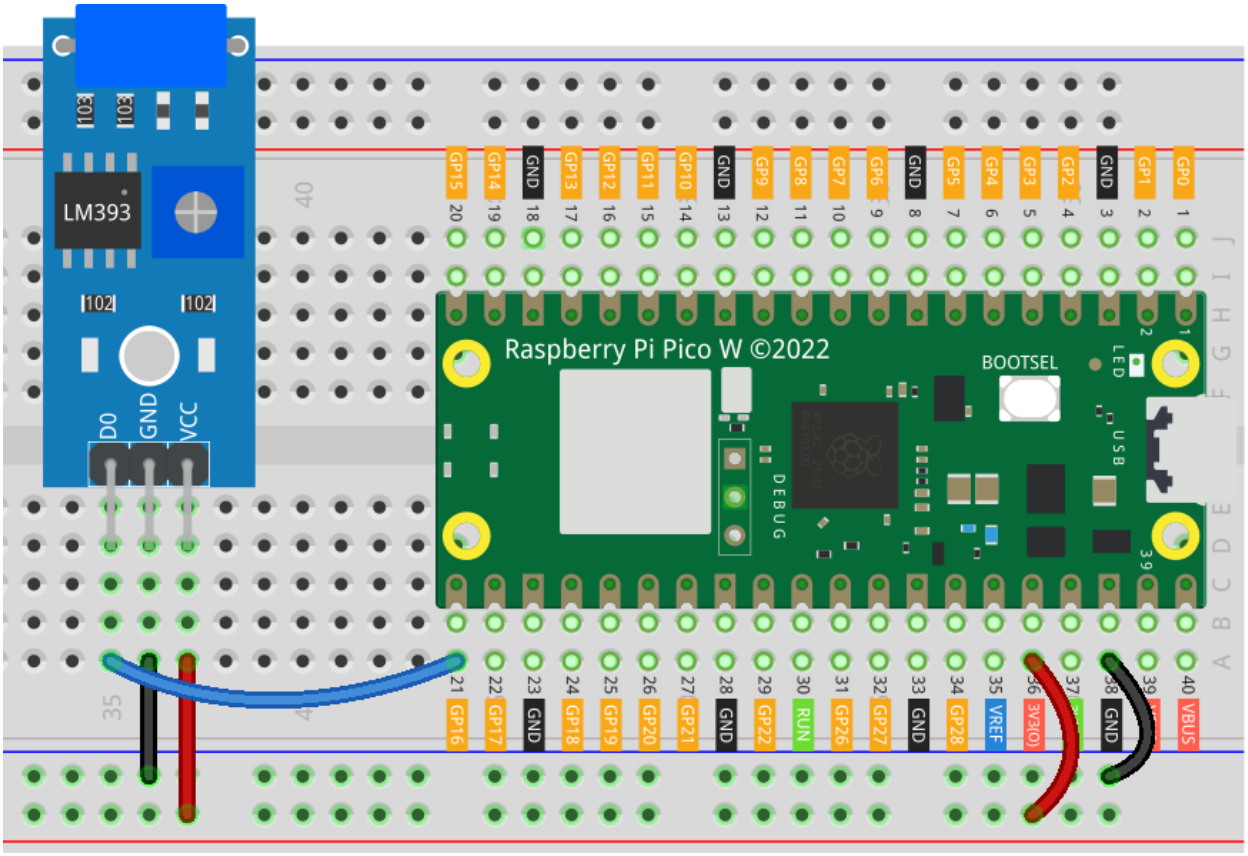
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Vibration Sensor Module (SW-420)	
Breadboard	

Wiring



Code

```
from machine import Pin
import time

# Initialize GPIO 16 as an input pin for the vibration sensor
vibration_sensor = Pin(16, Pin.IN)

# Continuously check the vibration sensor's state
while True:
    # If the sensor detects vibration (value is 1), print a message
    if vibration_sensor.value() == 1:
        print("Vibration detected!")
```

(continues on next page)

(continued from previous page)

```
    # If no vibration is detected, print ellipses
    else:
        print("...")

    # Pause for 0.1 seconds to lower the demand on the CPU
    time.sleep(0.1)
```

Code Analysis

1. Importing Required Libraries

   ```
   from machine import Pin
   import time
   ```

   This imports the `machine` module for hardware related operations and `time` module for handling time-related tasks.

2. Initializing the Vibration Sensor

   ```
   # Initialize GPIO 16 as an input pin for the vibration sensor
   vibration_sensor = Pin(16, Pin.IN)
   ```

   Here, GPIO 16 is set up as an input pin. The `Pin` class from the `machine` module is used to interact with the GPIO pins. `Pin.IN` configures it as an input.

3. Continuous Sensor Monitoring

   ```
   # Continuously check the vibration sensor's state
   while True:
   ```

   A `while True` loop is used to create an endless loop for continuously checking the sensor's state.

4. Checking Sensor State and Responding

   ```
   # If the sensor detects vibration (value is 1), print a message
   if vibration_sensor.value() == 1:
       print("Vibration detected!")
   # If no vibration is detected, print ellipses
   else:
       print("...")
   ```

   Within the loop, `vibration_sensor.value()` checks the current state of the sensor. If it returns 1, it indicates vibration is detected, and a message is printed. Otherwise, ellipses are printed.

5. Managing CPU Usage

   ```
   # Pause for 0.1 seconds to lower the demand on the CPU
   time.sleep(0.1)
   ```

   `time.sleep(0.1)` pauses the loop for 0.1 seconds. This is important to prevent the script from consuming too much CPU time.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.26 Lesson 25: Water Level Sensor Module

In this lesson, you will learn how to use the Raspberry Pi Pico W to measure water levels with a water level sensor. You'll understand how to connect the sensor to the board, read its analog output using MicroPython, and interpret these readings to determine water levels. This practical session is aimed at developing your skills in sensor integration and data acquisition with the Raspberry Pi Pico W.

Required Components

In this project, we need the following components.

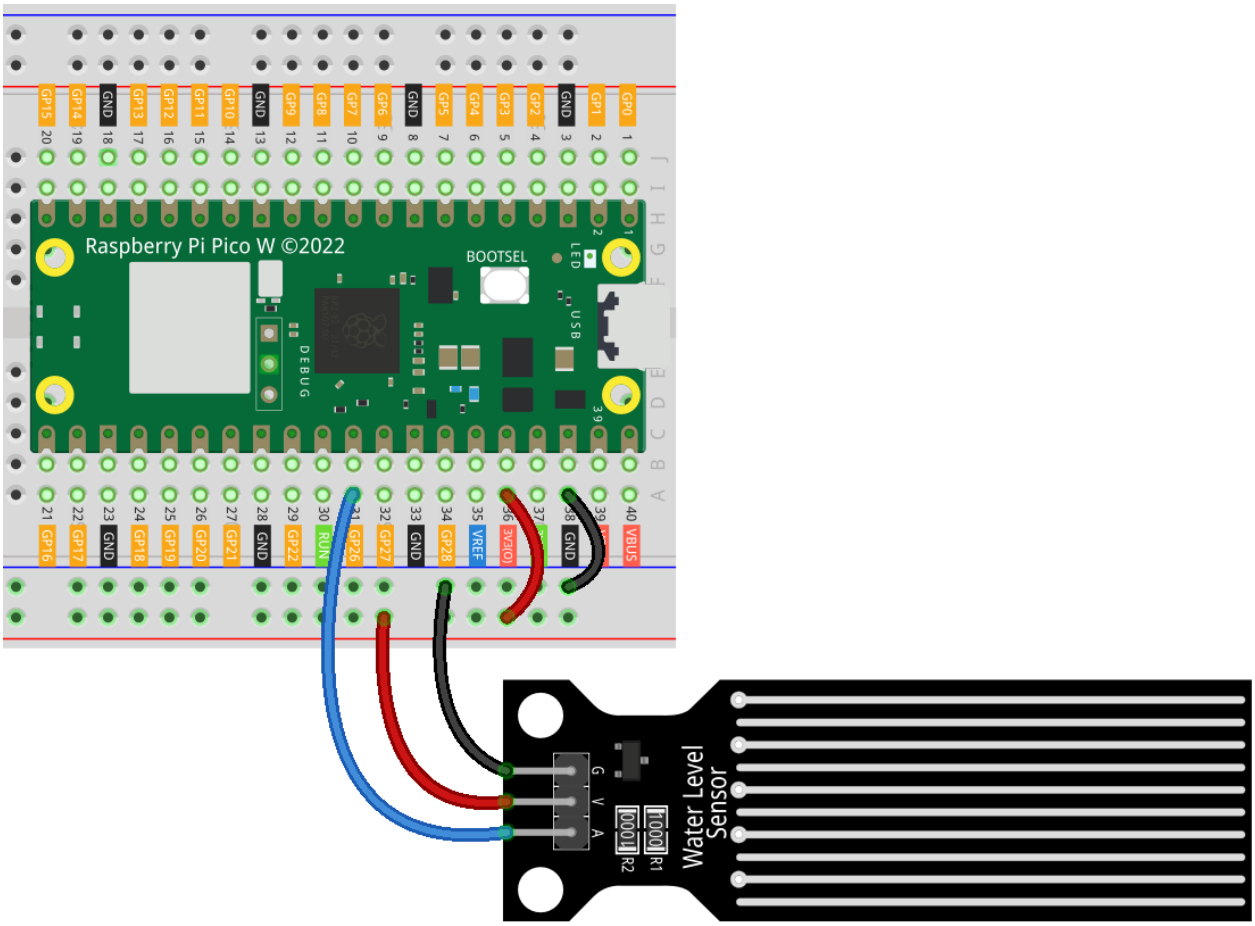
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Water Level Sensor Module	-
Breadboard	

Wiring



Code

```
import machine
import utime

# Initialize an ADC object on GPIO pin 26.
# This is typically used for reading analog signals.
water_level_sensor = machine.ADC(26)

# Continuously read and print sensor data.
while True:
    value = water_level_sensor.read_u16()  # Read and convert analog value to 16-bit␣
↪integer
    print("AO:", value)  # Print the analog value

    utime.sleep_ms(200)  # Wait for 200 milliseconds before the next read
```

Code Analysis

1. Importing Libraries

   Here, we import necessary libraries: `machine` for hardware interactions and `utime` for time-based functions.

   ```
   import machine
   import utime
   ```

2. Initializing the Water Level Sensor

   An ADC object is created on GPIO pin 26 to read analog signals from the water level sensor. ADC is crucial for converting the sensor's analog signals to digital format that the microcontroller can process.

   ```
   # Initialize an ADC object on GPIO pin 26.
   water_level_sensor = machine.ADC(26)
   ```

3. Reading and Printing Sensor Data

   The `while True` loop enables continuous reading of the sensor data. `read_u16` method converts the analog signal to a 16-bit integer. The value is printed, and the loop pauses for 200 milliseconds using `utime.sleep_ms(200)` to prevent rapid firing.

   ```
   while True:
       value = water_level_sensor.read_u16()  # Read and convert analog value to 16-
   ↪bit integer
       print("AO:", value)  # Print the analog value

       utime.sleep_ms(200)  # Wait for 200 milliseconds before the next read
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.27 Lesson 26: I2C LCD 1602

In this lesson, you will learn to connect an I2C LCD 1602 display to a Raspberry Pi Pico W. You'll understand how to set up I2C communication, display and clear messages on the LCD using MicroPython.

Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

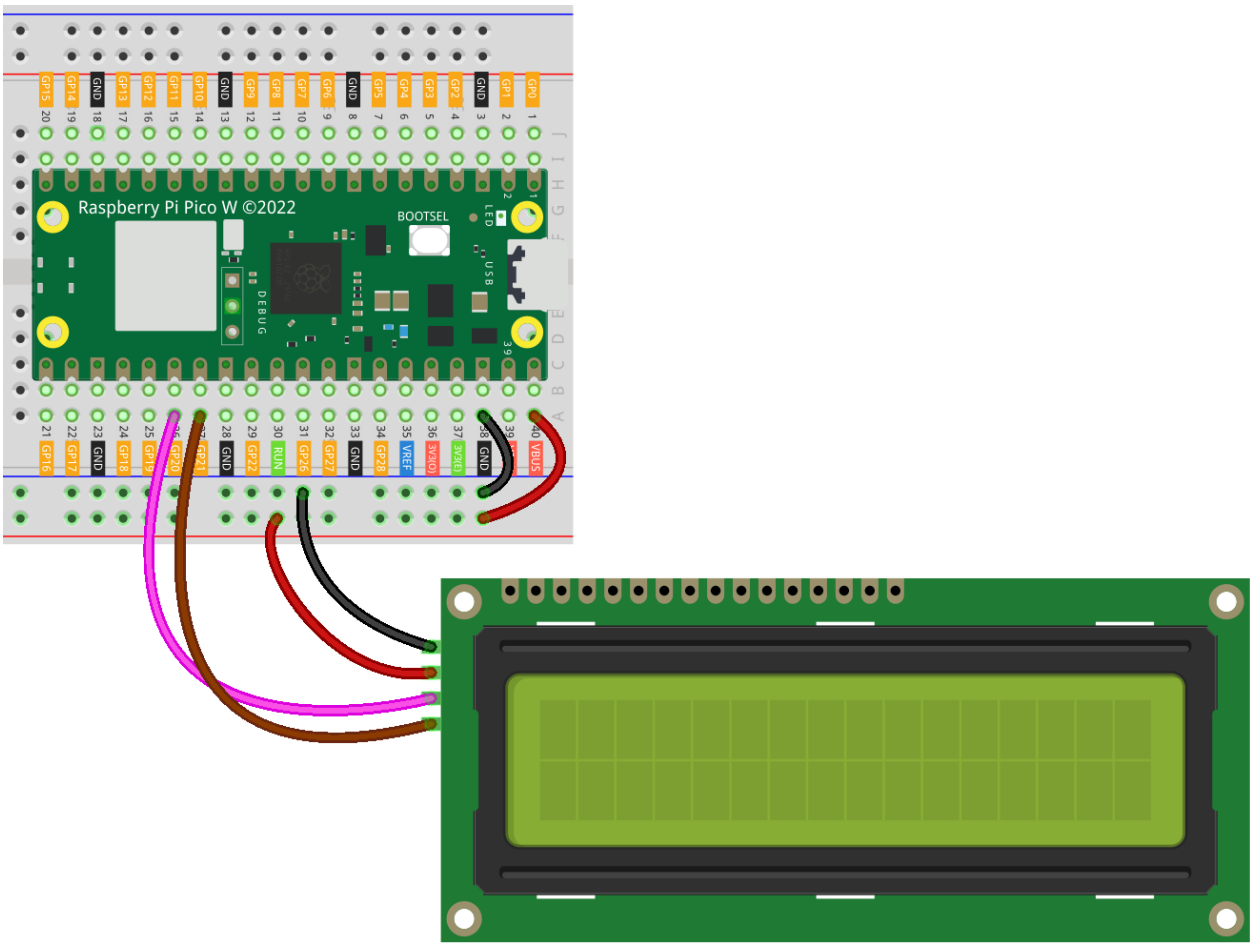
Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
I2C LCD 1602	
Breadboard	

Wiring

Note: To ensure the LCD module operates normally, please power it using the VBUS pin on the Pico.



Code

Note:

- Open the `26_lcd1602_module.py` file under the path of `universal-maker-sensor-kit-main/pico/Lesson_26_I2C_LCD1602_Module` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it. For detailed tutorials, please refer to *Open and Run Code Directly*.
- Here you need to use the `lcd1602.py`, please check if it has been uploaded to Pico W, for a detailed tutorial refer to *Upload the Libraries to Pico*.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.

```
from machine import I2C, Pin
from lcd1602 import LCD
import time

# Initialize I2C communication;
# Set SDA to pin 20, SCL to pin 21, and frequency to 400kHz
i2c = I2C(0, sda=Pin(20), scl=Pin(21), freq=400000)

# Create an LCD object for interfacing with the LCD1602 display
```

(continues on next page)

(continued from previous page)

```
lcd = LCD(i2c)

# Display the first message on the LCD
# Use '\n' to create a new line.
string = "SunFounder\n    LCD Tutorial"
lcd.message(string)
# Wait for 2 seconds
time.sleep(2)
# Clear the display
lcd.clear()

# Display the second message on the LCD
string = "Hello\n  World!"
lcd.message(string)
# Wait for 5 seconds
time.sleep(5)
# Clear the display before exiting
lcd.clear()
```

Code Analysis

1. Setting up I2C Communication

   The `machine` module is used to set up I2C communication. SDA (Serial Data) and SCL (Serial Clock) pins are defined (pin 20 and 21 respectively), along with the I2C frequency (400kHz).

   ```
   from machine import I2C, Pin
   i2c = I2C(0, sda=Pin(20), scl=Pin(21), freq=400000)
   ```

2. Initializing the LCD Display

   The `LCD` class from the `lcd1602` module is instantiated. This class handles the communication with the LCD display through I2C. An `LCD` object is created using the `i2c` object.

   For more usage of the `lcd1602` library, please refer to `lcd1602.py`.

   ```
   from lcd1602 import LCD
   lcd = LCD(i2c)
   ```

3. Displaying Messages on the LCD

   The `message` method of the `LCD` object is used to display text on the screen. The `\n` character creates a new line on the LCD. The `time.sleep()` function pauses execution for a specified number of seconds.

   ```
   string = "SunFounder\n    LCD Tutorial"
   lcd.message(string)
   time.sleep(2)
   lcd.clear()
   ```

4. Clearing the Display

   The `clear` method of the `LCD` object is called to clear the text from the display.

   ```
   lcd.clear()
   ```

5. Displaying a Second Message

   A new message is displayed, followed by a delay and then clearing the screen again.

```
string = "Hello\n  World!"
lcd.message(string)
time.sleep(5)
lcd.clear()
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.28 Lesson 27: OLED Display Module (SSD1306)

In this lesson, you will learn how to connect and display text and graphics on an OLED display module (SSD1306) using the Raspberry Pi Pico W. You'll set up the I2C communication, use MicroPython to program the Pico W to control the OLED display, and practice displaying simple text messages.

Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

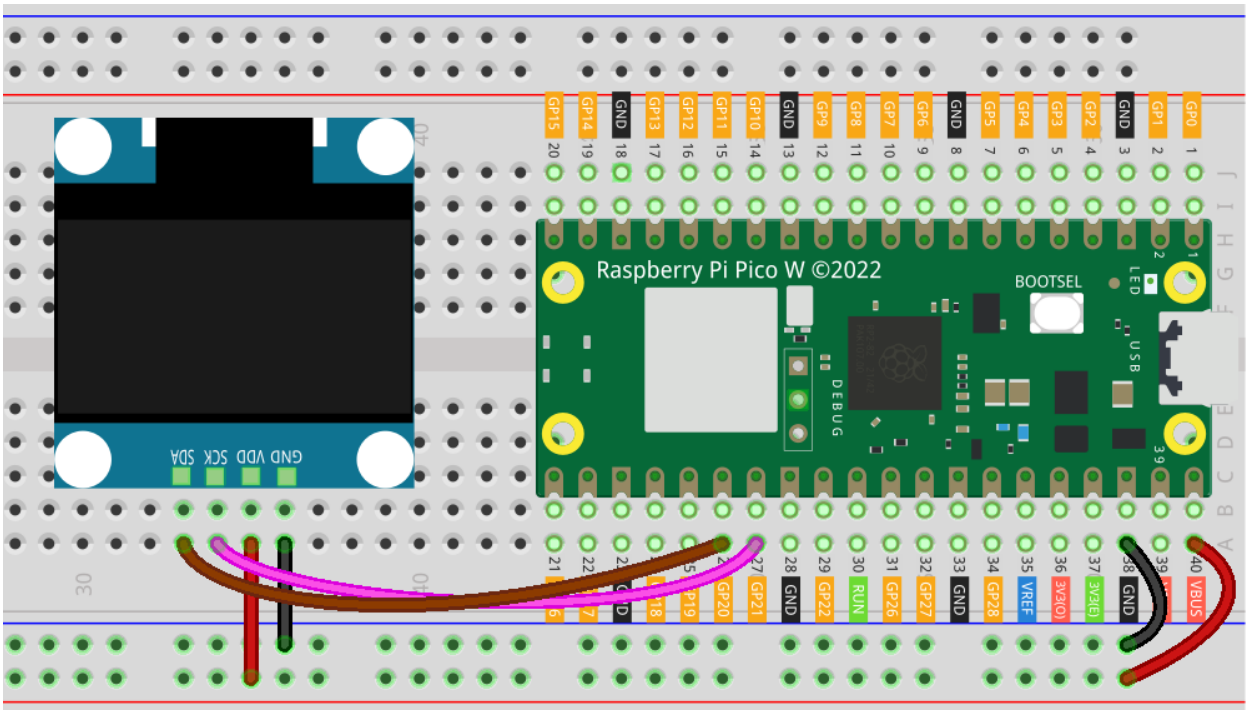
Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
OLED Display Module (SSD1306)	-
Breadboard	

Wiring

Note: To ensure the OLED module operates normally, please power it using the VBUS pin on the Pico.



Code

Note:

- Open the `27_ssd1306_oled_module.py` file under the path of `universal-maker-sensor-kit-main/pico/Lesson_27_SSD1306_OLED_Module` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it. For detailed tutorials, please refer to *Open and Run Code Directly*.
- Here you need to use the `ssd1306.py`, please check if it has been uploaded to Pico W, for a detailed tutorial refer to *Upload the Libraries to Pico*.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.

```
from machine import Pin, I2C
import ssd1306
import time

# setup the I2C communication
i2c = I2C(0, sda=Pin(20), scl=Pin(21))

# Set up the OLED display (128x64 pixels) on the I2C bus
# SSD1306_I2C is a subclass of FrameBuffer. FrameBuffer provides support for graphics␣
↪primitives.
```

(continues on next page)

(continued from previous page)

```
# http://docs.micropython.org/en/latest/pyboard/library/framebuf.html
oled = ssd1306.SSD1306_I2C(128, 64, i2c)

# Clear the display by filling it with white and then showing the update
oled.fill(1)
oled.show()
time.sleep(1)  # Wait for 1 second

# Clear the display again by filling it with black
oled.fill(0)
oled.show()
time.sleep(1)  # Wait for another second

# Display text on the OLED screen
oled.text('Hello,', 0, 0)  # Display "Hello," at position (0, 0)
oled.text('sunfounder.com', 0, 16)  # Display "sunfounder.com" at position (0, 16)

# The following line sends what to show to the display
oled.show()
```

Code Analysis

1. Initializing the I2C communication:

   This code segment sets up the I2C communication protocol. I2C is a standard protocol for communication between devices. It uses two lines: SDA (data line) and SCL (clock line).

   ```
   from machine import Pin, I2C
   i2c = I2C(0, sda=Pin(20), scl=Pin(21))
   ```

2. Setting up the OLED display:

   Here, we initialize the SSD1306 OLED display with the I2C protocol. The parameters 128 and 64 define the width and height of the display in pixels, respectively.

   For more information about the `ssd1306` library, please visit .

   ```
   import ssd1306
   oled = ssd1306.SSD1306_I2C(128, 64, i2c)
   ```

3. Clearing the display:

   The display is cleared by filling it with white (1) and then updating the display with `oled.show()`. The `time.sleep(1)` command adds a one-second delay. Then, the display is cleared again by filling it with black (0).

   SSD1306_I2C is a subclass of FrameBuffer, which supports graphics primitives. If you want to display other patterns, please refer to .

   ```
   oled.fill(1)
   oled.show()
   time.sleep(1)
   oled.fill(0)
   oled.show()
   time.sleep(1)
   ```

4. Displaying text:

   The `oled.text` method is used to display text on the screen. The parameters are the text to display and the x, y coordinates on the screen. Finally, `oled.show()` updates the display to show the text.

```
oled.text('Hello,', 0, 0)
oled.text('sunfounder.com', 0, 16)
oled.show()
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.29 Lesson 28: RGB LED Module

In this lesson, you'll learn how to control an RGB LED using the Raspberry Pi Pico W. You'll discover how to set up PWM (Pulse Width Modulation) on different GPIO pins for each color channel of the RGB LED, allowing you to create various colors by adjusting the intensity of red, green, and blue components. This project offers beginners a great opportunity to gain practical experience with PWM and color mixing on Raspberry Pi Pico W using MicroPython. Additionally, you'll learn how to handle interrupts to safely turn off the LED. This lesson provides a fun and interactive way to explore the basics of electronics and programming.

Required Components

In this project, we need the following components.

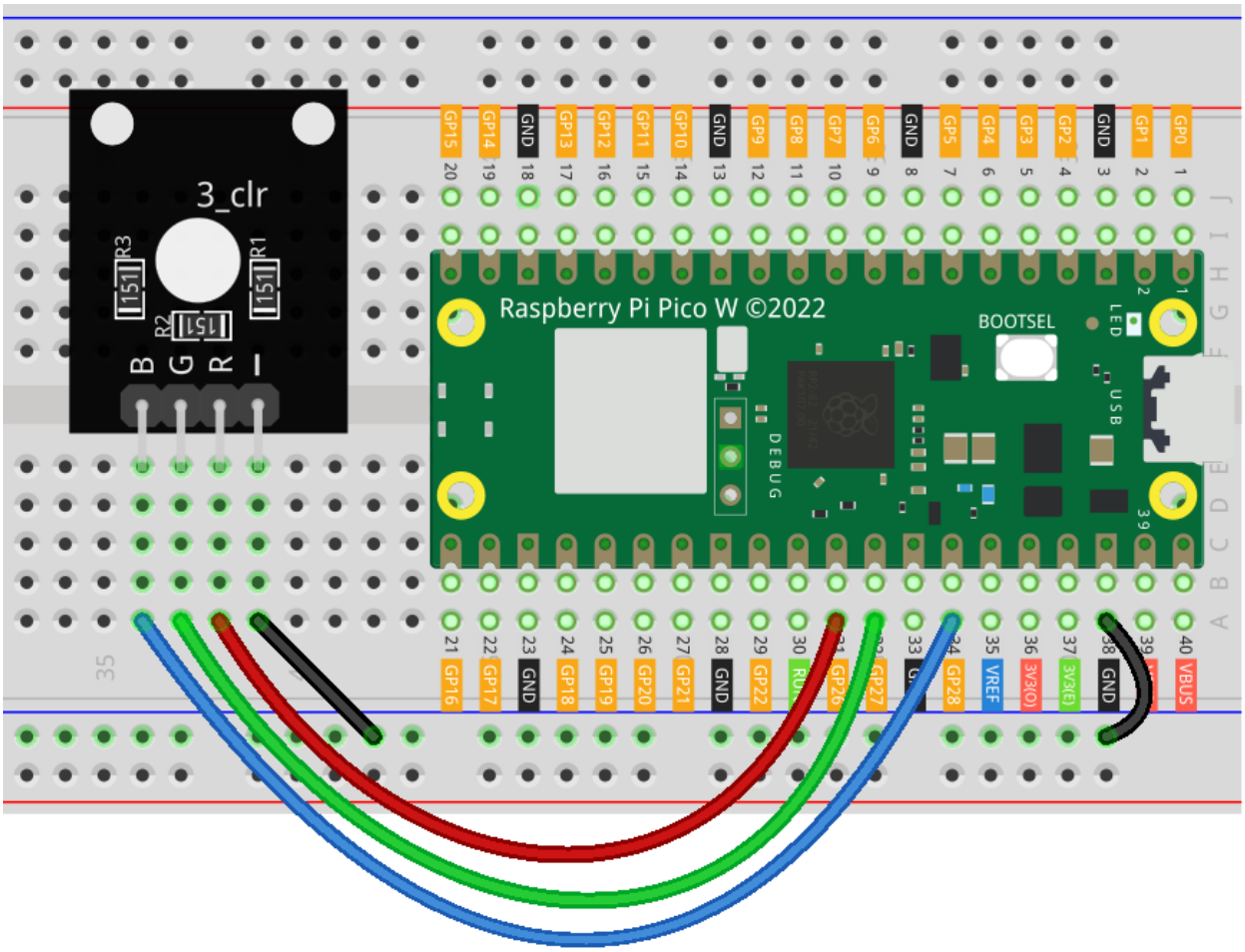
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
RGB LED Module	-
Breadboard	

Wiring



Code

```
from machine import Pin, PWM
from time import sleep

# Initialize PWM for each color channel of an RGB LED
red = PWM(Pin(26))  # Red channel on GPIO pin 26
green = PWM(Pin(27))  # Green channel on GPIO pin 27
blue = PWM(Pin(28))  # Blue channel on GPIO pin 28

# Set 1000 Hz frequency for all channels
red.freq(1000)
green.freq(1000)
blue.freq(1000)


# Function to set RGB LED color
def set_color(r, g, b):
    red.duty_u16(r)  # Red intensity
```

(continues on next page)

(continued from previous page)

```
    green.duty_u16(g)  # Green intensity
    blue.duty_u16(b)  # Blue intensity


try:
    while True:
        set_color(65535, 0, 0)  # Red
        sleep(1)
        set_color(0, 65535, 0)  # Green
        sleep(1)
        set_color(0, 0, 65535)  # Blue
        sleep(1)
except KeyboardInterrupt:
    set_color(0, 0, 0)  # Turn off RGB LED on interrupt
```

Code Analysis

1. Importing Libraries

   The `machine` module is imported to use the PWM class and Pin class. The `time` module is imported to use the `sleep` function for creating delays.

   ```
   from machine import Pin, PWM
   from time import sleep
   ```

2. Initializing PWM for RGB LED

   The RGB LED has three channels (Red, Green, Blue), each controlled by a separate PWM signal. The PWM signals are connected to GPIO pins 26, 27, and 28.

   ```
   red = PWM(Pin(26))  # Red channel on GPIO pin 26
   green = PWM(Pin(27))  # Green channel on GPIO pin 27
   blue = PWM(Pin(28))  # Blue channel on GPIO pin 28
   ```

3. Setting Frequency for PWM Signals

   The frequency of the PWM signals is set to 1000 Hz for all three channels.

   ```
   red.freq(1000)
   green.freq(1000)
   blue.freq(1000)
   ```

4. Defining the set_color Function

   This function sets the color of the RGB LED. The `duty_u16` method is used to set the duty cycle for each color channel, which determines the intensity of that color.

   ```
   def set_color(r, g, b):
       red.duty_u16(r)
       green.duty_u16(g)
       blue.duty_u16(b)
   ```

5. Main Program Loop

An infinite loop is used to change the color of the LED. The `set_color` function is called with different values to display red, green, and blue colors. Each color is displayed for 1 second.

```
try:
    while True:
        set_color(65535, 0, 0)  # Red
        sleep(1)
        set_color(0, 65535, 0)  # Green
        sleep(1)
        set_color(0, 0, 65535)  # Blue
        sleep(1)
except KeyboardInterrupt:
    set_color(0, 0, 0)  # Turn off RGB LED on interrupt
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.30 Lesson 29: Traffic Light Module

In this lesson, you will learn to create a traffic light system using the Raspberry Pi Pico W. You'll program the Pico W to control three LEDs – red, yellow, and green – mimicking a real traffic light. This project offers a practical introduction to using Pulse Width Modulation (PWM) for LED brightness control and basic control structures in MicroPython. It's ideal for beginners looking to explore digital signal processing and gain confidence in coding on the Raspberry Pi Pico W platform.

Required Components

In this project, we need the following components.

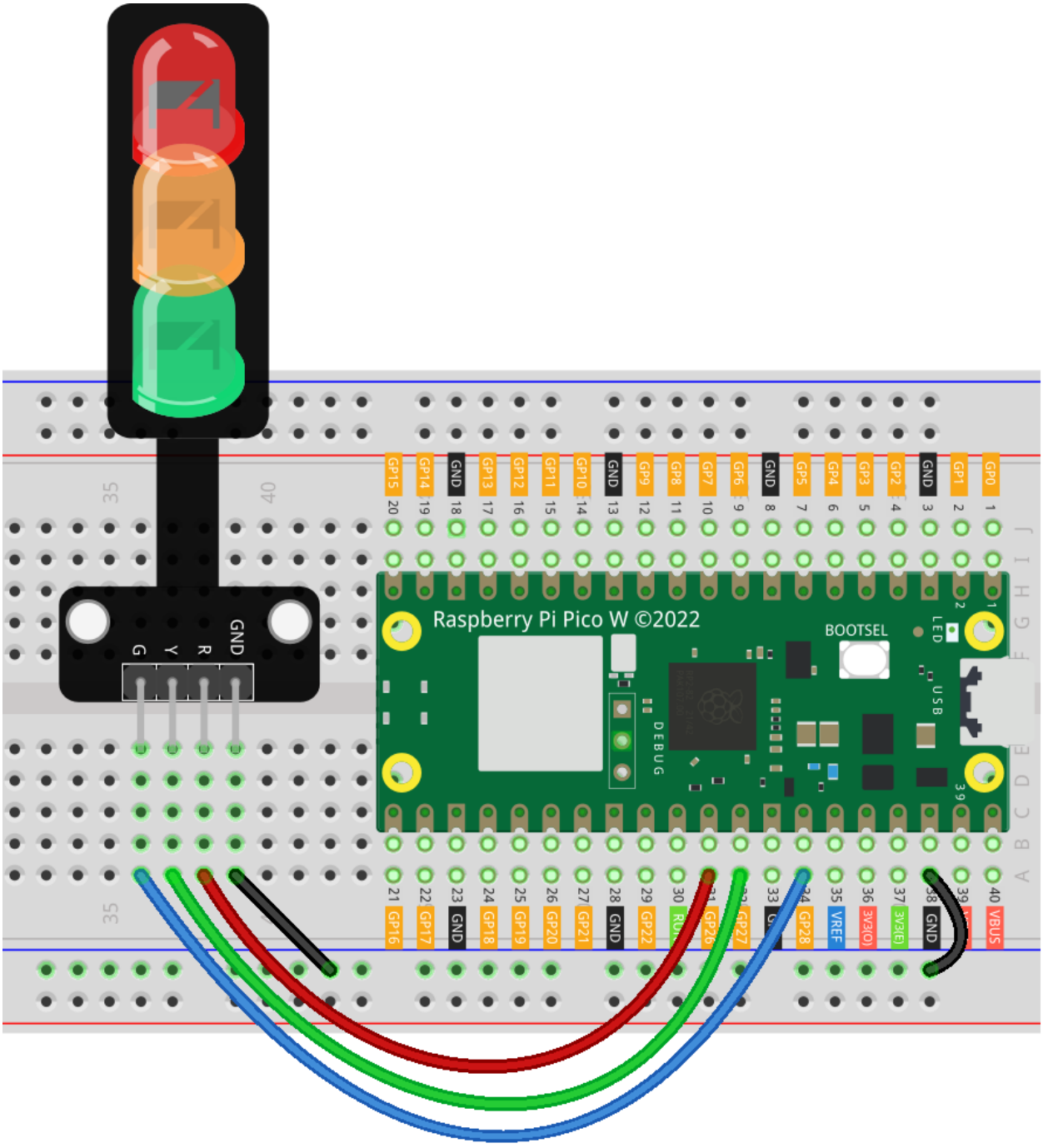
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Traffic Light Module	
Breadboard	

Wiring


G
Y
R
GND
Raspberry Pi Pico W ©2022
BOOTSEL
LED
USB
DEBUG
GP15
GP14
GND
GP13
GP12
GP11
GP10
GND
GP9
GP8
GP7
GP6
GND
GP5
GP4
GP3
GP2
GND
GP1
GP0
GP16
GP17
GND
GP18
GP19
GP20
GP21
GND
GP22
GP26
GP27
GP28
VREF
3V3(O)
3V3(E)
GND
VBUS

Code

```
from machine import Pin, PWM
import time

# Initialize pins for LEDs
red = PWM(Pin(26), freq=1000)  #  red LED
yellow = PWM(Pin(27), freq=1000)  #  yellow LED
green = PWM(Pin(28), freq=1000)  # green LED


# Function to set the brightness of an LED (0-100%)
def set_brightness(led, brightness):
    if brightness < 0 or brightness > 100:
        raise ValueError("Brightness should be between 0 and 100")
    led.duty_u16(int(brightness / 100 * 65535))


try:
    # Example sequence
    while True:

        # Green light for 5 seconds
        set_brightness(green, 100)
        time.sleep(5)
        set_brightness(green, 0)

        # Blink Yellow light
        set_brightness(yellow, 100)
        time.sleep(0.5)
        set_brightness(yellow, 0)
        time.sleep(0.5)
        set_brightness(yellow, 100)
        time.sleep(0.5)
        set_brightness(yellow, 0)
        time.sleep(0.5)
        set_brightness(yellow, 100)
        time.sleep(0.5)
        set_brightness(yellow, 0)
        time.sleep(0.5)

        # Red light for 5 seconds
        set_brightness(red, 100)
        time.sleep(5)
        set_brightness(red, 0)

except KeyboardInterrupt:
    # Turn off RGB LED on interrupt
    set_brightness(red, 0)
    set_brightness(yellow, 0)
    set_brightness(green, 0)
```

Code Analysis

1. Importing Libraries

   The `machine` library is used for controlling hardware components, and `time` is used for creating delays.

```
from machine import Pin, PWM
import time
```

2. Initializing LED Pins

   Here, we initialize the pins connected to the LEDs. PWM is used to control the brightness of the LEDs.

```
red = PWM(Pin(26), freq=1000)  #  red LED
yellow = PWM(Pin(27), freq=1000)  #  yellow LED
green = PWM(Pin(28), freq=1000)  # green LED
```

3. Defining the Set Brightness Function

   Note: Due to the fact that the pins of Raspberry Pi Pico can only output a maximum voltage of 3.3V, the green LED will appear dim.

   This function sets the brightness of the LEDs. It takes two parameters: the LED and the desired brightness level (0-100%). The `duty_u16` method is used to set the PWM duty cycle.

```
def set_brightness(led, brightness):
    if brightness < 0 or brightness > 100:
        raise ValueError("Brightness should be between 0 and 100")
    led.duty_u16(int(brightness / 100 * 65535))
```

4. Main Loop and Traffic Light Sequence

   The `while True` loop makes the code run continuously. It controls the sequence of the traffic light: green, yellow (blinking), and red.

```
try:
    while True:
        # Green light for 5 seconds
        set_brightness(green, 100)
        time.sleep(5)
        set_brightness(green, 0)
        ...
```

5. Handling Keyboard Interrupt

   The `except KeyboardInterrupt` block is used to handle a manual interruption (like Ctrl+C). It turns off all LEDs when the script is interrupted.

```
except KeyboardInterrupt:
    set_brightness(red, 0)
    set_brightness(yellow, 0)
    set_brightness(green, 0)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.31 Lesson 30: Relay Module

In this lesson, you will learn how to use the Raspberry Pi Pico W to control a relay module. We will set up a basic circuit connecting the relay to the Pi and write a MicroPython script to toggle the relay on and off at one-second intervals. This project introduces you to controlling external devices such as relays and demonstrates practical output operations using the GPIO pins on the Raspberry Pi Pico W. Ideal for those interested in delving into home automation or managing other high-power devices, this lesson offers fundamental insight into how microcontrollers can interact with and control external hardware.

Required Components

In this project, we need the following components.

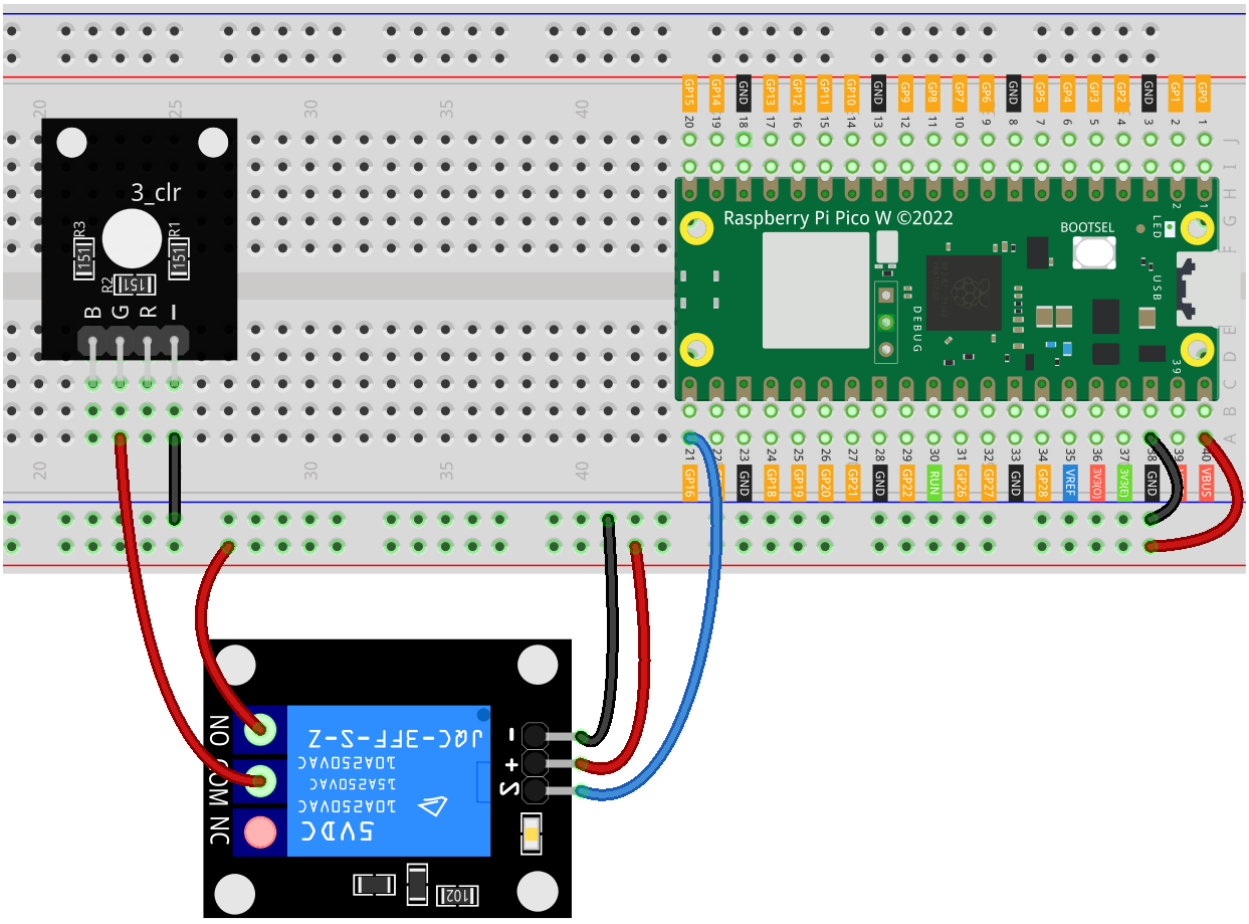
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
5V Relay Module	-
RGB LED Module	-
Breadboard	

Wiring



Code

```
from machine import Pin
import time

# Replace this number with the GPIO pin number your relay is connected to
relay_pin = Pin(16, Pin.OUT)

def relay_on():
    relay_pin.value(1)  # Set relay to ON state

def relay_off():
    relay_pin.value(0)  # Set relay to OFF state

try:
    while True:
        relay_on()
        print("on....")
        time.sleep(1)  # Wait for 1 second
        relay_off()
```

(continues on next page)

(continued from previous page)

```
        print("off....")
        time.sleep(1)  # Wait for 1 second
except:
    relay_off()  # Ensure the relay is turned off in case of an exception
    print("Program interrupted, relay turned off.")
```

Code Analysis

1. Importing Libraries

   The `machine` and `time` libraries are imported to interact with GPIO pins and handle time-related functions, respectively.

   ```
   from machine import Pin
   import time
   ```

2. Initializing Relay Pin

   A GPIO pin is set up as an output pin to control the relay. The `relay_pin` variable represents the GPIO pin connected to the relay.

   ```
   relay_pin = Pin(16, Pin.OUT)
   ```

3. Defining Relay Control Functions

   Two functions, `relay_on` and `relay_off`, are defined to turn the relay on and off, respectively. These functions change the GPIO pin's value to high (1) or low (0).

   ```
   def relay_on():
       relay_pin.value(1)  # Set relay to ON state

   def relay_off():
       relay_pin.value(0)  # Set relay to OFF state
   ```

4. Main Loop and Exception Handling

   A continuous loop is created using `while True`. Inside this loop, the relay is turned on and off with a 1-second delay between each state. If an interruption occurs (like a keyboard interruption), the relay is turned off for safety, and a message is printed.

   ```
   try:
       while True:
           relay_on()
           print("on....")
           time.sleep(1)  # Wait for 1 second
           relay_off()
           print("off....")
           time.sleep(1)  # Wait for 1 second
   except:
       relay_off()  # Ensure the relay is turned off in case of an exception
       print("Program interrupted, relay turned off.")
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.32 Lesson 31: Centrifugal Pump

In this lesson, you will learn how to operate a centrifugal pump using the Raspberry Pi Pico W and an L9110 motor control board. We'll guide you through the process of configuring two PWM (Pulse Width Modulation) pins to control the motor. You'll set up the pump to run for 5 seconds and then turn off. This practical exercise offers a valuable opportunity to delve into motor control mechanisms and PWM signals, crucial in microcontroller programming.

Required Components

In this project, we need the following components.

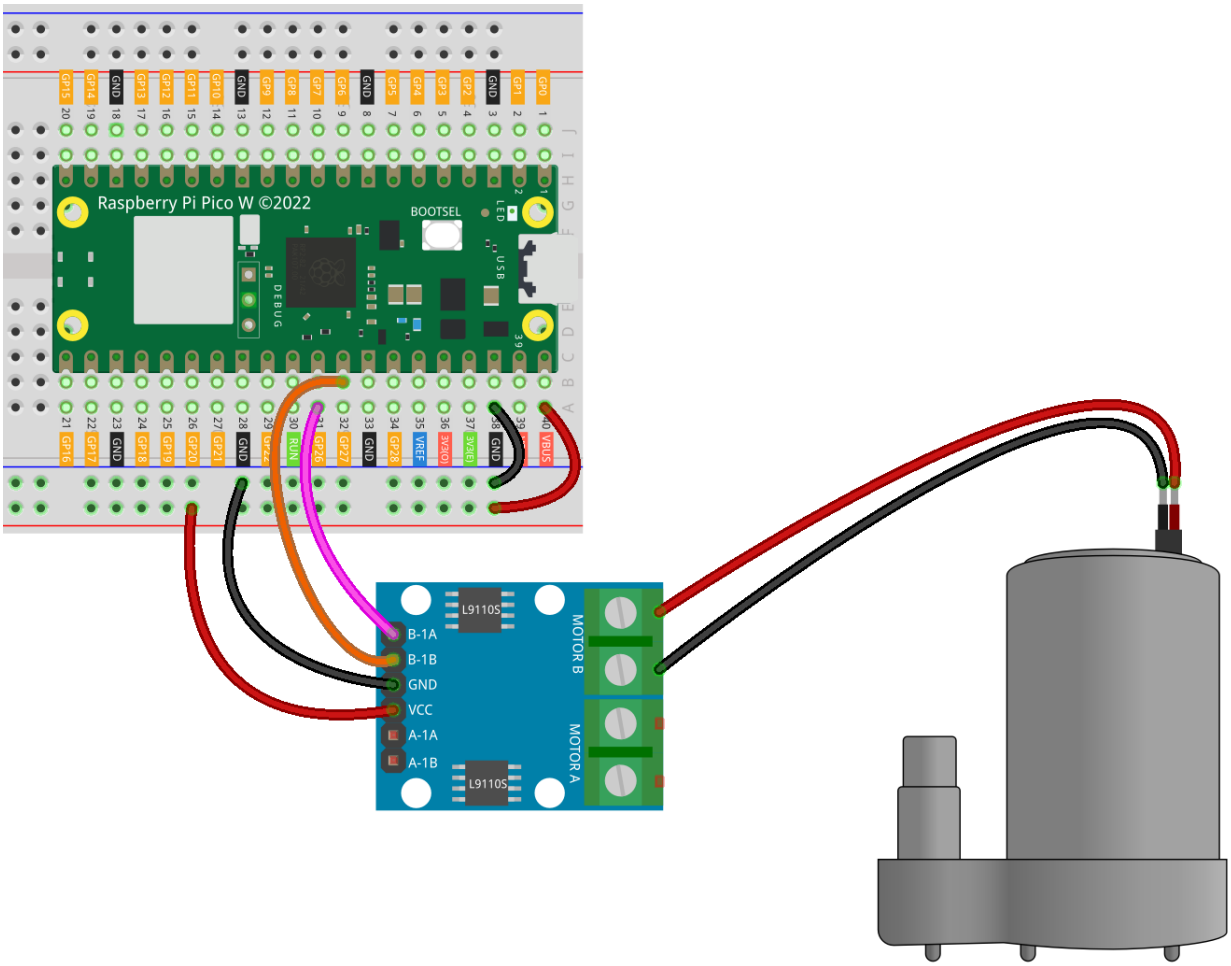
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Centrifugal Pump	-
L9110 Motor Driver Module	-
Breadboard	

Wiring



Code

```
from machine import Pin, PWM
import time

pump_a = PWM(Pin(26), freq=1000)
pump_b = PWM(Pin(27), freq=1000)

# turn on pump
pump_a.duty_u16(0)
pump_b.duty_u16(65535)  # speed(0-65535)

time.sleep(5)

# turn off pump
pump_a.duty_u16(0)
pump_b.duty_u16(0)
```

Code Analysis

1. Importing Libraries

    - The `machine` module is imported to interact with the GPIO pins and PWM functionalities of the Raspberry Pi Pico W.
    - The `time` module is used for creating delays in the code.

```
from machine import Pin, PWM
import time
```

2. Initializing PWM Objects

    - Two PWM objects, `pump_a` and `pump_b`, are created. They correspond to GPIO pins 26 and 27, respectively.
    - The frequency for PWM is set to 1000 Hz, a common frequency for motor control.

```
pump_a = PWM(Pin(26), freq=1000)
pump_b = PWM(Pin(27), freq=1000)
```

3. Turning on the Pump

    - `pump_a.duty_u16(0)` sets the duty cycle of `pump_a` pin to 0, while `pump_b.duty_u16(65535)` sets the duty cycle of `pump_b` pin to 65535, running the motor at full speed. For more details, please refer to *the working principle of L9110*.
    - The pump runs for 5 seconds, controlled by `time.sleep(5)`.

```
# turn on pump
pump_a.duty_u16(0)
pump_b.duty_u16(65535)  # speed(0-65535)
time.sleep(5)
```

4. Turning off the Pump

    Both `pump_a` and `pump_b` are set to a duty cycle of 0, stopping the motor.

```
# turn off pump
pump_a.duty_u16(0)
pump_b.duty_u16(0)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.33 Lesson 32: Passive Buzzer Module

In this lesson, you will learn how to use the passive buzzer on Raspberry Pi Pico W to play single notes and perform music. You will understand how to use PWM (Pulse Width Modulation) to set up the buzzer on GPIO 16 and use the music class in the buzzer_music library to play complete songs. This course will guide you step by step through playing single notes, and then further execute full melodies such as "Happy Birthday". This project is very suitable for beginners, providing a practical way to understand musical tones and integrate external libraries in MicroPython on Raspberry Pi Pico W.

Required Components

In this project, we need the following components.

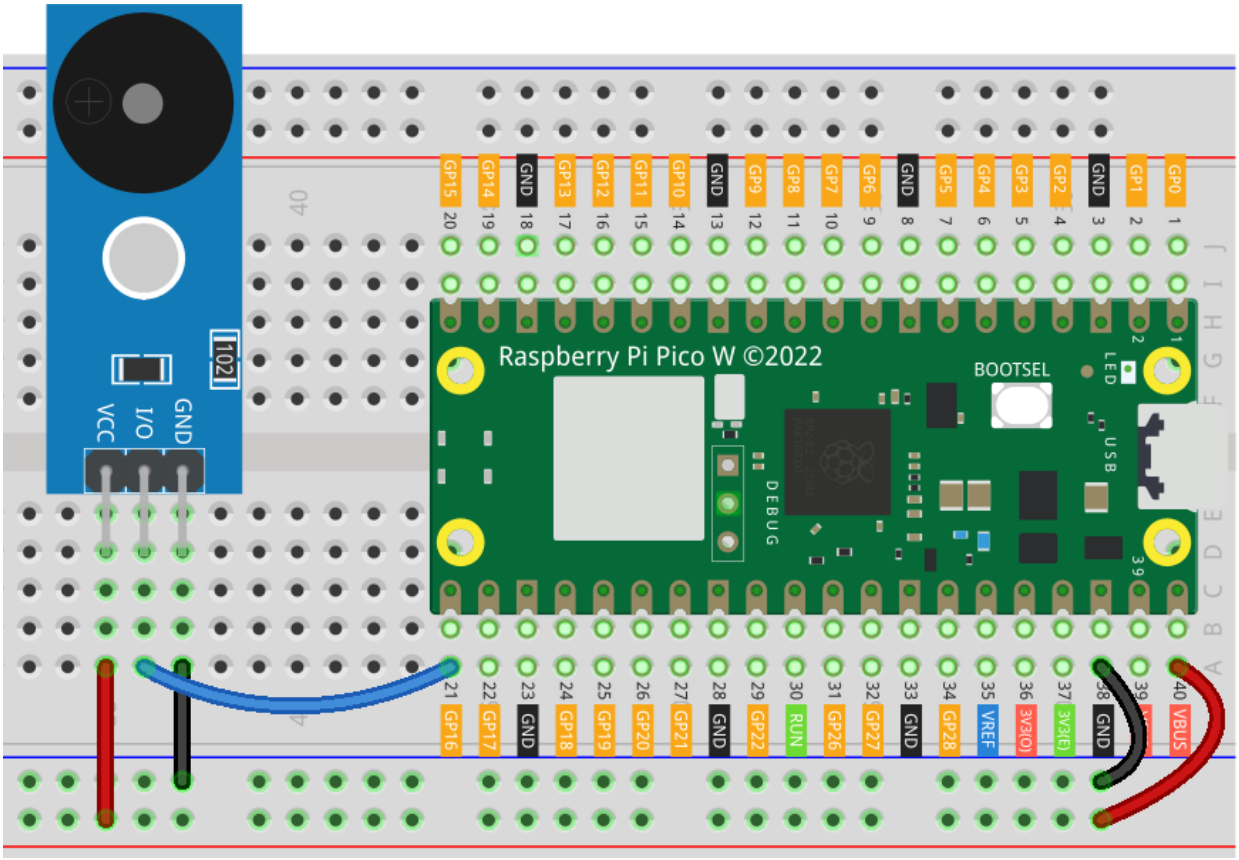
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Passive Buzzer Module	
Breadboard	

Wiring



Code

```
import machine
import time

# Initialize the PWM on GPIO 16 for the buzzer
buzzer = machine.PWM(machine.Pin(16))

def tone(pin, frequency, duration):
    """Play a tone on the given pin at the specified frequency and duration."""
    pin.freq(frequency)
    pin.duty_u16(30000)
    time.sleep_ms(duration)
    pin.duty_u16(0)

# Play individual notes
tone(buzzer, 440, 250)  # A4
time.sleep(0.5)
tone(buzzer, 494, 250)  # B4
time.sleep(0.5)
tone(buzzer, 523, 250)  # C5
time.sleep(1)
```

(continues on next page)

(continued from previous page)

```
# Import the music class from the buzzer_music module for easy song playback.
from buzzer_music import music

# Find some music on onlinesequencer.net, click edit, select all notes with CTRL + A and␣
↪then copy them with CTRL + C
# Paste the string to song, making sure to remove the "Online Sequencer:120233:" from␣
↪the start and the ";:" from the end
# https://onlinesequencer.net/2474257 Happy Birthday (by Sudirth)
song = "0 G4 3 0;3 G4 1 0;4 A4 4 0;8 G4 4 0;12 C5 4 0;16 B4 8 0;24 G4 3 0;27 G4 1 0;28␣
↪A4 4 0;32 G4 4 0;36 D5 4 0;40 C5 8 0;48 G4 3 0;51 G4 1 0;52 G5 4 0;56 E5 4 0;60 C5 4 0;
↪64 B4 4 0;68 A4 4 0;72 F5 3 0;75 F5 1 0;76 E5 4 0;80 C5 4 0;84 D5 4 0;88 C5 8 0"

# Initialize the music class with the song and set the buzzer pin
mySong = music(song, pins=[machine.Pin(16)])

# Play music using the music class.
while True:
    print(mySong.tick())
    time.sleep(0.04)
```

Code Analysis

1. Initialization

   Import necessary modules and initialize the PWM on a specific GPIO pin to control the buzzer.

   ```
   import machine
   import time

   # Initialize the PWM on GPIO 16 for the buzzer
   buzzer = machine.PWM(machine.Pin(16))
   ```

2. Defining the tone function

   This function allows playing a single tone at a specified frequency and duration. It sets the frequency and duty cycle (volume) of the PWM signal.

   ```
   def tone(pin, frequency, duration):
       """Play a tone on the given pin at the specified frequency and duration."""
       pin.freq(frequency)
       pin.duty_u16(30000)
       time.sleep_ms(duration)
       pin.duty_u16(0)
   ```

3. Playing individual notes

   Here, the `tone` function is used to play individual notes. The parameters include the note's frequency (in Hz) and its duration (in milliseconds).

```
# Play individual notes
tone(buzzer, 440, 250)  # A4
time.sleep(0.5)
tone(buzzer, 494, 250)  # B4
time.sleep(0.5)
tone(buzzer, 523, 250)  # C5
time.sleep(1)
```

4. Using the buzzer_music library

   The `buzzer_music` library is imported, and a song string is prepared.

   You can find some music on onlinesequencer.net, click edit, select all notes with CTRL + A and then copy them with CTRL + C. Paste the string to `song`, making sure to remove the "Online Sequencer:120233:" from the start and the ";:" from the end.

   For more information about the `buzzer_music` library, please visit .

```
# Import the music class from the buzzer_music module for easy song playback.
from buzzer_music import music

# https://onlinesequencer.net/2474257 Happy Birthday (by Sudirth)
song = "0 G4 3 0;3 G4 1 0;4 A4 4 0;8 G4 4 0;12 C5 4 0;16 B4 8 0;24 G4 3 0;27 G4 1 0;
↪28 A4 4 0;32 G4 4 0;36 D5 4 0;40 C5 8 0;48 G4 3 0;51 G4 1 0;52 G5 4 0;56 E5 4 0;
↪60 C5 4 0;64 B4 4 0;68 A4 4 0;72 F5 3 0;75 F5 1 0;76 E5 4 0;80 C5 4 0;84 D5 4 0;
↪88 C5 8 0"
```

5. Initializing and playing the song

   The `music` class is initialized with the song string and the GPIO pin for the buzzer. The music is played in a loop using the `tick` method of the `music` class.

```
# Initialize the music class with the song and set the buzzer pin
mySong = music(song, pins=[machine.Pin(16)])

# Play music using the music class.
while True:
    print(mySong.tick())
    time.sleep(0.04)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.34 Lesson 33: Servo Motor (SG90)

In this lesson, you will learn how to control a servo motor (SG90) using the Raspberry Pi Pico W. You will be introduced to the concepts of Pulse Width Modulation (PWM) for controlling the angle of the servo motor. The lesson includes writing a MicroPython script to make the servo sweep smoothly through its entire range of motion, from 0 to 180 degrees and back.

Required Components

In this project, we need the following components.

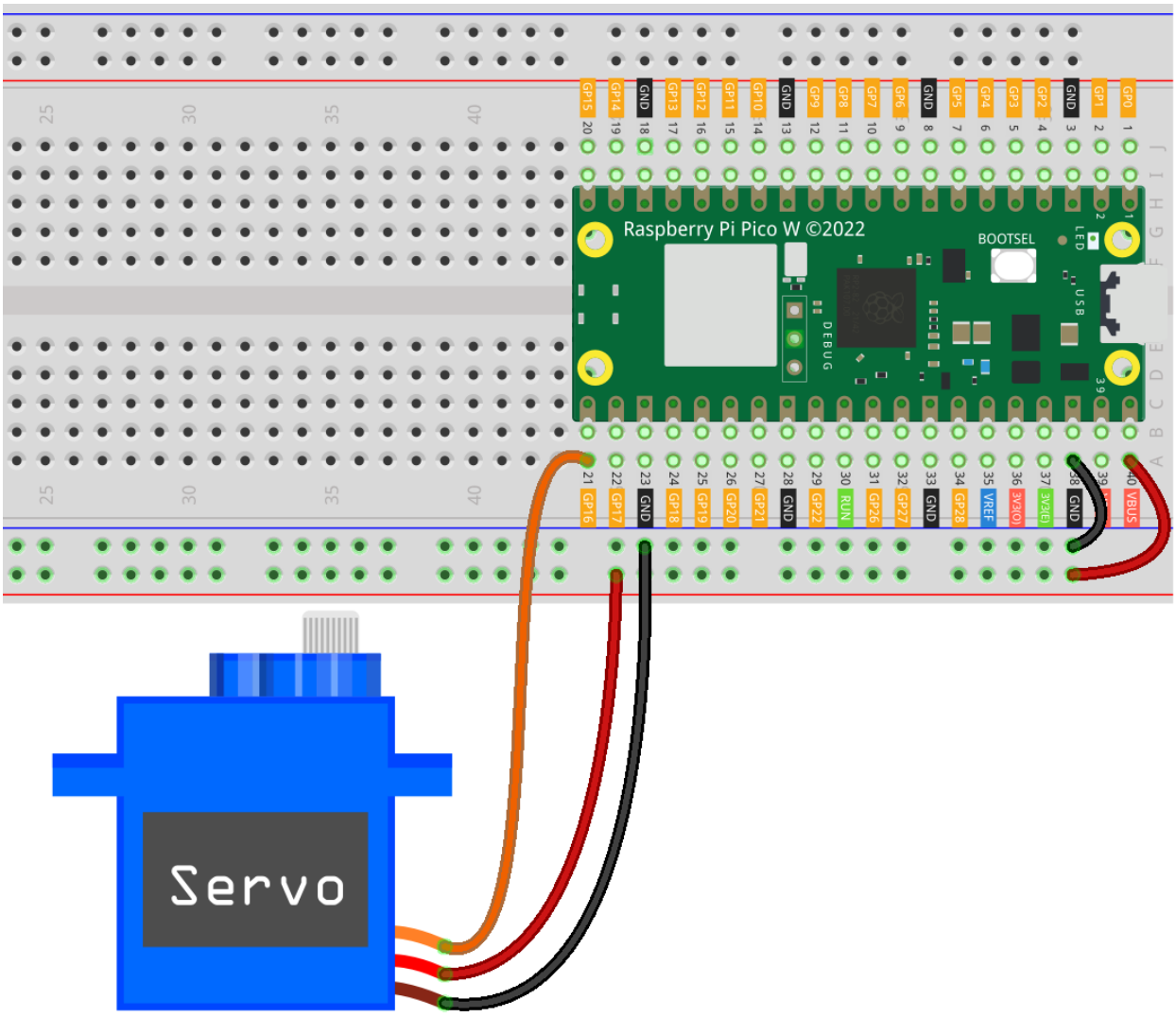
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
Servo Motor (SG90)	
Breadboard	

Wiring



Code

```
import machine
import time

# Initialize PWM on pin 16 for servo control
servo = machine.PWM(machine.Pin(16))
servo.freq(50)  # Set PWM frequency to 50Hz, common for servo motors


def interval_mapping(x, in_min, in_max, out_min, out_max):
    """
    Maps a value from one range to another.
    This function is useful for converting servo angle to pulse width.
    """
```

(continues on next page)

(continued from previous page)

```
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min


def servo_write(pin, angle):
    """
    Moves the servo to a specific angle.
    The angle is converted to a suitable duty cycle for the PWM signal.
    """
    pulse_width = interval_mapping(
        angle, 0, 180, 0.5, 2.5
    )  # Map angle to pulse width in ms
    duty = int(
        interval_mapping(pulse_width, 0, 20, 0, 65535)
    )  # Map pulse width to duty cycle
    pin.duty_u16(duty)  # Set PWM duty cycle


# Main loop to continuously move the servo
while True:
    # Sweep the servo from 0 to 180 degrees
    for angle in range(180):
        servo_write(servo, angle)
        time.sleep_ms(20)  # Short delay for smooth movement

    # Sweep the servo back from 180 to 0 degrees
    for angle in range(180, -1, -1):
        servo_write(servo, angle)
        time.sleep_ms(20)  # Short delay for smooth movement
```

Code Analysis

1. Importing Modules and Initializing Servo:

   The `machine` module is crucial for accessing the PWM functionality needed to control the servo, and `time` is used for implementing delays. The servo is initialized on pin 16 of the Raspberry Pi Pico W, setting its frequency to 50Hz, a typical value for servo control.

   ```
   import machine
   import time
   servo = machine.PWM(machine.Pin(16))
   servo.freq(50)
   ```

2. Mapping and Servo Control Functions:

   The `interval_mapping` function translates the desired servo angle into a PWM pulse width. The `servo_write` function then converts this pulse width into a duty cycle, which is used to set the servo's position. These functions are central to converting the angular position into an appropriate PWM signal.

   Please refer to *Work Pulse* for information about the work pulse of the servo.

   ```
   def interval_mapping(x, in_min, in_max, out_min, out_max):
       return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
   ```

(continues on next page)

(continued from previous page)

```
def servo_write(pin, angle):
    pulse_width = interval_mapping(angle, 0, 180, 0.5, 2.5)
    duty = int(interval_mapping(pulse_width, 0, 20, 0, 65535))
    pin.duty_u16(duty)
```

3. Main Loop for Continuous Movement:

   The main loop is where the servo is controlled to sweep from 0 to 180 degrees and back. This is achieved by looping through the range of angles and calling `servo_write` for each angle, with a short delay to ensure smooth movement.

```
while True:
    for angle in range(180):
        servo_write(servo, angle)
        time.sleep_ms(20)
    for angle in range(180, -1, -1):
        servo_write(servo, angle)
        time.sleep_ms(20)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.5.35 Lesson 34: TT Motor

In this lesson, you will learn how to operate a TT motor using the Raspberry Pi Pico W and an L9110 motor control board. We'll guide you through the process of configuring two PWM (Pulse Width Modulation) pins to control the motor. You'll set up the motor to run for 5 seconds and then turn off. This practical exercise offers a valuable opportunity to delve into motor control mechanisms and PWM signals, crucial in microcontroller programming.

Required Components

In this project, we need the following components.

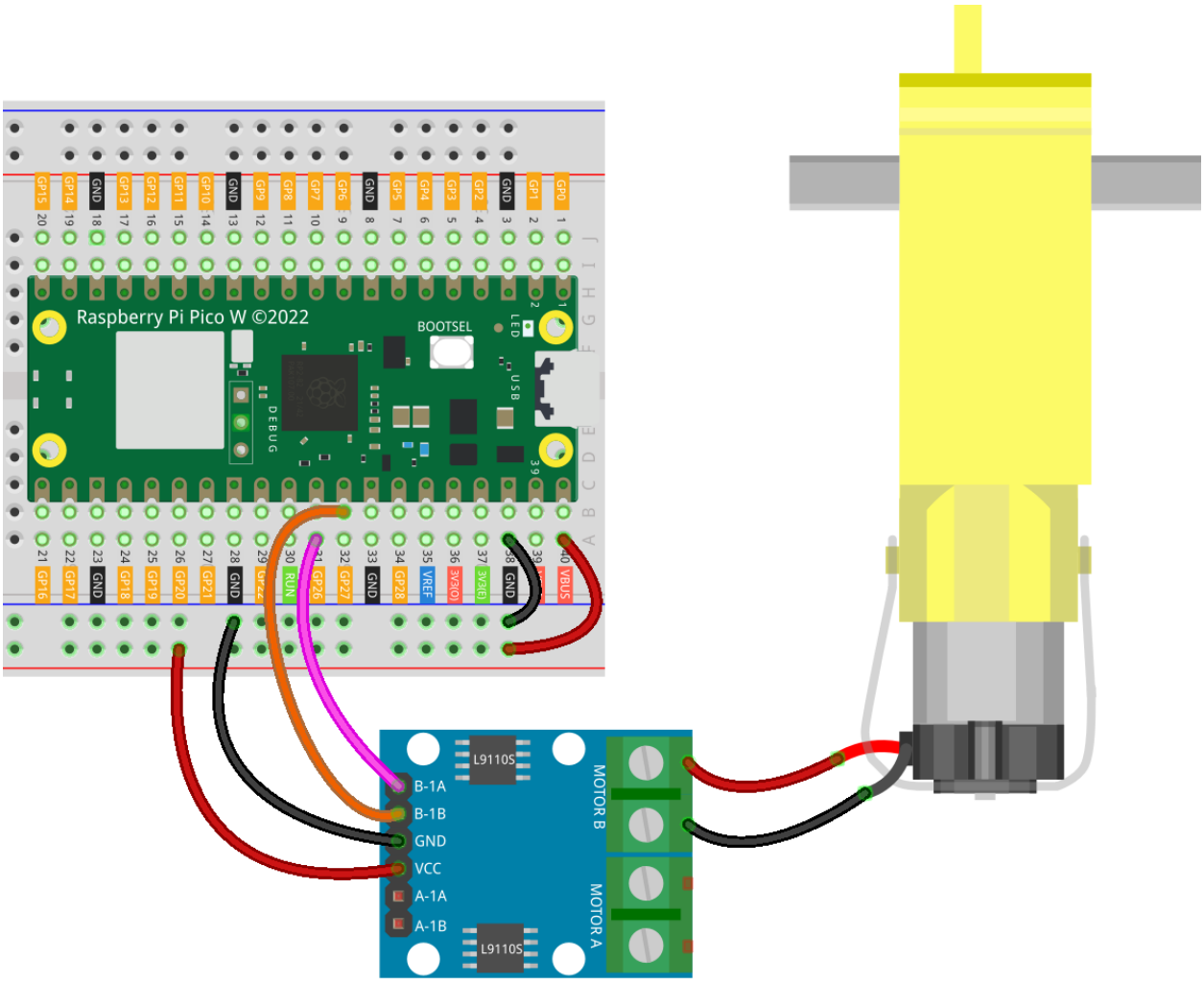
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi Pico W	-
TT Motor	-
L9110 Motor Driver Module	-
Breadboard	

Wiring



Code

```
from machine import Pin, PWM
import time

motor_a = PWM(Pin(26), freq=1000)
motor_b = PWM(Pin(27), freq=1000)

# turn on motor
motor_a.duty_u16(0)
motor_b.duty_u16(65535)  # speed(0-65535)

time.sleep(5)

# turn off motor
motor_a.duty_u16(0)
motor_b.duty_u16(0)
```

Code Analysis

1. Importing Libraries

   - The `machine` module is imported to interact with the GPIO pins and PWM functionalities of the Raspberry Pi Pico W.
   - The `time` module is used for creating delays in the code.

   ```
   from machine import Pin, PWM
   import time
   ```

2. Initializing PWM Objects

   - Two PWM objects, `motor_a` and `motor_b`, are created. They correspond to GPIO pins 26 and 27, respectively.
   - The frequency for PWM is set to 1000 Hz, a common frequency for motor control.

   ```
   motor_a = PWM(Pin(26), freq=1000)
   motor_b = PWM(Pin(27), freq=1000)
   ```

3. Turning on the Motor

   - `motor_a.duty_u16(0)` sets the duty cycle of `motor_a` pin to 0, while `motor_b.duty_u16(65535)` sets the duty cycle of `motor_b` pin to 65535, running the motor at full speed. For more details, please refer to *the working principle of L9110*.
   - The motor runs for 5 seconds, controlled by `time.sleep(5)`.

   ```
   # turn on motor
   motor_a.duty_u16(0)
   motor_b.duty_u16(65535)  # speed(0-65535)
   time.sleep(5)
   ```

4. Turning off the Motor

   Both `motor_a` and `motor_b` are set to a duty cycle of 0, stopping the motor.

```
# turn off motor
motor_a.duty_u16(0)
motor_b.duty_u16(0)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6 For Raspberry Pi

Raspberry Pi users, please refer to the following tutorial. This tutorial is based on Raspberry Pi 5 as an example and has been verified on Raspberry Pi 4 and Raspberry Pi 5, but may also be applicable to other versions of Raspberry Pi (not guaranteed).

We will program the Raspberry Pi using the `GPIO Zero` library in Python, providing simple code examples for each component to help you get started quickly.

The `GPIO Zero` library is a remarkable tool in Python designed for interfacing with the GPIO (General Purpose Input/Output) pins on a Raspberry Pi. It offers a straightforward and intuitive API, making it exceptionally accessible for beginners in electronics and programming. The library simplifies tasks such as reading sensors, controlling motors, and activating LEDs, allowing for seamless interaction with a wide array of hardware components. Its user-friendly nature encourages experimentation and learning, making it a popular choice in educational environments and DIY projects.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.1 Getting Started with Raspberry Pi

In this chapter, we will learn how to get started with Raspberry Pi. The content covers installing the OS, setting up the Raspberry Pi network, and accessing the terminal.

Note: You can also find the full tutorial for setting up the Raspberry Pi on the Raspberry Pi Foundation official website at: .

Once your Raspberry Pi is set up, you can skip this part and move on to the next chapter.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

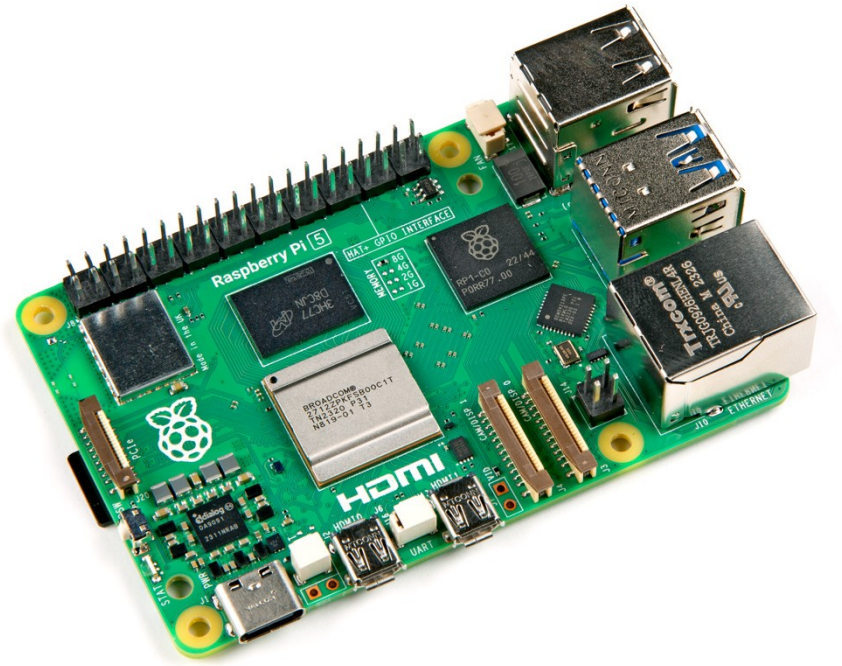
Ready to explore and create with us? Click [] and join today!

What Do We Need?

Required Components

Raspberry Pi

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV and used with a standard keyboard and mouse. It's a versatile device that allows people of all ages to explore computing and learn programming languages such as Scratch and Python.



Power Adapter

Model	Recommended power supply (voltage/current)
Raspberry Pi 5	5V/5A, 5V/3A limits peripherals to 600mA
Raspberry Pi 4 Model B	5V/3A
Raspberry Pi 3 (all models)	5V/2.5A

Micro SD Card

Your Raspberry Pi needs an Micro SD card to store all its files and the Raspberry Pi OS. You will need a micro SD card with a capacity of at least 8 GB.

Optional Components

Screen

To access the Raspberry Pi's desktop environment, you can connect it to a TV or computer monitor. If the screen includes speakers, audio will be output through them.

Mouse & Keyboard

When you use a screen, a USB keyboard and a USB mouse are also needed.

HDMI

The Raspberry Pi has HDMI (or Micro HDMI) output ports, which are compatible with the HDMI ports of most modern TVs and computer monitors. If your screen only comes with a DVI or VGA port, you will need to use the corresponding adapter cable.

Case

You can place the Raspberry Pi in a case to protect your device. On our official website, we offer related products for sale; you can view or purchase Raspberry Pi cases .

Sound or Earphone

Most Raspberry Pi models come with a 3.5mm audio port, which can be used when your screen doesn't have built-in speakers or isn't in use. However, it's important to note that the latest Raspberry Pi 5 doesn't come with a 3.5mm audio port.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

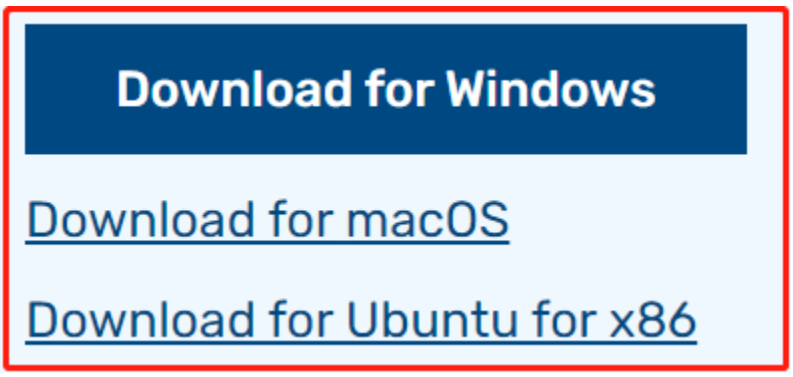
Ready to explore and create with us? Click [] and join today!

Write Raspberry Pi OS to SD Card

Step 1

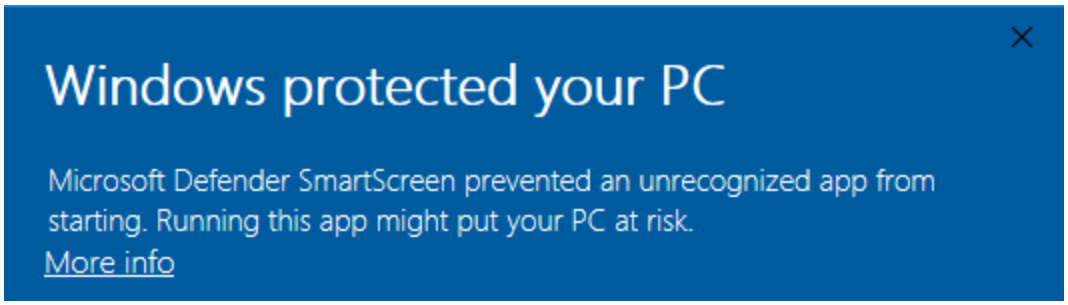
The Raspberry Pi team offers a user-friendly graphical SD card writing tool compatible with Mac OS, Ubuntu 18.04, and Windows. This is the most convenient option for most users, as it automatically downloads and installs the OS image to the SD card.

Visit the download page: https://www.raspberrypi.org/software/. Choose the **Raspberry Pi Imager** for your operating system. Once downloaded, open it to begin the installation.



Step 2

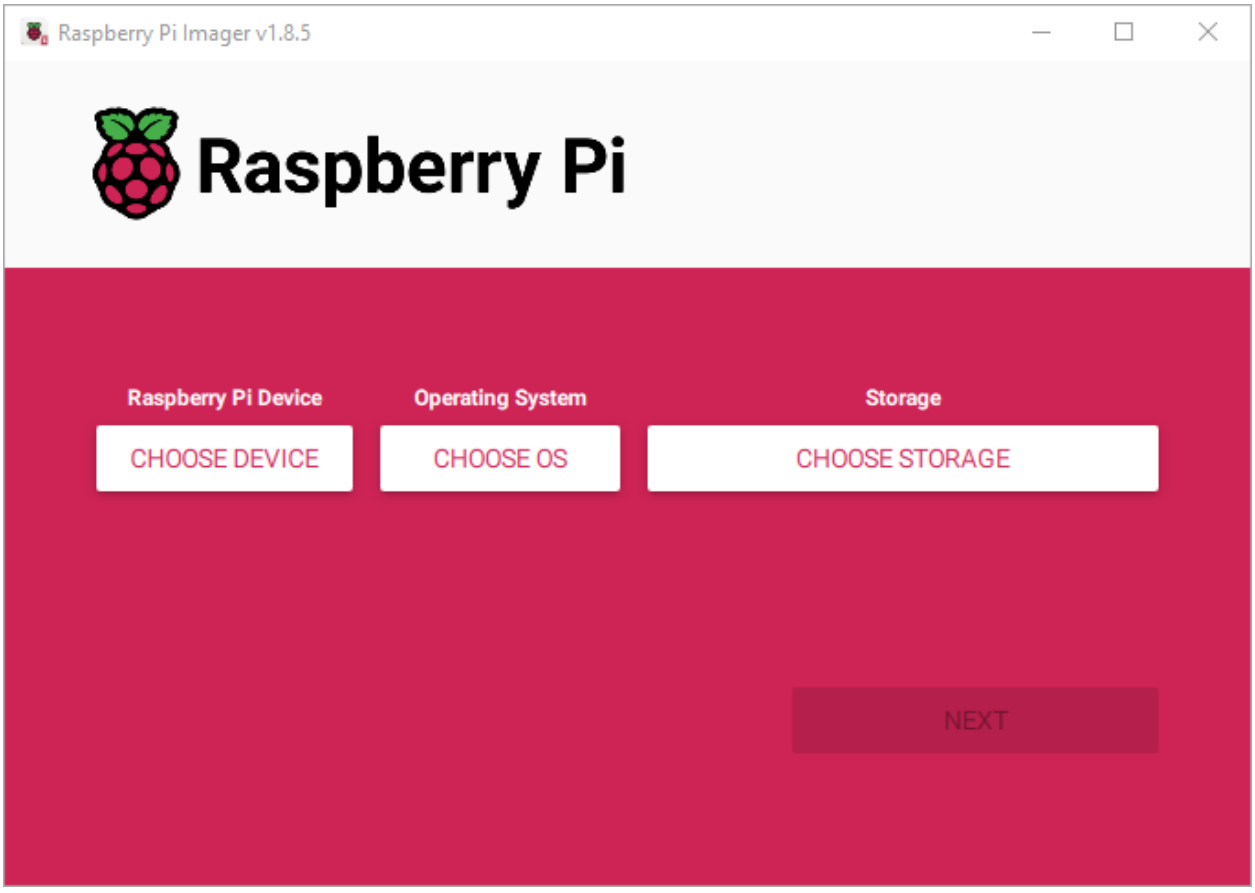
Upon launching the installer, your OS might prompt a security warning. For instance, Windows may show this message:



If you see this warning, click on **More info** and then choose **Run anyway**. Continue by following the instructions on your screen to complete the installation of the Raspberry Pi Imager.

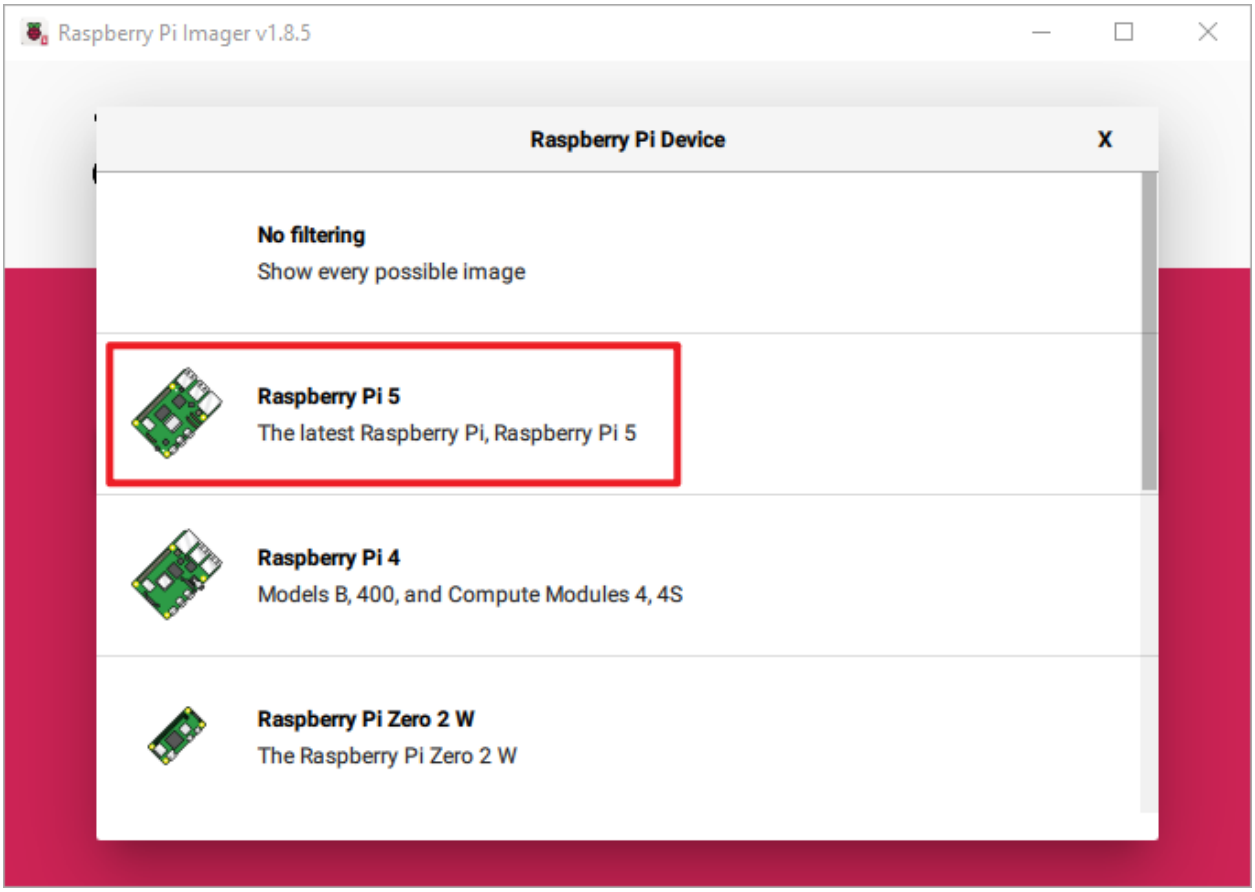
Step 3

After installing the Imager, open the application by clicking the **Raspberry Pi Imager** icon or executing `rpi-imager`.



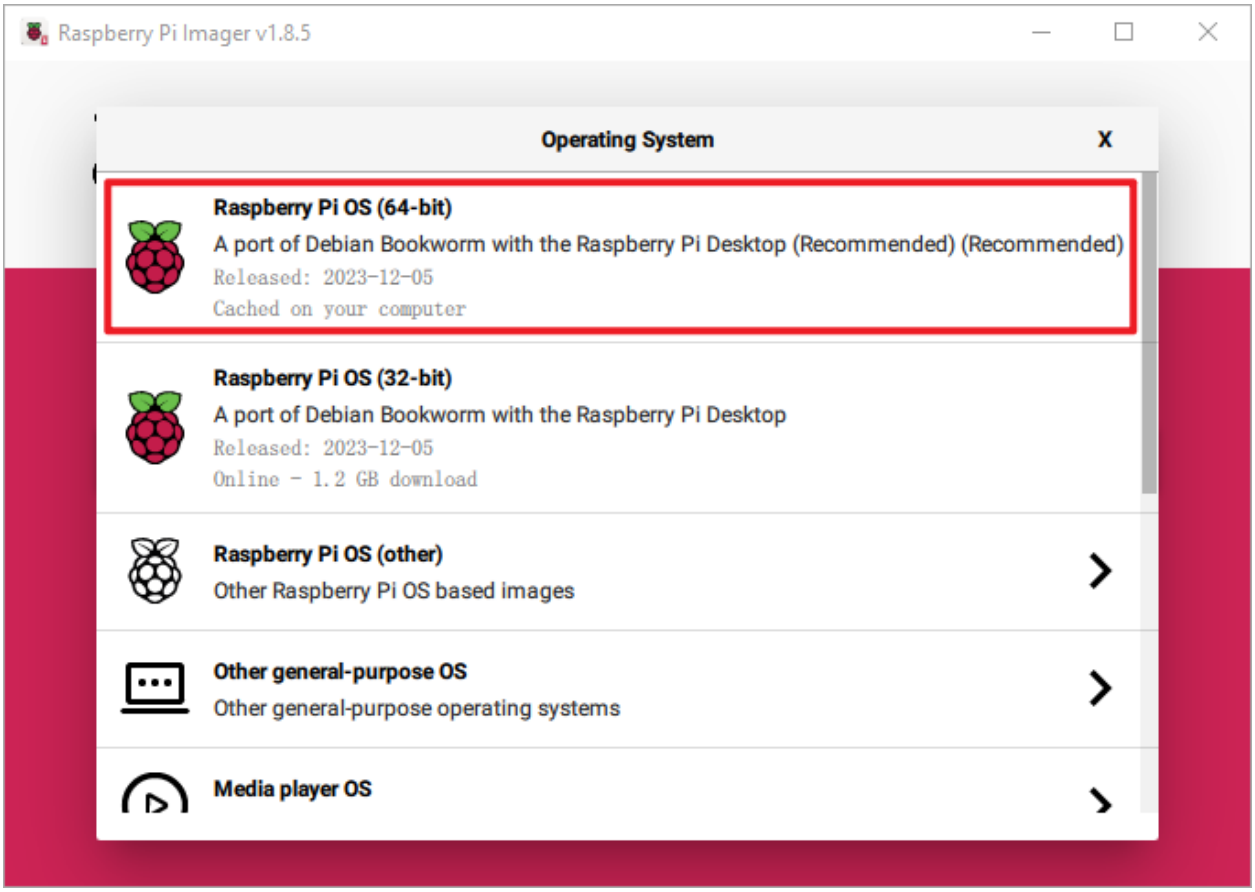
Step 4

Click **Choose device** and select your Raspberry Pi model from the list.



Step 5

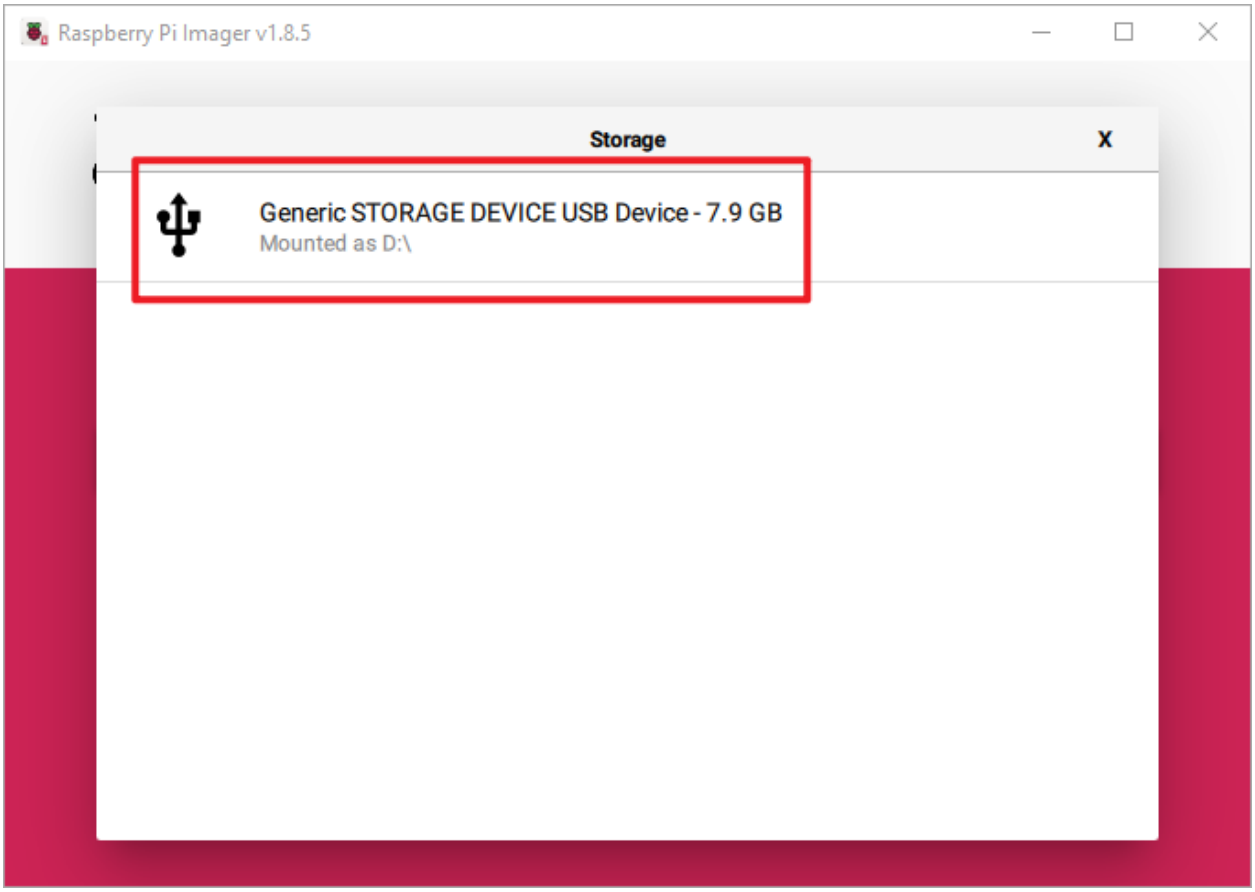
Next, click **Choose OS** and pick an operating system to install.



Step 6

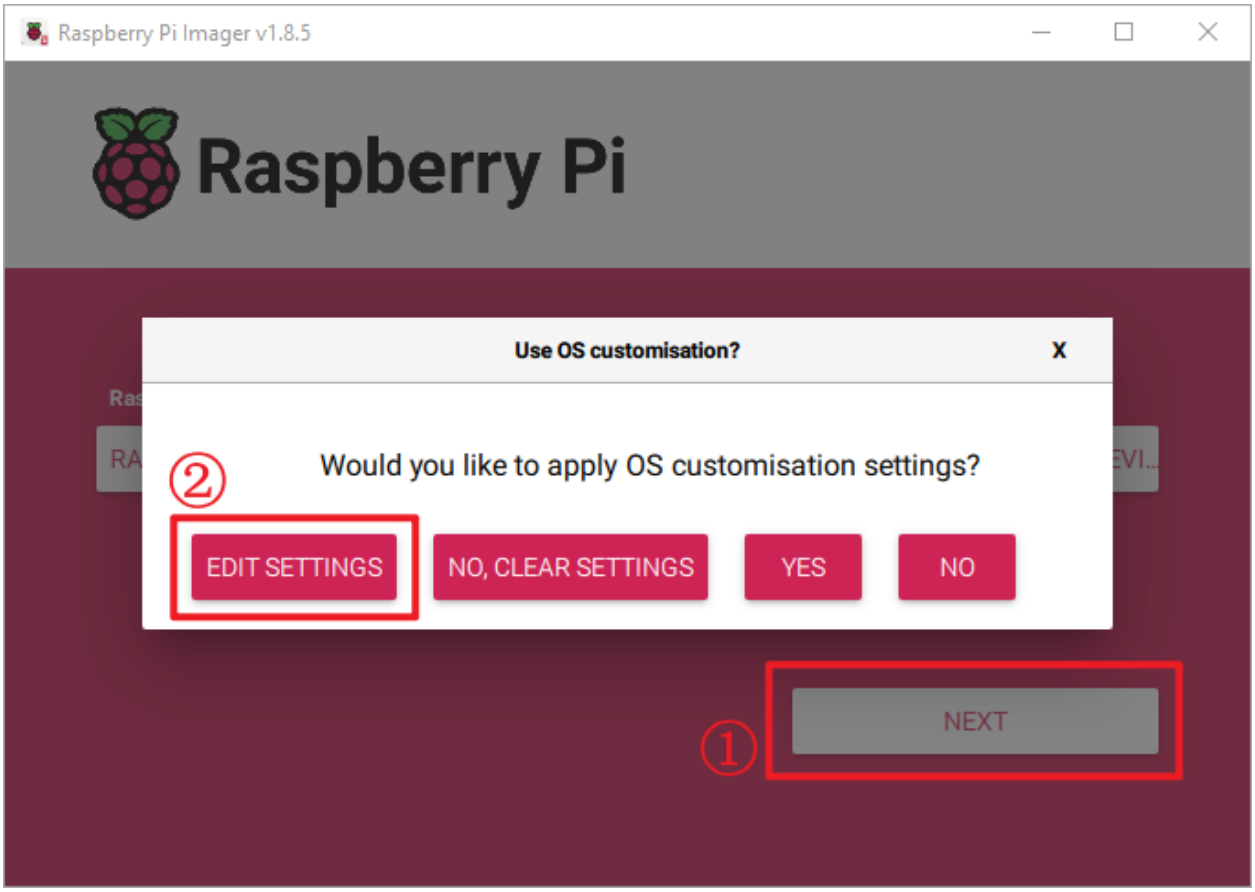
Insert your preferred storage medium, such as a microSD card, into an external or integrated SD card reader. Next, click "Choose Storage" and select your device.

Note: **Ensure you select the correct storage device when multiple devices are connected**; they can often be distinguished by their capacity. If uncertain, disconnect the other devices. **Be aware that installing the system on the chosen storage device will erase all data on it.**



Step 7

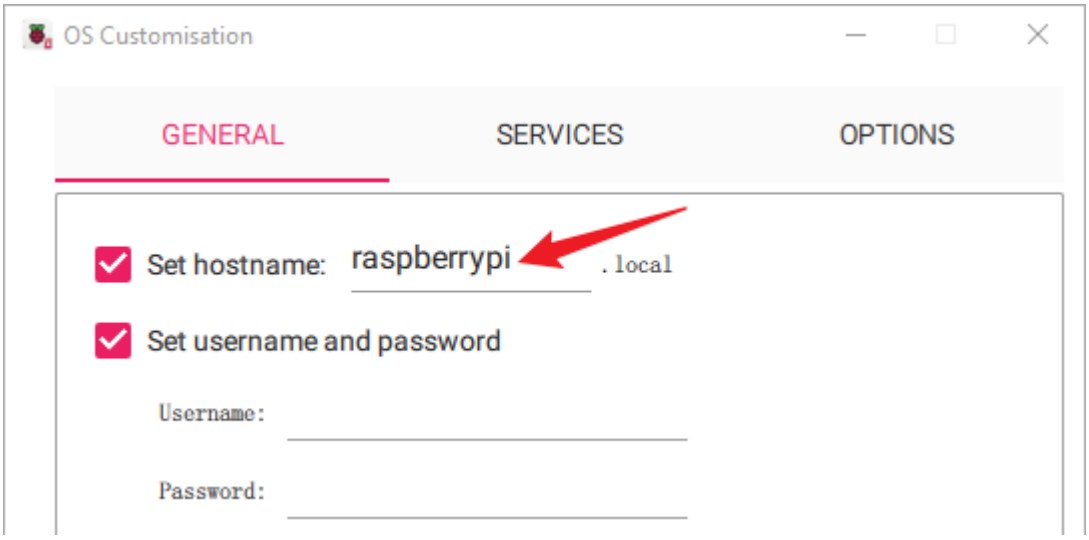
Press the **NEXT** button and choose **EDIT SETTINGS** to access the OS Customization page.



Step 8

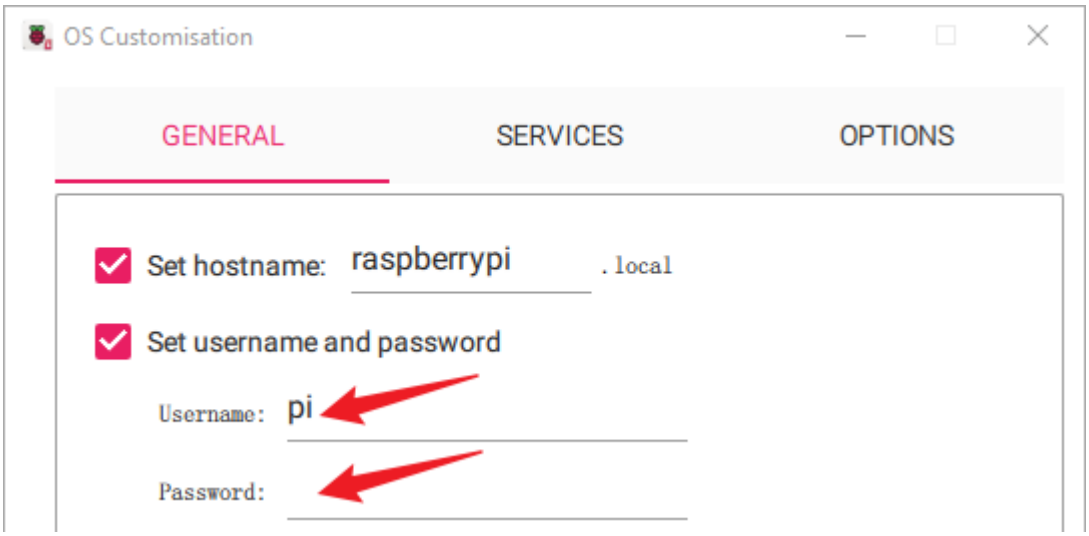
Set the **hostname**.

Note: The hostname option defines the hostname your Raspberry Pi broadcasts to the network using mDNS. By connecting your Raspberry Pi to the network, it allows other devices to interact with it using `<hostname>.local` or `<hostname>.lan`.



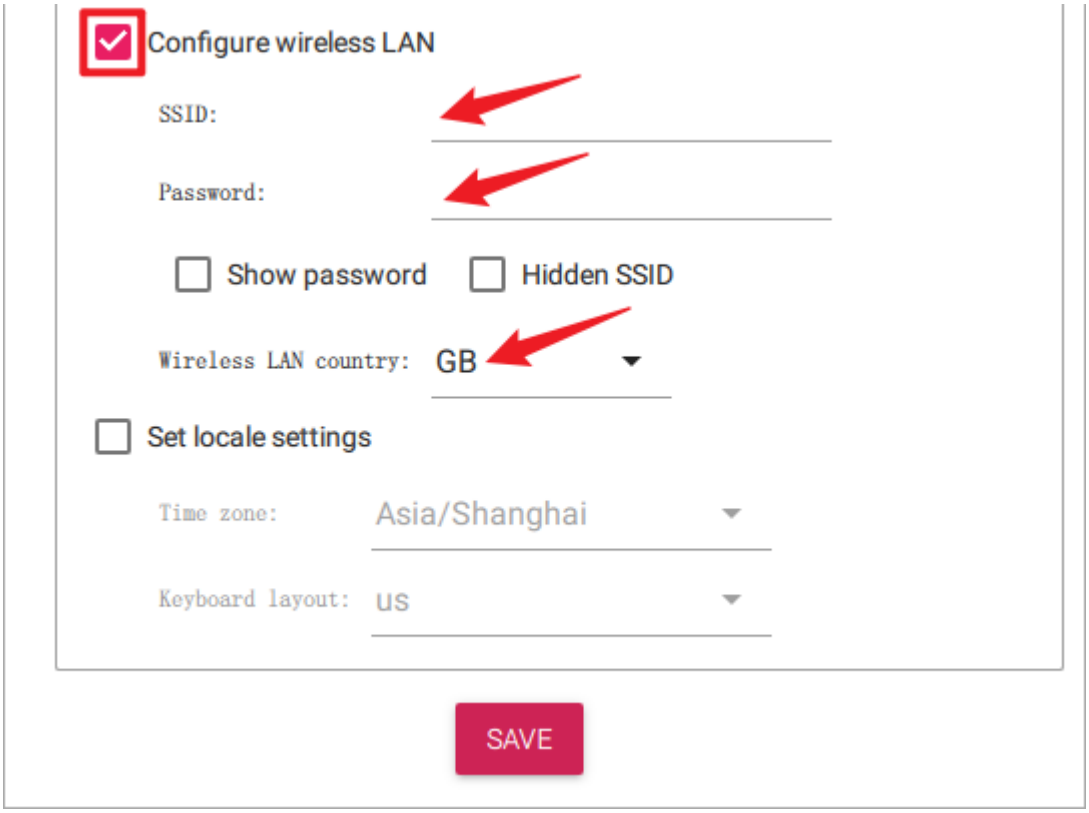
Set the **username** and **password** for the Raspberry Pi's administrator account.

Note: The Raspberry Pi does not come with a default password, making it crucial to set one. Additionally, you have the option to personalize the username.



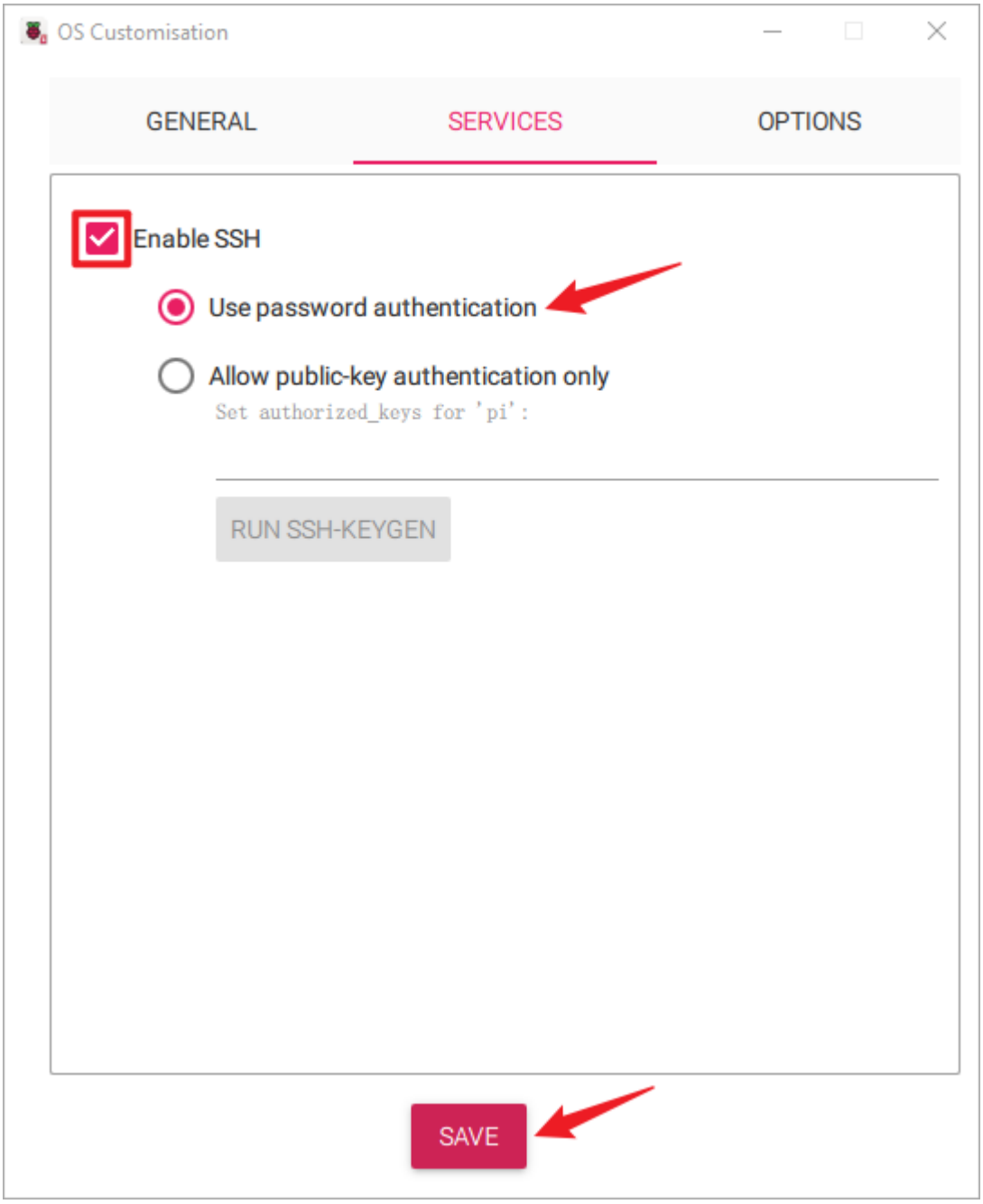
Configure wireless LAN by entering your network's **SSID** and **password**.

Note: Configure the "Wireless LAN country" using your nation's two-letter .



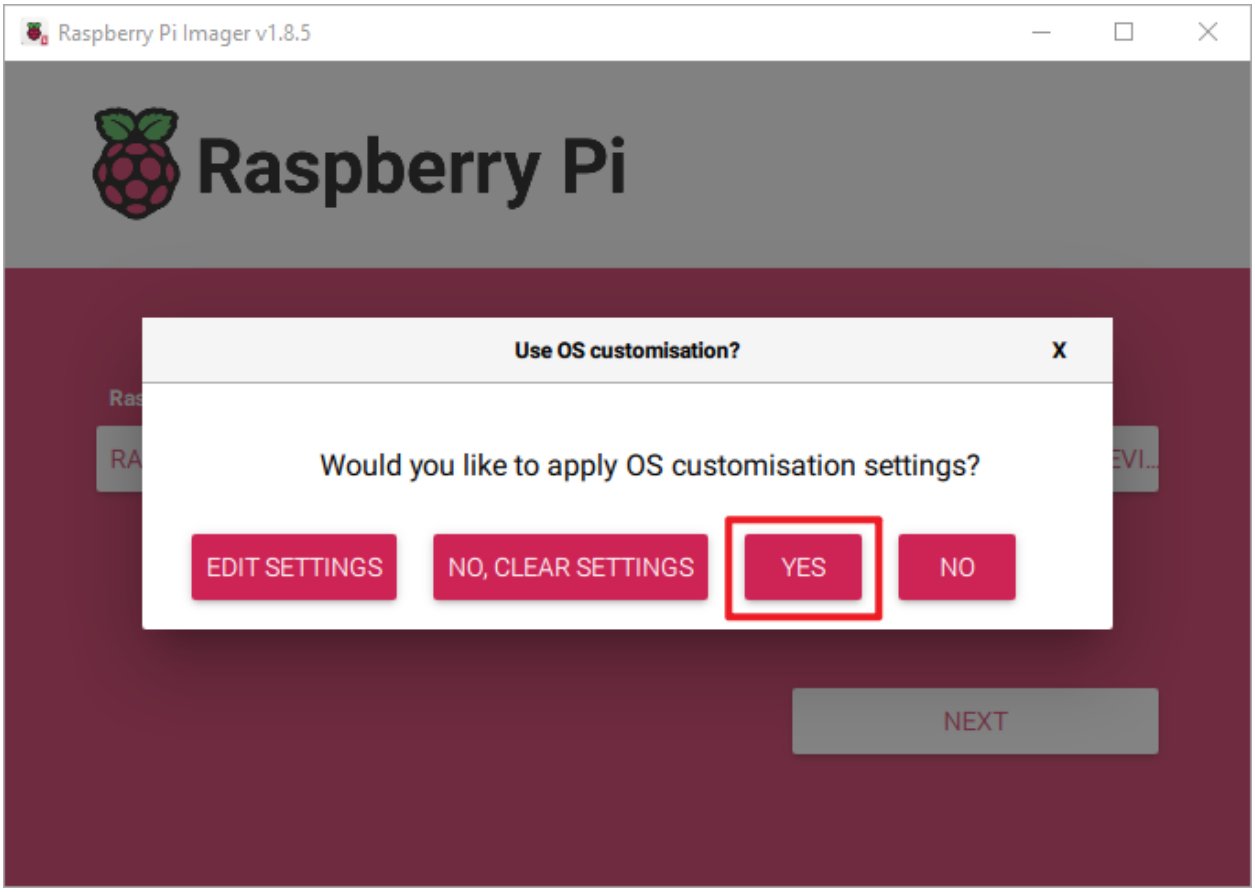
Step 9

Navigate to the **SERVICES** page, choose **Enable SSH option** to turn on SSH, and select "Use password authentication" (recommended for beginners). Click **Save** to apply your changes.



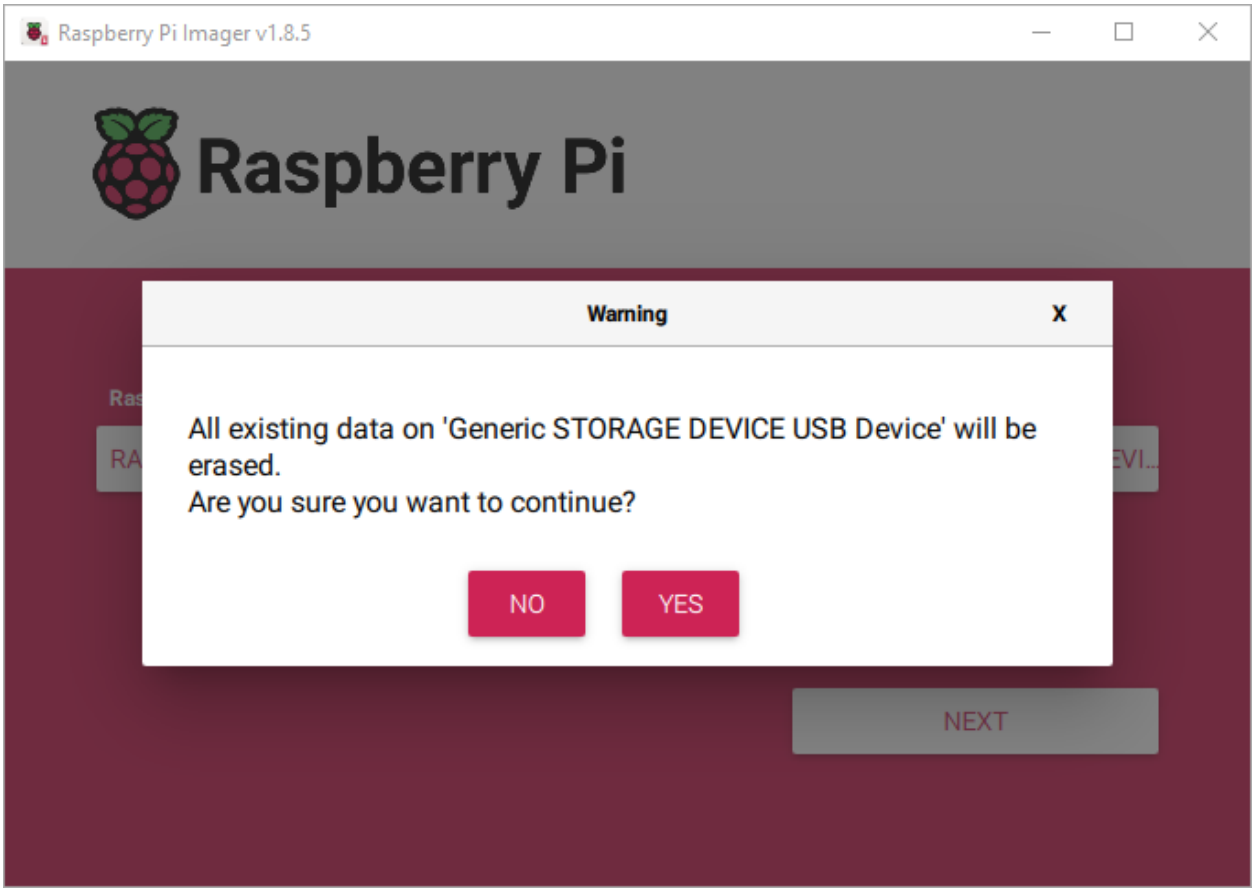
Step 10

Click the **Yes** button.



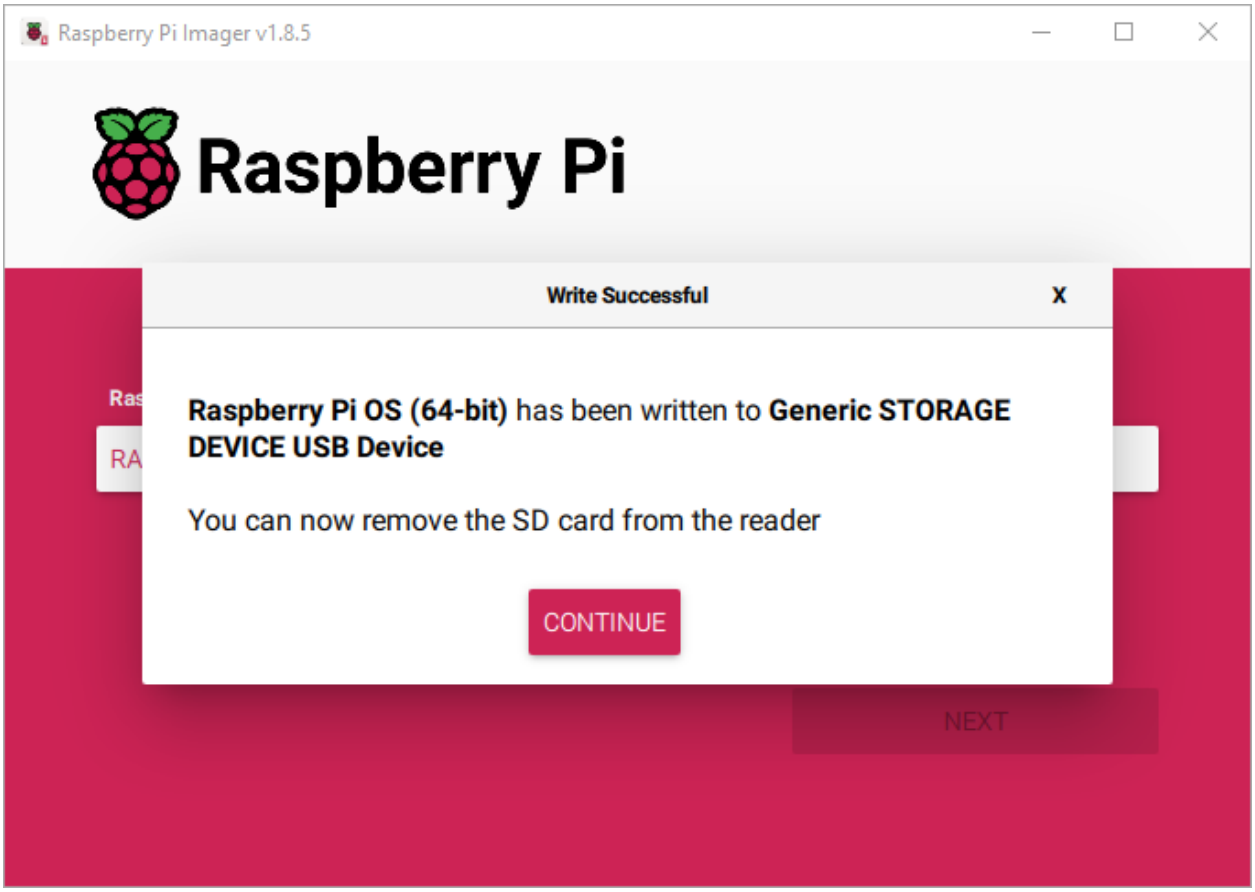
Step 11

If your SD card contains files, consider backing them up to avoid permanent loss. If no backup is needed, click **Yes**.



Step 12

The window below will appear once the writing process is complete. Writing process takes some time and varies based on the SD card's read-write performance; please be patient.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Set up Your Raspberry Pi

If You Have a Screen

If you have a screen, it will be easy for you to operate on the Raspberry Pi.

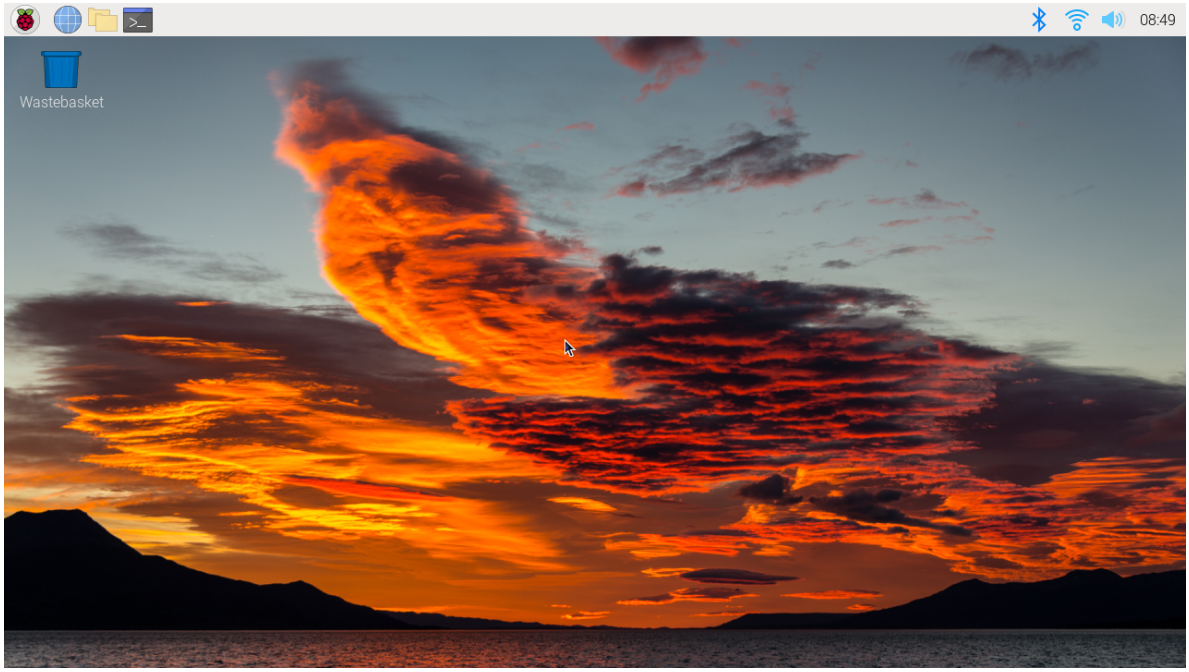
Required Components

- Any Raspberry Pi
- 1 * Power Adapter
- 1 * Micro SD card
- 1 * Screen Power Adapter
- 1 * HDMI cable
- 1 * Screen
- 1 * Mouse
- 1 * Keyboard

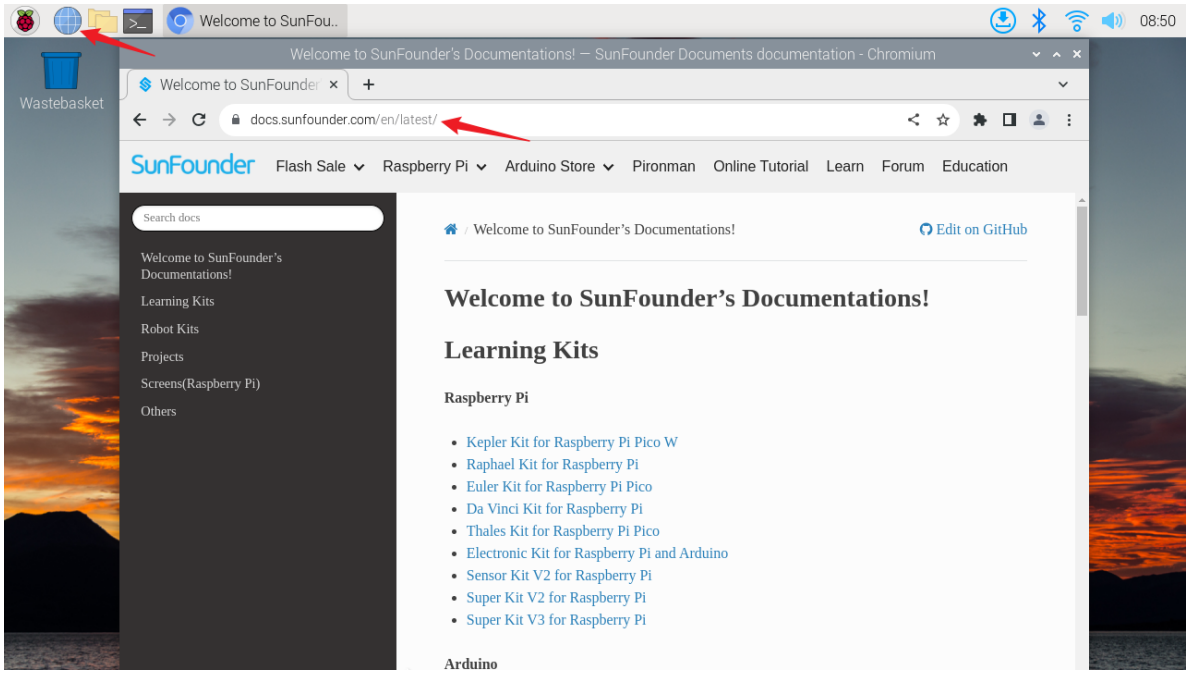
1. Insert the SD card you've set up with Raspberry Pi OS into the micro SD card slot on the underside of your Raspberry Pi.
2. Plug in the Mouse and Keyboard.
3. Connect the screen to Raspberry Pi's HDMI port and make sure your screen is plugged into a wall socket and switched on.

   Note: If you use a Raspberry Pi 4, you need to connect the screen to the HDMI0 (nearest the power in port).

4. Use the power adapter to power the Raspberry Pi. After a few seconds, the Raspberry Pi OS desktop will be displayed.



5. You can launch a web browser on your Raspberry Pi system and access this tutorial page. This makes it convenient to copy instructions and execute them in the Terminal.



If You Have No Screen

If you don't have a monitor, you can remotely log into your Raspberry Pi.

Required Components

- Any Raspberry Pi
- 1 * Power Adapter
- 1 * Micro SD card

You can use the SSH command to access the Raspberry Pi's Bash shell, which is Linux's default interface. The shell lets you perform most tasks with simple commands on Unix/Linux systems.

If you'd rather not use the command line for your Raspberry Pi, you can utilize the remote desktop feature to operate the Raspberry Pi's desktop environment without a dedicated screen.

See below for detailed tutorials for each system.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Mac OS X user

For Mac users, accessing the Raspberry Pi desktop directly via VNC is more convenient than from the command line. You can access it via Finder by entering the set account password after enabling VNC on the Raspberry Pi side.

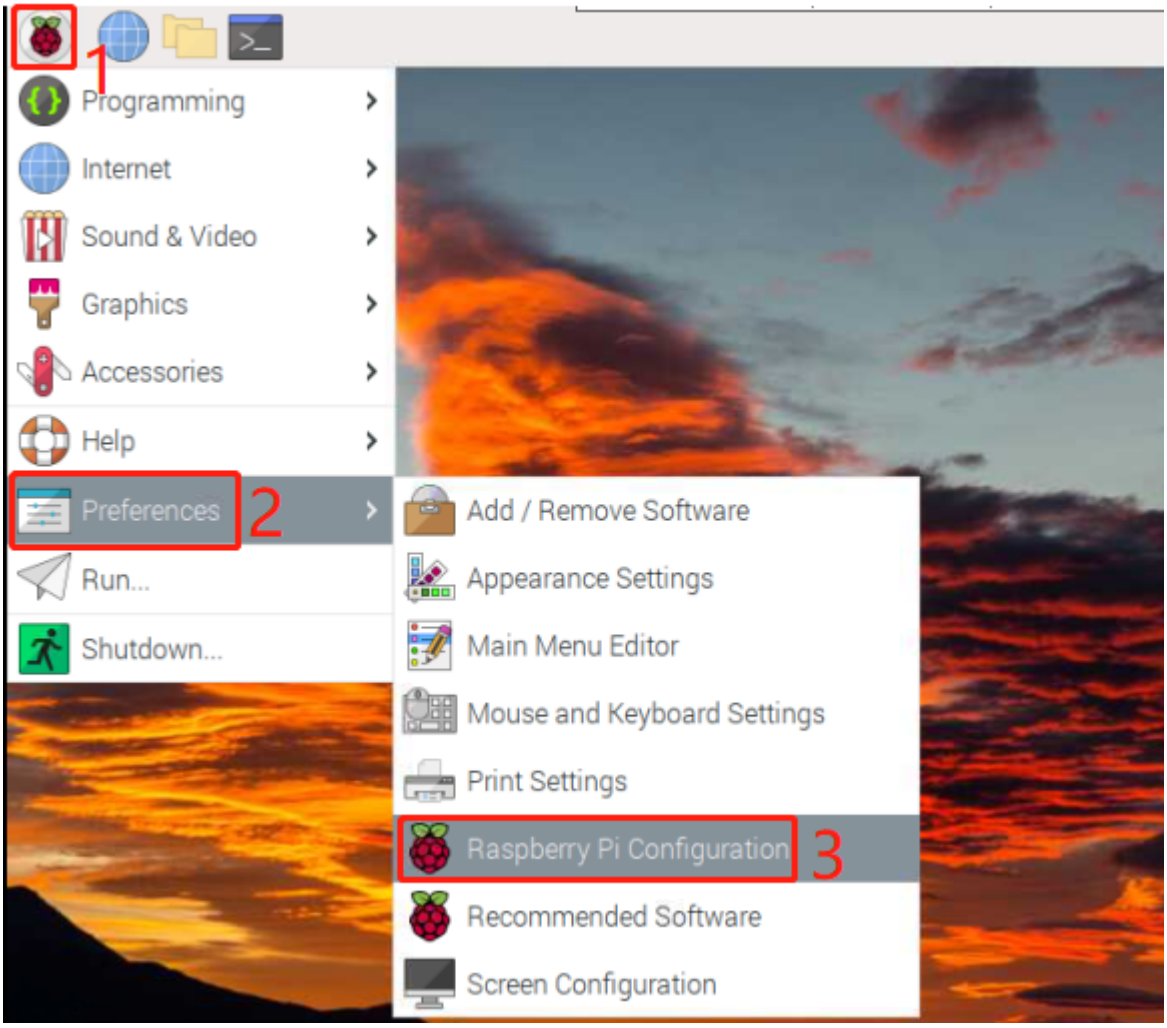
Note that this method does not encrypt communication between the Mac and Raspberry Pi. The communication will take place within your home or business network, so even if it's unprotected, it won't be an issue. However, if you are concerned about it, you can install a VNC application such as VNC® Viewer.

Alternatively it would be handy if you could use a temporary monitor (TV), mouse and keyboard to open the Raspberry Pi desktop directly to set up VNC. If not, it doesn't matter, you can also use the SSH command to open the Raspberry Pi's Bash shell and then using the command to set up the VNC.

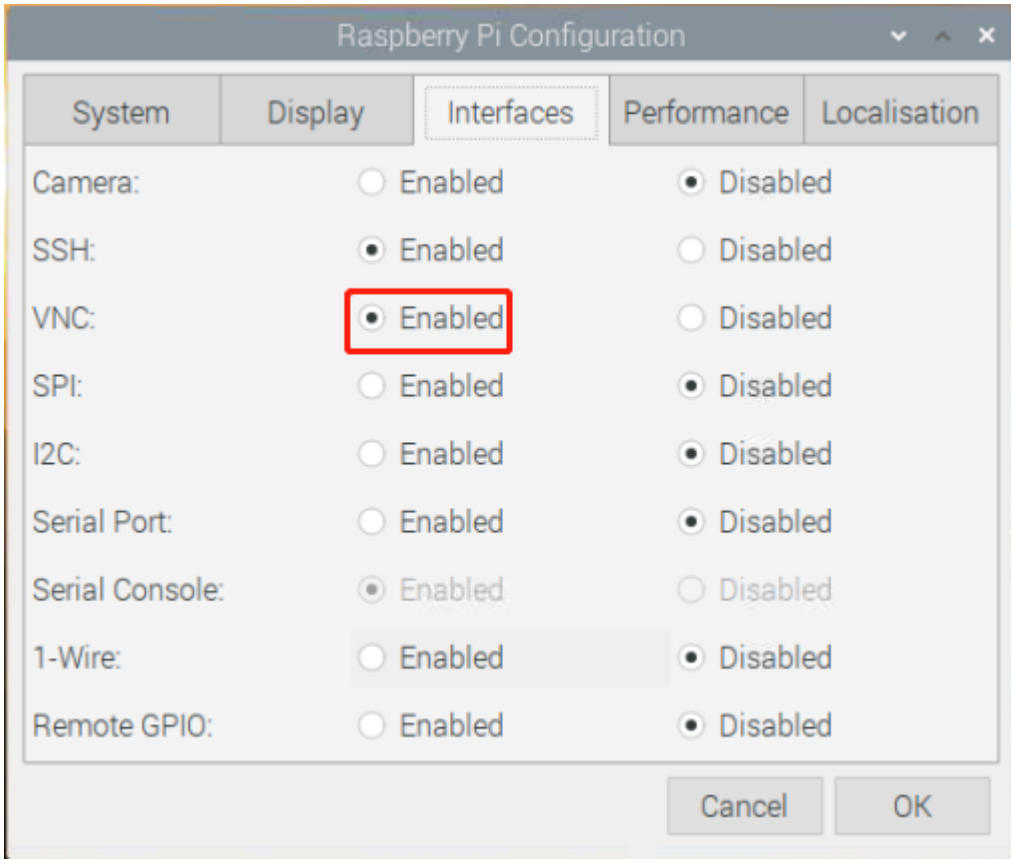
- *Have Temporarily Monitor (or TV)?*
- *Don't Have Temporarily Monitor (or TV)?*

Have Temporarily Monitor (or TV)?

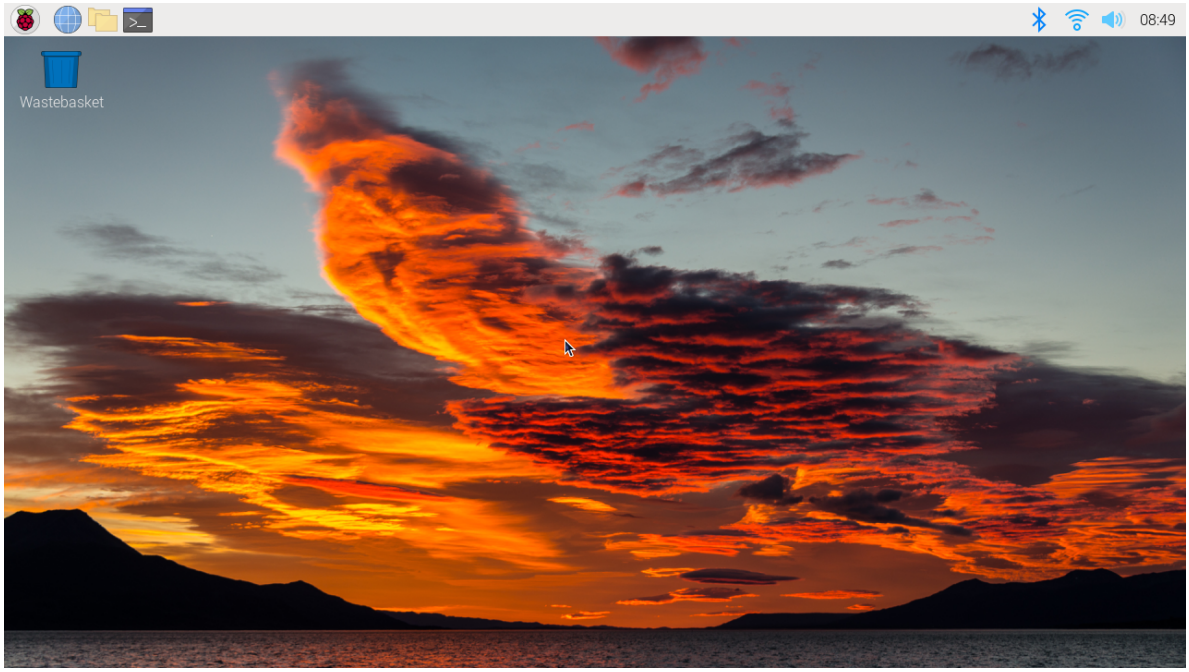
1. Connect a monitor (or TV), mouse and keyboard to the Raspberry Pi and power it on. Select the menu according to the numbers in the figure.



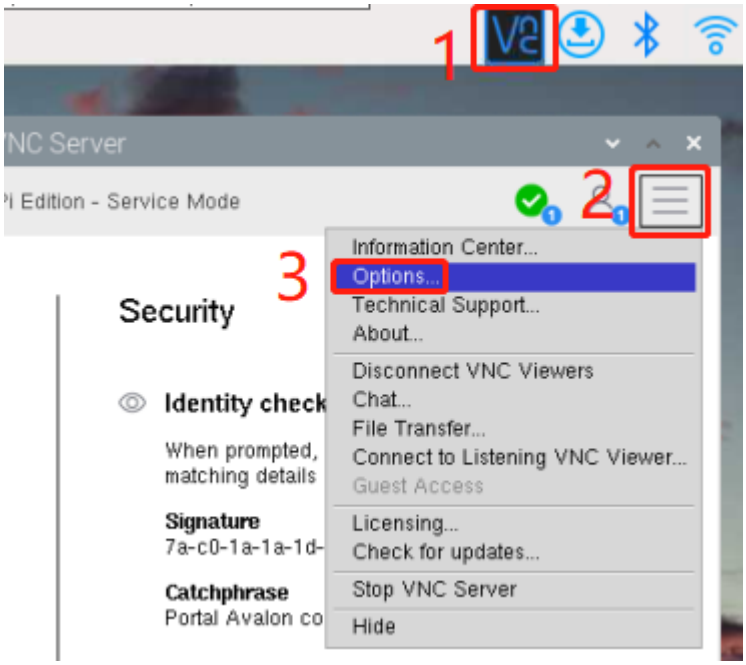
2. The following screen will be displayed. Set **VNC** to **Enabled** on the **Interfaces** tab, and click **OK**.



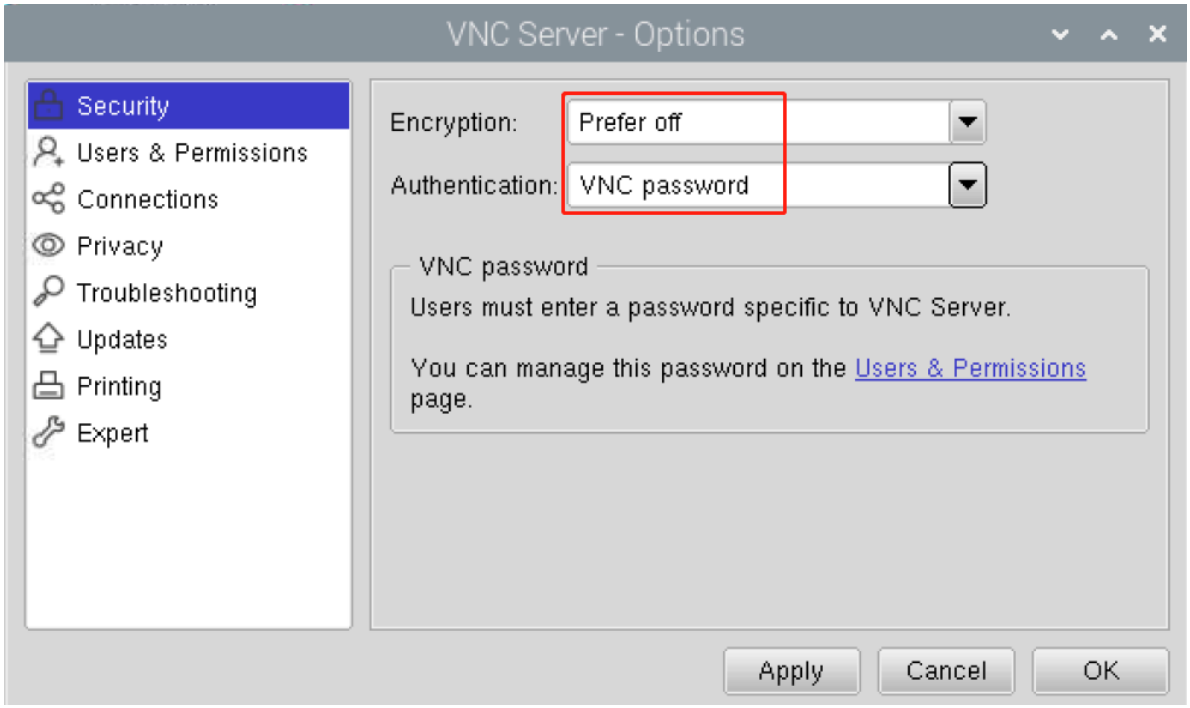
3. A VNC icon appears on the upper right of the screen and the VNC server starts.



4. Open the VNC server window by clicking on the **VNC** icon, then click on the **Menu** button in the top right corner and select **Options**.

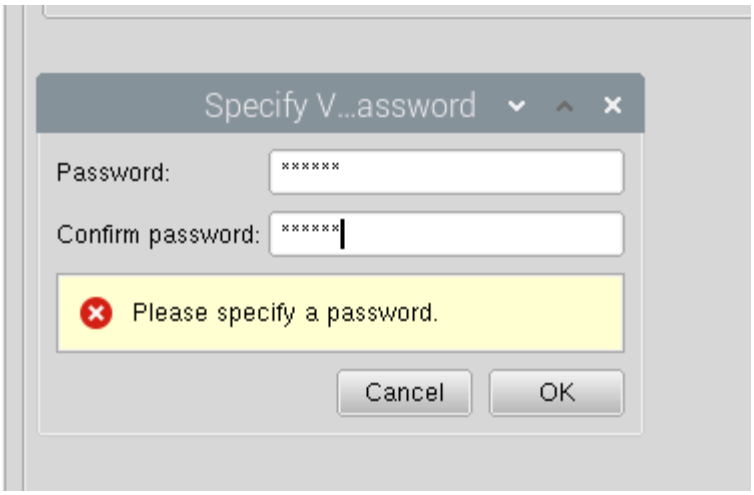


5. You will be presented with the following screen where you can change the options.



   Set **Encryption** to **Prefer off** and **Authentication** to **VNC password**.

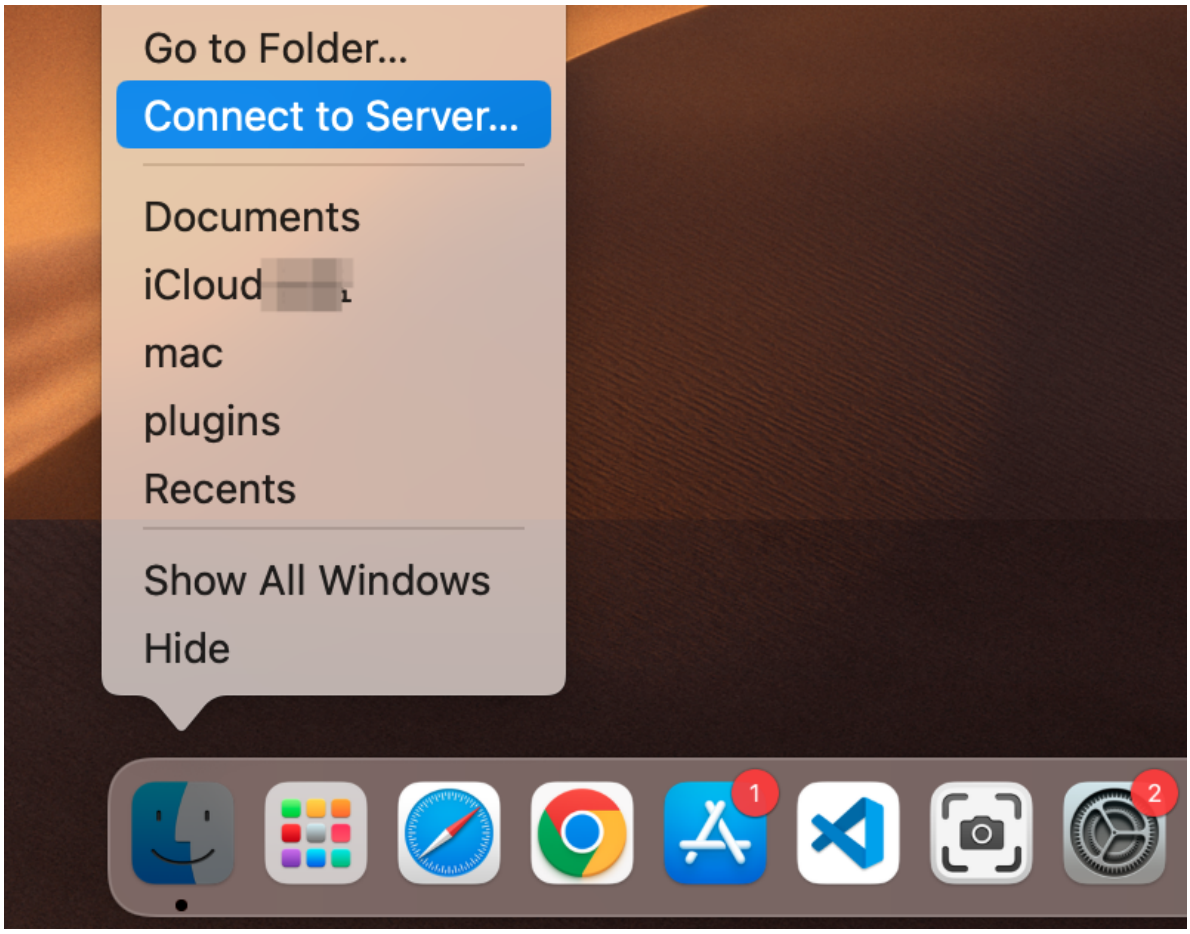
6. When you click the **OK** button, the password input screen is displayed. You can use the same password as the Raspberry pi password or a different password, so enter it and click **OK**.



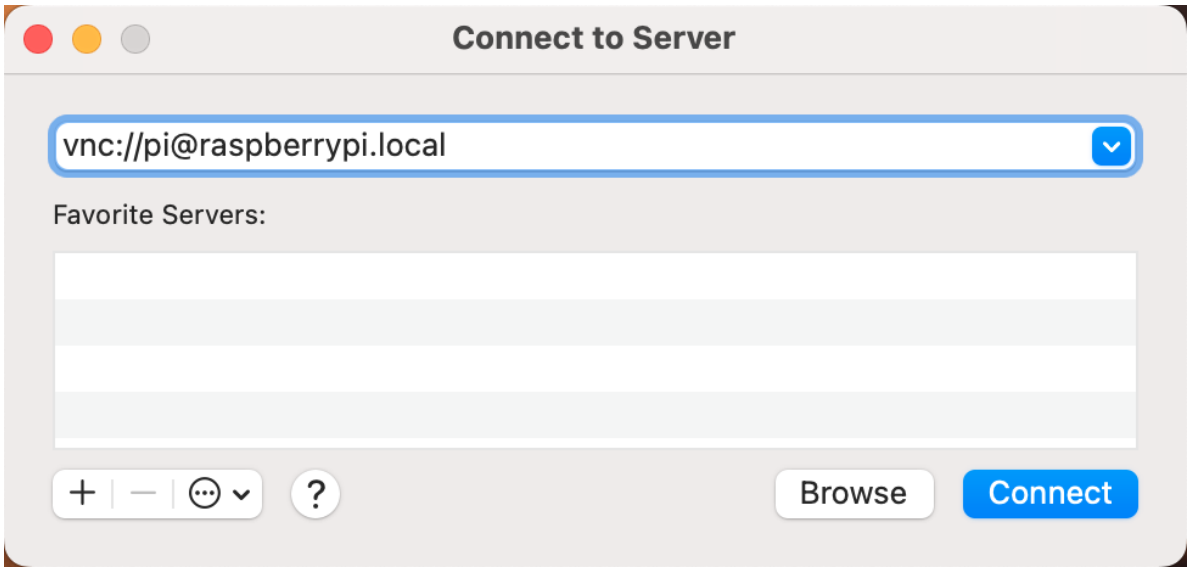
You are now ready to connect from your Mac. It's okay to disconnect the monitor.

From here, it will be the operation on the Mac side.

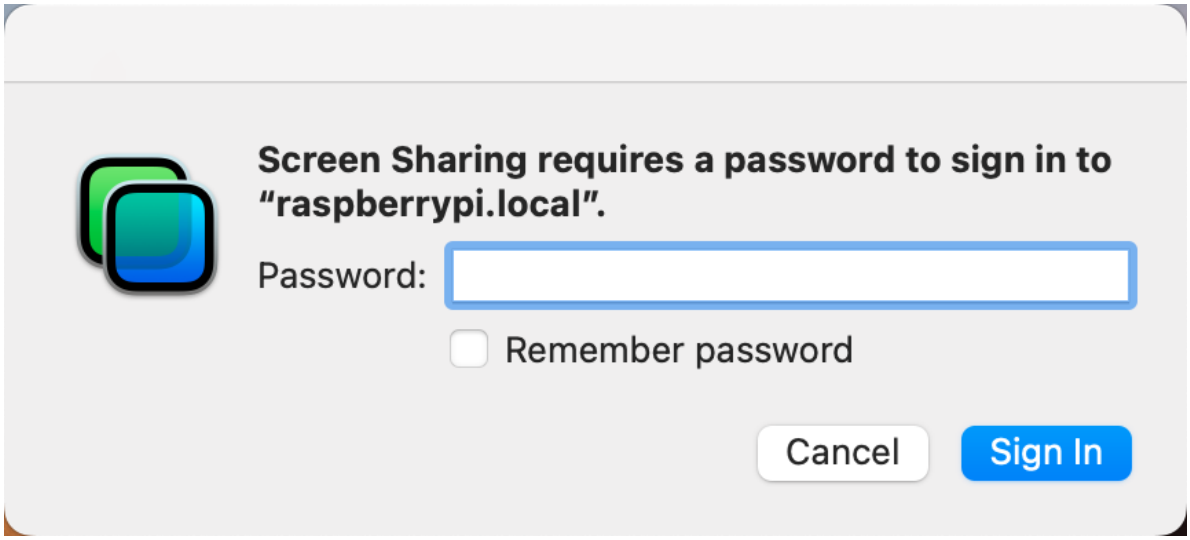
1. Now, select **Connect to Server** from the Finder's menu, which you can open by right-clicking.



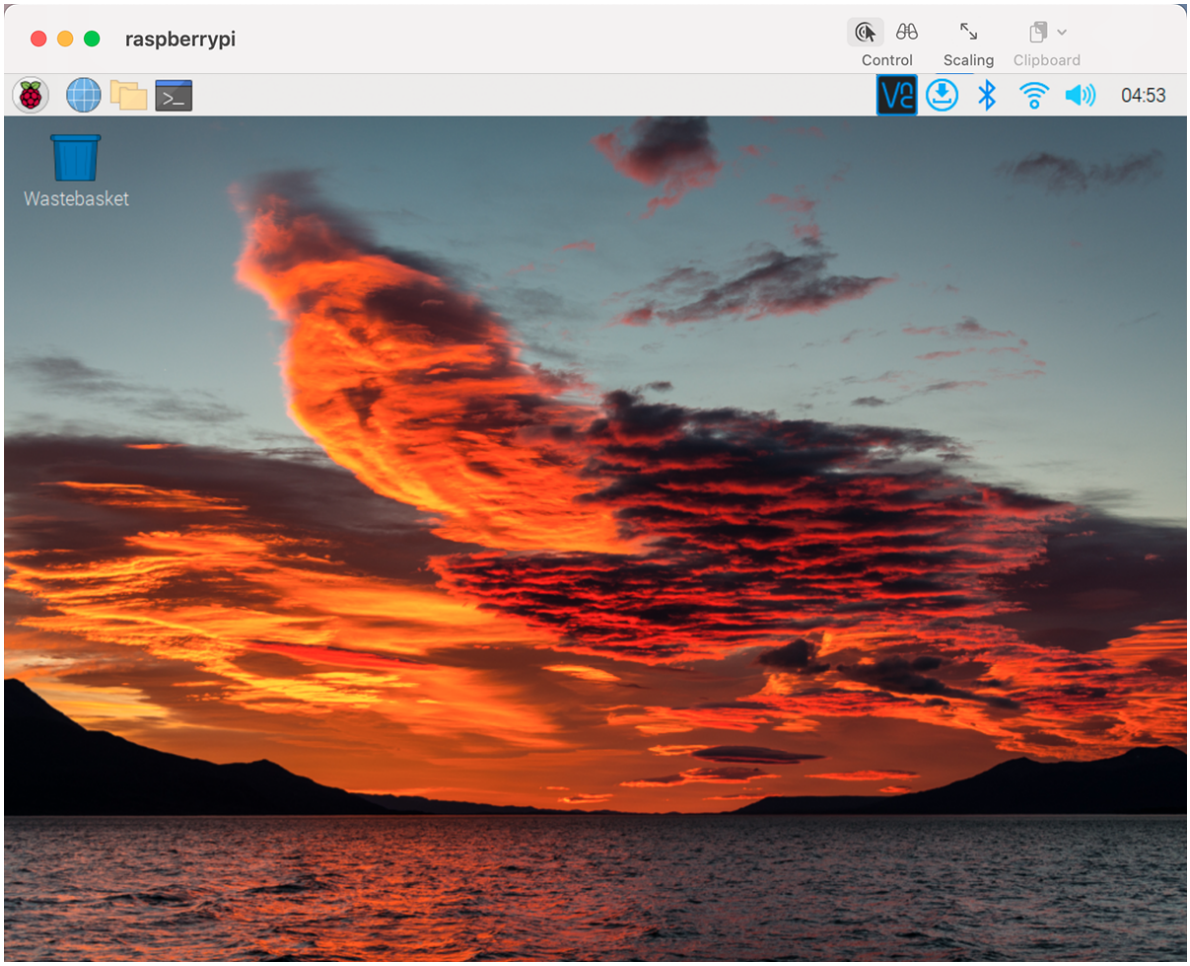
2. Type in `vnc://<username>@<hostname>.local` (or `vnc://<username>@<IP address>`). After entering, click **Connect**.



3. You will be asked for a password, so please enter it.



4. The desktop of the Raspberry pi will be displayed, and you will be able to operate it from the Mac as it is.

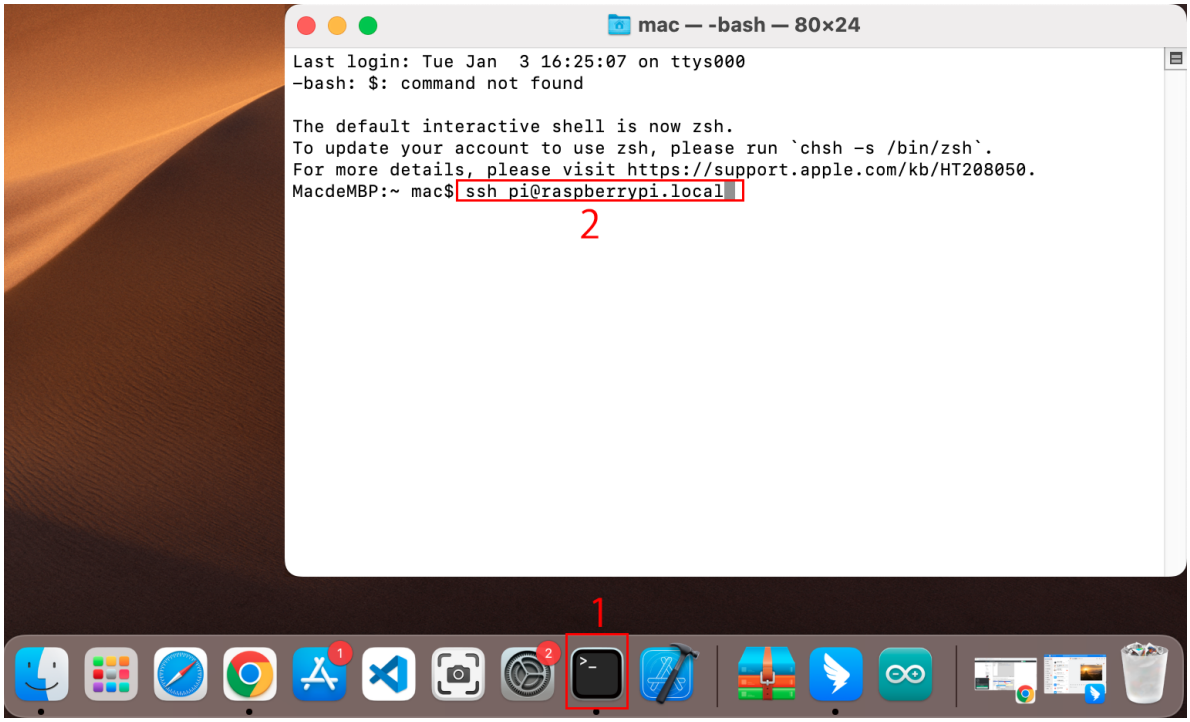


Don't Have Temporarily Monitor (or TV)?

- You can apply the SSH command to open the Raspberry Pi's Bash shell.
- Bash is the standard default shell for Linux.
- The shell itself is a command (instruction) when the user uses Unix/Linux.
- Most of what you need to do can be done through the shell.
- After setting up the Raspberry pi side, you can access the desktop of the Raspberry Pi using the **Finder** from the Mac.

1. Type `ssh <username>@<hostname>.local` to connect to the Raspberry Pi.

```
ssh pi@raspberrypi.local
```



2. The following message will be displayed only when you log in for the first time, so enter **yes**.

```
The authenticity of host 'raspberrypi.local␣
↪(2400:2410:2101:5800:635b:f0b6:2662:8cba)' can't be established.
ED25519 key fingerprint is SHA256:oo7x3ZSgAo032wD1tE8eW0fFM/kmewIvRwkBys6XRwg.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

3. Enter the password for the Raspberry pi. The password you enter will not be displayed, so be careful not to make a mistake.

```
pi@raspberrypi.local's password:
Linux raspberrypi 5.15.61-v8+ #1579 SMP PREEMPT Fri Aug 26 11:16:44 BST 2022 aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Sep 22 12:18:22 2022
pi@raspberrypi:~ $
```

4. Set up your Raspberry Pi so that you can log in via VNC from your Mac once you have successfully logged into it. The first step is to update your operating system by running the following commands.

```
sudo apt update
sudo apt upgrade
```

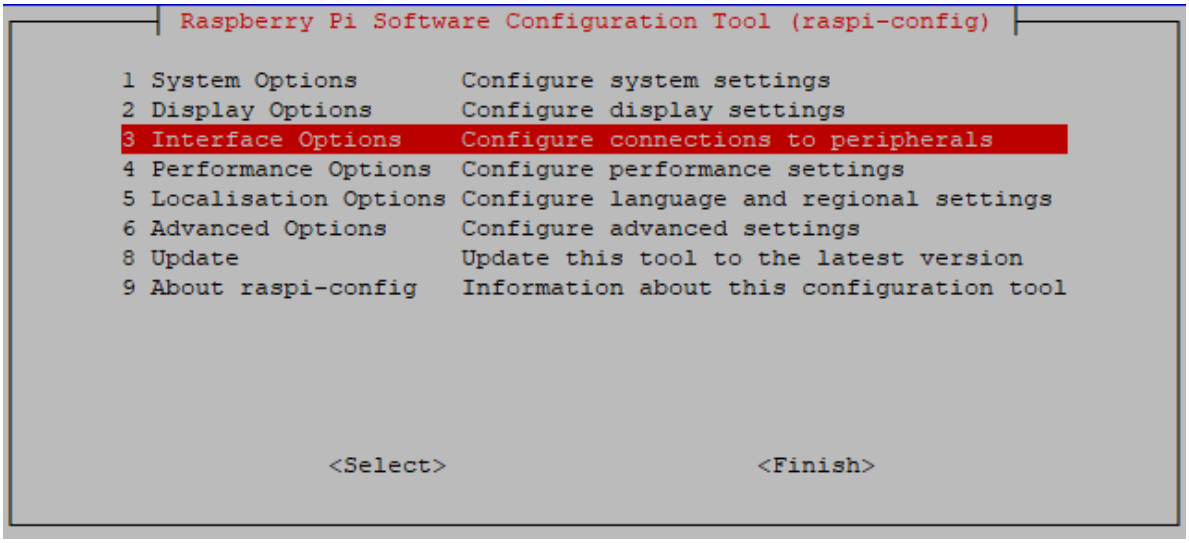
`Do you want to continue? [Y/n]`, Enter `Y` when prompted.

It may take some time for the update to finish. (It depends on the amount of updates at that time.)

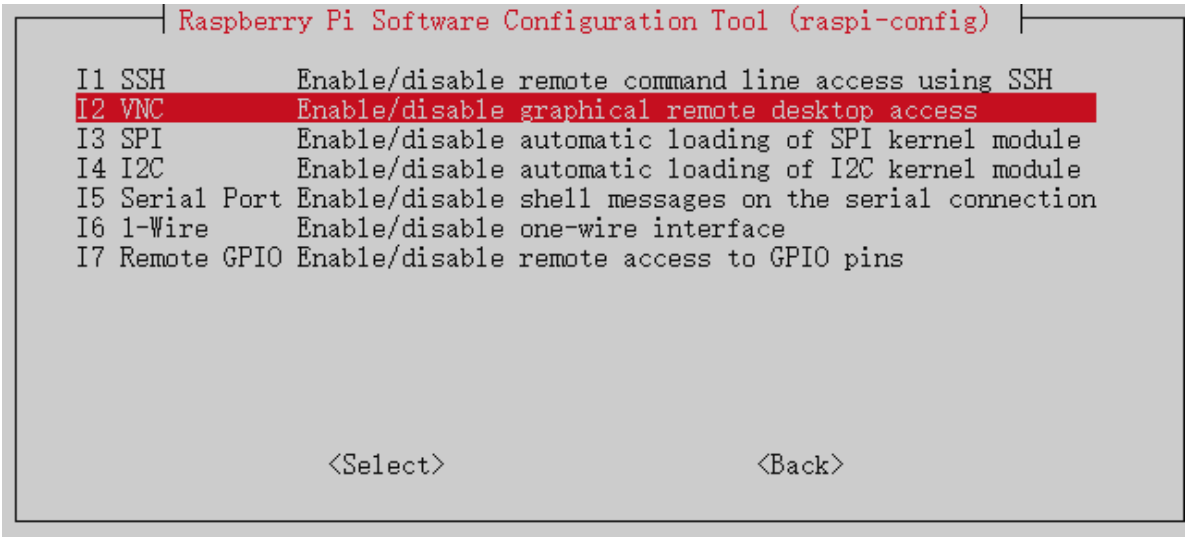
5. Enter the following command to enable the **VNC Server**.

```
sudo raspi-config
```

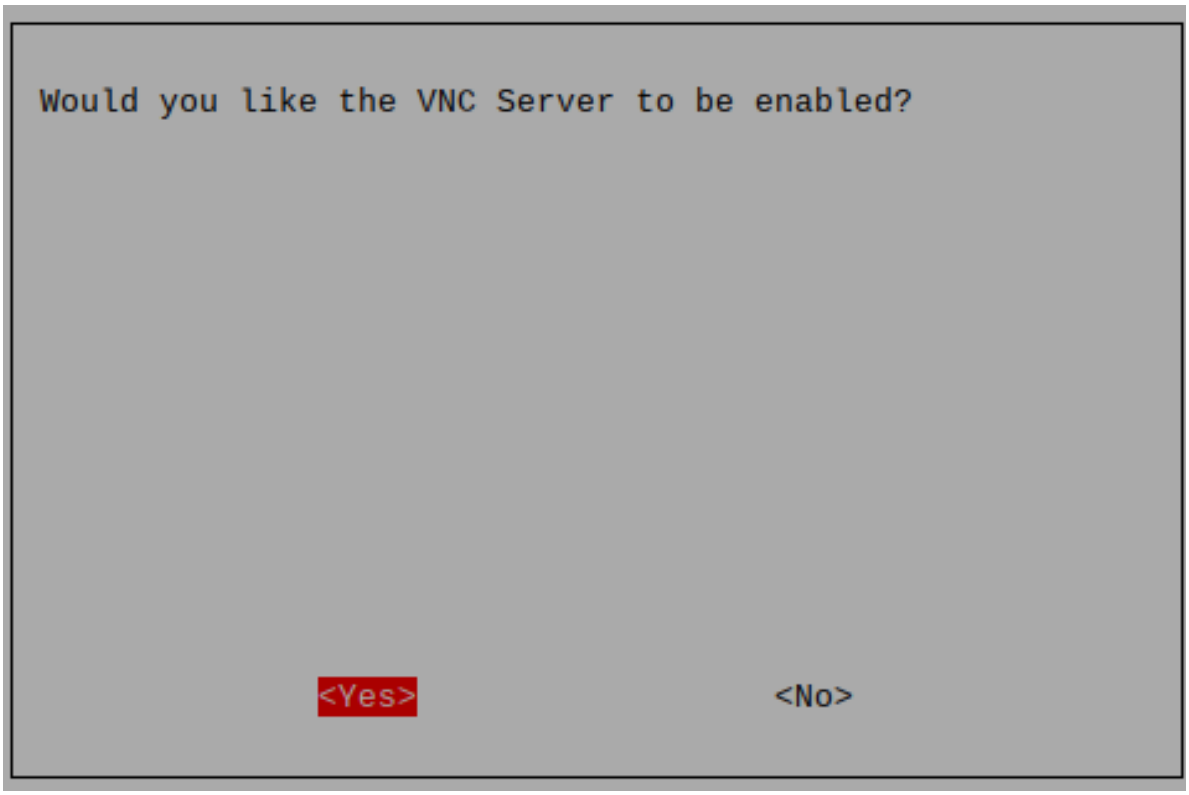
6. The following screen will be displayed. Select **3 Interface Options** with the arrow keys on the keyboard and press the **Enter** key.



7. Then select **VNC**.



8. Use the arrow keys on the keyboard to select **<Yes>** -> **<OK>** -> **<Finish>** to complete the setup.

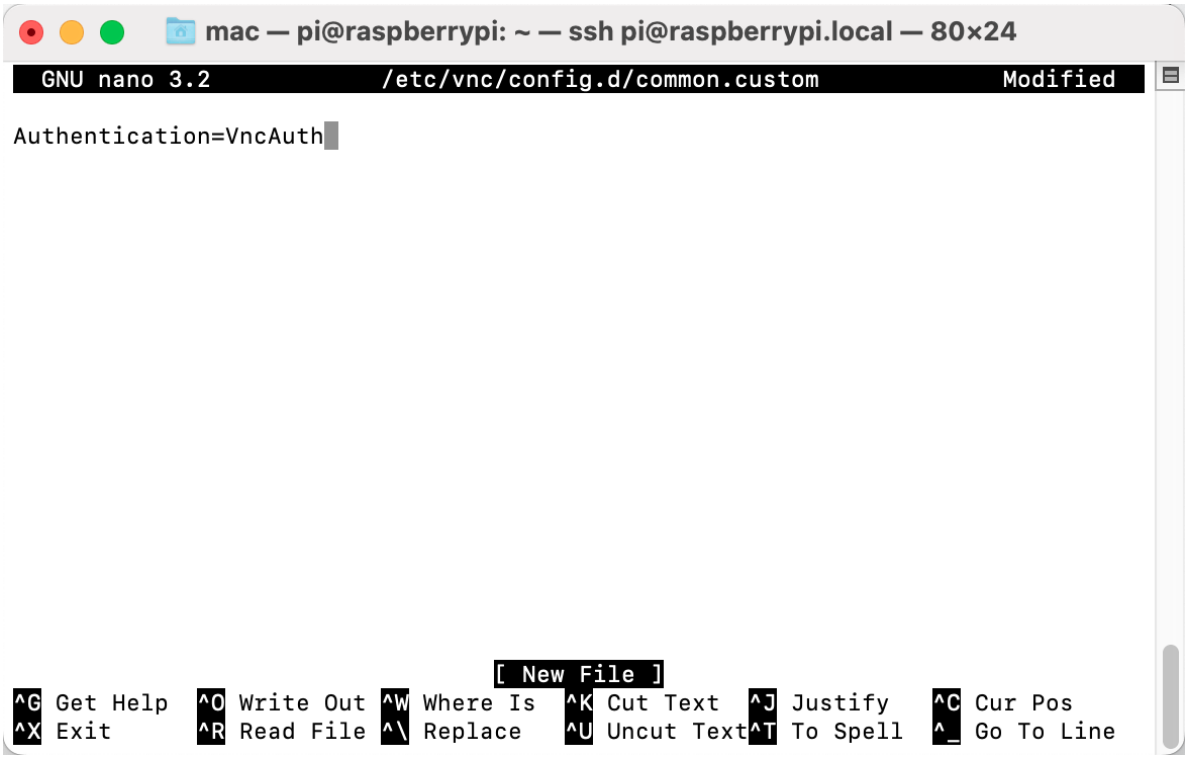


9. Now that the VNC server has started, let's change the settings for connecting from a Mac.

   To specify parameters for all programs for all user accounts on the computer, create `/etc/vnc/config.d/common.custom`.

   ```
   sudo nano /etc/vnc/config.d/common.custom
   ```

   After entering `Authentication=VncAuthenter`, press `Ctrl+X` -> `Y` -> `Enter` to save and exit.



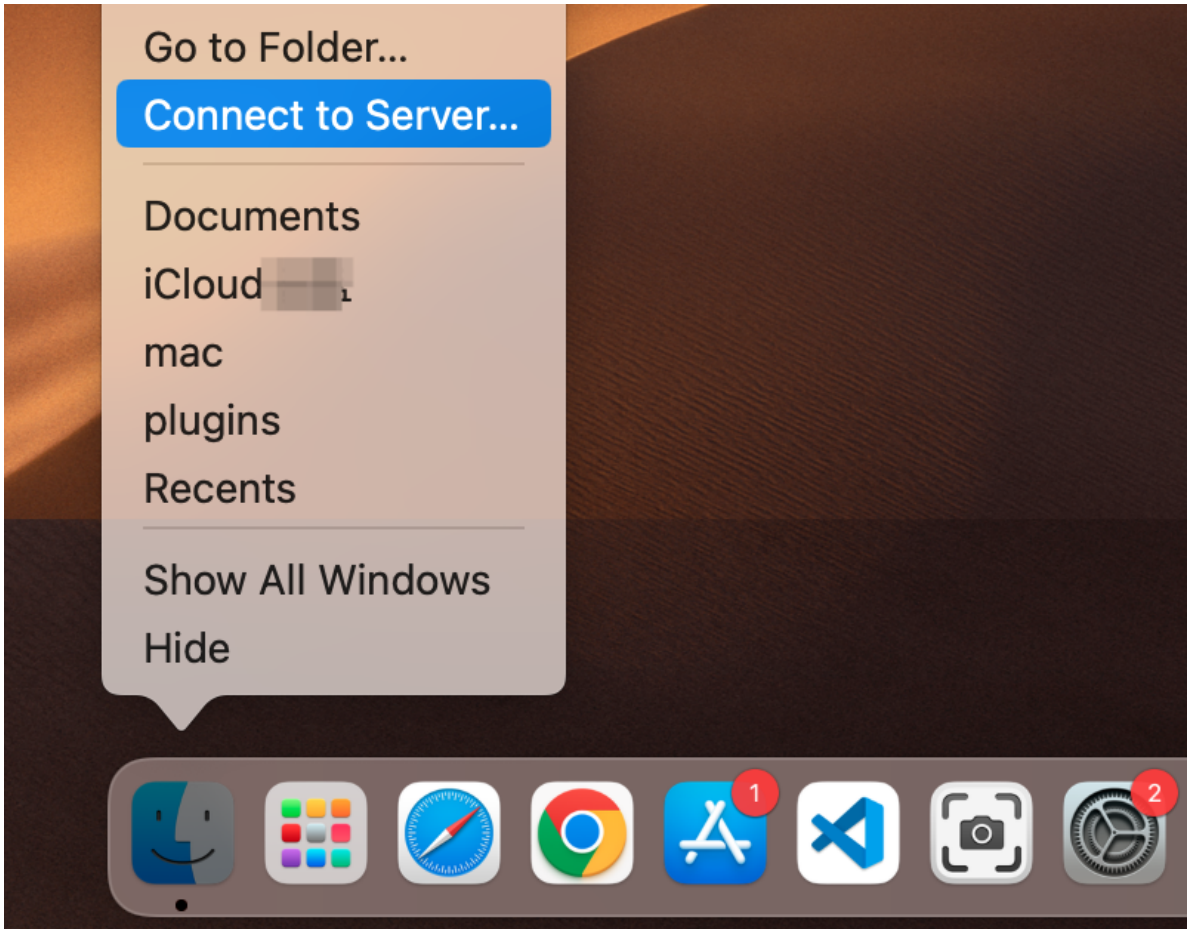
10. In addition, set a password for logging in via VNC from a Mac. You can use the same password as the Raspberry pi password or a different password.

```
sudo vncpasswd -service
```

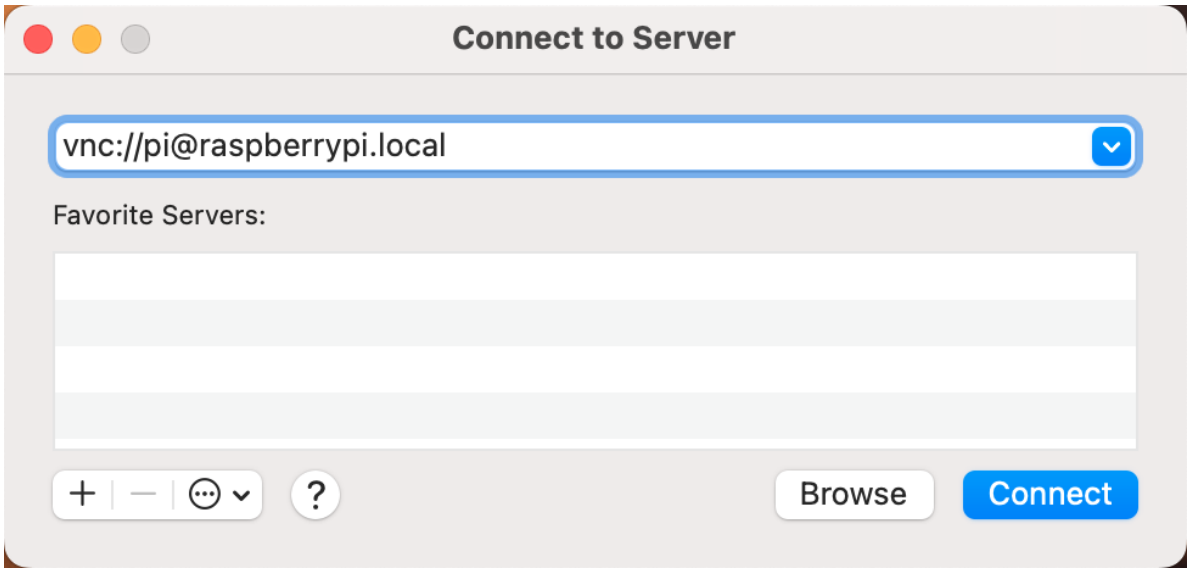
11. Once the setup is complete, restart the Raspberry Pi to apply the changes.

```
sudo sudo reboot
```

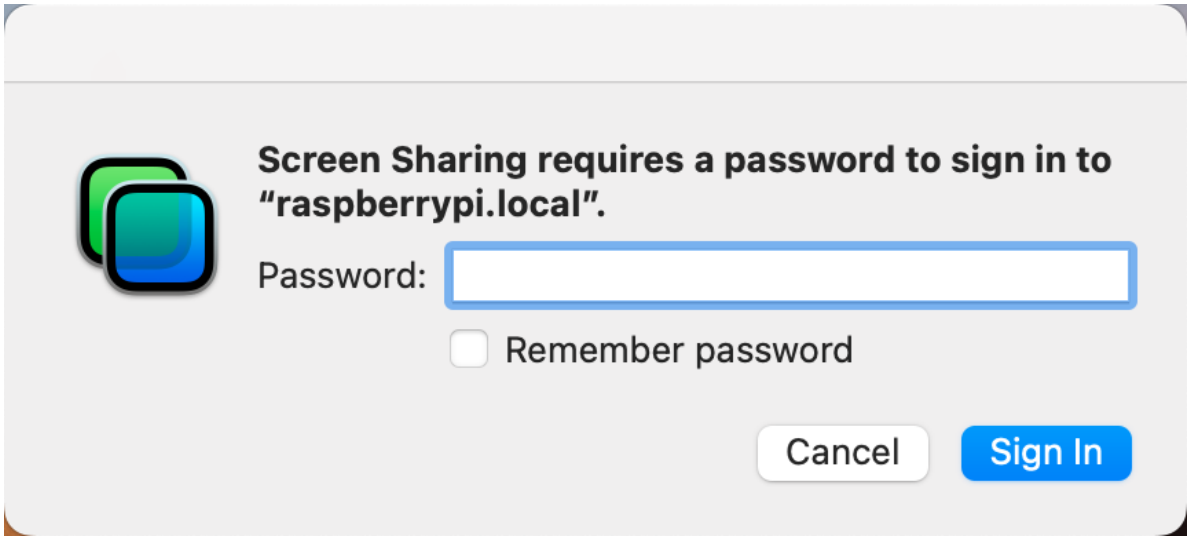
12. Now, select **Connect to Server** from the **Finder**'s menu, which you can open by right-clicking.



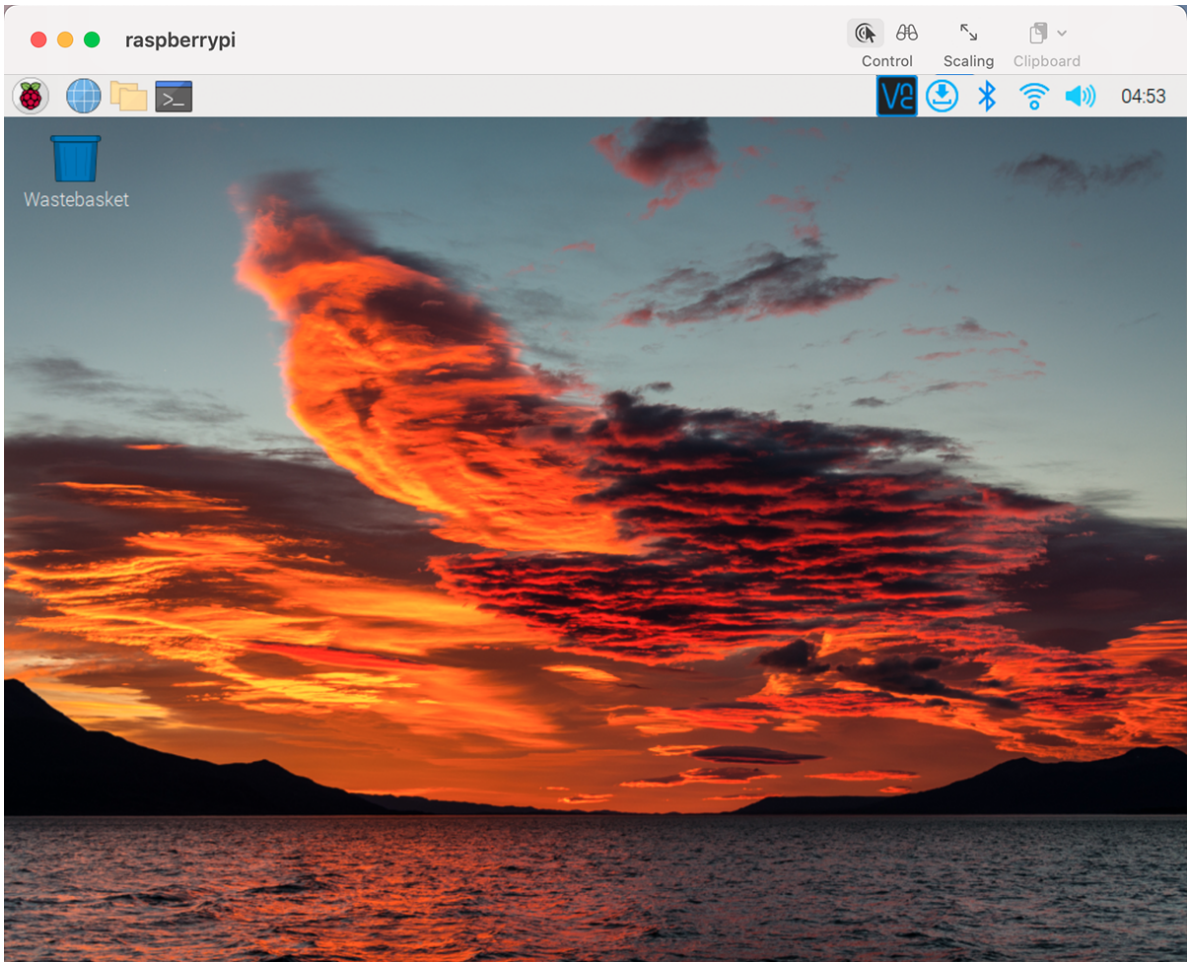
13. Type in `vnc://<username>@<hostname>.local` (or `vnc://<username>@<IP address>`). After entering, click **Connect**.



14. You will be asked for a password, so please enter it.



15. The desktop of the Raspberry pi will be displayed, and you will be able to operate it from the Mac as it is.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

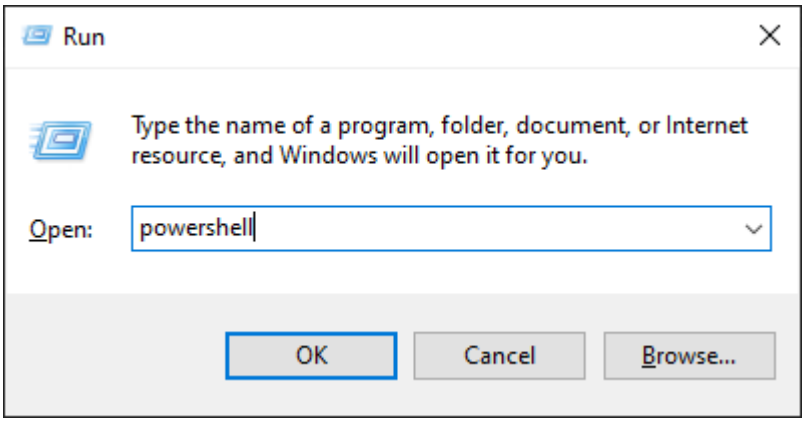
- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Windows Users

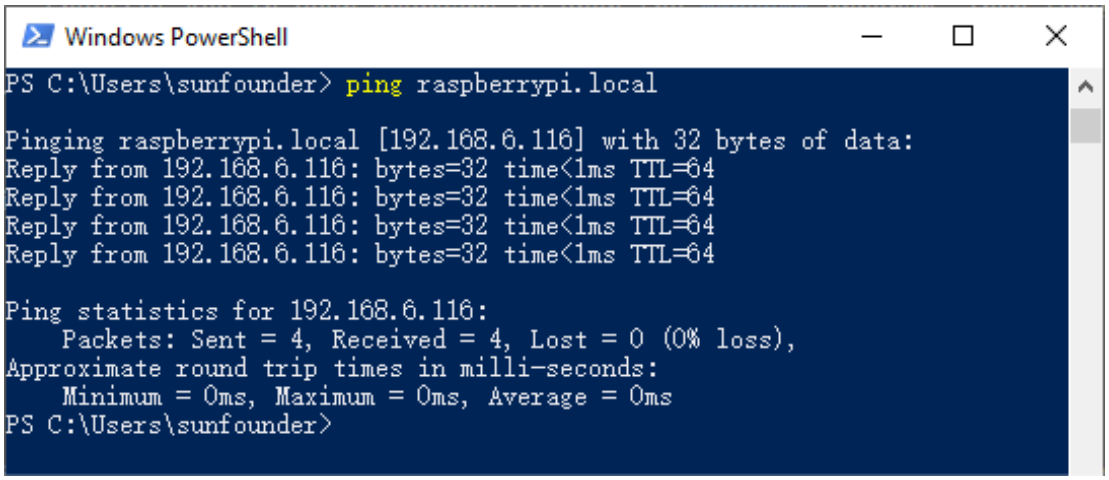
If you're a Windows user, you can use Windows PowerShell to login Raspberry Pi remotely.

1. Press the `windows` + `R` shortcut key in your keyboard to open the **Run** program. Then type **powershell** in the input box.



2. Check if your Raspberry Pi is on the same network by type in `ping <hostname>.local`.

```
ping raspberrypi.local
```



- If terminal prompts `Ping request could not find host <hostname>.local`, it is possible that the Raspberry Pi failed to connect to the network.lease check the network.

- If you really can't ping <hostname>.local, try to *Get the IP address* and ping <IP address> instead. (e.g., ping 192.168.6.116)
- If multiple prompts like "Reply from <IP address>: bytes=32 time<1ms TTL=64" appear, it means your computer can access the Raspberry Pi.

3. Type in ssh <username>@<hostname>.local (or ssh <username>@<IP address>).

```
ssh pi@raspberrypi.local
```

4. The following message may appear.

```
The authenticity of host 'raspberrypi.local (192.168.6.116)' can't be established.
ECDSA key fingerprint is SHA256:7ggckKZ2EEgS76a557cddfxFNDOBBuzcJsgaqA/igz4.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

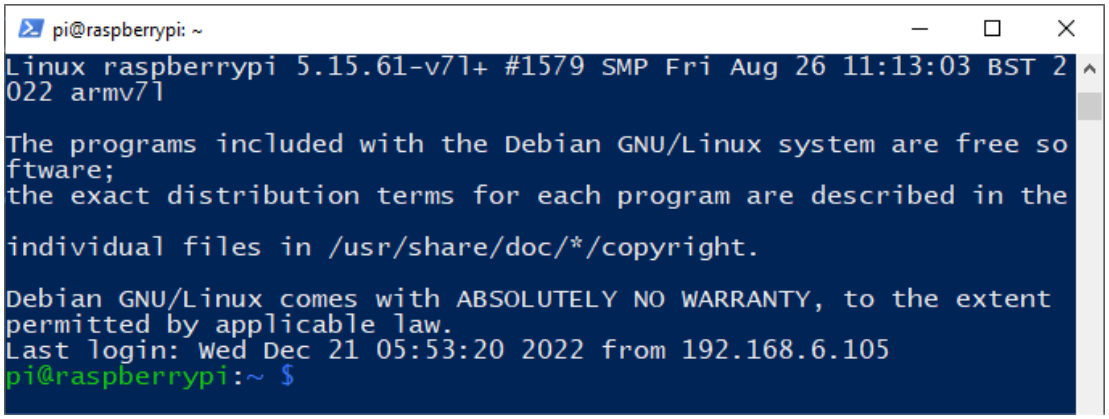
Input "yes".

5. Input the password you set before. (Mine is raspberry.)

Note: When you input the password, the characters do not display on window accordingly, which is normal. What you need is to input the correct password.

---

6. We now get the Raspberry Pi connected and are ready to go to the next step.



Remote Desktop

If you're not satisfied with using the command window to access your Raspberry Pi, you can also use the remote desktop feature to easily manage files on your Raspberry Pi using a GUI.

Here we use VNC® Viewer.

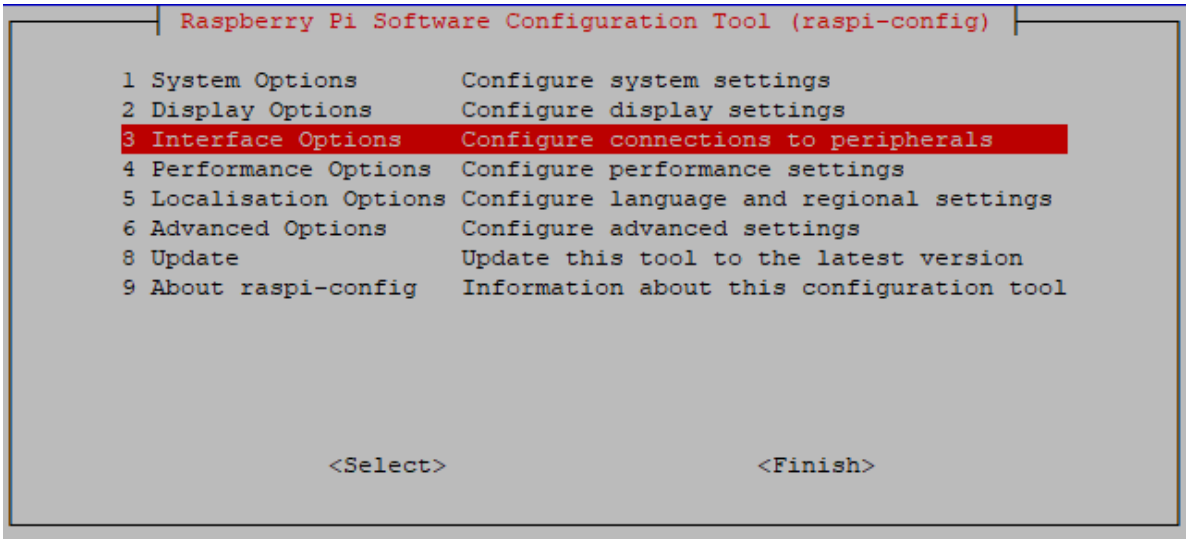
Enable VNC service

The VNC service has been installed in the system. By default, VNC is disabled. You need to enable it in config.

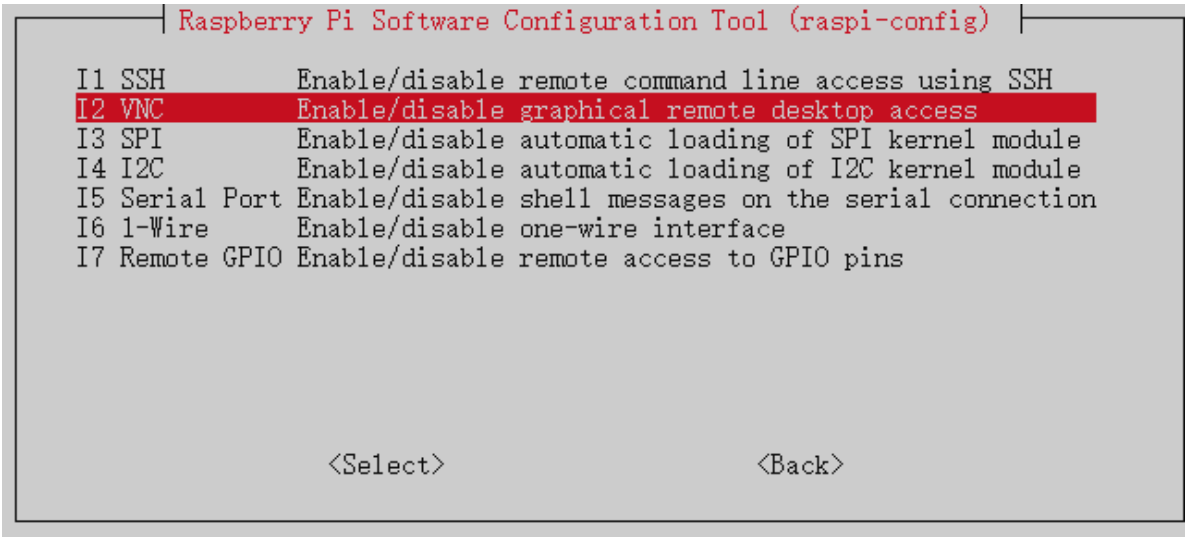
1. Input the following command:

```
sudo raspi-config
```

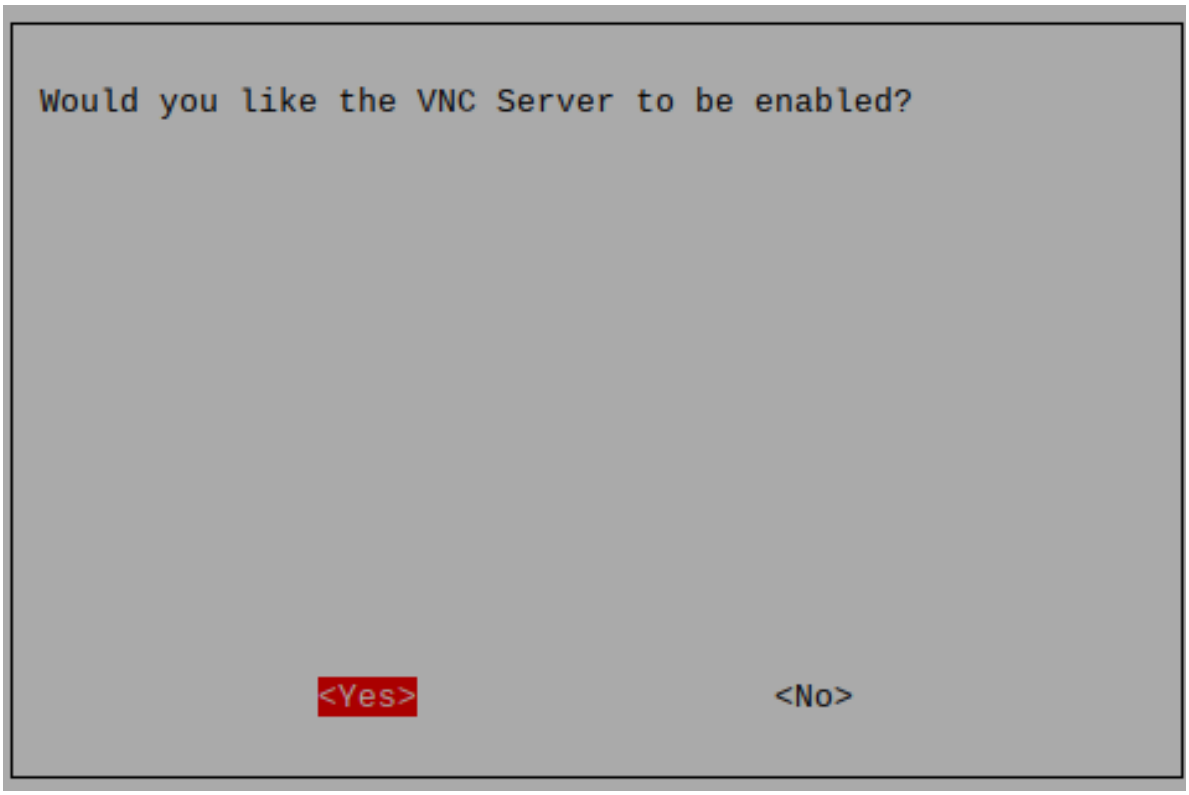
2. Choose **3 Interfacing Options** by press the down arrow key on your keyboard, then press the **Enter** key.



3. Then **VNC**.

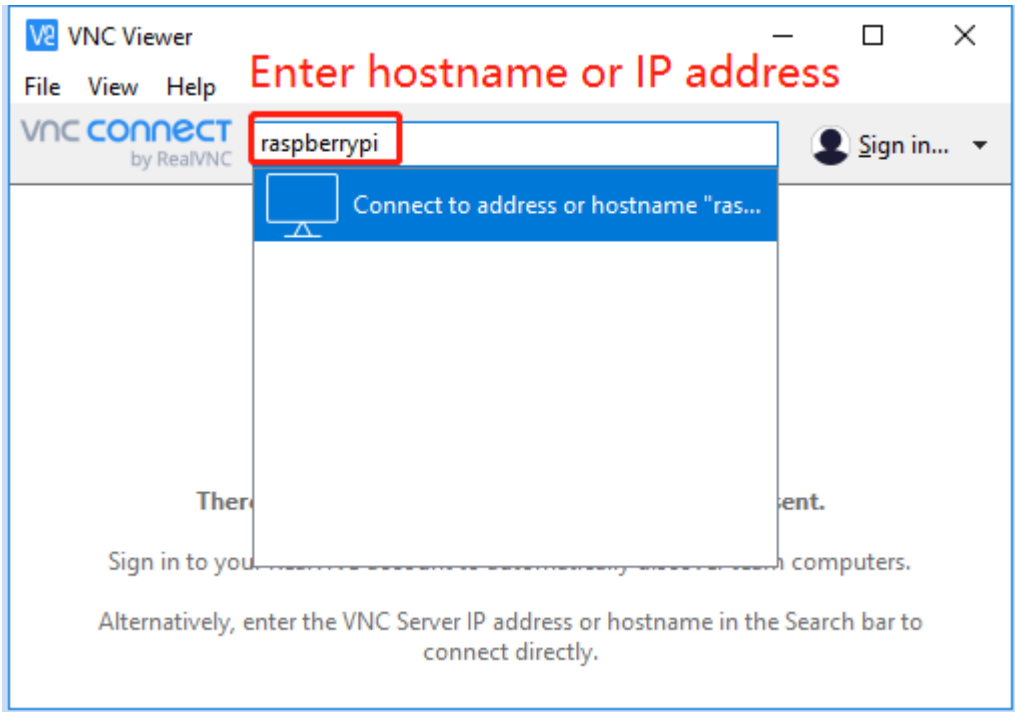


4. Use the arrow keys on the keyboard to select **<Yes>** -> **<OK>** -> **<Finish>** to complete the setup.

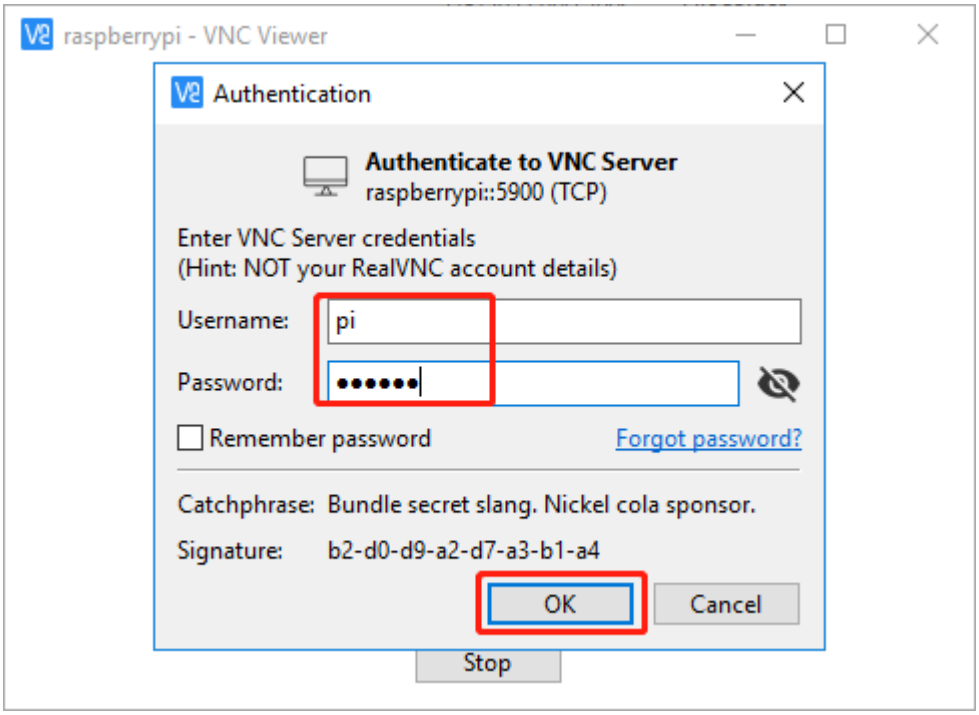


Login to VNC

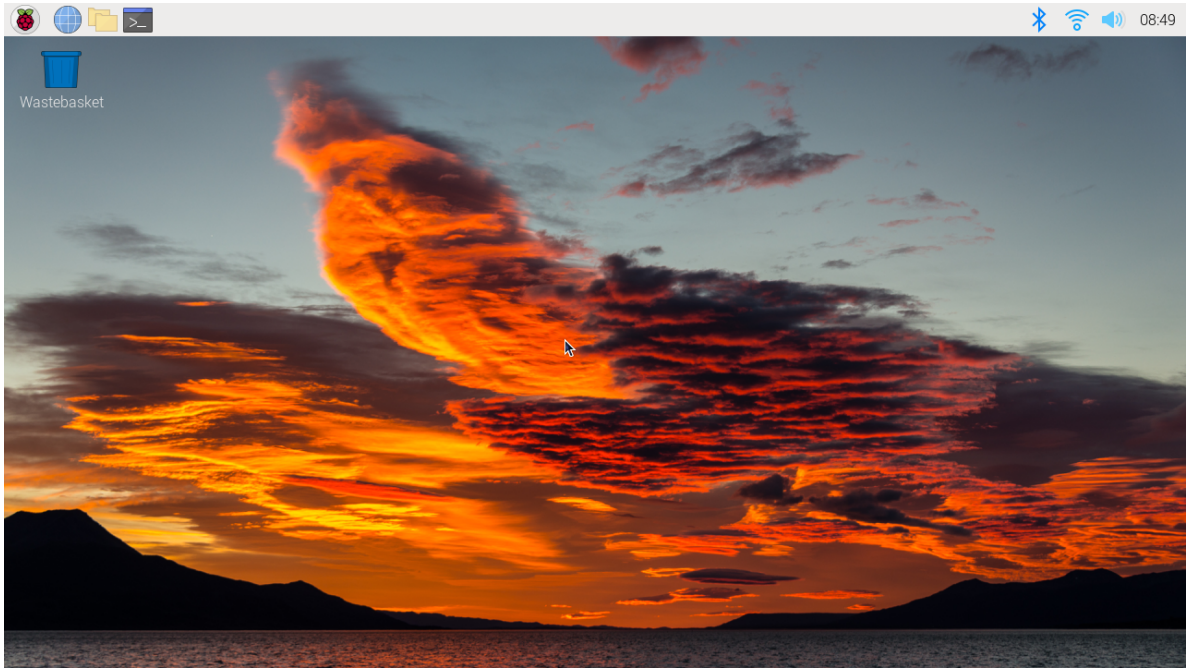
1. You need to download and install the VNC Viewer on personal computer.
2. Open it once the installation is complete. Then, enter the host name or IP address and press Enter.



3. After entering your Raspberry Pi name and password, click **OK**.



4. Now you can see the desktop of the Raspberry Pi.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

For Linux/Unix Users

1. Locate and open the **Terminal** on your Linux/Unix system.
2. Ensure your Raspberry Pi is connected to the same network. Verify this by typing `ping <hostname>.local`. For example:

   ```
   ping raspberrypi.local
   ```

   You should see the Raspberry Pi's IP address if it's connected to the network.

   - If the terminal shows a message like `Ping request could not find host pi.local. Please check the name and try again.`, double-check the hostname you've entered.
   - If you're unable to retrieve the IP address, inspect your network or WiFi settings on the Raspberry Pi.
3. Initiate an SSH connection by typing `ssh <username>@<hostname>.local` or `ssh <username>@<IP address>`. For instance:

   ```
   ssh pi@raspberrypi.local
   ```
4. On your first login, you'll encounter a security message. Type `yes` to proceed.

   ```
   The authenticity of host 'raspberrypi.local␣
   ↪(2400:2410:2101:5800:635b:f0b6:2662:8cba)' can't be established.
   ED25519 key fingerprint is SHA256:oo7x3ZSgAo032wD1tE8eW0fFM/kmewIvRwkBys6XRwg.
   Are you sure you want to continue connecting (yes/no/[fingerprint])?
   ```
5. Enter the password you previously set. Note that for security reasons, the password won't be visible as you type.

   Note: It's normal for the password characters not to display in the terminal. Just ensure to enter the correct password.
6. Once you've successfully logged in, your Raspberry Pi is now connected, and you're ready to proceed to the next step.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Check the `GPIO Zero`

`GPIO Zero` is a module for controlling Raspberry Pi GPIO pins. This package provides a range of user-friendly classes and functions to control GPIO on a Raspberry Pi. For examples and documentation, visit: https://gpiozero.readthedocs.io/en/latest/.

The latest Raspberry Pi OS includes GPIO Zero by default. To verify its installation, open the Terminal and enter:

```
python
```

```
pi@raspberrypi:~ $ python
Python 3.11.2 (main, Mar 13 2023, 12:18:29) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Next, type `import gpiozero` within the Python CLI. If no errors appear, GPIO Zero is successfully installed.

```
import gpiozero
```

```
pi@raspberrypi:~ $ python
Python 3.11.2 (main, Mar 13 2023, 12:18:29) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import gpiozero
>>>
```

If you want to quit python CLI, type in:

```
exit()
```

```
>>> exit()
pi@raspberrypi:~ $
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

How to download and run the Code

Downloading Code to Your Raspberry Pi

Before downloading the code, please be aware that the example code has been tested **ONLY** on the latest version of **Raspberry Pi OS**. We offer two download methods:

If you're not accessing your Raspberry Pi with a direct screen connection, consider utilizing remote access options. For detailed guidance, refer to the instructions in *If You Have No Screen*.

Method 1: Using Git Clone (Recommended)

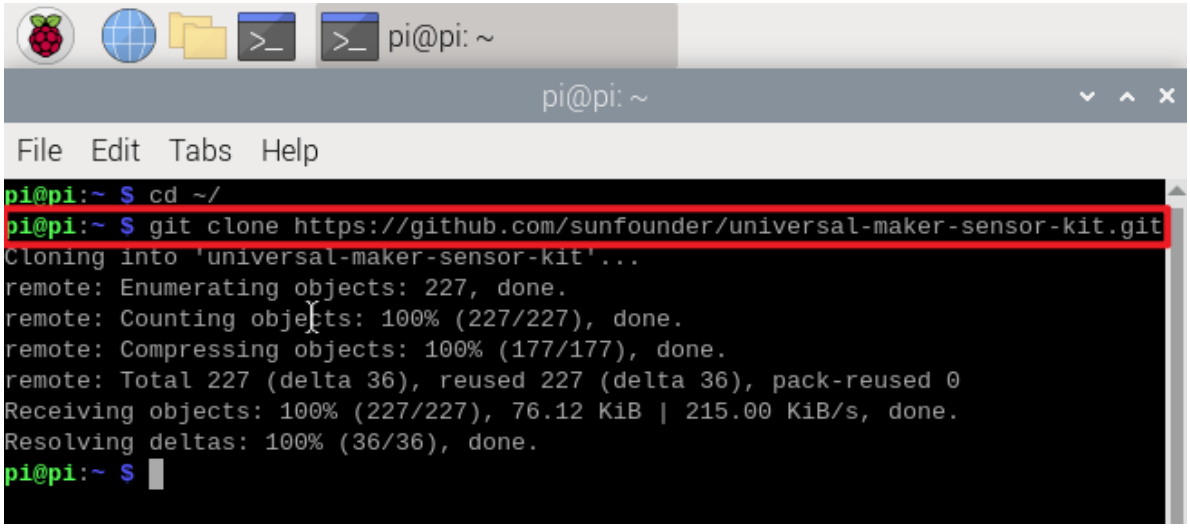
1. Log into your Raspberry Pi, open Terminal, and navigate to the home directory (~). (You can also access the terminal using SSH.)

```
cd ~
```

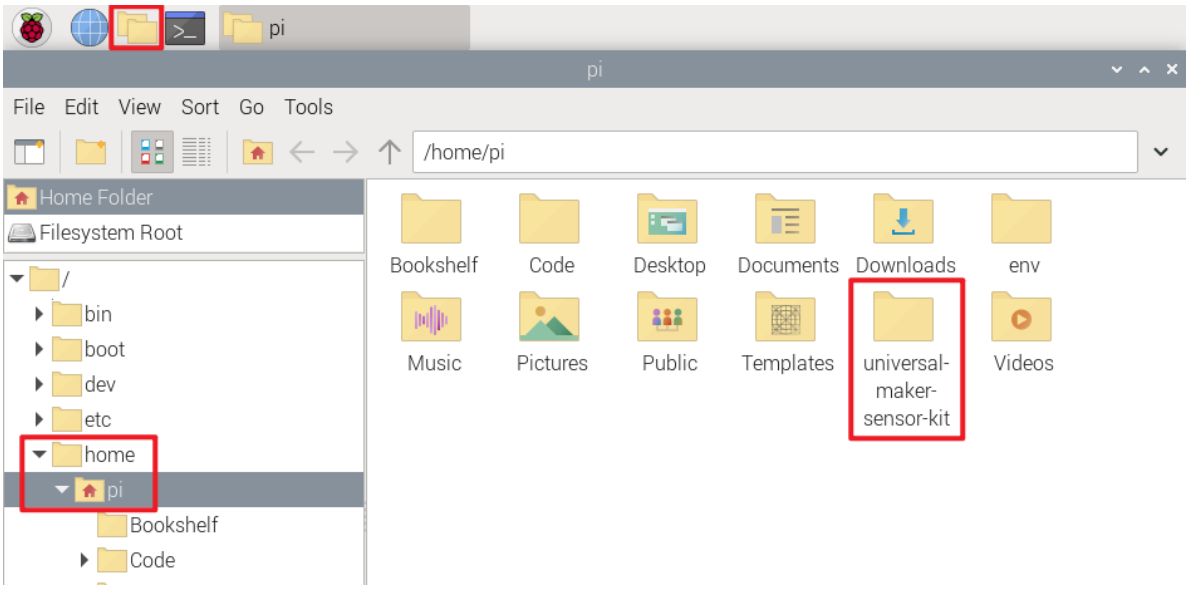
Note: Use the `cd` command to change directories. Here, `~/` denotes the home directory.

2. Clone the GitHub repository.

```
git clone https://github.com/sunfounder/universal-maker-sensor-kit.git
```

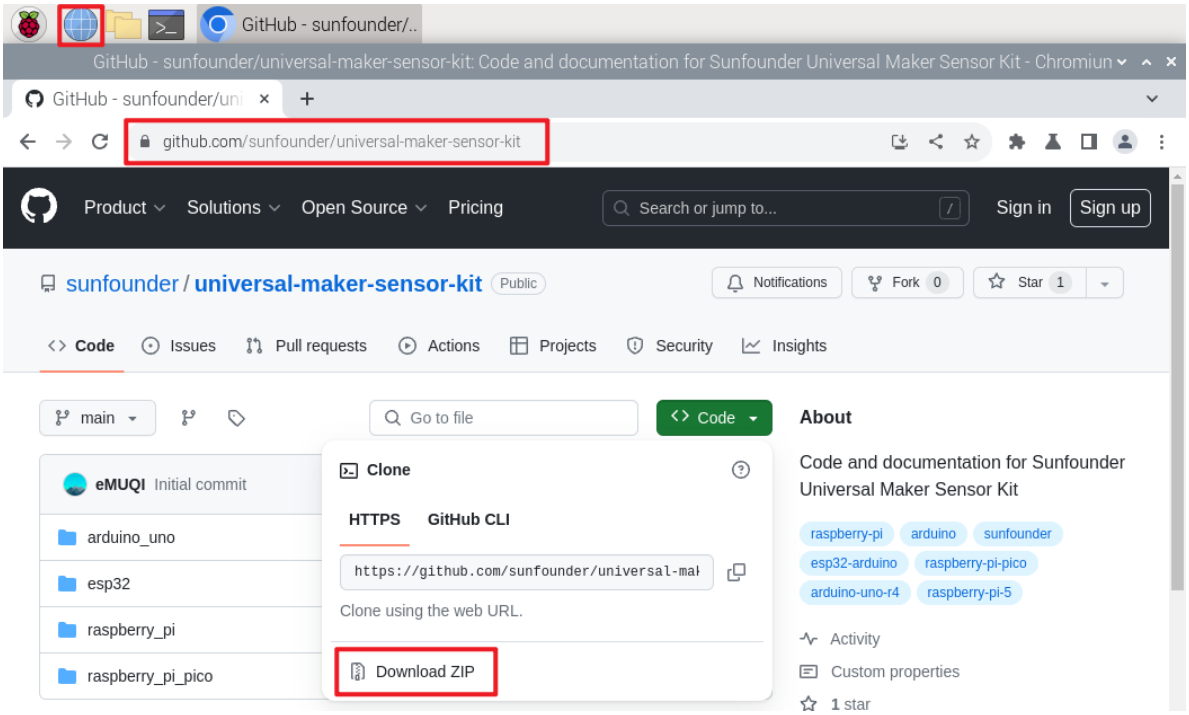


3. Use File Manager to access the downloaded code files.

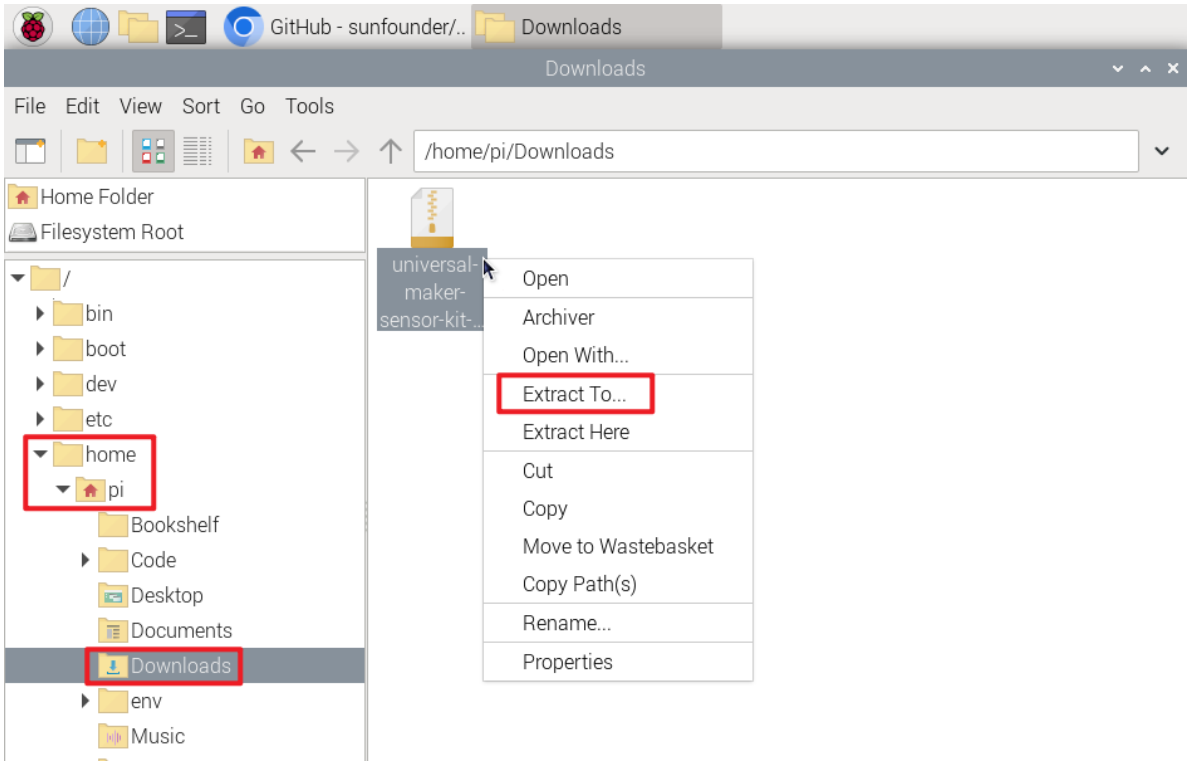


Method 2: Downloading Code Directly from GitHub

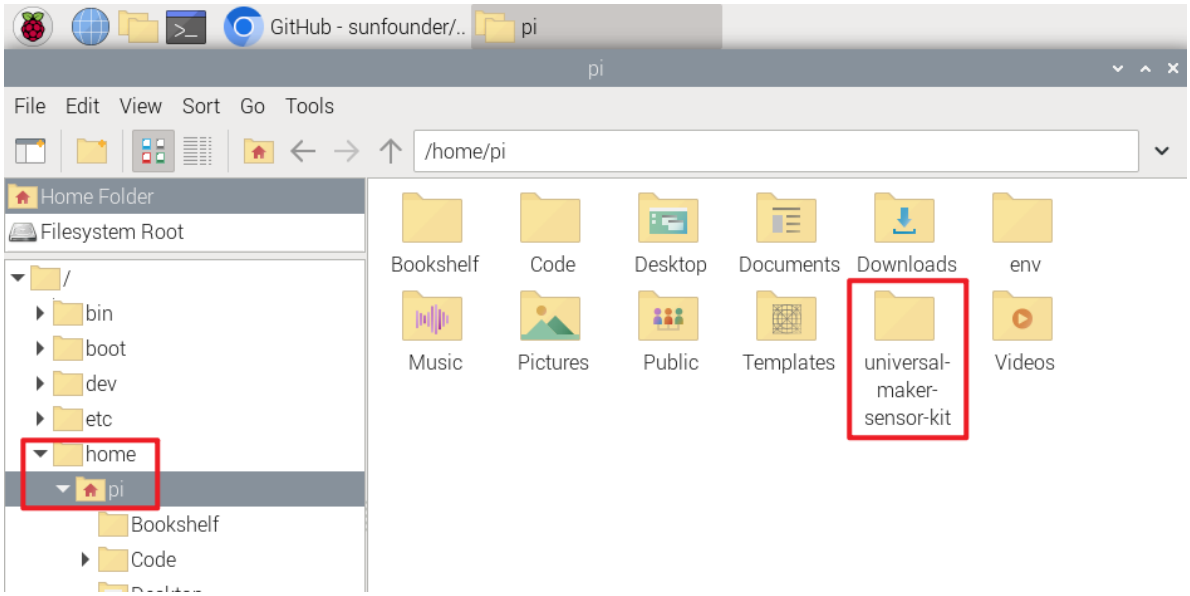
1. Open a web browser and go to https://github.com/sunfounder/universal-maker-sensor-kit, then click on the download button.



2. Once downloaded, locate the code file in `File Manager` > `Downloads` and unzip it into the `/home/pi` directory.



3. Navigate to the `/home/pi` directory to access the extracted code files.



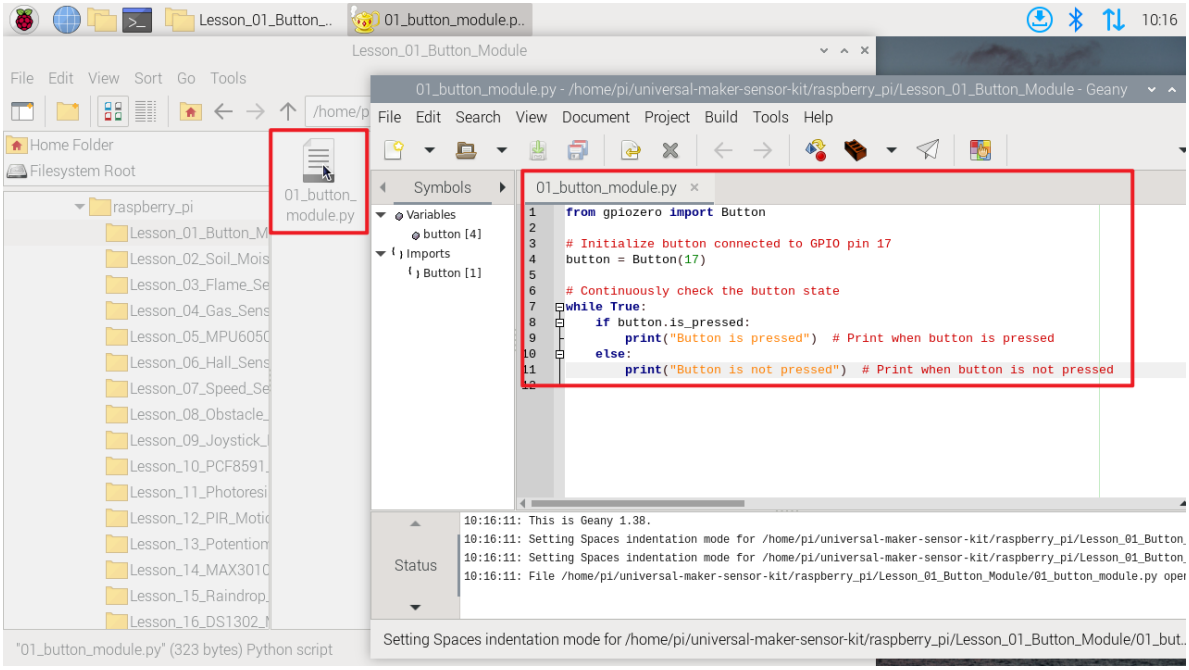
Opening and Running Code

You can find the code for each project in its respective code section. Alternatively, you can locate the code in the provided code directory. For instance, in `universal-maker-sensor-kit/raspberry_pi/`, you will find Lesson 1's code named `01_button_module.py`.

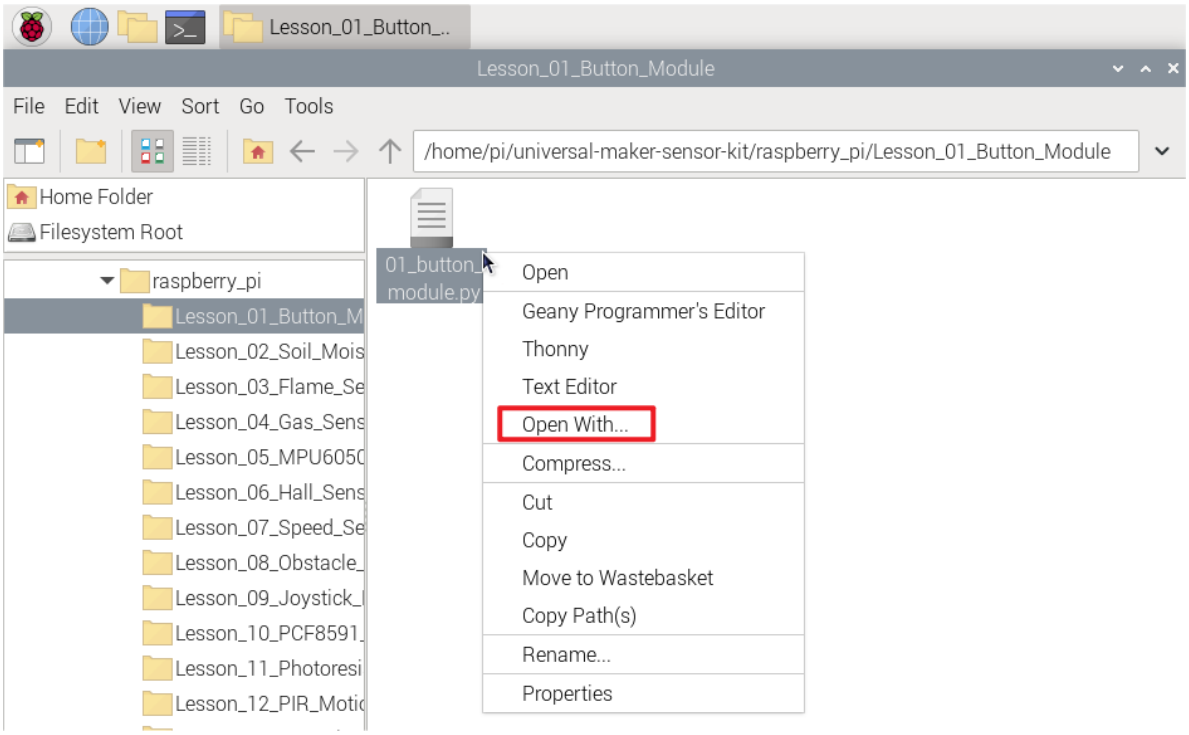
There are two ways to run Python code below

Method 1: Using Geany

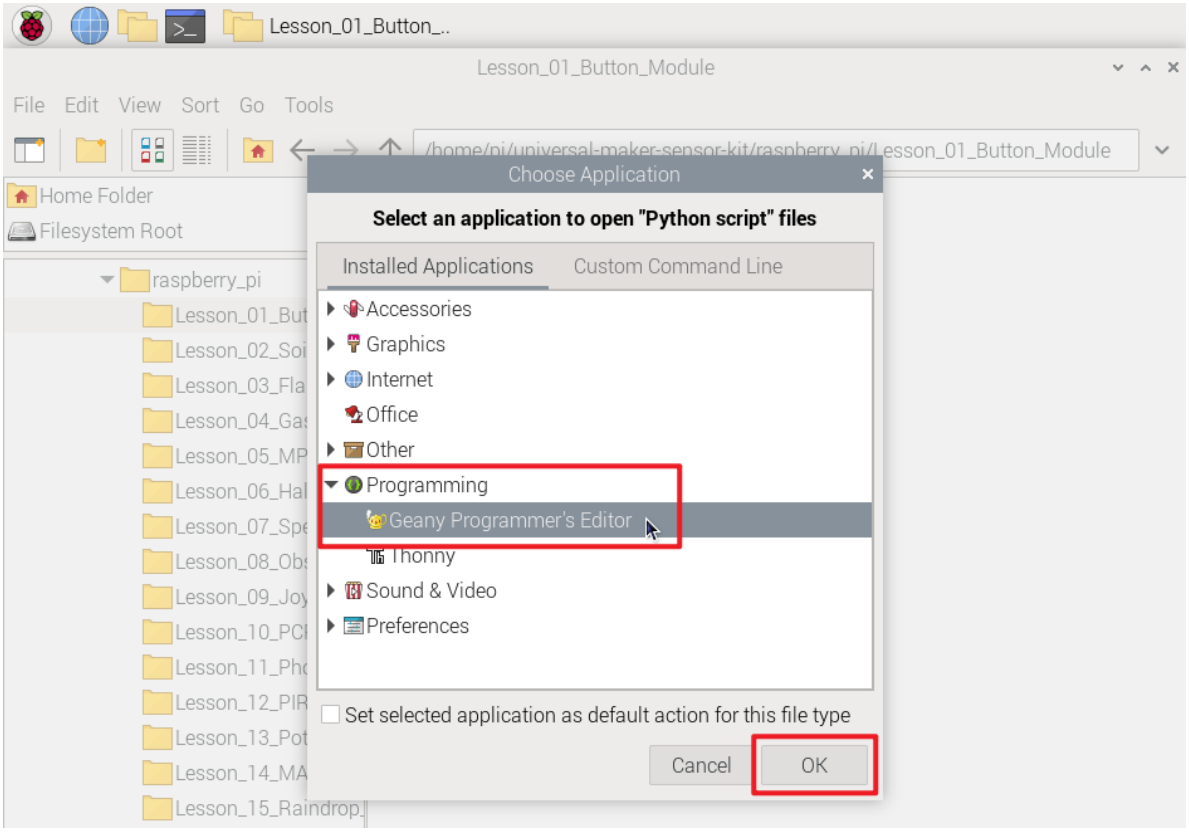
1. Open the code file by double-clicking on it.



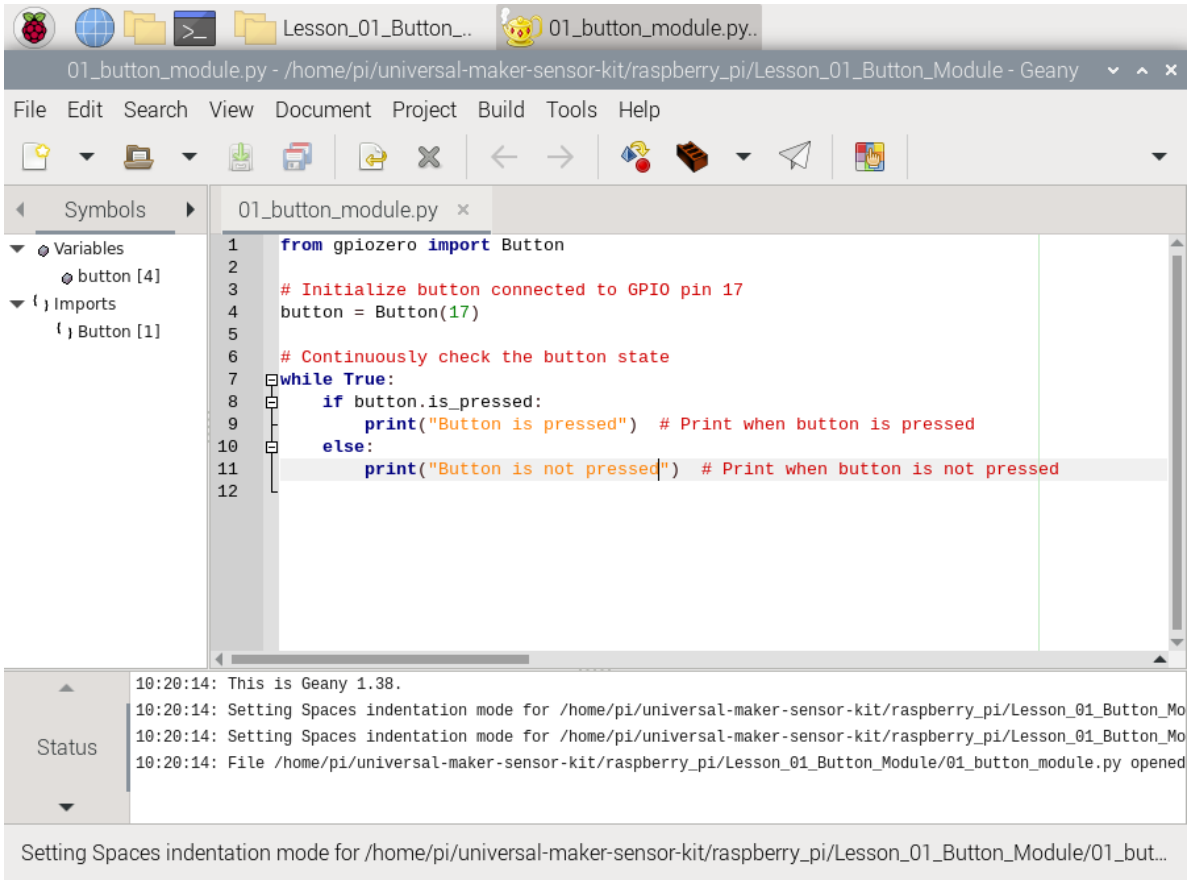
Alternatively, right-click the file and select **Open With…**.



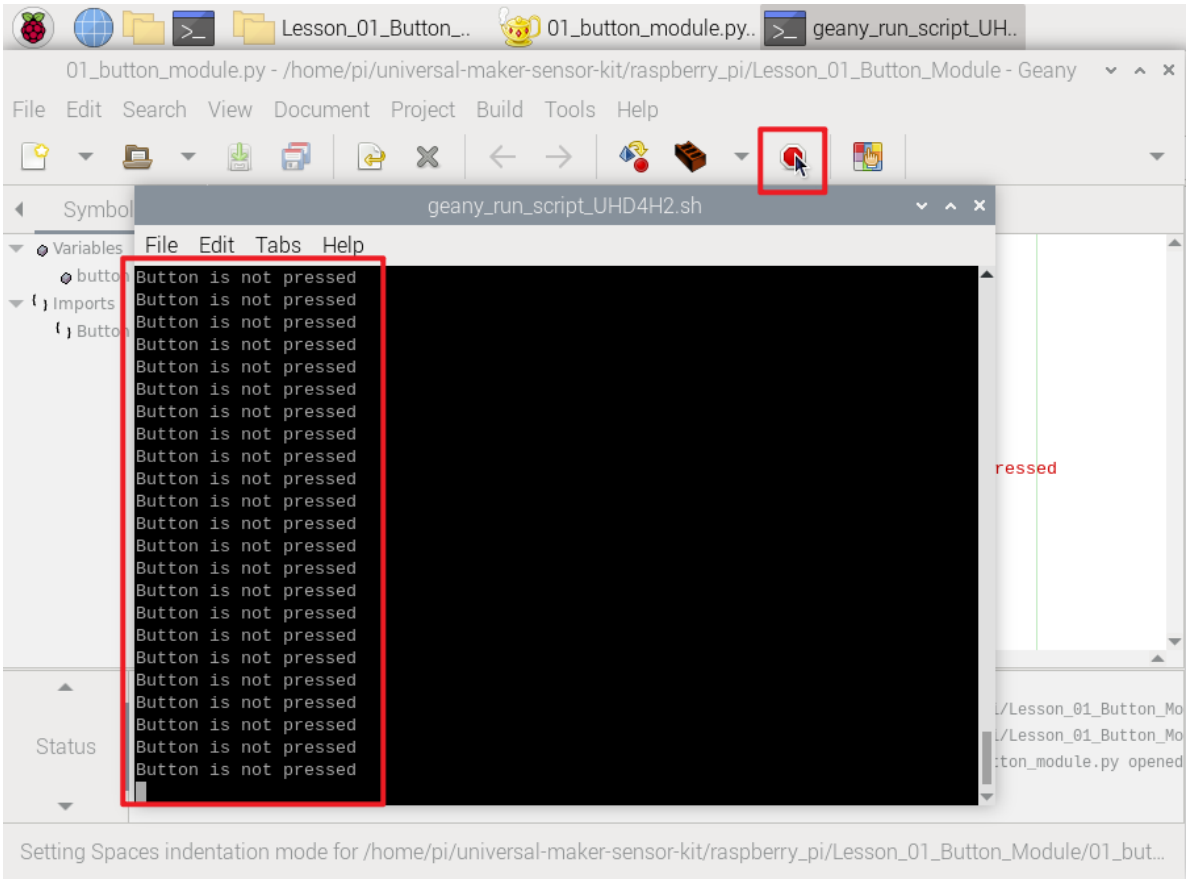
Choose **Programming > Geany Programmer's Editor** and click **OK**.



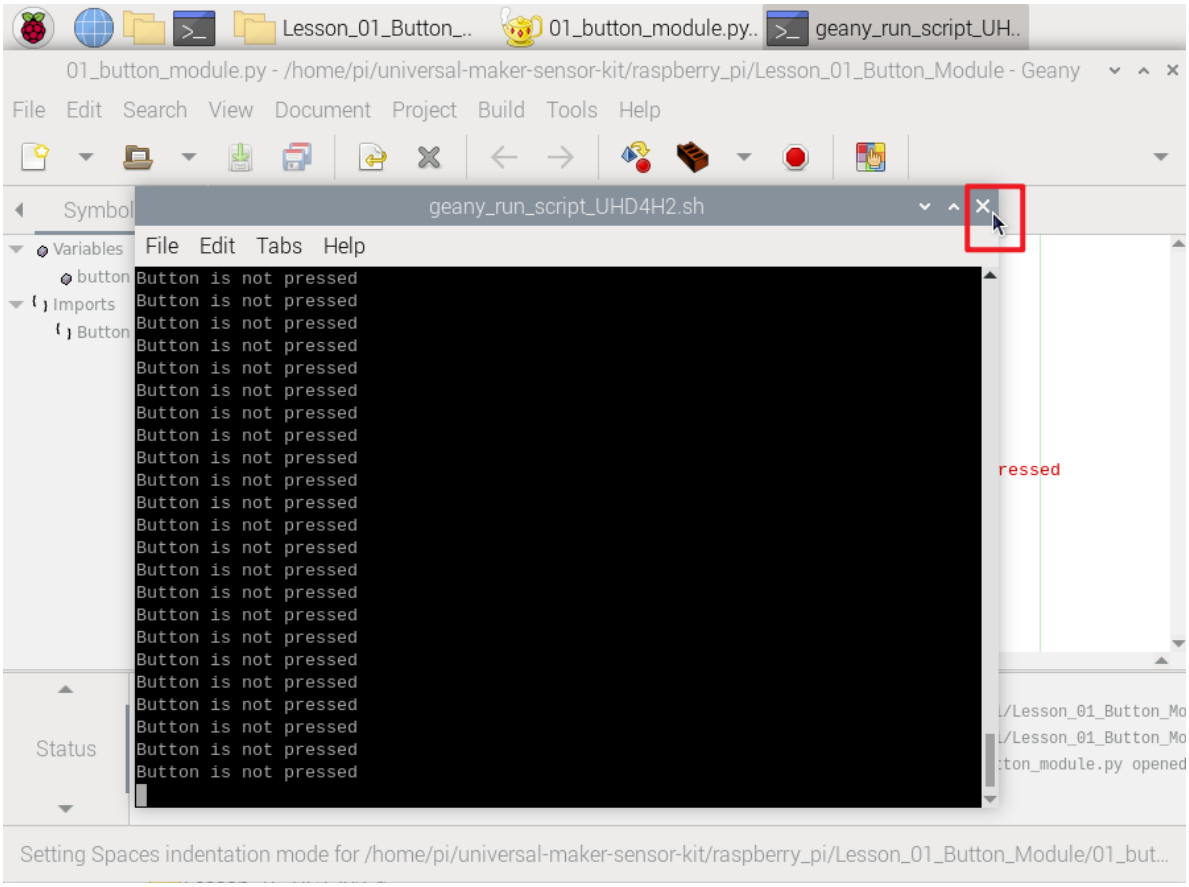
The code will be displayed for editing or review.



2. Click **Run** in the window and the following contents will appear.



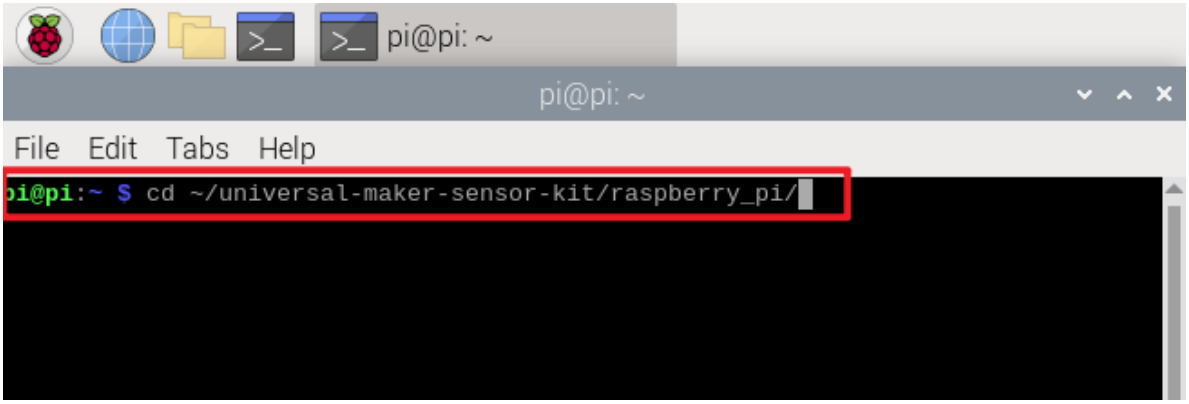
3. To stop it from running, just click the X button in the top right corner to close it and you'll return to the code. Alternatively, you can terminate the program by typing ctrl+c.



Method 2: Using Terminal

1. Log into your Raspberry Pi, open Terminal, and navigate to the home directory (~). (You can also access the terminal using SSH.)

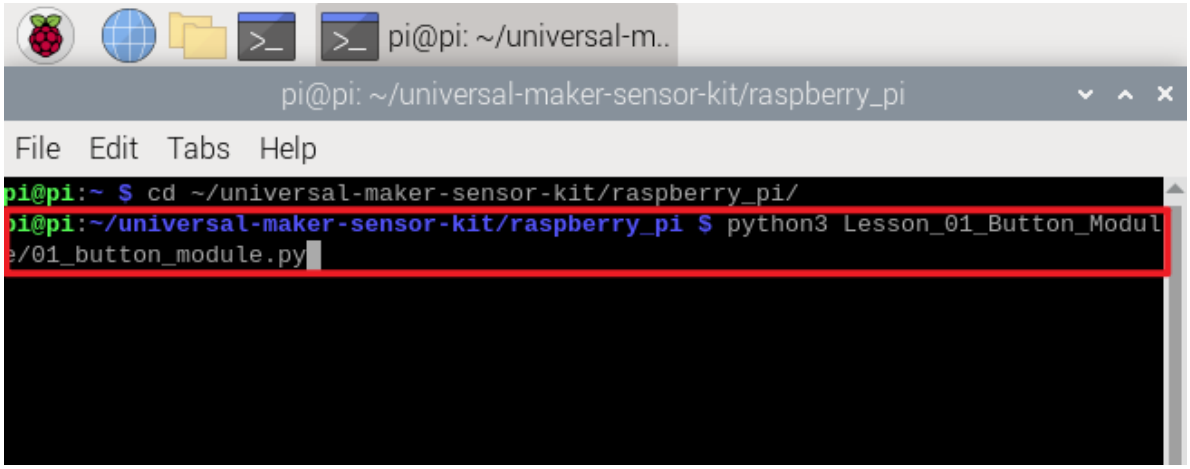
```
cd ~/universal-maker-sensor-kit/raspberry_pi/
```



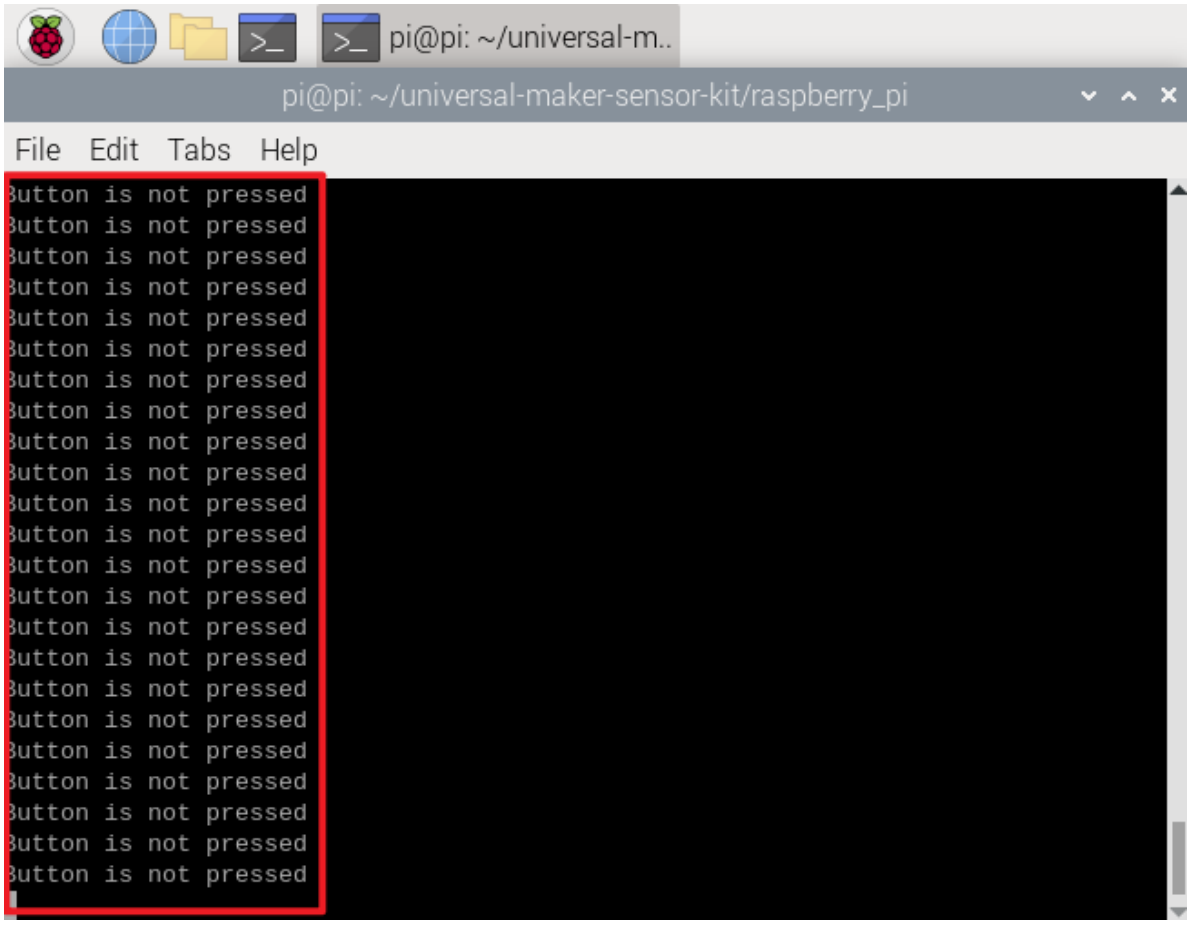
Note: Use the cd command to navigate to the experiment's code directory.

2. Execute the code:

```
python3 Lesson_01_Button_Module/01_button_module.py
```

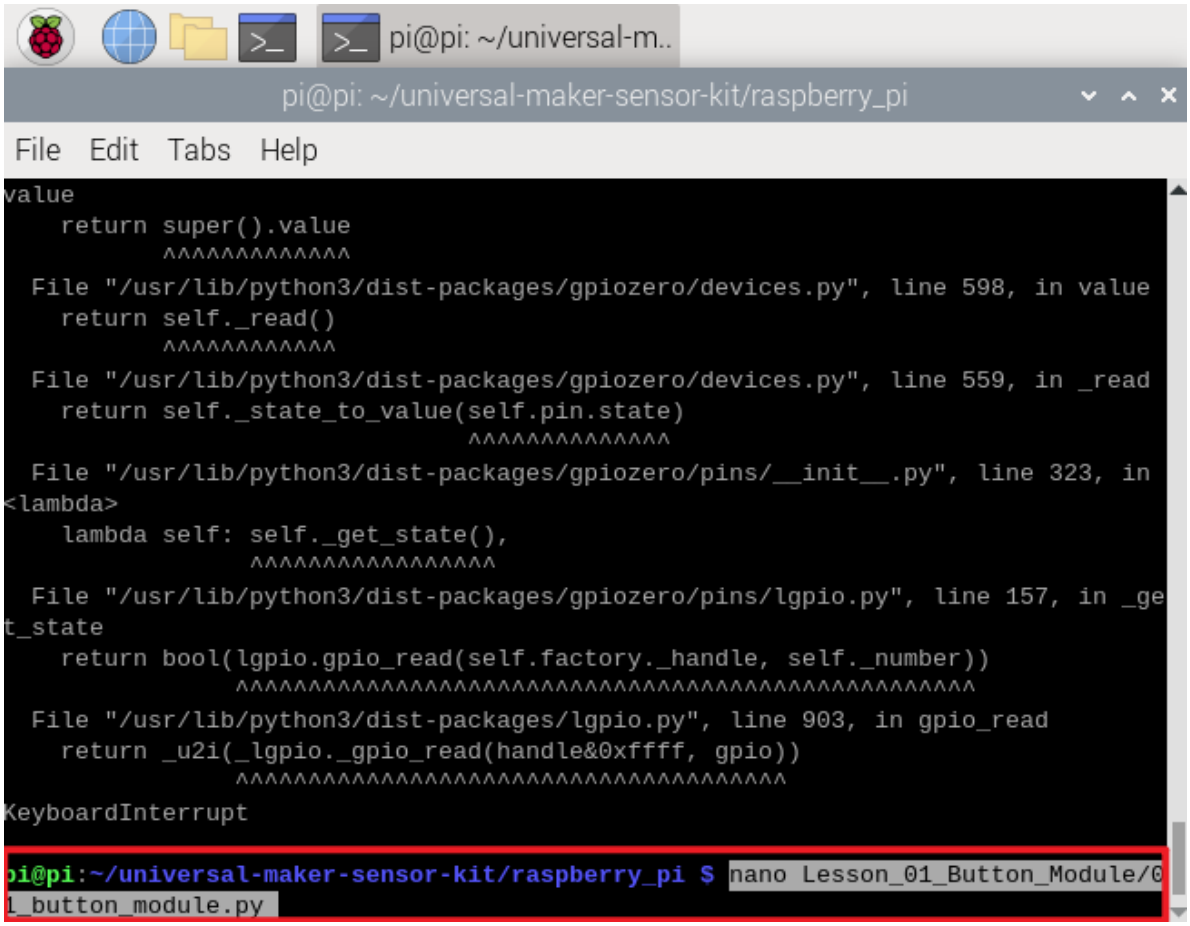


3. Upon running the code, the output will indicate whether the Button is pressed or not.

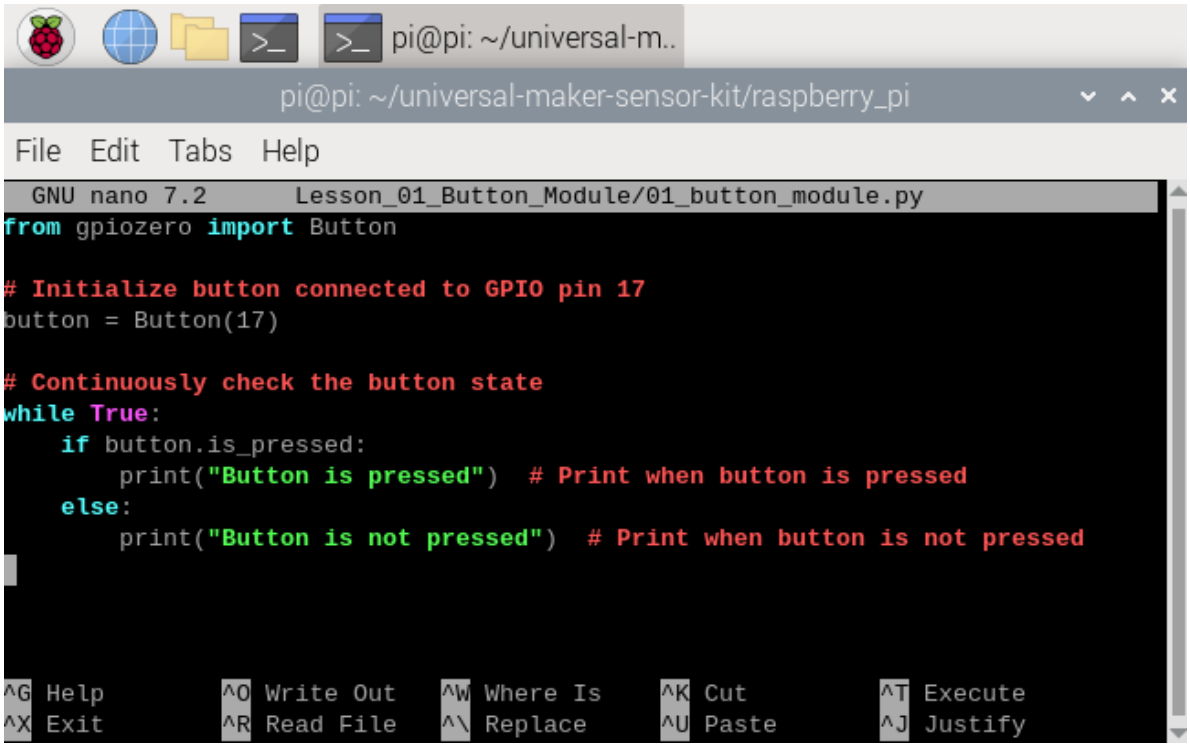


4. To edit the `Lesson_01_Button_Module/01_button_module.py` file, stop the code by pressing `Ctrl + C`. Then, open the file with:

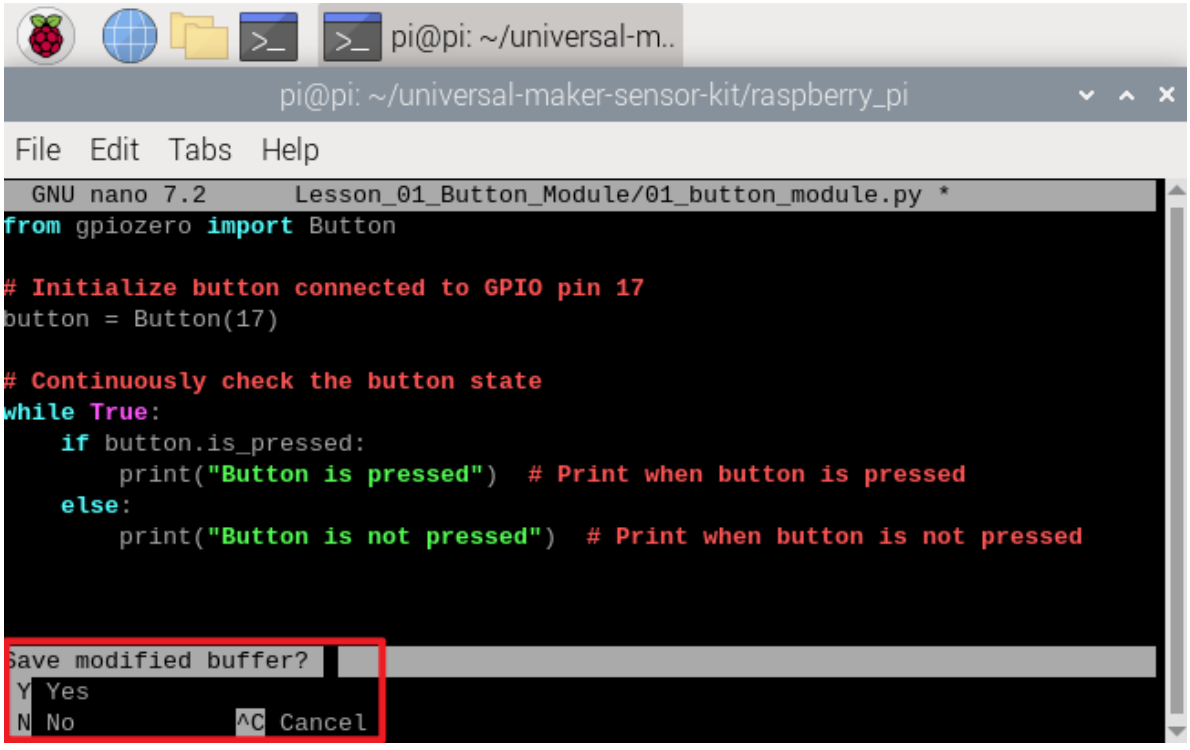
```
nano Lesson_01_Button_Module/01_button_module.py
```



5. nano is a text editor. This command opens `nano Lesson_01_Button_Module/01_button_module.py` for editing.



6. To exit nano, press `Ctrl+X`. If you've made changes, a prompt will ask whether to save them. Respond with `Y` (yes) to save or `N` (no) to discard. Press `Enter` to confirm and exit. Reopen the file with `nano Lesson_01_Button_Module/nano 01_button_module.py` to view your changes.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Configuring Your Raspberry Pi

I2C Configuration

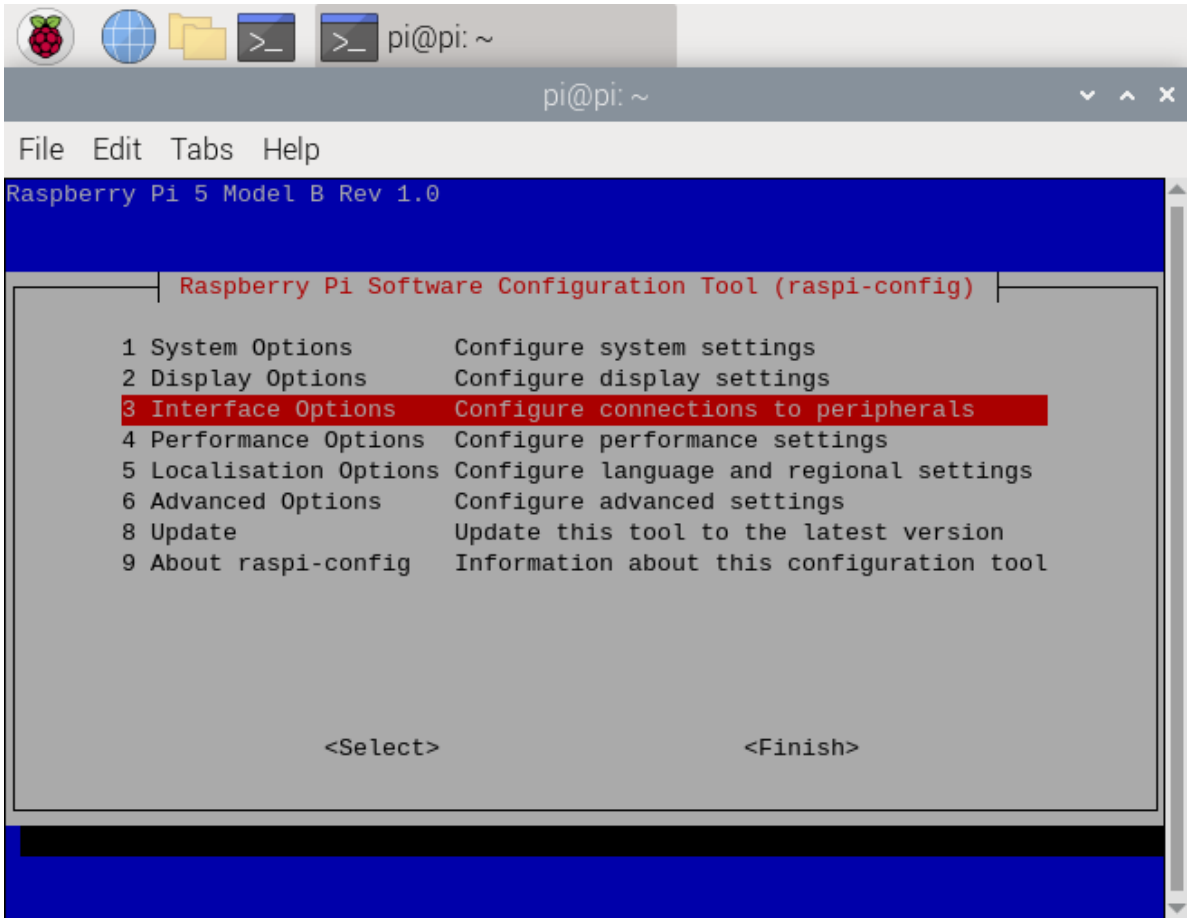
To enable the I2C port on your Raspberry Pi, follow these steps (skip if already enabled; if unsure, proceed with the instructions).

1. Log into your Raspberry Pi, open the Terminal, and enter the command below to access the Raspberry Pi Software Configuration Tool. (You can also access the terminal using SSH.)

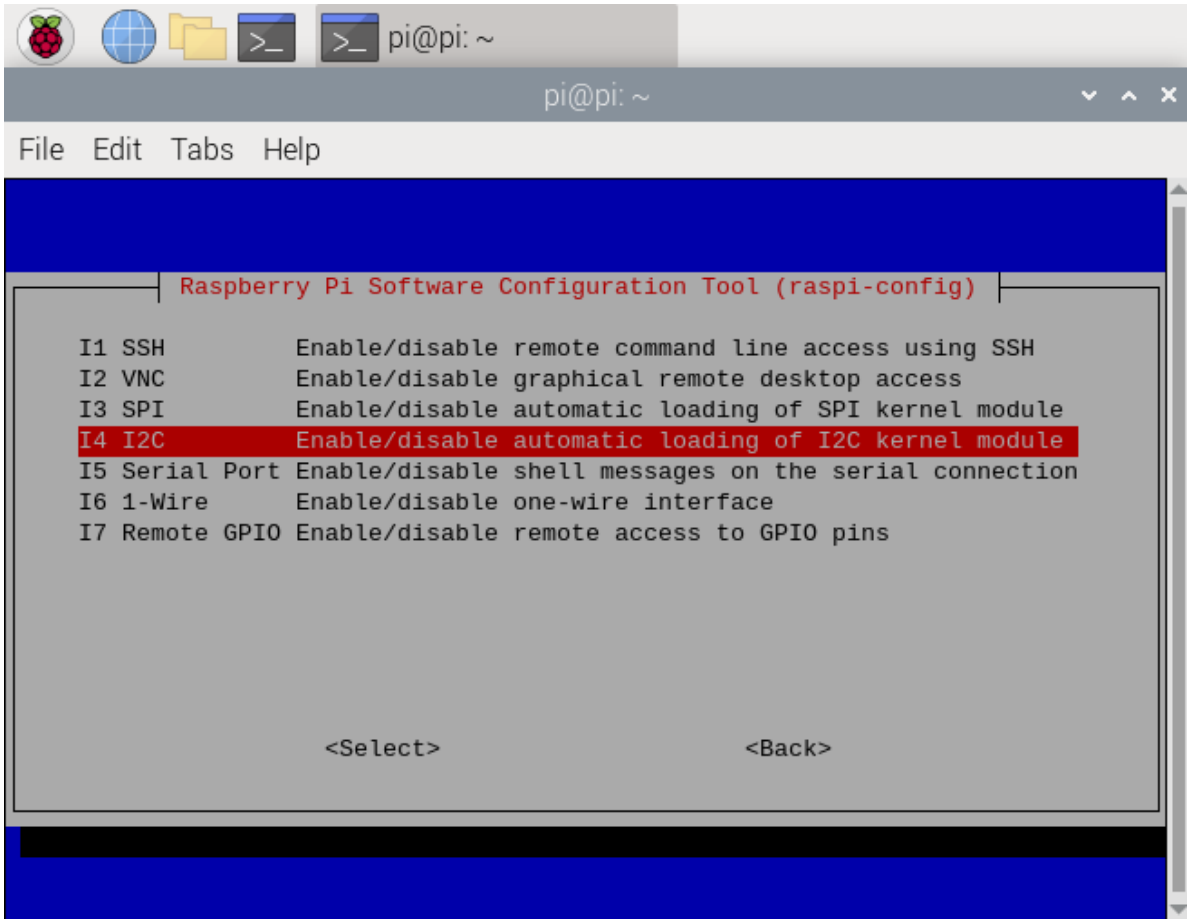
```
sudo raspi-config
```

2. Go to **Interfacing options**.

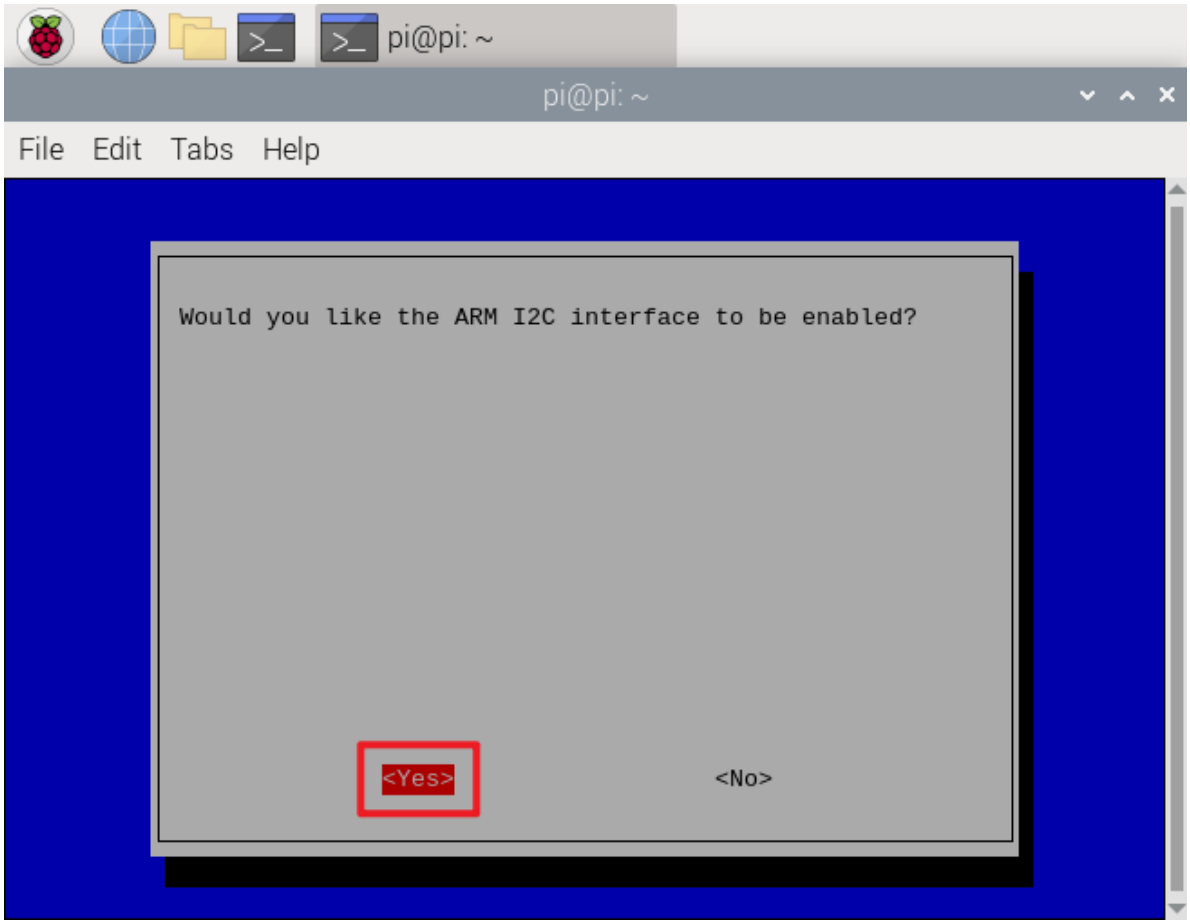
Note: Use the `up` and `down` arrow keys to move the highlighted selection between the options available. Pressing the `right` arrow key will jump out of the Options menu and take you to the `<Select>` and `<Finish>` buttons. Pressing `left` will take you back to the options. Alternatively, you can use the `Tab` key to switch between these.


pi@pi: ~
pi@pi: ~
File Edit Tabs Help
Raspberry Pi 5 Model B Rev 1.0
Raspberry Pi Software Configuration Tool (raspi-config)
1 System Options Configure system settings
2 Display Options Configure display settings
3 Interface Options Configure connections to peripherals
4 Performance Options Configure performance settings
5 Localisation Options Configure language and regional settings
6 Advanced Options Configure advanced settings
8 Update Update this tool to the latest version
9 About raspi-config Information about this configuration tool
<Select>
<Finish>

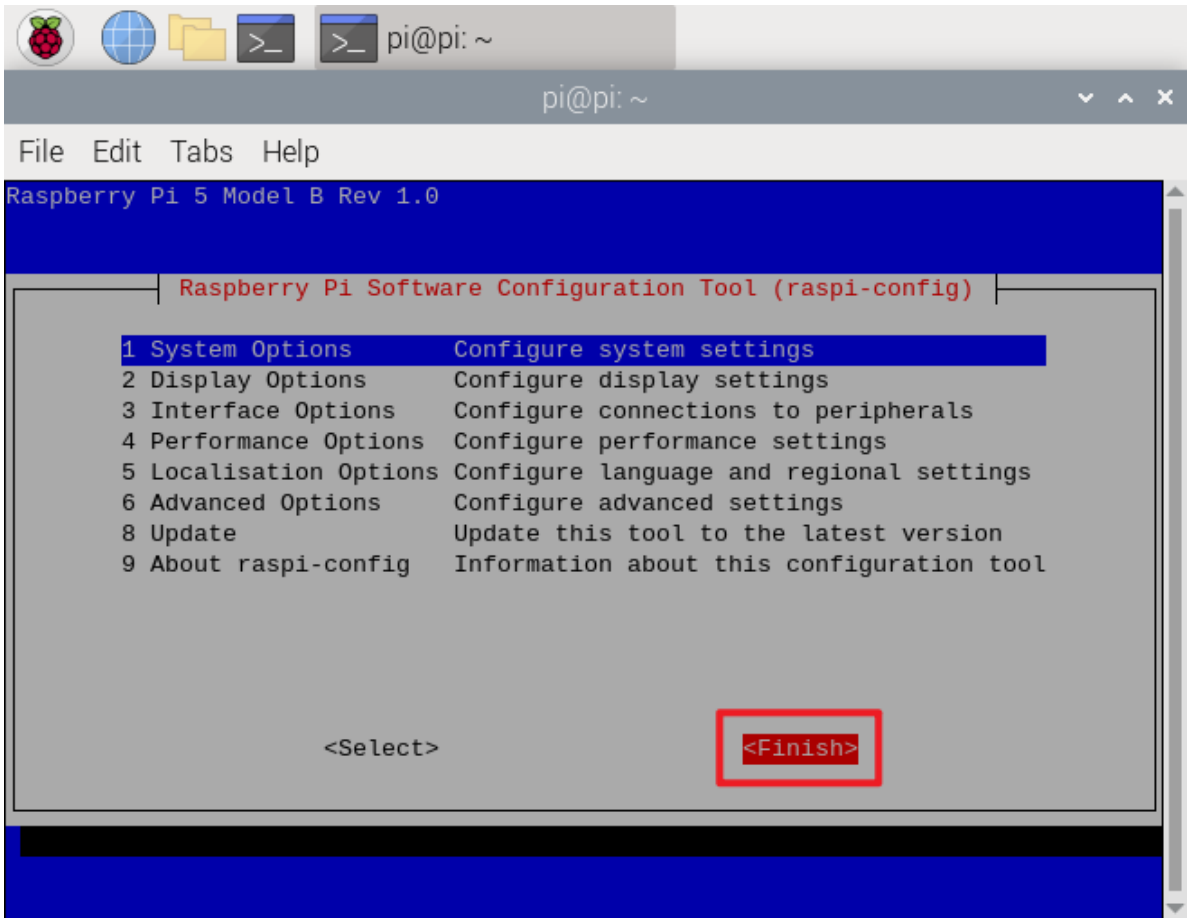
3. Select **I2C**.



4. Choose **<Yes>** to activate the I2C interface, then choose **<Ok>**.

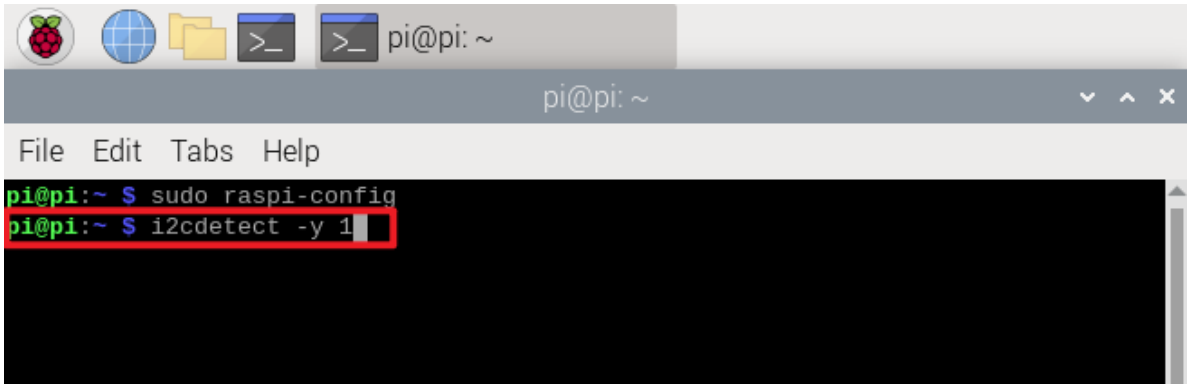


5. Select **<Finish>** to exit the Raspberry Pi Software Configuration Tool.

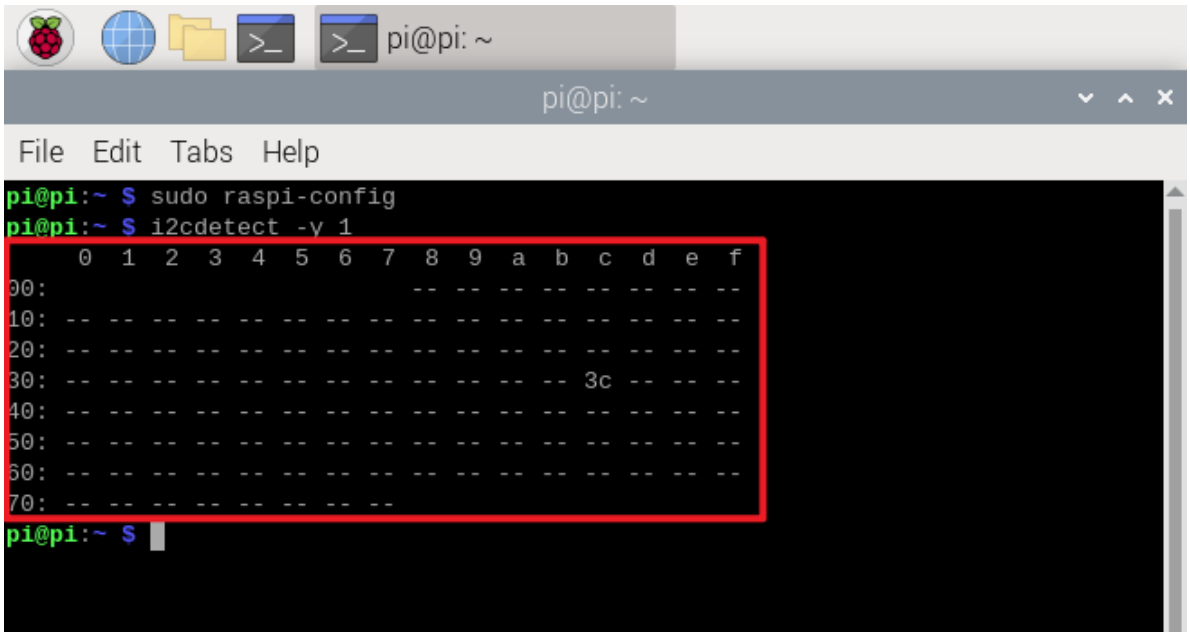


6. Verify the address of the connected I2C device using the following command.

```
i2cdetect -y 1
```



Addresses of any connected I2C devices will be shown.



1-Wire Configuration

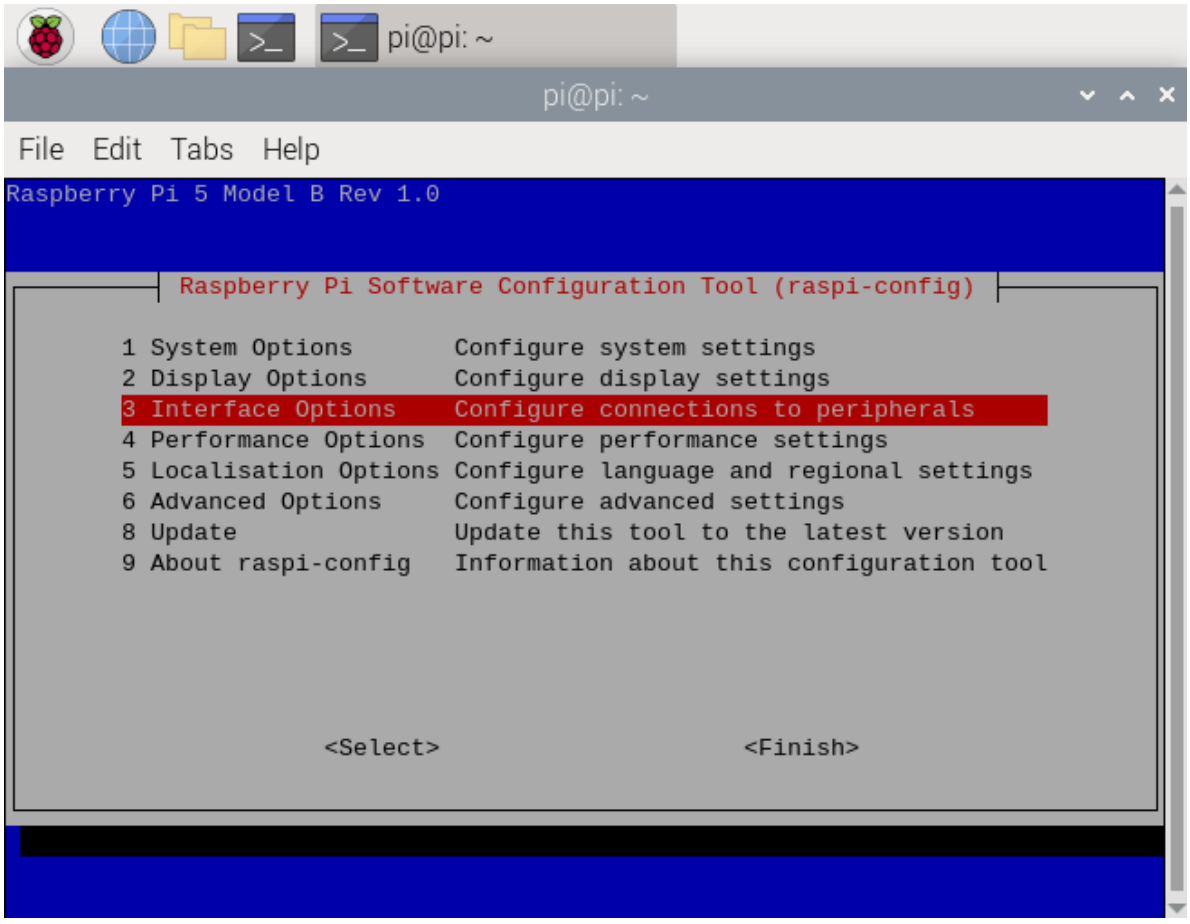
To enable the 1-Wire port on your Raspberry Pi, follow these steps (skip if already enabled; if unsure, proceed with the instructions).

1. Log into your Raspberry Pi, open the Terminal, and enter this command to access the Raspberry Pi Software Configuration Tool. (You can also access the terminal using SSH.)

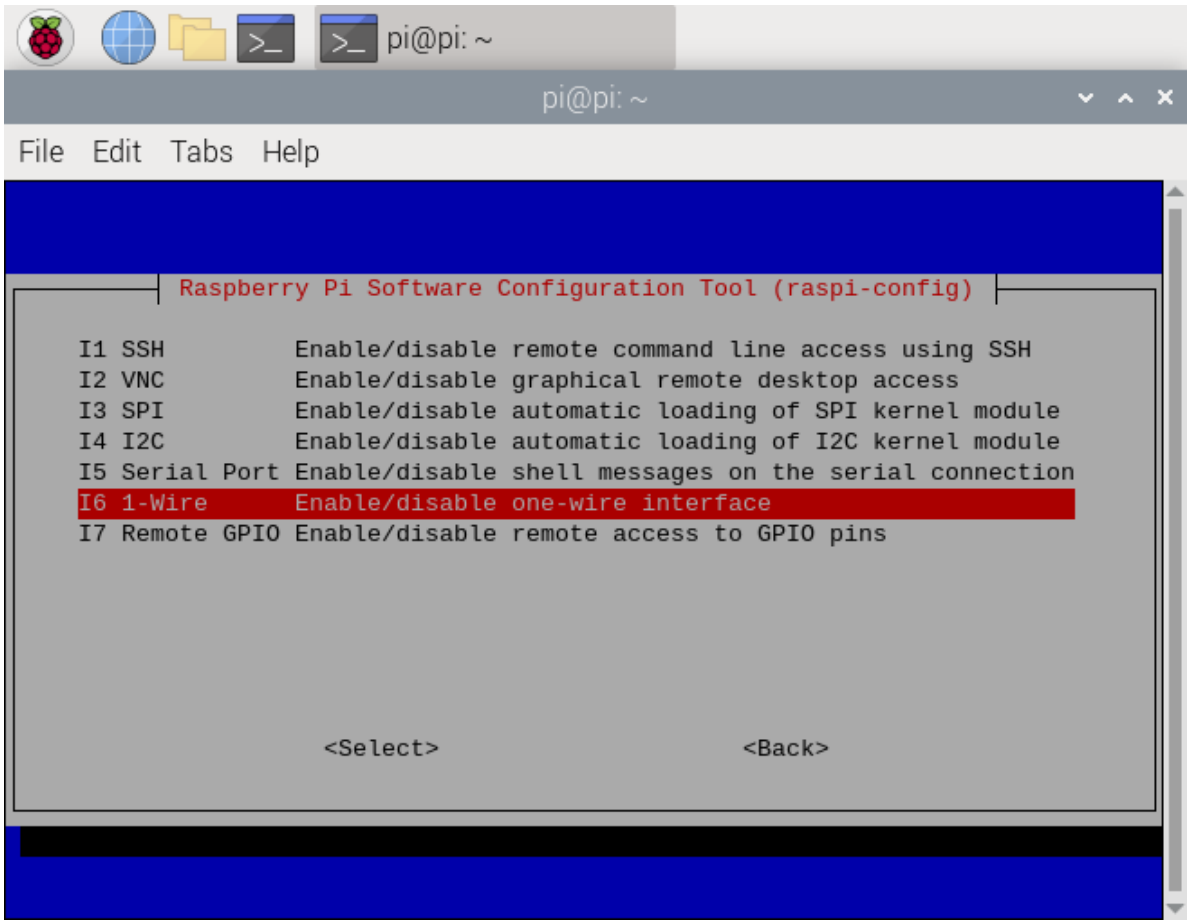
```
sudo raspi-config
```

2. Go to **Interfacing options**.

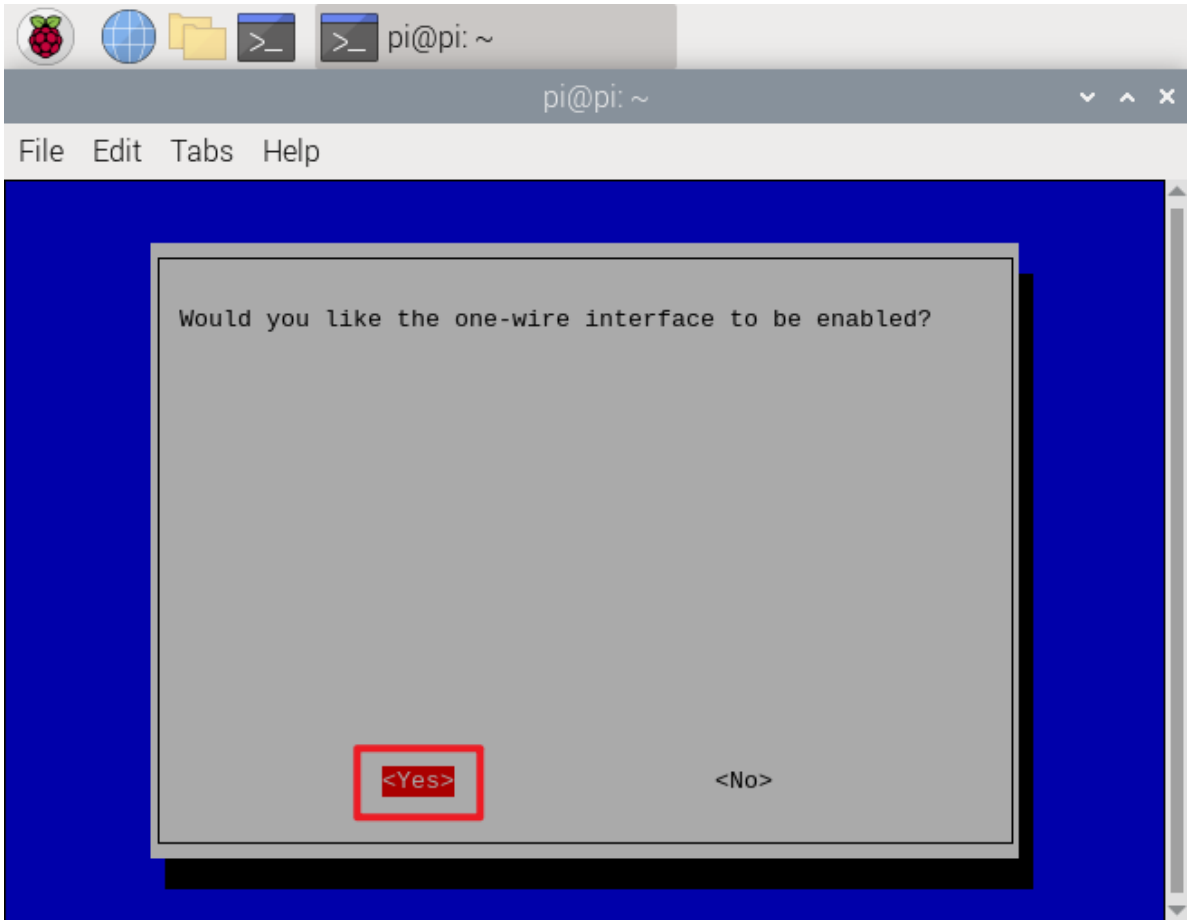
Note: Use the up and down arrow keys to move the highlighted selection between the options available. Pressing the `right` arrow key will jump out of the Options menu and take you to the <Select> and <Finish> buttons. Pressing `left` will take you back to the options. Alternatively, you can use the `Tab` key to switch between these.



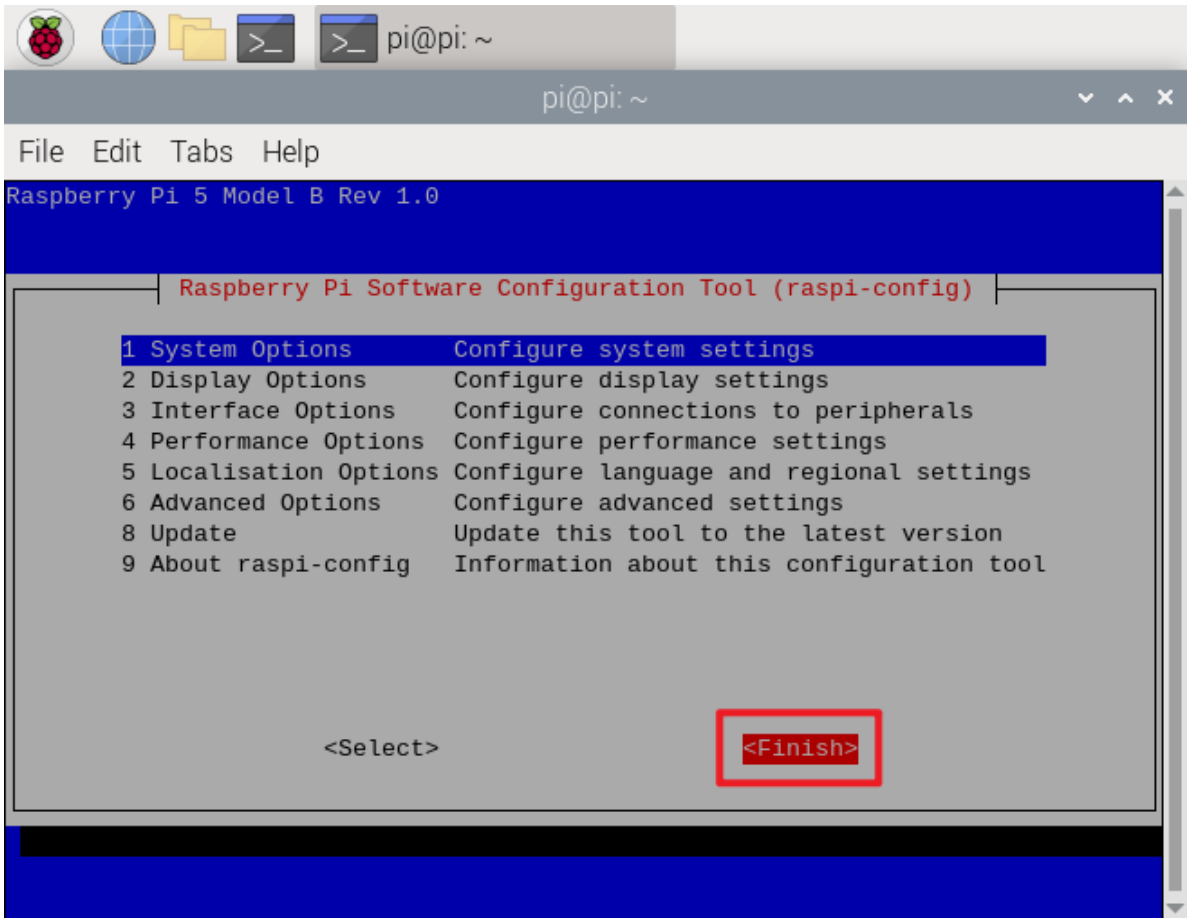
3. Select **1-Wire**.



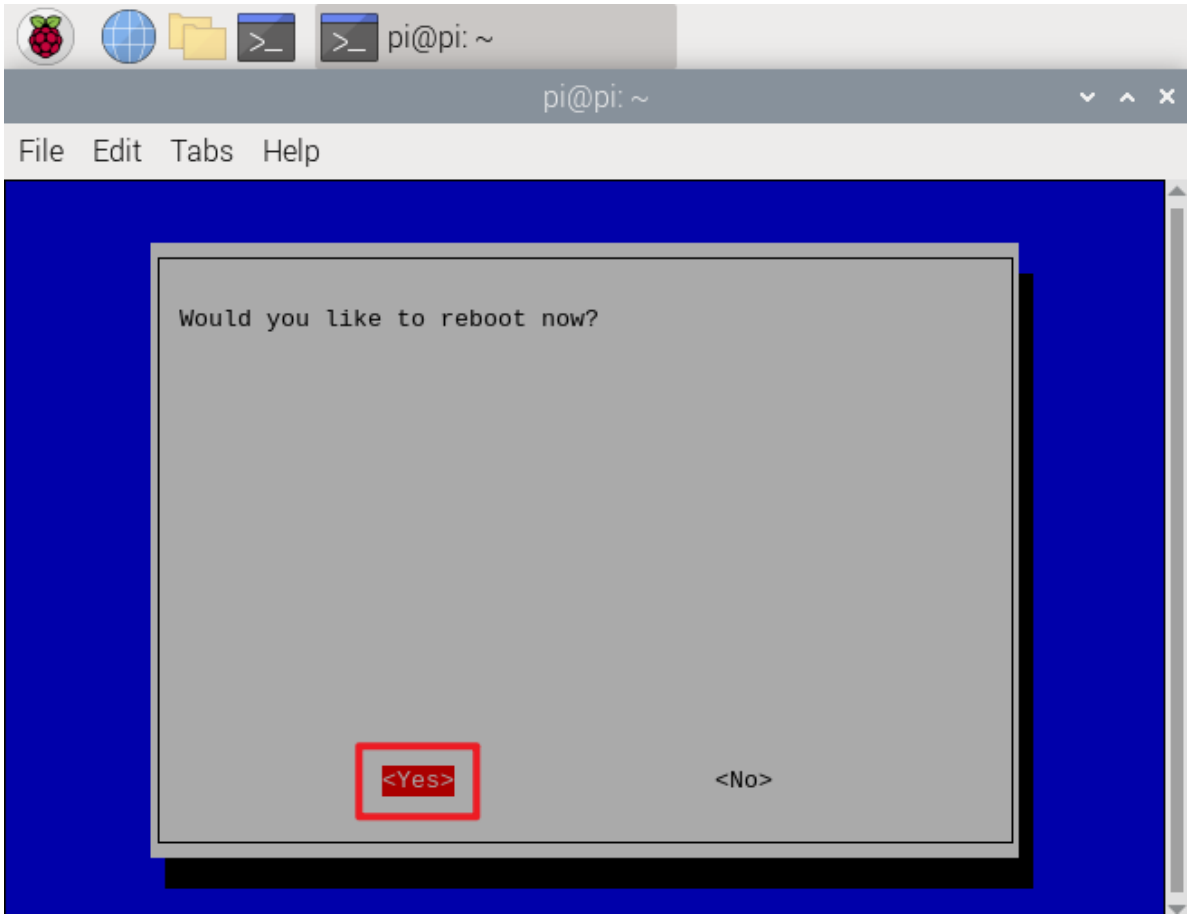
4. Choose **<Yes>** to activate the 1-Wire interface, then choose **<Ok>**.



5. Select **<Finish>** to exit the Raspberry Pi Software Configuration Tool.



6. Select **<yes>** to reboot the Raspberry Pi.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Insatll `Adafruit_Blinka` (CircuitPython) - Optional

For an enhanced experience with advanced modules, we recommend using the `Adafruit_Blinka` library, a key component of the CircuitPython environment. The unique feature of Blinka is its ability to enable code written for CircuitPython to run seamlessly and effortlessly on Linux computers like the Raspberry Pi.

This library makes it simpler to use complex modules like BMP280, VL53L0X, and OLED, streamlining your project development process. With CircuitPython, programming becomes more accessible, allowing you to focus on creating robust applications without needing extensive hardware knowledge.

Additionally, you'll gain the benefit of a large support community and a variety of resources to aid your learning and development.

We will guide you through the straightforward process of installing Adafruit_Blinka, setting the stage for you to quickly start working on your projects.

Update your Raspberry Pi and Python

Before installing Blinka, please use the following commands to ensure that your Raspberry Pi and Python versions are up to date:

```
sudo apt-get update
sudo apt-get upgrade
```

Setup Virtual Environment

Starting from Bookworm (OS version), packages installed using `pip` must be installed into a Python virtual environment using `venv`. A virtual environment is a secure container where you can install third-party modules without affecting or disrupting your system's Python.

The following command will create an "env" directory in your user directory (~) for the virtual Python environment.

```
cd ~
python -m venv env --system-site-packages
```

You will need to activate the virtual environment every time the Pi is rebooted. To activate it:

```
source env/bin/activate
```

You'll see that your prompt is now prepended with (env) to indicate that you're no longer using the system Python. Instead, you're using the version of Python contained inside your virtual environment. Any changes you make here won't cause problems for your system Python; nor will any new modules you install into your environment.

```
pi@pi:~ $ python -m venv env --system-site-packages
pi@pi:~ $ source env/bin/activate
(env) pi@pi:~ $
```

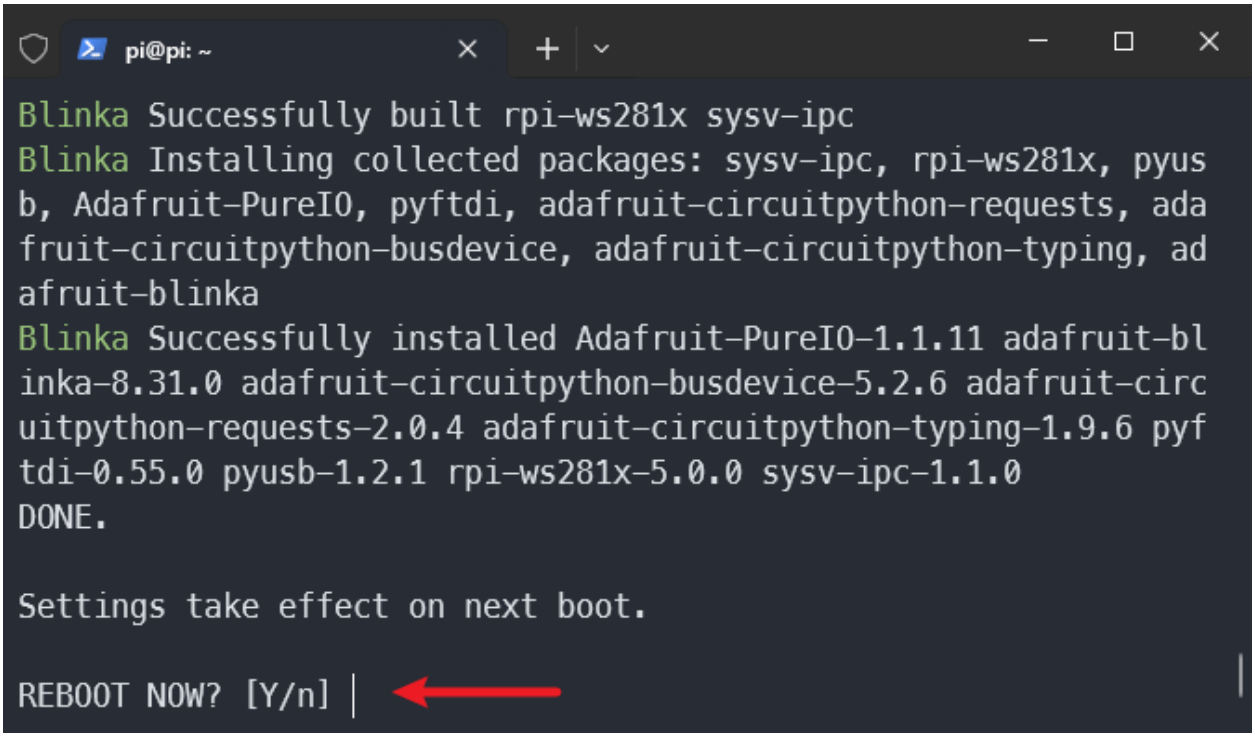
To deactivate, you can use `deactivate`, but leave it active for now.

Automatic Installation

When activated in the virtual environment (you will see `(env)` at the beginning of the terminal command), run the following code in order. This code will execute the installation script provided by adafruit and automatically complete the remaining installation steps.

```
pip3 install --upgrade adafruit-python-shell
pip3 install rpi-lgpio
wget https://raw.githubusercontent.com/adafruit/Raspberry-Pi-Installer-Scripts/master/
↪raspi-blinka.py
sudo -E env PATH=$PATH python3 raspi-blinka.py
```

It may take a few minutes to run. When it finishes, it will ask you if you would like to reboot. Press Enter directly to reboot, or if you want to reboot later, enter "n" and then press Enter. When you are ready, manually reboot your raspberry pi.



Once it reboots, the connection will close. After a couple of minutes, you can reconnect.

Blinka Test

Create a new file called `blinkatest.py` with nano or your favorite text editor and put the following in:

```
import board
import digitalio
import busio

print("Hello blinka!")

# Try to great a Digital input
```

(continues on next page)

(continued from previous page)

```
pin = digitalio.DigitalInOut(board.17)
print("Digital IO ok!")

# Try to create an I2C device
i2c = busio.I2C(board.SCL, board.SDA)
print("I2C ok!")

# Try to create an SPI device
spi = busio.SPI(board.SCLK, board.MOSI, board.MISO)
print("SPI ok!")

print("done!")
```

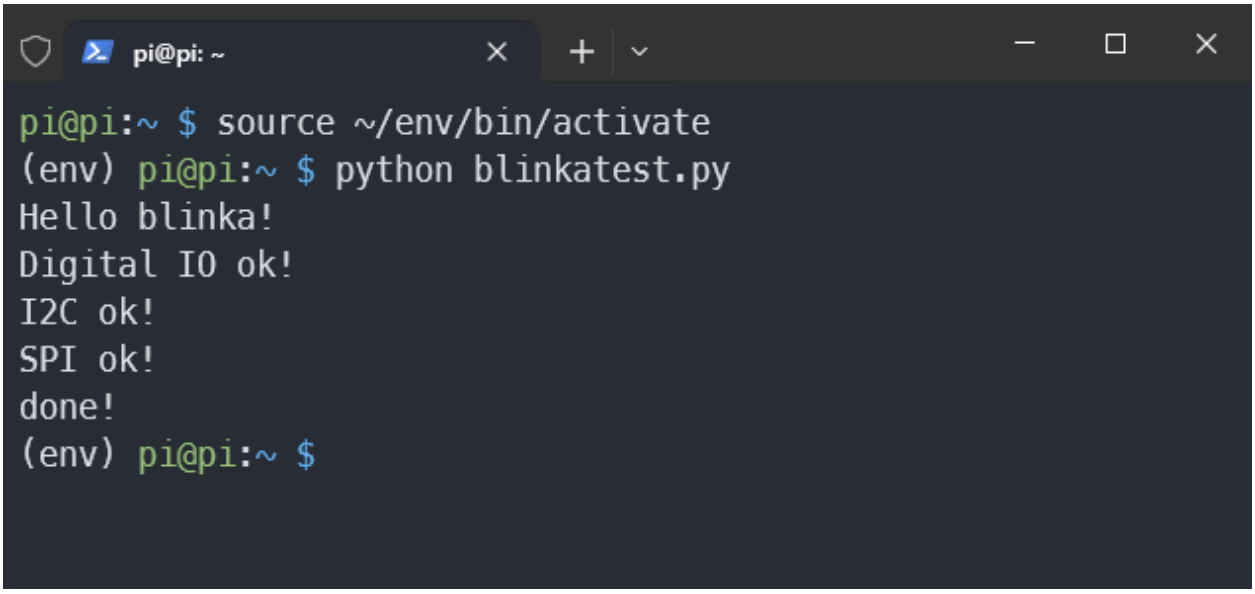
Before running the code, please make sure that you have activated the virtual python environment with blinka installed:

```
source ~/env/bin/activate
```

Then run the following command in the command line:

```
python blinkatest.py
```

You should see the following, indicating digital i/o, I2C and SPI all worked.



Reference

-
-

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.2 Lesson 01: Button Module

In this lesson, you will learn the basics of using a button with Raspberry Pi. We will show you how to connect a button to GPIO pin 17 and write a simple Python script to monitor its state. You'll learn how to program the Raspberry Pi to detect when the button is pressed and released, and respond with appropriate messages. This introductory project is an excellent way to get familiar with GPIO interaction and basic Python scripting, making it well-suited for beginners starting their journey in Raspberry Pi and hardware programming.

Required Components

In this project, we need the following components.

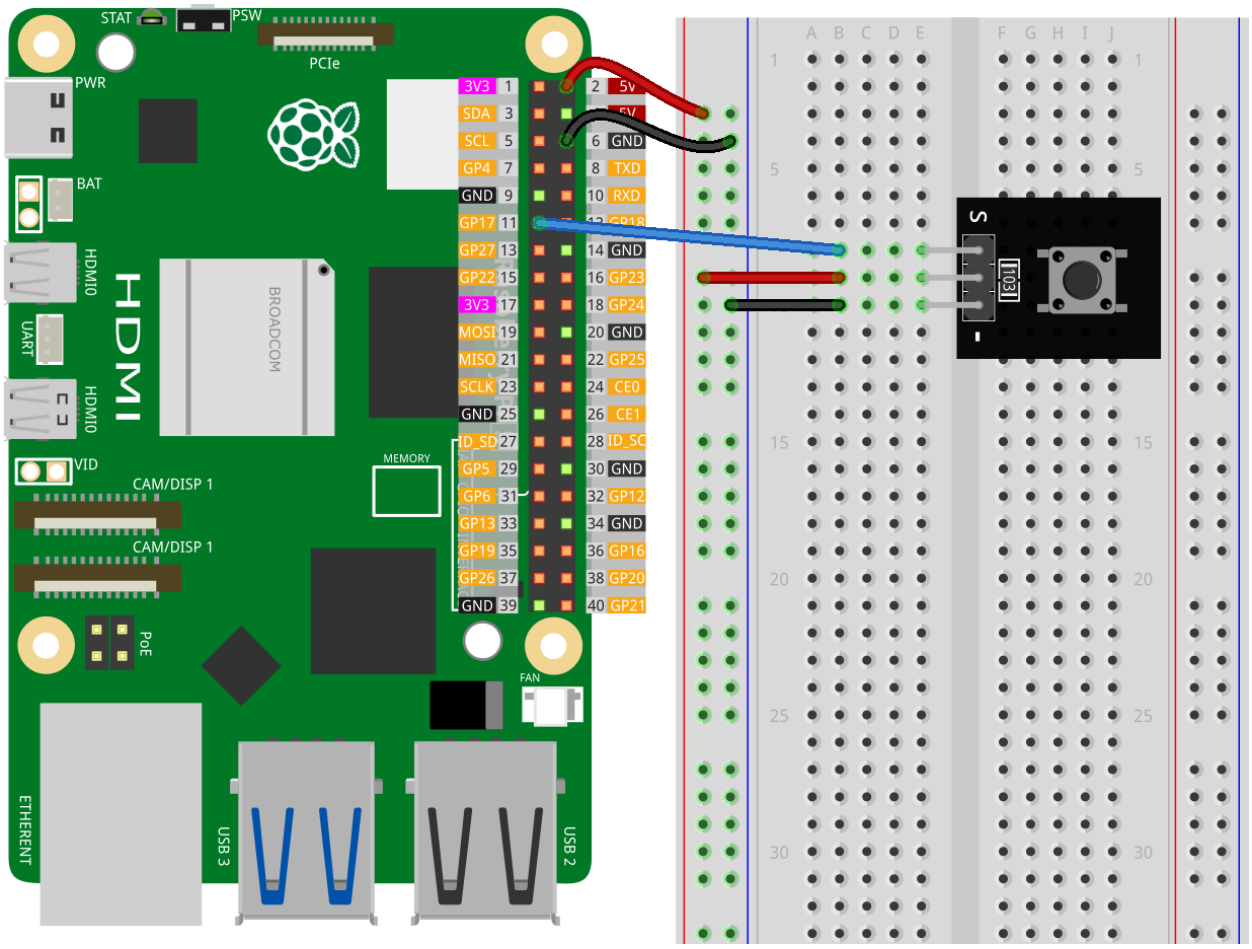
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Button Module	-
Breadboard	

Wiring



Code

```
from gpiozero import Button

# Initialize button connected to GPIO pin 17
button = Button(17)

# Continuously check the button state
while True:
   if button.is_pressed:
      print("Button is pressed")  # Print when button is pressed
   else:
      print("Button is not pressed")  # Print when button is not pressed
```

Code Analysis

1. Import Library

   Import the `Button` class from the `gpiozero` library for button control.

   ```
   from gpiozero import Button
   ```

2. Initialize the Button

   Create a `Button` object connected to GPIO pin 17.

   ```
   button = Button(17)
   ```

3. Monitor Button State Continuously

   Use a `while True` loop to continuously check the state of the button. If the button is pressed (`button.is_pressed`), it prints "Button is pressed". Otherwise, it prints "Button is not pressed".

   ```
   while True:
       if button.is_pressed:
           print("Button is pressed")
       else:
           print("Button is not pressed")
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.3 Lesson 02: Capacitive Soil Moisture Module

Note: The Raspberry Pi does not have analog input capabilities, so it needs a module like the *PCF8591 ADC DAC Converter Module* to read analog signals for processing.

In this tutorial, we'll explore how to monitor soil moisture levels using a Raspberry Pi. You'll learn to set up a Capacitive Soil Moisture Sensor with the PCF8591 module for analog-to-digital conversion and use Python to continuously track the soil's moisture content. This project is a hands-on introduction to sensors, ADCs (analog-to-digital converters), and real-time data monitoring on the Raspberry Pi.

Required Components

In this project, we need the following components.

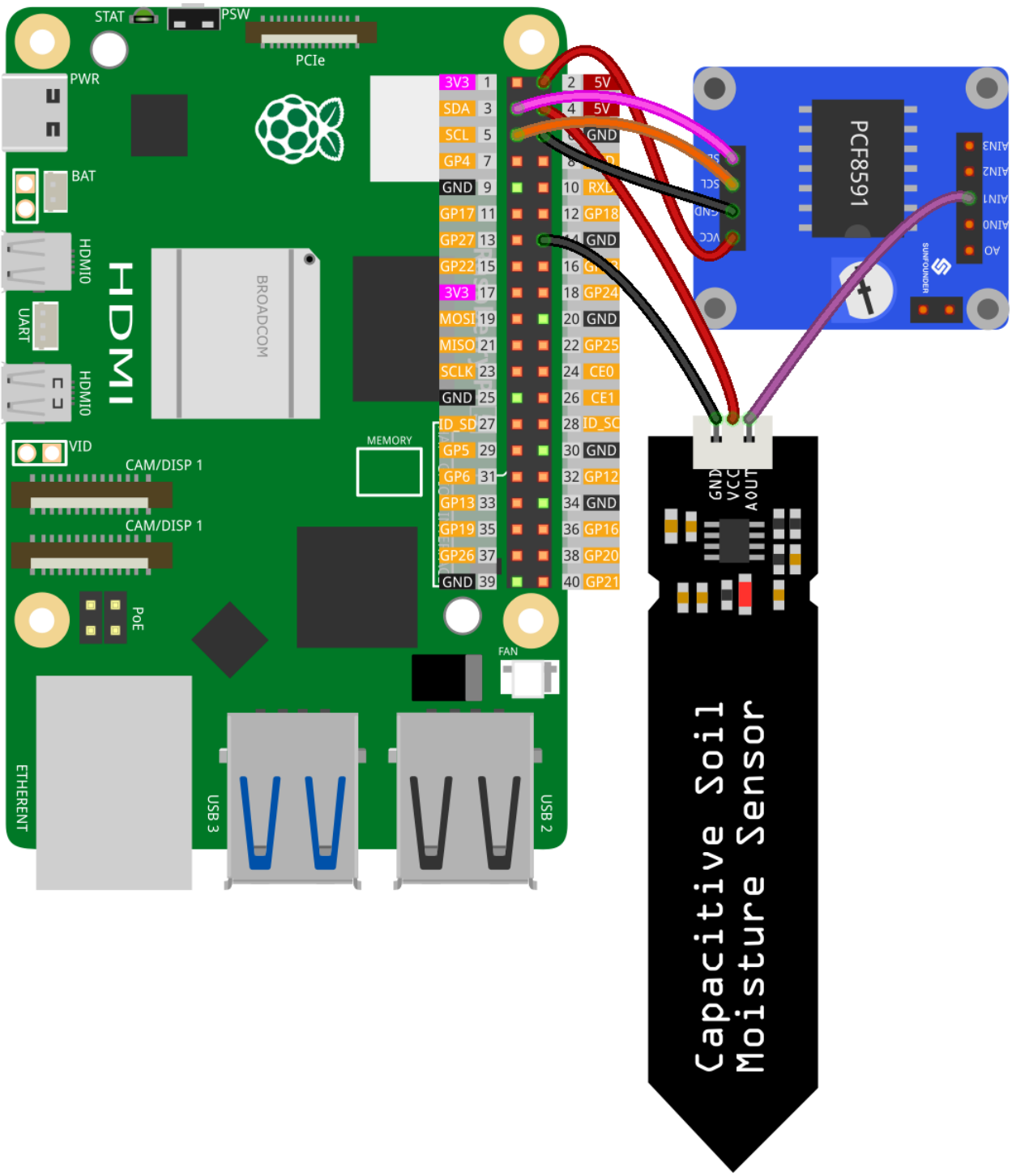
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Capacitive Soil Moisture Module	
PCF8591 ADC DAC Converter Module	

Wiring


STAT
PSW
PCIe
PWR
BAT
HDMI0
UART
HDMI0
HDMI
VID
CAM/DISP 1
CAM/DISP 1
PoE
BROADCOM
MEMORY
FAN
ETHERENT
USB 3
USB 2
3V3 1
2 5V
SDA 3
4 5V
SCL 5
GND
GP4 7
GND 9
10 RXD
GP17 11
12 GP18
GP27 13
GND
GP22 15
16
3V3 17
18 GP24
MOSI 19
20 GND
MISO 21
22 GP25
SCLK 23
24 CE0
GND 25
26 CE1
ID_SD 27
28 ID_SC
GP5 29
30 GND
GP6 31
32 GP12
GP13 33
34 GND
GP19 35
36 GP16
GP26 37
38 GP20
GND 39
40 GP21
PCF8591
AIN3
AIN2
AIN1
AIN0
AO
SUNFOUNDER
SCL
GND
VCC
GND
VCC
AOUT
Capacitive Soil
Moisture Sensor

Code

```
import PCF8591 as ADC  # Import PCF8591 module
import time  # Import time for delay

ADC.setup(0x48)  # Initialize PCF8591 at address 0x48

try:
    while True:  # Continuously read and print moisture level
        print(ADC.read(1))  # Read from Soil Moisture Sensor at AIN1
        time.sleep(0.2)  # Delay of 0.2 seconds
except KeyboardInterrupt:
    print("Exit")  # Exit on CTRL+C
```

Code Analysis

1. **Import Libraries**:

   This section imports necessary Python libraries. The `PCF8591` library is used for interacting with the PCF8591 module, and `time` is for implementing delays in the code.

   ```
   import PCF8591 as ADC  # Import PCF8591 module
   import time  # Import time for delay
   ```

2. **Initialize PCF8591 Module**:

   Here, the PCF8591 module is initialized. The address `0x48` is the I^2C address of the PCF8591 module. This is necessary for the Raspberry Pi to communicate with the module.

   ```
   ADC.setup(0x48)  # Initialize PCF8591 at address 0x48
   ```

3. **Main Loop and Reading Data**:

   The `try` block includes a continuous loop that consistently reads data from the capacitive soil moisture module. The `ADC.read(1)` function captures the analog input from the sensor connected to channel 1 (AIN1) of the PCF8591 module. Incorporating a `time.sleep(0.2)` creates a 0.2-second pause between each reading. This not only helps in reducing CPU usage on the Raspberry Pi by avoiding excessive data processing demands, but also prevents the terminal from being overrun with rapidly scrolling information, making it easier to monitor and analyze the output.

   ```
   try:
       while True:  # Continuously read and print moisture level
           print(ADC.read(1))  # Read from Soil Moisture Sensor at AIN1
           time.sleep(0.2)  # Delay of 0.2 seconds
   ```

4. **Handling Keyboard Interrupt**:

   The `except` block is designed to catch a KeyboardInterrupt (like pressing CTRL+C). When this interrupt occurs, the script prints "exit" and stops running. This is a common way to gracefully exit a continuously running script in Python.

   ```
   except KeyboardInterrupt:
       print("exit")  # Exit on CTRL+C
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.4 Lesson 03: Flame Sensor Module

In this lesson, you will learn to use a flame sensor with Raspberry Pi for fire detection. We'll show you how to connect the flame sensor to GPIO17 and write a Python script to read its output. You'll learn to identify when the sensor detects a flame, indicated by a change in the sensor's state. This practical project introduces you to the basics of sensor interfacing and Python coding on the Raspberry Pi, suitable for beginners interested in building safety-related projects.

Required Components

In this project, we need the following components.

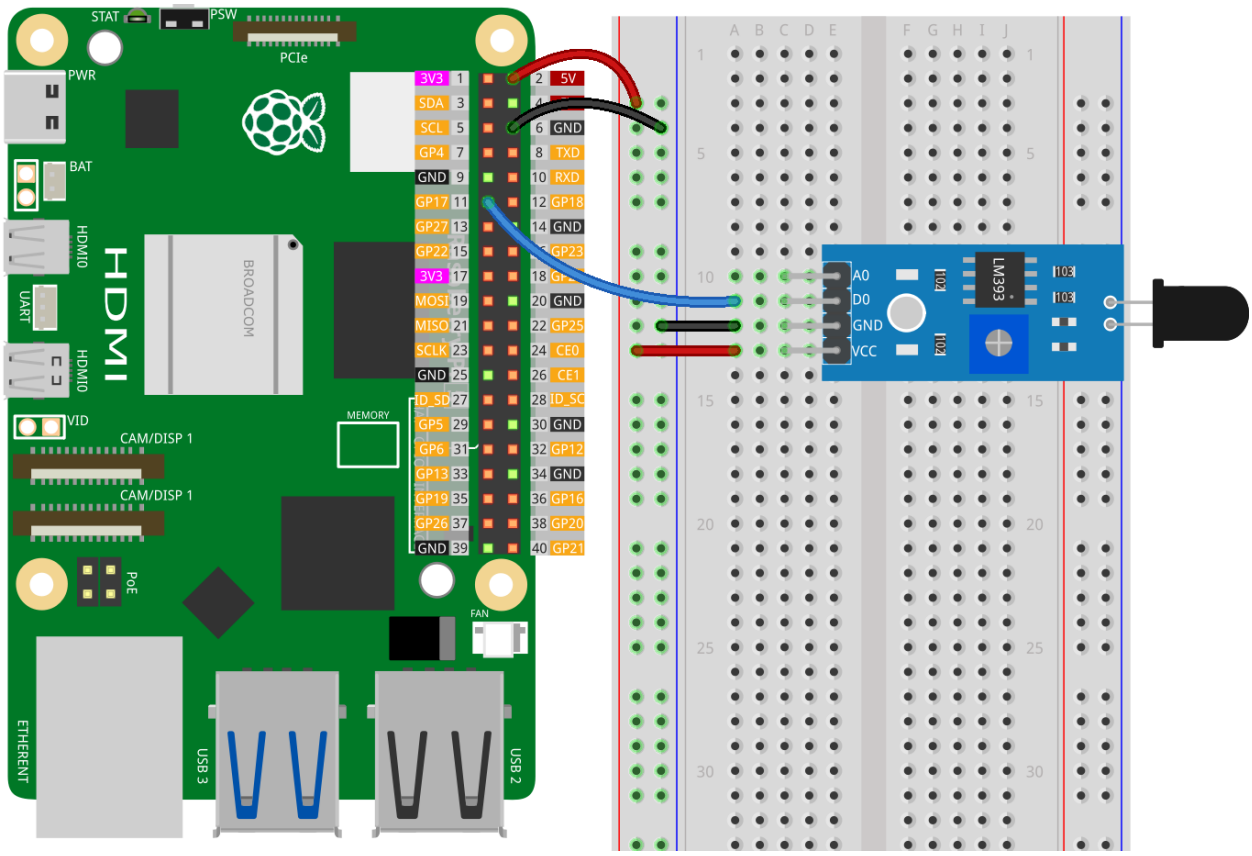
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Flame Sensor Module	
Breadboard	

Wiring



Code

```
from gpiozero import InputDevice
import time

# Connect the digital output of the flame sensor to GPIO17 on the Raspberry Pi
flame_sensor = InputDevice(17)

# Continuous loop to read from the sensor
while True:
    # Check if the sensor is active (no flame detected)
    if flame_sensor.is_active:
        print("No flame detected.")
    else:
        # When the sensor is inactive (flame detected)
        print("Flame detected!")
    # Wait for 1 second before reading the sensor again
    time.sleep(1)
```

Code Analysis

1. Importing Libraries

   The script starts by importing the necessary classes from the gpiozero library and the time module from Python's standard library.

```
from gpiozero import InputDevice
import time
```

2. Initializing the Flame Sensor

   An `InputDevice` object named `flame_sensor` is created, representing the flame sensor connected to GPIO pin 17 of the Raspberry Pi. This setup assumes that the digital output of the flame sensor is connected to GPIO17.

```
flame_sensor = InputDevice(17)
```

3. Continuous Reading Loop

   - The script uses a `while True:` loop to continuously read the sensor's data. This loop will run indefinitely.
   - Inside the loop, an `if` statement checks the state of the flame sensor using the `is_active` property.
   - If `flame_sensor.is_active` is `True`, it indicates no flame is detected, and "No flame detected." is printed.
   - If `flame_sensor.is_active` is `False`, it indicates a flame is detected, and "Flame detected!" is printed.
   - The `time.sleep(1)` command pauses the loop for 1 second between each sensor reading, preventing the script from overloading the CPU.

```
while True:
    if flame_sensor.is_active:
        print("No flame detected.")
    else:
        print("Flame detected!")
    time.sleep(1)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.5 Lesson 04: Gas Sensor Module (MQ-2)

In this lesson, you will learn to use the MQ2 gas sensor with Raspberry Pi for gas detection. The course covers connecting the MQ2 sensor to the GPIO17 pin and programming the Raspberry Pi in Python to read the sensor output. You'll understand how to detect gas presence, with a low signal from the sensor indicating the detection of gas. This project offers a practical introduction to sensor usage and Python scripting on the Raspberry Pi, ideal for beginners interested in environmental monitoring and safety applications.

Required Components

In this project, we need the following components.

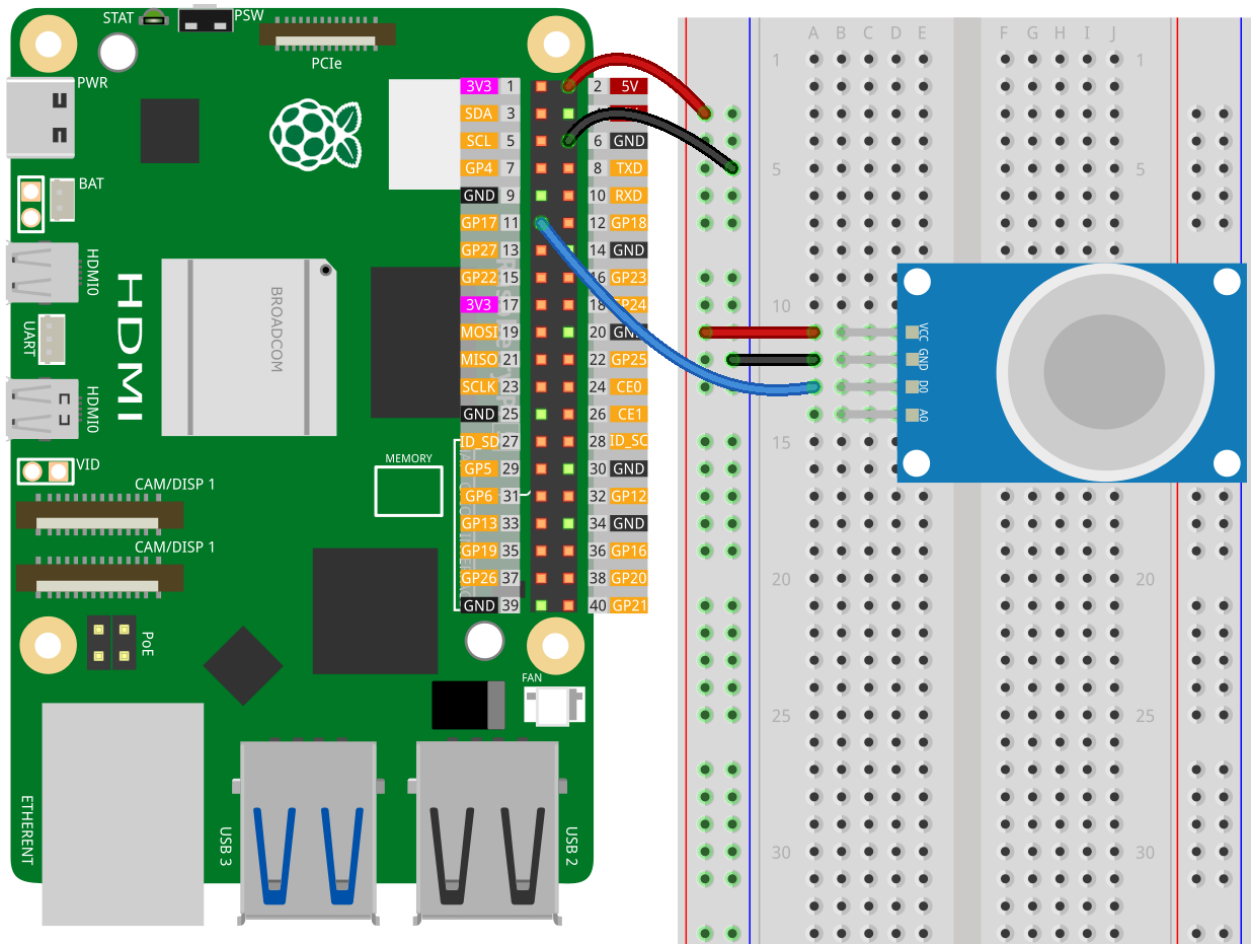
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Gas/Smoke Sensor Module (MQ2)	
Breadboard	

Wiring



Code

```
from gpiozero import DigitalInputDevice
import time

# Initialize MQ2 sensor on GPIO17
mq2 = DigitalInputDevice(17)

while True:
    # Detect gas presence (LOW signal indicates gas)
    if mq2.value == 0:
        print("Gas detected!")
    else:
        print("No gas detected.")

    # Delay between readings
    time.sleep(1)
```

Code Analysis

1. Importing Libraries

```
from gpiozero import DigitalInputDevice
import time
```

This section imports necessary libraries. `gpiozero` is used for interacting with the GPIO pins of the Raspberry Pi, and `time` is used for handling time-related tasks such as delays.

2. Initializing the MQ2 Sensor

```
mq2 = DigitalInputDevice(17)
```

Here, the MQ2 sensor is initialized as a digital input device on GPIO pin 17 of the Raspberry Pi. The `DigitalInputDevice` class from gpiozero is used for this purpose.

3. Infinite Loop for Sensor Reading

```
while True:
    if mq2.value == 0:
        print("Gas detected!")
    else:
        print("No gas detected.")
    time.sleep(1)
```

In this segment:

Note: The DO pin on the MQ-2 sensor module indicates the presence of combustible gases. When the gas concentration exceeds the threshold value (as set by the potentiometer on the module), D0 becomes LOW; otherwise, it remains HIGH.

- An infinite loop is created using `while True`. This loop will continue to run until the program is manually stopped.
- Inside the loop, the value of the MQ2 sensor is checked using `mq2.value`. If the value is 0, it indicates the presence of gas, and "Gas detected!" is printed. Otherwise, "No gas detected." is printed.
- `time.sleep(1)` creates a delay of 1 second between each reading, reducing the frequency of the sensor checks and the output messages.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.6 Lesson 05: Gyroscope & Accelerometer Module (MPU6050)

In this lesson, you'll learn how to interface the Raspberry Pi with the MPU6050, a sensor that integrates a 3-axis gyroscope and accelerometer. You'll explore how to measure acceleration, orientation, and rotation. This project offers hands-on experience with reading sensor data, utilizing Python for hardware interaction, and grasping I2C communication fundamentals. You will also learn to continuously capture acceleration in three axes, rotational speed, and temperature from the sensor. It's an ideal starting point for beginners eager to delve into sensors and motion tracking using the Raspberry Pi.

Required Components

In this project, we need the following components.

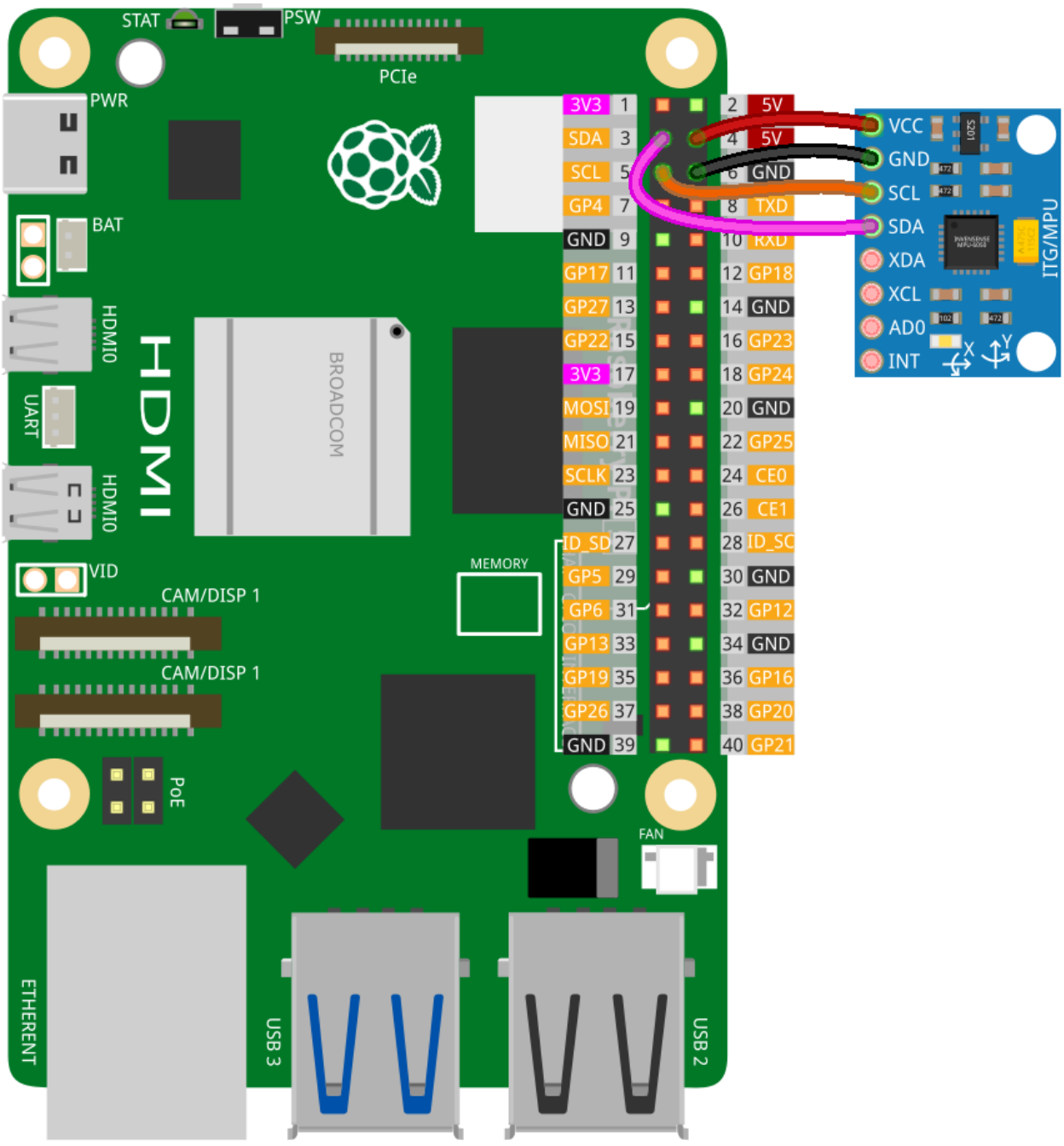
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Gyroscope & Accelerometer Module (MPU6050)	

Wiring


STAT
PSW
PCIe
PWR
BAT
HDMI0
UART
HDMI0
HDMI
BROADCOM
VID
CAM/DISP 1
CAM/DISP 1
MEMORY
PoE
FAN
ETHERENT
USB 3
USB 2
3V3 1
2 5V
SDA 3
4 5V
SCL 5
6 GND
GP4 7
8 TXD
GND 9
10 RXD
GP17 11
12 GP18
GP27 13
14 GND
GP22 15
16 GP23
3V3 17
18 GP24
MOSI 19
20 GND
MISO 21
22 GP25
SCLK 23
24 CE0
GND 25
26 CE1
ID_SD 27
28 ID_SC
GP5 29
30 GND
GP6 31
32 GP12
GP13 33
34 GND
GP19 35
36 GP16
GP26 37
38 GP20
GND 39
40 GP21
VCC
GND
SCL
SDA
XDA
XCL
AD0
INT
ITG/MPU

Code

```
# Import the mpu6050 class and sleep function from respective modules.
from mpu6050 import mpu6050
from time import sleep

# Initialize the MPU-6050 sensor with the I2C address 0x68.
sensor = mpu6050(0x68)

# Infinite loop to continuously read data from the sensor.
while True:
    # Retrieve accelerometer data from the sensor.
    accel_data = sensor.get_accel_data()
    # Retrieve gyroscope data from the sensor.
    gyro_data = sensor.get_gyro_data()
    # Retrieve temperature data from the sensor.
    temp = sensor.get_temp()

    # Print accelerometer data.
    print("Accelerometer data")
    print("x: " + str(accel_data['x']))
    print("y: " + str(accel_data['y']))
    print("z: " + str(accel_data['z']))

    # Print gyroscope data.
    print("Gyroscope data")
    print("x: " + str(gyro_data['x']))
    print("y: " + str(gyro_data['y']))
    print("z: " + str(gyro_data['z']))

    # Print the temperature in Celsius.
    print("Temp: " + str(temp) + " C")

    # Pause for 0.5 seconds before the next read cycle.
    sleep(0.5)
```

Code Analysis

1. Import Statements

   The `mpu6050` class is imported from the `mpu6050` library, and the `sleep` function is imported from the `time` module. These imports are necessary for interacting with the MPU-6050 sensor and introducing delays in the code.

   For more information about the `mpu6050` library, please visit .

   ```
   from mpu6050 import mpu6050
   from time import sleep
   ```

2. Sensor Initialization

   An instance of the `mpu6050` class is created with the I2C address 0x68 (the default address of the MPU-6050 sensor). This step initializes the sensor for data reading.

```
sensor = mpu6050(0x68)
```

3. Infinite Loop for Continuous Reading

   An infinite loop (`while True`) is used to continuously read data from the sensor. This is a common practice for sensor-based applications where constant monitoring is required.

```
while True:
```

4. Reading Sensor Data

   Inside the loop, data from the accelerometer, gyroscope, and temperature sensor is read using the `get_accel_data`, `get_gyro_data`, and `get_temp` methods of the `mpu6050` class instance. These methods return the sensor data in a user-friendly format.

```
accel_data = sensor.get_accel_data()
gyro_data = sensor.get_gyro_data()
temp = sensor.get_temp()
```

5. Printing Sensor Data

   The retrieved data is then printed out. Accelerometer and gyroscope data are accessed as dictionary values (x, y, z axes), and temperature is directly printed as a Celsius value.

```
print("Accelerometer data")
print("x: " + str(accel_data['x']))
print("y: " + str(accel_data['y']))
print("z: " + str(accel_data['z']))

print("Gyroscope data")
print("x: " + str(gyro_data['x']))
print("y: " + str(gyro_data['y']))
print("z: " + str(gyro_data['z']))

print("Temp: " + str(temp) + " C")
```

6. Delay Between Readings

   Finally, a half-second delay is introduced using `sleep(0.5)`. This delay is crucial to prevent overwhelming the Raspberry Pi with continuous data readings.

```
sleep(0.5)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.7 Lesson 06: Hall Sensor Module

Note: The Raspberry Pi does not have analog input capabilities, so it needs a module like the *PCF8591 ADC DAC Converter Module* to read analog signals for processing.

In this lesson, we will learn how to use a Raspberry Pi to read from a hall sensor module. You will learn how to connect a photoresistor module to the PCF8591 for analog-to-digital conversion and monitor its output in real-time using Python. Additionally, you will explore reading analog values and interpreting them to detect the presence and type of magnetic poles.

Required Components

In this project, we need the following components.

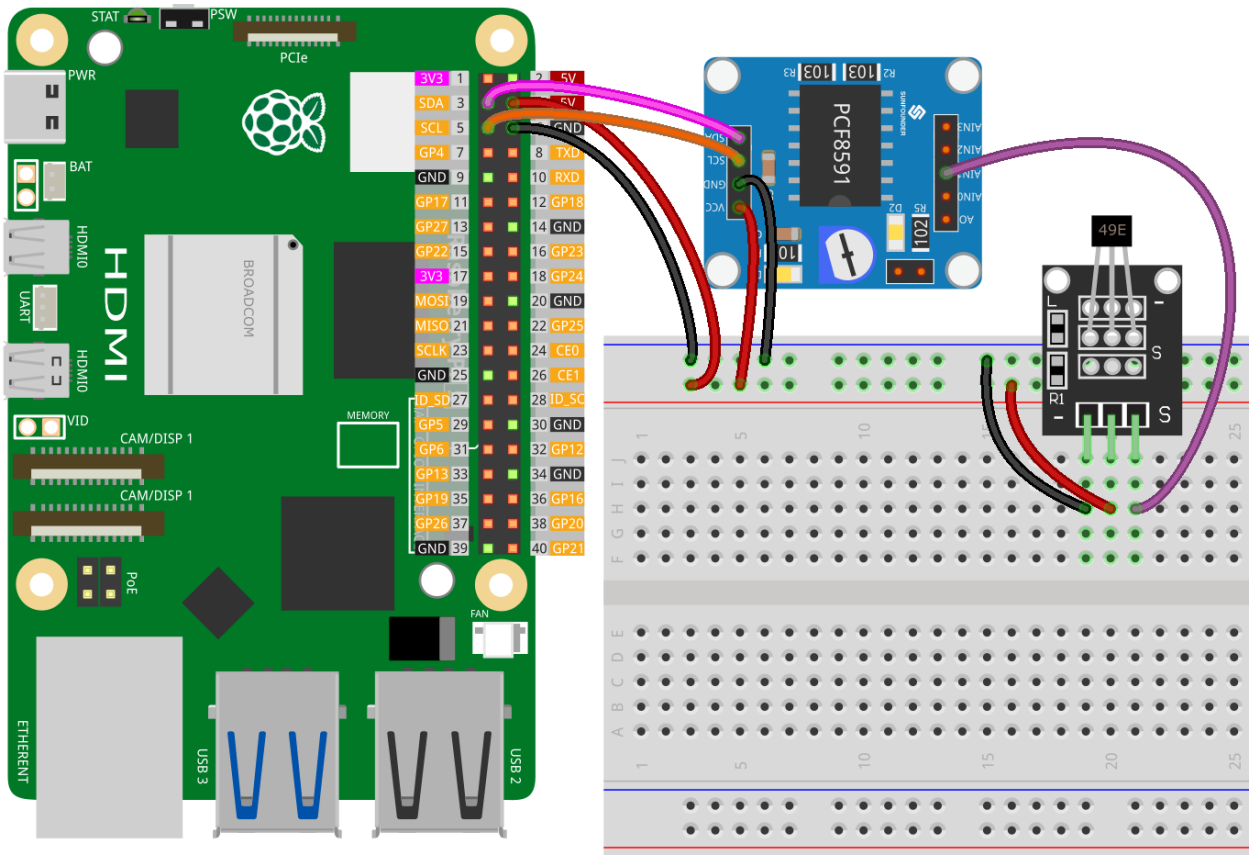
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Hall Sensor Module	-
PCF8591 ADC DAC Converter Module	
Breadboard	

Wiring



Code

```
import PCF8591 as ADC  # Import PCF8591 module
import time  # Import time for delay

ADC.setup(0x48)  # Initialize PCF8591 at address 0x48

try:
    while True:  # Continuously read and print
        sensor_value = ADC.read(1) # Read from hall sensor module at AIN1
        print(sensor_value,end="")  # Print the sensor raw data

        # Determine the polarity of the magnet
        if sensor_value >= 180:
            print(" - South pole detected")   # Determined as South pole.
        elif sensor_value <= 80:
            print(" - North pole detected")   # Determined as North pole.

        time.sleep(0.2)  # Wait for 0.2 seconds before the next read

except KeyboardInterrupt:
    print("Exit")  # Exit on CTRL+C
```

Code Analysis

1. **Import Libraries**:

```
import PCF8591 as ADC  # Import PCF8591 module
import time  # Import time for delay
```

This imports necessary libraries. `PCF8591` is used to interact with the ADC module, and `time` is for implementing delays in the loop.

2. **Initialize ADC Module**:

```
ADC.setup(0x48)  # Initialize PCF8591 at address 0x48
```

Sets up the PCF8591 module. `0x48` is the I2C address of the PCF8591 module. This line prepares the Raspberry Pi to communicate with the module.

3. **Main Loop for Reading Sensor Data**:

```
try:
    while True:  # Continuously read and print
        sensor_value = ADC.read(1) # Read from hall sensor module at AIN1
        print(sensor_value, end="")  # Print the sensor raw data
```

In this loop, `sensor_value` is read continuously from the Hall sensor (connected to AIN1 on the PCF8591). The `print` statement outputs the raw sensor data.

4. **Determine Magnet Polarity**:

```
# Determine the polarity of the magnet
if sensor_value >= 180:
    print(" - South pole detected")   # Determined as South pole.
elif sensor_value <= 80:
    print(" - North pole detected")   # Determined as North pole.
```

Here, the code determines the polarity of the magnet. If `sensor_value` is 180 or higher, it is identified as the South pole. If it is 80 or lower, it is considered the North pole. You need to modify these two threshold values based on your actual measurement results.

The Hall sensor module is equipped with a 49E linear Hall effect sensor, which can measure the polarity of the magnetic field's north and south poles as well as the relative strength of the magnetic field. If you place a magnet's south pole near the side marked with 49E (the side with text engraved on it), the value read by the code will increase linearly in proportion to the applied magnetic field strength. Conversely, if you place a north pole near this side, the value read by the code will decrease linearly in proportion to that magnetic field strength. For more details, please refer to *Hall Sensor Module*.

5. **Delay and Exception Handling**:

```
time.sleep(0.2)  # Wait for 0.2 seconds before the next read

except KeyboardInterrupt:
    print("Exit")  # Exit on CTRL+C
```

`time.sleep(0.2)` creates a 0.2-second delay between each loop iteration to prevent excessive reading speed. The `except` block catches a keyboard interrupt (CTRL+C) to exit the program gracefully.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.8 Lesson 07: Infrared Speed Sensor Module

In this lesson, you will learn how to measure rotational speed using a Raspberry Pi and a simple sensor. We'll connect a digital input sensor to GPIO pin 17 and use Python to monitor its state changes. The focus will be on calculating revolutions per second by counting the sensor activations over a specific time period. You'll write a Python function to accurately capture this data and convert it into a measurable speed. This hands-on project is a straightforward yet practical introduction to real-world data collection and analysis with Raspberry Pi, ideal for beginners interested in applied Python programming and hardware interaction.

Required Components

In this project, we need the following components.

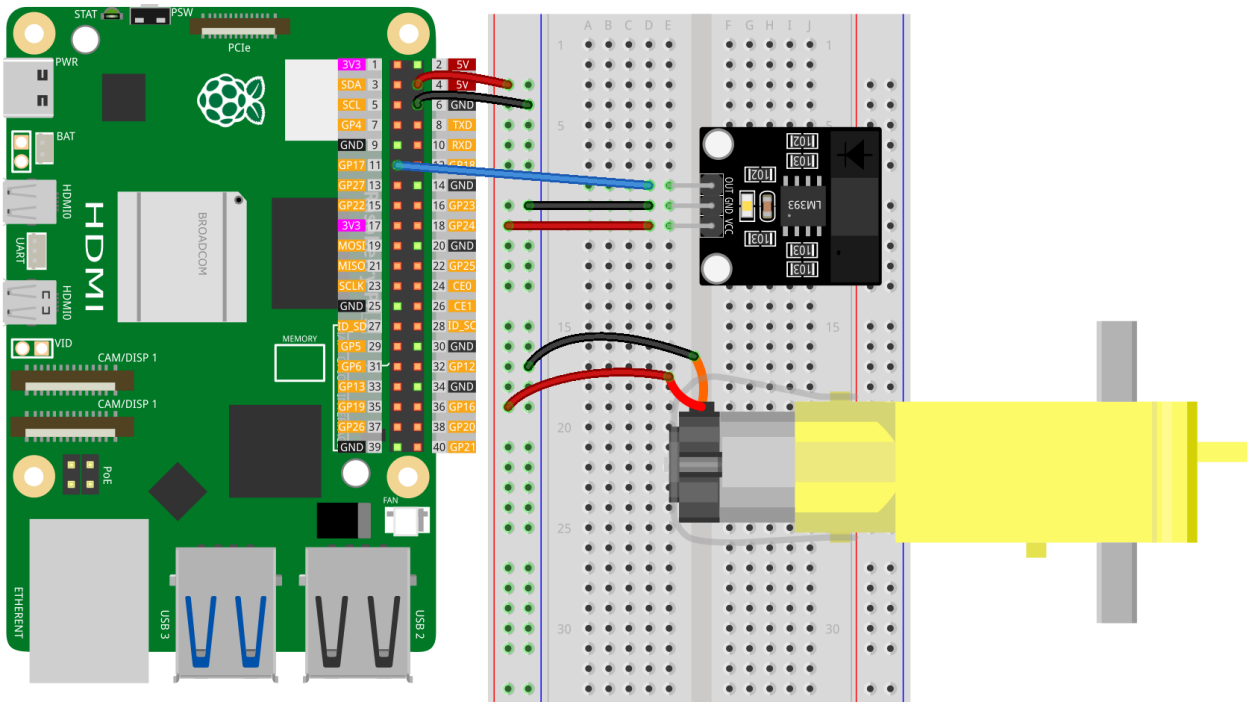
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Infrared Speed Sensor Module	
TT Motor	-
Breadboard	

Wiring



Code

```
from gpiozero import DigitalInputDevice
from time import time

# Initialize the sensor
sensor = DigitalInputDevice(17)  # Assuming the sensor is connected to GPIO17

def calculate_rps(sample_time=1, steps_per_revolution=20):
    """
    Calculate Revolutions Per Second (RPS)

    :param sample_time: Sampling time in seconds
    :param steps_per_revolution: Number of steps in each complete revolution
    :return: Revolutions per second
    """
    start_time = time()
    end_time = start_time + sample_time
    steps = 0
    last_state = False

    while time() < end_time:
        current_state = sensor.is_active
        if current_state and not last_state:
            # Detect a transition from inactive to active state
            steps += 1
        last_state = current_state
```

(continues on next page)

(continued from previous page)

```
    # Calculate RPS
    rps = steps / steps_per_revolution
    return rps

# Example usage
print("Measuring RPS...")

try:
    while True:
        rps = calculate_rps()  # Default sampling for 1 second
        print(f"RPS: {rps}")
except KeyboardInterrupt:
    # Safely exit the program when a keyboard interrupt is detected
    pass
```

Code Analysis

1. Importing Libraries

   The script starts by importing `DigitalInputDevice` from gpiozero for sensor interaction and `time` for time management.

   ```
   from gpiozero import DigitalInputDevice
   from time import time
   ```

2. Initializing the Sensor

   A `DigitalInputDevice` object named `sensor` is created, connected to GPIO pin 17. This setup assumes that the digital sensor is connected to GPIO17.

   ```
   sensor = DigitalInputDevice(17)
   ```

3. Defining the `calculate_rps` Function

   - This function calculates the Revolutions Per Second (RPS) of a rotating object.
   - `sample_time` is the duration in seconds for which the sensor's output is sampled.
   - `steps_per_revolution` represents the number of sensor activations per complete revolution.
   - The function uses a while loop to count the number of steps (sensor activations) within the sample time.
   - It detects transitions from inactive to active states and increments the `steps` count accordingly.
   - RPS is calculated as the number of steps divided by `steps_per_revolution`.

   ```
   def calculate_rps(sample_time=1, steps_per_revolution=20):
       """
       Calculate Revolutions Per Second (RPS)

       :param sample_time: Sampling time in seconds
       :param steps_per_revolution: Number of steps in each complete revolution
       :return: Revolutions per second
       """
   ```

(continues on next page)

(continued from previous page)

```
    start_time = time()
    end_time = start_time + sample_time
    steps = 0
    last_state = False

    while time() < end_time:
        current_state = sensor.is_active
        if current_state and not last_state:
            # Detect a transition from inactive to active state
            steps += 1
        last_state = current_state

    # Calculate RPS
    rps = steps / steps_per_revolution
    return rps
```

4. Running the Main Loop
    - The script then enters a continuous loop where it calls `calculate_rps` to calculate and print the RPS.
    - The loop runs indefinitely until a keyboard interrupt (Ctrl+C) is detected.
    - A `try-except` block is used to handle the interrupt gracefully, allowing for a safe exit.

```
try:
    while True:
        rps = calculate_rps()  # Default sampling for 1 second
        print(f"RPS: {rps}")
except KeyboardInterrupt:
    pass
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.9 Lesson 08: IR Obstacle Avoidance Sensor Module

In this lesson, you will learn how to detect obstacles using a sensor with the Raspberry Pi. We will guide you through connecting a digital input sensor to GPIO pin 17. You'll learn how to write a Python script that continuously monitors the sensor to determine the presence of an obstacle. The program will output a message indicating whether an obstacle is detected or not. This straightforward yet practical project is an excellent way to get started with GPIO interfacing and Python programming, making it ideal for beginners interested in exploring sensor integration with the Raspberry Pi.

Required Components

In this project, we need the following components.

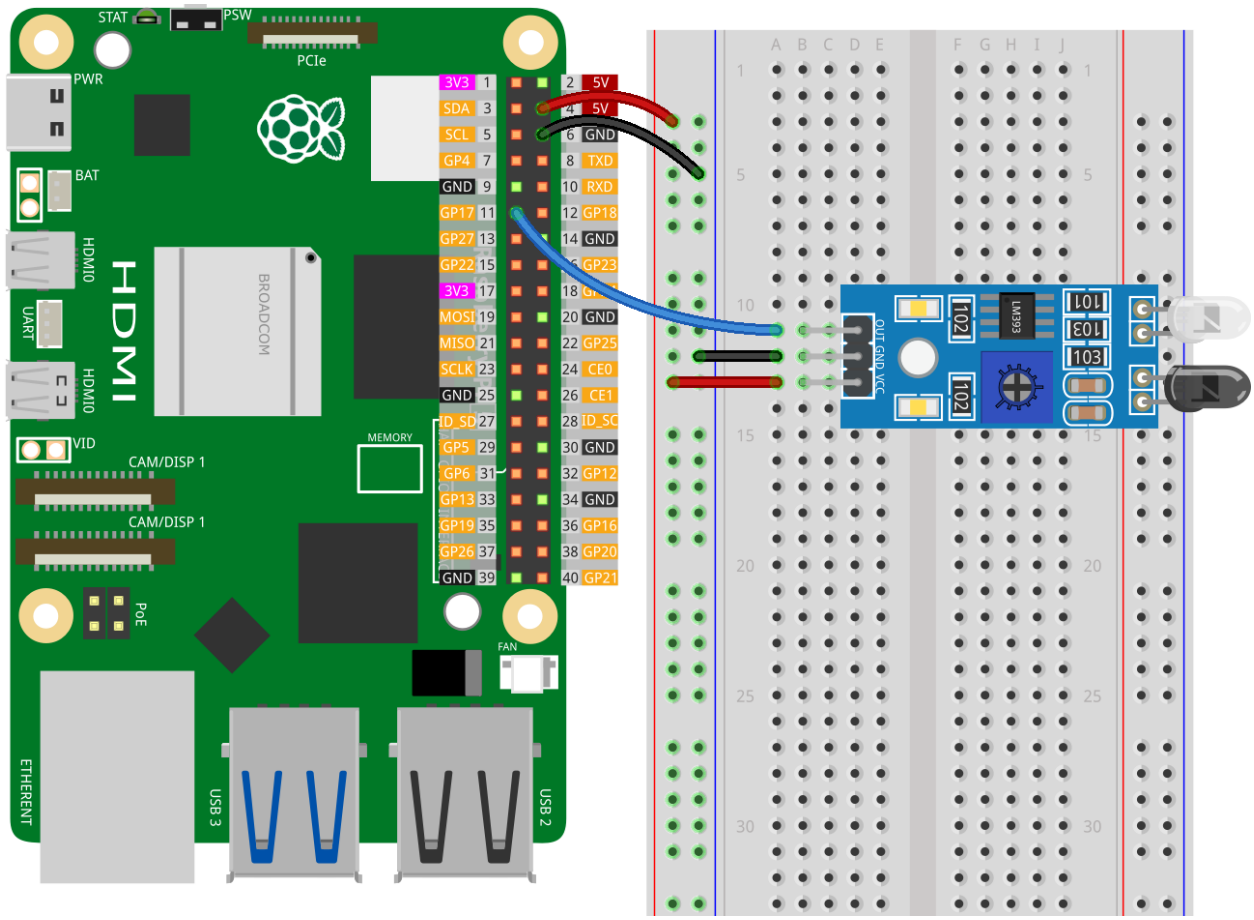
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
IR Obstacle Avoidance Sensor Module	
Breadboard	

Wiring



Code

```
from gpiozero import InputDevice
from time import sleep

# Initialize the sensor as a digital input device on GPIO 17
sensor = InputDevice(17)

while True:
    if sensor.is_active:
        print("No obstacle detected")  # Prints when no obstacle is detected
    else:
        print("Obstacle detected")     # Prints when an obstacle is detected
    sleep(0.5)
```

Code Analysis

1. Importing Libraries

   The script begins by importing the `InputDevice` class from the gpiozero library for interacting with the sensor, and the `sleep` function from Python's time module for pausing execution.

```
from gpiozero import InputDevice
from time import sleep
```

2. Initializing the Sensor

   An `InputDevice` object named `sensor` is created, connected to GPIO pin 17. This line assumes that the obstacle sensor is connected to this specific GPIO pin.

```
sensor = InputDevice(17)
```

3. Implementing the Continuous Monitoring Loop

   - The script uses a `while True:` loop to continuously check the sensor's state. This loop will run indefinitely until the program is stopped.
   - Inside the loop, an `if` statement checks the `is_active` property of the `sensor`.
   - If `is_active` is `True`, it indicates no obstacle is detected, and "No obstacle detected" is printed.
   - If `is_active` is `False`, indicating an obstacle is detected, "Obstacle detected" is printed.
   - `sleep(0.5)` pauses the loop for 0.5 seconds between each check, which helps in reducing the script's processing demand and provides a delay between consecutive sensor readings.

```
while True:
    if sensor.is_active:
        print("No obstacle detected")
    else:
        print("Obstacle detected")
    sleep(0.5)
```

Note: If the sensor is not working properly, adjust the IR transmitter and receiver to make them parallel. Additionally, you can adjust the detection range using the built-in potentiometer.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.10 Lesson 09: Joystick Module

Note: The Raspberry Pi does not have analog input capabilities, so it needs a module like the *PCF8591 ADC DAC Converter Module* to read analog signals for processing.

In this lesson, you will learn how to use a Raspberry Pi to interface with a joystick module using the PCF8591 ADC. You'll be able to read the X and Y positions of the joystick from its analog outputs and detect button presses. This setup demonstrates how to handle both analog and digital inputs on a Raspberry Pi.

Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

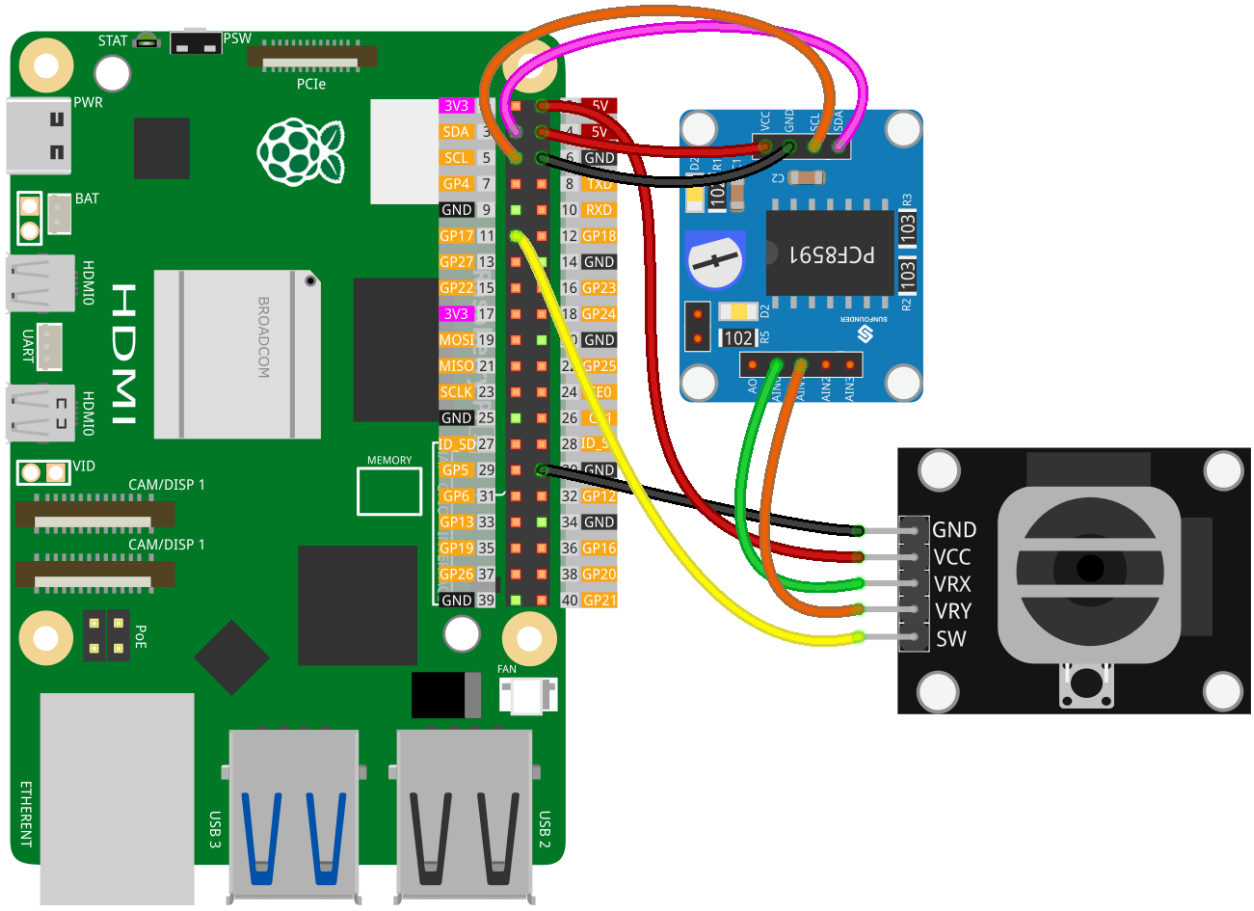
Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Joystick Module	
PCF8591 ADC DAC Converter Module	

Wiring

Note: In this project, we utilized the AIN0 pin of the PCF8591 module, which is linked to a potentiometer on the module through a jumper cap. **To prevent data interference, please disconnect the jumper cap from the module.** For more details, please refer to the PCF8591 module *schematic*.



Code

```
import PCF8591 as ADC  # Import ADC module for analog input
import time  # Import time module for creating delay
from gpiozero import Button  # Import Button for button input

ADC.setup(0x48)  # Set up PCF8591 module at I2C address 0x48

button = Button(17)  # Initialize button connected to GPIO 17

try:
    while True:  # Loop continuously
        print("x:", ADC.read(0))  # Read analog value from channel AIN0
        print("y:", ADC.read(1))  # Read analog value from channel AIN1
        print("sw:", button.is_active)  # Check if button is pressed
        time.sleep(0.2)  # Wait for 0.2 seconds before next loop
except KeyboardInterrupt:
    print("Exit")  # End program on keyboard interrupt
```

Code Analysis

1. **Import Libraries**:

   The script starts with importing necessary libraries for the project.

```
import PCF8591 as ADC  # Import ADC module for analog input
import time  # Import time module for creating delay
from gpiozero import Button  # Import Button for button input
```

2. **Setup PCF8591 Module**:

   The PCF8591 module is set up at I2C address 0x48 which allows the Raspberry Pi to communicate with it.

```
ADC.setup(0x48)  # Set up PCF8591 module at I2C address 0x48
```

3. **Initialize Button**:

   A button is initialized, connected to GPIO pin 17 on the Raspberry Pi.

```
button = Button(17)  # Initialize button connected to GPIO 17
```

4. **Main Loop**:

   The main part of the script is an infinite loop that continuously reads analog values from two channels of the PCF8591 (AIN0 and AIN1) and checks if the button is pressed. `AIN0` and `AIN1` are analog pins for the joystick's X and Y axes.

```
try:
    while True:  # Loop continuously
        print("x:", ADC.read(0))  # Read analog value from channel AIN0
        print("y:", ADC.read(1))  # Read analog value from channel AIN1
        print("sw:", button.is_active)  # Check if button is pressed
        time.sleep(0.2)  # Wait for 0.2 seconds before next loop
```

5. **Interrupt Handling**:

   The script can be exited gracefully using a keyboard interrupt (CTRL+C), which is a common practice in Python for stopping an infinite loop.

```
except KeyboardInterrupt:
    print("Exit")  # End program on keyboard interrupt
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.11 Lesson 10: PCF8591 ADC DAC Converter Module

Note: The Raspberry Pi does not have analog input capabilities, so it needs a module like the *PCF8591 ADC DAC Converter Module* to read analog signals for processing.

In this lesson, you will learn how to use a Raspberry Pi to interact with the PCF8591 module for analog-to-digital and digital-to-analog conversion. We'll cover reading analog values from input AIN0, sending these values to the DAC(AOUT). The module's potentiometer is connected to AIN0 using jumper caps, and the D2 LED on the module is connected to AOUT, so you can see that the brightness of D2 LED changes as you rotate the potentiometer.

Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

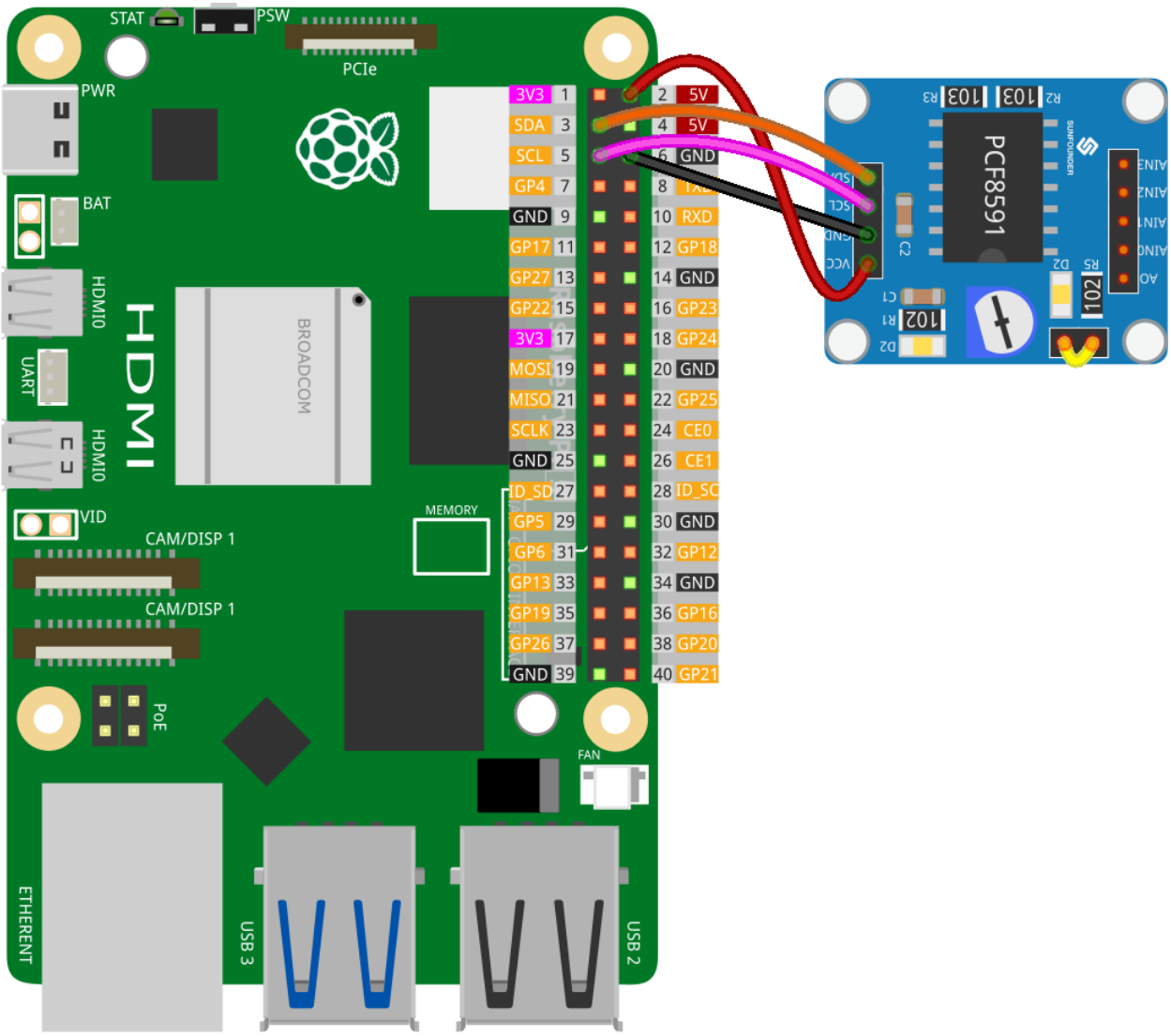
Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
PCF8591 ADC DAC Converter Module	
Breadboard	

Wiring

Note: In this project, we utilized the AIN0 pin of the PCF8591 module, which is linked to a potentiometer on the module through a jumper cap. **Please make sure that the jumper cap on the module is correctly placed.** For more details, please refer to the PCF8591 module *schematic*.



Code

```
import PCF8591 as ADC  # Import the library for the PCF8591 module
import time  # Import the time library for adding delays

# Initialize the PCF8591 module at I2C address 0x48.
# This address is used for communication with the Raspberry Pi.
ADC.setup(0x48)
```

(continues on next page)

(continued from previous page)

```
try:
    while True:  # Start an infinite loop to continuously monitor the sensor.
        # Read the analog value from the potentiometer connected to AIN0.
        # Channel range from 0 to 3 represents AIN0 to AIN3.
        # The potentiometer's rotation alters the voltage, which is read by the PCF8591.
        potentiometer_value = ADC.read(0)
        print(potentiometer_value)

        # Write the value back to AOUT. This will change the brightness of the D2 LED on␣
↪the module.
        # LED won't light up below 80, so convert '0-255' to '80-255'
        # As the potentiometer is adjusted, the LED's brightness varies proportionally.
        tmp = potentiometer_value*(255-80)/255+80
        ADC.write(tmp)

        # Add a short delay of 0.2 seconds to make the loop more manageable.
        time.sleep(0.2)

except KeyboardInterrupt:
    # If a KeyboardInterrupt (CTRL+C) is detected, exit the loop and end the program.
    print("Exit")
```

Code Analysis

1. **Importing Libraries**:

   The script starts by importing required libraries. The `PCF8591` library is used for interacting with the ADC/DAC module, and `time` is for creating delays.

   ```
   import PCF8591 as ADC  # Import the library for the PCF8591 module
   import time  # Import the time library for adding delays
   ```

2. **Initializing PCF8591 Module**:

   The PCF8591 module is initialized at the I^2C address 0x48. This step is crucial for setting up communication between the Raspberry Pi and the module.

   ```
   ADC.setup(0x48)  # Initialize the PCF8591 module at I2C address 0x48
   ```

3. **Reading from Potentiometer and Writing to LED**:

   Within a `try` block, a continuous `while True` loop reads the value from the potentiometer connected to AIN0 and writes this value to the DAC connected to AOUT. Jumper caps link the module's potentiometer to AIN0, and the D2 LED is connected to AOUT; please refer to the PCF8591 module *schematic* for details. The brightness of the LED changes as the potentiometer is rotated.

   - Use `ADC.read(channel)` to read the analog input of the specific channel. The channel range from 0 to 3 represents AIN0 to AIN3.
   - Use `ADC.write(Value)` to set the analog output of the AOUT pin with a Value range from 0 to 255.

   ```
   try:
       while True:  # Start an infinite loop to continuously monitor the sensor.
           potentiometer_value = ADC.read(0)
   ```

(continues on next page)

(continued from previous page)

```
        print(potentiometer_value)
        tmp = potentiometer_value*(255-80)/255+80
        ADC.write(tmp)
        time.sleep(0.2)
```

4. **Handling Keyboard Interrupts**:

   A `KeyboardInterrupt` (such as pressing CTRL+C) allows for a graceful exit from the loop without generating errors.

```
except KeyboardInterrupt:
    print("Exit")
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.12 Lesson 11: Photoresistor Module

Note: The Raspberry Pi does not have analog input capabilities, so it needs a module like the *PCF8591 ADC DAC Converter Module* to read analog signals for processing.

In this lesson, we'll learn how to read from a photoresistor module using a Raspberry Pi. You'll find out how to connect a photoresistor Module to the PCF8591 for analog-to-digital conversion and monitor its output in real-time with Python.

Required Components

In this project, we need the following components.

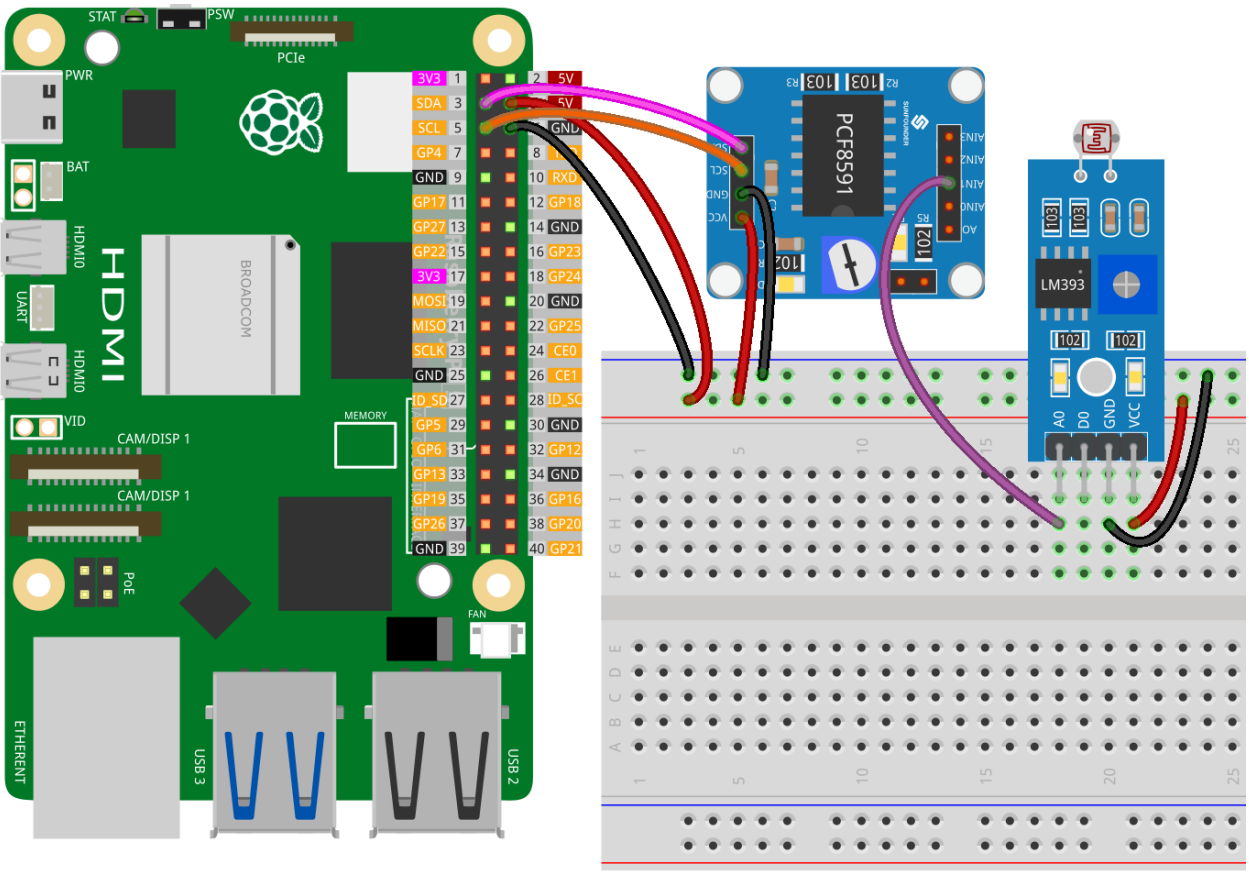
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Photoresistor Module	
PCF8591 ADC DAC Converter Module	
Breadboard	

Wiring



Code

```
import PCF8591 as ADC  # Import PCF8591 module
import time  # Import time for delay

ADC.setup(0x48)  # Initialize PCF8591 at address 0x48

try:
    while True:  # Continuously read and print
        print(ADC.read(1))  # Read from Photoresistor at AIN1
```

(continues on next page)

(continued from previous page)

```
        time.sleep(0.2)  # Delay of 0.2 seconds
except KeyboardInterrupt:
    print("Exit")  # Exit on CTRL+C
```

Code Analysis

1. **Import Libraries**:

   This section imports necessary Python libraries. The `PCF8591` library is used for interacting with the PCF8591 module, and `time` is for implementing delays in the code.

   ```
   import PCF8591 as ADC  # Import PCF8591 module
   import time  # Import time for delay
   ```

2. **Initialize PCF8591 Module**:

   Here, the PCF8591 module is initialized. The address `0x48` is the I^2C address of the PCF8591 module. This is necessary for the Raspberry Pi to communicate with the module.

   ```
   ADC.setup(0x48)  # Initialize PCF8591 at address 0x48
   ```

3. **Main Loop and Reading Data**:

   The `try` block includes a continuous loop that consistently reads data from the Photoresistor module. The `ADC.read(1)` function captures the analog input from the sensor connected to channel 1 (AIN1) of the PCF8591 module. Incorporating a `time.sleep(0.2)` creates a 0.2-second pause between each reading. This not only helps in reducing CPU usage on the Raspberry Pi by avoiding excessive data processing demands, but also prevents the terminal from being overrun with rapidly scrolling information, making it easier to monitor and analyze the output.

   ```
   try:
       while True:  # Continuously read and print
           print(ADC.read(1))  # Read from Photoresistor at AIN1
           time.sleep(0.2)  # Delay of 0.2 seconds
   ```

4. **Handling Keyboard Interrupt**:

   The `except` block is designed to catch a KeyboardInterrupt (like pressing CTRL+C). When this interrupt occurs, the script prints "exit" and stops running. This is a common way to gracefully exit a continuously running script in Python.

   ```
   except KeyboardInterrupt:
       print("exit")  # Exit on CTRL+C
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.

- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.13 Lesson 12: PIR Motion Module (HC-SR501)

In this lesson, you will learn how to set up and use a motion sensor with the Raspberry Pi. We'll walk you through connecting a digital motion sensor to GPIO pin 17. You'll write a Python script to continually check the sensor's state, printing a message when motion is detected and another when the area is clear. This hands-on tutorial is focused on practical skills in electronic circuitry and Python programming, making it perfect for beginners who want to explore real-world applications of the Raspberry Pi in monitoring and automation projects.

Required Components

In this project, we need the following components.

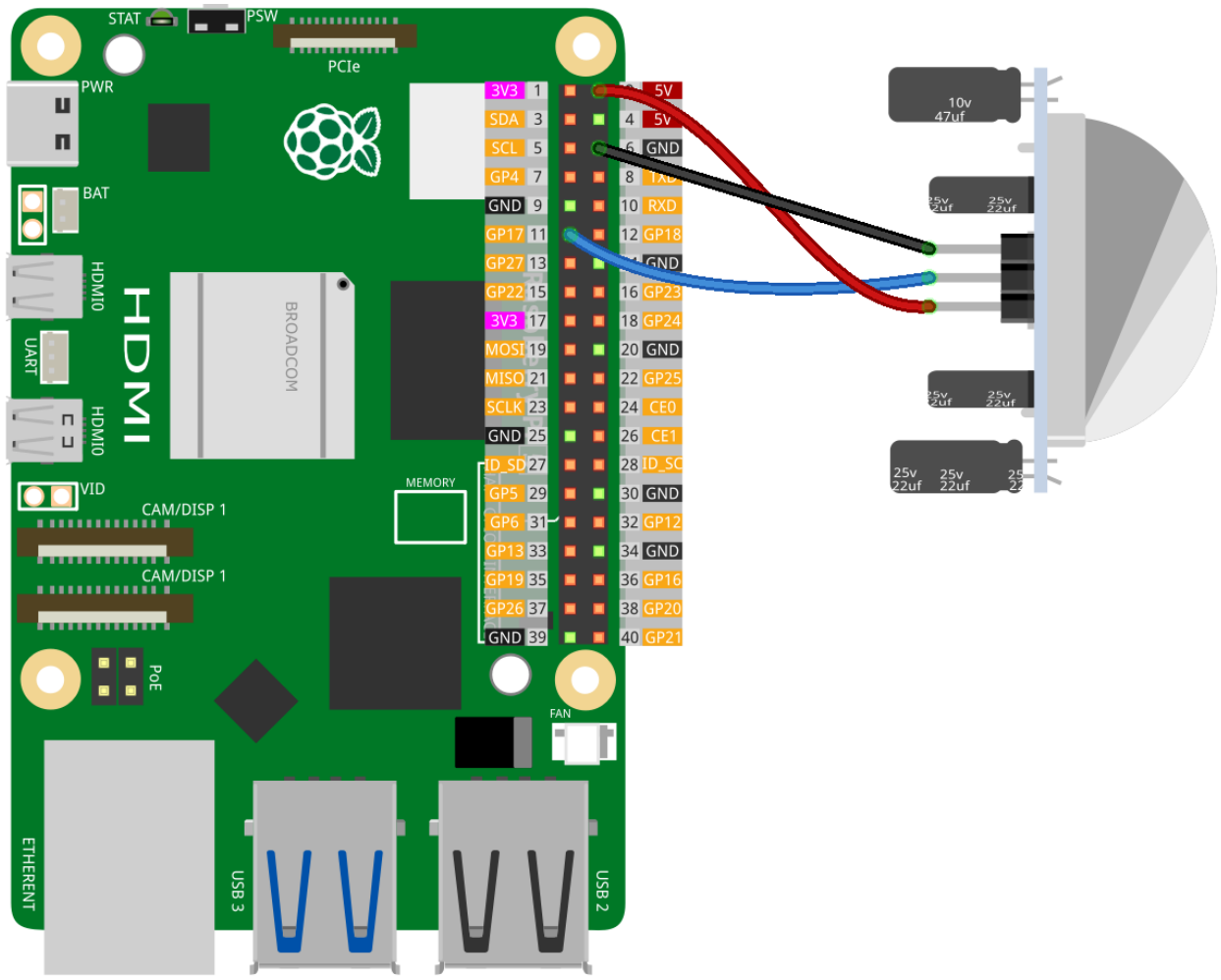
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
PIR Motion Module (HC-SR501)	-
Breadboard	

Wiring



Code

```
from gpiozero import DigitalInputDevice
from time import sleep

# Initialize the motion sensor as a digital input device on GPIO pin 17
motion_sensor = DigitalInputDevice(17)

# Continuously monitor the state of the motion sensor
while True:
    if motion_sensor.is_active:
        print("Somebody here!")
    else:
        print("Monitoring...")

    # Wait for 0.5 seconds before the next sensor check
    sleep(0.5)
```

Code Analysis

1. Importing Libraries

   The script starts by importing the `DigitalInputDevice` class from the gpiozero library for interfacing with the motion sensor, and the `sleep` function from the time module to introduce delays.

```
from gpiozero import DigitalInputDevice
from time import sleep
```

2. Initializing the Motion Sensor

   A `DigitalInputDevice` object named `motion_sensor` is created, connected to GPIO pin 17. This assumes that the motion sensor is connected to this GPIO pin on the Raspberry Pi.

```
motion_sensor = DigitalInputDevice(17)
```

3. Implementing Continuous Monitoring Loop

   - The script employs a `while True:` loop for continuous monitoring.
   - Inside the loop, an `if` statement checks the `is_active` property of the `motion_sensor`.
   - If `is_active` is `True`, it suggests that motion is detected, and "Somebody here!" is printed.
   - If `is_active` is `False`, suggesting no motion is detected, "Monitoring…" is printed.
   - The `sleep(0.5)` function is used to pause the loop for 0.5 seconds between each sensor check, reducing the processing demand and controlling the frequency of sensor polling.

```
while True:
    if motion_sensor.is_active:
        print("Somebody here!")
    else:
        print("Monitoring...")
    sleep(0.5)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.14 Lesson 13: Potentiometer Module

Note: The Raspberry Pi does not have analog input capabilities, so it needs a module like the *PCF8591 ADC DAC Converter Module* to read analog signals for processing.

In this lesson, we'll learn how to read from a potentiometer using a Raspberry Pi. You'll find out how to connect a potentiometer module to the PCF8591 for analog-to-digital conversion and monitor its output in real-time with Python.

Required Components

In this project, we need the following components.

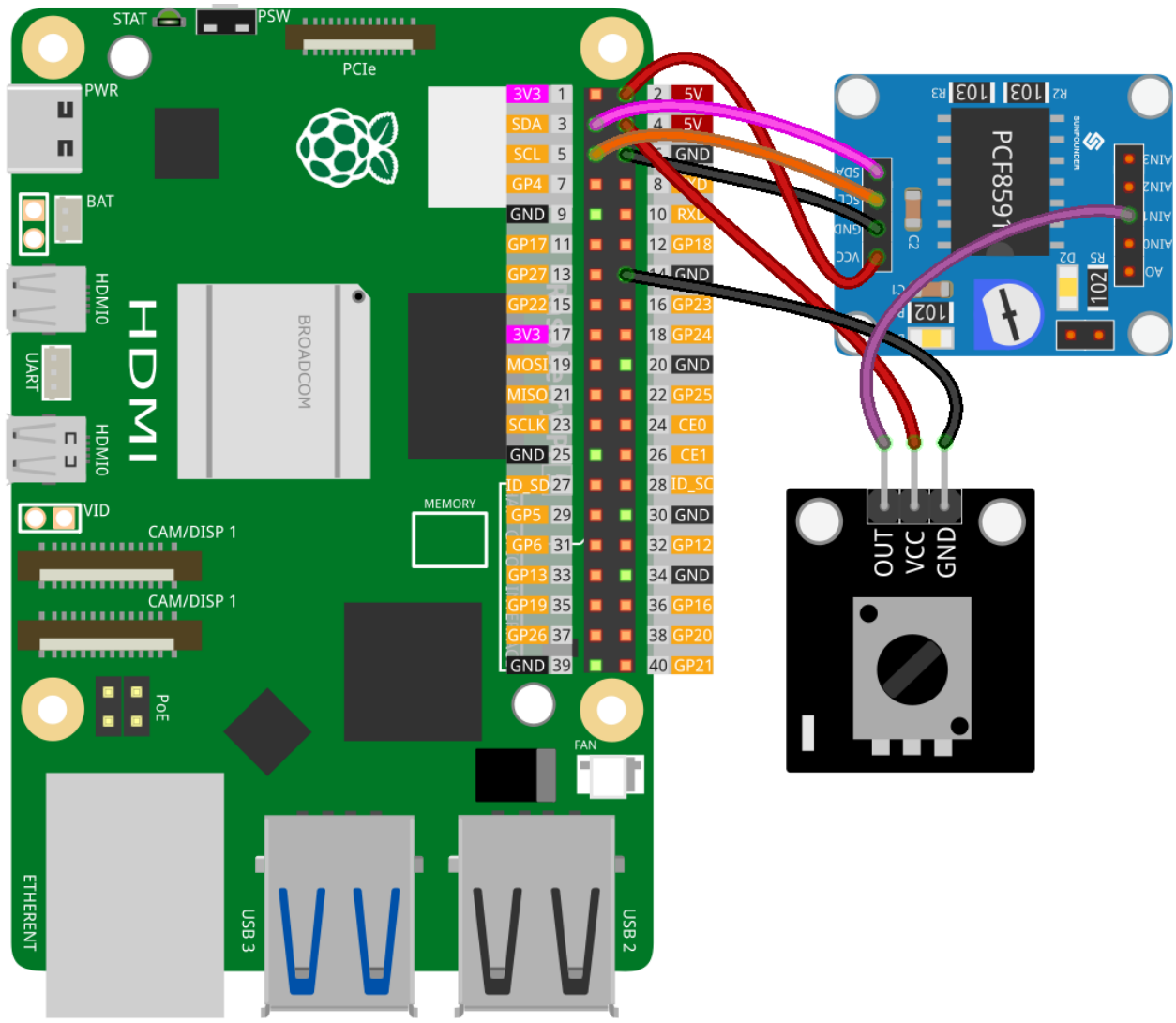
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Potentiometer Module	
PCF8591 ADC DAC Converter Module	

Wiring



Code

```
import PCF8591 as ADC  # Import PCF8591 module
import time  # Import time for delay

ADC.setup(0x48)  # Initialize PCF8591 at address 0x48

try:
    while True:  # Continuously read and print
        print(ADC.read(1))  # Read from Potentiometer at AIN1
        time.sleep(0.2)  # Delay of 0.2 seconds
except KeyboardInterrupt:
    print("Exit")  # Exit on CTRL+C
```

Code Analysis

1. **Import Libraries**:

   This section imports necessary Python libraries. The `PCF8591` library is used for interacting with the PCF8591 module, and `time` is for implementing delays in the code.

```
import PCF8591 as ADC  # Import PCF8591 module
import time  # Import time for delay
```

2. **Initialize PCF8591 Module**:

   Here, the PCF8591 module is initialized. The address `0x48` is the I^2C address of the PCF8591 module. This is necessary for the Raspberry Pi to communicate with the module.

```
ADC.setup(0x48)  # Initialize PCF8591 at address 0x48
```

3. **Main Loop and Reading Data**:

   The `try` block includes a continuous loop that consistently reads data from the potentiometer module. The `ADC.read(1)` function captures the analog input from the sensor connected to channel 1 (AIN1) of the PCF8591 module. Incorporating a `time.sleep(0.2)` creates a 0.2-second pause between each reading. This not only helps in reducing CPU usage on the Raspberry Pi by avoiding excessive data processing demands, but also prevents the terminal from being overrun with rapidly scrolling information, making it easier to monitor and analyze the output.

```
try:
    while True:  # Continuously read and print
        print(ADC.read(1))  # Read from Potentiometer at AIN1
        time.sleep(0.2)  # Delay of 0.2 seconds
```

4. **Handling Keyboard Interrupt**:

   The `except` block is designed to catch a KeyboardInterrupt (like pressing CTRL+C). When this interrupt occurs, the script prints "exit" and stops running. This is a common way to gracefully exit a continuously running script in Python.

```
except KeyboardInterrupt:
    print("exit")  # Exit on CTRL+C
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.15 Lesson 14: Pulse Oximeter and Heart Rate Sensor Module (MAX30102)

In this tutorial, you'll learn to operate the MAX30102 sensor using a Raspberry Pi, streamlined through the use of the open-source MAX30102 Python driver available on GitHub. This approach makes it easier to interface with the module, allowing you to focus on understanding the basics of sensor data collection and analysis. Ideal for novices, the project provides hands-on experience with sensor implementation and Python coding on the Raspberry Pi platform.

Required Components

In this project, we need the following components.

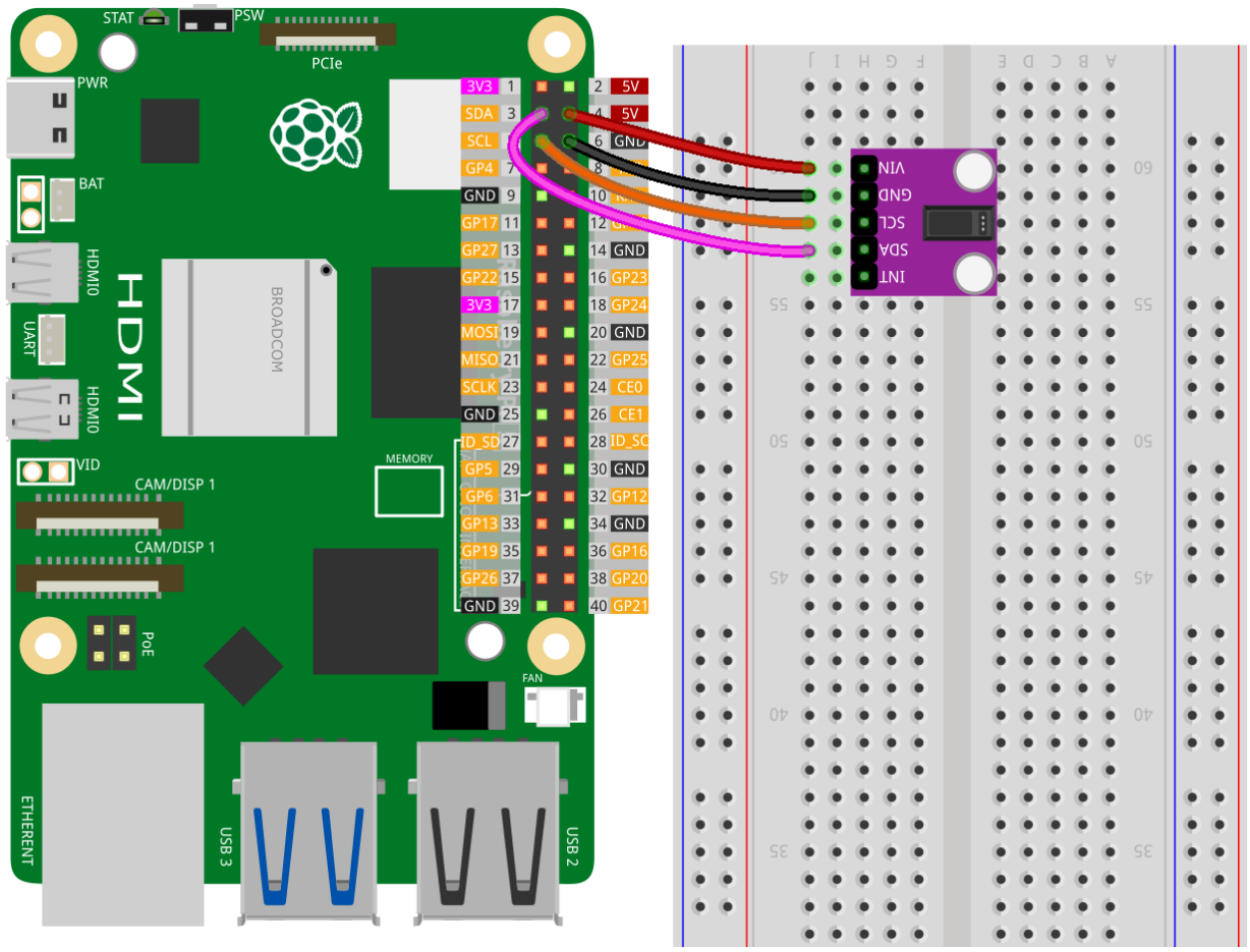
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Pulse Oximeter and Heart Rate Sensor Module (MAX30102)	
Breadboard	

Wiring



Code

```
from heartrate_monitor import HeartRateMonitor
import time

# Print a message indicating the sensor is starting
print('sensor starting...')

# Set the duration for which the sensor data will be read (in seconds)
duration = 30

# Initialize the HeartRateMonitor object
# Set print_raw to False to avoid printing raw data
# Set print_result to True to print the calculated results
hrm = HeartRateMonitor(print_raw=False, print_result=True)

# Start the heart rate sensor
hrm.start_sensor()
```

(continues on next page)

(continued from previous page)

```
try:
    time.sleep(duration)
except KeyboardInterrupt:
    print('keyboard interrupt detected, exiting...')

# Stop the sensor after the duration has elapsed
hrm.stop_sensor()

# Print a message indicating the sensor has stopped
print('sensor stopped!')
```

Code Analysis

1. Importing Modules

   - The `heartrate_monitor` module is used to interface with the sensor. For more information about the `heartrate_monitor` library, please visit .
   - The `time` module helps in managing the duration of the sensor data collection.

   ```
   from heartrate_monitor import HeartRateMonitor
   import time
   ```

2. Initializing the Heart Rate Monitor

   - A `HeartRateMonitor` object is created with specific print options.
   - `print_raw` controls whether raw sensor data is printed.
   - `print_result` controls the printing of processed results (heart rate and SpO2).

   ```
   hrm = HeartRateMonitor(print_raw=False, print_result=True)
   ```

3. Starting the Sensor

   The `start_sensor` method activates the heart rate sensor.

   ```
   hrm.start_sensor()
   ```

4. Running the Sensor for a Set Duration

   - The program sleeps for a specified duration, during which the sensor collects data.
   - `time.sleep(duration)` halts the program for the given number of seconds.

   ```
   try:
       time.sleep(duration)
   except KeyboardInterrupt:
       print('keyboard interrupt detected, exiting...')
   ```

5. Stopping the Sensor

   After the duration, the `stop_sensor` method is called to stop data collection.

   ```
   hrm.stop_sensor()
   ```

6. Finalizing the Program

   Prints a message when the sensor stops.

```
print('sensor stopped!')
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.16 Lesson 15: Raindrop Detection Module

In this lesson, you will learn how to detect rain using a digital rain sensor with Raspberry Pi. We will guide you through connecting a rain sensor to GPIO pin 17 on your Raspberry Pi. You'll learn how to program the Raspberry Pi using Python to continuously monitor the sensor. The program will identify whether it's raining or not and display a message accordingly. This practical project is an excellent introduction to environmental sensing, GPIO interfacing, and Python programming, making it ideal for beginners interested in weather-related projects using Raspberry Pi.

Required Components

In this project, we need the following components.

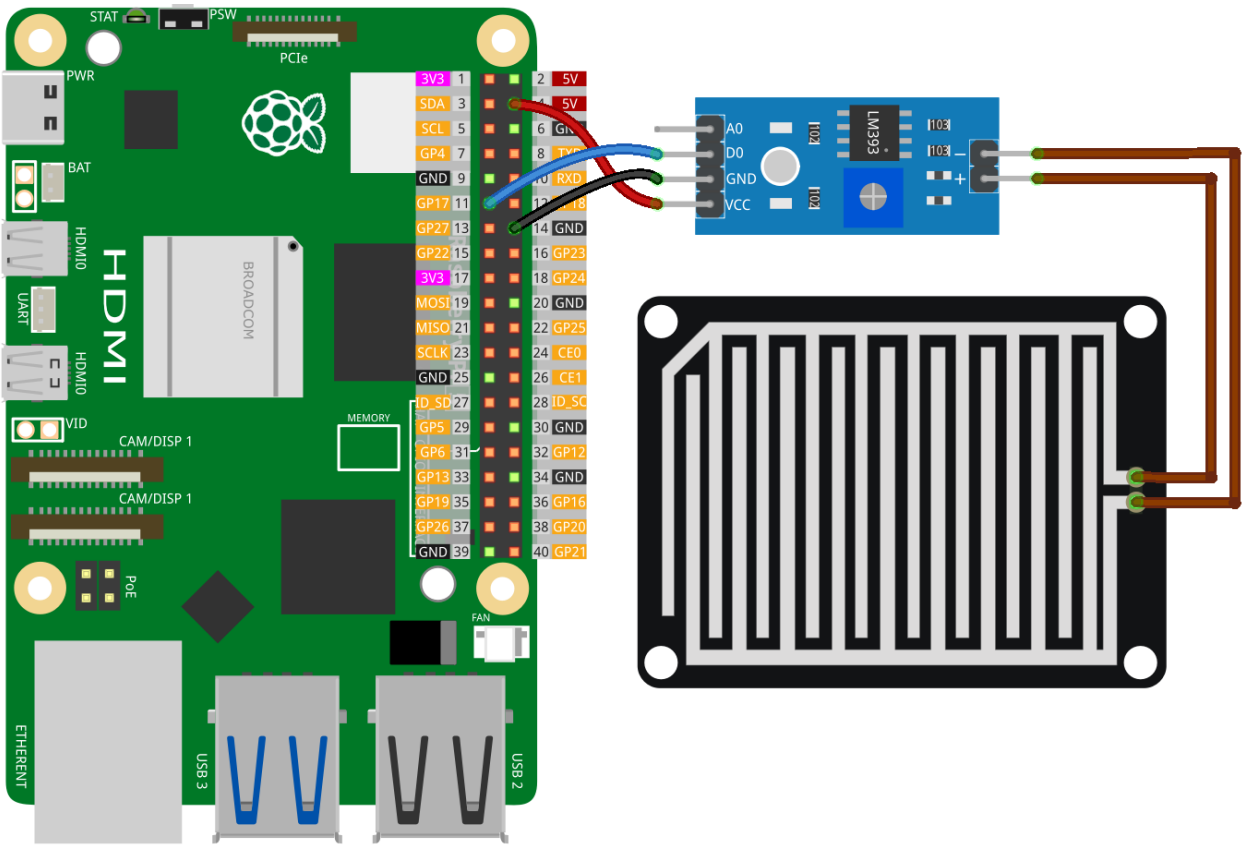
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Raindrop Detection Module	
Breadboard	

Wiring



Code

```
from gpiozero import DigitalInputDevice
from time import sleep

# Initialize the sensor as a digital input device on GPIO pin 17
rain_sensor = DigitalInputDevice(17)

while True:  # Infinite loop to continuously check the sensor status
    if rain_sensor.is_active:  # Check if the sensor is active (no rain)
        print("No rain detected.")  # Print message for no rain detected
    else:
        print("Rain detected!")  # Print message for rain detected
    sleep(1)  # Wait for 1 second before next check
```

Code Analysis

1. Importing Libraries

   The script starts with importing `DigitalInputDevice` from gpiozero for interfacing with the rain sensor, and `sleep` from the time module for implementing delays.

```
from gpiozero import DigitalInputDevice
from time import sleep
```

2. Initializing the Rain Sensor

   A `DigitalInputDevice` object named `rain_sensor` is created, connected to GPIO pin 17. This line configures the rain sensor to communicate with the Raspberry Pi through this GPIO pin.

```
rain_sensor = DigitalInputDevice(17)
```

3. Implementing Continuous Monitoring Loop

   - An infinite loop (`while True:`) is set up to continuously monitor the rain sensor.
   - Inside the loop, an `if` statement checks the `is_active` property of the `rain_sensor`.
   - If `is_active` is `True`, it indicates no rain is detected, and "No rain detected." is printed.
   - If `is_active` is `False`, it indicates rain is detected, and "Rain detected!" is printed.
   - `sleep(1)` pauses the loop for 1 second between each check, controlling the frequency of sensor polling and reducing CPU usage.

```
while True:
    if rain_sensor.is_active:
        print("No rain detected.")
    else:
        print("Rain detected!")
    sleep(1)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.17 Lesson 17: Rotary Encoder Module

In this lesson, you will learn how to connect and program a rotary encoder with a Raspberry Pi. We will provide step-by-step instructions on writing a Python script that monitors the encoder's position and button state, with outputs displayed in the console.

Required Components

In this project, we need the following components.

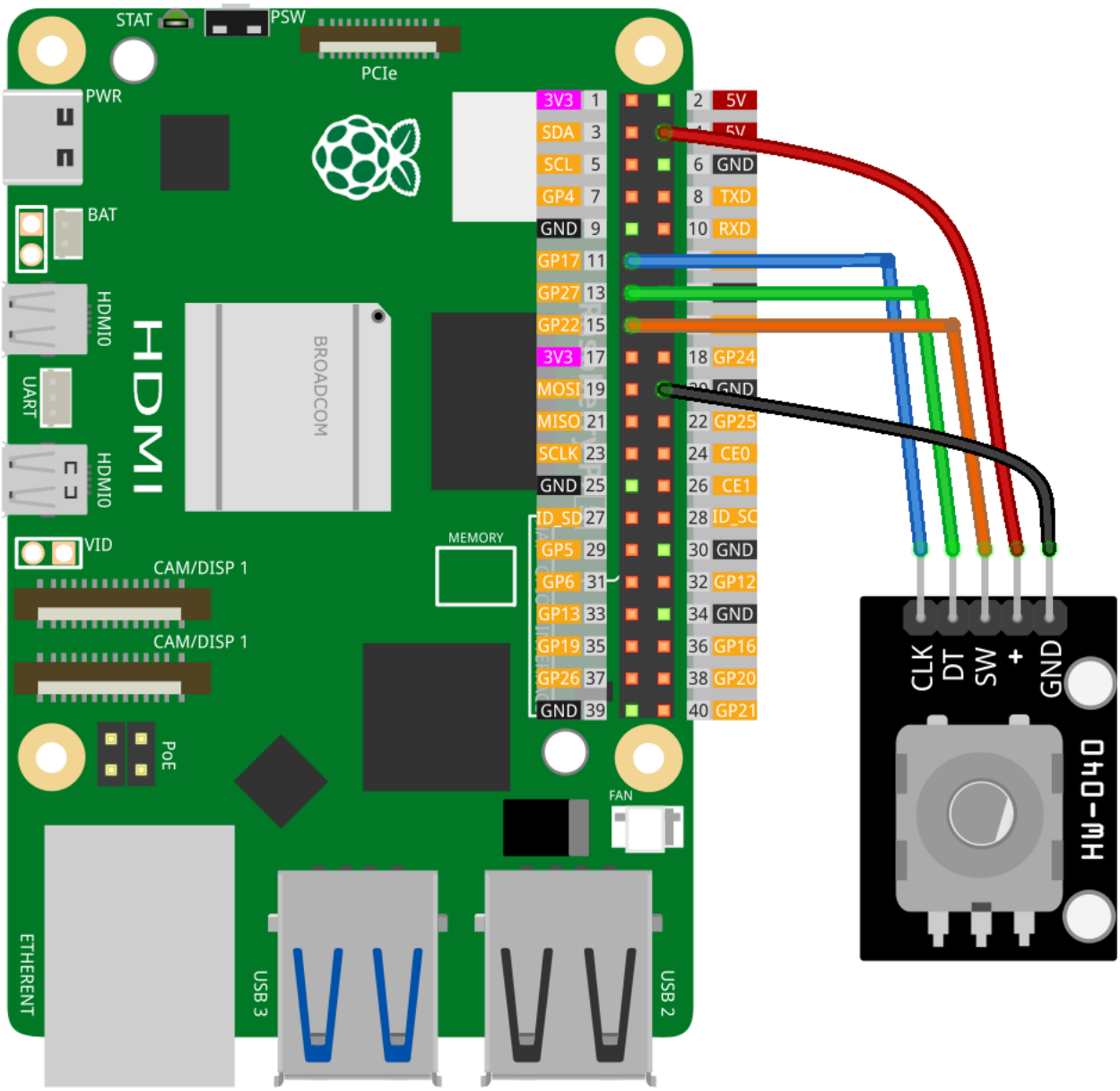
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Rotary Encoder Module	-
Breadboard	

Wiring



Code

```
from gpiozero import RotaryEncoder, Button
from time import sleep

# Initialize the rotary encoder on GPIO pins 17(CLK) and 27(DT) with wrap-around at max_
↪steps of 16
encoder = RotaryEncoder(a=17, b=27, wrap=True, max_steps=16)
# Initialize the rotary encoder's SW pin on GPIO pin 22
button = Button(22)
```

(continues on next page)

(continued from previous page)

```
last_rotary_value = 0  # Variable to store the last value of rotary encoder

try:
    while True:  # Infinite loop to continuously monitor the encoder
        current_rotary_value = encoder.steps  # Read current step count from rotary␣
↪encoder

        # Check if the rotary encoder value has changed
        if last_rotary_value != current_rotary_value:
            print("Result =", current_rotary_value)  # Print the current value
            last_rotary_value = current_rotary_value  # Update the last value

        # Check if the rotary encoder is pressed
        if button.is_pressed:
            print("Button pressed!")  # Print message on button press
            button.wait_for_release()  # Wait until button is released

        sleep(0.1)  # Short delay to prevent excessive CPU usage

except KeyboardInterrupt:
    print("Program terminated")  # Print message when program is terminated via keyboard␣
↪interrupt
```

Code Analysis

1. Importing Libraries

   The script starts with importing the `RotaryEncoder` and `Button` classes from gpiozero for interfacing with the rotary encode, respectively, and the `sleep` function from the time module for adding delays.

   ```
   from gpiozero import RotaryEncoder, Button
   from time import sleep
   ```

2. Initializing the Rotary Encoder and Button

   - This line initializes a `RotaryEncoder` object from the `gpiozero` library. The encoder is connected to GPIO pins 17 and 27.
   - The `wrap=True` parameter means the encoder's value will reset after reaching `max_steps` (16 in this case), mimicking a circular dial behavior.
   - Here, a `Button` object is created, connected to GPIO pin 22. This object will be used to detect when the rotary encoder is pressed.

   ```
   encoder = RotaryEncoder(a=17, b=27, wrap=True, max_steps=16)
   button = Button(22)
   ```

3. Implementing the Monitoring Loop

   - An infinite loop (`while True:`) is used to continuously monitor the rotary encoder.
   - The current value of the rotary encoder is read and compared with its last recorded value. If there's a change, the new value is printed.

- The script checks if the rotary encoder is pressed. On detection of a press, it prints a message and waits until the rotary encoder is released.
- A `sleep(0.1)` is included to add a brief delay, preventing excessive CPU usage.

```
last_rotary_value = 0

try:
    while True:
        current_rotary_value = encoder.steps
        if last_rotary_value != current_rotary_value:
            print("Result =", current_rotary_value)
            last_rotary_value = current_rotary_value

        if button.is_pressed:
            print("Button pressed!")
            button.wait_for_release()

        sleep(0.1)

except KeyboardInterrupt:
    print("Program terminated")
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.18 Lesson 18: Temperature Sensor Module (DS18B20)

In this lesson, you will learn how to use a Raspberry Pi to read temperature data from a DS18B20 temperature sensor. You will understand how to locate the sensor's device file, read and parse its raw data, and convert this data into Celsius and Fahrenheit readings.

Required Components

In this project, we need the following components.

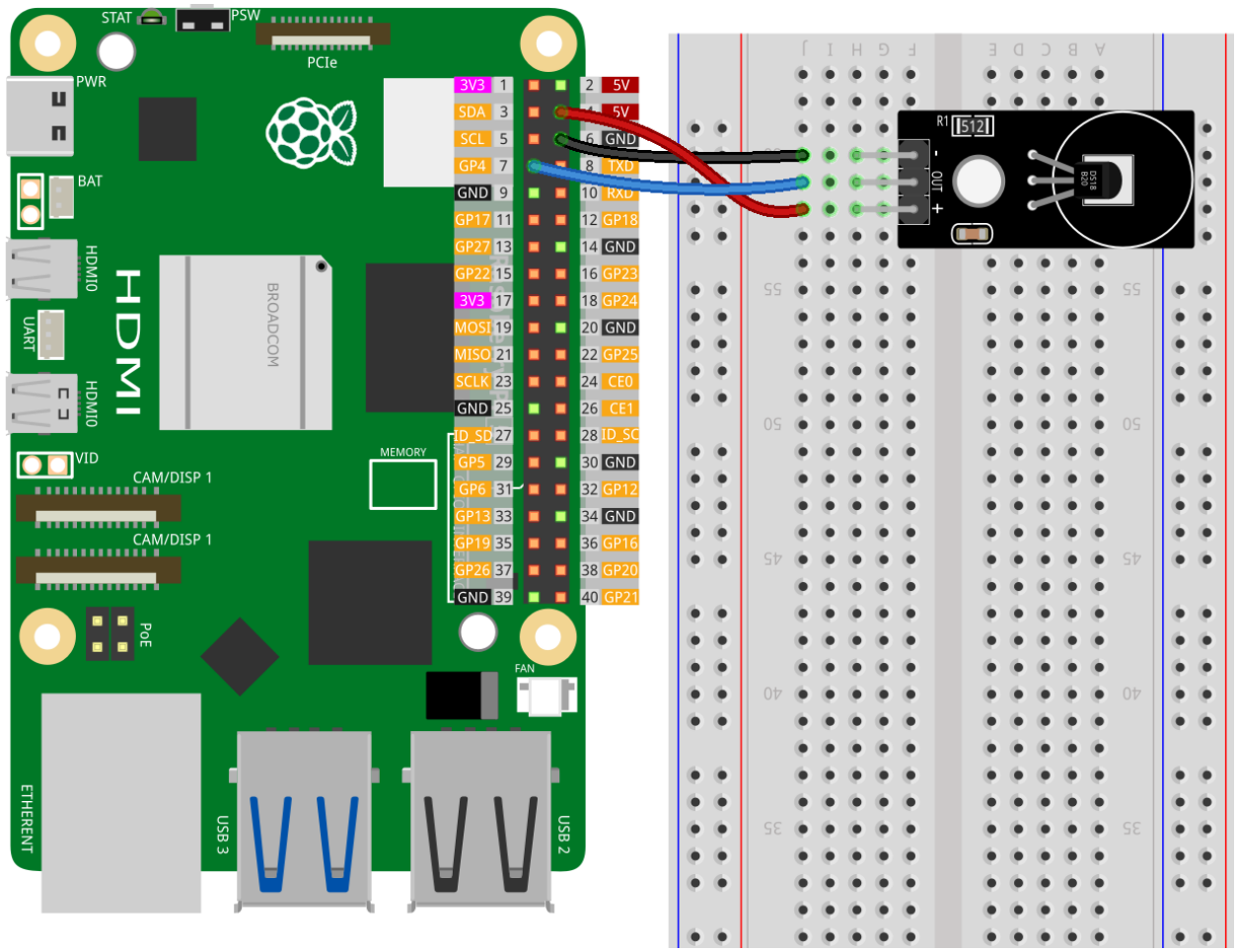
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Temperature Sensor Module (DS18B20)	-
Breadboard	

Wiring



Code

Note: The DS18B20 module communicates with the Raspberry Pi using the onewire protocol. Before running the code, you need to enable the onewire function of the Raspberry Pi. You can refer to this tutorial: *1-Wire Configuration*.

```
import glob
import time

# Path to the directory containing device files for 1-wire devices
base_dir = "/sys/bus/w1/devices/"

# Finds the first device folder that starts with "28", specific to DS18B20
device_folder = glob.glob(base_dir + "28*")[0]

# Device file containing the temperature data
device_file = device_folder + "/w1_slave"
```

(continues on next page)

(continued from previous page)

```
def read_temp_raw():
    # Reads raw temperature data from the sensor
    f = open(device_file, "r")
    lines = f.readlines()
    f.close()
    return lines


def read_temp():
    # Parses the raw temperature data and converts it to Celsius and Fahrenheit
    lines = read_temp_raw()
    # Waits for a valid temperature reading
    while lines[0].strip()[-3:] != "YES":
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find("t=")
    if equals_pos != -1:
        temp_string = lines[1][equals_pos + 2 :]
        temp_c = float(temp_string) / 1000.0  # Convert to Celsius
        temp_f = temp_c * 9.0 / 5.0 + 32.0  # Convert to Fahrenheit
        return temp_c, temp_f


try:
    # Main loop to continuously read and print temperature
    while True:
        temp_c, temp_f = read_temp()
        formatted_output = f"Temperature: {temp_c:.2f}°C / {temp_f:.2f}°F"
        print(formatted_output)
        time.sleep(1)  # Wait for 1 second between readings
except KeyboardInterrupt:
    # Gracefully exit the program on CTRL+C
    print("Exit")
```

Code Analysis

1. Importing Necessary Libraries

   The `glob` library is used to search for the temperature sensor's device folder. The `time` library is used for implementing delays in the program.

   ```
   import glob
   import time
   ```

2. Locating the Temperature Sensor Device File

   The code searches for the directory of the DS18B20 sensor by looking for a folder name starting with "28". The device file `w1_slave` contains the temperature data.

   ```
   base_dir = "/sys/bus/w1/devices/"
   device_folder = glob.glob(base_dir + "28*")[0]
   device_file = device_folder + "/w1_slave"
   ```

3. Reading Raw Temperature Data

   This function opens the device file and reads its content. It returns the raw temperature data as a list of strings.

```
def read_temp_raw():
    f = open(device_file, "r")
    lines = f.readlines()
    f.close()
    return lines
```

4. Parsing and Converting Temperature Data

   The `read_temp` function calls `read_temp_raw` to get the raw data. It waits for a valid temperature reading and then extracts, parses, and converts the temperature to Celsius and Fahrenheit.

```
def read_temp():
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != "YES":
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find("t=")
    if equals_pos != -1:
        temp_string = lines[1][equals_pos + 2 :]
        temp_c = float(temp_string) / 1000.0
        temp_f = temp_c * 9.0 / 5.0 + 32.0
        return temp_c, temp_f
```

5. Main Program Loop and Graceful Exit

   The `try` block contains an infinite loop to continuously read and display the temperature. The `except` block catches a KeyboardInterrupt to exit the program gracefully.

```
try:
    while True:
        temp_c, temp_f = read_temp()
        formatted_output = f"Temperature: {temp_c:.2f}°C / {temp_f:.2f}°F"
        print(formatted_output)
        time.sleep(1)
except KeyboardInterrupt:
    print("Exit")
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.19 Lesson 19: Temperature and Humidity Sensor Module (DHT11)

In this lesson, you will learn how to connect and read data from a DHT11 temperature and humidity sensor using a Raspberry Pi. You will learn how to set up the sensor, read temperature in both Celsius and Fahrenheit, and obtain humidity readings. This project introduces you to working with external sensors, handling real-time data, and basic exception handling in Python.

Required Components

In this project, we need the following components.

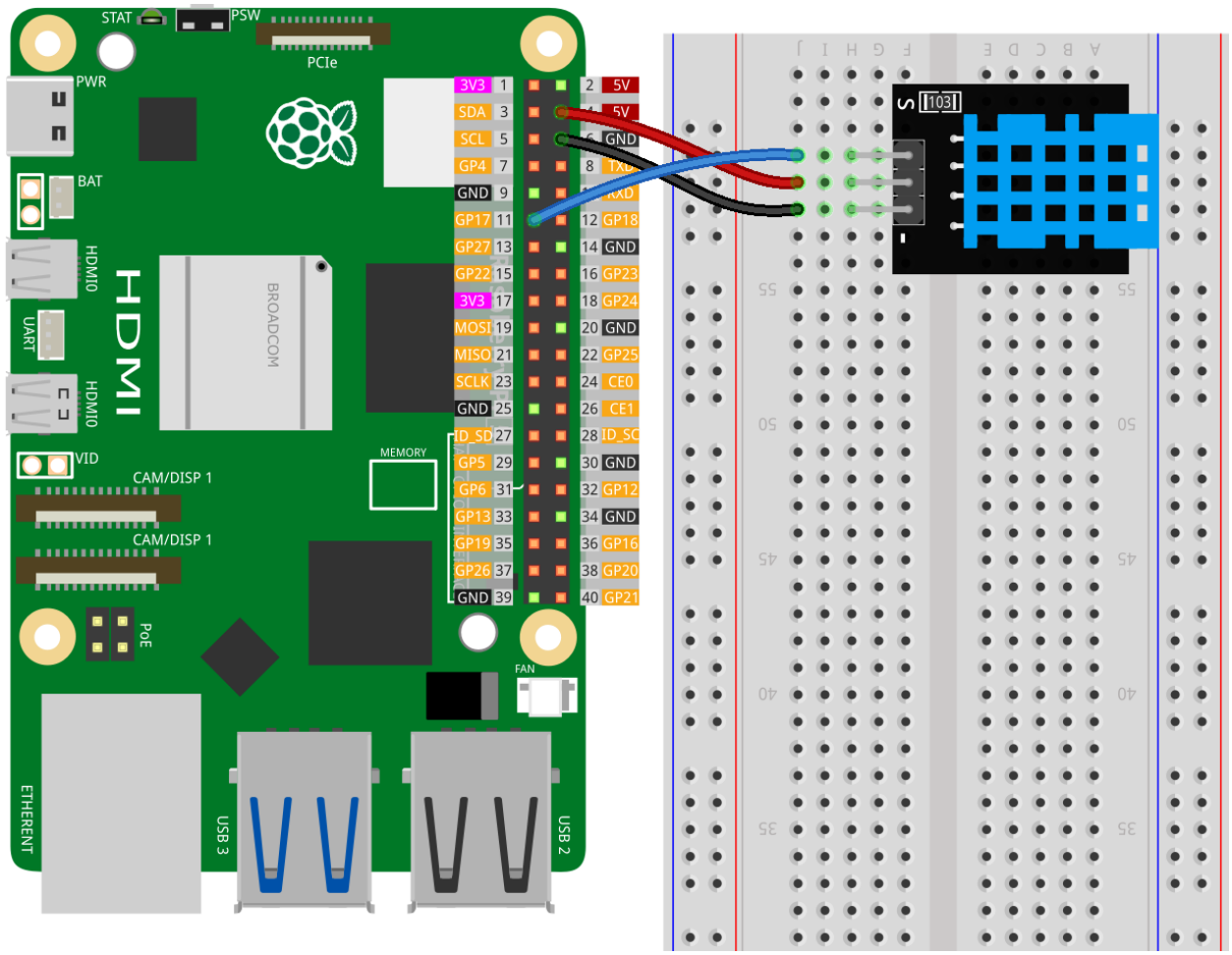
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Temperature and Humidity Sensor Module (DHT11)	
Breadboard	

Wiring



Install Library

Note: The adafruit-circuitpython-dht library relies on Blinka, so please ensure that Blinka has been installed. To install libraries, refer to *Insatll Adafruit_Blinka (CircuitPython) - Optional*.

Before installing the library, please make sure that the virtual Python environment is activated:

```
source ~/env/bin/activate
```

Install adafruit-circuitpython-dht library:

```
pip install adafruit-circuitpython-dht
```

Code

Note:

- Please ensure that you have installed the Python library required for running the code according to the "Install Library" steps.
- Before running the code, please make sure that you have activated the virtual Python environment with blinka installed. You can activate the virtual environment using a command like this:

```
source ~/env/bin/activate
```

- Find the code for this lesson in `universal-maker-sensor-kit-main/pi/` directory, or directly copy and paste the code below. Execute the code by running the following commands in terminal:

```
python 19_dht11_module.py
```

```
import time
import board
import adafruit_dht

# Initial the dht device, with data pin connected to:
dhtDevice = adafruit_dht.DHT11(board.D17)

while True:
    try:
        # Print the values to the serial port
        temperature_c = dhtDevice.temperature
        temperature_f = temperature_c * (9 / 5) + 32
        humidity = dhtDevice.humidity
        print(
            "Temp: {:.1f} F / {:.1f} C    Humidity: {}% ".format(
                temperature_f, temperature_c, humidity
            )
        )

    except RuntimeError as error:
        # Errors happen fairly often, DHT's are hard to read, just keep going
        print(error.args[0])
        time.sleep(2.0)
        continue
    except Exception as error:
        dhtDevice.exit()
        raise error

    time.sleep(2.0)
```

Code Analysis

1. Importing Libraries:

   The code begins by importing necessary libraries. `time` for handling delays, `board` for accessing Raspberry Pi GPIO pins, and `adafruit_dht` for interacting with the DHT11 sensor. For more detail about the `adafruit_dht` library, please refer to .

   ```
   import time
   import board
   import adafruit_dht
   ```

2. Initializing the Sensor:

   The DHT11 sensor is initialized with the data pin connected to GPIO 17 of the Raspberry Pi. This setup is crucial for the sensor to communicate with the Raspberry Pi.

   ```
   dhtDevice = adafruit_dht.DHT11(board.D17)
   ```

3. Reading Sensor Data in a Loop:

   The `while True` loop allows the program to continuously check the sensor for new data.

   ```
   while True:
   ```

4. Try-Except Blocks:

   Within the loop, a try-except block is used to handle potential runtime errors. Reading from DHT sensors can often result in errors due to timing issues or sensor quirks.

   ```
   try:
       # Sensor data reading code here
   except RuntimeError as error:
       # Handling common sensor reading errors
       print(error.args[0])
       time.sleep(2.0)
       continue
   except Exception as error:
       # Handling other exceptions and exiting
       dhtDevice.exit()
       raise error
   ```

5. Reading and Printing Sensor Data:

   The temperature and humidity are read from the sensor and converted into human-readable formats. The temperature is also converted from Celsius to Fahrenheit.

   ```
   temperature_c = dhtDevice.temperature
   temperature_f = temperature_c * (9 / 5) + 32
   humidity = dhtDevice.humidity
   print("Temp: {:.1f} F / {:.1f} C    Humidity: {}% ".format(temperature_f,␣
   ↪temperature_c, humidity))
   ```

6. Handling Read Errors:

   The DHT11 sensor can often return errors, so the code uses a try-except block to handle these. If an error occurs, the program waits for 2 seconds before attempting to read from the sensor again.

```
except RuntimeError as error:
    print(error.args[0])
    time.sleep(2.0)
    continue
```

7. General Exception Handling:

   Any other exceptions that might occur are handled by safely exiting the sensor and re-raising the error. This ensures the program doesn't continue in an unstable state.

```
except Exception as error:
    dhtDevice.exit()
    raise error
```

8. Delay Between Readings:

   A 2-second delay is added at the end of the loop to avoid constant polling of the sensor, which can lead to erroneous readings.

```
time.sleep(2.0)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.20 Lesson 20: Temperature, Humidity & Pressure Sensor (BMP280)

In this lesson, you will learn how to connect and read data from a BMP280 sensor that measures temperature, humidity, and pressure using a Raspberry Pi. You'll set up the sensor and write a Python script to measure environmental data including temperature, atmospheric pressure, and altitude.

Required Components

In this project, we need the following components.

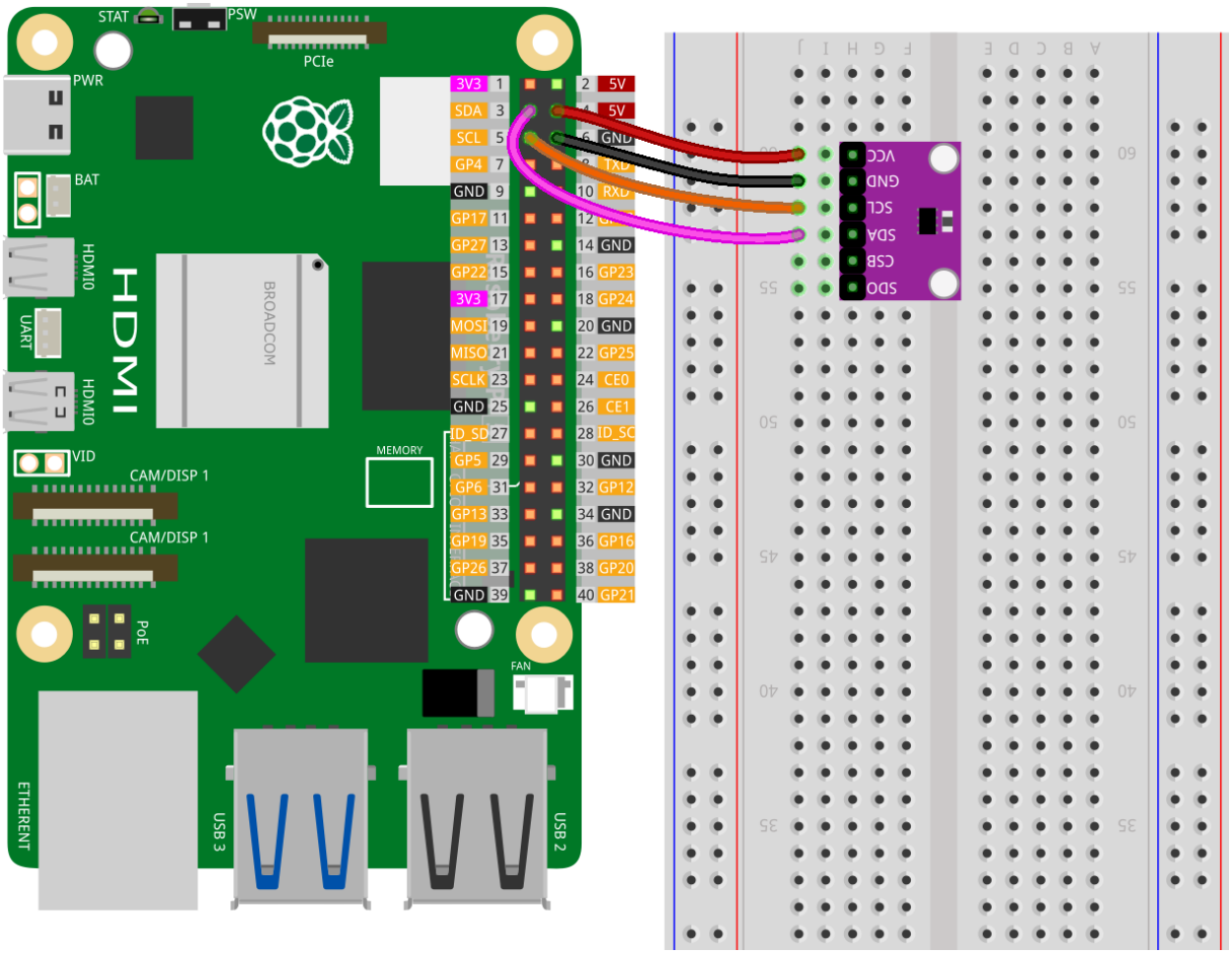
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Temperature, Humidity & Pressure Sensor (BMP280)	
Breadboard	

Wiring



Install Library

Note: The adafruit-circuitpython-bmp280 library relies on Blinka, so please ensure that Blinka has been installed. To install libraries, refer to *Insatll Adafruit_Blinka (CircuitPython) - Optional*.

Before installing the library, please make sure that the virtual Python environment is activated:

```
source ~/env/bin/activate
```

Install adafruit-circuitpython-bmp280 library:

```
pip install adafruit-circuitpython-bmp280
```

Run the Code

Note:

- Please ensure that you have installed the Python library required for running the code according to the "Install Library" steps.
- Before running the code, please make sure that you have activated the virtual Python environment with blinka installed. You can activate the virtual environment using a command like this:

  ```
  source ~/env/bin/activate
  ```

- Find the code for this lesson in `universal-maker-sensor-kit-main/pi/` directory, or directly copy and paste the code below. Execute the code by running the following commands in terminal:

  ```
  python 22_touch_sensor_module.py
  ```

```
import time
import board

import adafruit_bmp280

# Create sensor object, communicating over the board's default I2C bus
i2c = board.I2C()  # uses board.SCL and board.SDA
bmp280 = adafruit_bmp280.Adafruit_BMP280_I2C(i2c,address=0x76)

# change this to match the location's pressure (hPa) at sea level
bmp280.sea_level_pressure = 1013.25

try:
   while True:
      print("\nTemperature: %0.1f C" % bmp280.temperature)
      print("Pressure: %0.1f hPa" % bmp280.pressure)
      print("Altitude = %0.2f meters" % bmp280.altitude)
      time.sleep(2)
except KeyboardInterrupt:
    print("Exit")  # Exit on CTRL+C
```

Code Analysis

1. Setting up the sensor

   Import necessary libraries and create an object to interact with the BMP280 sensor. `board.I2C()` sets up the I2C communication. `adafruit_bmp280.Adafruit_BMP280_I2C(i2c, address=0x76)` initializes the BMP280 sensor with its I2C address.

   For more detail about the `adafruit_bmp280` library, please refer to .

```
import time
import board
import adafruit_bmp280
i2c = board.I2C()
bmp280 = adafruit_bmp280.Adafruit_BMP280_I2C(i2c, address=0x76)
```

2. Configuring sea-level pressure

   Set the `sea_level_pressure` property of the BMP280 object. This value is needed to calculate altitude.

```
bmp280.sea_level_pressure = 1013.25
```

3. Reading data in a loop

   Use a `while True` loop to continuously read data from the sensor. `bmp280.temperature`, `bmp280.pressure`, and `bmp280.altitude` read the temperature, pressure, and altitude, respectively. `time.sleep(2)` pauses the loop for 2 seconds.

```
try:
    while True:
        print("\nTemperature: %0.1f C" % bmp280.temperature)
        print("Pressure: %0.1f hPa" % bmp280.pressure)
        print("Altitude = %0.2f meters" % bmp280.altitude)
        time.sleep(2)
except KeyboardInterrupt:
    print("Exit")
```

4. Handling interruptions

   The `try` and `except KeyboardInterrupt:` block allows the program to exit gracefully when you press CTRL+C.

```
try:
    # while loop code here
except KeyboardInterrupt:
    print("Exit")
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.

- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.21 Lesson 21: Time of Flight Micro-LIDAR Distance Sensor (VL53L0X)

In this lesson, you'll learn how to use the Raspberry Pi to connect with a Time of Flight Micro-LIDAR Distance Sensor (VL53L0X). You'll be guided through setting up the sensor, initializing I2C communication, and measuring distances in real-time. This project will enhance your comprehension of connecting hardware with the Raspberry Pi and utilizing Python for practical applications. Additionally, you'll delve into adjusting measurement parameters to meet varying accuracy and speed needs.

Required Components

In this project, we need the following components.

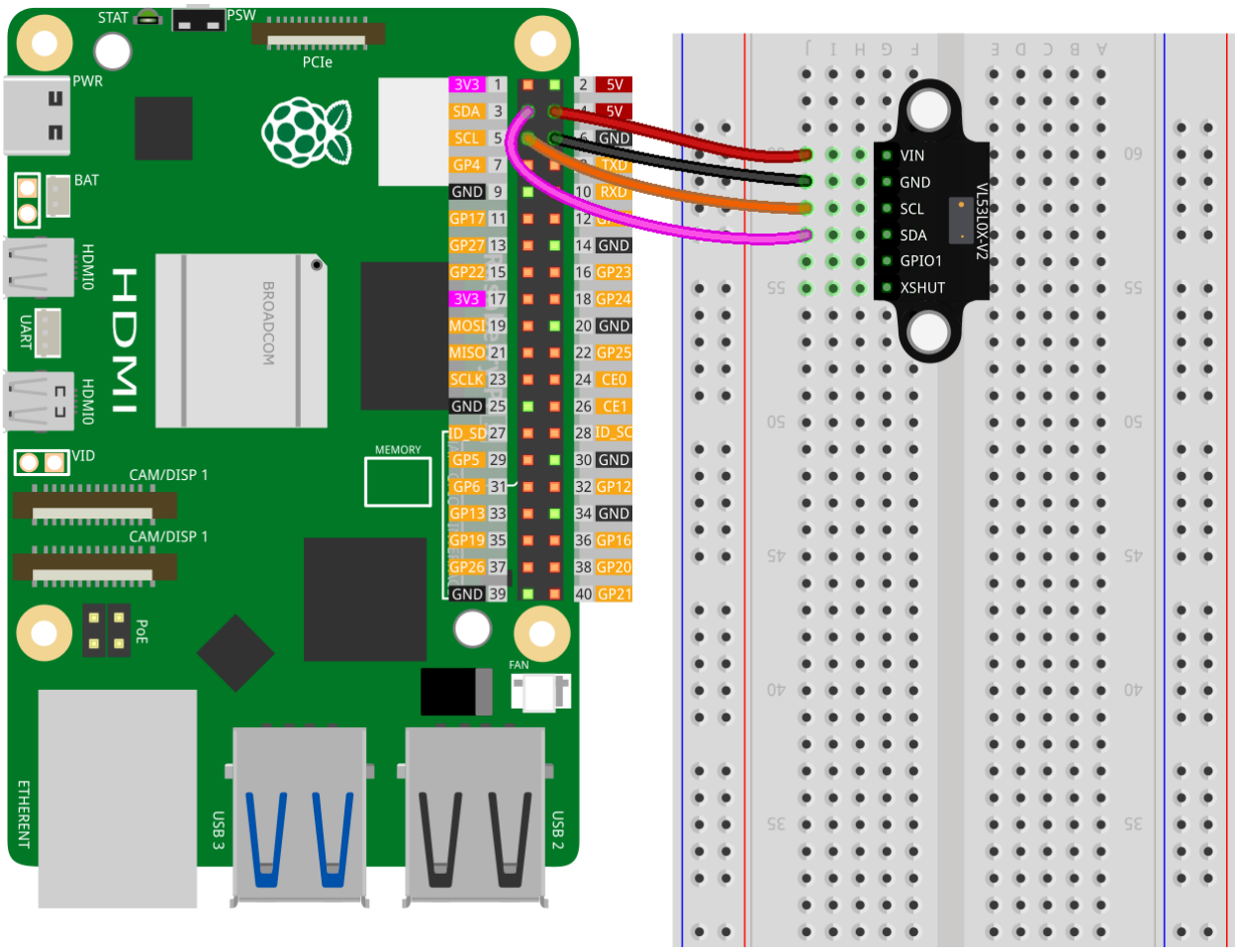
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Time of Flight Micro-LIDAR Distance Sensor (VL53L0X)	
Breadboard	

Wiring



Install Library

Note: The adafruit-circuitpython-vl53l0x library relies on Blinka, so please ensure that Blinka has been installed. To install libraries, refer to *Insatll Adafruit_Blinka (CircuitPython) - Optional*.

Before installing the library, please make sure that the virtual Python environment is activated:

```
source ~/env/bin/activate
```

Install adafruit-circuitpython-vl53l0x library:

```
pip3 install adafruit-circuitpython-vl53l0x
```

Code

Note:

- Please ensure that you have installed the Python library required for running the code according to the "Install Library" steps.
- Before running the code, please make sure that you have activated the virtual Python environment with blinka installed. You can activate the virtual environment using a command like this:

  ```
  source ~/env/bin/activate
  ```

- Find the code for this lesson in `universal-maker-sensor-kit-main/pi/` directory, or directly copy and paste the code below. Execute the code by running the following commands in terminal:

  ```
  python 21_vl53l0x_module.py
  ```

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Simple demo of the VL53L0X distance sensor.
# Will print the sensed range/distance every second.
import time

import board
import busio

import adafruit_vl53l0x

# Initialize I2C bus and sensor.
i2c = busio.I2C(board.SCL, board.SDA)
vl53 = adafruit_vl53l0x.VL53L0X(i2c)

# Optionally adjust the measurement timing budget to change speed and accuracy.
# See the example here for more details:
#   https://github.com/pololu/vl53l0x-arduino/blob/master/examples/Single/Single.ino
# For example a higher speed but less accurate timing budget of 20ms:
# vl53.measurement_timing_budget = 20000
# Or a slower but more accurate timing budget of 200ms:
# vl53.measurement_timing_budget = 200000
# The default timing budget is 33ms, a good compromise of speed and accuracy.

try:
    # Main loop will read the range and print it every second.
    while True:
        print("Range: {0}mm".format(vl53.range))
        time.sleep(1.0)
except KeyboardInterrupt:
    print("Exit")  # Exit on CTRL+C
```

Code Analysis

1. **Importing Libraries**

```
import time
import board
import busio
import adafruit_vl53l0x
```

- `time`: Used for implementing delays.
- `board`: Provides access to the physical pins on the Raspberry Pi.
- `busio`: Manages I2C communication between the Pi and the sensor.
- `adafruit_vl53l0x`: The specific library for the VL53L0X sensor. For more detail about the `adafruit_vl53l0x` library, please refer to .

2. **Initializing the Sensor**

```
# Initialize I2C bus and sensor.
i2c = busio.I2C(board.SCL, board.SDA)
vl53 = adafruit_vl53l0x.VL53L0X(i2c)
```

- This sets up the I2C communication using SCL (clock line) and SDA (data line) pins.
- The VL53L0X sensor is then initialized with this I2C bus.

3. **Configuration (Optional)**

```
# Optionally adjust the measurement timing budget...
# vl53.measurement_timing_budget = 20000
# ...
```

This part of the code, which is commented out, allows for adjusting the sensor's timing budget, affecting the balance between speed and accuracy.

4. **Main Loop**

```
try:
    while True:
        print("Range: {0}mm".format(vl53.range))
        time.sleep(1.0)
except KeyboardInterrupt:
    print("Exit")
```

- In an infinite loop, the sensor's range is read and printed every second.
- The loop can be exited with a CTRL+C interrupt, which is handled by the KeyboardInterrupt exception.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.

- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.22 Lesson 22: Touch Sensor Module

In this lesson, you will learn how to connect and program a touch sensor with the Raspberry Pi using Python. The focus will be on setting up the sensor on GPIO pin 17 and writing a simple script to detect and respond to touch and release events. This practical session is aimed at teaching the basics of sensor integration and event handling in Python, providing you with the skills needed for more advanced sensor-based projects. It's an ideal starting point for those new to working with electronics and the Raspberry Pi.

Required Components

In this project, we need the following components.

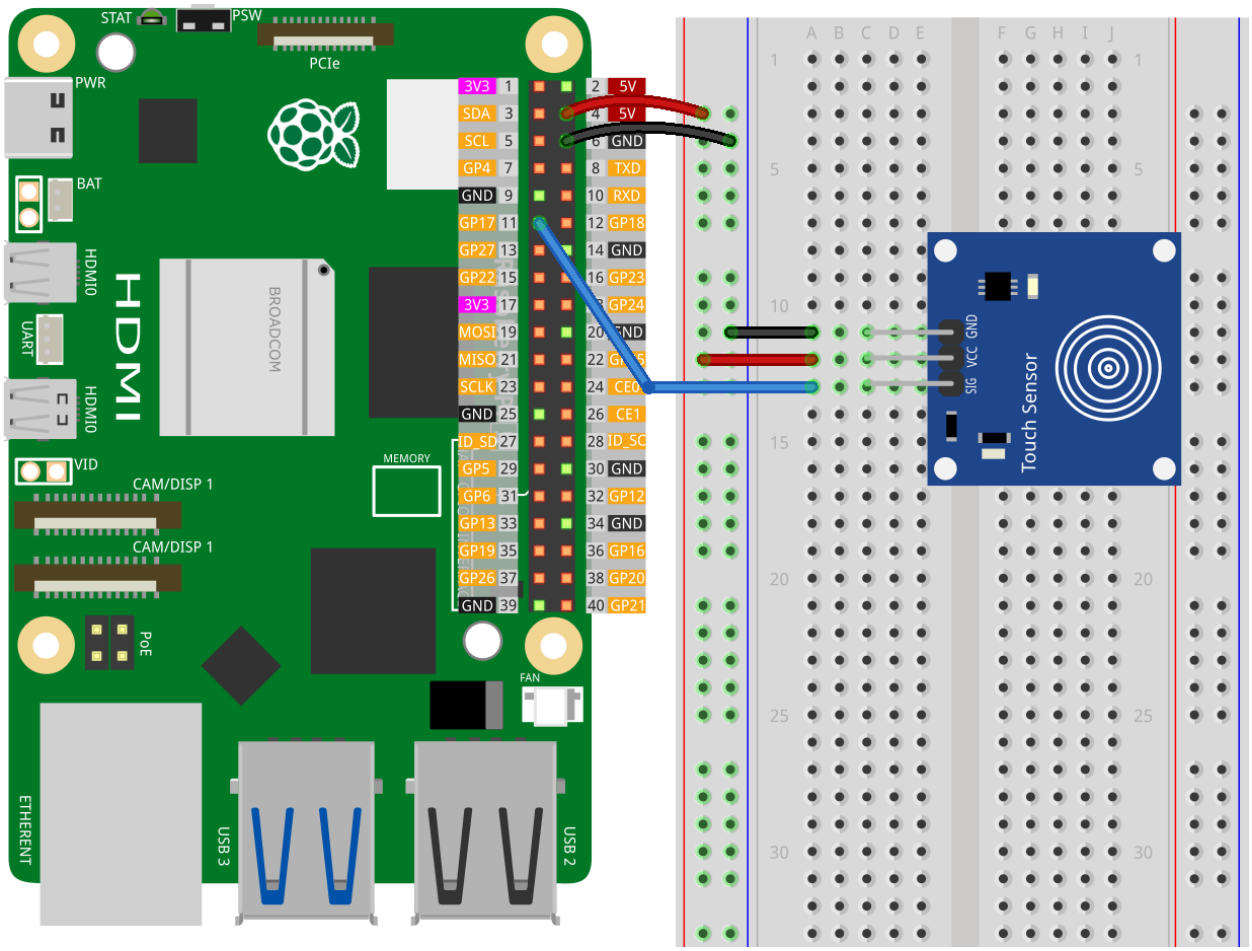
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Touch Sensor Module	
Breadboard	

Wiring



Code

```
from gpiozero import Button
from signal import pause

# Function called when the sensor is touched
def touched():
    # Print a message indicating the sensor is touched
    print("Touched!")

# Function called when the sensor is not touched
def not_touched():
    # Print a message indicating the sensor is not touched
    print("Not touched!")

# Initialize a Button object for the touch sensor
# GPIO 17: pin connected to the sensor
# pull_up=None: disable internal pull-up/pull-down resistors
# active_state=True: high voltage is considered the active state
```

(continues on next page)

(continued from previous page)

```
touch_sensor = Button(17, pull_up=None, active_state=True)

# Assign functions to sensor events
touch_sensor.when_pressed = touched
touch_sensor.when_released = not_touched

pause()  # Keep the program running to detect touch events
```

Code Analysis

1. Importing Libraries

   The script starts by importing the `Button` class from gpiozero for interfacing with the touch sensor, and `pause` from the signal module to keep the program running and responsive to events.

   ```
   from gpiozero import Button
   from signal import pause
   ```

2. Defining Callback Functions

   Two functions, `touched` and `not_touched`, are defined to handle touch and release events from the sensor. Each function prints a message indicating the sensor's state.

   ```
   def touched():
       print("Touched!")

   def not_touched():
       print("Not touched!")
   ```

3. Initializing the Touch Sensor

   A `Button` object named `touch_sensor` is created for the touch sensor on GPIO pin 17. The `pull_up` parameter is set to `None` to disable internal pull-up/pull-down resistors, and `active_state` is set to `True` to consider high voltage as the active state.

   ```
   touch_sensor = Button(17, pull_up=None, active_state=True)
   ```

4. Assigning Functions to Sensor Events

   The `when_pressed` event of the `touch_sensor` is linked to the `touched` function, and the `when_released` event is linked to the `not_touched` function. This setup allows the script to react to touch and release events from the sensor.

   ```
   touch_sensor.when_pressed = touched
   touch_sensor.when_released = not_touched
   ```

5. Keeping the Program Running

   The `pause()` function is called to keep the program running indefinitely. This is necessary to continuously monitor and respond to touch sensor events.

   ```
   pause()
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.23 Lesson 23: Ultrasonic Sensor Module (HC-SR04)

In this lesson, you'll learn how to connect an ultrasonic distance sensor to a Raspberry Pi and write a Python script for reading distance measurements. We'll guide you through the process of wiring the sensor's trigger pin to GPIO 17 and the echo pin to GPIO 27. The provided Python code will assist you in measuring distances and displaying them in centimeters.

Required Components

In this project, we need the following components.

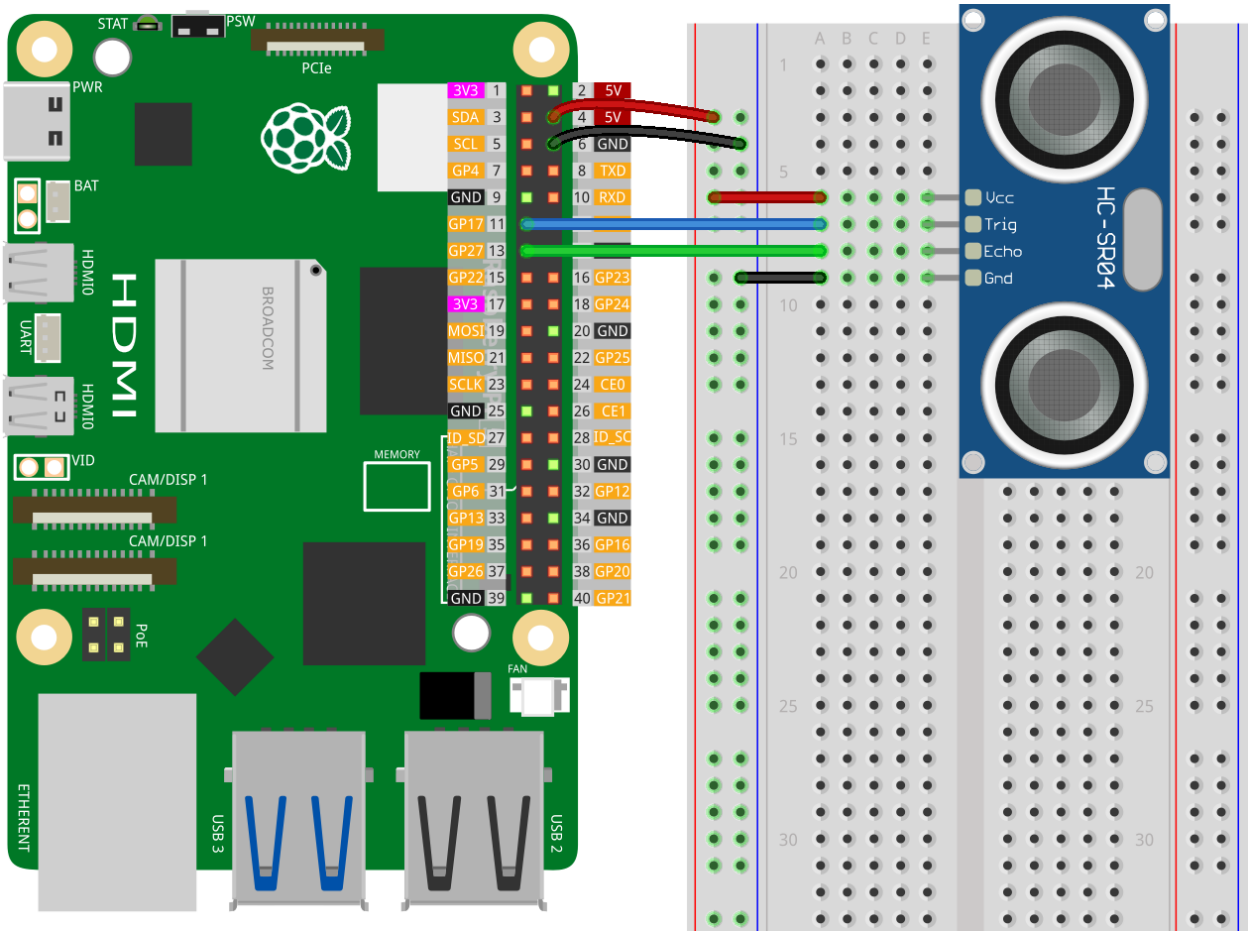
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Ultrasonic Sensor Module (HC-SR04)	
Breadboard	

Wiring



Code

```
#!/usr/bin/env python3
from gpiozero import DistanceSensor
from time import sleep

# Initialize the DistanceSensor using GPIO Zero library
# Trigger pin is connected to GPIO 17, Echo pin to GPIO 27
sensor = DistanceSensor(echo=27, trigger=17)

try:
    # Main loop to continuously measure and report distance
    while True:
        dis = sensor.distance * 100  # Measure distance and convert from meters to␣
↪centimeters
        print('Distance: {:.2f} cm'.format(dis))  # Print the distance with two decimal␣
↪precision
        sleep(0.3)  # Wait for 0.3 seconds before the next measurement

except KeyboardInterrupt:
```

(continues on next page)

(continued from previous page)

```
    # Handle KeyboardInterrupt (Ctrl+C) to gracefully exit the loop
    pass
```

Code Analysis

1. Importing Libraries

   The script begins by importing `DistanceSensor` from the gpiozero library for the ultrasonic sensor, and `sleep` from the time module for timing control.

   ```
   from gpiozero import DistanceSensor
   from time import sleep
   ```

2. Initializing the Distance Sensor

   A `DistanceSensor` object named `sensor` is created with `echo` and `trigger` pins connected to GPIO 27 and GPIO 17, respectively. These pins are used to send and receive the ultrasonic signals for distance measurement.

   ```
   sensor = DistanceSensor(echo=27, trigger=17)
   ```

3. Implementing the Continuous Monitoring Loop

   - A `try` block with an infinite loop (`while True:`) is used to continuously measure the distance.
   - Within the loop, `sensor.distance` gives the measured distance in meters, which is then converted to centimeters and stored in `dis`.
   - The distance is printed with two decimal points of precision using the `format` method.
   - `sleep(0.3)` adds a 0.3-second delay between each measurement to control the frequency of readings and reduce CPU load.

   ```
   try:
       while True:
           dis = sensor.distance * 100
           print('Distance: {:.2f} cm'.format(dis))
           sleep(0.3)
   ```

4. Handling KeyboardInterrupt for Graceful Exit

   The `except` block is used to catch a KeyboardInterrupt (typically Ctrl+C). When this occurs, the script exits the loop gracefully without any additional actions.

   ```
   except KeyboardInterrupt:
       pass
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.

- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.24 Lesson 24: Vibration Sensor Module (SW-420)

In this lesson, you will learn how to use a vibration sensor with the Raspberry Pi. We'll help you connect the sensor to GPIO pin 17 and guide you through writing a simple Python script. This script will monitor the sensor and print a message whenever vibration is detected. This lesson is focused on giving beginners a hands-on experience in connecting a simple sensor to the Raspberry Pi and writing a straightforward script to interact with it.

Required Components

In this project, we need the following components.

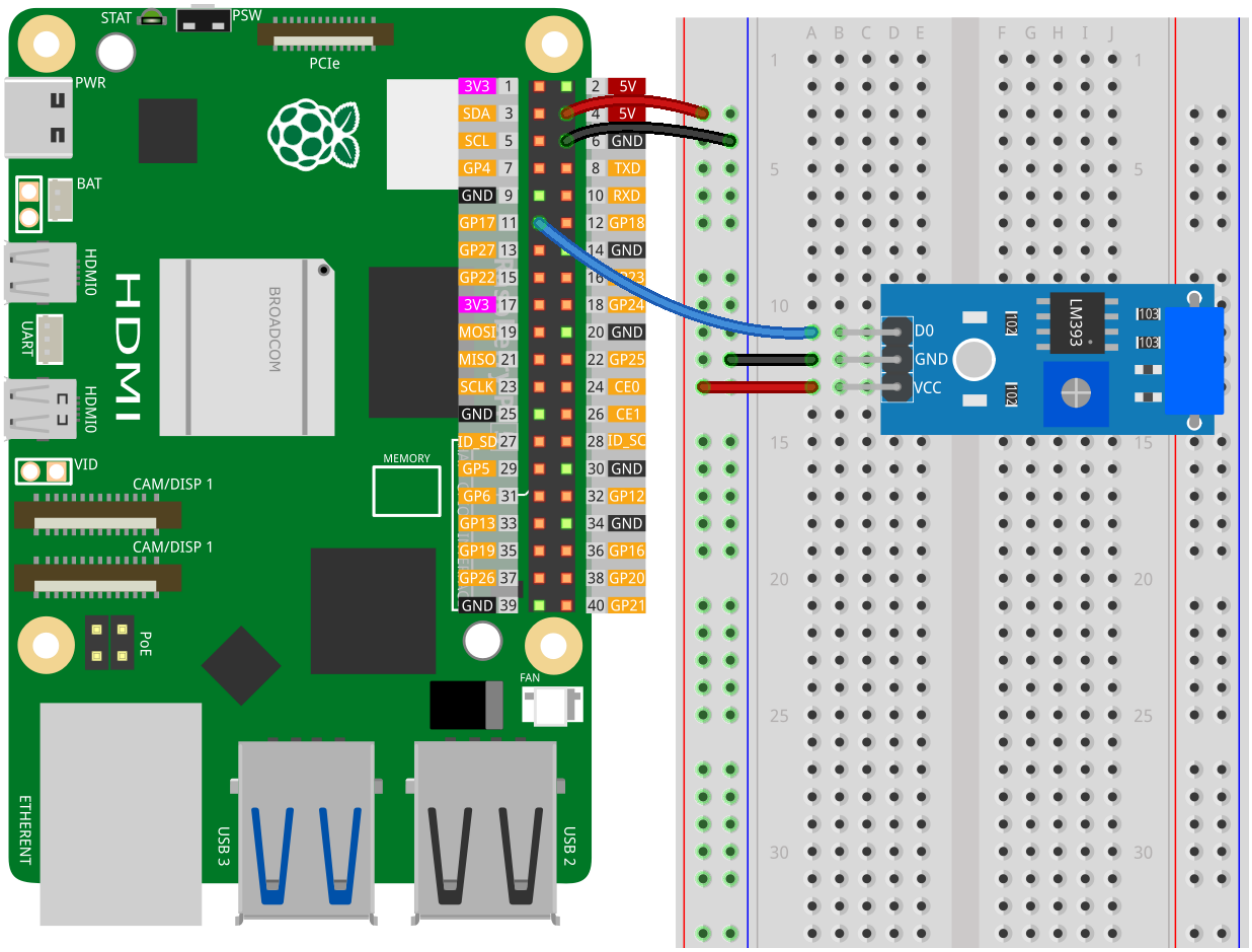
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Vibration Sensor Module (SW-420)	
Breadboard	

Wiring



Code

```
from gpiozero import InputDevice
import time

# Connect the digital output of the vibration sensor to GPIO17 on the Raspberry Pi
vibration_sensor = InputDevice(17)

# Continuous loop to read from the sensor
while True:
    # Check if the sensor is active (no vibration detected)
    if vibration_sensor.is_active:
        print("Vibration detected!")
    else:
        # When the sensor is inactive (vibration detected)
        print("...")
    # Wait for 1 second before reading the sensor again
    time.sleep(1)
```

Code Analysis

1. **Importing Libraries**

   First, we import necessary libraries: `gpiozero` for interacting with the GPIO pins, and `time` for handling time-related functions.

```
from gpiozero import InputDevice
import time
```

2. **Setting Up the Vibration Sensor**

   We initialize the vibration sensor by creating an instance of `InputDevice` from the `gpiozero` library. The vibration sensor is connected to GPIO pin 17 on the Raspberry Pi.

```
vibration_sensor = InputDevice(17)
```

3. **Continuous Monitoring Loop**

   A `while True` loop is used for continuous monitoring. This loop will run indefinitely until the program is manually stopped.

```
while True:
```

4. **Sensor State Check and Output**

   - Inside the loop, we use an `if` statement to check the state of the vibration sensor. If `vibration_sensor.is_active` is `True`, it means no vibration is detected, and "Vibration detected!" is printed.
   - If `vibration_sensor.is_active` is `False`, indicating vibration, "..." is printed instead.
   - This distinction is crucial for understanding how the sensor's output is interpreted in the code.

```
if vibration_sensor.is_active:
    print("Vibration detected!")
else:
    print("...")
```

5. **Delay**

   Finally, `time.sleep(1)` adds a 1-second delay between each iteration of the loop. This delay is crucial to prevent the program from overloading the CPU and to make the output readable.

```
time.sleep(1)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.25 Lesson 25: Water Level Sensor Module

Note: The Raspberry Pi does not have analog input capabilities, so it needs a module like the *PCF8591 ADC DAC Converter Module* to read analog signals for processing.

In this lesson, we'll learn how to read from a water level sensor using a Raspberry Pi. You'll find out how to connect a water level sensor module to the PCF8591 for analog-to-digital conversion and monitor its output in real-time with Python.

Required Components

In this project, we need the following components.

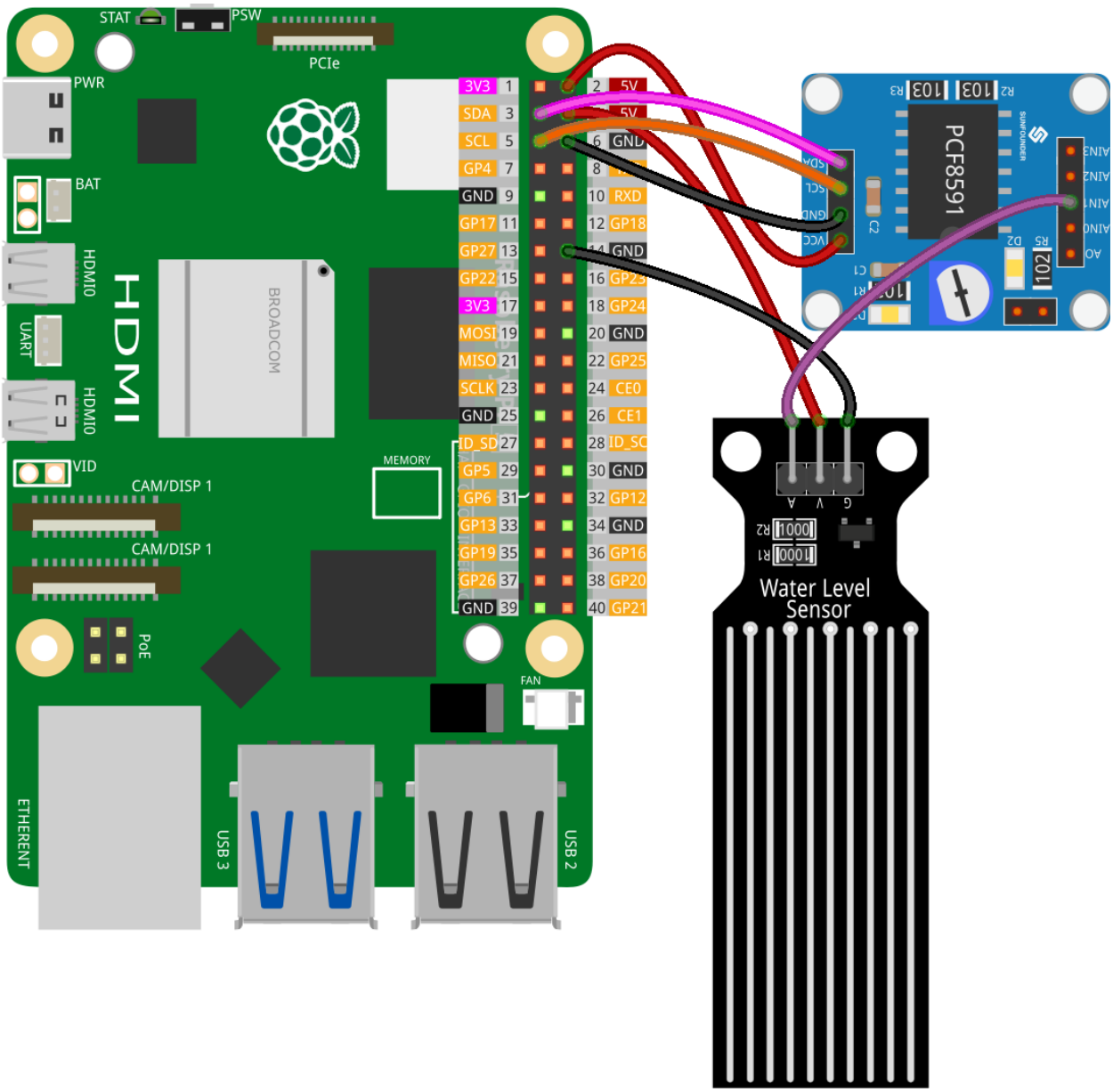
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Water Level Sensor Module	-
PCF8591 ADC DAC Converter Module	

Wiring



Code

```
import PCF8591 as ADC  # Import PCF8591 module
import time  # Import time for delay

ADC.setup(0x48)  # Initialize PCF8591 at address 0x48

try:
    while True:  # Continuously read and print
        print(ADC.read(1))  # Read from Water level sensor module at AIN1
```

(continues on next page)

(continued from previous page)

```
        time.sleep(0.2)  # Delay of 0.2 seconds
except KeyboardInterrupt:
    print("Exit")  # Exit on CTRL+C
```

Code Analysis

1. **Import Libraries**:

   This section imports necessary Python libraries. The `PCF8591` library is used for interacting with the PCF8591 module, and `time` is for implementing delays in the code.

   ```
   import PCF8591 as ADC  # Import PCF8591 module
   import time  # Import time for delay
   ```

2. **Initialize PCF8591 Module**:

   Here, the PCF8591 module is initialized. The address `0x48` is the I^2C address of the PCF8591 module. This is necessary for the Raspberry Pi to communicate with the module.

   ```
   ADC.setup(0x48)  # Initialize PCF8591 at address 0x48
   ```

3. **Main Loop and Reading Data**:

   The `try` block includes a continuous loop that consistently reads data from the water level sensor module. The `ADC.read(1)` function captures the analog input from the sensor connected to channel 1 (AIN1) of the PCF8591 module. Incorporating a `time.sleep(0.2)` creates a 0.2-second pause between each reading. This not only helps in reducing CPU usage on the Raspberry Pi by avoiding excessive data processing demands, but also prevents the terminal from being overrun with rapidly scrolling information, making it easier to monitor and analyze the output.

   ```
   try:
       while True:  # Continuously read and print
           print(ADC.read(1))  # Read from Water level sensor module at AIN1
           time.sleep(0.2)  # Delay of 0.2 seconds
   ```

4. **Handling Keyboard Interrupt**:

   The `except` block is designed to catch a KeyboardInterrupt (like pressing CTRL+C). When this interrupt occurs, the script prints "exit" and stops running. This is a common way to gracefully exit a continuously running script in Python.

   ```
   except KeyboardInterrupt:
       print("exit")  # Exit on CTRL+C
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.

- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.26 Lesson 26: I2C LCD 1602

In this lesson, you will learn the basics of displaying text on an LCD screen using a Raspberry Pi. We'll start by showing you how to connect a standard LCD to the Raspberry Pi using the I2C interface. You'll learn how to set up the LCD with simple parameters like the Raspberry Pi model and I2C address. Then, we'll walk you through writing a basic Python script to display messages like "Hello World!" on the screen. This straightforward project is aimed at beginners, offering a foundational introduction to interfacing hardware with the Raspberry Pi and basic Python programming.

Required Components

In this project, we need the following components.

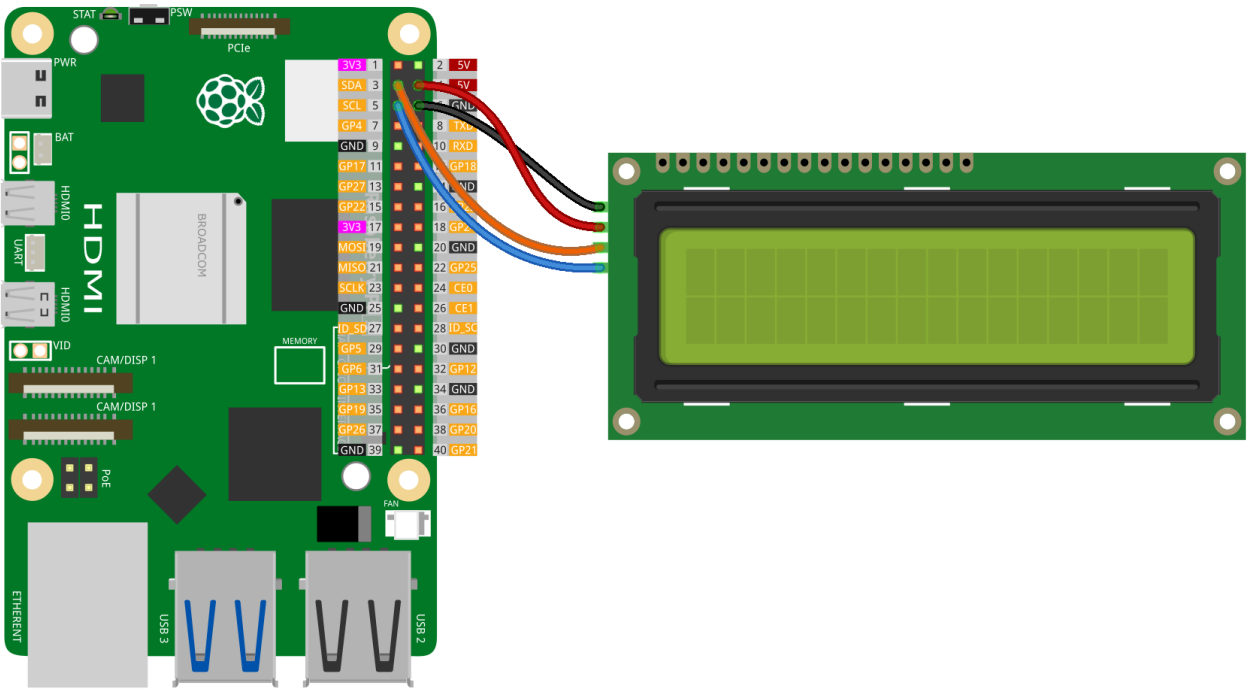
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
I2C LCD 1602	

Wiring



Code

```
import time
from LCD import LCD

# Initialize the LCD with specific parameters: Raspberry Pi revision, I2C address, and␣
↪backlight status
lcd = LCD(2, 0x27, True)  # Using Raspberry Pi revision 2, I2C address 0x27, backlight␣
↪enabled

# Display messages on the LCD
lcd.message("Hello World!", 1)         # Display 'Hello World!' on line 1
lcd.message("    - Sunfounder", 2)     # Display '    - Sunfounder' on line 2

# Keep the messages displayed for 5 seconds
time.sleep(5)

# Clear the LCD display
lcd.clear()
```

Code Analysis

1. Import Libraries

   Import the `time` module for creating delays and the `LCD` module for controlling the LCD.

   For more detail about the `LCD` library, please refer to .

   ```
   import time
   from LCD import LCD
   ```

2. Initialize the LCD

   Create an `LCD` object with specific parameters: the Raspberry Pi revision, the I2C address of the LCD, and the backlight status. In this case, Raspberry Pi revision 2(and higher version), I2C address 0x27, and backlight enabled.

   ```
   lcd = LCD(2, 0x27, True)
   ```

3. Display Messages on the LCD

   Use the `message` method of the `LCD` object to display text on the LCD. The first argument is the text, and the second argument is the line number.

   ```
   lcd.message("Hello World!", 1)
   lcd.message("    - Sunfounder", 2)
   ```

4. Keep the Messages Displayed

   Pause the program for 5 seconds, keeping the messages on the LCD during this time.

   ```
   time.sleep(5)
   ```

5. Clear the LCD Display

   After the delay, clear the display using the `clear` method of the `LCD` object.

   ```
   lcd.clear()
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.27 Lesson 27: OLED Display Module (SSD1306)

In this lesson, you will learn how to connect a Raspberry Pi with an OLED Display Module (SSD1306) using Python. You'll learn how to establish I2C communication between the Raspberry Pi and the OLED display, and use the Python Imaging Library (PIL) for creating graphics and text. The lesson will guide you through drawing shapes and text on the OLED screen, providing a practical example with the message "Hello World!".

Required Components

In this project, we need the following components.

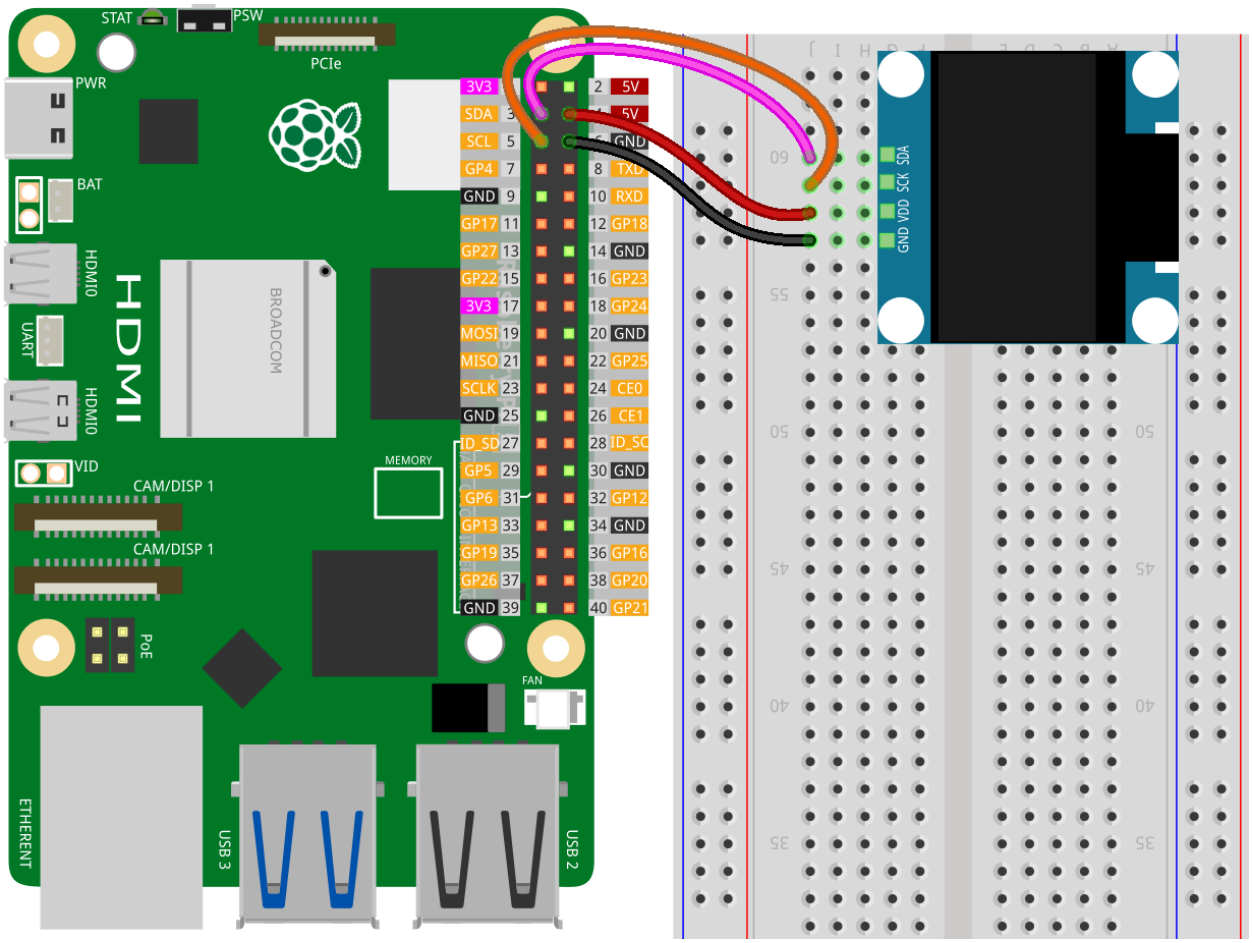
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
OLED Display Module (SSD1306)	-
Breadboard	

Wiring



Install Library

Note: The adafruit-circuitpython-ssd1306 library relies on Blinka, so please ensure that Blinka has been installed. To install libraries, refer to *Insatll Adafruit_Blinka (CircuitPython) - Optional*.

Before installing the library, please make sure that the virtual Python environment is activated:

```
source ~/env/bin/activate
```

Install adafruit-circuitpython-ssd1306 library:

```
pip install adafruit-circuitpython-ssd1306
```

Run the Code

Note:

- Please ensure that you have installed the Python library required for running the code according to the "Install Library" steps.
- Before running the code, please make sure that you have activated the virtual Python environment with blinka installed. You can activate the virtual environment using a command like this:

```
source ~/env/bin/activate
```

- Find the code for this lesson in `universal-maker-sensor-kit-main/pi/` directory, or directly copy and paste the code below. Execute the code by running the following commands in terminal:

```
python 27_ssd1306_oled_module.py
```

```
import board
import digitalio
from PIL import Image, ImageDraw, ImageFont
import adafruit_ssd1306

# Initialize OLED display dimensions
WIDTH = 128
HEIGHT = 64

# Set up I2C communication with the OLED display
i2c = board.I2C()  # Utilizes board's SCL and SDA pins
oled = adafruit_ssd1306.SSD1306_I2C(WIDTH, HEIGHT, i2c, addr=0x3C)

# Clear the OLED display
oled.fill(0)
oled.show()

# Create a new image with 1-bit color for drawing
image = Image.new("1", (oled.width, oled.height))

# Obtain a drawing object to manipulate the image
draw = ImageDraw.Draw(image)

# Draw a filled white rectangle as the background
draw.rectangle((0, 0, oled.width, oled.height), outline=255, fill=255)

# Define border size for an inner rectangle
BORDER = 5
# Draw a smaller black rectangle inside the larger one
draw.rectangle(
    (BORDER, BORDER, oled.width - BORDER - 1, oled.height - BORDER - 1),
    outline=0,
    fill=0,
)
```

(continues on next page)

(continued from previous page)

```
# Load the default font for text
font = ImageFont.load_default()

def getfontsize(font, text):
    # Calculate the size of the text in pixels
    left, top, right, bottom = font.getbbox(text)
    return right - left, bottom - top

# Define the text to be displayed
text = "Hello World!"
# Get the width and height of the text in pixels
(font_width, font_height) = getfontsize(font, text)
# Draw the text, centered on the display
draw.text(
    (oled.width // 2 - font_width // 2, oled.height // 2 - font_height // 2),
    text,
    font=font,
    fill=255,
)

# Send the image to the OLED display
oled.image(image)
oled.show()
```

Code Analysis

1. Importing Necessary Libraries

   Here, we import the libraries needed for the project. `board` is for interfacing with the Raspberry Pi hardware, `PIL` for image processing, and `adafruit_ssd1306` for controlling the OLED display.

   For more detail about the `adafruit_ssd1306` library, please refer to .

   ```
   import board
   import digitalio
   from PIL import Image, ImageDraw, ImageFont
   import adafruit_ssd1306
   ```

2. Initializing the OLED Display

   The OLED display dimensions are set, and I2C communication is established. The `adafruit_ssd1306.SSD1306_I2C` object is created to interact with the OLED.

   ```
   # Initialize OLED display dimensions
   WIDTH = 128
   HEIGHT = 64

   # Set up I2C communication with the OLED display
   i2c = board.I2C()
   oled = adafruit_ssd1306.SSD1306_I2C(WIDTH, HEIGHT, i2c, addr=0x3C)
   ```

3. Clearing the Display

   The OLED display is cleared by filling it with zeros (black).

```
# Clear the OLED display
oled.fill(0)
oled.show()
```

4. Creating an Image Buffer

   An image buffer is created using PIL. This is where the graphics are drawn before being displayed on the screen.

   The PIL(Python Imaging Library) adds image processing capabilities to your Python interpreter. For more detail, please refer to .

```
# Create a new image with 1-bit color for drawing
image = Image.new("1", (oled.width, oled.height))

# Obtain a drawing object to manipulate the image
draw = ImageDraw.Draw(image)
```

5. Drawing Graphics

   Here, a white rectangle (background) and a smaller black rectangle (border effect) are drawn on the image buffer.

```
# Draw a filled white rectangle as the background
draw.rectangle((0, 0, oled.width, oled.height), outline=255, fill=255)

# Define border size for an inner rectangle
BORDER = 5
# Draw a smaller black rectangle inside the larger one
draw.rectangle(
    (BORDER, BORDER, oled.width - BORDER - 1, oled.height - BORDER - 1),
    outline=0,
    fill=0,
)
```

6. Adding Text

   The default font is loaded, and a function to calculate the text size is defined. Then, “Hello World!” is centered and drawn on the image buffer.

```
# Load the default font for text
font = ImageFont.load_default()

def getfontsize(font, text):
    # Calculate the size of the text in pixels
    left, top, right, bottom = font.getbbox(text)
    return right - left, bottom - top

# Define the text to be displayed
text = "Hello World!"
# Get the width and height of the text in pixels
(font_width, font_height) = getfontsize(font, text)
# Draw the text, centered on the display
draw.text(
    (oled.width // 2 - font_width // 2, oled.height // 2 - font_height // 2),
    text,
    font=font,
```

(continues on next page)

(continued from previous page)

```
    fill=255,
)
```

7. Displaying the Image

   Finally, the image buffer is sent to the OLED display for visualization.

```
# Send the image to the OLED display
oled.image(image)
oled.show()
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.28 Lesson 28: RGB Module

In this lesson, you will learn how to control an RGB LED module with a Raspberry Pi. You'll learn how to use Python to change the LED's color to red, green, blue, and yellow, and then turn it off. This project is a straightforward introduction to working with RGB LEDs and GPIO interfacing, making it ideal for beginners starting with Raspberry Pi and Python programming.

Required Components

In this project, we need the following components.

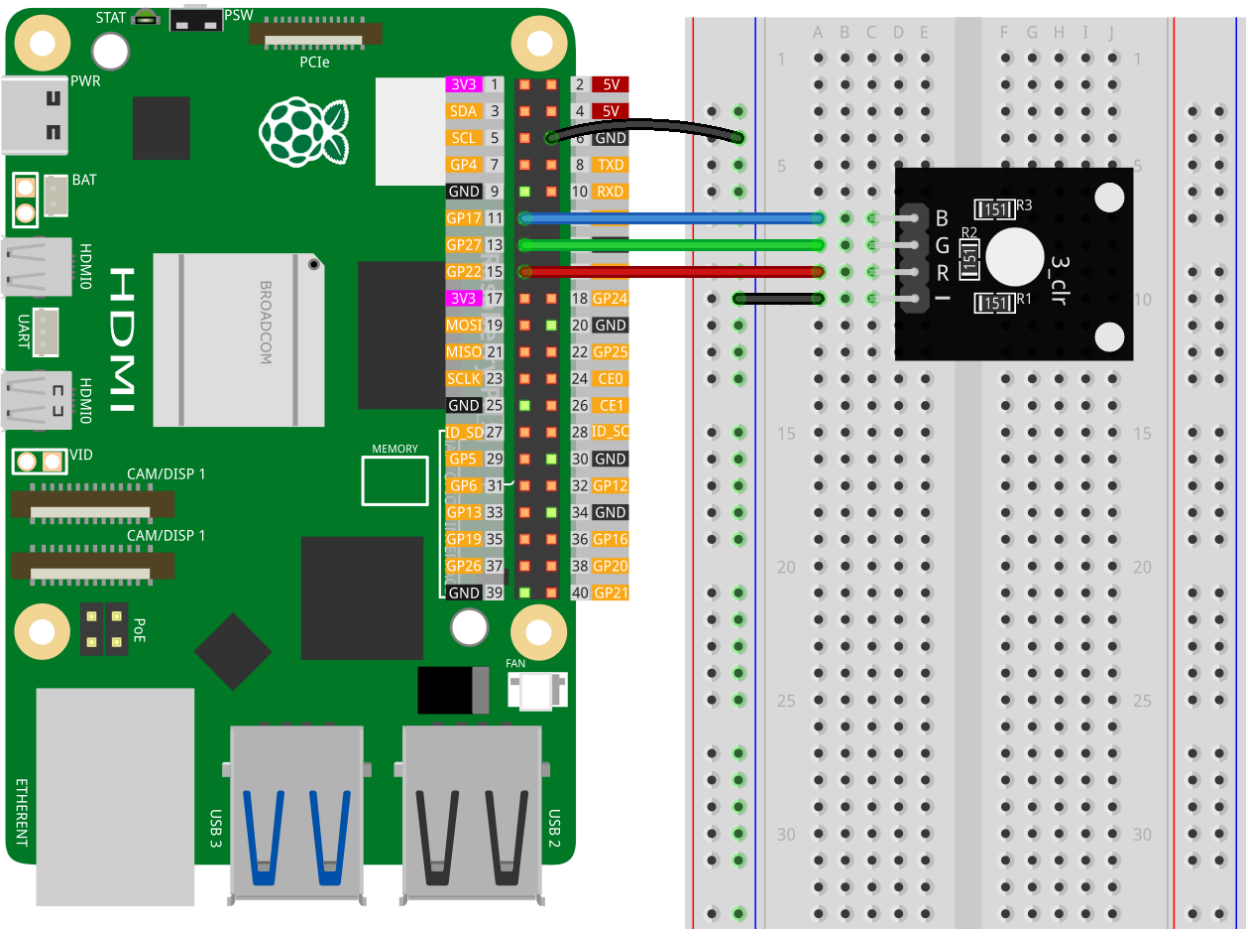
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
RGB LED Module	-
Breadboard	

Wiring



Code

```
from gpiozero import RGBLED
from time import sleep
from colorzero import Color

# GPIO pin assignments for the RGB LED
red_pin = 22
green_pin = 27
blue_pin = 17

# Initialize the RGB LED with red, green, and blue components connected to their␣
↪respective GPIO pins
led = RGBLED(red=red_pin, green=green_pin, blue=blue_pin)

# Set the LED to red color (red: 100%, green: 0%, blue: 0%) and wait for 1 second
led.color = (1, 0, 0)
sleep(1)

# Set the LED to green color (red: 0%, green: 100%, blue: 0%) and wait for 1 second
```

(continues on next page)

(continued from previous page)

```
led.color = (0, 1, 0)
sleep(1)

# Set the LED to blue color (red: 0%, green: 0%, blue: 100%) and wait for 1 second
led.color = (0, 0, 1)
sleep(1)

# Set the LED to yellow color using the Color class and wait for 1 second
led.color = Color('yellow')
sleep(1)

# Turn the LED off
led.off()
```

Code Analysis

1. Importing Libraries

   The script starts with importing the `RGBLED` class from gpiozero for controlling the RGB LED and the `sleep` function from the time module for delays. It also imports the `Color` class from colorzero for color definitions.

   ```
   from gpiozero import RGBLED
   from time import sleep
   from colorzero import Color
   ```

2. Initializing the RGB LED

   - GPIO pins for each color component of the RGB LED are defined.
   - The RGB LED is initialized with its red, green, and blue components connected to GPIO pins 22, 27, and 17 respectively.

   ```
   red_pin = 22
   green_pin = 27
   blue_pin = 17
   led = RGBLED(red=red_pin, green=green_pin, blue=blue_pin)
   ```

3. Setting LED Colors

   - The color of the LED is set to red, green, and blue in sequence, each followed by a 1-second pause.
   - Colors are represented by tuples (red, green, blue), where each value is between 0 and 1, indicating the intensity.

   ```
   led.color = (1, 0, 0)
   sleep(1)
   led.color = (0, 1, 0)
   sleep(1)
   led.color = (0, 0, 1)
   sleep(1)
   ```

4. Using the Color Class

   The script demonstrates how to use the `Color` class from colorzero to set the LED to a named color (`yellow`) and then waits for 1 second.

In addition to using the pre-defined colors directly, you can also define colors in various ways. For more details, please refer to .

```
led.color = Color('yellow')
sleep(1)
```

5. Turning the LED Off

   Finally, the script turns off the LED using `led.off()`.

```
led.off()
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.29 Lesson 29: Traffic Light Module

In this lesson, you will learn to simulate traffic lights using a Raspberry Pi. You'll program the Raspberry Pi to control these LEDs in a sequence that resembles traffic lights: the red LED will be active for 3 seconds, the yellow LED will blink in a specific pattern, and then the green LED will turn on for 3 seconds. This project is a practical way to get started with GPIO interfacing and Python programming, suitable for those new to combining hardware and software with the Raspberry Pi.

Required Components

In this project, we need the following components.

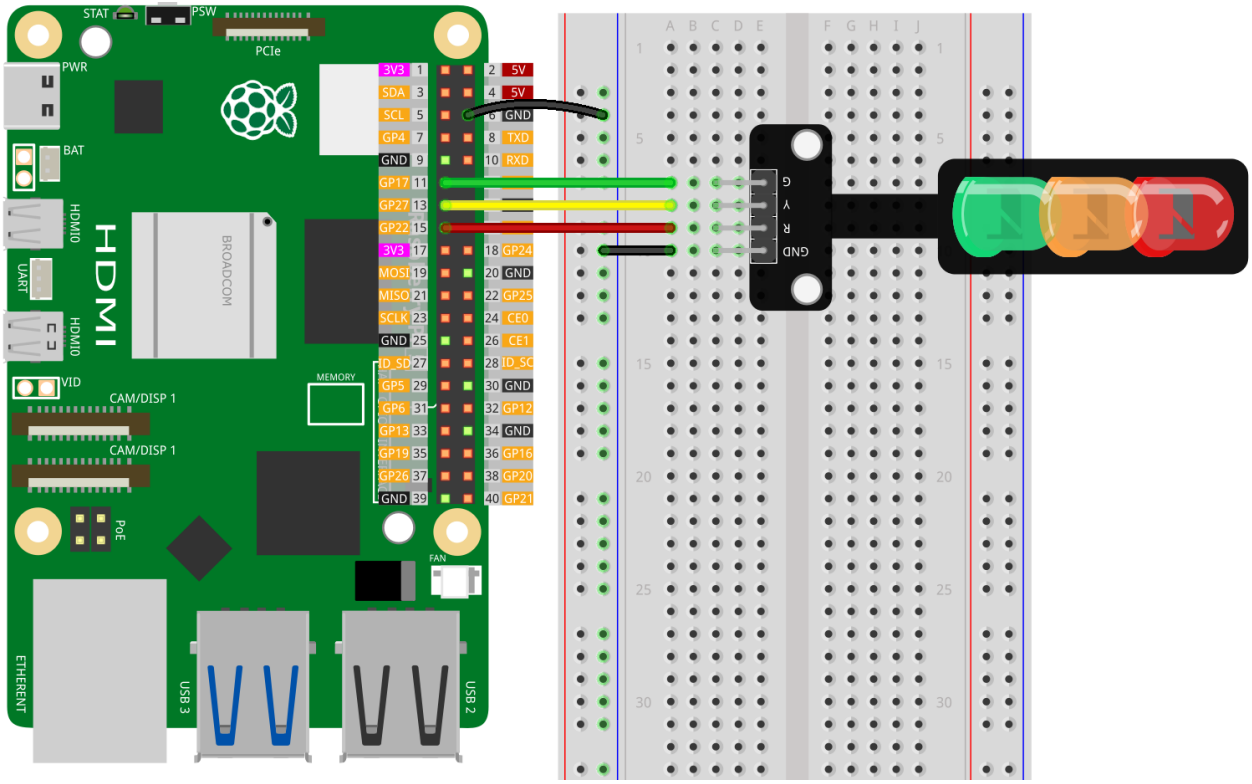
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Traffic Light Module	
Breadboard	

Wiring



Code

```
from gpiozero import LED
from time import sleep

# Initialize LED pins
red = LED(22)    # Red LED connected to GPIO pin 22
yellow = LED(27) # Yellow LED connected to GPIO pin 27
green = LED(17)  # Green LED connected to GPIO pin 17

# LED control in a continuous loop
try:
    while True:
        # Red LED cycle
        red.on()     # Turn on red LED
```

(continues on next page)

(continued from previous page)

```
        sleep(3)     # Red LED on for 3 seconds
        red.off()    # Turn off red LED

        # Yellow LED blinking pattern
        yellow.on()  # Turn on yellow LED
        sleep(0.5)   # Yellow LED on for 0.5 second
        yellow.off() # Turn off yellow LED
        sleep(0.5)   # Off for 0.5 second
        yellow.on()  # Repeat blinking
        sleep(0.5)   # Yellow LED on for 0.5 second
        yellow.off() # Turn off yellow LED
        sleep(0.5)   # Off for 0.5 second
        yellow.on()  # Repeat blinking
        sleep(0.5)   # Yellow LED on for 0.5 second
        yellow.off() # Turn off yellow LED
        sleep(0.5)   # Off for 0.5 second

        # Green LED cycle
        green.on()   # Turn on green LED
        sleep(3)     # Green LED on for 3 seconds
        green.off()  # Turn off green LED

except KeyboardInterrupt:
    # Turn off all LEDs and exit safely on keyboard interrupt
    red.off()
    yellow.off()
    green.off()
```

Code Analysis

1. Import Libraries

   The `gpiozero` library is imported to control the GPIO pins, and the `time` library's `sleep` function is used for timing delays.

   ```
   from gpiozero import LED
   from time import sleep
   ```

2. Initialize LED pins

   Here, each LED is associated with a specific GPIO pin on the Raspberry Pi using the `LED` class from the `gpiozero` library.

   ```
   red = LED(22)    # Red LED connected to GPIO pin 22
   yellow = LED(27) # Yellow LED connected to GPIO pin 27
   green = LED(17)  # Green LED connected to GPIO pin 17
   ```

3. LED Control Loop

   The `while True:` loop runs continuously, cycling through each LED. It turns each LED on and off in a specific pattern, using `on()`, `off()`, and `sleep()` functions.

   - Red LED is turned on for 3 seconds.

- Yellow LED blinks: 0.5 seconds on, 0.5 seconds off, repeated three times.
- Green LED is turned on for 3 seconds.

```
try:
    while True:
        # Red LED cycle
        red.on()
        sleep(3)
        red.off()

        # Yellow LED blinking pattern
        # [The pattern is repeated three times]

        # Green LED cycle
        green.on()
        sleep(3)
        green.off()
```

4. Exception Handling

   The `except` block catches a `KeyboardInterrupt` (usually generated by pressing Ctrl+C). It ensures all LEDs are turned off before the program exits, preventing the LEDs from being left in an undefined state.

```
except KeyboardInterrupt:
    red.off()
    yellow.off()
    green.off()
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.30 Lesson 30: Relay Module

In this lesson, you will learn how to control a relay module using a Raspberry Pi. You'll learn how to write a simple Python script to turn the relay on and off at one-second intervals. This project is a practical introduction to using GPIO pins for controlling external devices, providing a basic understanding of how relays work in electronic circuits. It's a straightforward and informative exercise, well-suited for beginners starting with Raspberry Pi and hardware control.

Required Components

In this project, we need the following components.

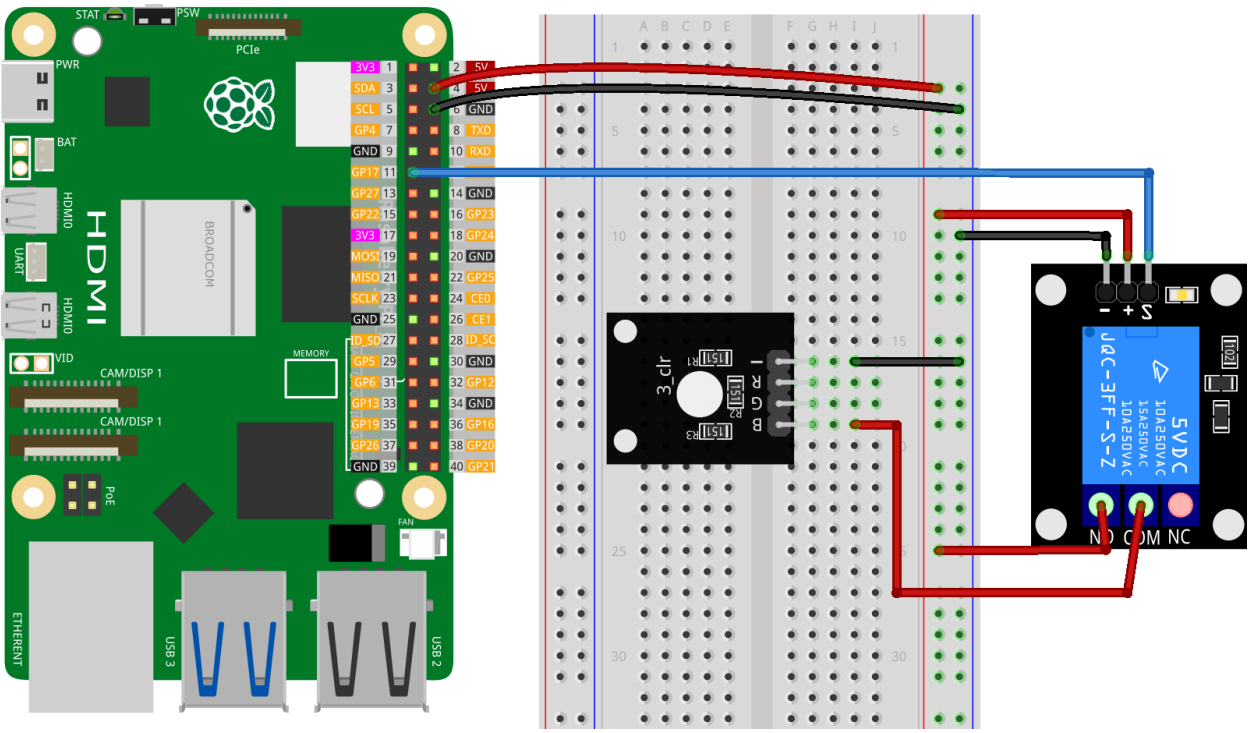
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
5V Relay Module	-
RGB LED Module	-
Breadboard	

Wiring



Code

```
from gpiozero import OutputDevice
from time import sleep

# Replace with your GPIO pin number
relay_pin = 17  # Example using GPIO17

# Initialize relay object
relay = OutputDevice(relay_pin)

try:
   while True:
      # Turn on the relay
      relay.on()
      sleep(1)  # Relay remains on for 1 second

      # Turn off the relay
      relay.off()
      sleep(1)  # Relay remains off for 1 second

except KeyboardInterrupt:
   # Capture Ctrl+C and safely close the program
   relay.off()
   print("Program interrupted by user")
```

Code Analysis

1. Import Libraries

   Import the `gpiozero` library for GPIO control and the `time` library for delays.

   ```
   from gpiozero import OutputDevice
   from time import sleep
   ```

2. Initialize the Relay

   Define the GPIO pin connected to the relay and initialize an `OutputDevice` object with that pin.

   ```
   relay_pin = 17  # Example using GPIO17
   relay = OutputDevice(relay_pin)
   ```

3. Relay Control in a Loop

   The `while True:` loop continuously toggles the relay. `relay.on()` and `relay.off()` are used to control the relay, and `sleep(1)` creates a one-second delay between each state.

   ```
   try:
       while True:
           relay.on()
           sleep(1)  # Relay remains on for 1 second
           relay.off()
           sleep(1)  # Relay remains off for 1 second
   ```

4. Exception Handling

   The `except` block captures a `KeyboardInterrupt` (Ctrl+C). It ensures the relay is turned off and the program exits safely.

```
except KeyboardInterrupt:
    relay.off()
    print("Program interrupted by user")
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.31 Lesson 31: Centrifugal Pump

In this lesson, you will learn how to control a pump using a Raspberry Pi. You'll learn how to write a Python script to activate the pump, control its speed, and then stop it after a set period. This project provides a basic understanding of pump control through GPIO interfacing and Python programming, making it a suitable starting point for beginners interested in Raspberry Pi and simple pump applications.

Required Components

In this project, we need the following components.

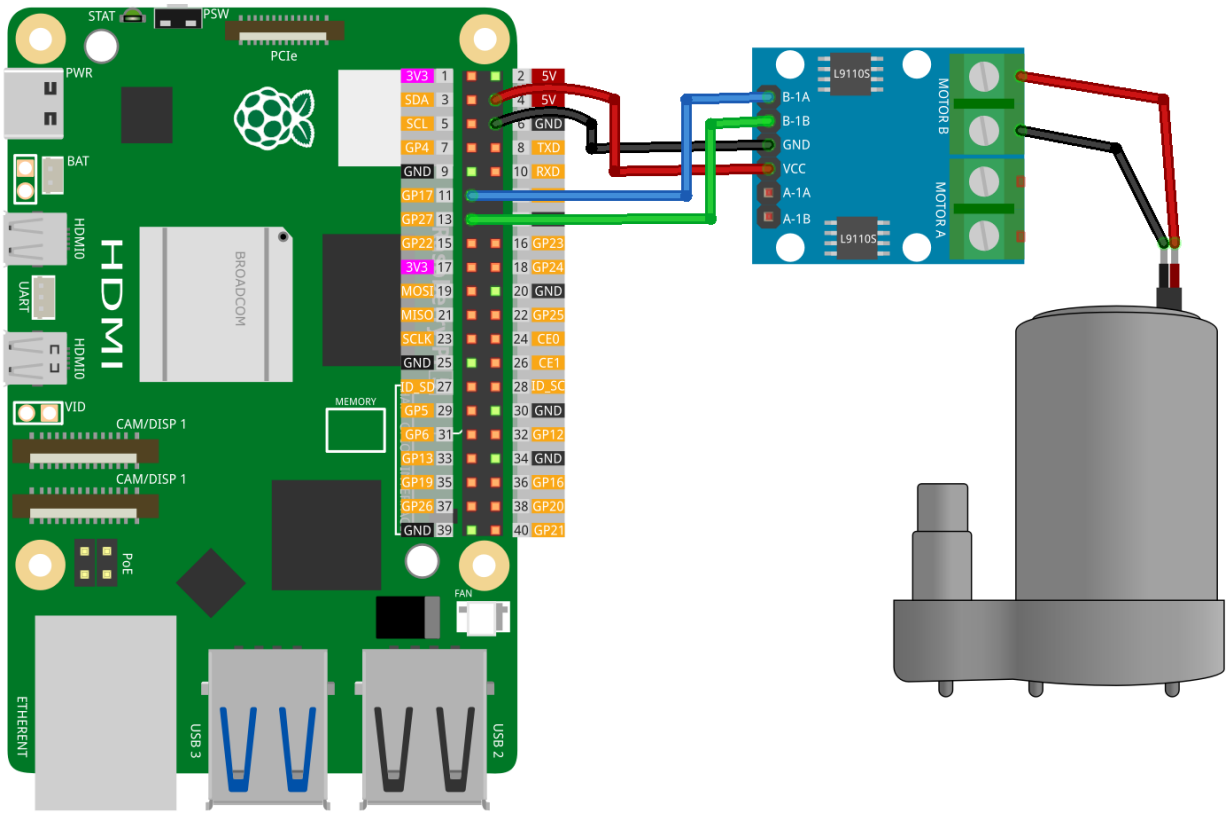
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Centrifugal Pump	-
L9110 Motor Driver Module	-

Wiring



Code

```
from gpiozero import Motor
from time import sleep

# Define pump pins
pump = Motor(forward=17, backward=27)  # Using Raspberry Pi GPIO pin numbers

# Activate the pump
pump.forward(speed=1)  # Set pump speed, range is 0 to 1
sleep(5)               # Run the pump for 5 seconds

# Deactivate the pump
pump.stop()            # Stop the pump
```

Code Analysis

1. Import Libraries

   The `gpiozero` library is used for controlling the motor, and the `time` library's `sleep` function is for delays.

   ```
   from gpiozero import Motor
   from time import sleep
   ```

2. Define Pump Pins

   A `Motor` object is created with two GPIO pins: one for forward and one for backward operation. In this case, GPIO 17 and 27 are used.

   ```
   pump = Motor(forward=17, backward=27)
   ```

3. Activate the pump

   The motor is activated in the forward direction with a specified speed using `pump.forward(speed=1)`. The speed parameter ranges from 0 (stopped) to 1 (full speed). The motor runs for 5 seconds, as defined by `sleep(5)`.

   ```
   pump.forward(speed=1)
   sleep(5)
   ```

4. Deactivate the pump

   The motor is stopped using `pump.stop()`. This is essential for safely halting the motor's operation after the required duration.

   ```
   pump.stop()
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.32 Lesson 32: Passive Buzzer Module

In this lesson, you will learn how to create musical tones using a TonalBuzzer with a Raspberry Pi. You'll learn how to program the Raspberry Pi to play a sequence of musical notes using Python. The lesson includes defining a tune as a list of notes and durations, and writing a function to play these notes through the buzzer. This project offers a straightforward introduction to working with sound output and Python programming, making it a practical choice for beginners interested in exploring musical applications with the Raspberry Pi.

Required Components

In this project, we need the following components.

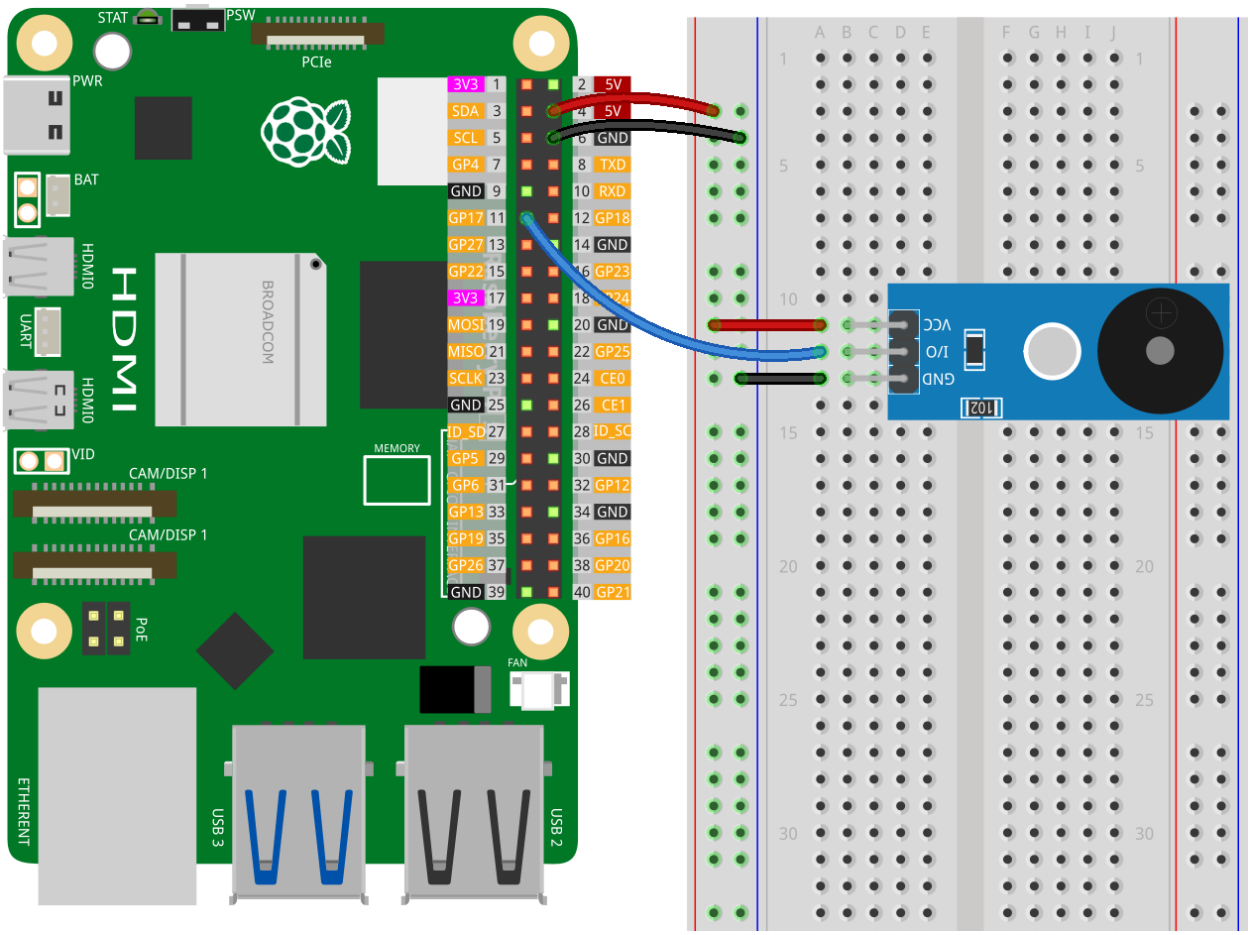
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Passive Buzzer Module	
Breadboard	

Wiring



Code

```
from gpiozero import TonalBuzzer
from time import sleep

# Initialize the TonalBuzzer on GPIO pin 17
tb = TonalBuzzer(17)  # Change to the pin number your buzzer is connected to

def play(tune):
    """
    Play a musical tune using the buzzer.
    :param tune: List of tuples, where each tuple contains a note and its duration.
    """
    for note, duration in tune:
        print(note)  # Print the current note being played
        tb.play(note)  # Play the note on the buzzer
        sleep(float(duration))  # Wait for the duration of the note
    tb.stop()  # Stop the buzzer after playing the tune

# Define the musical tune as a list of notes and their durations
```

(continues on next page)

(continued from previous page)

```
tune = [('C#4', 0.2), ('D4', 0.2), (None, 0.2),
   ('Eb4', 0.2), ('E4', 0.2), (None, 0.6),
   ('F#4', 0.2), ('G4', 0.2), (None, 0.6),
   ('Eb4', 0.2), ('E4', 0.2), (None, 0.2),
   ('F#4', 0.2), ('G4', 0.2), (None, 0.2),
   ('C4', 0.2), ('B4', 0.2), (None, 0.2),
   ('F#4', 0.2), ('G4', 0.2), (None, 0.2),
   ('B4', 0.2), ('Bb4', 0.5), (None, 0.6),
   ('A4', 0.2), ('G4', 0.2), ('E4', 0.2),
   ('D4', 0.2), ('E4', 0.2)]

# Play the tune
play(tune)
```

Code Analysis

1. Import Libraries

   Import `TonalBuzzer` from `gpiozero` for sound generation and `sleep` from `time` for timing control.

   ```
   from gpiozero import TonalBuzzer
   from time import sleep
   ```

2. Initialize the TonalBuzzer

   Create a `TonalBuzzer` object connected to GPIO pin 17.

   ```
   tb = TonalBuzzer(17)
   ```

3. Define the Play Function

   The `play` function takes a list of tuples as input, where each tuple represents a musical note and its duration. It iterates through each tuple, playing the note and waiting for its duration.

   ```
   def play(tune):
       for note, duration in tune:
           print(note)
           tb.play(note)
           sleep(float(duration))
       tb.stop()
   ```

4. Define the Musical Tune

   The tune is defined as a list of tuples. Each tuple contains a note and its duration in seconds. `None` is used to represent a pause.

   ```
   tune = [('C#4', 0.2), ('D4', 0.2), (None, 0.2), ...]
   ```

5. Play the Tune

   The `play` function is called with the `tune` list, causing the buzzer to play the defined sequence of notes.

   ```
   play(tune)
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.33 Lesson 33: Servo Motor (SG90)

In this lesson, you will learn how to control a servo motor using a Raspberry Pi. You'll learn how to adjust the servo's pulse width settings for precise control and write a Python script to move the servo to different positions: minimum, middle, and maximum.

Required Components

In this project, we need the following components.

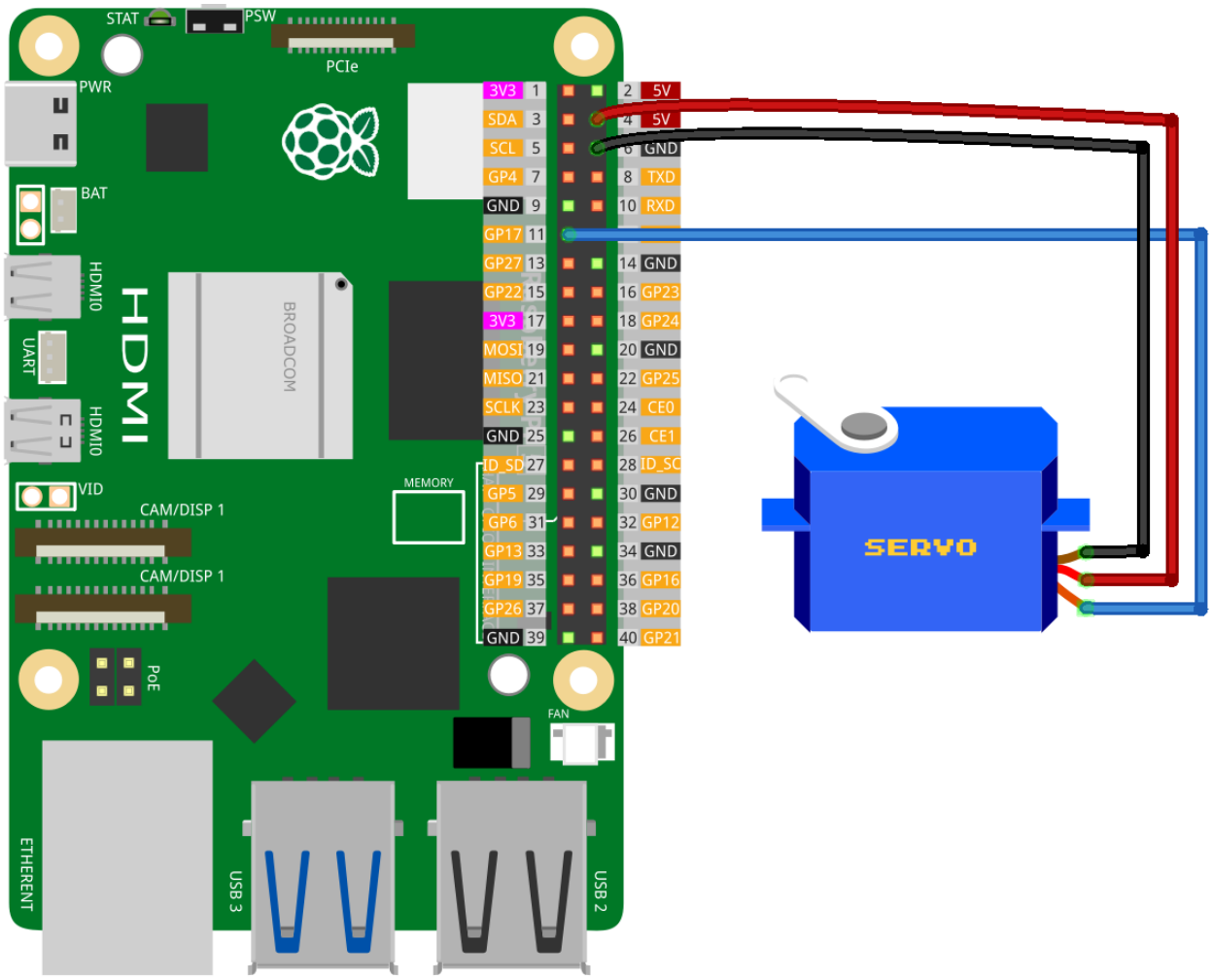
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
Servo Motor (SG90)	

Wiring



Code

```
from gpiozero import Servo
from time import sleep

# GPIO pin for the servo
myGPIO = 17

# Correction factor for the servo
myCorrection = 0.45
maxPW = (2.0 + myCorrection) / 1000  # Maximum pulse width
minPW = (1.0 - myCorrection) / 1000  # Minimum pulse width

# Initialize the servo with adjusted pulse width range
servo = Servo(myGPIO, min_pulse_width=minPW, max_pulse_width=maxPW)

# Continuously move servo between positions
```

(continues on next page)

(continued from previous page)

```
while True:
    # Move servo to middle position
    servo.mid()
    print("mid")
    sleep(0.5)

    # Move servo to minimum position
    servo.min()
    print("min")
    sleep(1)

    # Move servo to middle position
    servo.mid()
    print("mid")
    sleep(0.5)

    # Move servo to maximum position
    servo.max()
    print("max")
    sleep(1)
```

Code Analysis

1. Import Libraries

   Import the `Servo` class from `gpiozero` for servo control and `sleep` from `time` for timing.

   ```
   from gpiozero import Servo
   from time import sleep
   ```

2. GPIO Pin and Servo Correction Factor

   Define the GPIO pin connected to the servo and set a correction factor to calibrate the servo's pulse width range.

   ```
   myGPIO = 17
   myCorrection = 0.45
   maxPW = (2.0 + myCorrection) / 1000
   minPW = (1.0 - myCorrection) / 1000
   ```

3. Initialize the Servo

   Create a `Servo` object with the specified GPIO pin and adjusted pulse width range.

   ```
   servo = Servo(myGPIO, min_pulse_width=minPW, max_pulse_width=maxPW)
   ```

4. Move the Servo Continuously

   Use a `while True` loop to move the servo between its minimum, middle, and maximum positions, printing the current position and pausing between movements.

   ```
   while True:
       servo.mid()
       print("mid")
       sleep(0.5)
   ```

(continues on next page)

(continued from previous page)

```
servo.min()
print("min")
sleep(1)

servo.mid()
print("mid")
sleep(0.5)

servo.max()
print("max")
sleep(1)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.6.34 Lesson 34: TT Motor

In this lesson, you will learn how to control the speed and direction of a motor using a Raspberry Pi. You'll learn how to program the Raspberry Pi to run the motor at different speeds and in both forward and backward directions. The project will involve setting the motor speed, running it for a specified duration, and then stopping it. This exercise is a practical introduction to motor control with the Raspberry Pi, offering a clear and straightforward experience in hardware control and Python programming, suitable for beginners.

Required Components

In this project, we need the following components.

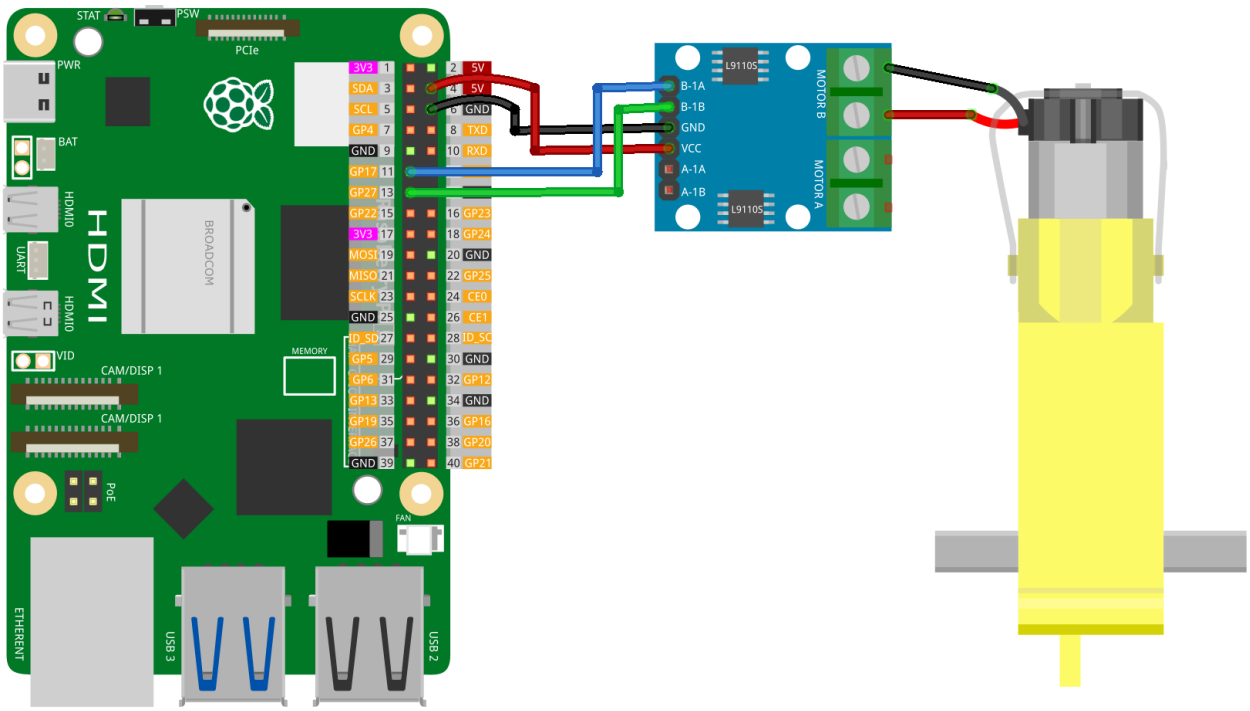
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Universal Maker Sensor Kit	94	

You can also buy them separately from the links below.

Component Introduction	Purchase Link
Raspberry Pi 5	-
TT Motor	-
L9110 Motor Driver Module	-

Wiring



Code

```
from gpiozero import Motor
from time import sleep

# Define motor pins
motor = Motor(forward=17, backward=27)  # Using Raspberry Pi GPIO pin numbers

# Run the motor forward at half speed
motor.forward(speed=0.5)  # Set motor speed, range is 0 to 1
sleep(5)                  # Run the motor for 5 seconds

# Increase to full speed forward
motor.forward(speed=1)    # Set motor speed, range is 0 to 1
sleep(5)                  # Run the motor for 5 seconds

# Run the motor backward at full speed
motor.backward(speed=1)   # Set motor speed, range is 0 to 1
sleep(5)                  # Run the motor for 5 seconds
```

(continues on next page)

(continued from previous page)

```
# Stop the motor
motor.stop()
```

Code Analysis

1. Import Libraries

   Import the `Motor` class from `gpiozero` for motor control, and `sleep` from `time` for timing control.

   ```
   from gpiozero import Motor
   from time import sleep
   ```

2. Define Motor Pins

   Create a `Motor` object to control a motor connected to GPIO pins 17 and 27 for forward and backward movements, respectively.

   ```
   motor = Motor(forward=17, backward=27)
   ```

3. Run the Motor Forward at Half Speed

   The motor is run forward at half speed (`speed=0.5`) for 5 seconds. The speed range is between 0 (stopped) and 1 (full speed).

   ```
   motor.forward(speed=0.5)
   sleep(5)
   ```

4. Increase to Full Speed Forward

   Increase the motor speed to full speed (`speed=1`) in the forward direction, running for another 5 seconds.

   ```
   motor.forward(speed=1)
   sleep(5)
   ```

5. Run the Motor Backward at Full Speed

   The motor is then run backward at full speed for 5 seconds.

   ```
   motor.backward(speed=1)
   sleep(5)
   ```

6. Stop the Motor

   Finally, stop the motor using the `stop` method.

   ```
   motor.stop()
   ```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.7 FAQ

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.8 Appendix

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

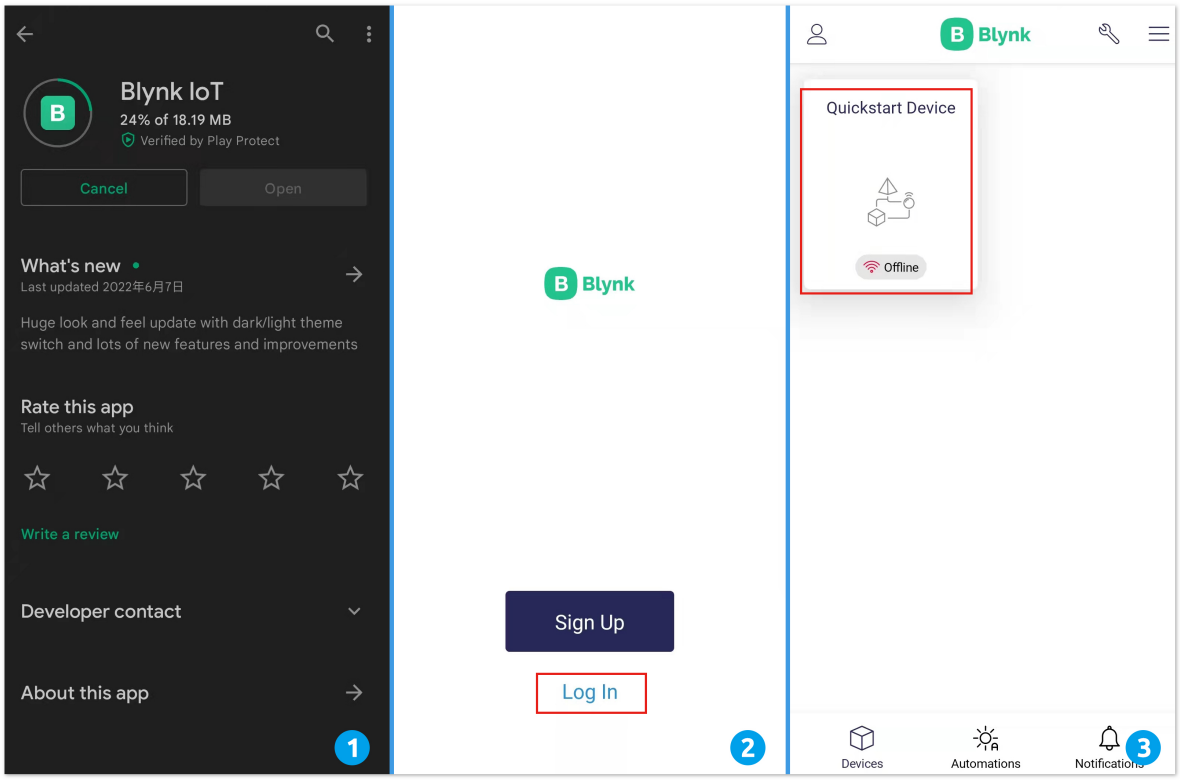
- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

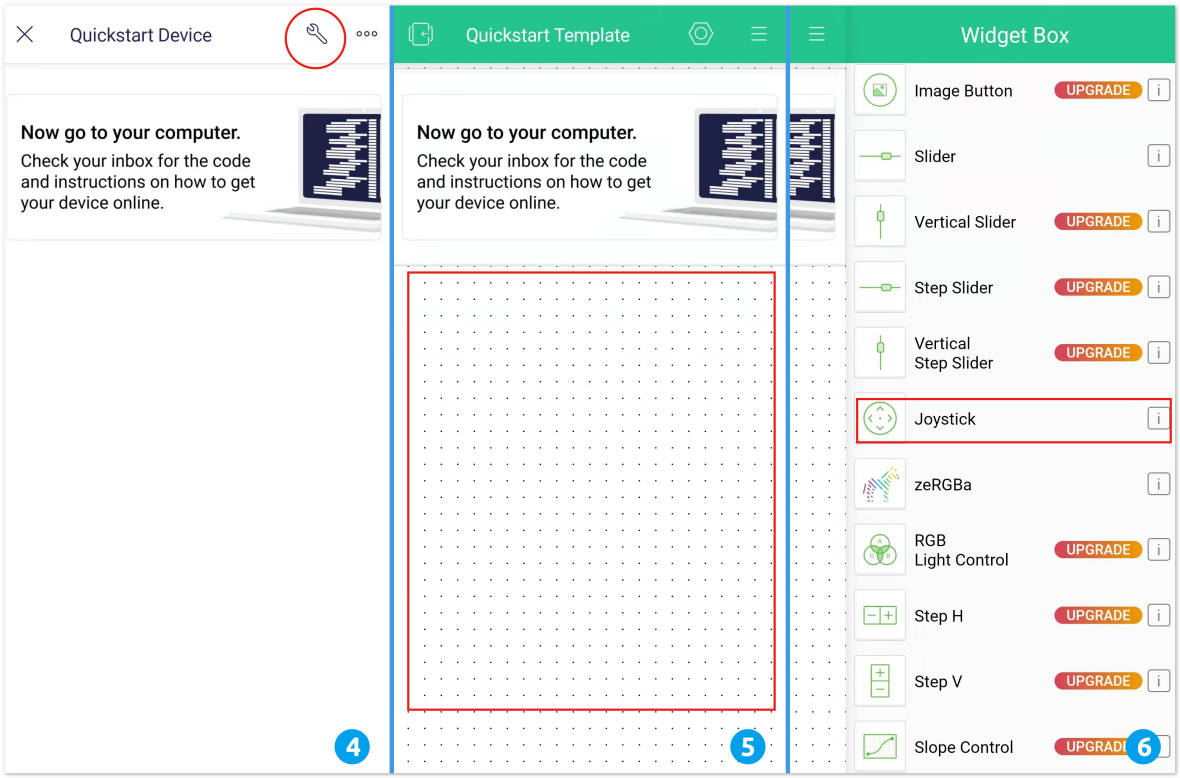
1.8.1 How to use Blynk on mobile device?

Note: As datastreams can only be created in Blynk on the web, you will need to reference different projects to create datastreams on the web, then follow the tutorial below to create widgets in Blynk on your mobile device.

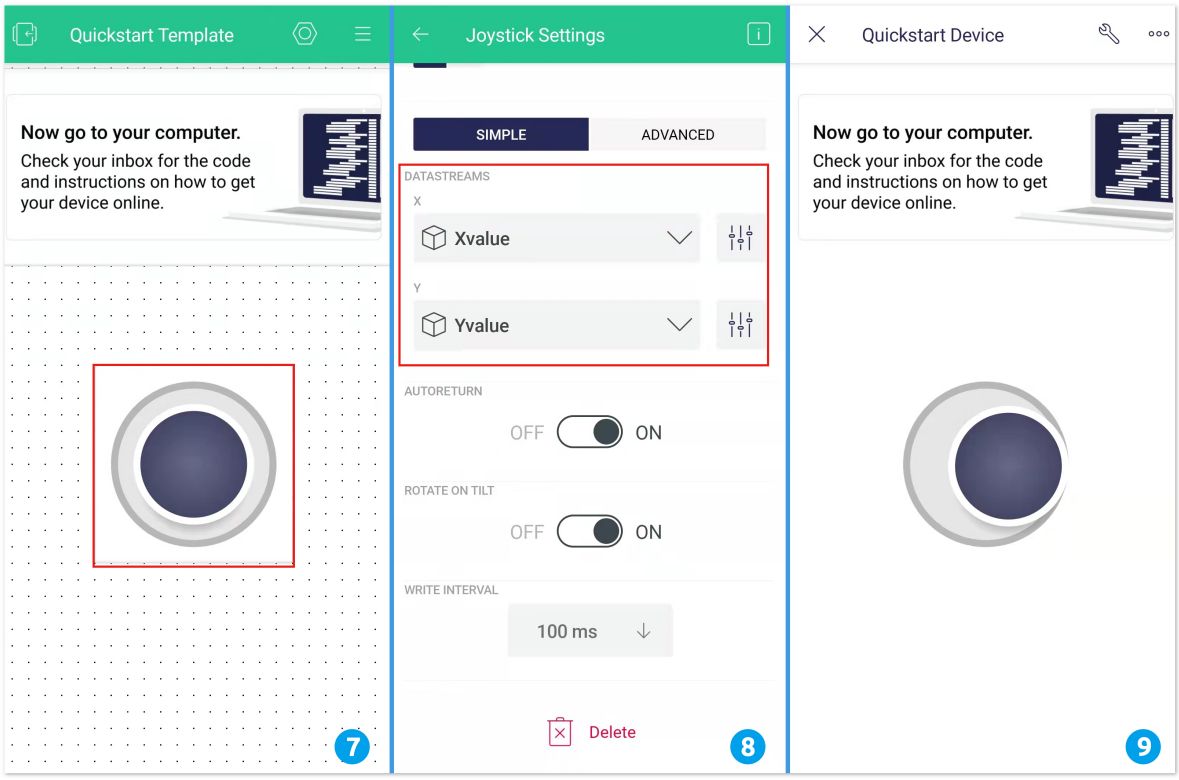
1. Open Google Play or APP Store on your mobile device and search for "Blynk IoT" (not Blynk(legacy)) to download.
2. After opening the APP, login in, this account should be the same as the account used on the web client.
3. Then go to **Dashboard** (if you don't have one, create one) and you will see that the **Dashboard** for mobile and web are independent of each other.



4. Click **Edit** Icon.
5. Click on the blank area.
6. Choose the same widget as on the web page, such as select a **Joystick** widget.



7. Now you will see a **Joystick** widget appear in the blank area, click on it.
8. **Joystick** Settings will appear, select the **Xvalue** and **Yvalue** datastreams you just set in the web page. Note that each widget corresponds to a different datastream in each project.
9. Go back to the **Dashboard** page and you can operate the **Joystick** when you want.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

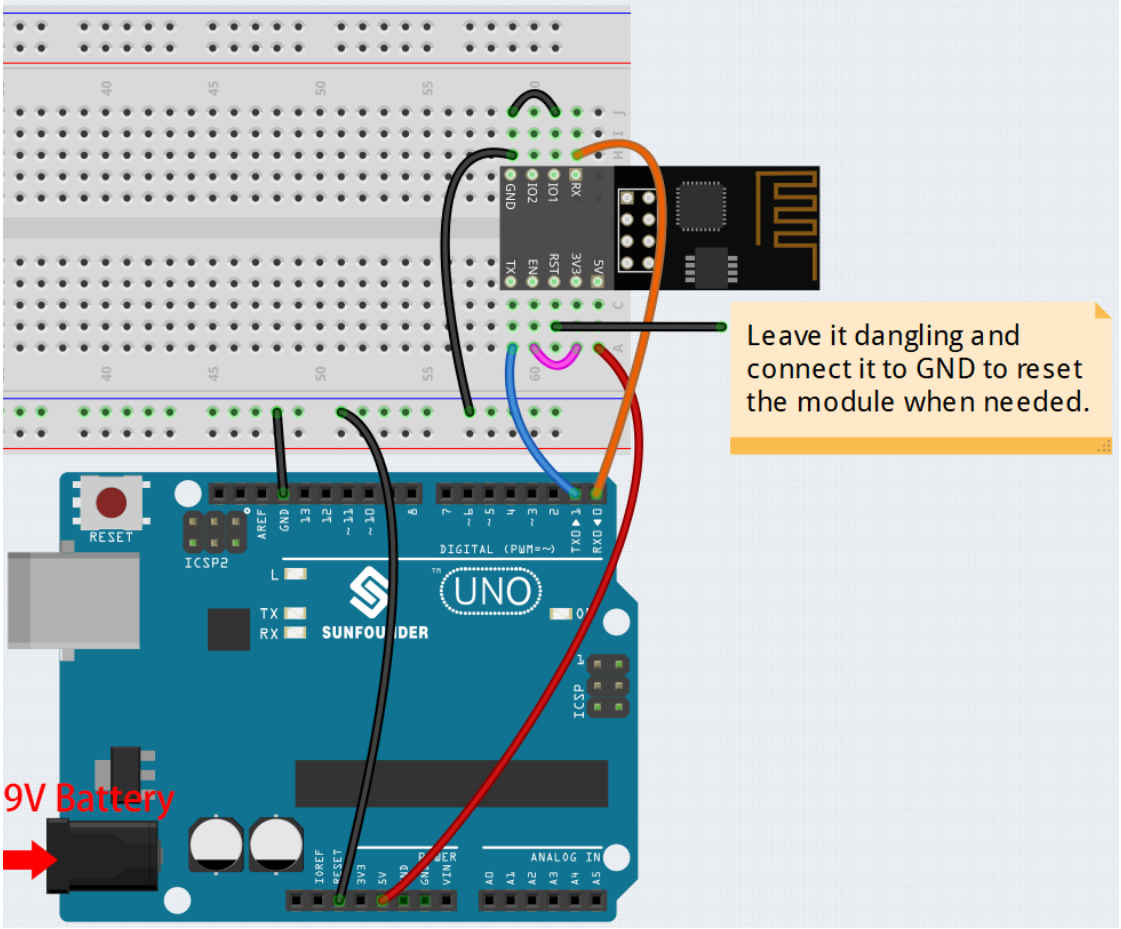
Ready to explore and create with us? Click [] and join today!

1.8.2 How to re-burn the AT firmware for ESP8266 module?

Re-burn the Firmware with R3

1. Build the circuit

Connect ESP8266 and SunFounder R3 board.



2. Burning the firmware

- Follow the steps below to burn the firmware if you are using **Windows**.

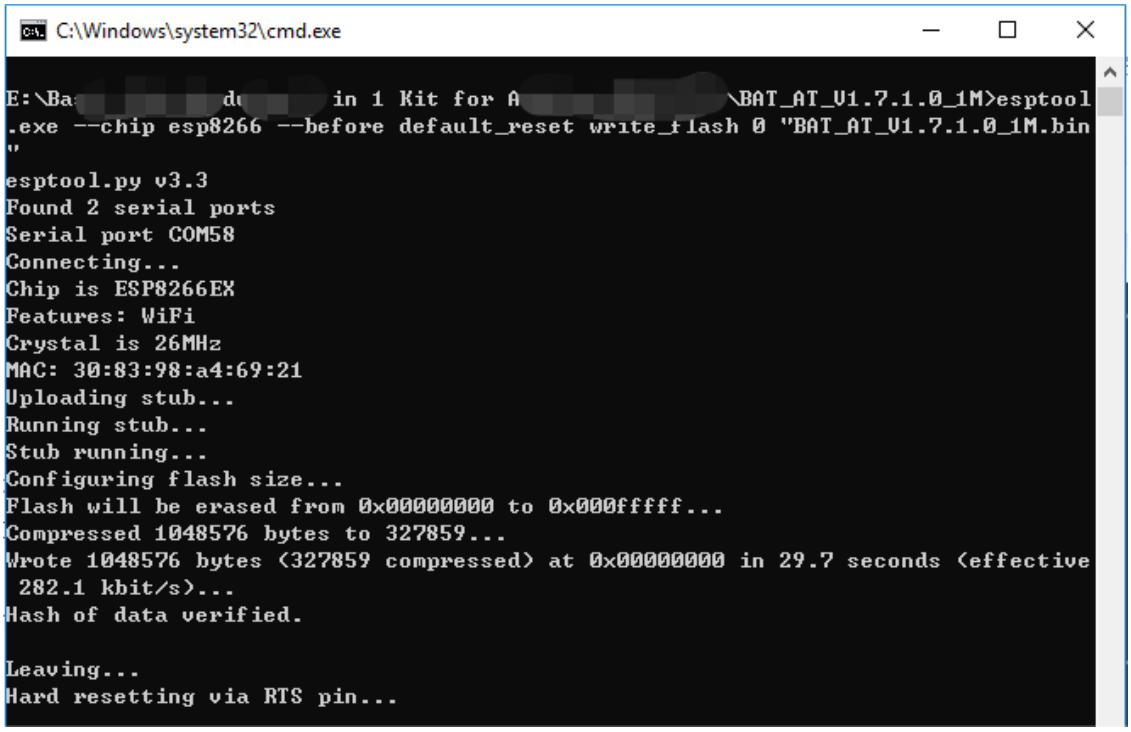
  1. Download firmware and burn-in tool.

     - `ESP8266 Firmware`

  2. After unzipping, you will see 4 files.

     - `BAT_AT_V1.7.1.0_1M.bin`: The firmware to burn to the ESP8266 module.
     - `esptool.exe`: This is a command-line utility for Windows.
     - `install_r3.bat`: This is the command package for Windows system, double click this file will run all the commands inside the file.
     - `install_r4.bat`: Same as `install_r3.bat`, but dedicated to UNO R4 board.

  3. Double click `install_r3.bat` to start the firmware burning. If you see the following prompt, the firmware has been installed successfully.



Note: If the burn-in fails, please check the following points.

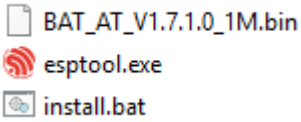
- Reset the ESP8266 module by inserting the RST on the ESP8266 Adapter to GND and then unplugging it.
- Check if the wiring is correct.
- Whether the computer has recognized your board properly, and make sure the port is not occupied.
- Reopen the install.bat file.

- To burn the firmware, follow these steps if you are using a **Mac OS** system.

1. Use the following commands to install Esptool. Esptool is a Python-based, open-source, platform-independent utility to communicate with the ROM bootloader in Espressif chips.

```
python3 -m pip install --upgrade pip
python3 -m pip install esptool
```

2. If esptool is properly installed, it will output a message such as [usage: esptool] if you run `python3 -m esptool`.
3. Download firmware.
   - `ESP8266 Firmware`
4. After unzipping, you will see 3 files.

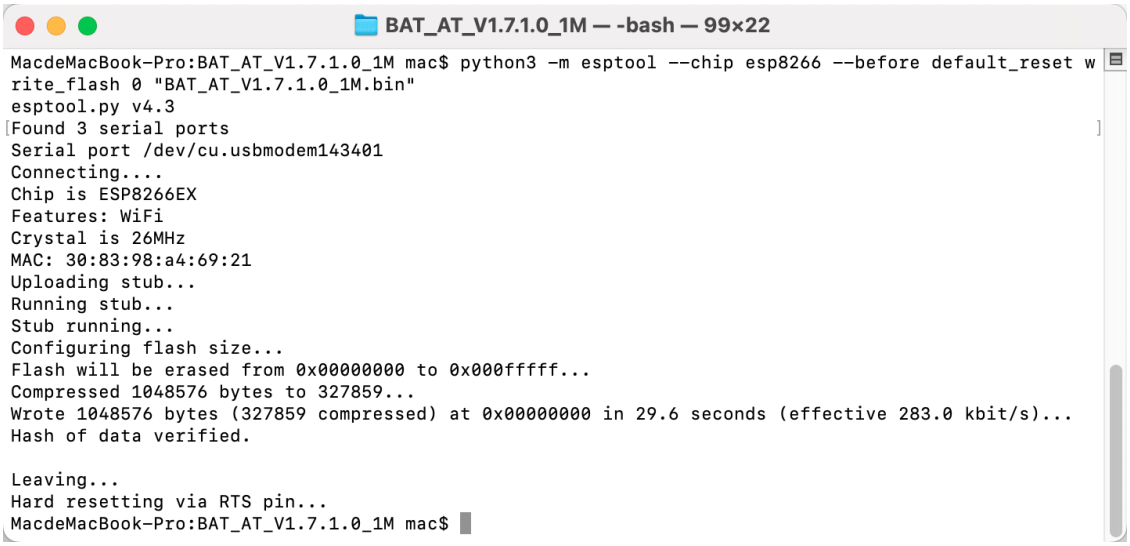


- `BAT_AT_V1.7.1.0_1M.bin`: The firmware to burn to the ESP8266 module.
- `esptool.exe`: This is a command-line utility for Windows.
- `install_r3.bat`: This is the command package for Windows system.
- `install_r4.bat`: Same as `install_r3.bat`, but dedicated to UNO R4 board.

5. Open a terminal and use the `cd` command to go into the firmware folder you just downloaded, then run the following command to erase the existing firmware and re-burn the new firmware.

```
python3 -m esptool --chip esp8266 --before default_reset erase_flash
python3 -m esptool --chip esp8266 --before default_reset write_flash 0 "BAT_AT_
↪V1.7.1.0_1M.bin"
```

6. If you see the following prompt, the firmware has been installed successfully.



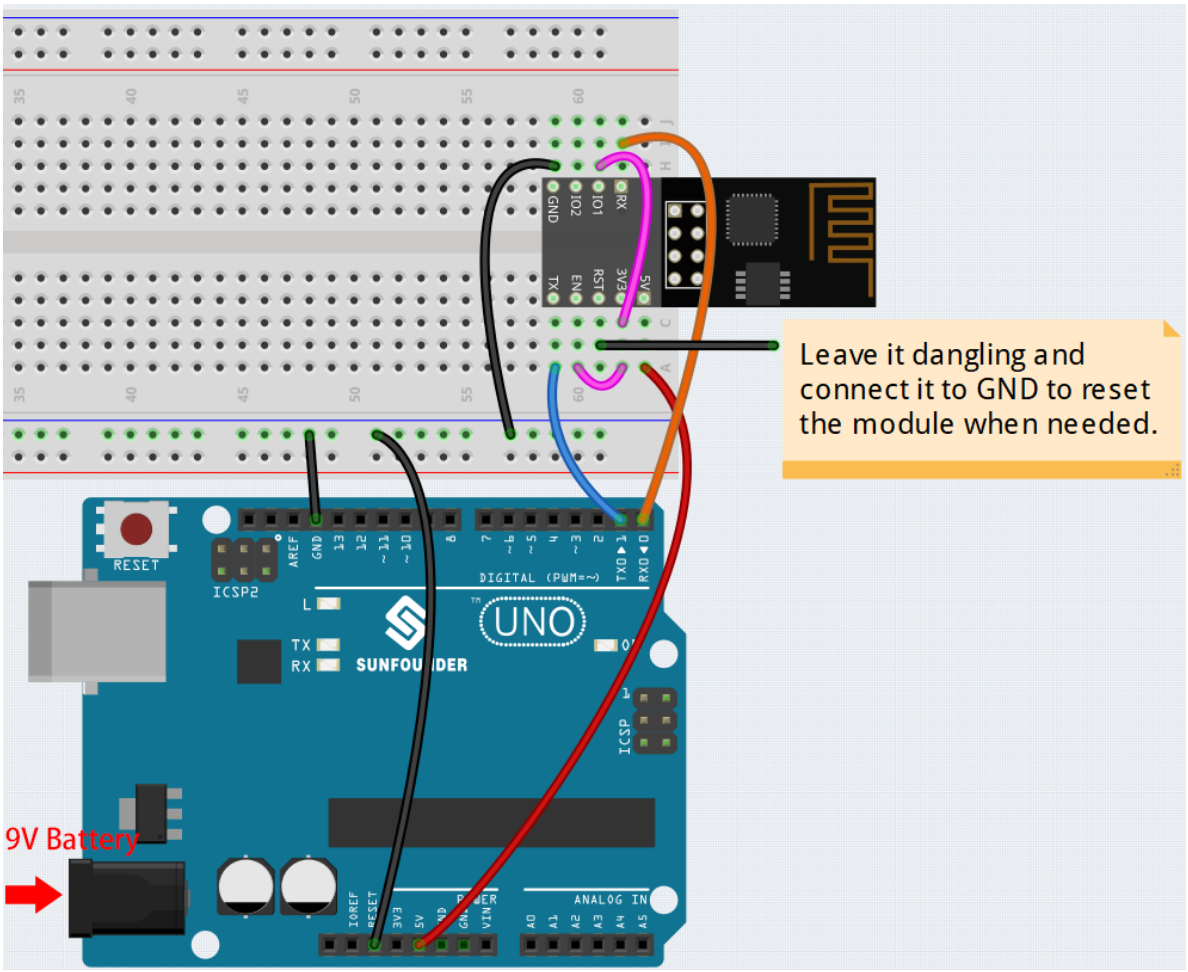
Note: If the burn-in fails, please check the following points.

- Reset the ESP8266 module by inserting the RST on the ESP8266 Adapter to GND and then unplugging it.
- Check if the wiring is correct.
- Whether the computer has recognized your board properly, and make sure the port is not occupied.
- Reopen the install.bat file.

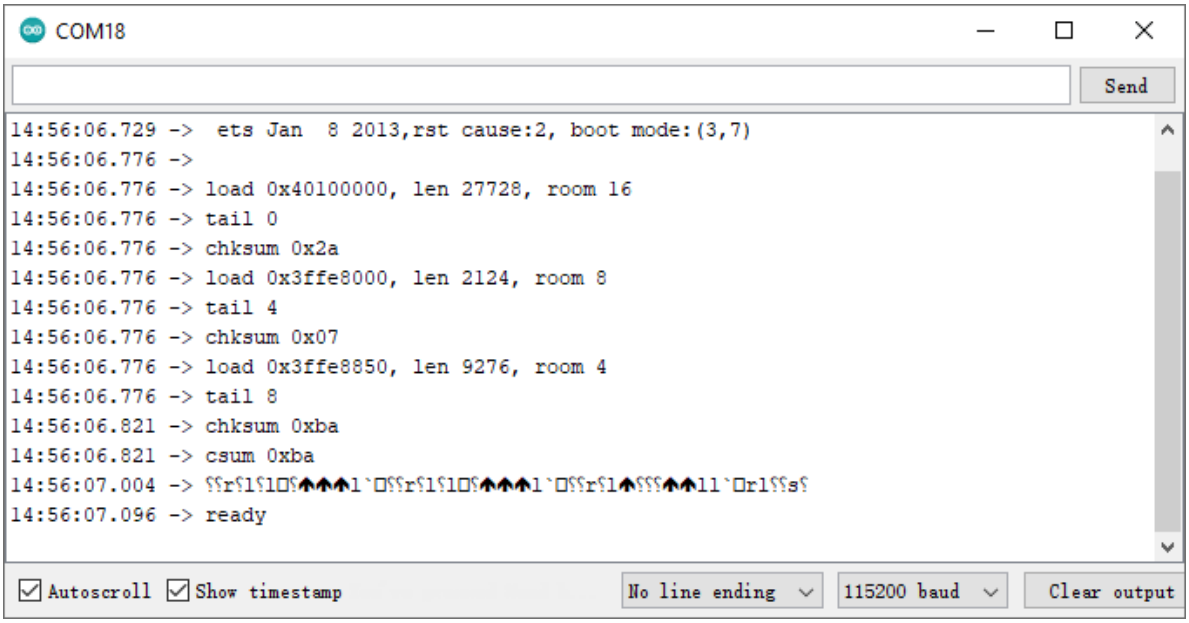
---

3. Test

1. On the basis of the original wiring, connect IO1 to 3V3.



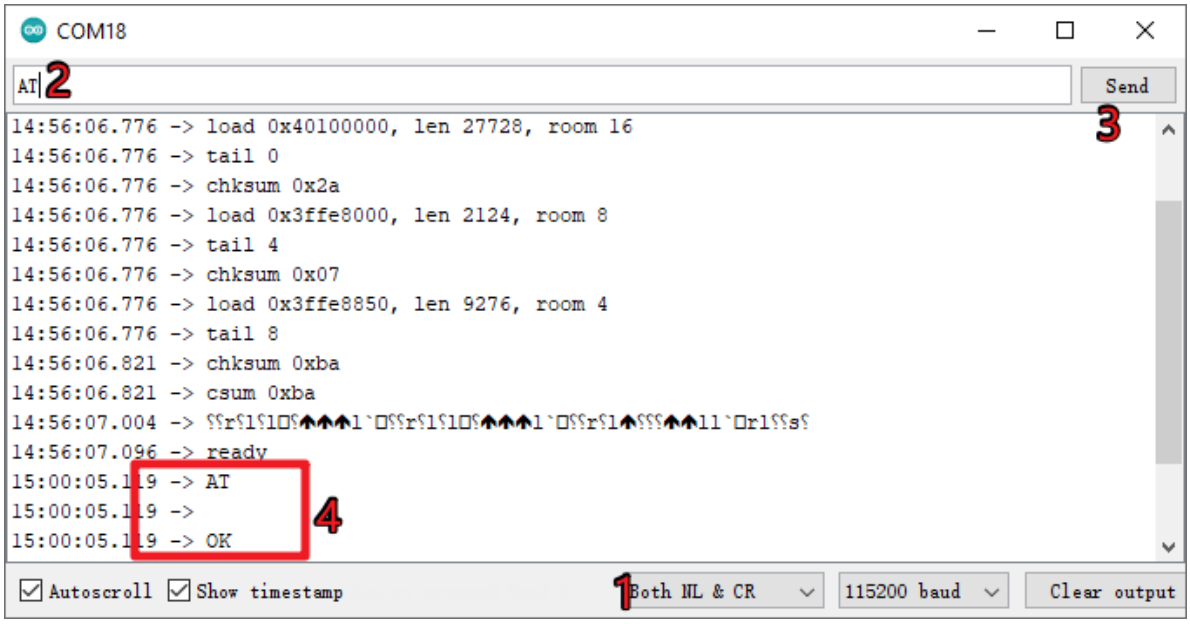
2. You will be able to see information about the ESP8266 module if you click the magnifying glass icon(Serial Monitor) in the upper right corner and set the baud rate to **115200**.



Note:

- If `ready` doesn't appear, you can try to reset the ESP8266 module(connect RST to GND) and re-open the Serial Monitor.

3. Click on **NEWLINE DROPDOWN BOX**, select `both NL & CR` in the drop down option, enter `AT`, if it returns OK, it means ESP8266 has successfully established connection with R3 board.

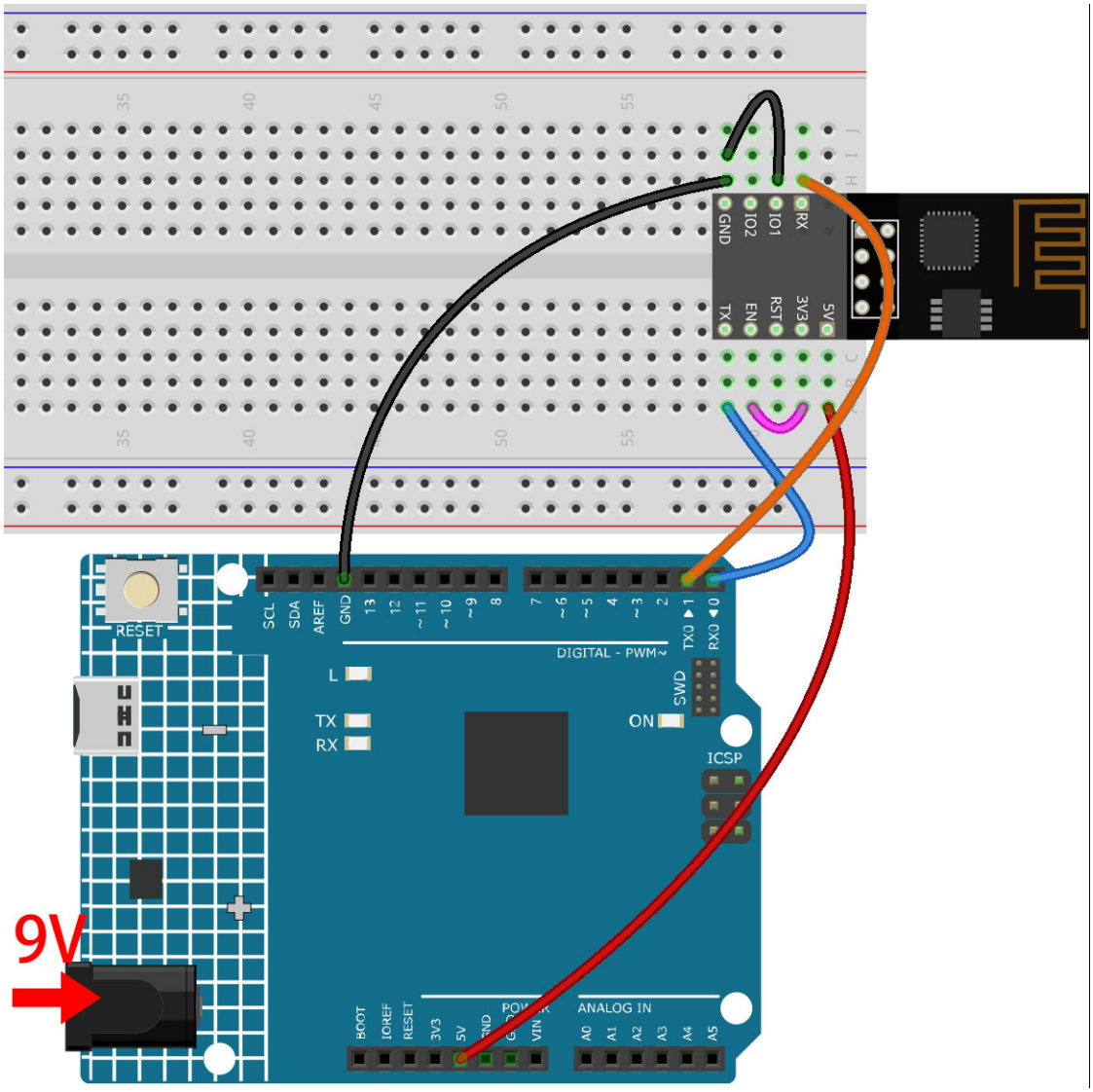


Now you can continue to follow *1.1 Configuring the ESP8266* to set the working mode and baud rate of the ESP8266 module.

Re-burn the Firmware with R4

1. Build the circuit

Connect ESP8266 and Arduino UNO R4 board.



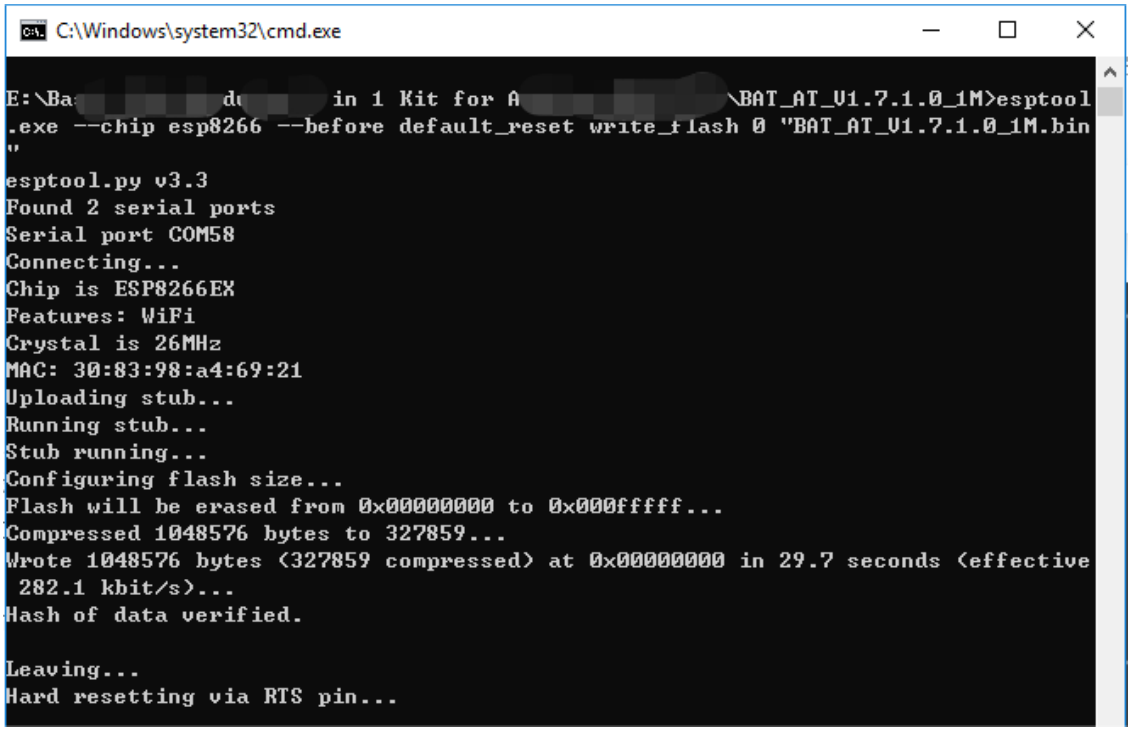
2. Upload the Following Code to R4

```
void setup() {
    Serial.begin(115200);
    Serial1.begin(115200);
}

void loop() {
    if (Serial.available()) {      // If anything comes in Serial (USB),
        Serial1.write(Serial.read());   // read it and send it out Serial1 (pins 0 & 1)
    }
        if (Serial1.available()) {     // If anything comes in Serial1 (pins 0 & 1)
        Serial.write(Serial1.read());   // read it and send it out Serial (USB)
    }
}
```

3. Burning the firmware

- Follow the steps below to burn the firmware if you are using **Windows**.

1. Download firmware and burn-in tool.
   - `ESP8266 Firmware`
2. After unzipping, you will see 4 files.
   - `BAT_AT_V1.7.1.0_1M.bin`: The firmware to burn to the ESP8266 module.
   - `esptool.exe`: This is a command-line utility for Windows.
   - `install_r3.bat`: This is the command package for Windows system, double click this file will run all the commands inside the file.
   - `install_r4.bat`: Same as `install_r3.bat`, but dedicated to UNO R4 board.
3. Double click `install_r4.bat` to start the firmware burning. If you see the following prompt, the firmware has been installed successfully.



Note: If the burn-in fails, please check the following points.

- Reset the ESP8266 module by inserting the RST on the ESP8266 Adapter to GND and then unplugging it.
- Check if the wiring is correct.
- Whether the computer has recognized your board properly, and make sure the port is not occupied.
- Reopen the install.bat file.

- To burn the firmware, follow these steps if you are using a **Mac OS** system.
  1. Use the following commands to install Esptool. Esptool is a Python-based, open-source, platform-independent utility to communicate with the ROM bootloader in Espressif chips.

```
python3 -m pip install --upgrade pip
python3 -m pip install esptool
```

2. If esptool is properly installed, it will output a message such as [usage: esptool] if you run `python3 -m esptool`.
3. Download firmware.
   - `ESP8266 Firmware`
4. After unzipping, you will see 4 files.
   - `BAT_AT_V1.7.1.0_1M.bin`: The firmware to burn to the ESP8266 module.
   - `esptool.exe`: This is a command-line utility for Windows.
   - `install_r3.bat`: This is the command package for Windows system.
   - `install_r4.bat`: Same as `install_r3.bat`, but dedicated to UNO R4 board.
5. Open a terminal and use the `cd` command to go into the firmware folder you just downloaded, then run the following command to erase the existing firmware and re-burn the new firmware.

```
python3 -m esptool --chip esp8266 --before no_reset_no_sync erase_flash
python3 -m esptool --chip esp8266 --before no_reset_no_sync write_flash 0 "BAT_
↪AT_V1.7.1.0_1M.bin"
```

6. If you see the following prompt, the firmware has been installed successfully.

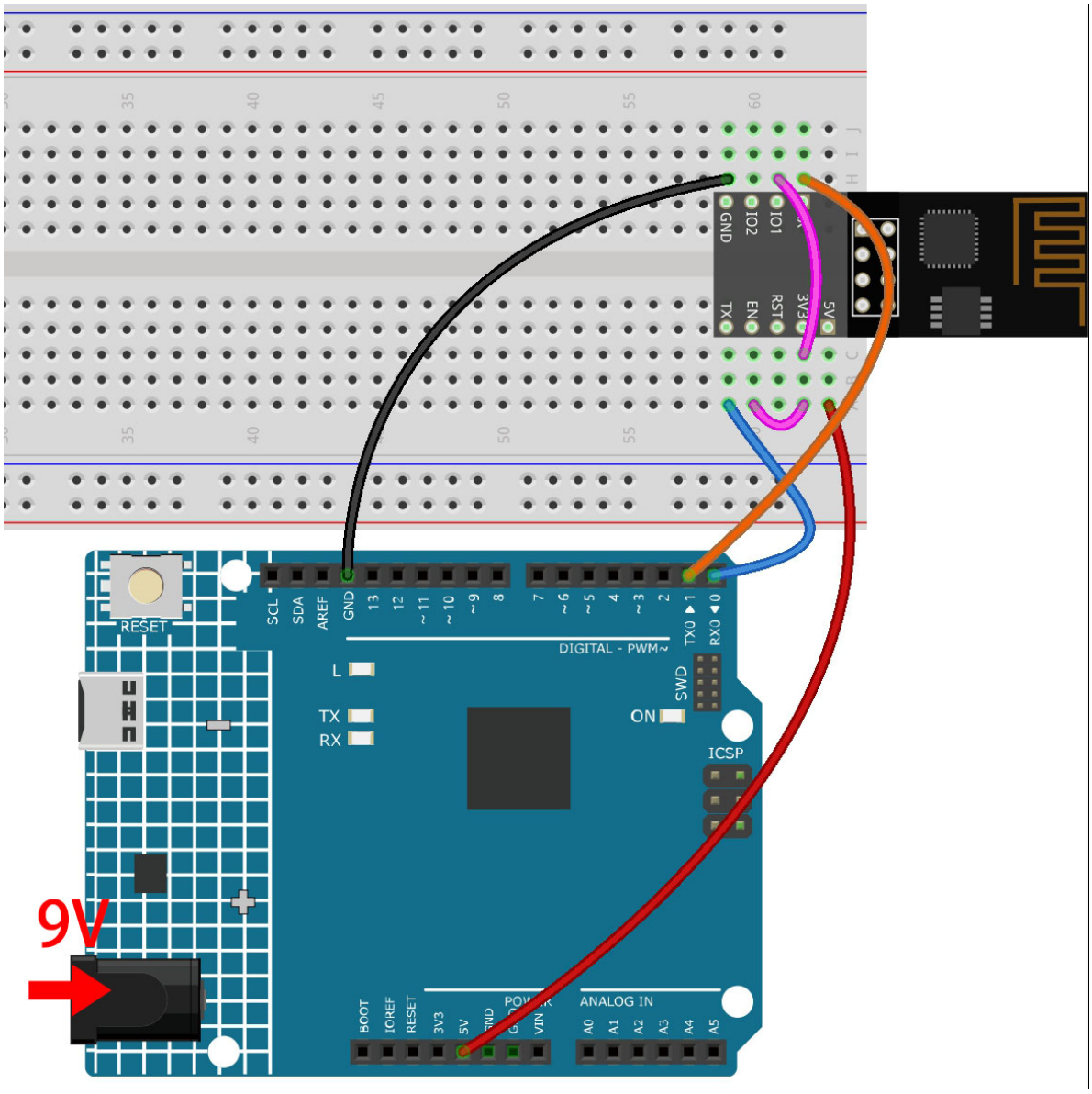
Note: If the burn-in fails, please check the following points.

- Reset the ESP8266 module by inserting the RST on the ESP8266 Adapter to GND and then unplugging it.
- Check if the wiring is correct.
- Whether the computer has recognized your board properly, and make sure the port is not occupied.
- Reopen the install.bat file.

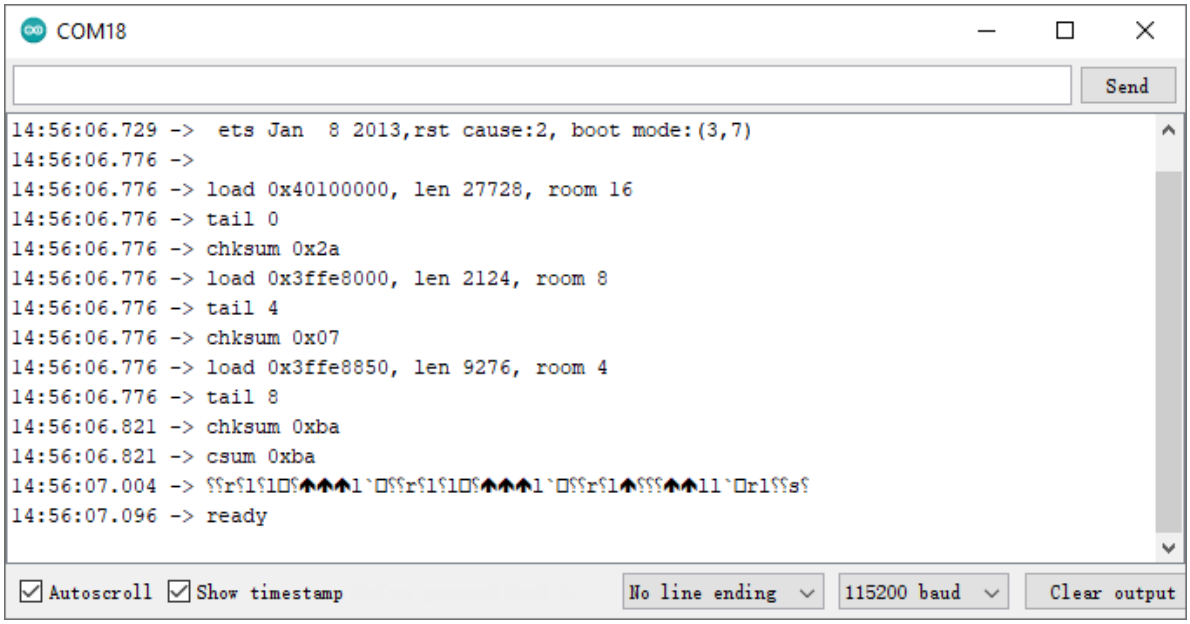
---

4. Test

1. On the basis of the original wiring, connect IO1 to 3V3.



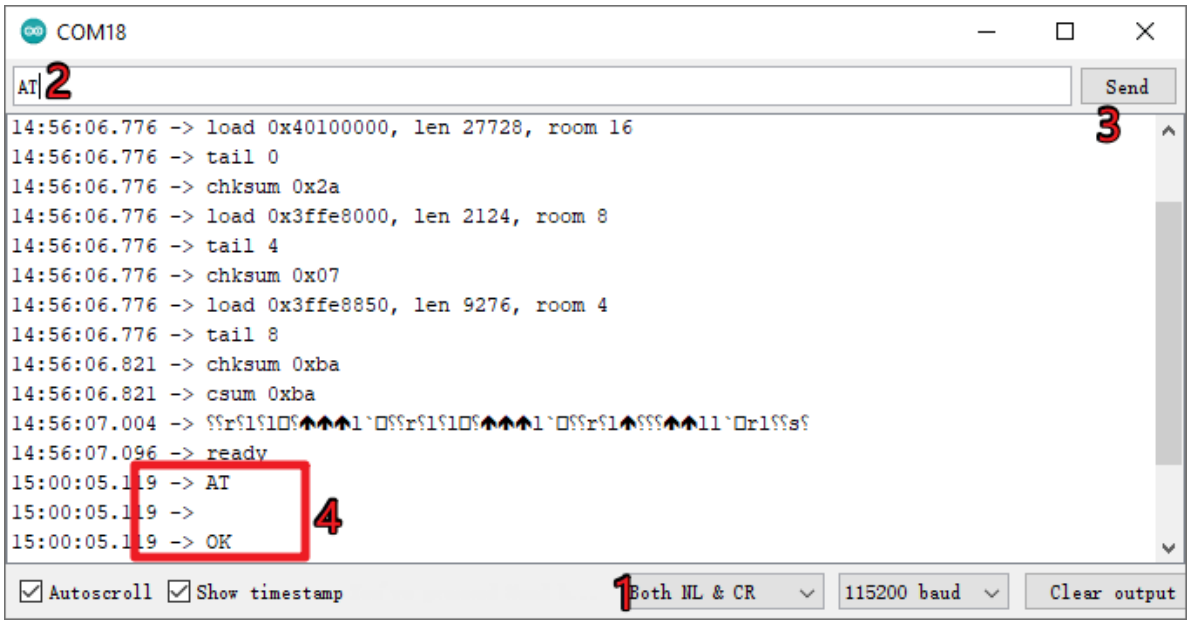
2. You will be able to see information about the ESP8266 module if you click the magnifying glass icon(Serial Monitor) in the upper right corner and set the baud rate to **115200**.



Note:

- If `ready` doesn't appear, you can try to reset the ESP8266 module(connect RST to GND) and re-open the Serial Monitor.

3. Click on **NEWLINE DROPDOWN BOX**, select `both NL & CR` in the drop down option, enter `AT`, if it returns OK, it means ESP8266 has successfully established connection with R4 board.



Now you can continue to follow *Lesson 35: Get Started with ESP8266 Module* to set the working mode and baud rate of the ESP8266 module.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.8.3 Get the IP address

There are many ways to know the IP address, and two of them are listed as follows.

Checking via the router

If you have permission to log in the router(such as a home network), you can check the addresses assigned to Raspberry Pi on the admin interface of router.

The default hostname of the Raspberry Pi OS is raspberrypi, and you need to find it. (If you are using ArchLinuxARM system, please find alarmpi.)

Network Segment Scanning

You can also use network scanning to look up the IP address of Raspberry Pi. You can apply the software, **Advanced IP scanner** and so on.

Scan the IP range set, and the name of all connected devices will be displayed. Similarly, the default hostname of the Raspberry Pi OS is raspberrypi, if you haven't modified it.

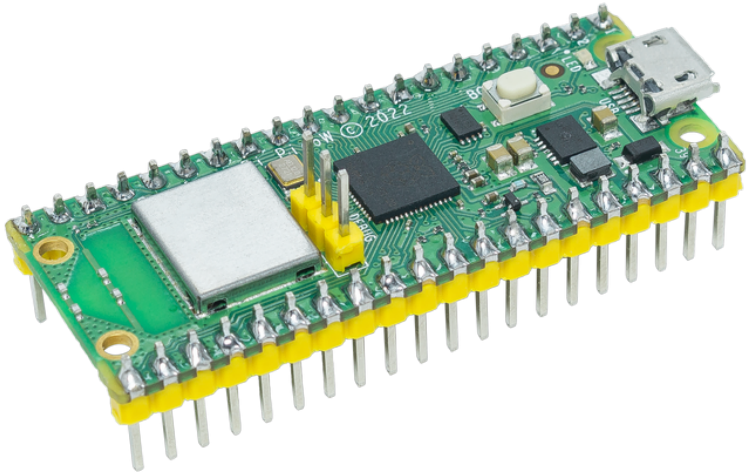
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.8.4 Raspberry Pi Pico W



Raspberry Pi Pico W brings wireless connectivity to the best-selling Raspberry Pi Pico product line. Built around our RP2040 silicon platform, Pico products bring our signature values of high performance, low cost, and ease of use to the microcontroller space.

Raspberry Pi Pico W offers 2.4GHz 802.11 b/g/n wireless LAN support, with an on-board antenna, and modular compliance certification. It is able to operate in both station and access-point modes. Full access to network functionality is available to both C and MicroPython developers.

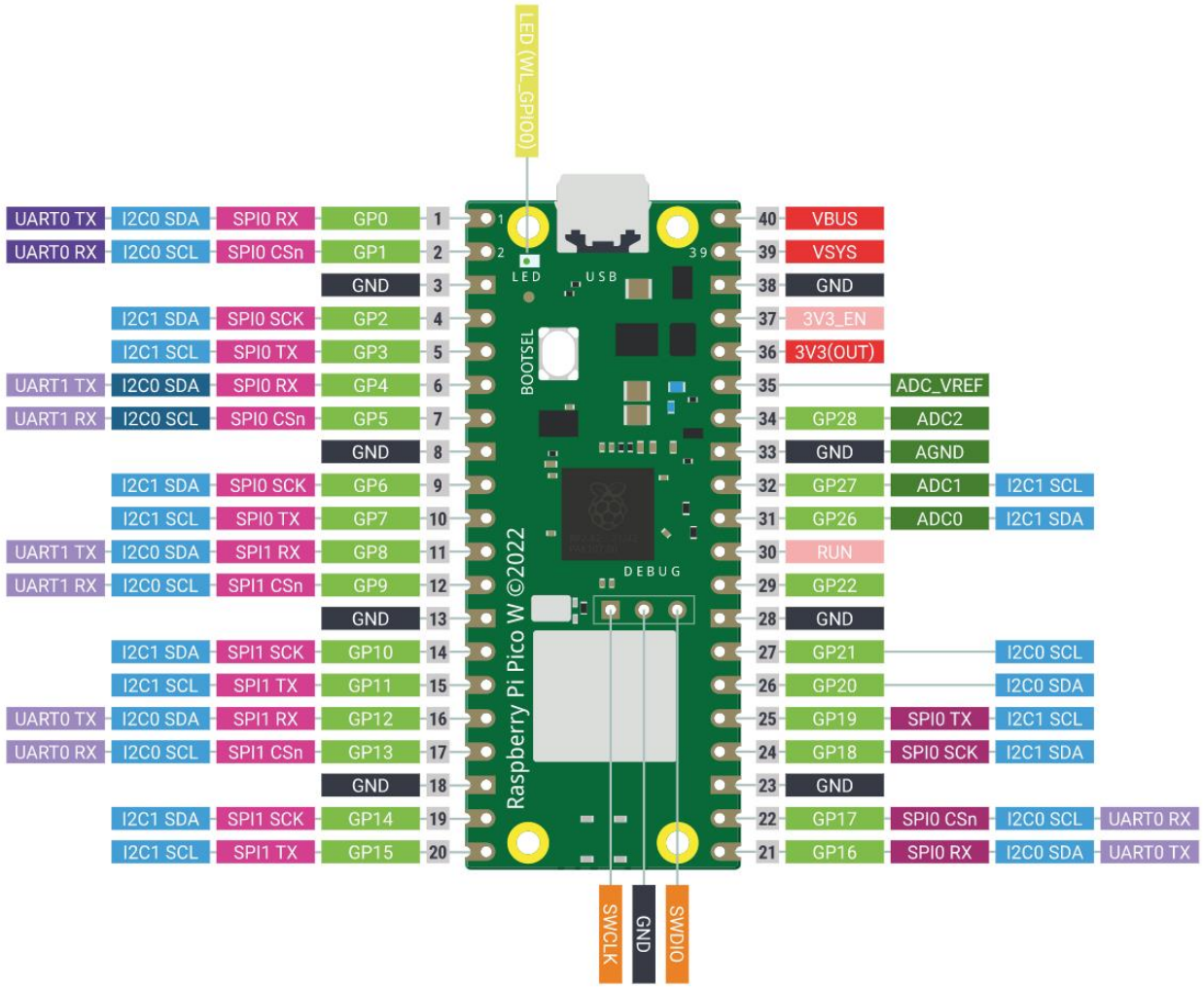
Raspberry Pi Pico W pairs RP2040 with 2MB of flash memory, and a power supply chip supporting input voltages from 1.8–5.5V. It provides 26 GPIO pins, three of which can function as analogue inputs, on 0.1”-pitch through-hole pads with castellated edges. Raspberry Pi Pico W is available as an individual unit, or in 480-unit reels for automated assembly.

Features

- 21 mm x 51 mm form factor
- RP2040 microcontroller chip designed by Raspberry Pi in the UK
- Dual-core Arm Cortex-M0+ processor, flexible clock running up to 133 MHz
- 264kB on-chip SRAM
- 2MB on-board QSPI flash
- 2.4GHz 802.11n wireless LAN
- 26 multifunction GPIO pins, including 3 analogue inputs
- 2 x UART, 2 x SPI controllers, 2 x I2C controllers, 16 x PWM channels
- 1 x USB 1.1 controller and PHY, with host and device support
- 8 x Programmable I/O (PIO) state machines for custom peripheral support
- Supported input power 1.8-5.5V DC
- Operating temperature -20°C to +70°C
- Castellated module allows soldering direct to carrier boards
- Drag-and-drop programming using mass storage over USB

- Low-power sleep and dormant modes
- Accurate on-chip clock
- Temperature sensor
- Accelerated integer and floating-point libraries on-chip

Pico's Pins



Name	Description	Function
GP0-GP28	General-purpose input/output pins	Act as either input or output and have no fixed purpose of their own
GND	0 volts ground	Several GND pins around Pico W to make wiring easier.
RUN	Enables or disables your Pico	Start and stop your Pico W from another microcontroller.
GPxx_ADCx	General-purpose input/output or analog input	Used as an analog input as well as a digital input or output – but not both at the same time.
ADC_VREF	Analog-to-digital converter (ADC) voltage reference	A special input pin which sets a reference voltage for any analog inputs.
AGND	Analog-to-digital converter (ADC) 0 volts ground	A special ground connection for use with the ADC_VREF pin.
3V3(O)	3.3 volts power	A source of 3.3V power, the same voltage your Pico W runs at internally, generated from the VSYS input.
3v3(E)	Enables or disables the power	Switch on or off the 3V3(O) power, can also switches your Pico W off.
VSYS	2-5 volts power	A pin directly connected to your Pico's internal power supply, which cannot be switched off without also switching Pico W off.
VBUS	5 volts power	A source of 5 V power taken from your Pico's micro USB port, and used to power hardware which needs more than 3.3 V.

The best place to find everything you need to get started with your Raspberry Pi Pico W is here.

Or you can click on the links below:

- Raspberry Pi Pico W product brief
- Raspberry Pi Pico W datasheet
- Getting started with Raspberry Pi Pico: C/C++ development
- Raspberry Pi Pico C/C++ SDK
- API-level Doxygen documentation for the Raspberry Pi Pico C/C++ SDK
- Raspberry Pi Pico Python SDK
- Raspberry Pi RP2040 datasheet
- Hardware design with RP2040
- Raspberry Pi Pico W design files
- Raspberry Pi Pico W STEP file

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

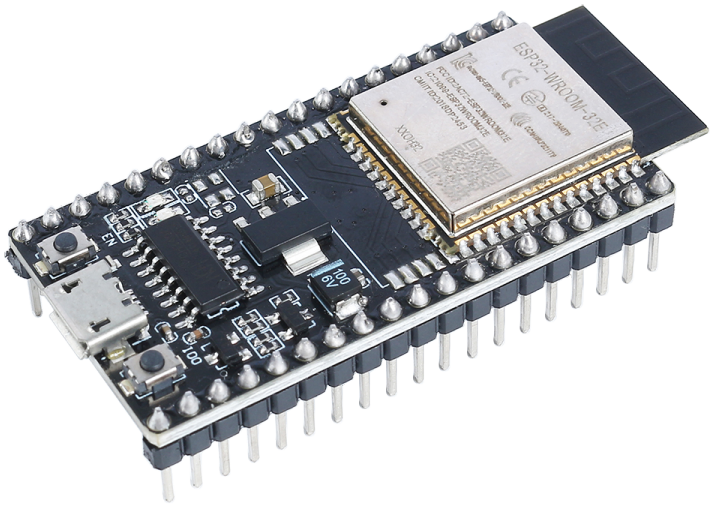
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.8.5 ESP32 WROOM 32E

The ESP32 WROOM-32E is a versatile and powerful module built around Espressif's ESP32 chipset. It offers dual-core processing, integrated Wi-Fi and Bluetooth connectivity, and boasts a wide range of peripheral interfaces. Known for its low-power consumption, the module is ideal for IoT applications, enabling smart connectivity and robust performance in compact form factors.



Key features include:

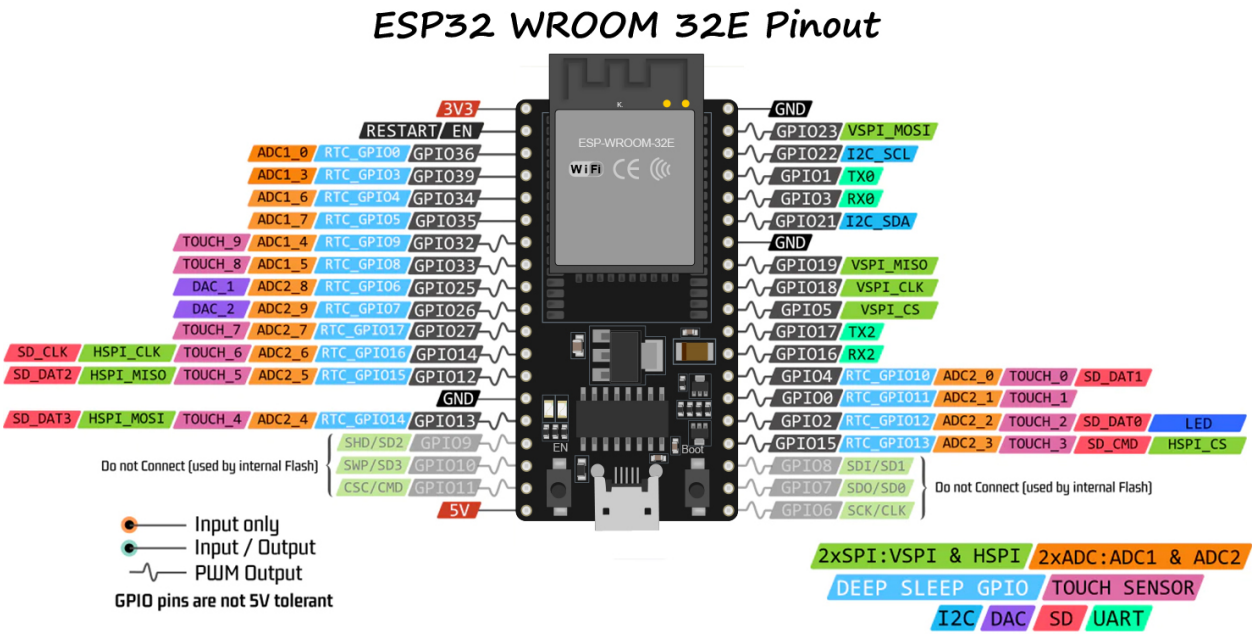
- **Processing Power**: It's equipped with a dual-core Xtensa® 32-bit LX6 microprocessor, offering scalability and flexibility.
- **Wireless Capabilities**: With integrated 2.4 GHz Wi-Fi and dual-mode Bluetooth, it's perfectly suited for applications demanding stable wireless communication.
- **Memory & Storage**: It comes with ample SRAM and high-performance flash storage, catering to user programs and data storage needs.
- **GPIO**: Offering up to 38 GPIO pins, it supports a variety of external devices and sensors.
- **Low Power Consumption**: Multiple power-saving modes are available, making it ideal for battery-powered or energy-efficient scenarios.
- **Security**: Integrated encryption and security features ensure user data and privacy are well-protected.
- **Versatility**: From simple household appliances to complex industrial machinery, the WROOM-32E delivers consistent, efficient performance.

In summary, the ESP32 WROOM-32E not only offers robust processing capabilities and diverse connectivity options but also boasts an array of features making it a preferred choice in the IoT and smart device sectors.

-

Pinout Diagram

The ESP32 has some pin usage limitations due to various functionalities sharing certain pins. When designing a project, it's a good practice to carefully plan the pin usage and cross-check for potential conflicts to ensure proper functioning and avoid issues.



Here are some of the key restrictions and considerations:

- **ADC1 and ADC2**: ADC2 cannot be used when WiFi or Bluetooth is active. However, ADC1 can be used without any restrictions.
- **Bootstrapping Pins**: GPIO0, GPIO2, GPIO5, GPIO12, and GPIO15 are used for bootstrapping during the boot process. Care should be taken not to connect external components that could interfere with the boot process on these pins.
- **JTAG Pins**: GPIO12, GPIO13, GPIO14, and GPIO15 can be used as JTAG pins for debugging purposes. If JTAG debugging is not required, these pins can be used as regular GPIOs.
- **Touch Pins**: Some pins support touch functionalities. These pins should be used carefully if you intend to use them for touch sensing.
- **Power Pins**: Some pins are reserved for power-related functions and should be used accordingly. For example, avoid drawing excessive current from power supply pins like 3V3 and GND.
- **Input-only Pins**: Some pins are input-only and should not be used as outputs.

Strapping Pins

ESP32 has five strapping pins:

Strapping Pins	Description
IO5	Defaults to pull-up, the voltage level of IO5 and IO15 affects the Timing of SDIO Slave.
IO0	Defaults to pull-up, if pulled low, it enters download mode.
IO2	Defaults to pull-down, IO0 and IO2 will make ESP32 enter download mode.
IO12(MTDI)	Defaults to pull-down, if pulled high, ESP32 will fail to boot up normally.
IO15(MTDO)	Defaults to pull-up, if pulled low, debug log will not be visible. Additionally, the voltage level of IO5 and IO15 affects the Timing of SDIO Slave.

Software can read the values of these five bits from register "GPIO_STRAPPING". During the chip's system reset release (power-on-reset, RTC watchdog reset and brownout reset), the latches of the strapping pins sample the voltage level as strapping bits of "0" or "1", and hold these bits until the chip is powered down or shut down. The strapping bits configure the device's boot mode, the operating voltage of VDD_SDIO and other initial system settings.

Each strapping pin is connected to its internal pull-up/pull-down during the chip reset. Consequently, if a strapping pin is unconnected or the connected external circuit is high-impedance, the internal weak pull-up/pull-down will determine the default input level of the strapping pins.

To change the strapping bit values, users can apply the external pull-down/pull-up resistances, or use the host MCU's GPIOs to control the voltage level of these pins when powering on ESP32.

After reset release, the strapping pins work as normal-function pins. Refer to following table for a detailed boot-mode configuration by strapping pins.

Voltage of Internal LDO (VDD_SDIO)			
Pin	Default	3.3 V	1.8 V
MTDI	Pull-down	0	1
Booting Mode			
Pin	Default	SPI Boot	Download Boot
GPIO0	Pull-up	1	0
GPIO2	Pull-down	Don't-care	0
Enabling/Disabling Debugging Log Print over U0TXD During Booting			
Pin	Default	U0TXD Active	U0TXD Silent
MTDO	Pull-up	1	0
Timing of SDIO Slave			

Pin	Default	FE Sampling FE Output	FE Sampling RE Output	RE Sampling FE Output	RE Sampling RE Output
MTDO	Pull-up	0	0	1	1
GPIO5	Pull-up	0	1	0	1

- FE: falling-edge, RE: rising-edge
- Firmware can configure register bits to change the settings of "Voltage of Internal LDO (VDD_SDIO)" and "Timing of SDIO Slave", after booting.
- The module integrates a 3.3 V SPI flash, so the pin MTDI cannot be set to 1 when the module is powered up.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

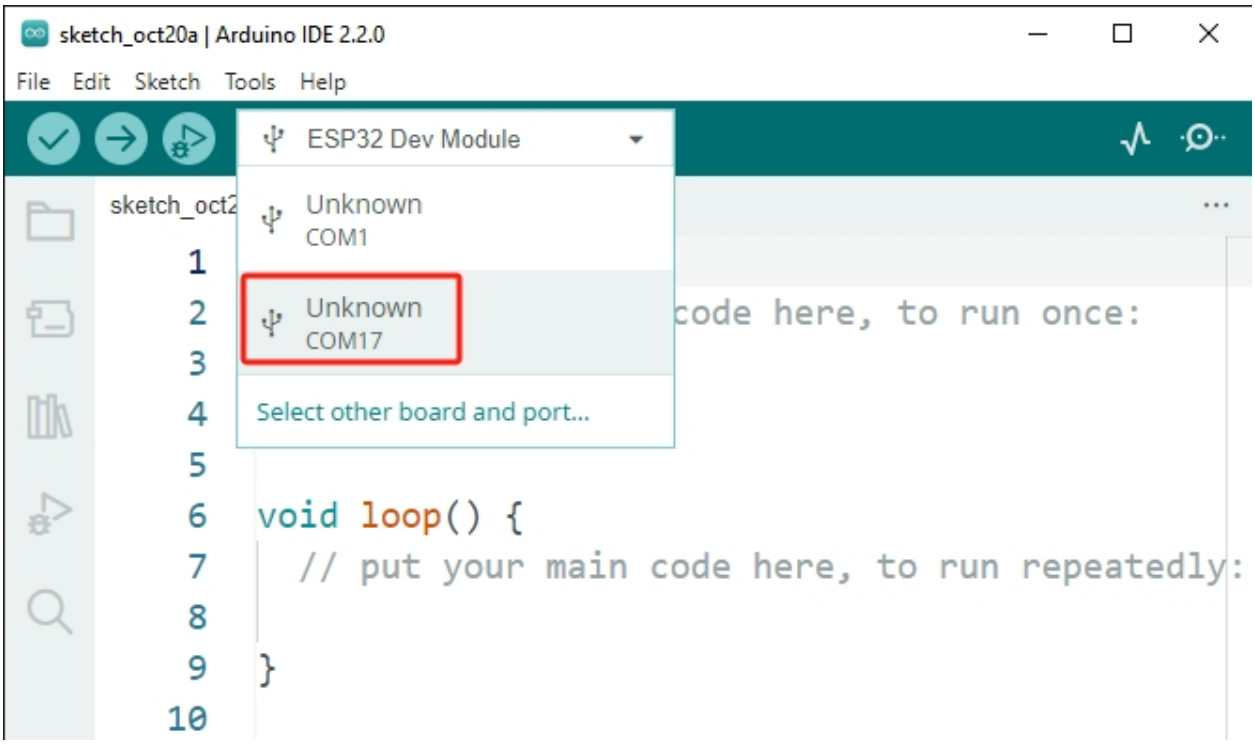
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.8.6 Always displaying "Unknown COMxx"?

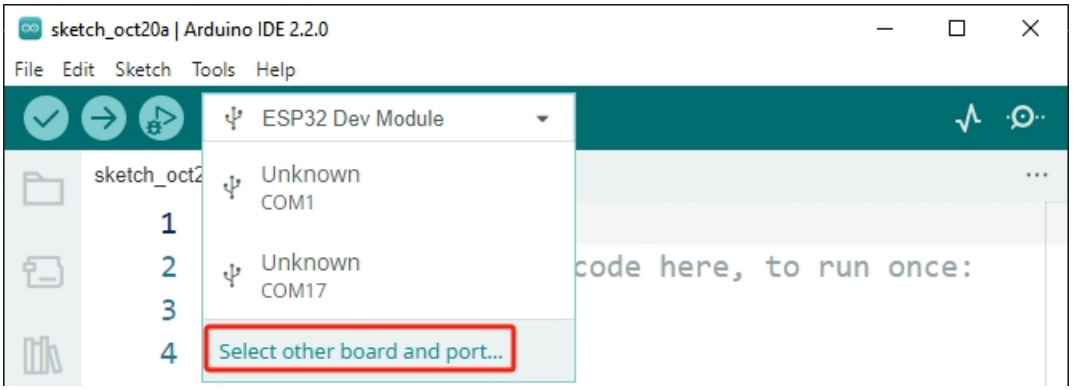
When plugging the ESP32 into the computer, the Arduino IDE often displays `Unknown COMxx`. Why does this happen?



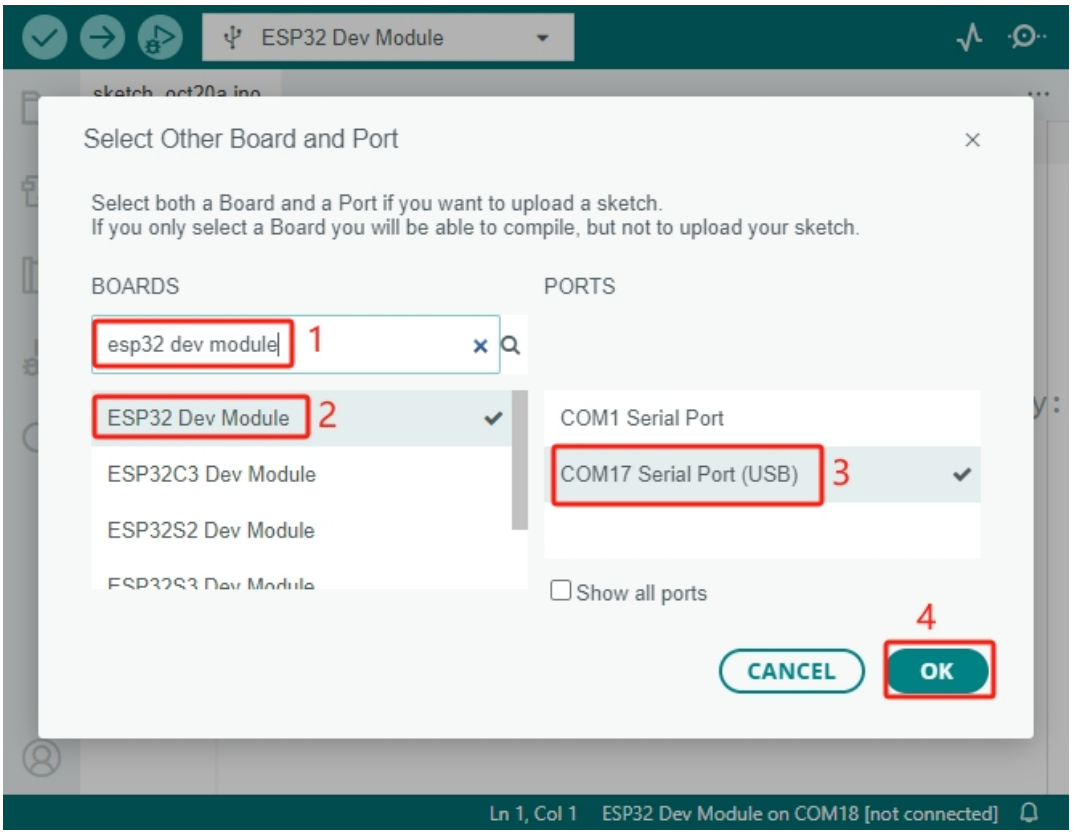
This is because the USB driver for ESP32 is different from the regular Arduino Boards. The Arduino IDE can't automatically recognize this board.

In such a scenario, you need to manually select the correct board by following these steps:

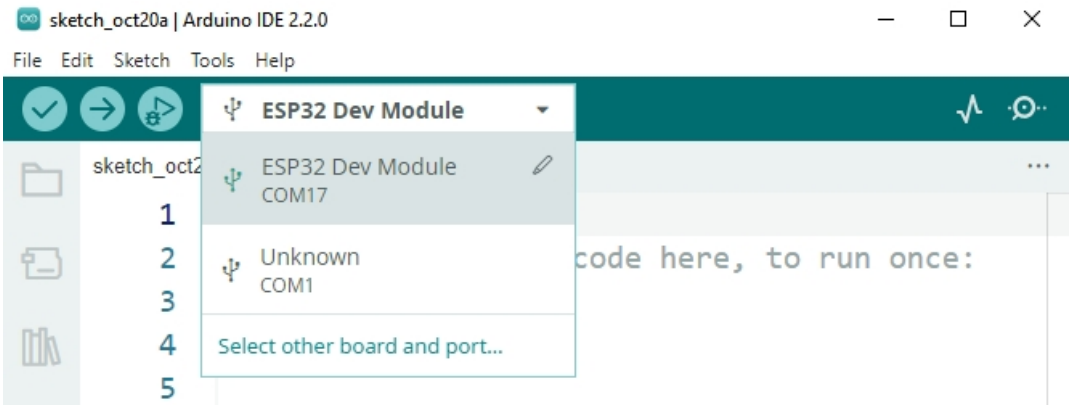
1. Click on **"Select the other board and port"**.



2. In the search, type **"esp32 dev module"**, then select the board that appears. Afterward, select the correct port and click **OK**.



3. Now, you should be able to see your board and port in this quick view window.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

1.9 Thank You

Thanks to the evaluators who evaluated our products, the veterans who provided suggestions for the tutorial, and the users who have been following and supporting us. Your valuable suggestions to us are our motivation to provide better products!

If you have any questions or other interesting ideas, feel free to send an email to service@sunfounder.com.

CHAPTER

TWO

COPYRIGHT NOTICE