



# Keyestudio Raspberry Pi Pico 24 in 1 Sensor Kit

## Contents

1. Introduction .....	3
2. Kit List .....	4
Keyestudio 8002b Audio Power Amplifier .....	4
3. Raspberry Pi Pico and Thonny .....	8
3.1. Raspberry Pi Pico .....	8
3.2. MicroPython IDE-----Thonny .....	15
Download and Burn Firmware .....	16
Download and Install Thonny IDE .....	17
3.3 Install Drivers .....	23
3.4 Thonny User Interface .....	26
3.5 Add Modules .....	36
4. Projects .....	42
Project 1: Lighting up LED .....	42
Project 2: Traffic Light Module .....	48
Project 3: Button Sensor .....	52
Project 4: Obstacle Avoidance Sensor .....	58
Project 5: Tilt Module .....	63
Project 6: Reed Switch Module .....	68
Project 7: PIR Motion Sensor .....	73



Project 8: Active Buzzer .....	78
Project 9: 8002b Audio Power Amplifier .....	82
Project 10: RGB Module .....	88
Project 11: Potentiometer .....	99
Project 12: Sound Sensor .....	104
Project 13: Photoresistor .....	109
Project 14: NTC-MF52AT Thermistor .....	114
Project 15 Thin-film Pressure Sensor .....	119
Project 16: Joystick Module .....	124
Project 17: SK6812 RGB Module .....	129
Project 18: Rotary Encoder .....	137
Project 19: Servo Control .....	144
Project 20: Ultrasonic Sensor .....	152
Project 21: IR Receiver Module .....	159
Project 22: DS1307 Clock Module .....	167
Project 23: TM1650 4-Digit Tube Display .....	175
Project 24: HT16K33_8X8 Dot Matrix Module .....	188
5. Comprehensive Experiments .....	196
Project 25: Breathing LED .....	196
Project 26: Button-controlled LED .....	201
Project 27: Alarm Experiment .....	205
Project 28: PIR Motion Sensor .....	209



Project 29: Speaker Module .....	213
Project 30: Rotary Encoder .....	220
Project 31: Rotary Potentiometer .....	224
Project 32: Sound Activated Light.....	227
Project 33: RGB Module .....	230
Project 34: Ultrasonic Sensor .....	235
Project 35: IR Remote Control .....	244
Project 36: Comprehensive Experiment.....	249
6. Resources:.....	259

## 1. Introduction

The Keyestudio Raspberry Pi Pico 24 in 1 sensor kit mainly contains 24 commonly used sensors/modules, the Raspberry Pi Pico board, the Raspberry Pi Pico expansion board and Dupont wires.

The 24 sensors and modules are fully compatible with the Raspberry Pi Pico shield. You only need to stack the Raspberry Pi Pico board onto the Raspberry Pi Pico shield, and hook up them with Dupont wires, which is simple and convenient.

To make you master the electronic knowledge, detailed tutorials (MicroPython), schematic diagrams, wiring methods and test code are included. Through these projects, you will have a better understanding



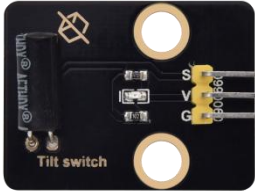
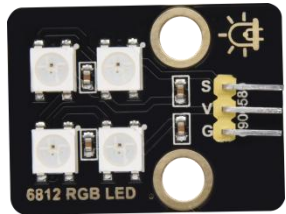
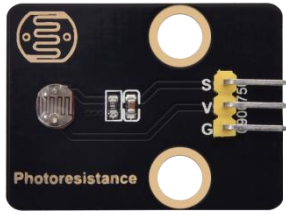


about programming, logic and electronics.

## 2. Kit List

No.	Picture	Name	QTY
1		Keyestudio White LED Module	1
2		Keyestudio Common Cathode RGB Module	1
3		Keyestudio Traffic Lights Module	1
4		Keyestudio Active Buzzer	1
5		Keyestudio 8002b Audio Power Amplifier	1
6		Keyestudio Button Module	1






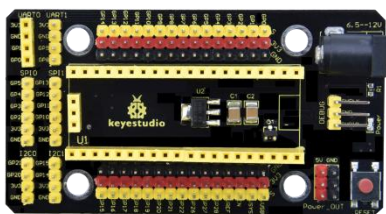
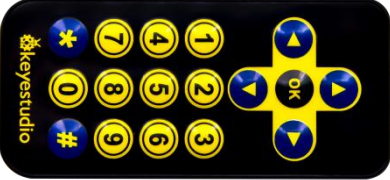

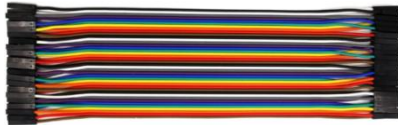


7		Keyestudio Tilt Sensor	1
8		Keyestudio PIR Motion Sensor	1
9		Keyestudio Obstacle Avoidance Sensor	1
10		Keyestudio 6812 RGB Module	1
11		Keyestudio NTC-MF52AT Analog Thermistor	1
12		Keyestudio Photoresistor	1
13		Keyestudio Sound Sensor	1
14		Keyestudio Rotary Potentiometer	1



15		Keyestudio IR Receiver	1
16		Keyestudio Reed Switch Sensor	1
17		Keyestudio Rotary Encoder Module	1
18		Keyestudio Joystick Module	1
19		Keyestudio HT16K33 8X8 Dot Matrix Module	1
20		Keyestudio TM1650 4-Digit Tube Display	1
21		Keyestudio Thin-film Pressure Sensor	1
22		Keyestudio DS1307 Clock Sensor	1

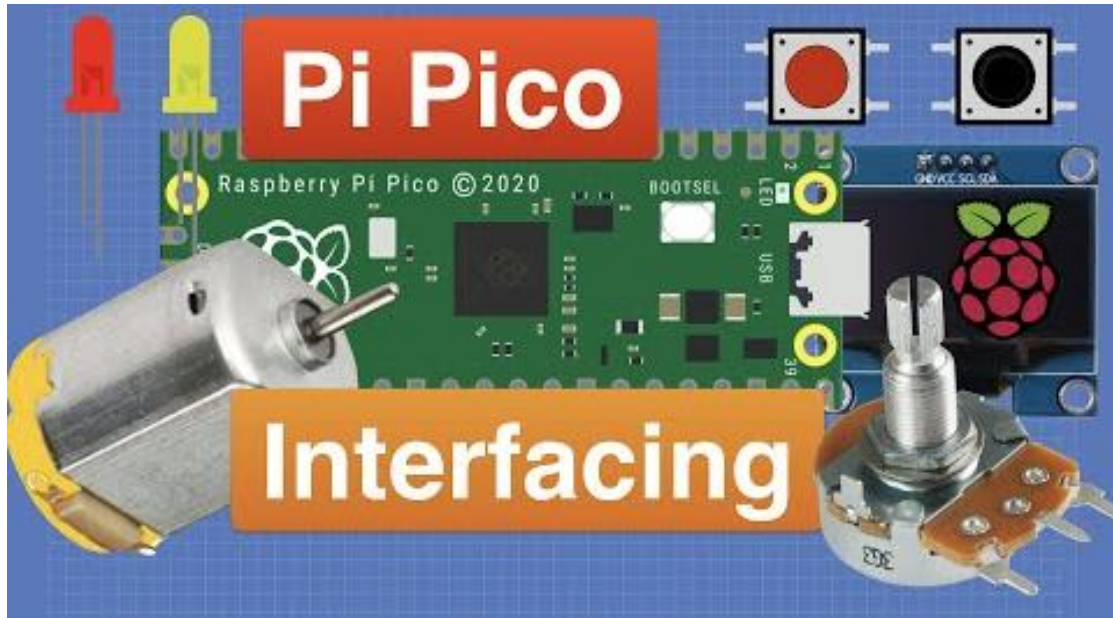


23		Keyestudio SR01 Ultrasonic Sensor	1
24		9G 90° Servo	1
25		Raspberry Pi Pico Board	1
26		Keyestudio Raspberry Pico IO Shield	1
27		Keyestudio JMFP-4 17-Key Remote Control	1
28		USB Cable	1
29		F-F Dupont Wire	1



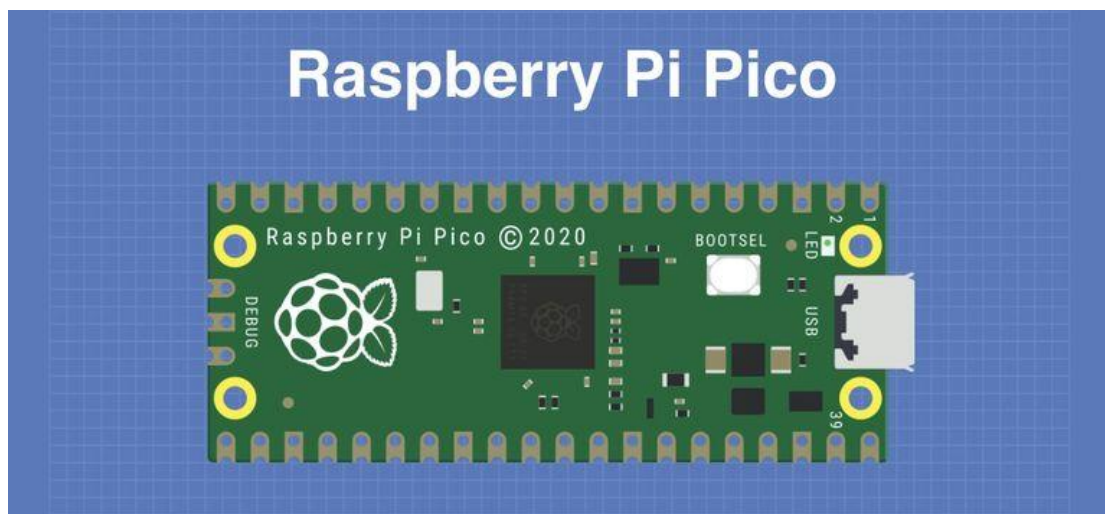
### 3. Raspberry Pi Pico and Thonny

#### 3.1. Raspberry Pi Pico



At the end of January 2021, the Raspberry Pi Foundation launched the Raspberry Pi Pico, which received a lot of attention due to its high-performance and low-cost.

The size of Pico is 21mm \*51mm, which is similar to Arduino Nano' s.

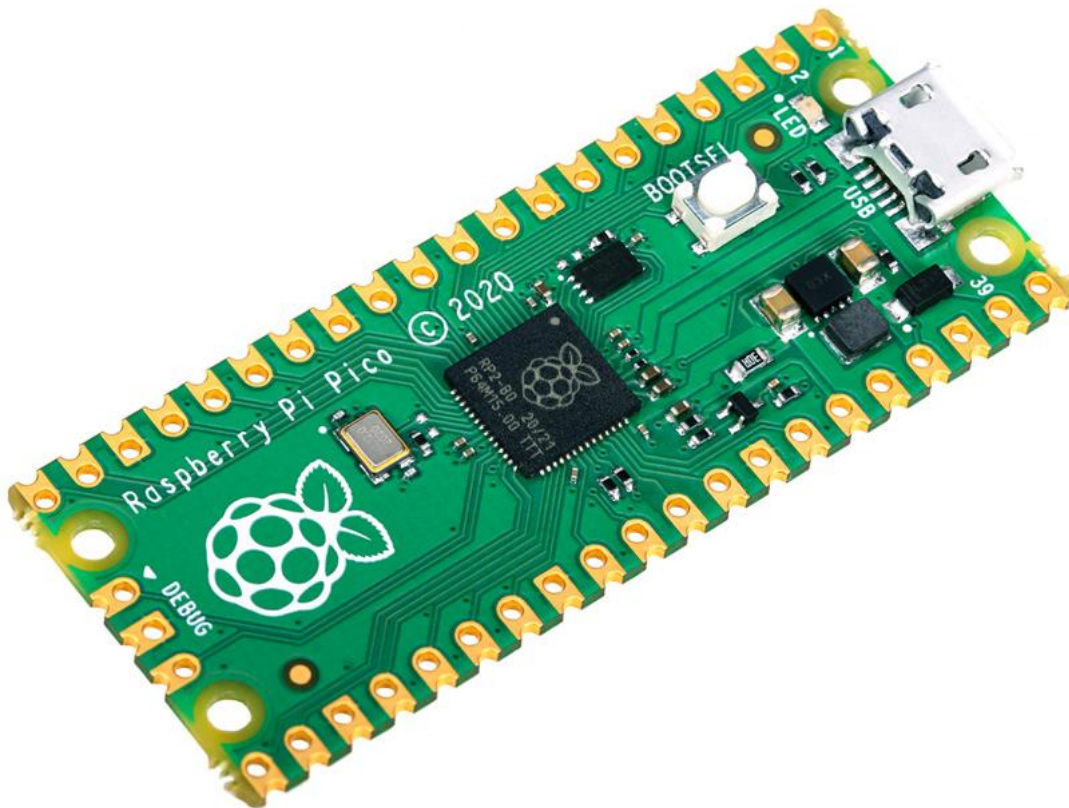


Raspberry Pi Pico is a low-cost, high-performance microcontroller board

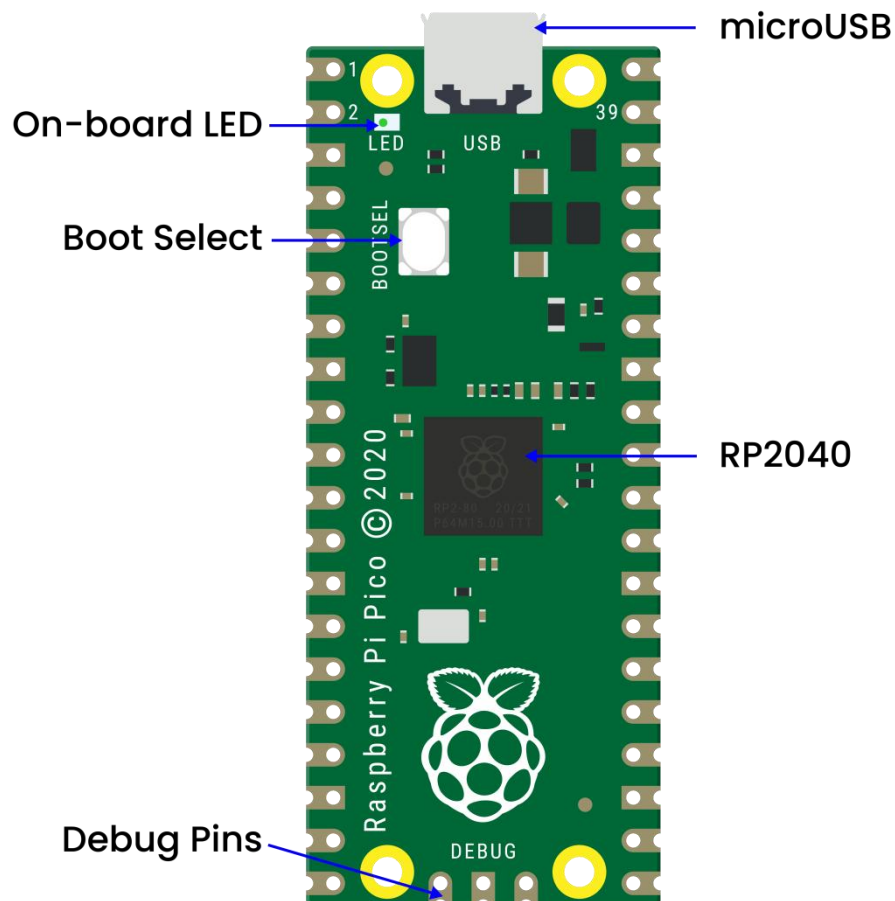




with flexible digital interfaces. It integrates RP2040 microcontroller chip designed by Raspberry Pi, with dual-core Arm Cortex M0+ processor running up to 133 MHz, embedded 264KB of SRAM and 2MB of on-board Flash memory, as well as 26 multi-function GPIO pins. For software development, either Raspberry Pi's C/C++ SDK, or the MicroPython is available. In this tutorial, we will use MicroPython.



The bare board does not come with pins and you need to solder them yourself. This is a well-made board that can also be used as an SMD component and soldered directly to a printed circuit board.



The most predominant feature on the board is the microUSB connector at one end. This is used both for communication and to supply power to the Pico. An on-board LED is mounted next to the microUSB connector, it is internally connected to GPIO pin 25. It's worthwhile to note that this is the only LED on the entire Pico board.

The BOOTSEL pushbutton switch is mounted a bit down from the LED, it allows you to change the boot mode of the Pico so that you can load MicroPython onto it and perform drag-and-drop programming.

At the bottom of the board, you'll see three connections, these are for a serial Debug option that we won't be exploring here.





3-pin Debug connector. They are square as opposed to rounded like the other connections.

**VBUS:** This is the power from the microUSB bus, 5V. If the Pico is not being powered by the microUSB connector then there will be no output here.

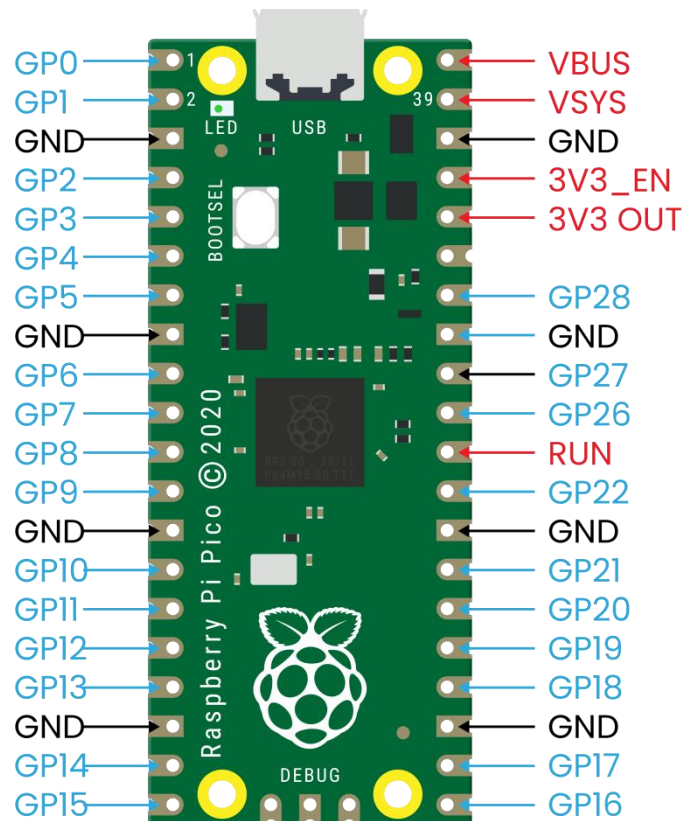
**VSYS:** This is the input voltage, which can range from 2 to 5V. The on-board voltage converter will change it to 3.3V for the Pico.

**3V3:** This is a 3.3V output from the Pico's internal regulator. It can be used to power additional components, providing you keep the load under 300ma.

**3V3\_EN:** You can use this input to disable the Pico's internal voltage regulator, which will shut off the Pico and any components powered by it.

**RUN:** It can enable or disable the RP2040 microcontroller, it can also reset it.





There are 26 exposed GPIO connections on the Raspberry Pi Pico board. They are laid out pretty well in order, with a “gap” between GP22 and GP26 (those “missing” pins are used internally). All these pins have multiple functions, and you can configure up to 16 of them for PWM. There are two I2C buses, two UARTs, and two SPI buses, these can be configured to use a wide variety of GPIO pins.

The Pico has three Analog-to-Digital Converters, they are ADC0-GP26, ADC1-GP27, ADC2-GP28, and plus ADC-VREF converter used internally for an on-board temperature sensor. **Note: The ADCs have a 12-bit resolution. However, the MicroPython has scaled the 12-bit resolution into a 16-bit resolution, which means that we will receive ADC values from 0 to 65535.**



The microcontroller's working voltage is 3.3V, indicating that 0 corresponds to 0V and 65535 corresponds to 3.3V.

You can also provide an external precision voltage-reference on the ADC\_VREF pin. One of the grounds, the ADC\_GND on pin 33 is used as a ground point for that reference.

### **Raspberry Pi Pico Configuration**

Dual-core Arm Cortex-M0 + @ 133MHz

2 × SPI, 2 × I2C, 2 × UART

264KB of SRAM, and 2MB of on-board Flash memory

16 PWM channels

QSPI bus controller, supporting up to 16 MB of  
external Flash memory

USB 1.1 with host and device support

DMA controller

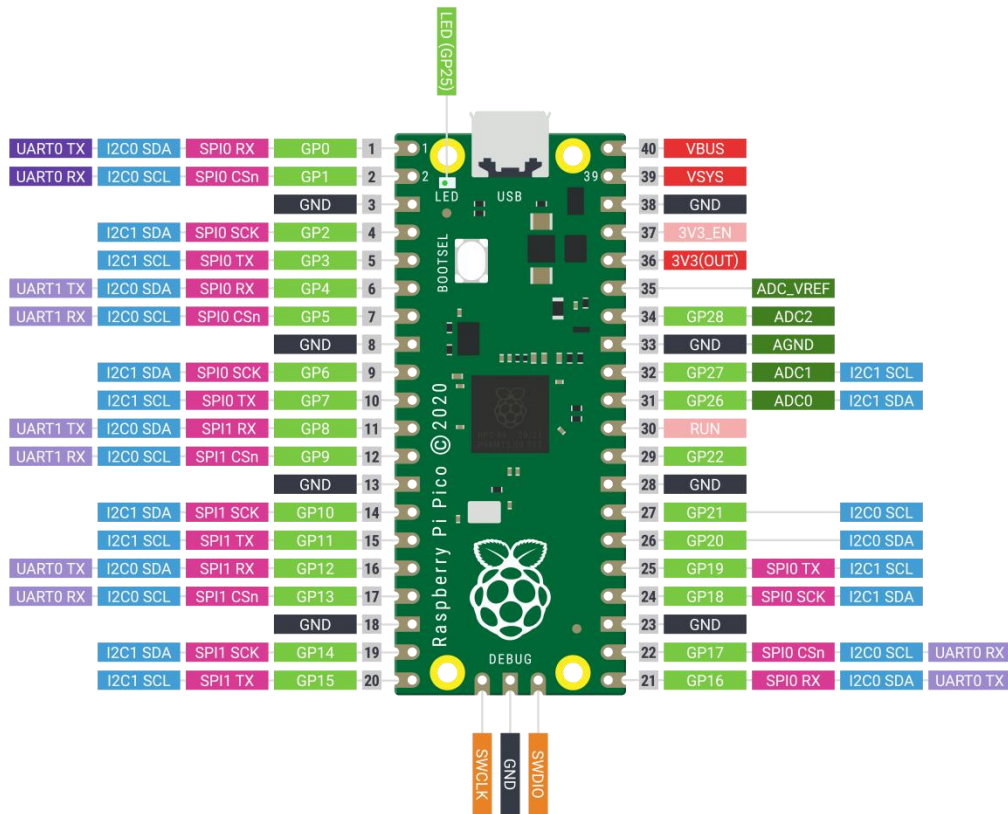
8 × Programmable I/O (PIO) state machines for  
custom peripheral support

30 GPIO pins, of which 4 can optionally be used as  
analog inputs

Drag-and-drop programming using mass storage over  
USB



## Pinout Diagram:



Raspberry Pi did release a ton of technical documentation, plus a great guide called *Get Started with MicroPython on Raspberry Pi Pico*. It's available in softcover, and as a PDF download as well. For more information, please refer to:

<https://www.raspberrypi.com/products/raspberry-pi-pico/>

### 3.2. MicroPython IDE-----Thonny

**Programming the Pico:** You could use C/C++ or MicroPython.



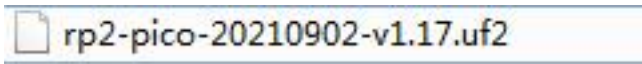
MicroPython is an interpreted language that is made specifically for microcontrollers. Many microcontroller users have familiarity with C/C++ as they are used on the Arduino and ESP32 boards. In this tutorial, we will use Thonny recommended by Raspberry Pi. Thonny bills itself as a “Python IDE for Beginners” , and it is available for Windows, Mac OSX and Linux. It was also part of the Raspberry Pi operating system(formerly Raspbian).

**Boot and Install MicroPython:** The first thing that we need to do is to get MicroPython installed onto the Pico.

## Download and Burn Firmware

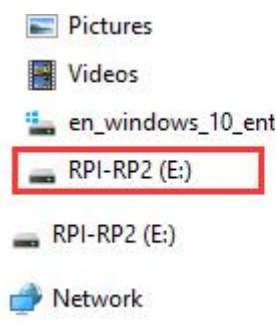
Go to the official website to download the UF2 file:

<https://www.raspberrypi.com/documentation/microcontrollers/#getting-started-with-micropython>

What I downloaded is  . Once the download is complete, we proceed to burn the firmware.

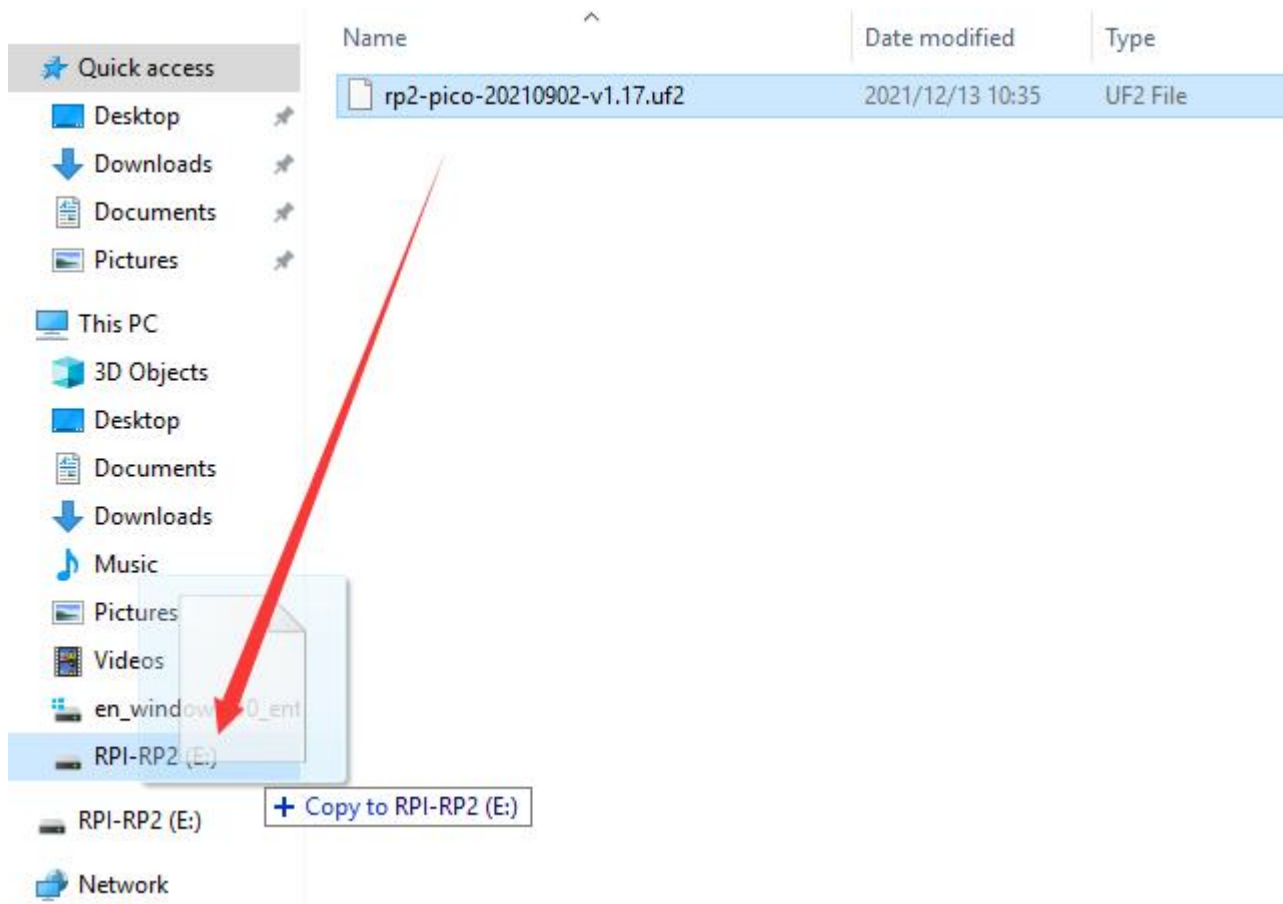
With BOOTSEL held down, then plug the Pico into Raspberry Pi or your computer' s USB port.

Release it after the connection was finished. You should see a drive appearing on your computer with the name “*RPI-RP2*” .





Move the UF2 file into "*RPI-RP2*" , and the Raspberry Pi Pico will automatically restart. At this point, the burning is complete.



## Download and Install Thonny IDE

Enter the official website to download Thonny, we should download the latest version for Windows.

Link: <https://thonny.org/>



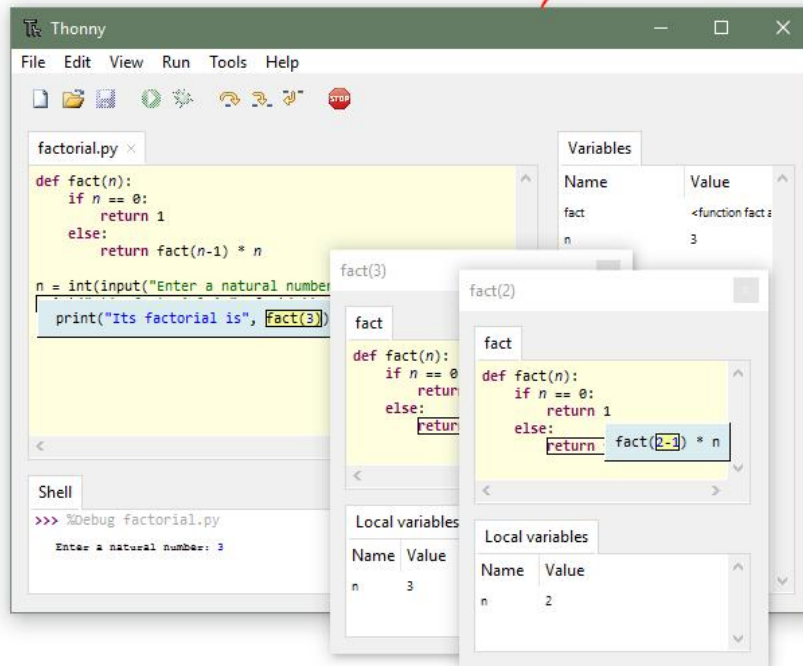
# Thonny

Python IDE for beginners

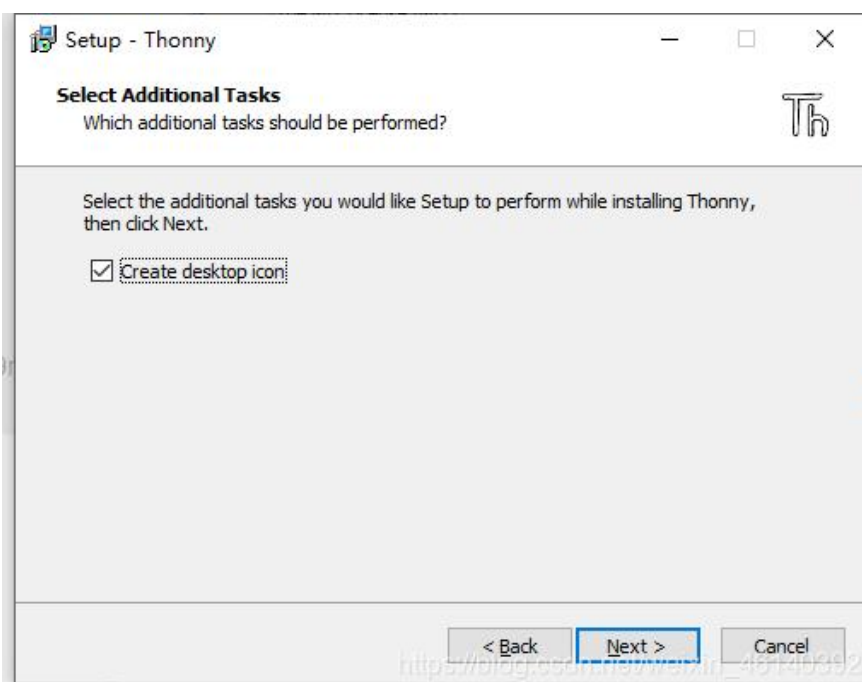


Download version 3.3.13 for

[Windows](#) • [Mac](#) • [Linux](#)

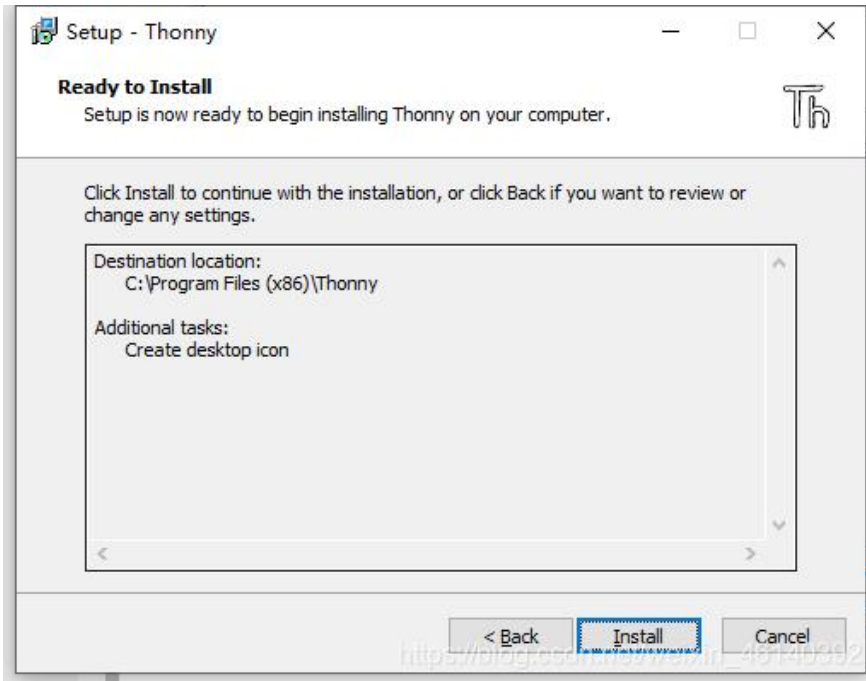


After downloading, we start installing the software. Click "Next" , then click "I accept the agreement" and click "Next" again. After that, we choose "Create desktop icon" and click "Next" , or just click "Next" to go to the next step (you can open the file in the corresponding folder).





When we see the contents shown below, click "Install" to complete the installation.



Finally, click "Finish".

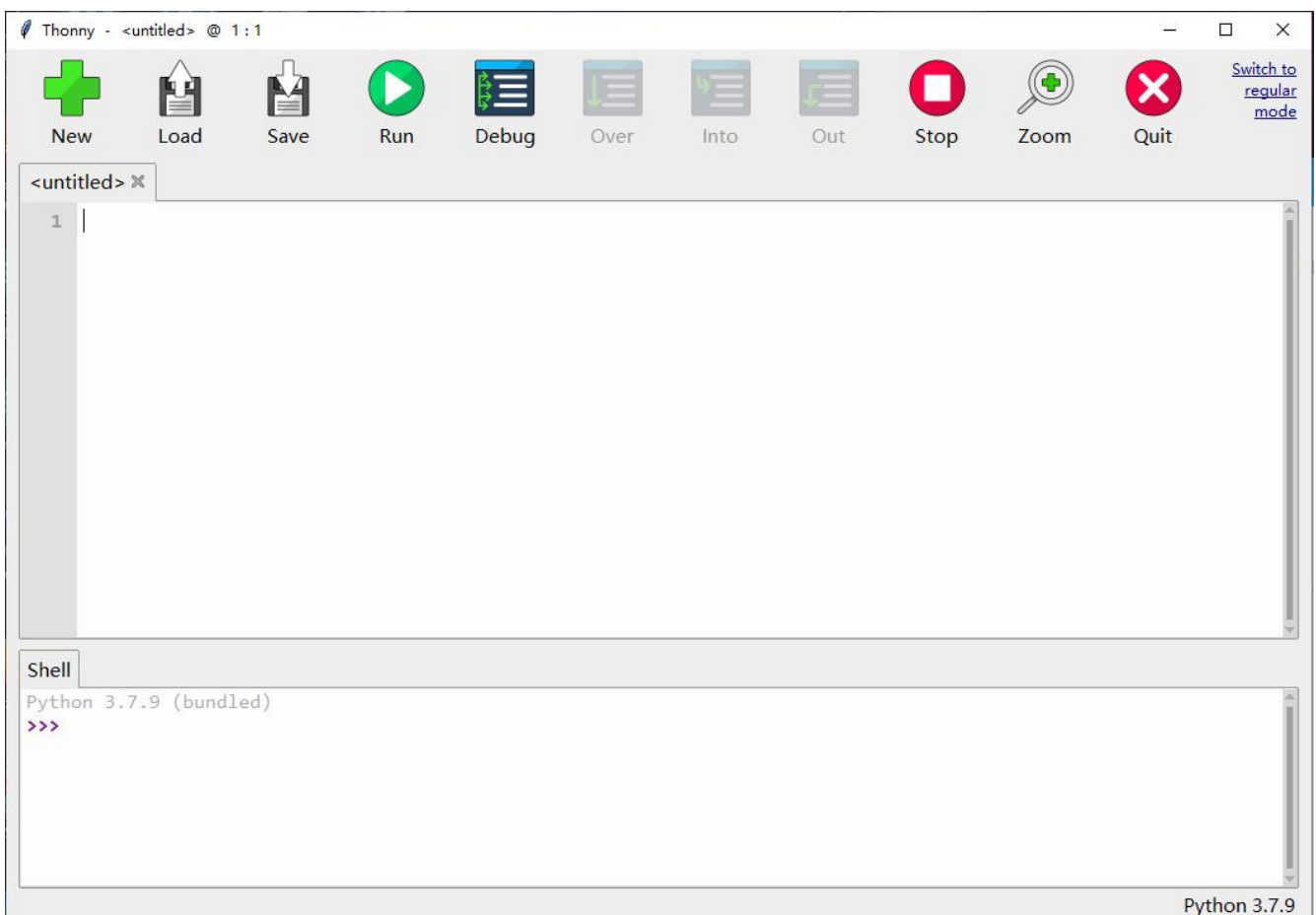


Now we run the Thonny software. First, choose the language we need and "Raspberry Pi" in "Initial settings" , then click "Let's go!" .





Next, we will see the interface as shown below.

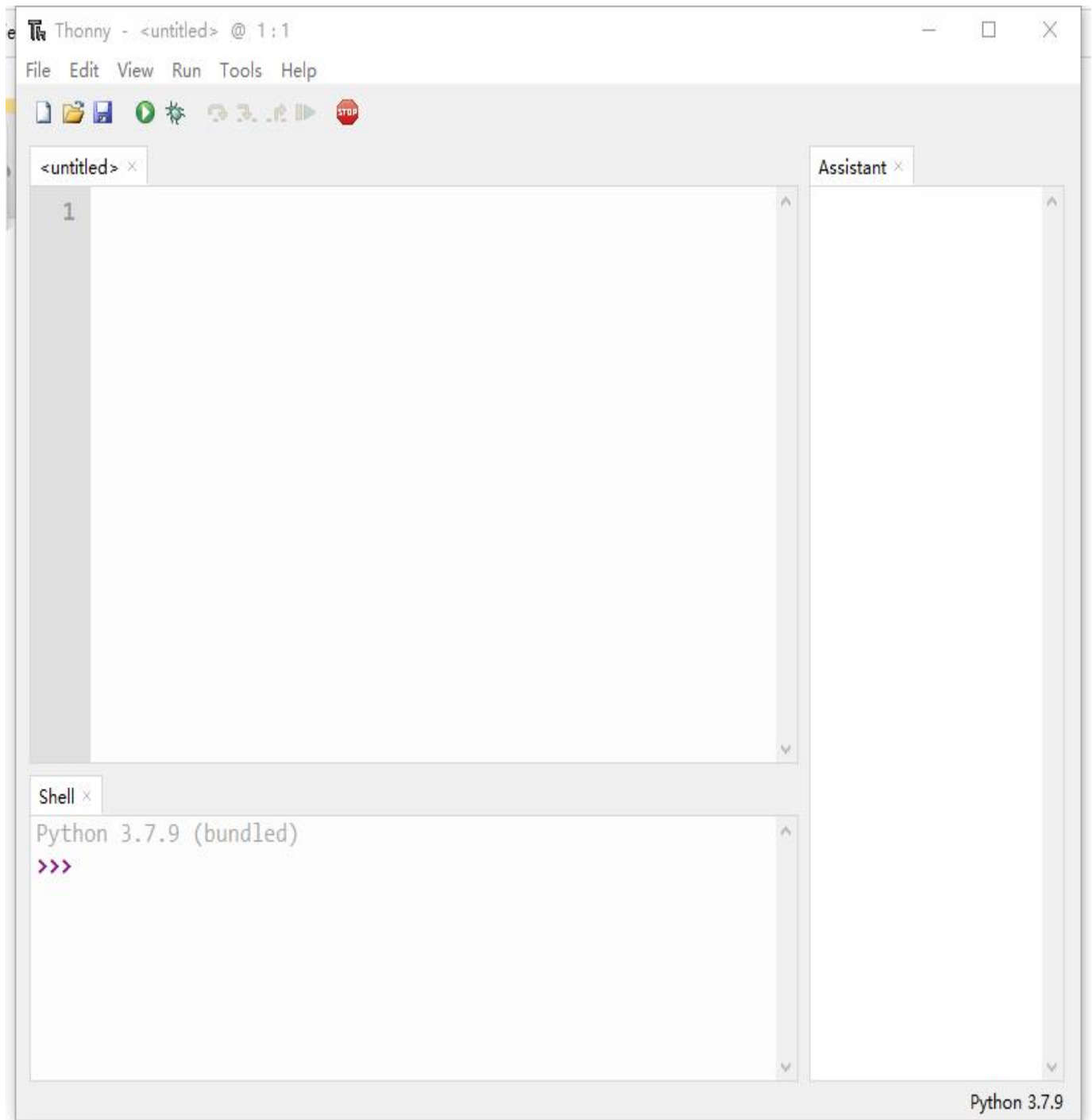


Click on the text in the top right of the window to switch to "Regular Mode". Then restart the program, the interface will be like this as illustrated

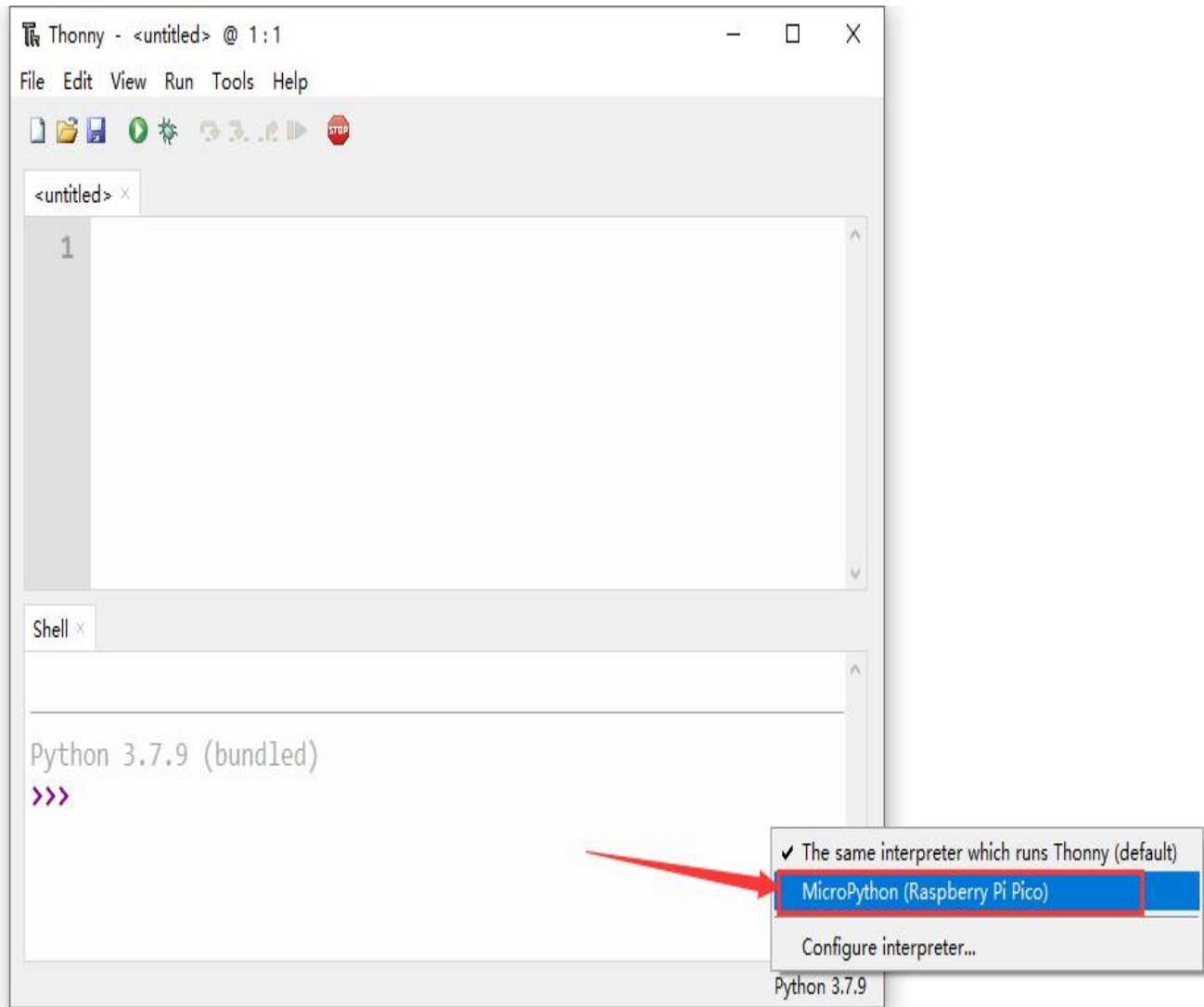




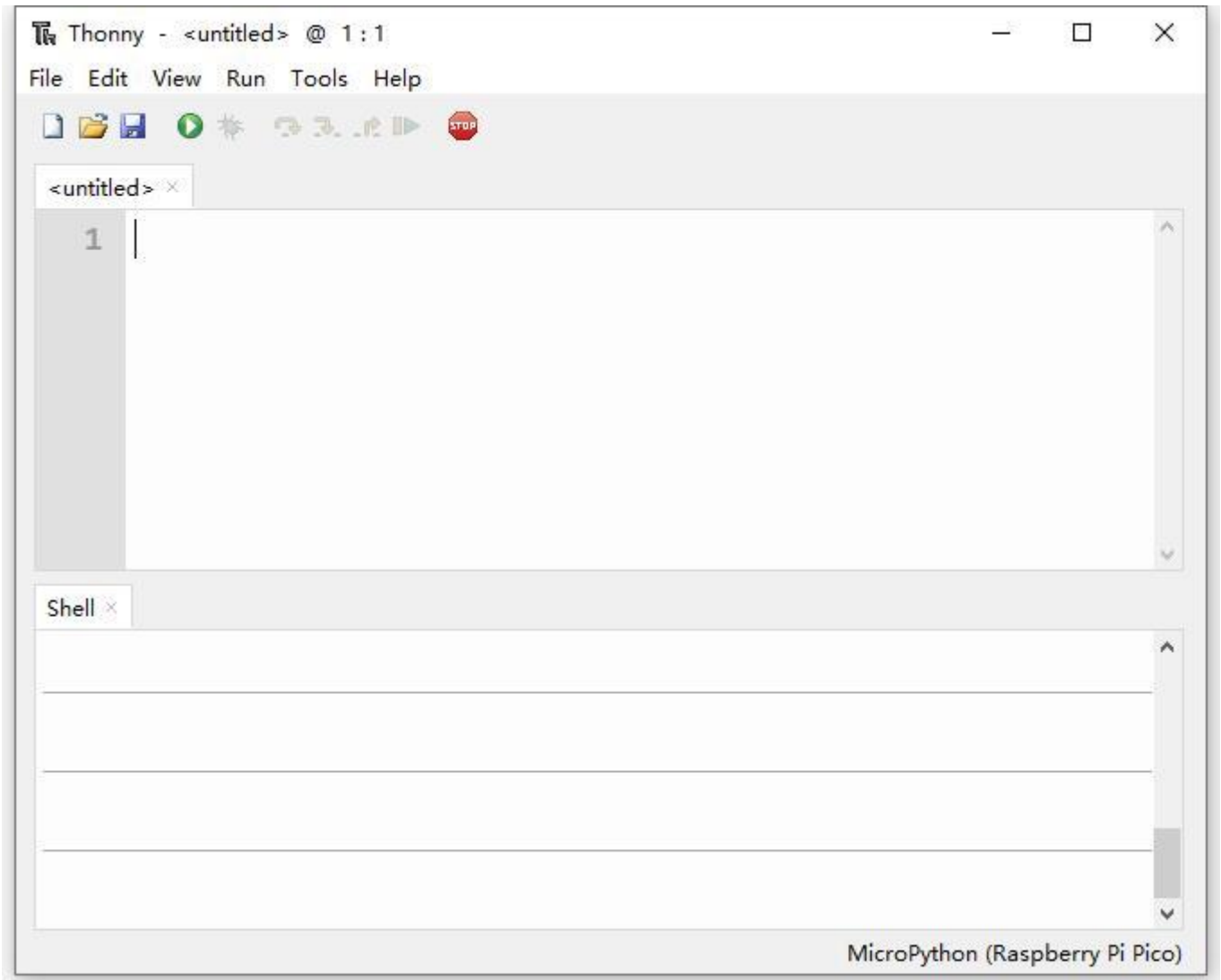
below.



Click on the word “Python” followed by a version number at the bottom-right of the Thonny window, then choose “MicroPython (Raspberry Pi Pico)” .



The Raspberry Pi Pico interpreter is only available in the latest version of Thonny. If you're running an older version, you can't choose the corresponding interpreter. After choosing the interpreter, the interface will be like this as follows.



### 3.3 Install Drivers

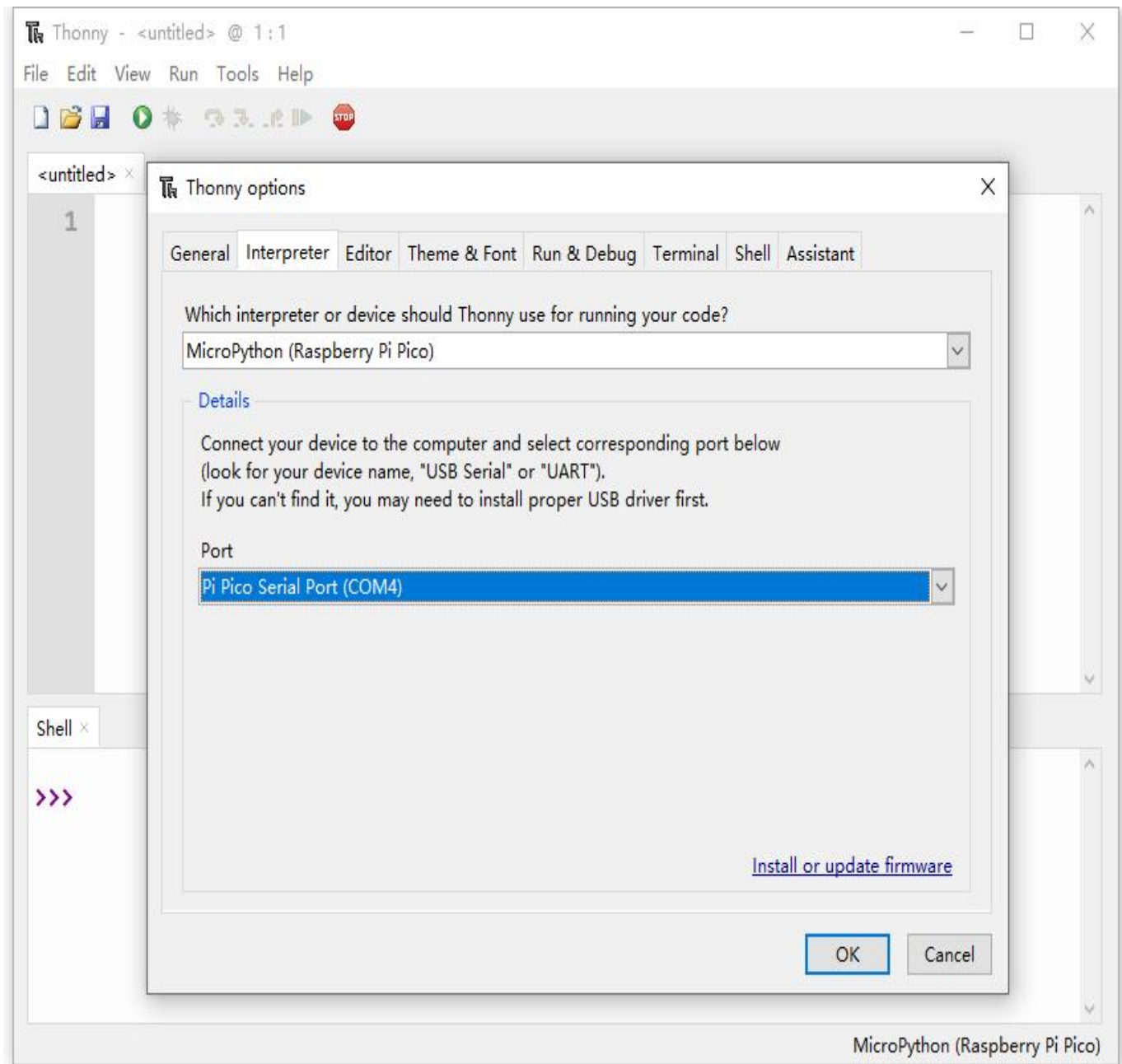
Wire the Pi Pico board with the USB port of a computer via a MicroUSB. If the Pi Pico shield has installed MicroPython, and installed "Board CDC " on the computer, then it will show corresponding ports of "Pi Pico Serial Port (COM) " on Device Manager. If you have a Raspberry Pi, you can connect the Pico to the Raspberry Pi directly. The Raspberry Pi has a lot of built-in software that can be used directly. If wire the Pico with the computer,



please follow the steps below.

## **Windows 10**

When plug the Pico into the computer, the system will automatically identify serial port and install corresponding driver. You can find "USB Serial" on Device Manager. On my computer is COM4. You can find the corresponding COM port in Thonny options (Tools-Options-Interpreter).



If it shows the following information, indicating that your Pico board is successfully connected to the computer.

MicroPython v1.17 on 2021-09-02; Raspberry Pi Pico with RP2040

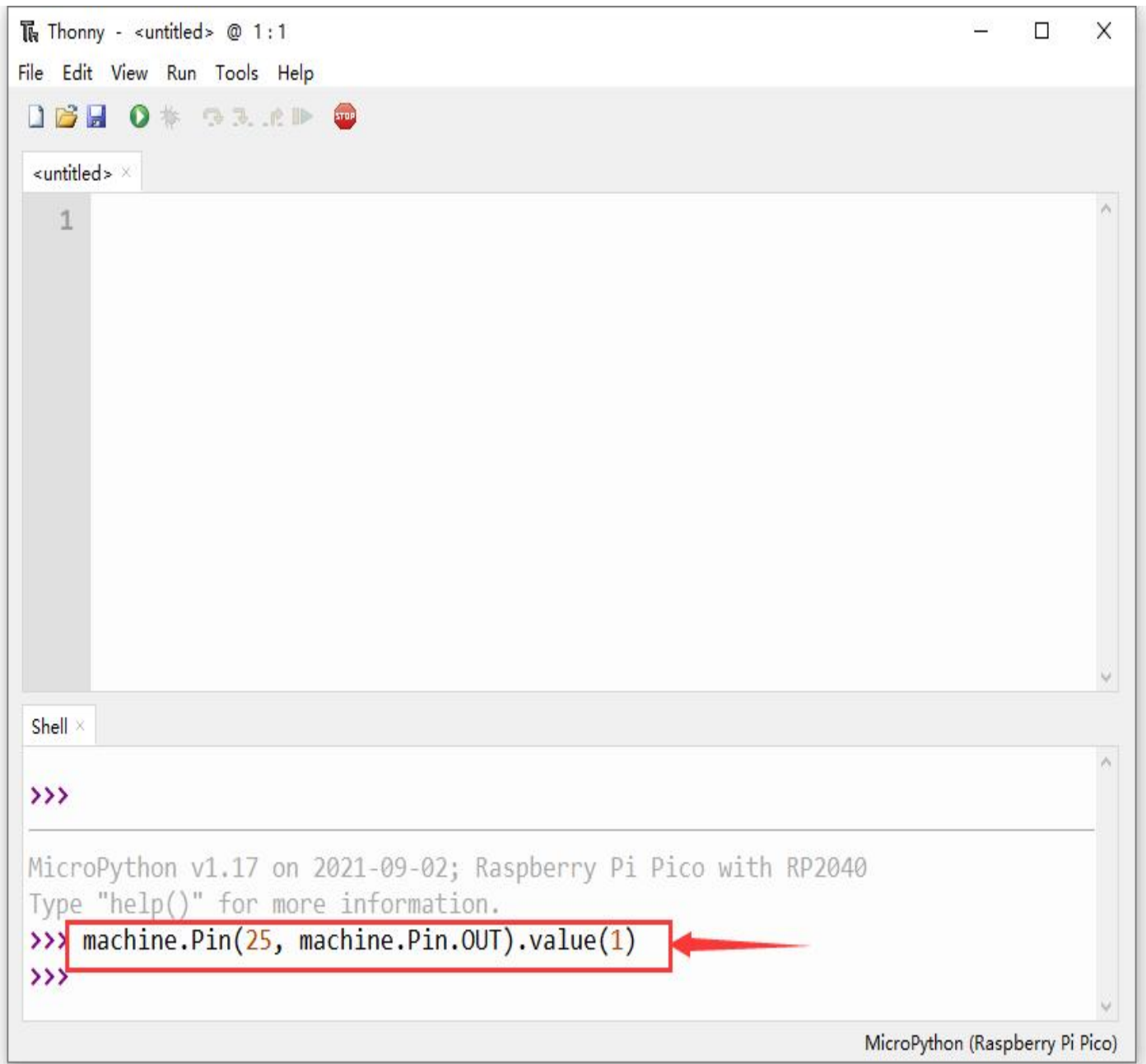
Type "help()" for more information.

Then we input the following command behind `>>>`.

`machine.Pin(25, machine.Pin.OUT).value(1)`

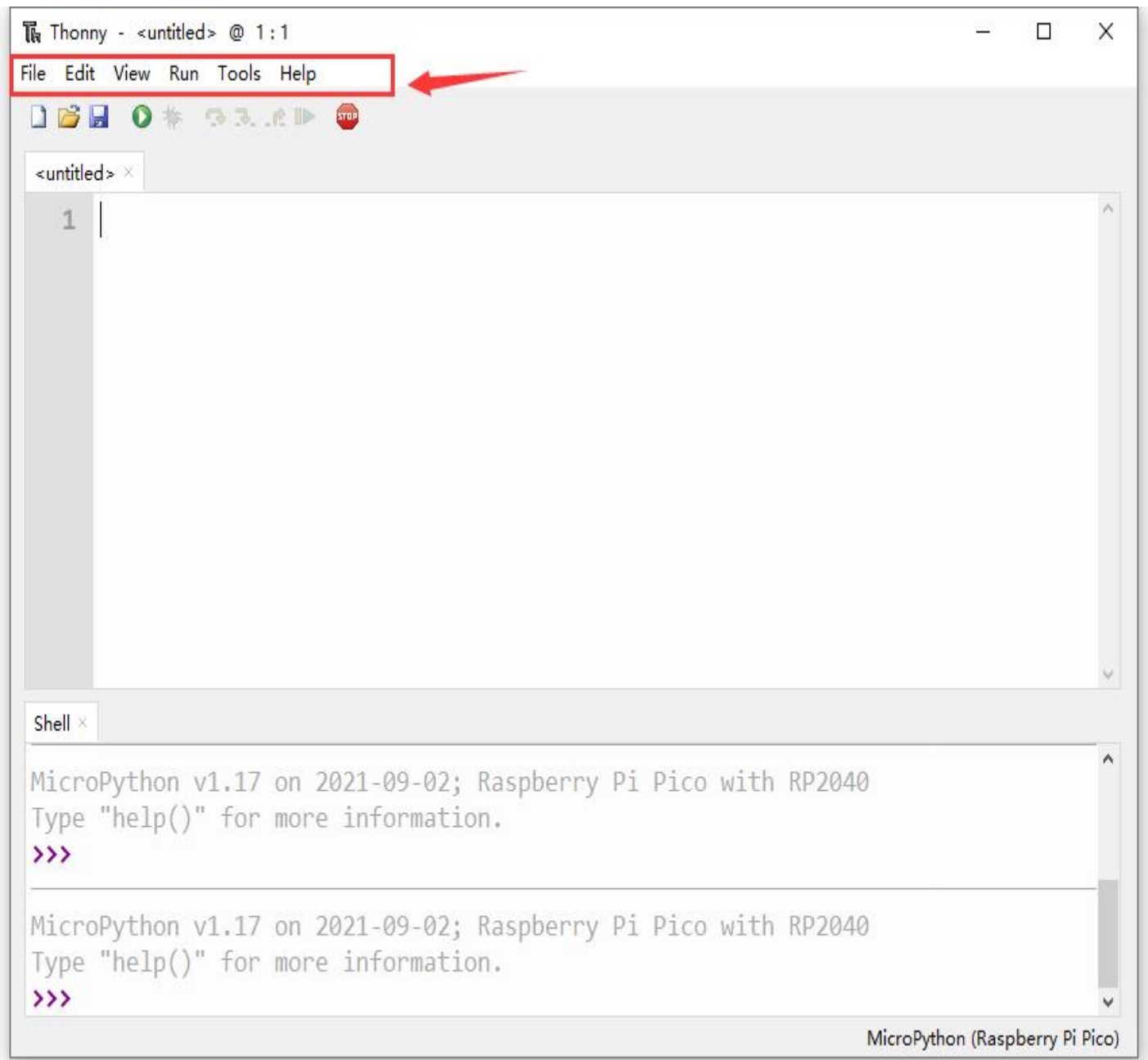


Press “Enter” , if the on-board LED lights up, it means that Thonny works.

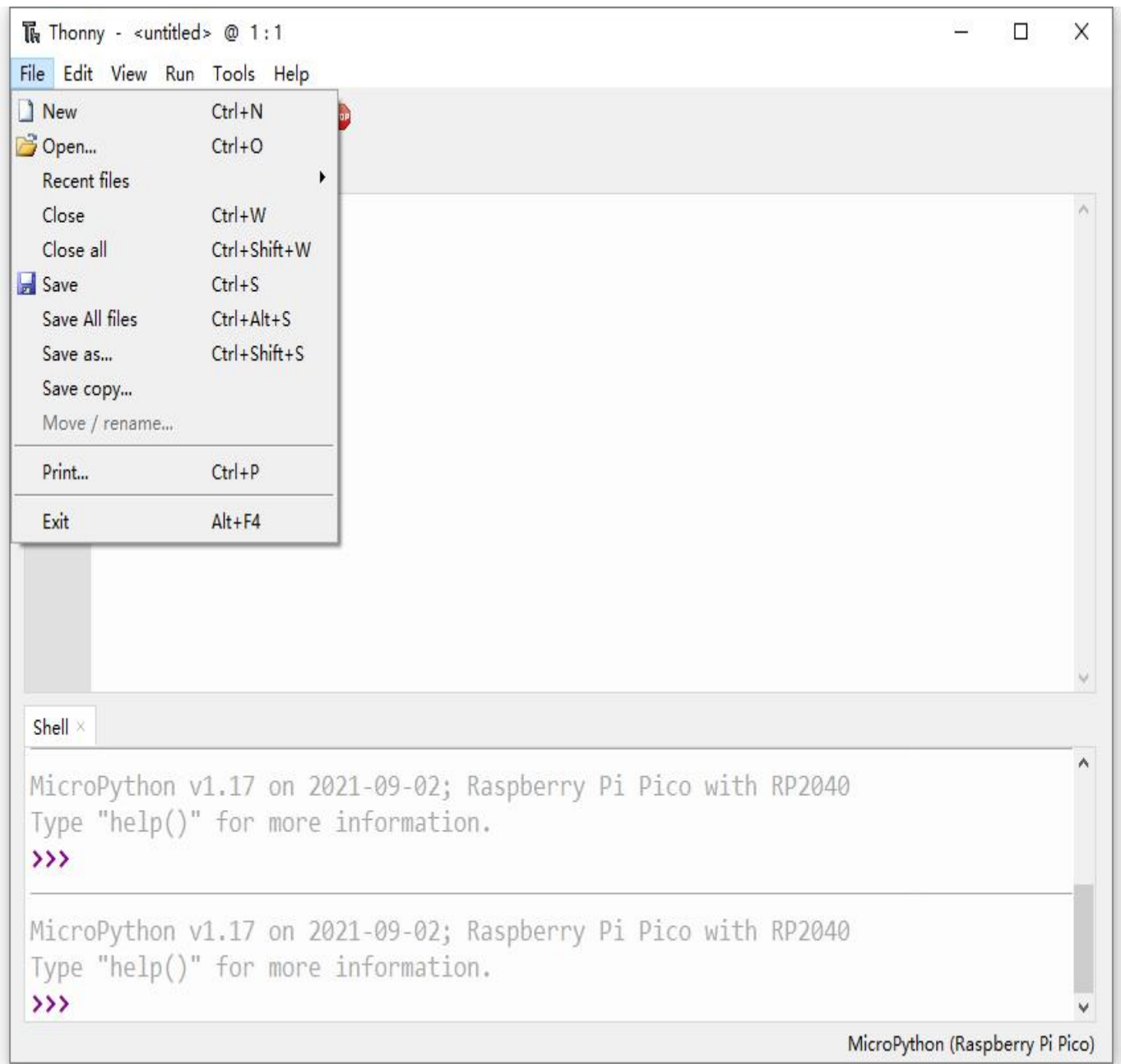


### 3.4 Thonny User Interface

After installing the IDE and the driver, now we will introduce Thonny user interface. At the top is the main menu, there are “File” , “Edit” , “View” , “Run” , “Tools” and “Help” .

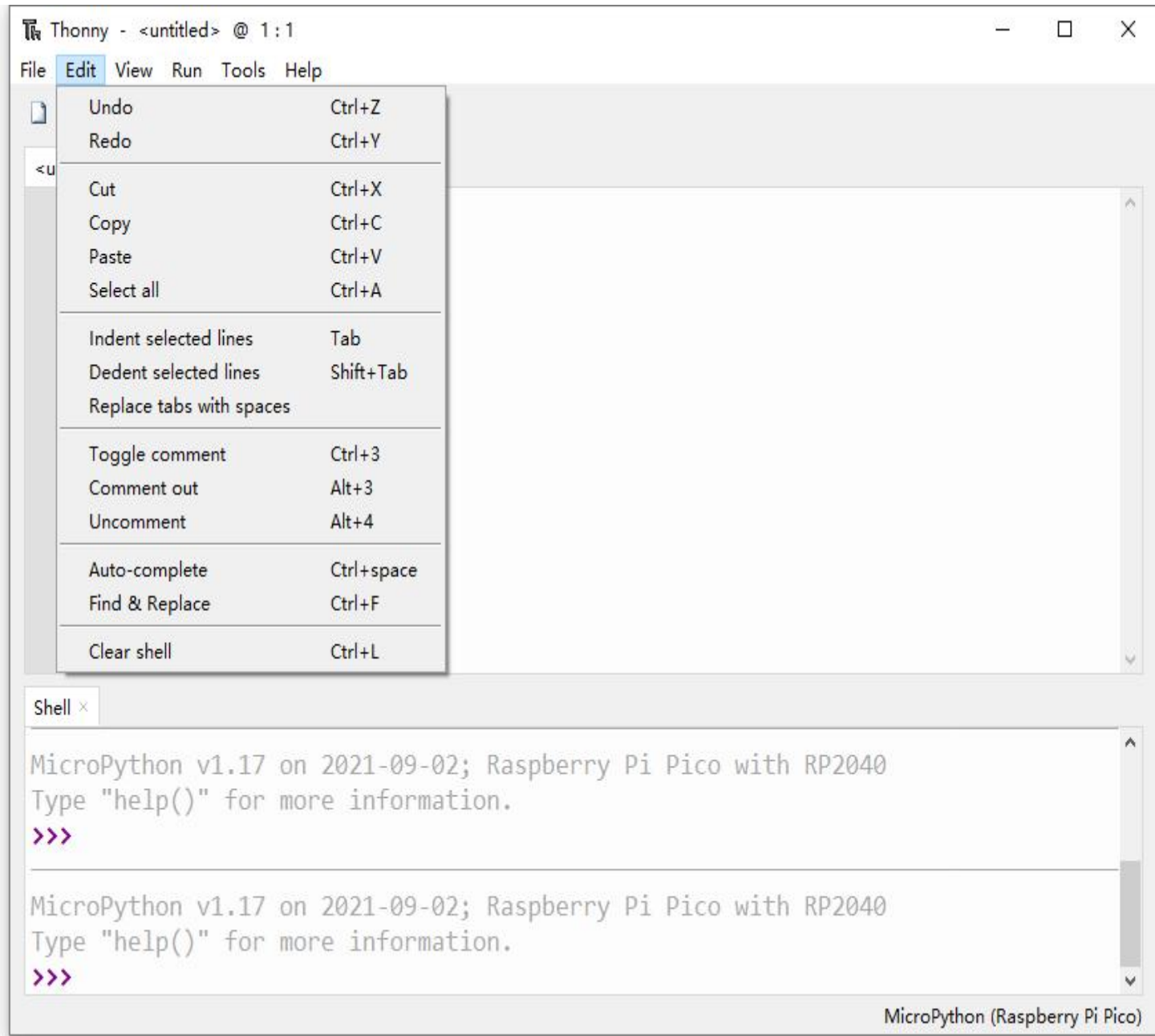


Click “File” , it shows some operations related to files.

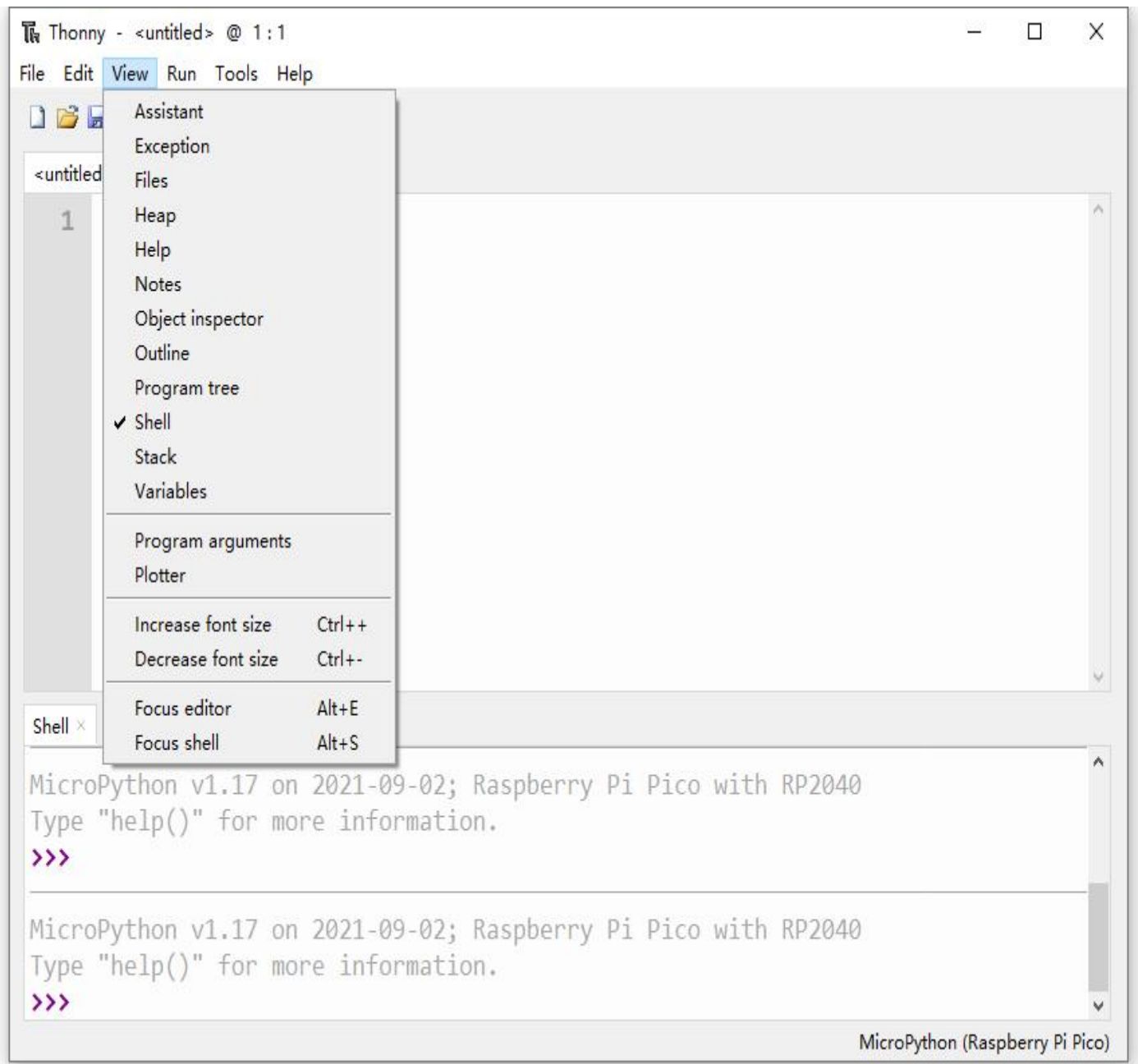


Click “Edit” , these are some options about code, such as copying, cutting, pasting.

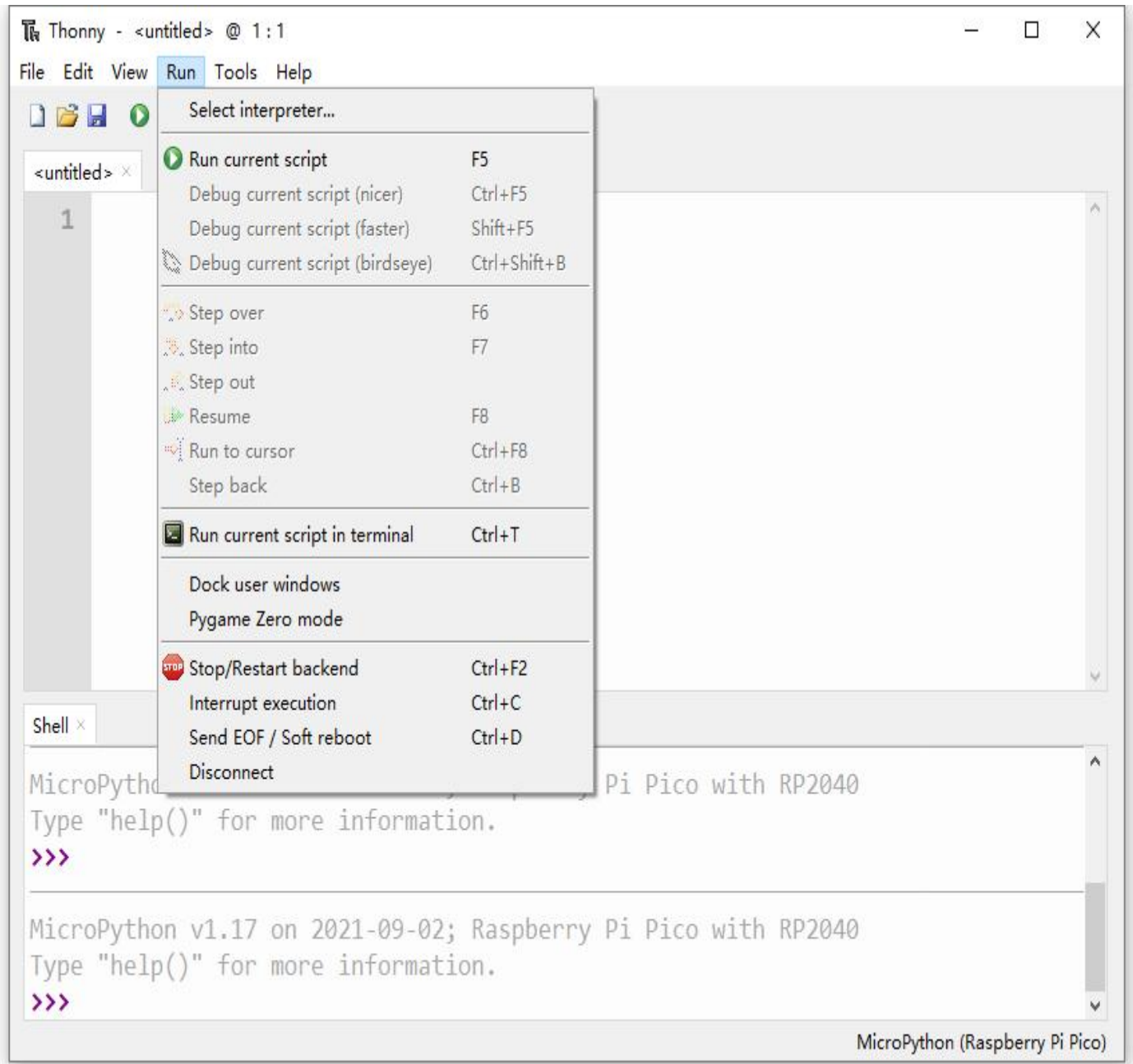




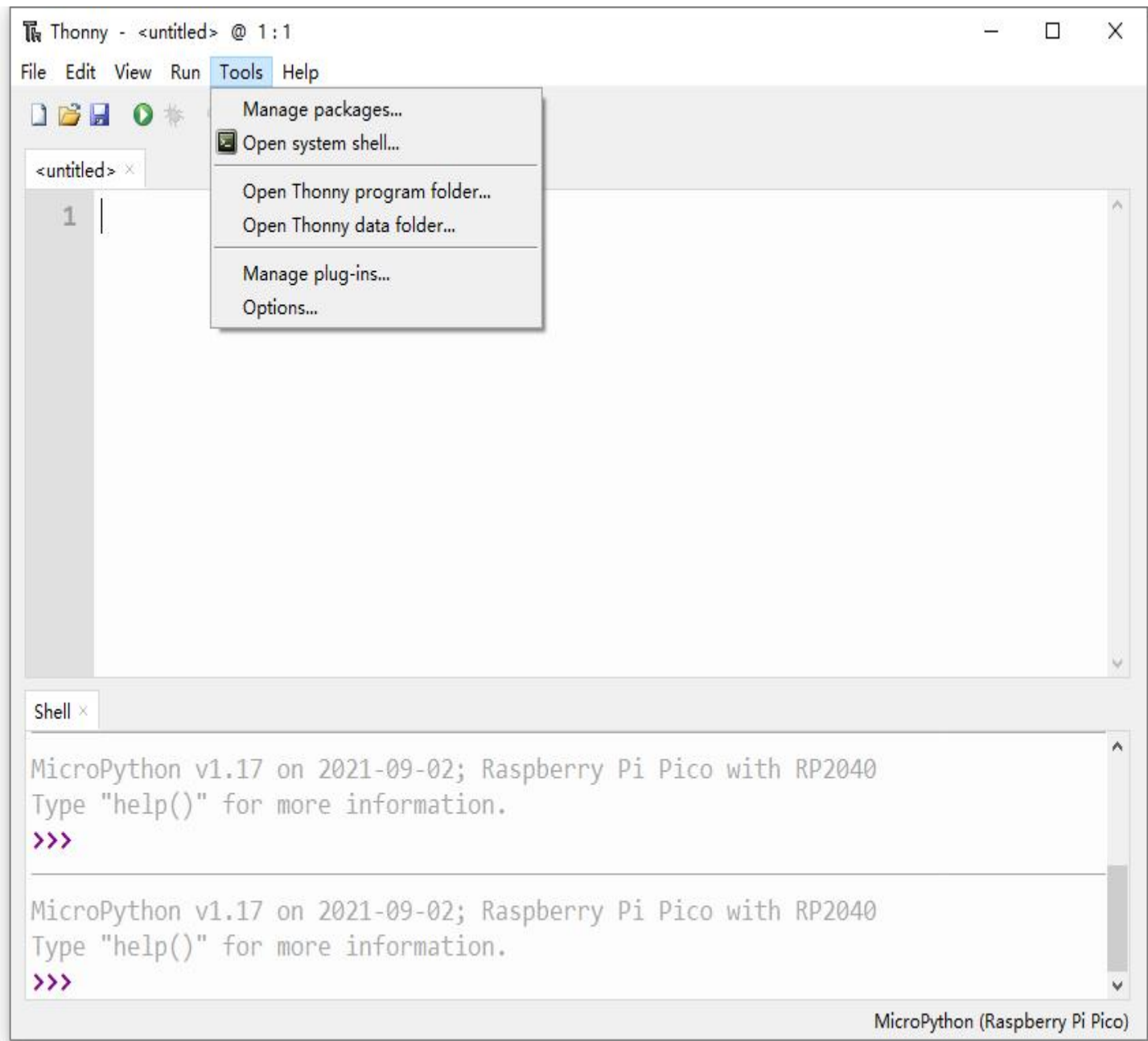
In the View drop-down menu, these are the tools to assist you. For example, if we do not tick Shell (the Shell is the “command line” of the Pico, and you can execute code directly here.), the result won’ t be displayed. Click “Files” , the files we saved will be shown on the left.

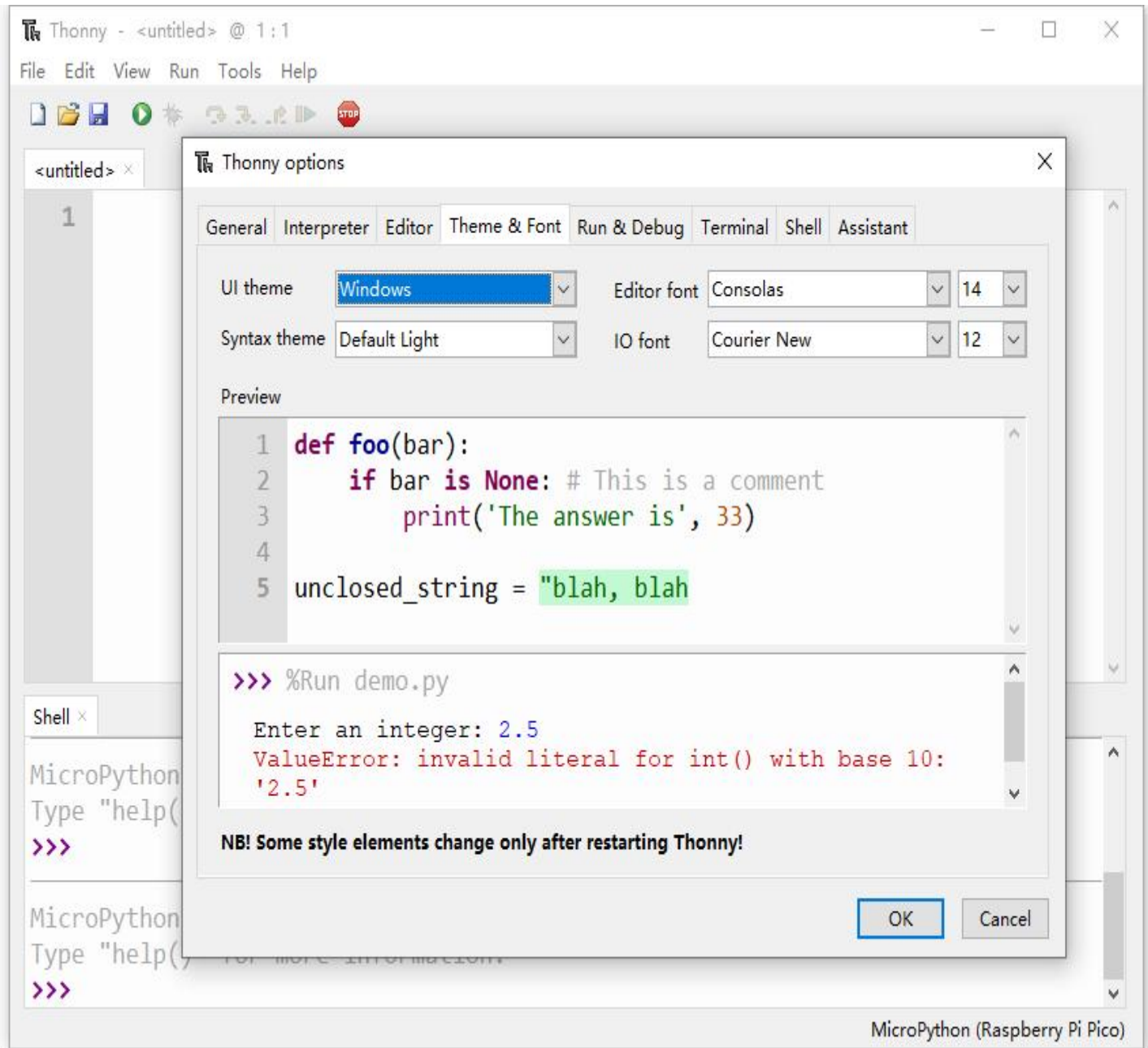


We can select interpreter in the Run drop-down menu, there are also some shortcuts used in programming.

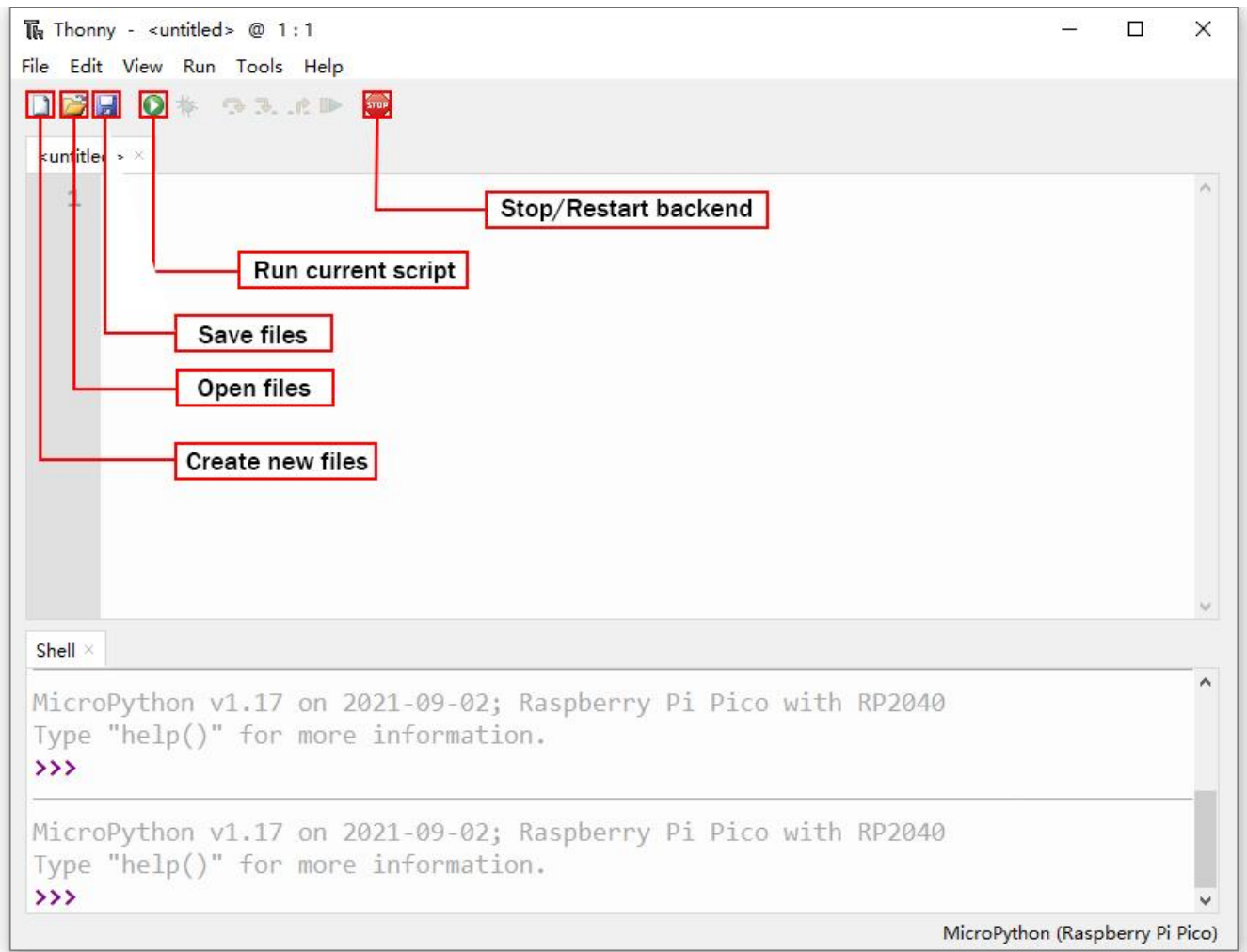


In Tools menu, we can select interpreter, font and import modules, etc.

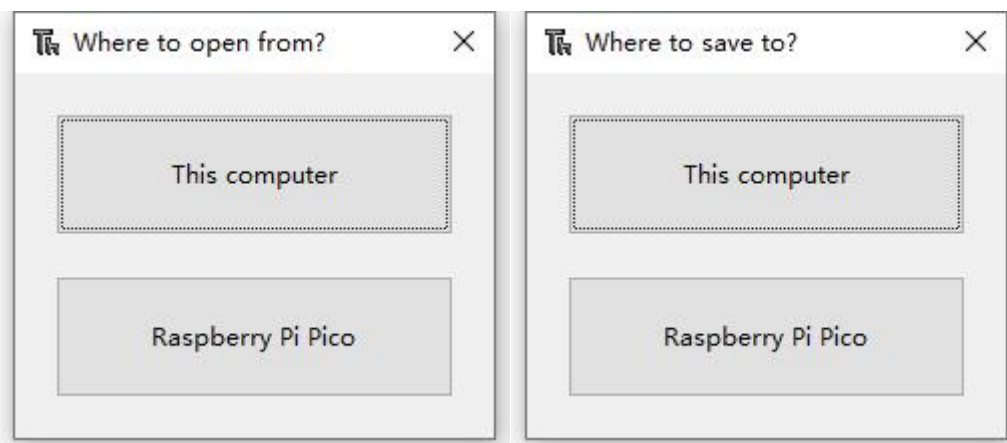




In Help menu, we will see “Help contents” , “Version history” and more.  
The icons below the main menu are our commonly used tool shortcuts.

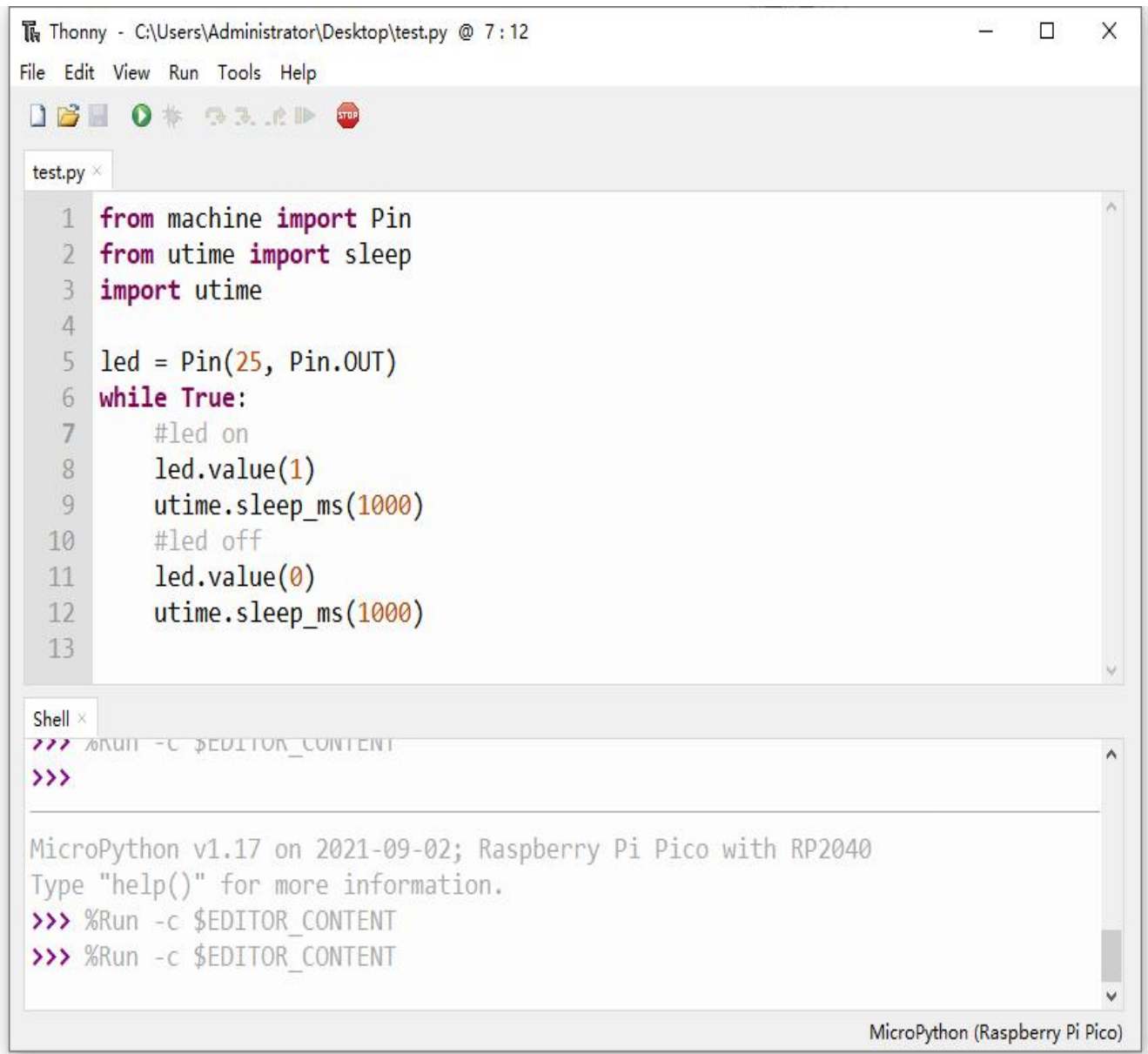


When we open or save files, it will shows the following contents.



We can open programs saved on the computer or the Pico, or save them on This computer or Raspberry Pi Pico.

Copy the code below to the Thonny and save it to the computer as test.py.





```
Thonny - C:\Users\Administrator\Desktop\test.py @ 7:12
File Edit View Run Tools Help

test.py x
1 from machine import Pin
2 from utime import sleep
3 import utime
4
5 led = Pin(25, Pin.OUT)
6 while True:
7     #led on
8     led.value(1)
9     utime.sleep_ms(1000)
10    #led off
11    led.value(0)
12    utime.sleep_ms(1000)
13

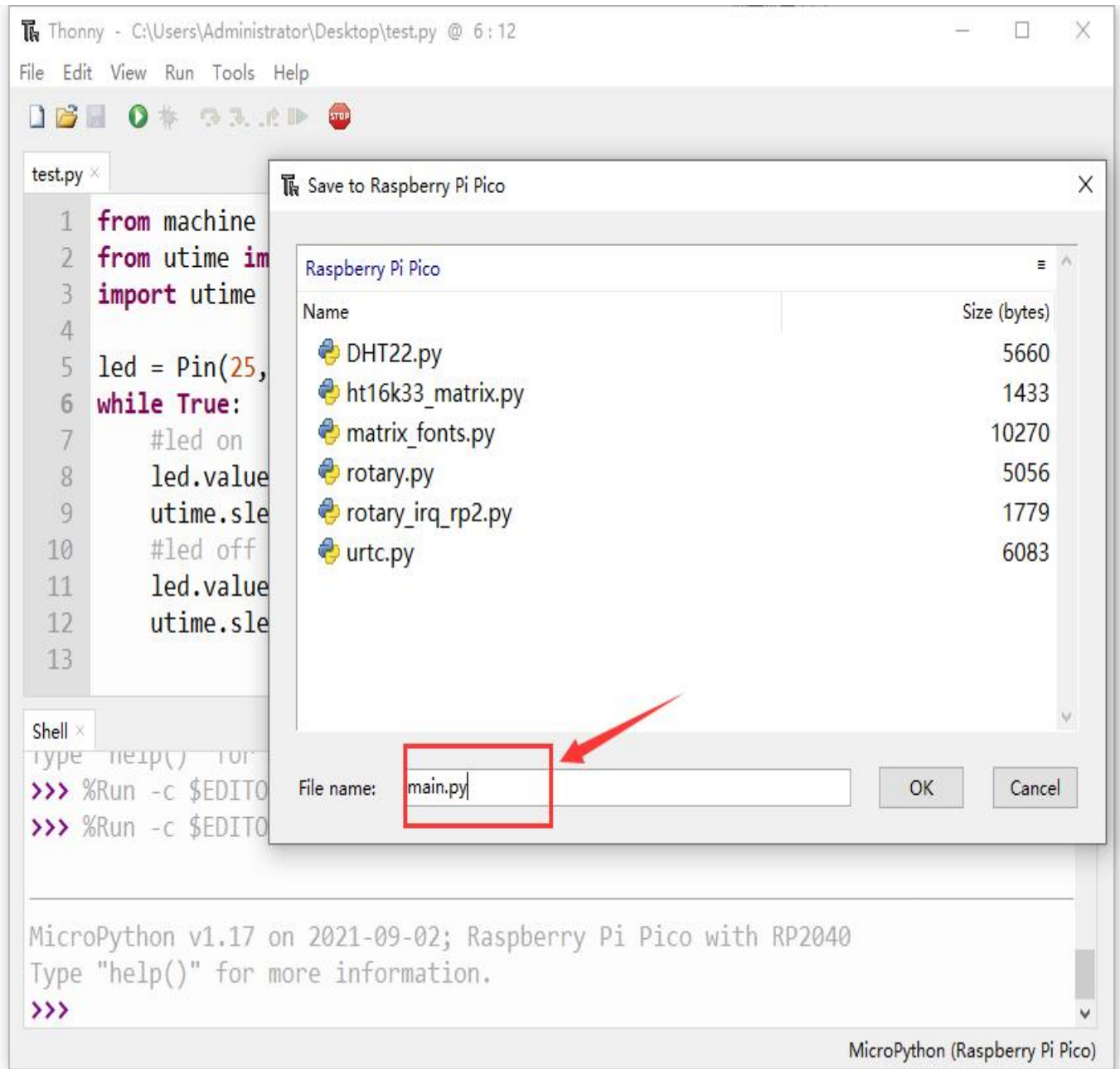
Shell x
>>> %Run -c $EDITOR_CONTENT
>>>

MicroPython v1.17 on 2021-09-02; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
>>> %Run -c $EDITOR_CONTENT

MicroPython (Raspberry Pi Pico)
```

Click  to run the code, the on-board LED will blink at 1 second intervals, then click  to stop, the LED won't blink. If we unplug the MicroUSB cable and plug it in again, the LED won't blink after powering up. This is because we did not name the file main.py and save it to the Pico. Click "File" , then click "Save as..." to choose Raspberry Pi Pico. After that, enter main.py as the file name (don't forget to enter the .py file extension) and click "OK" . Run the code again, the LED will continue to blink.





When we unplug the cable again, then plug it in and power on, the LED will blink. This is because the Raspberry Pi Pico starts running the program saved on `main.py` after powering up.

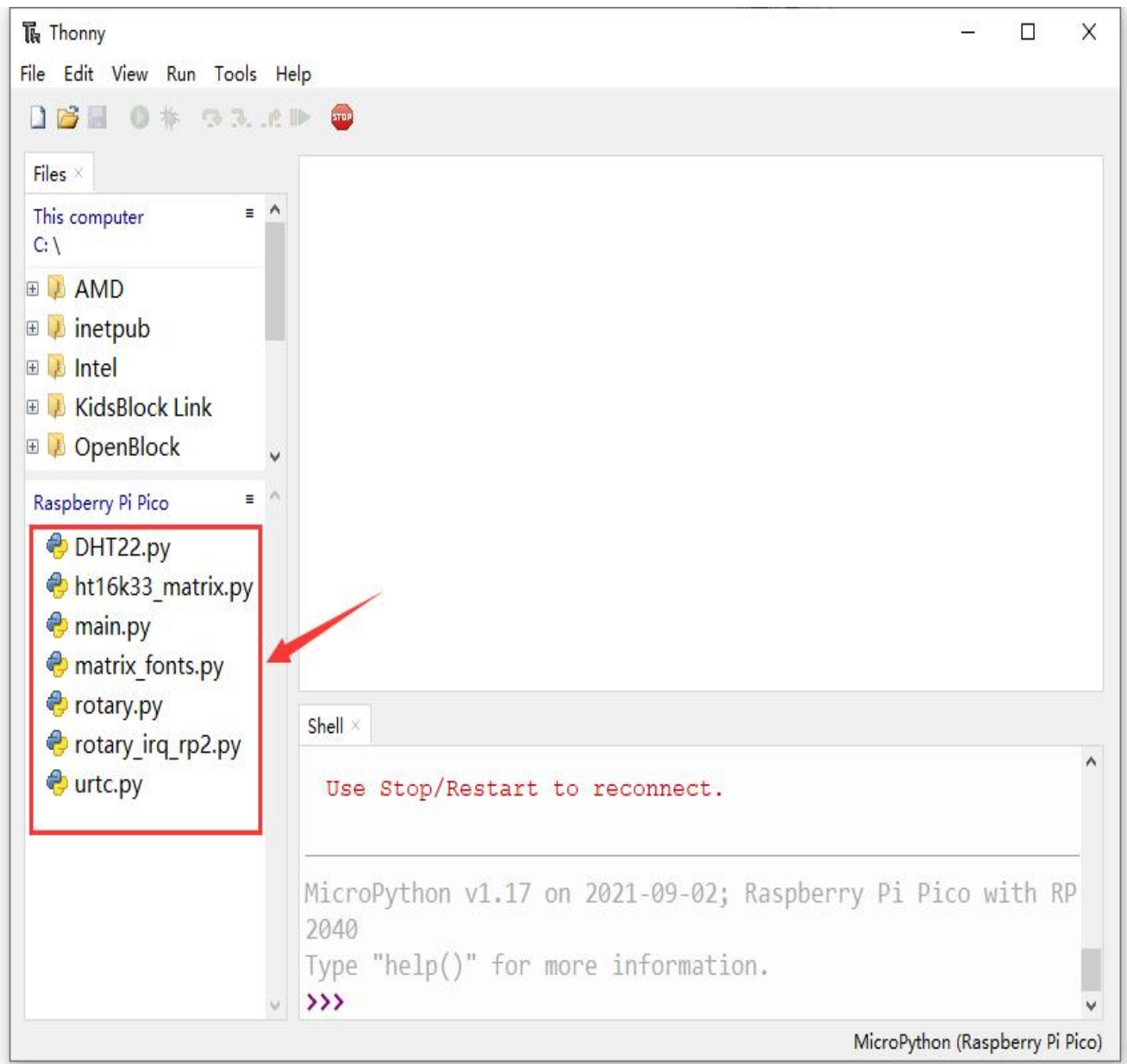
### 3.5 Add Modules





Python is a powerful language due to its modules. Python scripting language with the most rich and powerful class library, enough to support the vast majority of day-to-day applications. By importing modules, this makes it easier for us when using some complex sensors.

The method is simple, just save the module that we need to the Pico, or open the file saved on our computer, click "File" to choose "Save as" , then save it to the Pico board (right click the mouse, you can delete files). For instance, I saved some library files required for these courses on my Pico. Click "View" to choose "Files" , they will be displayed on the left of the interface.



When using sensors, we can import the corresponding modules directly.



```
* http://www.keyestudio.com
...
import machine
import time
import json
import matrix_fonts
from ht16k33_matrix import ht16k33_matrix
## tool to Make Sprites https://gurgieapps.com/tools/matrix
#i2c config
clock_pin = 1
data_pin = 0
bus = 0
i2c_addr_left = 0x70
use_i2c = True

def scan_for_devices():
    i2c = machine.I2C(bus,sda=machine.Pin(data_pin),scl=machine.Pin(clock_pin))
    devices = i2c.scan()
    if devices:
        for d in devices:
            print(hex(d))
    else:
        print('no i2c devices')
```

## 3.6 Keyestudio Raspberry Pico IO Shield

### (1) Overview

The Keyestudio Raspberry Pico IO shield is designed for Raspberry Pi Pico. No soldering required. To make the connection easier, the interfaces on the shield have silkscreen labels. The silkscreen labels of the 3pin interface generally are G, V, S. On the shield, G represents GND, V represents the VCC interface (3.3V), and S represents digital ports or analog ports. The pitch of the pin header on the shield is 2.54 mm. The sequence of the pin header is the same as the Pico board's when wiring. The shield also comes with a reset button, a PWR power indicator and four holes.

The shield offers a variety of communication interfaces including I2C, UART, SPI, analog IO and digital IO, and provides an interface of power supply ranging from 6.5V to 12V.

## Specifications:

Output current:  $\leq 500\text{mA}$

DC input voltage: 6.5 - 12V

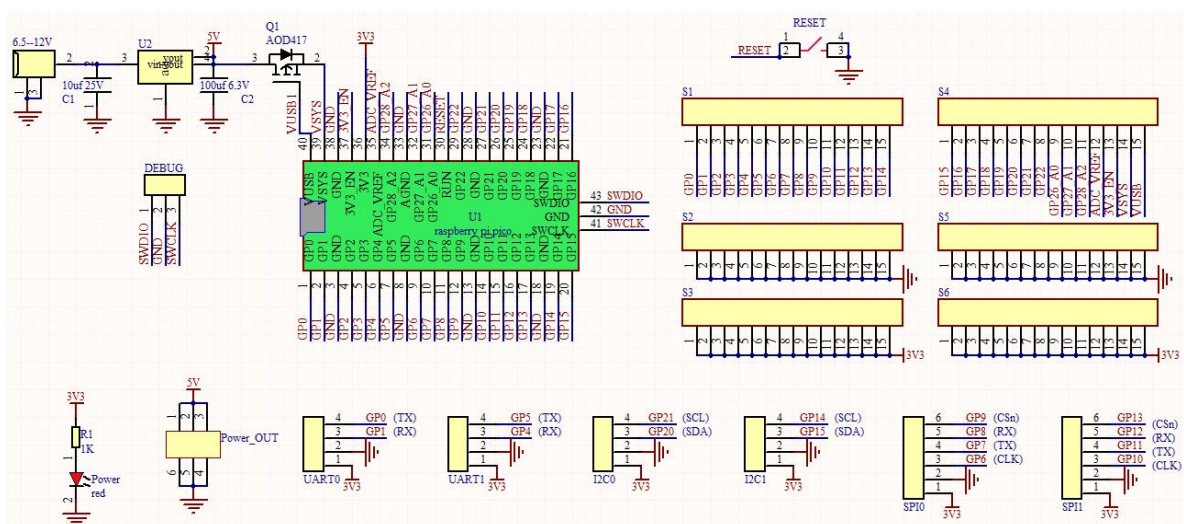
Output voltage: DC 3.3V/5V

Ambient temperature(recommended): -10°C ~ 50°C

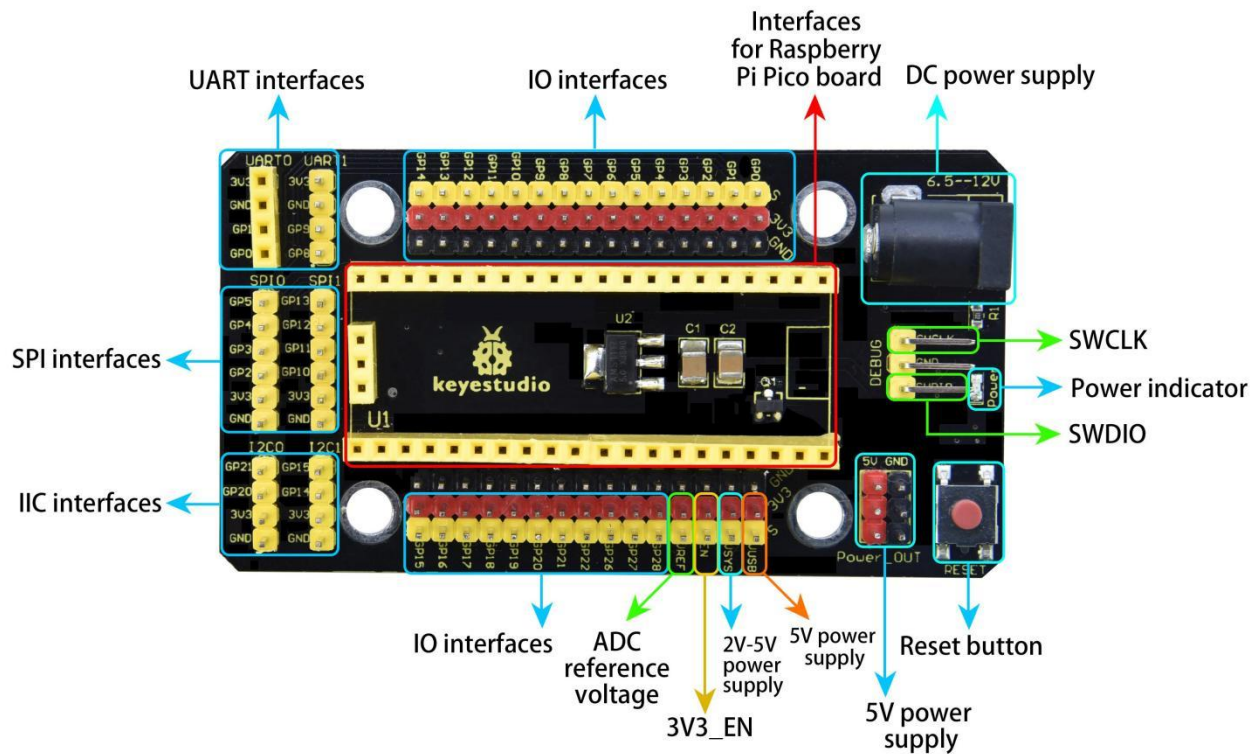
Dimensions: 45.339MM \*83.617MM

Pin pitch: 2.54mm

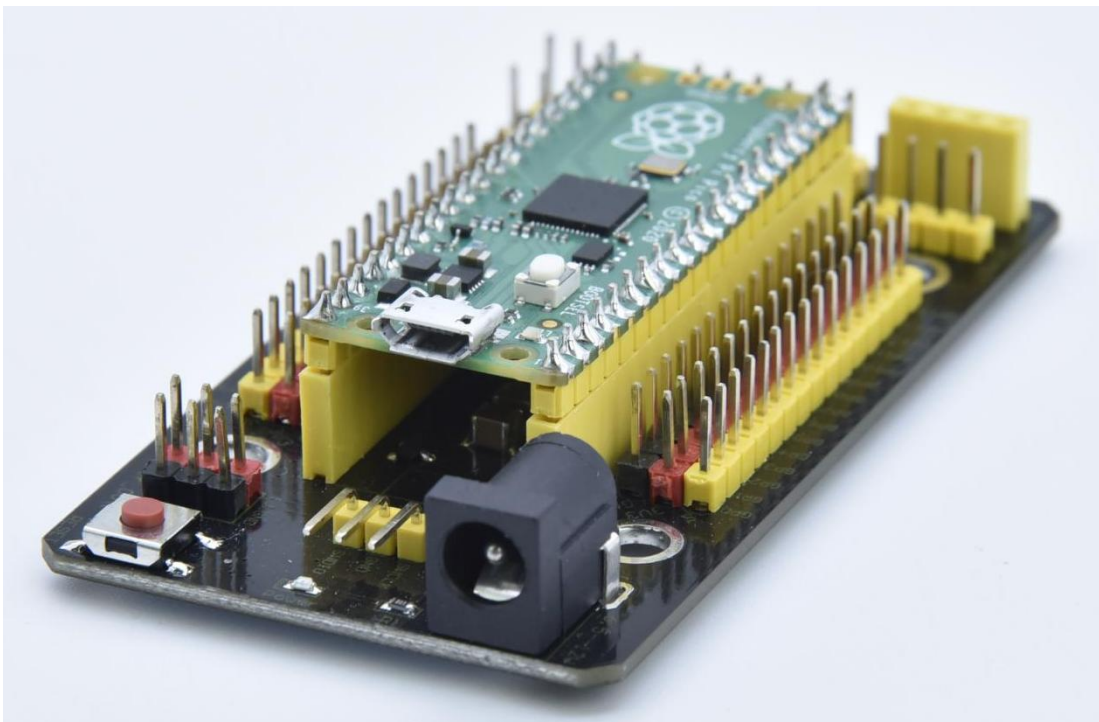
## Schematic diagram



## Pinout



As shown below, stack the Raspberry Pi Pico board onto the Raspberry Pi Pico shield.





## 4. Projects

There are 24 sensors and modules in this kit. Next, we will analyze and introduce how they work step by step. Interface sensors with the Raspberry Pi Pico board and Pico shield, run test codes then observe experimental phenomenon.

**Note: please wire up components according to the given connection diagrams.**

### Project 1: Lighting up LED



#### Overview

In this project, we will make an experiment to light up the white LED module. The high and low levels can be controlled by programming, then





the state of the LED can be controlled.

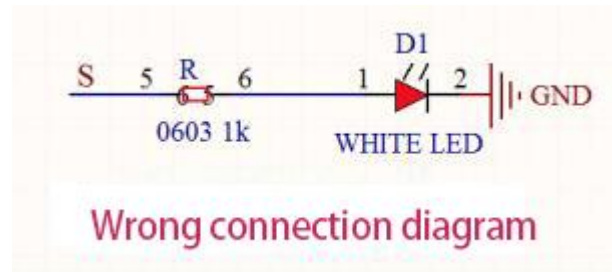
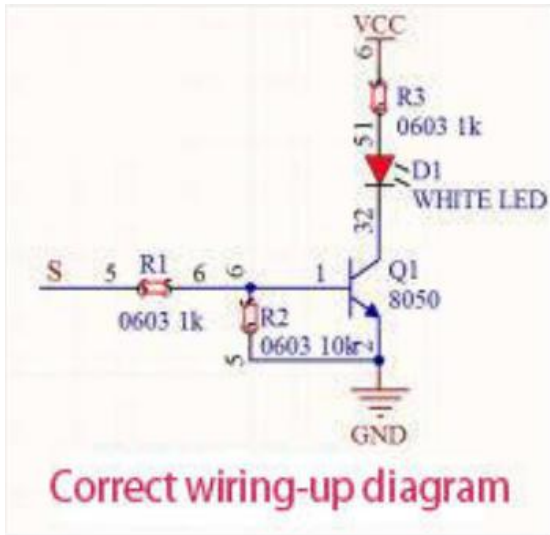
## Working Principle

The two circuit diagrams are given. The left one is wrong wiring-up diagram. Why? Theoretically, when the S terminal outputs high levels, LED will receive the voltage and light up.


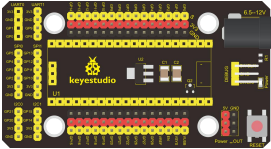
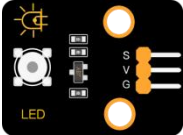


Due to limitation of IO ports of Pico board, weak current can't make LED brighten.

The right one is correct wiring-up diagram. GND and VCC are powered up. When the S terminal is a high level, the triode Q1 will be connected and LED will light up(note: current passes through LED and R3 to reach GND by VCC not IO ports). Conversely, when the S terminal is a low level, the triode Q1 will be disconnected and LED will go off.

The triode Q1 is equal to a switch and R1 and R3 stand for limited resistors which can curb the size of current to prevent from burning out components

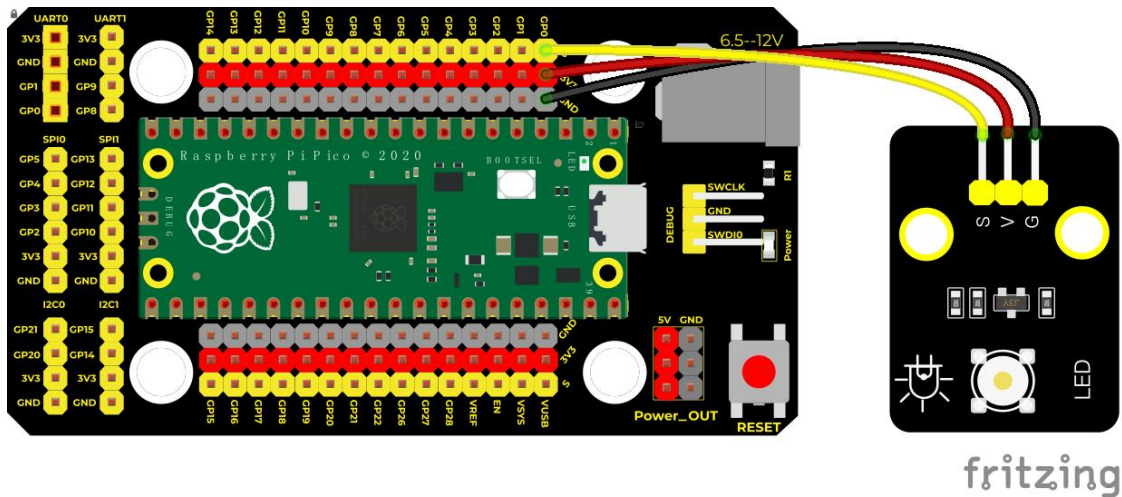


## Components

				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio White LED Module*1	3P Dupont Wire*1	Micro USB Cable*1

## Wiring Diagram





## Test Code:

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 1.1

\* turn on led

\* <http://www.keyestudio.com>

'''

from machine import Pin

led = Pin(0, Pin.OUT)# create led, connect LED to pin 0, and set pin0 to OUTPUT

led.value(1)# high levels

## Code 2:

'''

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 1.2

\* Blink

\* <http://www.keyestudio.com>

'''

from machine import Pin

import time

led = Pin(0, Pin.OUT)# create led, connect LED to pin 0, and set pin0 to OUTPUT

while True:

    led.value(1)# led lights up

    time.sleep(1)# wait for 1s

    led.value(0)# led goes off



```
time.sleep(1)# wait for 1s
```

## Code Explanation

**Machine** module is indispensable, we will use **import machine** or **from machine import...** to program pico with microPython.

`time.sleep()` function is used to set delayed time, as **`time.sleep(0.01)`**, which means, the delayed time is 10ms.

**1. `led = Pin(0, Pin.OUT)`**, created a pin example and we name **led**.

**0** is indicative of connected pin GP0, **`Pin.OUT`** represents output mode, can use **`.value()`** to output high levels (3.3V)**`led.value(1)`** or low levels (0V)**`led.value(0)`**.

**`import machine`** is used to import modules. When creating pins examples, it will change into **`led = machine.Pin(0, machine.Pin.OUT)`**

**2. `while True` is loop function,**

It means that sentences under this function will loop unless **True** changes into **False**. For the function **`while`**, **`led.value(1)`**, outputs high levels to the pin 0; then LED lights up. Then the delayed function **`time.sleep(1)`** will wait for 1s. When **`led.value(0)`** output low levels to the pin 0, the LED will go off,

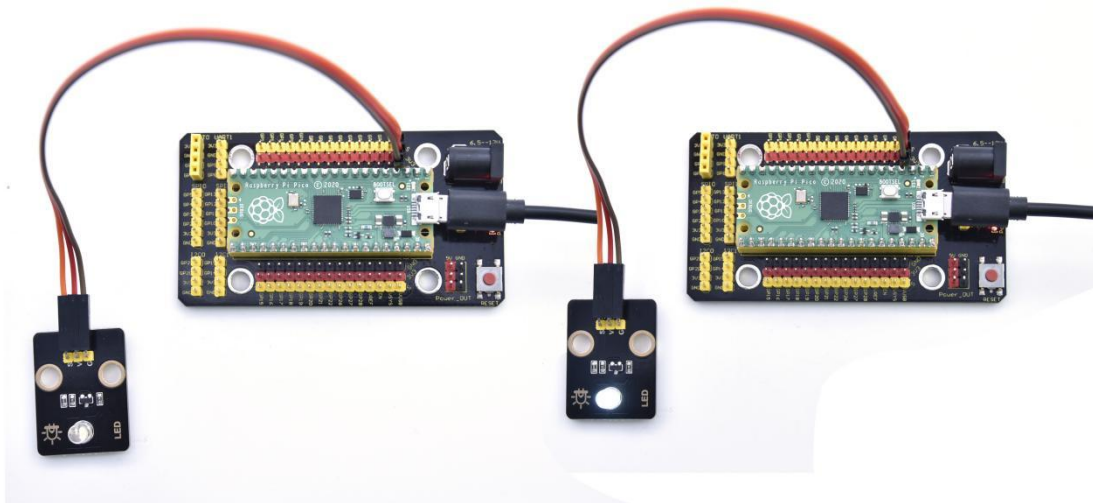


and the function **time.sleep(1)** will wait for 1s, cyclically, and LED will flash.

## Test Result

Code 1: upload the code and power on, the white LED lights up

Code 2: upload the code and power on, the white LED flashes with an interval of 1s.





## Project 2: Traffic Light Module



### Overview

In this lesson, we will learn how to control multiple LED lights and simulate the operation of traffic lights.

Traffic lights are signal devices positioned at road intersections, pedestrian crossings, and other locations to control flows of traffic.

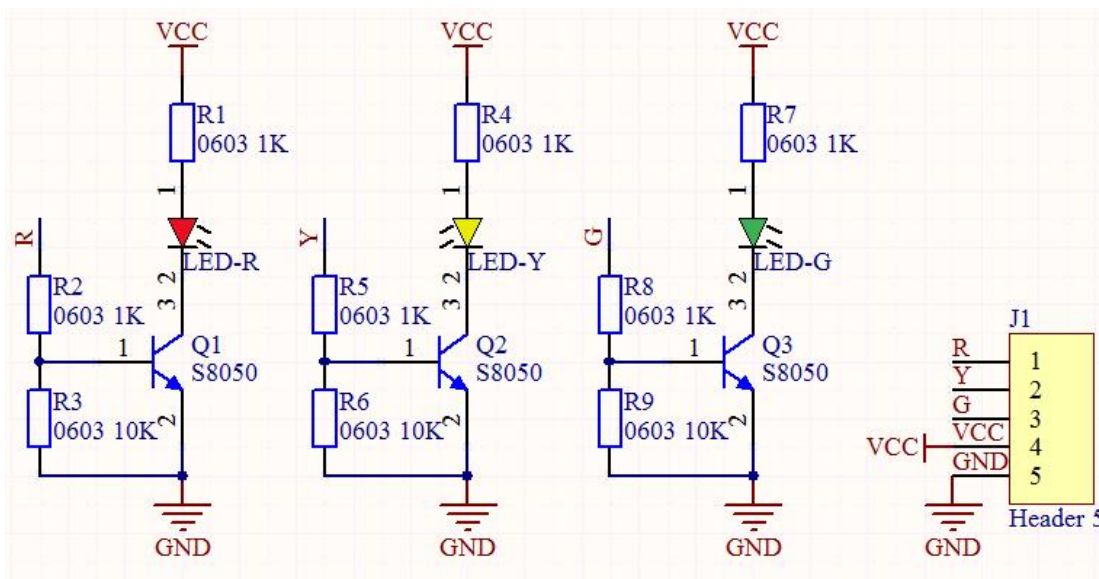
In this kit, we will use the traffic light module to simulate the traffic light.

### Working Principle

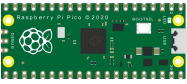
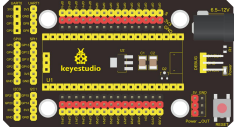



In previous lesson, we already know how to control an LED. In this part, we only need to control three separated LEDs. Output high levels to the signal



R(3.3V), then the red LED will be on.

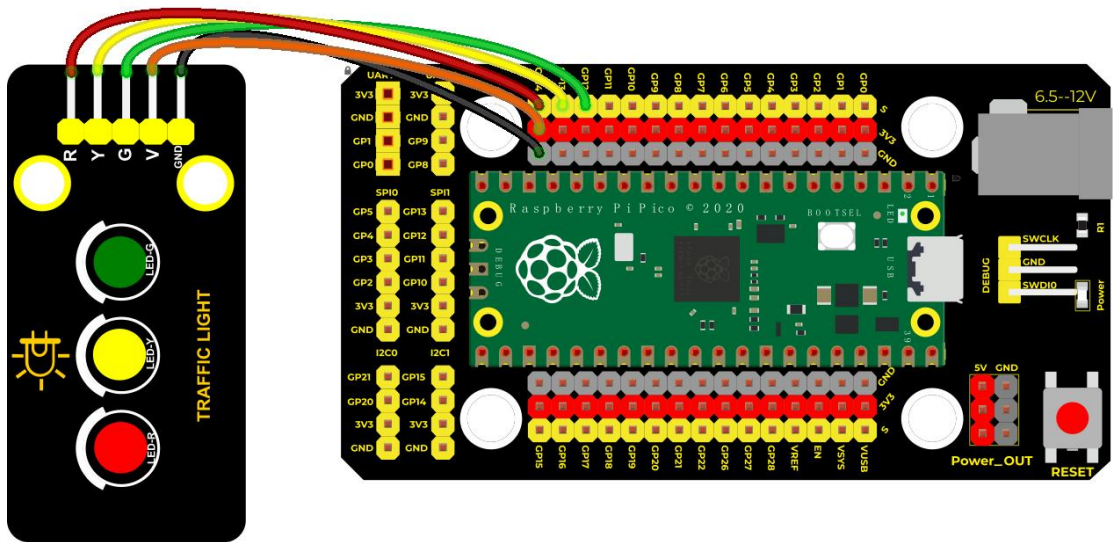


## Components

				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio DIY Traffic Lights Module*1	5P Dupont Wire *1	Micro USB Cable*1



## Wiring Diagram



fritzing

## Test Code

```
'''
* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico
* lesson 2
* Traffic_Light
* http://www.keyestudio.com
'''
```

```
import machine
import time
```

```
led_red = machine.Pin(14, machine.Pin.OUT)
led_amber = machine.Pin(13, machine.Pin.OUT)
led_green = machine.Pin(12, machine.Pin.OUT)
```

```
while True:
    led_green.value(1) # the green light is on for 5s
    time.sleep(5)# after 5s
    led_green.value(0)# the green LED will go off
    for i in range(3):# the yellow light is on for 3s
        led_amber.value(1)
        time.sleep(0.5)
        led_amber.value(0)
        time.sleep(0.5)
```



```
led_red.value(1) # the red LED light up for 5s  
time.sleep(5)  
led_red.value(0)
```

## Code Explanation

Create pins, set pins mode and delayed functions.

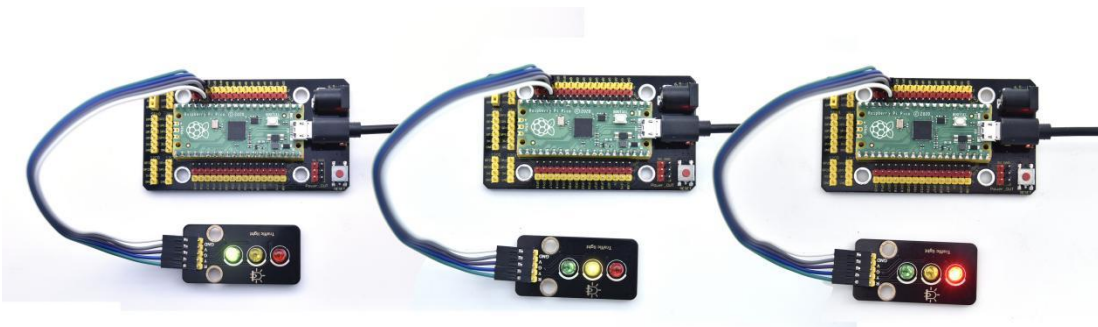
We use the **for** loop

The simplest form is **for i in range()**,

In the code, we used `range(3)`, which means the variable `i` starts from 0, increase 1 for each time

## Test Result

Upload the code, the green LED will be on for 5s then off, the yellow LED will flash for 3s then go off and the red one will be on 5s then off.







## Project 3: Button Sensor



### Overview

In this kit, there is a Keyestudio single-channel button module, which mainly uses a tact switch and comes with a yellow button cap.

In previous lessons, we learned how to make the pins of our single-chip microcomputer output a high level or low level. In this experiment, we will read the high level (3.3V) and low level (0V).

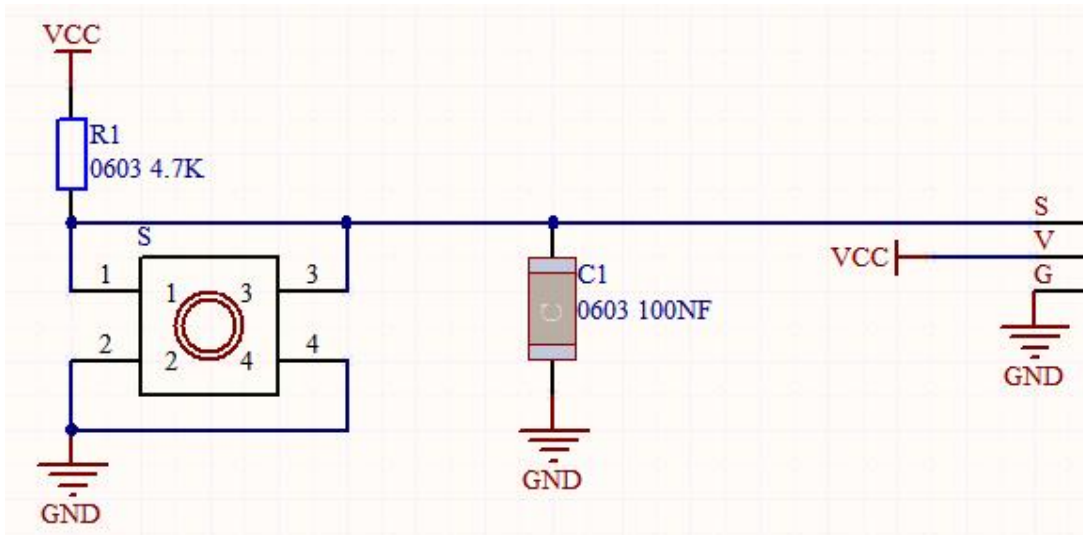
We can determine whether the button on the sensor is pressed by reading the high and low level of the S terminal on the sensor.

### Working Principle

The button module has four pins. The pin 1 is connected to the pin 3 and the pin 2 is linked with the pin 4. When the button is not pressed, they are


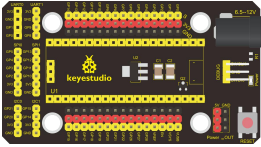
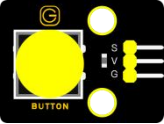




disconnected. Yet, when the button is pressed, they are connected. If the button is released, the signal end is high level.

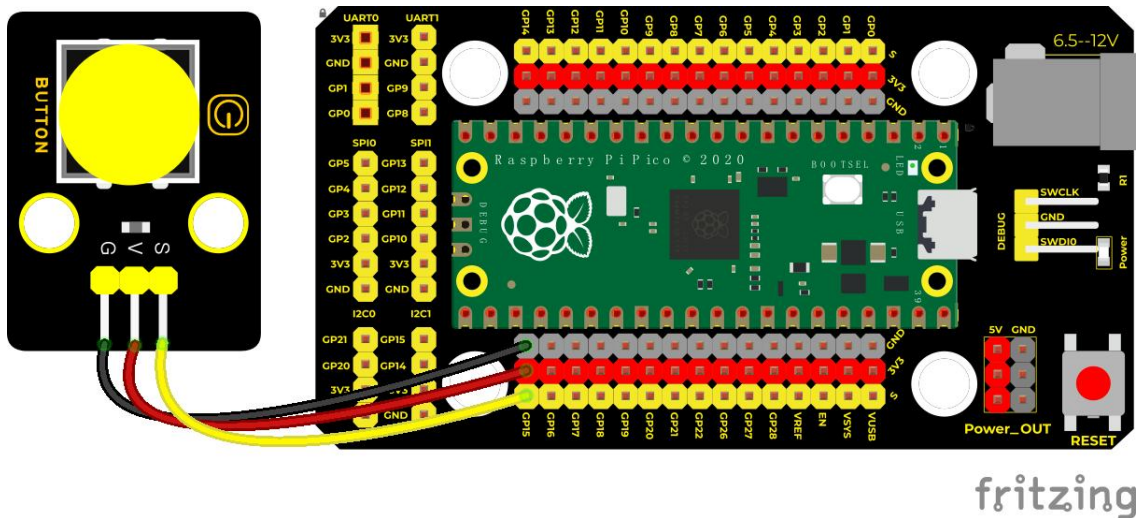




## Components

				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio Button Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

## Wiring Diagram



## Test Code:

- \* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico
- \* lesson 3
- \* button



```
* http://www.keyestudio.com
```

```
...
```

```
from machine import Pin
```

```
import time
```

```
button = Pin(15, Pin.IN, Pin.PULL_UP)
```

```
while True:
```

```
    if button.value() == 0:
```

```
        print("You pressed the button!")    #press to print the  
information
```

```
    else:
```

```
        print("You loosen the button!")
```

```
        time.sleep(0.1) #delay in 0.1s
```

## Code Explanation

**button = Pin(15, Pin.IN, Pin.PULL\_UP)**, we define the pin of the button as GP15 and set to PULL-UP mode

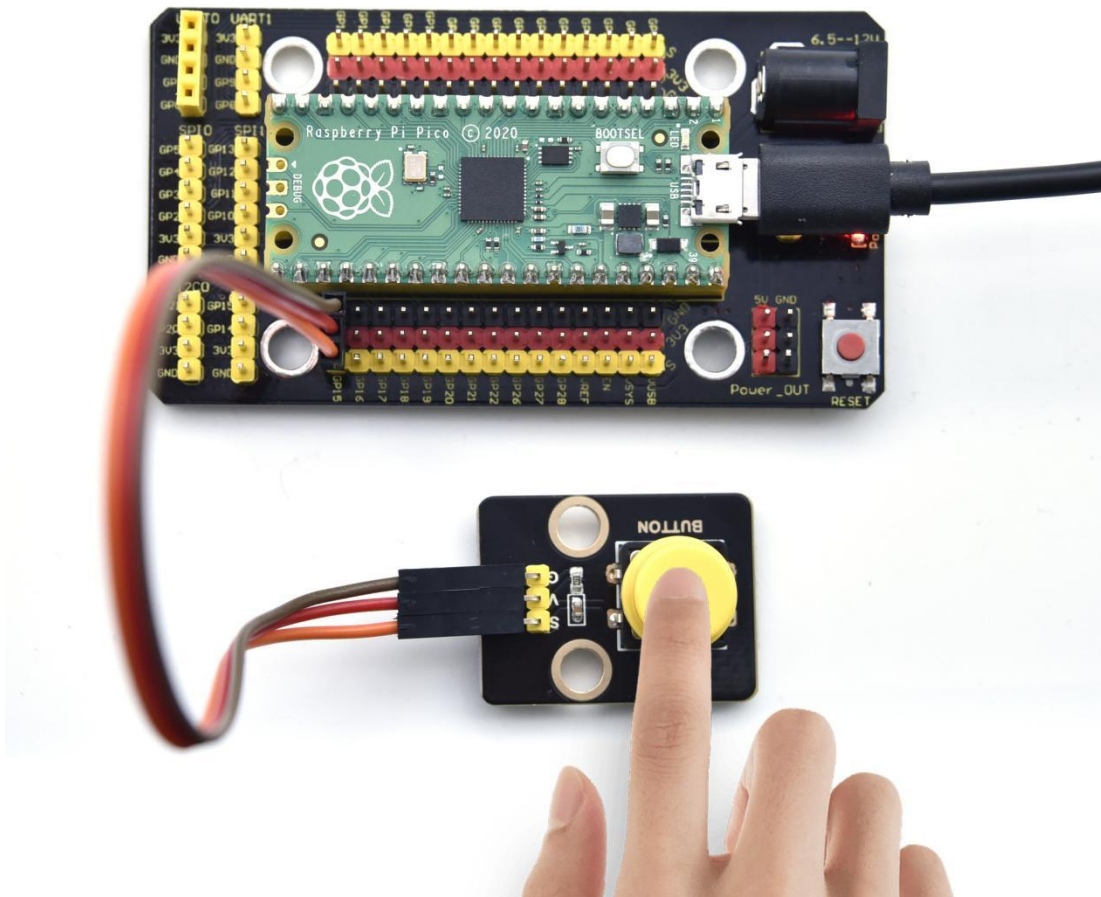
We can use **button = Pin(15, Pin.IN)** to set **INPUT mode**, at this time, the pins are in high resistance state.



1. **button.value()**, read levels of buttons. Function returns High or Low
2. **if..else.. sentence**, when the logic judge is TRUE, the code under the if will be activated; otherwise, the code under the else will be activated.
3. When pico detects the button pressed, the signal end is low level (GP 15 is low level). **button.value() is 0**. If pico detects the button unpressed, **button.value()** is 1 and else sentence will be activated.

## Test Result

Upload the code, and look at the Shell page. When the button is pressed, "You pressed the button!" will be displayed; if released, "You loosen the button!" will appear, as shown below;



Shell X

```
You loosen the button.  
You loosen the button!  
You loosen the button!  
You loosen the button!  
You loosen the button!  
You loosen the button!  
You loosen the button!  
You loosen the button!  
You pressed the button!  
You pressed the button!  
You pressed the button!
```



## Project 4: Obstacle Avoidance Sensor



### Overview

In this kit, there is a Keyestudio obstacle avoidance sensor, which mainly uses an infrared emitting and a receiving tube. In the experiment, we will determine whether there is an obstacle by reading the high and low level of the S terminal on the sensor.

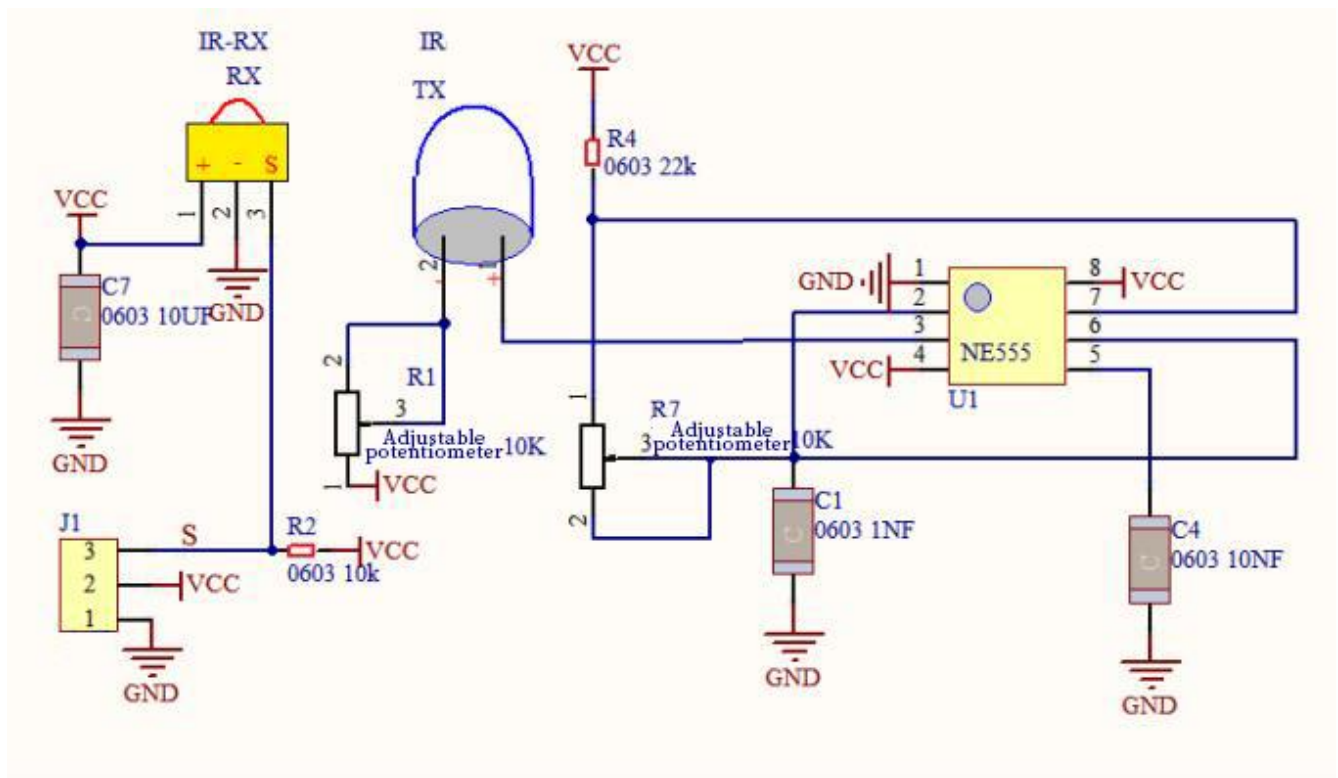
### Working Principle

NE555 circuit provides IR signals with frequency to the emitter TX, then the IR signals will fade with the increase of transmission distance. If encountering the obstacle, it will be reflected back.



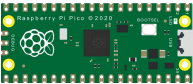
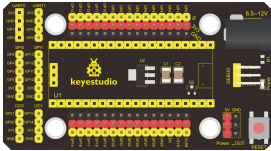
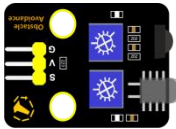




When the receiver RX meets the weak signals reflected back, the receiving pin will output high levels, which indicates the obstacle is far away. On the contrary, if the reflected signals are stronger, low levels will be output, which represents the obstacle is close. There are two potentiometers on the module, and one is for adjusting emission power, another one is for receiving frequency.

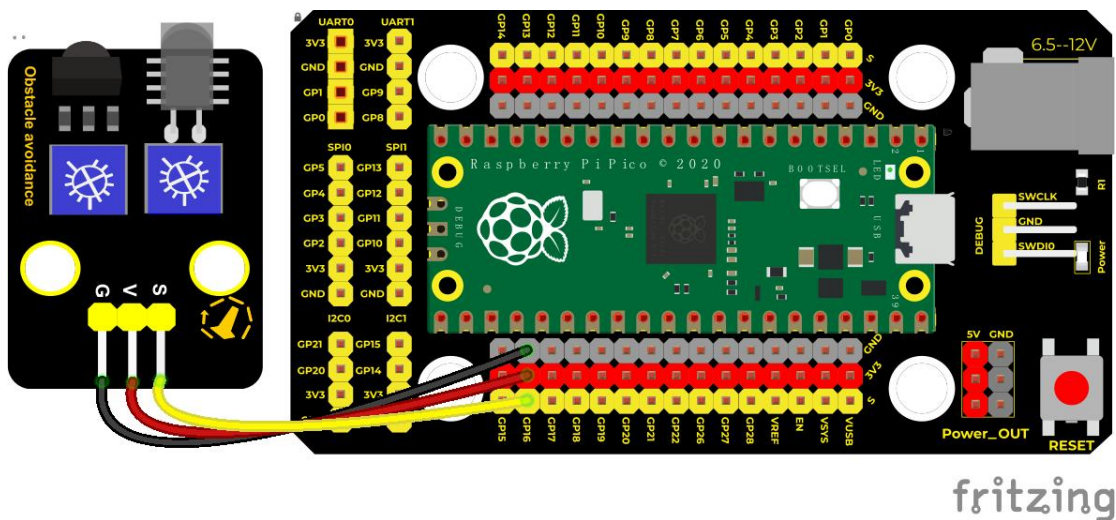




## Components

				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio Obstacle Avoidance Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

## Wiring Diagram



## Test Code

...

\* [Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico](#)

\* [lesson 4](#)



**\* Infrared obstacle avoidance sensor**

**\* <http://www.keyestudio.com>**

**...**

**from machine import Pin**

**import time**

**sensor = Pin(16, Pin.IN)**

**while True:**

**if sensor.value() == 0:**

**print("There are obstacles")**

**else:**

**print("All going well")**

**time.sleep(0.1)**

### **Note:**

Upload the test code and wire up according to the connection diagram. After powering on, we start to adjust the two potentiometers to sense distance.

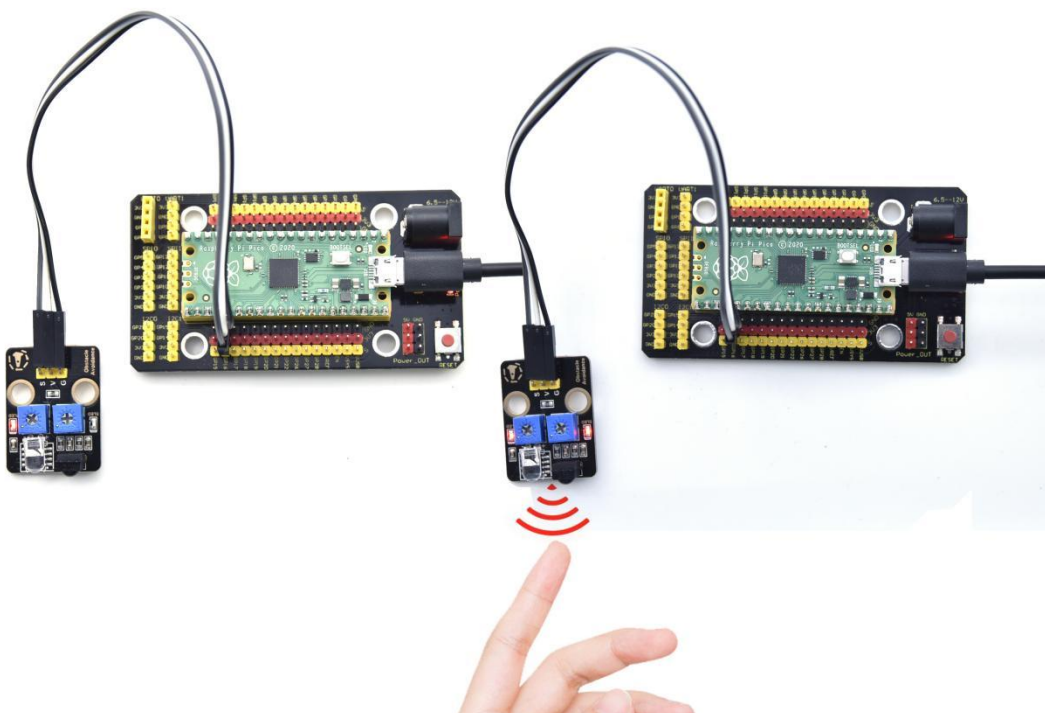
1. Adjust the potentiometer transmitting power. Make the P LED at the critical point of ON and OFF states.



2. Adjust the potentiometer receiving frequency. Rotate it clockwise, the frequency will increase. Make the S LED at the critical point of ON and OFF states, then the 38KHz square wave can be produced.

## Test Result

Upload the code, when the sensor detects the obstacle, the Shell page will show "There are obstacles" ; if the obstacle is not detected, "All going well" will be shown.





```
Shell X
All going well
All going well
All going well
All going well
All going well
All going well
All going well
All going well
There are obstacles
There are obstacles
There are obstacles
There are obstacles
There are obstacles
```

## Project 5: Tilt Module



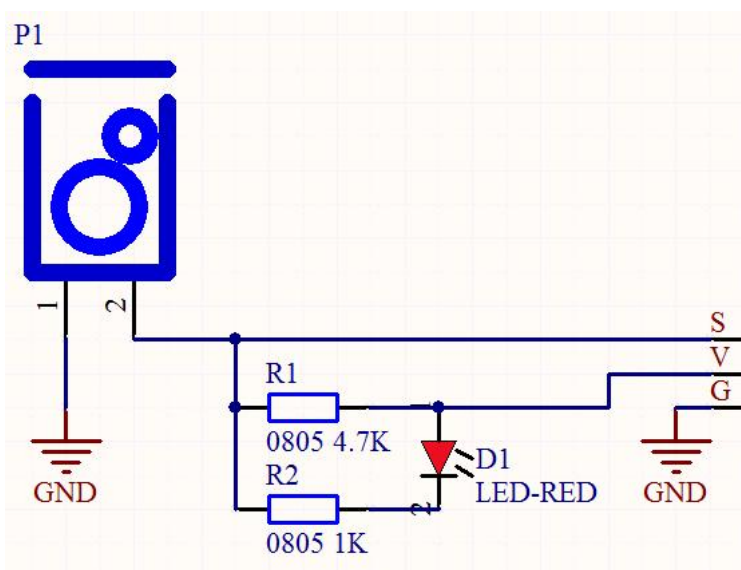
### Overview

In this kit, there is a Keyestudio tilt sensor. The tilt switch can output signals of different levels according to whether the module is tilted. There is a ball inside. When the switch is higher than the horizontal level, the switch is



turned on, and when it is lower than the horizontal level, the switch is turned off. This tilt module can be used for tilt detection, alarm or other detection.

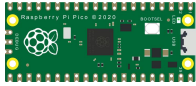
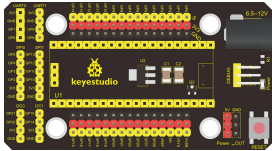
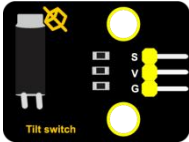


## Working Principle



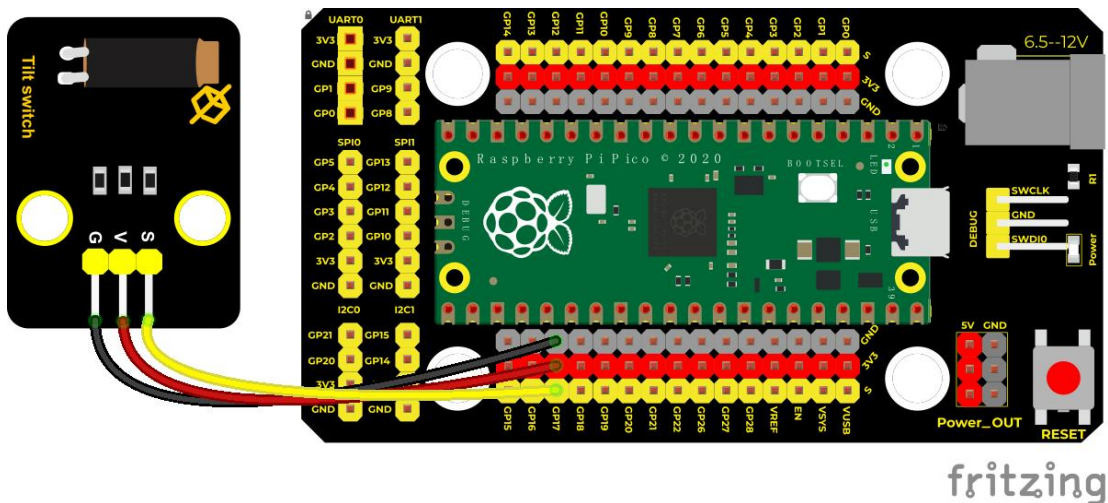
The working principle is pretty simple. When pin 1 and 2 of the ball switch P1 are connected, the signal S is low level and the red LED will light up; when they are disconnected, the pin will be pulled up by the 4.7K R1 and make S a high level, then LED will be off.

## Components



				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio Tilt Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

## Wiring Diagram



## Test Code'''

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 5

\* Tilt switch

\* <http://www.keyestudio.com>

'''





```
from machine import Pin
```

```
import time
```

```
TiltSensor = Pin(17, Pin.IN)
```

```
while True:
```

```
    value = TiltSensor.value()
```

```
    print(value, end = " ")
```

```
    if value == 0:
```

```
        print("The switch is turned on")
```

```
    else:
```

```
        print("The switch is turned off")
```

```
    time.sleep(0.1)
```

## Code Explanation

Code explanation is as same as the project 3.

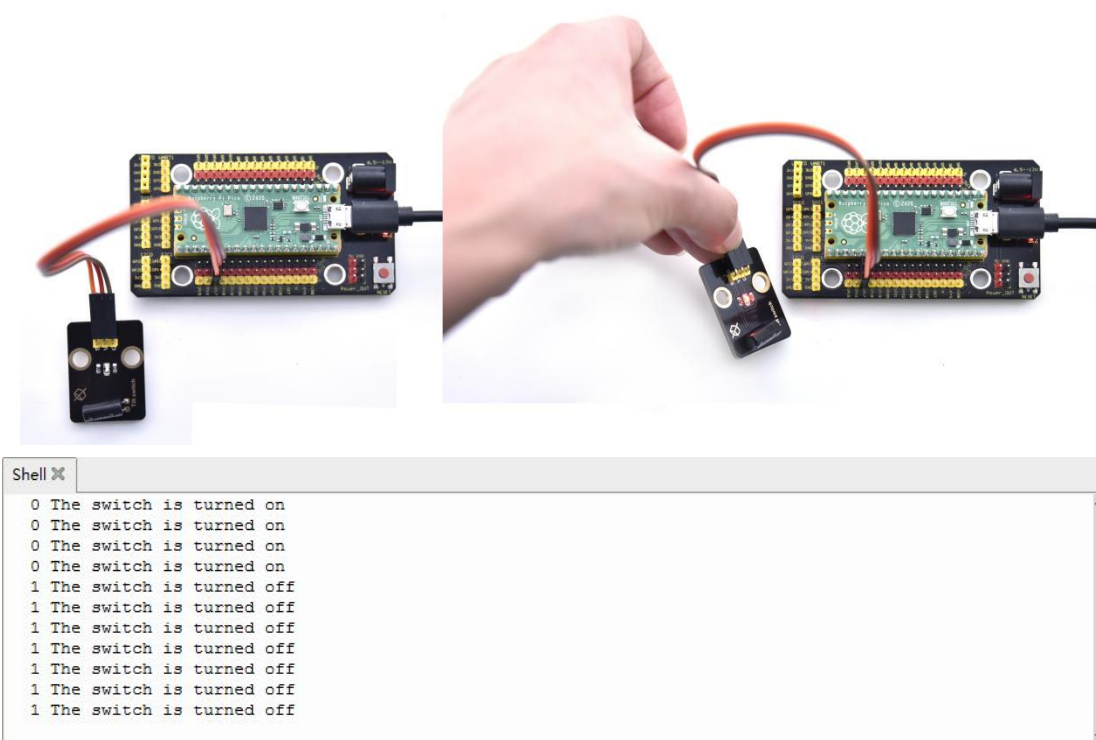
## Test Result

Upload the code successfully, and observe the Shell.

Make the tilt module incline to one side, the red LED on the module will be off and the Shell page will display "1 The switch is turned off" ; by contrast,

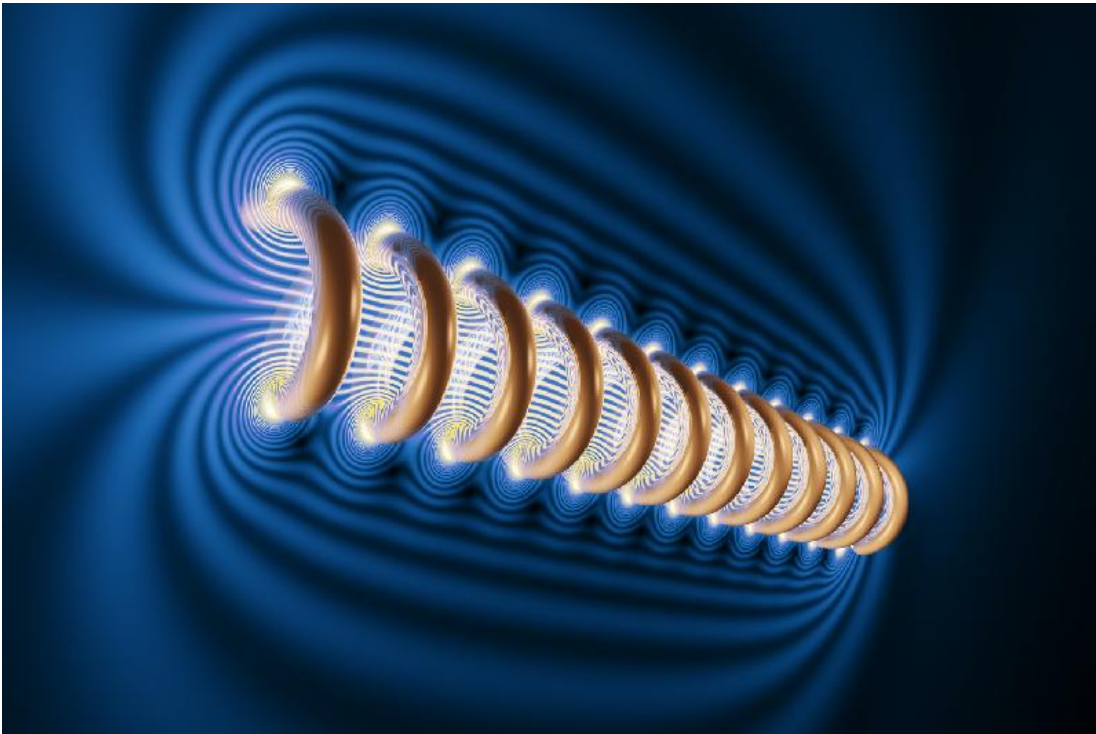


if you make it incline the other side, the red LED will light up and “0 The switch is turned on” will be shown.





## Project 6: Reed Switch Module



### Overview

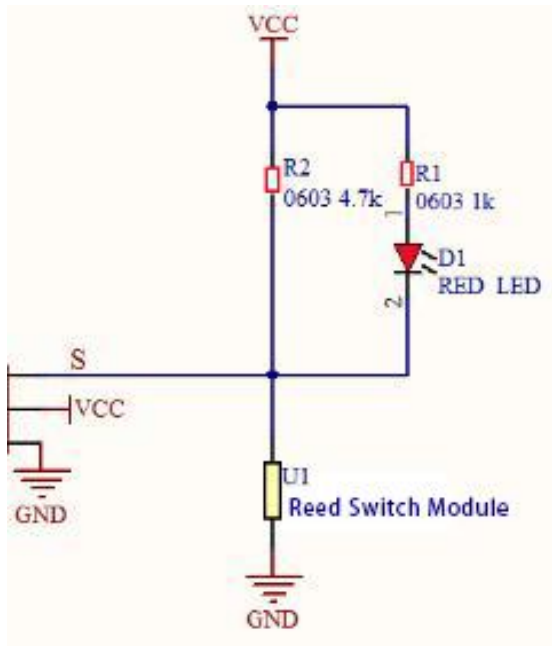
In this kit, there is a Keyestudio reed switch module, which mainly uses a MKA10110 green reed component.

The reed switch is the abbreviation of the dry reed switch. It is a passive electronic switch element with contacts.

It has the advantages of simple structure, small size and easy control.

Its shell is a sealed glass tube with two iron elastic reed electric plates.

In the experiment, we will determine whether there is a magnetic field near the module by reading the high and low level of the S terminal on the module; and, we display the test result in the shell.

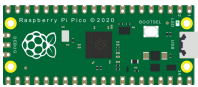
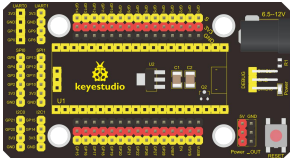
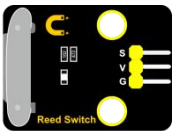




## Working Principle

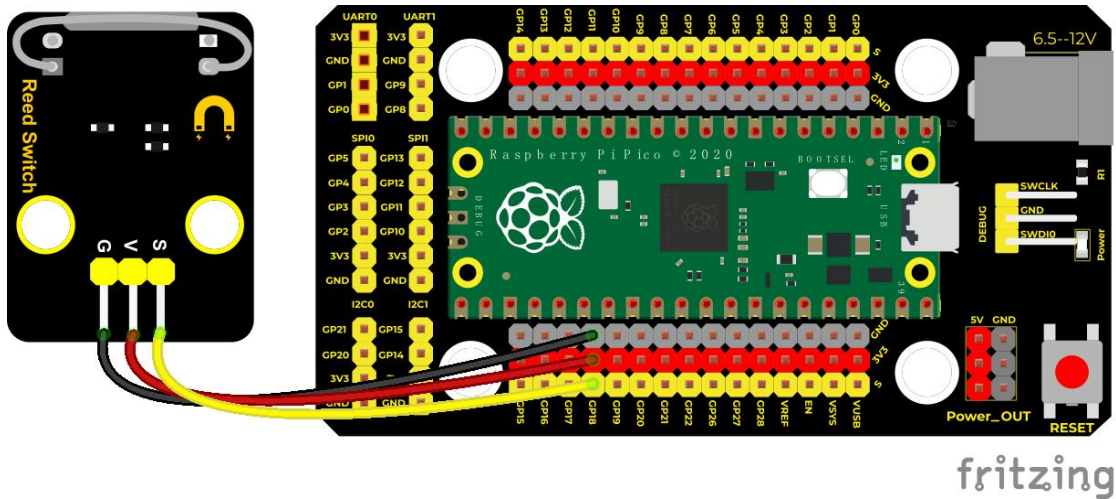
Reed switch is an abbreviation of the dry reed contacts a passive electronic switching elements, and has the advantages of simple structure, small size and ease of control, its shell is a sealed glass tube, the tubes are installed two iron elastic reed plate, but also filling called rhodium metal inert gas. In peacetime, the glass tube in the two reeds made of special materials are separated. When a magnetic substance close to the glass tube, in the role of the magnetic field lines, the pipe within the two reeds are magnetized to attract each other in contact, the reed will suck together, so that the junction point of the connected circuit communication. After the disappearance of the outer magnetic reed



because of their flexibility and separate, the line is disconnected. Therefore, as a use of the magnetic field signals to control the line switching device, reed tube can be used as a sensor for counting the number, spacing, etc., and also are widely used in a variety of communication devices

<b>Components</b> 				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio Reed Switch Module*1	3P Dupont Wire*1	Micro USB Cable*1

## Wiring Diagram



## Test Code

...

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 6

\* Reed Switch

\* <http://www.keyestudio.com>

...

```
from machine import Pin
```

```
import time
```

```
ReedSensor = Pin(18, Pin.IN)
```

```
while True:
```

```
    value = ReedSensor.value()
```

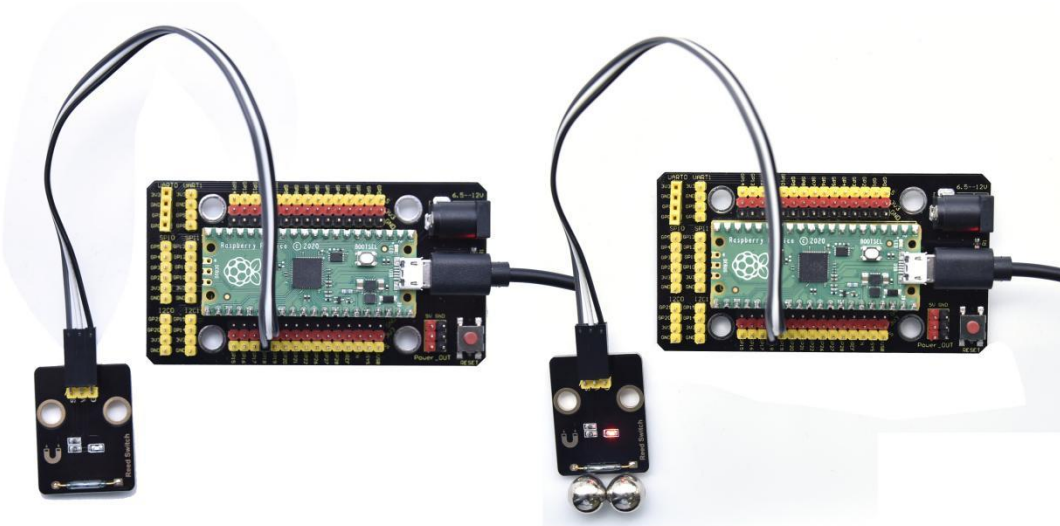
```
    print(value, end = " ")
```



```
if value == 0:  
    print("A magnetic field")  
  
else:  
    print("There is no magnetic field")  
  
time.sleep(0.1)
```

## Test Result

Upload the code. When the sensor detects a magnetic field, val is 0 and the red LED of the module lights up, "0 A magnetic field" will be displayed; when no magnetic field is detected, val is 1, and the LED on the module goes out, "1 There is no magnetic field" will be shown, as shown below.







```
Shell X
1 There is no magnetic field
1 There is no magnetic field
1 There is no magnetic field
1 There is no magnetic field
1 There is no magnetic field
0 A magnetic field
0 A magnetic field
0 A magnetic field
0 A magnetic field
0 A magnetic field
0 A magnetic field
0 A magnetic field
```

## Project 7: PIR Motion Sensor



### Overview

In this kit, there is a Keyestudio PIR motion sensor, which mainly uses an RE200B-P sensor elements. It is a human body pyroelectric motion sensor based on pyroelectric effect, which can detect infrared rays emitted by humans or animals, and the Fresnel lens can make the sensor's detection range farther and wider.

In the experiment, we determine if there is someone moving nearby by



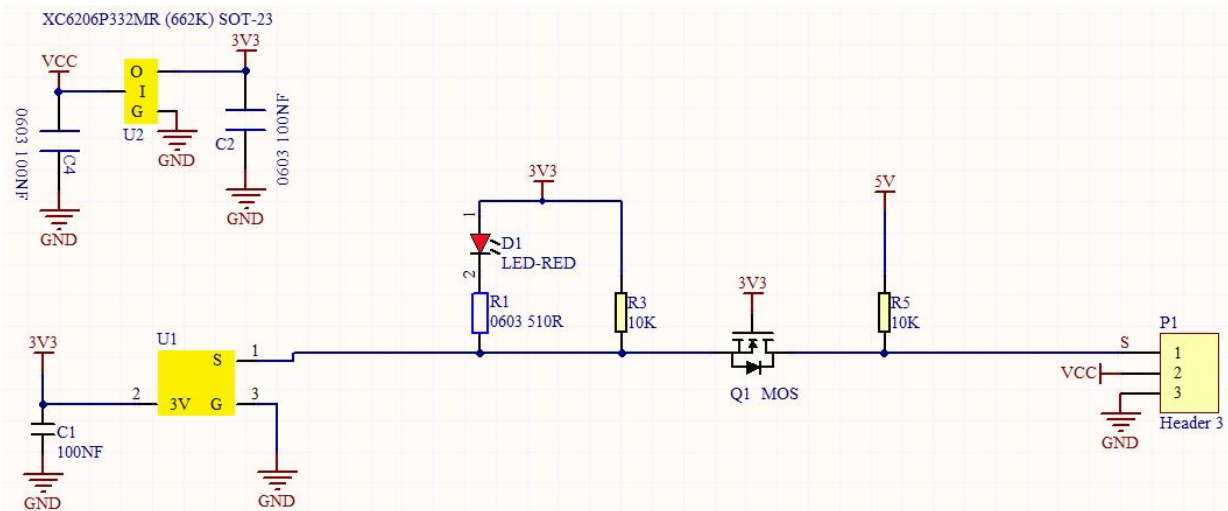
reading the high and low levels of the S terminal on the module. The detected results will be displayed on the Shell.

## **Working Principle**

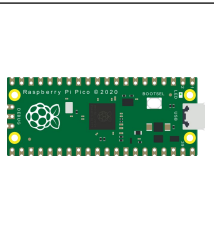
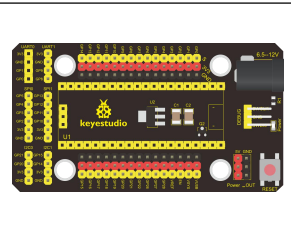
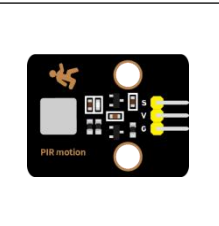


The upper left part is voltage conversion(VCC to 3.3V). The working voltage of sensors we use is 3.3V, therefore we can't use 5V directly. The voltage conversion circuit is needed.

When no person is detected or no infrared signal is received, and pin 1 of the sensor outputs low level. At this time, the LED on the module will light up and the MOS tube Q1 will be connected and the signal terminal S will detect Low levels.

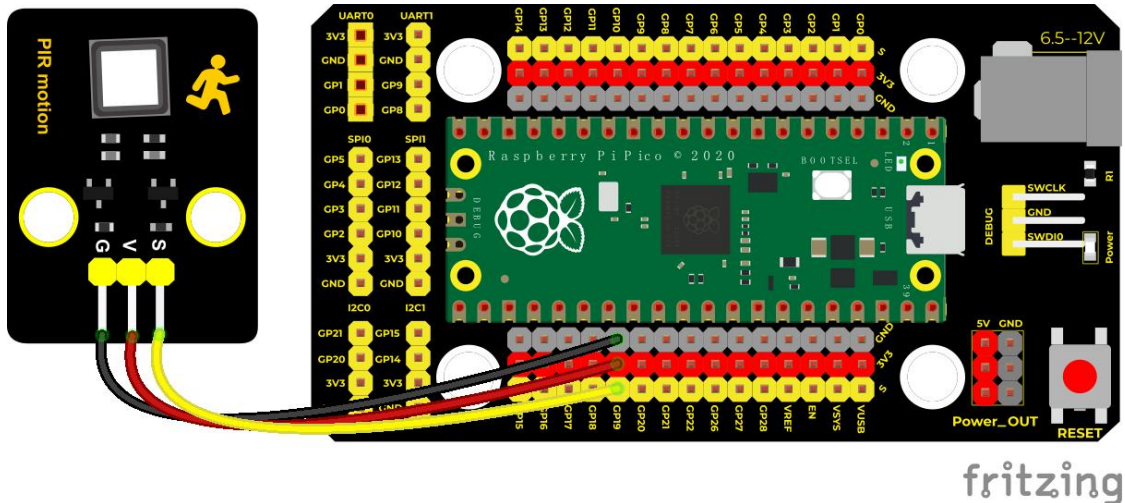
When one is detected or an infrared signal is received, and pin 1 of the sensor outputs a high level. Then LED on the module will go off, the MOS tube Q1 is disconnected and the signal terminal S will detect high levels.



## Components

				
<p>Raspberry Pi Pico Board*1</p>	<p>Raspberry Pi Pico Shield*1</p>	<p>Keyestudio PIR Motion Sensor*1</p>	<p>3P Dupont Wire*1</p>	<p>Micro USB Cable*1</p>

## Wiring Diagram



## Test Code

...

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 7

\* PIR motion

\* <http://www.keyestudio.com>

...

```
from machine import Pin
```

```
import time
```

```
PIR = Pin(19, Pin.IN)
```

```
while True:
```

```
    value = PIR.value()
```

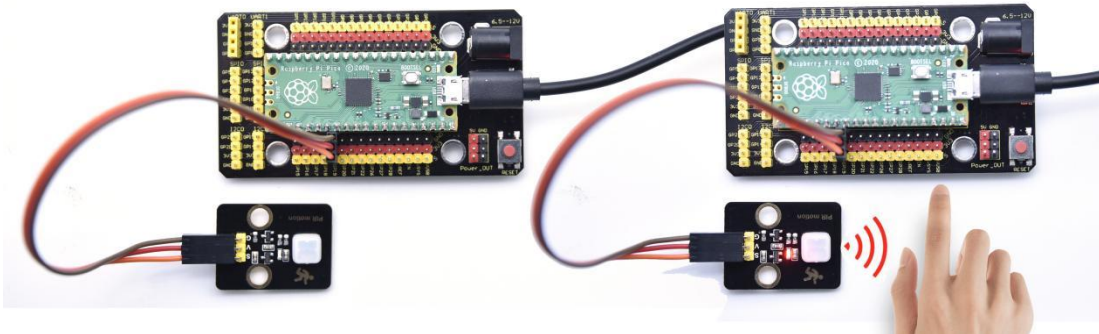
```
    print(value, end = " ")
```



```
if value == 1:  
    print("Some body is in this area!")  
  
else:  
    print("No one!")  
  
time.sleep(0.1)
```

## Test Result

Upload the code, when the sensor detects someone nearby, value is 1, the LED will go off and the Shell page will show "1 Somebody is in this area!" . In contrast, the value is 0, the LED will go up and "0 No one!" will be shown.



```
Shell x  
0 No one!  
0 No one!  
0 No one!  
0 No one!  
1 Some body is in this area!  
1 Some body is in this area!  
1 Some body is in this area!  
1 Some body is in this area!  
1 Some body is in this area!  
1 Some body is in this area!  
1 Some body is in this area!  
1 Some body is in this area!
```



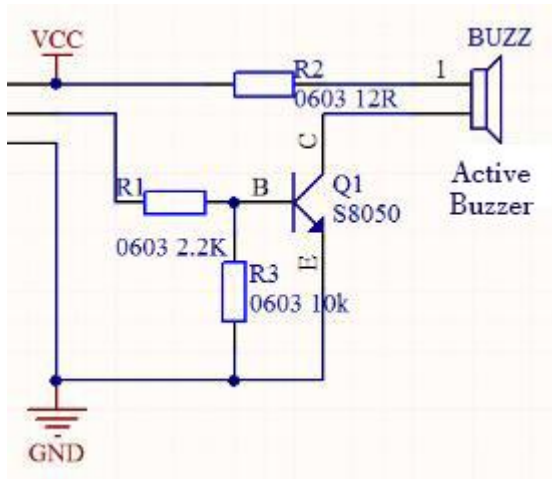
## Project 8: Active Buzzer



### Overview

In this kit, it contains an active buzzer module and a power amplifier module (the principle is equivalent to a passive buzzer). In this experiment, we control the active buzzer to emit sounds. Since it has its own oscillating circuit, the buzzer will automatically sound if given large voltage.

### Working Principle

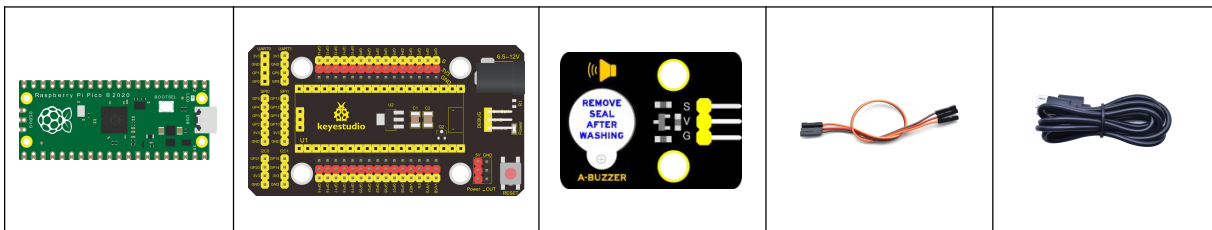


From the schematic diagram, the pin of buzzer is connected to a resistor R2 and another port is linked with a NPN triode Q1. So, if this triode Q1 is powered, the buzzer will sound.

If the base electrode of the triode connected to the R1 resistor is a high level, the triode Q1 will be connected. If the base electrode is pulled down by the resistor R3, the triode is disconnected.

When we output a high level from the IO port to the triode, the buzzer will emit sounds; if outputting low levels, the buzzer won't emit sounds.

## Components

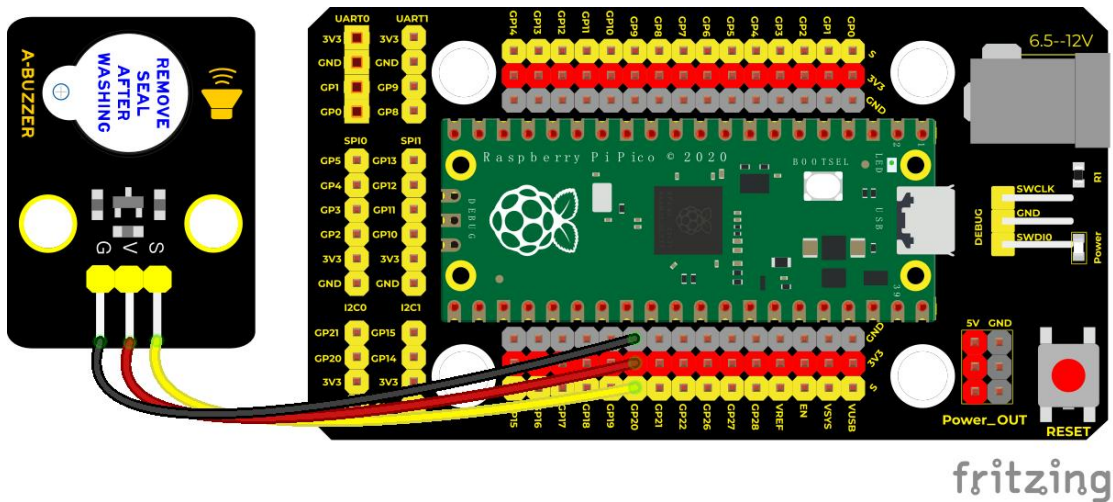






Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio Active Buzzer*1	3P Dupont Wire*1	Micro USB Cable*1
---------------------------------	-------------------------------	----------------------------------	---------------------	----------------------

## Wiring Diagram



## Test Code

...

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 8

\* Active buzzer

\* <http://www.keyestudio.com>

...

from machine import Pin

import time



```
buzzer = Pin(20, Pin.OUT)
```

```
while True:
```

```
    buzzer.value(1)
```

```
    time.sleep(1)
```

```
    buzzer.value(0)
```

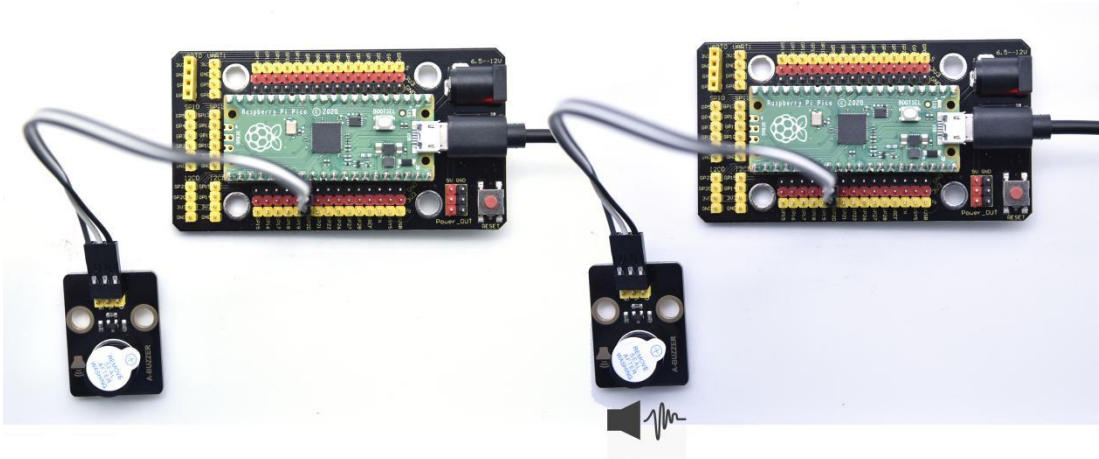
```
    time.sleep(1)
```

## **Code Explanation**

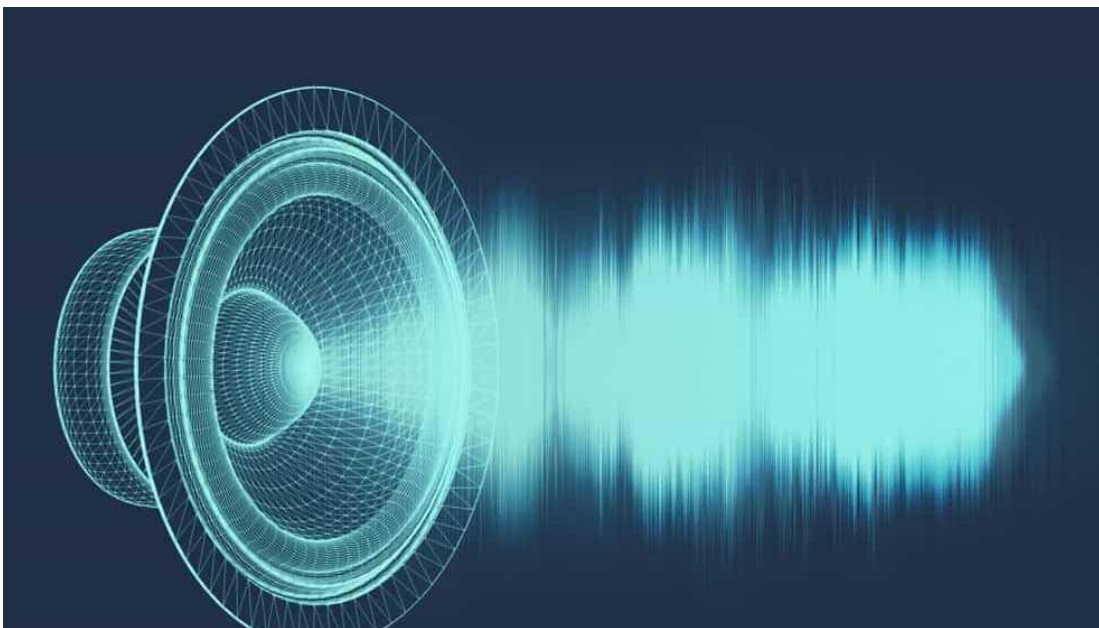
In the experiment, the pin is set to 20. When setting HIGH, the active buzzer on the module will emit sounds; when setting LOW, the buzzer won't chime.

## **Test Result**

Upload the code and power on. The buzzer chimes



## Project 9: 8002b Audio Power Amplifier



### Overview

In this kit, there is a Keyestudio 8002b audio power amplifier. The main components of this module are an adjustable potentiometer, a speaker, and an audio amplifier chip;

The main function of this module is: it can amplify the output audio signal,

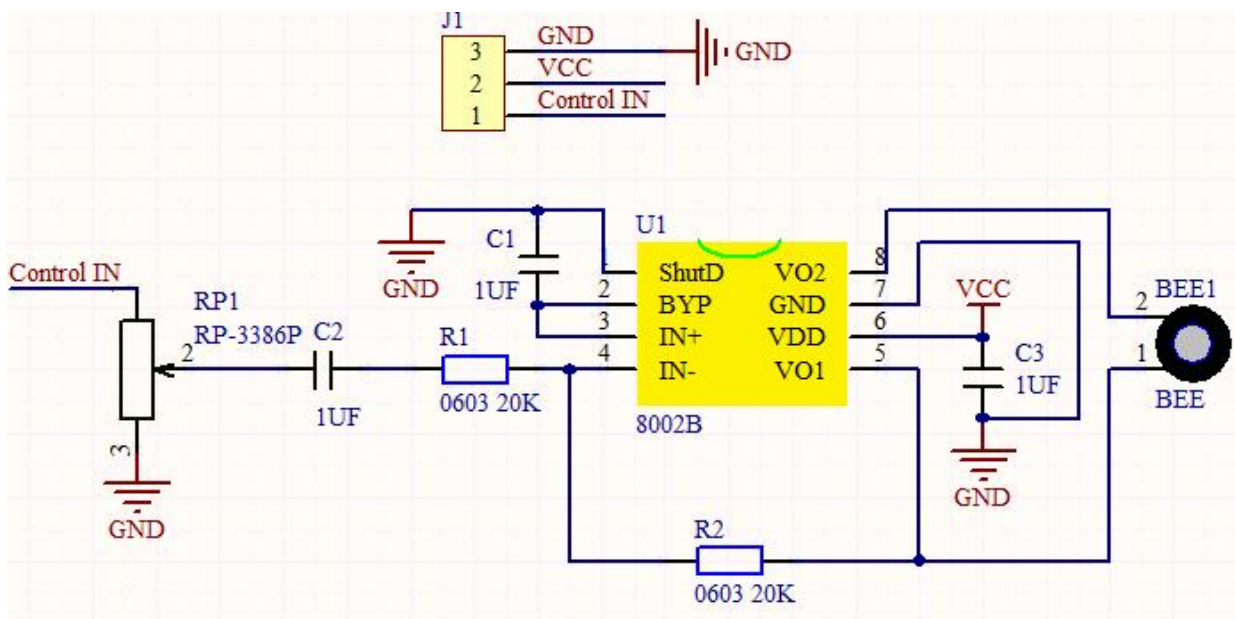


with a magnification of 8.5 times, and play sound or music through the built-in low-power speaker, as an external amplifying device for some music playing equipment.

In the experiment, we used the 8002b power amplifier speaker module to emit sounds of various frequencies.

## Working Principle

In fact, it is similar to a passive buzzer. The active buzzer has its own oscillation source. Yet, the passive buzzer does not have internal oscillation. When controlling the circuit, we need to input square waves of different frequencies to the positive pole of the component and ground the negative pole to control the buzzer to chime sounds of different frequencies.



## Wiring Diagram





## Test Code

...

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 9

\* Passive buzzer

\* <http://www.keyestudio.com>

...

```
from machine import Pin, PWM
```

```
from time import sleep
```

```
buzzer = PWM(Pin(21))
```

```
buzzer.duty_u16(1000)
```

```
buzzer.freq(523)#DO
```

```
sleep(0.5)
```

```
buzzer.freq(586)#RE
```

```
sleep(0.5)
```

```
buzzer.freq(658)#MI
```

```
sleep(0.5)
```

```
buzzer.freq(697)#FA
```

```
sleep(0.5)
```

```
buzzer.freq(783)#SO
```



```
sleep(0.5)
```

```
buzzer.freq(879)#LA
```

```
sleep(0.5)
```

```
buzzer.freq(987)#SI
```

```
sleep(0.5)
```

```
buzzer.duty_u16(0)
```

### Code Explanation

We use PWM of the machine,

**buzzer = PWM(Pin(21))** is a PWM example and the pin of the buzzer is connected to GP21

**buzzer.duty\_u16(1000)** is used to set duty cycle(1000/65535) and the larger this value, the louder the buzzer. When you set to 0, the buzzer doesn't emit sounds.

**buzzer.freq()** is frequency setting.

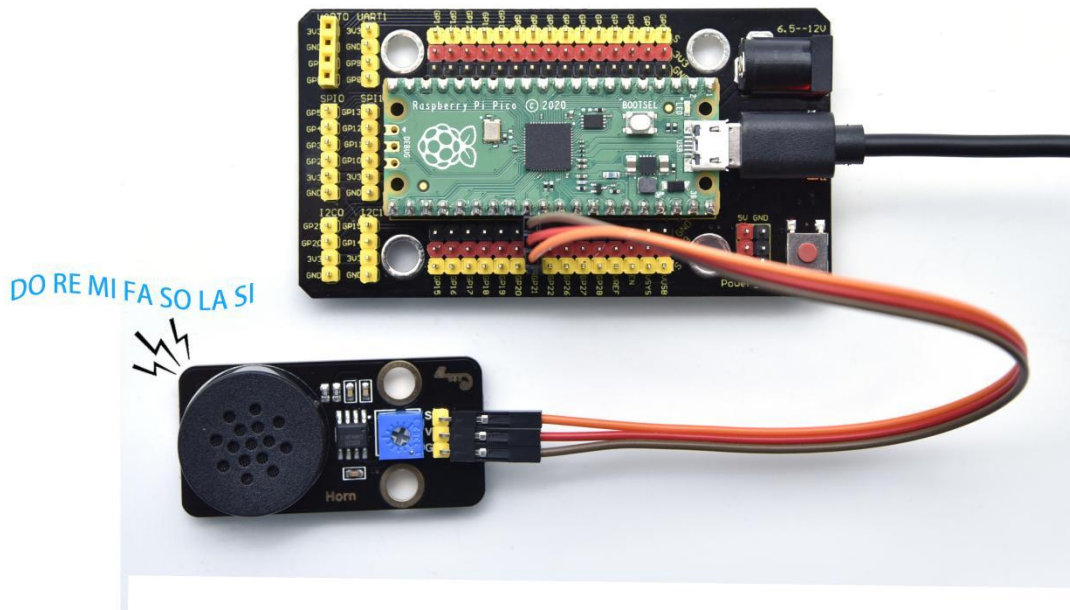
Firstly, we set duty cycle to 1000/65535, and frequency of DO, RE, MI, FA, SO, LA and SI and emit DO,RE,MI,FA,SO,LA and SI for 0.5s and turn off the buzzer.





## Test Result

Upload the code and power on. Then the audio power amplifier will emit DO, Re, Mi, Fa, So, La, Si.





## Project 10: RGB Module



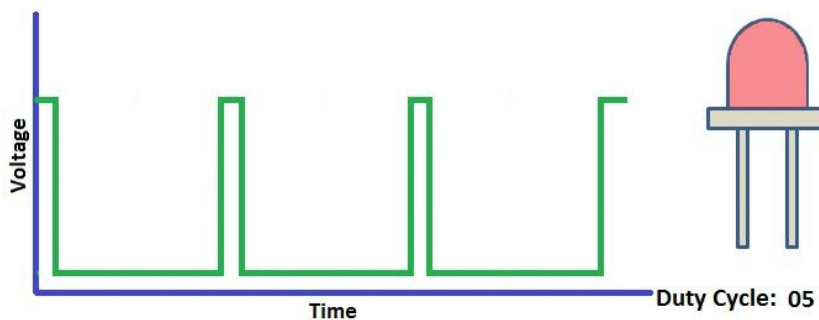
### Overview

Among these modules is a RGB module. It adopts a F10-full color RGB foggy common cathode LED. We connect the RGB module to the PWM port of MCU and the other pin to GND(for common anode RGB, the rest pin will be connected to VCC). So what is PWM?

PWM is a means of controlling the analog output via digital means. Digital control is used to generate square waves with different duty cycles (a signal that constantly switches between high and low levels) to control the analog



output. In general, the input voltages of ports are 0V and 5V. What if the 3V is required? Or a switch among 1V, 3V and 3.5V? We cannot change resistors constantly. For this reason, we resort to PWM.



For Arduino digital port voltage outputs, there are only LOW and HIGH levels, which correspond to the voltage outputs of 0V and 5V respectively. You can define LOW as "0" and HIGH as "1", and let the Arduino output five hundred '0' or "1" within 1 second. If output five hundred '1', that is 5V; if all of which is '0', that is 0V; if output 250 01 pattern, that is 2.5V. This process can be likened to showing a movie. The movie we watch are not completely continuous. Actually, it generates 25 pictures per second, which cannot be told by human eyes. Therefore, we mistake it as a continuous process. PWM works in the same way. To output different voltages, we need to control the ratio of 0 and 1. The more '0' or '1' output per unit time, the more accurate the control.

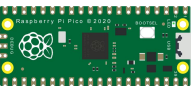
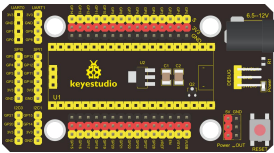





## Working Principle

For our experiment, we will control the RGB module to display different colors through three PWM values.

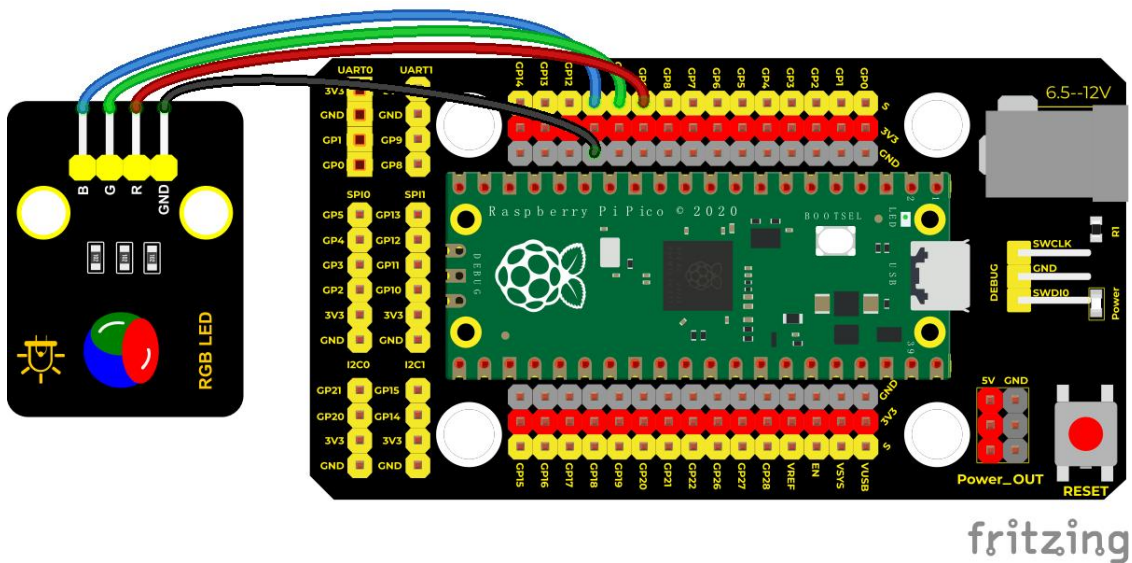
## Components



				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio Common Cathode RGB Module *1	4P Dupont Wire*1	Micro USB Cable*1



## Wiring Diagram



### Test Code 1:

...

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 10.1

\* RGB

\* <http://www.keyestudio.com>

...

from machine import Pin

from time import sleep

red = Pin(9, Pin.OUT)

green = Pin(10, Pin.OUT)



**blue = Pin(11, Pin.OUT)**

**while 1:**

**red.value(1)**

**green.value(0)**

**blue.value(0)**

**sleep(1)**

**red.value(0)**

**green.value(1)**

**blue.value(0)**

**sleep(1)**

**red.value(0)**

**green.value(0)**

**blue.value(1)**

**sleep(1)**

**Code 2:**

**...**

**\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico**

**\* lesson 10.2**

**\* RGB**

**\* <http://www.keyestudio.com>**



...

**from machine import Pin, PWM**

**from time import sleep**

**pwm\_r = PWM(Pin(9))**

**pwm\_g = PWM(Pin(10))**

**pwm\_b = PWM(Pin(11))**

**pwm\_r.freq(1000)**

**pwm\_g.freq(1000)**

**pwm\_b.freq(1000)**

**def light(red, green, blue):**

**pwm\_r.duty\_u16(red)**

**pwm\_g.duty\_u16(green)**

**pwm\_b.duty\_u16(blue)**

**while 1:**

**light(65535, 0, 0)#red**

**sleep(1)**

**light(65535, 25088, 0)#orange**

**sleep(1)**

**light(65535, 65535, 0)#yellow**





```
sleep(1)
light(0, 65535, 0)#green
sleep(1)
light(0, 0, 65535)#blue
sleep(1)
light(0, 65535, 65535)#green
sleep(1)
light(41216, 8448, 61696)#purple
sleep(1)
```

### Code Explanation:

Code 1

**Red, green and blue represent ports of red, green and blue color.**  
**Connect them to** GP9 GP10 GP11 and set to 9, 10 and 11.

The RGB will show red color, green color and blue color with an interval of one second.

### Code 2:

In the code 2, we use PWM output, the frequency we set is **freq(1000)**. **.duty\_u16()**. The data stands for the proportion of color red,



green and blue. The larger the data of the duty cycle, the larger the proportion of the color;

In the experiment, we can adjust the proportion of red, green and blue of RGB LED by setting corresponding values. Thus, the RGB can display the corresponding color.

Note: The duty ratio we set above is at most `.duty_u16(65535)`. **65535** is equal to  $256 \times 256 - 1$  (0~65535). When we compare the color table below, we only need to multiply the following value by 256. .

### **RGB Color Chart**



Color Name	Hex Code RGB	Decimal Code RGB	Color Name	Hex Code RGB	Decimal Code RGB
<b>Reds</b>			<b>Greens</b>		
IndianRed	CD5C5C	205,92,92	GreenYellow	ADFF2F	173,255,47
LightCoral	F08080	240,128,128	Chartreuse	7FFF00	127,255,0
Salmon	FA8072	250,128,114	LawnGreen	7CFC00	124,252,0
DarkSalmon	E9967A	233,150,122	Lime	00FF00	0,255,0
LightSalmon	FFA07A	255,160,122	LimeGreen	32CD32	50,205,50
Crimson	DC143C	220,20,60	PaleGreen	98FB98	152,251,152
Red	FF0000	255,0,0	LightGreen	90EE90	144,238,144
FireBrick	B22222	178,34,34	MediumSpringGreen	00FA9A	0,250,154
DarkRed	8B0000	139,0,0	SpringGreen	00FF7F	0,255,127
<b>Pinks</b>			MediumSeaGreen	3CB371	60,179,113
Pink	FFC0CB	255,192,203	SeaGreen	2E8B57	46,139,87
LightPink	FFB6C1	255,182,193	ForestGreen	228B22	34,139,34
HotPink	FF69B4	255,105,180	Green	008000	0,128,0
DeepPink	FF1493	255,20,147	DarkGreen	006400	0,100,0
MediumVioletRed	C71585	199,21,133	YellowGreen	9ACD32	154,205,50
PaleVioletRed	DB7093	219,112,147	OliveDrab	6B8E23	107,142,35
<b>Oranges</b>			Olive	808000	128,128,0
LightSalmon	FFA07A	255,160,122	DarkOliveGreen	556B2F	85,107,47
Coral	FF7F50	255,127,80	MediumAquamarine	66CDAA	102,205,170
Tomato	FF6347	255,99,71	DarkSeaGreen	8FBC8F	143,188,143
OrangeRed	FF4500	255,69,0	LightSeaGreen	20B2AA	32,178,170
DarkOrange	FF8C00	255,140,0	DarkCyan	008B8B	0,139,139
Orange	FFA500	255,165,0	Teal	008080	0,128,128
<b>Yellows</b>			<b>Blues/Cyans</b>		
Gold	FFD700	255,215,0	Aqua	00FFFF	0,255,255
ForestGreen	228B22	34,139,34	Cyan	00FFFF	0,255,255



Cornsilk	FFF8DC	255,248,220
BlanchedAlmond	FFEBCD	255,235,205
Bisque	FFE4C4	255,228,196
NavajoWhite	FFDEAD	255,222,173
Wheat	F5DEB3	245,222,179
BurlyWood	DEB887	222,164,135
Tan	D2B48C	210,180,140
RosyBrown	BC8F8F	188,143,143
SandyBrown	F4A460	244,164,96
Goldenrod	DAA520	216,165,32
DarkGoldenrod	B8860B	184,134,11
Peru	CD853F	205,133,63
Chocolate	D2691E	210,105,30
SaddleBrown	8B4513	139,69,19
Sienna	A0522D	160,82,45
Brown	A52A2A	165,42,42
Maroon	800000	128,0,0
Whites		
White	FFFFFF	255,255,255
Snow	FFFAFA	255,250,250
Honeydew	F0FFF0	240,255,240
MintCream	F5FFFA	245,255,250
Azure	F0FFFF	240,255,255
AliceBlue	F0F8FF	240,248,255
GhostWhite	F8F8FF	248,248,255
WhiteSmoke	F5F5F5	245,245,245



## Test Result

Upload the code 1, the RGB on the module will show red, green and blue color with an interval of 1s.

Upload the code 2, the RGB on the module will show red, orange, yellow, green, cyan-blue, blue, purple and white color with an interval of 1s.





## Project 11: Potentiometer

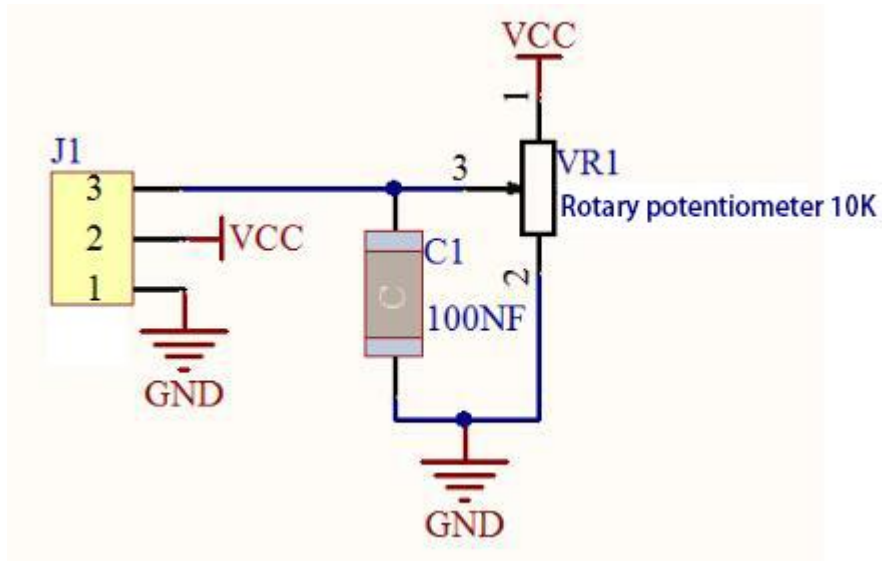


### Overview

The following we will introduce is the Keyestudio rotary potentiometer which is an analog sensor.

The digital IO ports can read the voltage value between 0 and 3.3V and the module only outputs high levels. However, the analog sensor can read the voltage value through ADC analog ports(GP26~GP28) on the pico board. In the experiment, we will display the test results on the Shell.

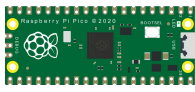
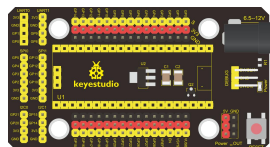
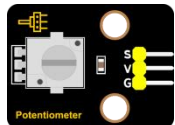






## Working Principle

It uses a 10K adjustable resistor. We can change the resistance by rotating the potentiometer. The signal S can detect the voltage changes(0-3.3V) which are analog quantity

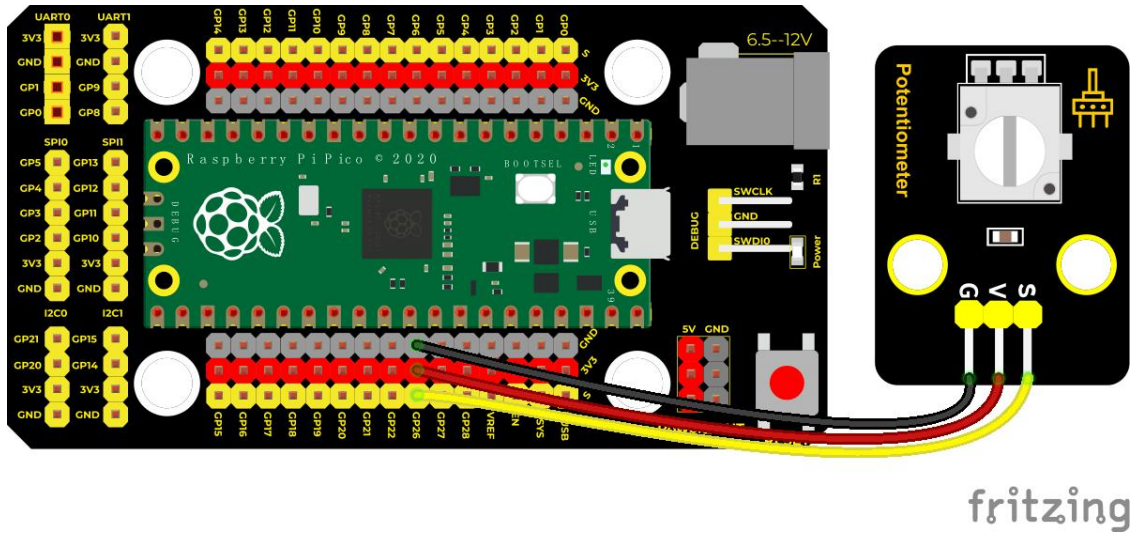
## Components

				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio Rotary Potentiometer*1	3P Dupont Wire*1	Micro USB Cable*1





## Wiring Diagram



## Test Code

...

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 11

\* Rotary potentiometer

\* <http://www.keyestudio.com>

...

```
import machine
```

```
import utime
```

```
potentiometer = machine.ADC(26)
```



**while True:**

```
    pot_value = potentiometer.read_u16()  
    print(pot_value)  
    utime.sleep(0.1)
```

### **1. Code Explanation**

**In the experiment, we create ADC examples, connect GP26(ADC(26) )**

**2. .read\_u16(): this is used to read analog value, the range is 0~65535, potentiometer.read\_u16() means that reading the output analog value of pin ADC(26), then name pot\_value**

**3. utime.sleep() delayed function can work as same as the function time.sleep().**

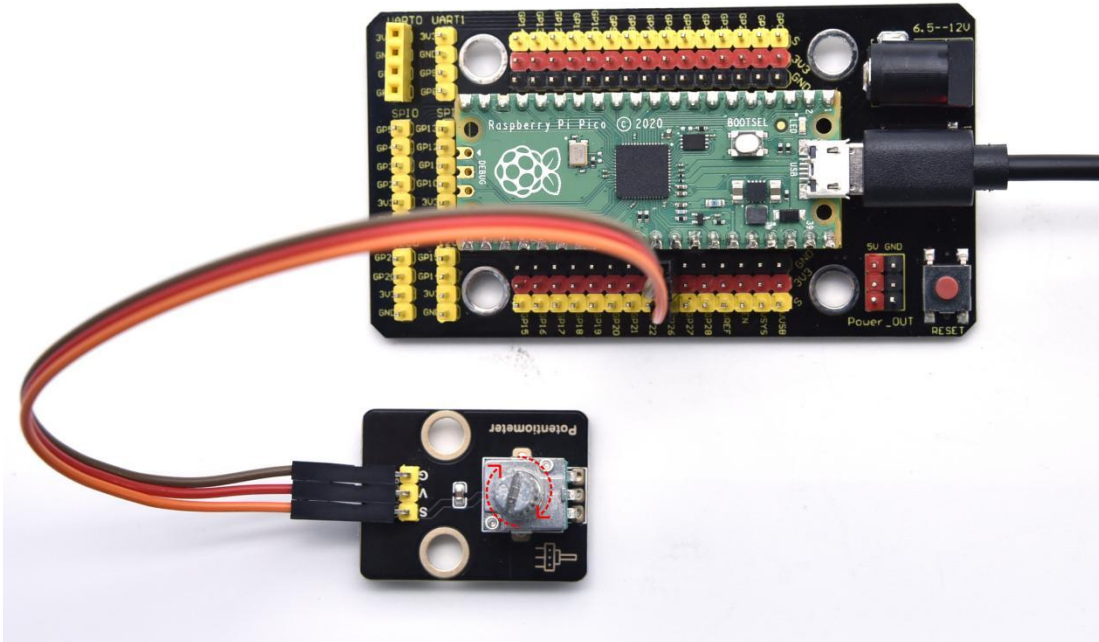
**utime.sleep() delayed function is as same as time.sleep()**

### **Test Result**

Run the test code and observe the corresponding simulation value displayed in the Shell below. In the experiment, rotate the potentiometer clockwise, the analog value increases, and turn the potentiometer



counterclockwise, the analog value decreases, the range is 65535, as shown in the figure below.



```
Shell x
65535
65535
65535
65535
63503
61422
59566
58366
57341
55581
52972
```



## Project 12: Sound Sensor



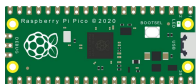
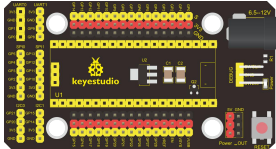
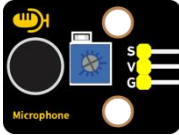


### Overview

In this kit, there is a sound sensor. In the experiment, we test the analog value corresponding to the sound level in the current environment with it. The louder the sound, the larger the analog value;

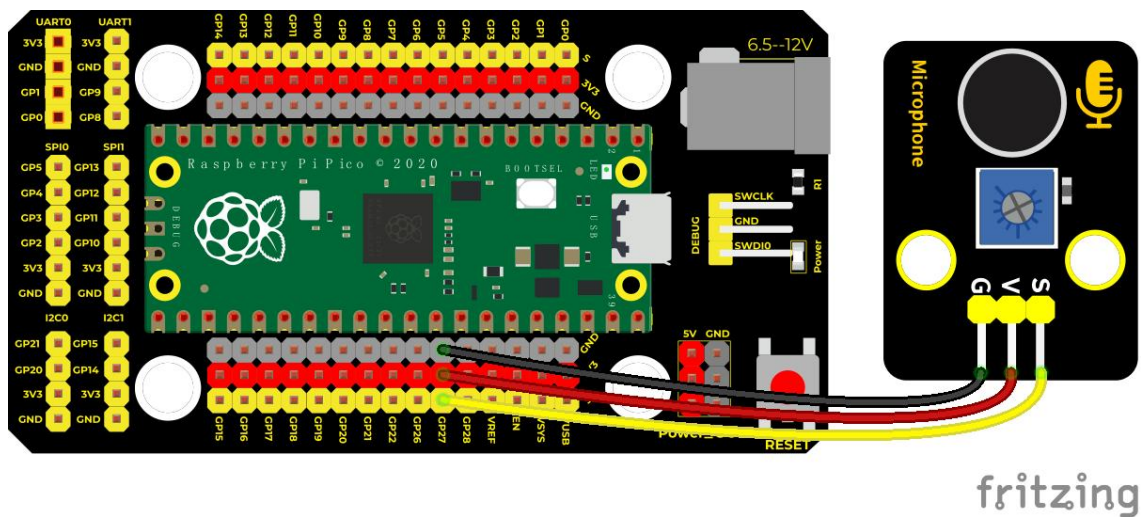




## Components

				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio Sound Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

## Wiring Diagram



## Test Code

...



**\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico**

**\* lesson 12**

**\* MicroPhone**

**\* <http://www.keyestudio.com>**

...

```
import machine
```

```
import utime
```

```
MicroPhone = machine.ADC(27)
```

```
while True:
```

```
    value = MicroPhone.read_u16()
```

```
    print(value)
```

```
    utime.sleep(0.1)
```

## **Code Explanation**

**The setting method is as same as the project 11. We use ADC(27) which is ADC(1).**

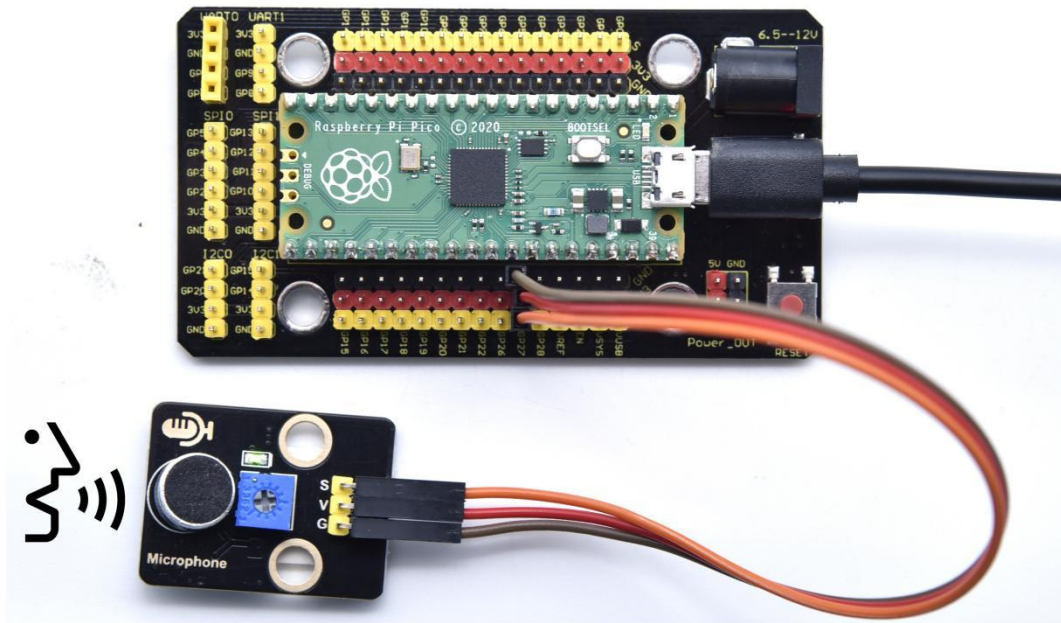




## Test Result

Upload test code, rotate clockwise the potentiometer and speak at the MIC.

Then you can see the analog value get larger, as shown below

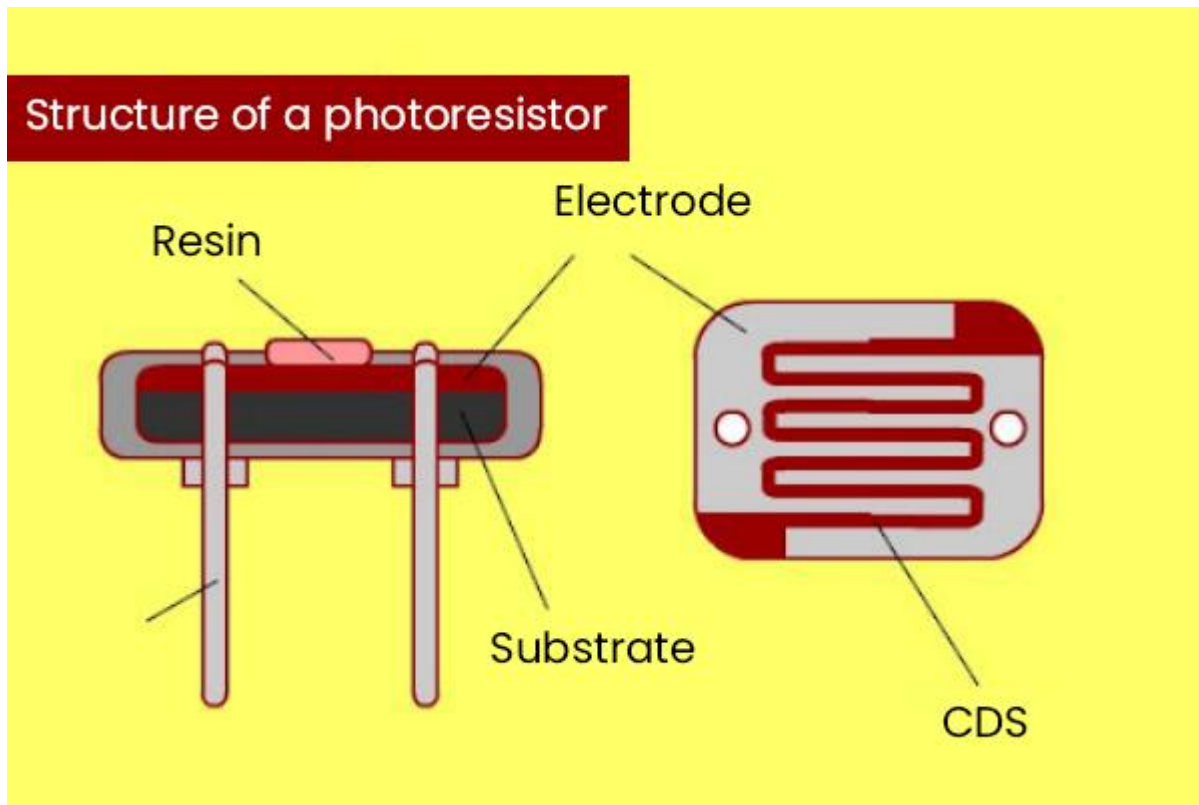


```
Shell X
560
0
464
0
0
512
0
0
0
3376
0
0
432
304
```





## Project 13: Photoresistor



### Overview

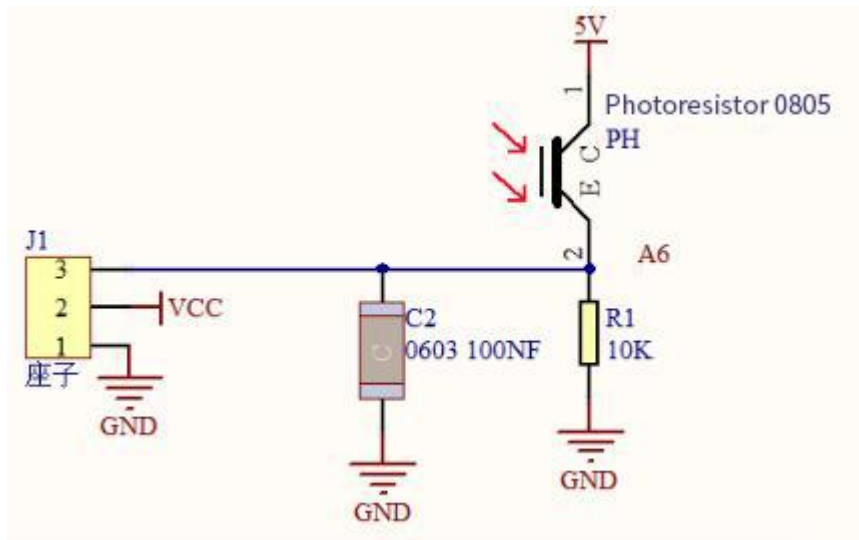
In this kit, there is a photoresistor which consists of a photosensitive resistance element. Its resistance changes with the light intensity. Also, it converts the resistance change into a voltage change.

We interface its signal terminal (S terminal) with the analog port of pico , so as to sense the change of the analog value, and display the corresponding analog value in the Shell.


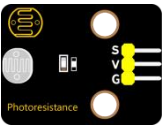


## Working Principle

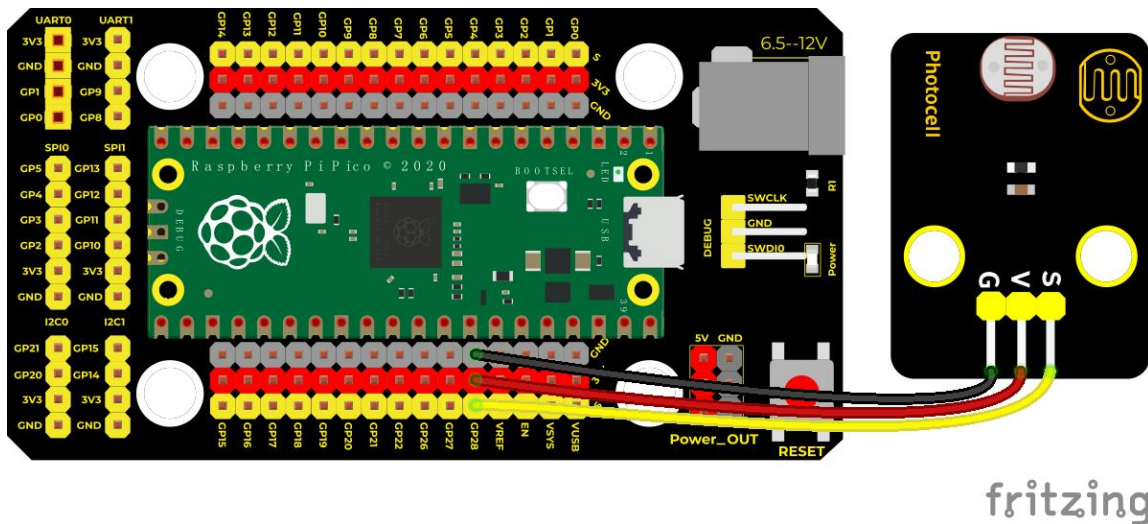
If there is no light, the resistance is  $0.2M\Omega$  and the detected voltage at the terminal 2 is close to 0. When the light intensity increases, the resistance of photoresistor and detected voltage will diminish.



## Components

				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio Photoresistor*1	3P Dupont Wire*1	Micro USB Cable*1

## Wiring Diagram



## Test Code

...

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 13

\* Photoresistance

\* <http://www.keyestudio.com>

...

```
import machine
```

```
import utime
```

```
photoresistance = machine.ADC(28)
```

```
while True:
```

```
    value = photoresistance.read_u16()
```

```
    print(value)
```



**utime.sleep(0.1)**

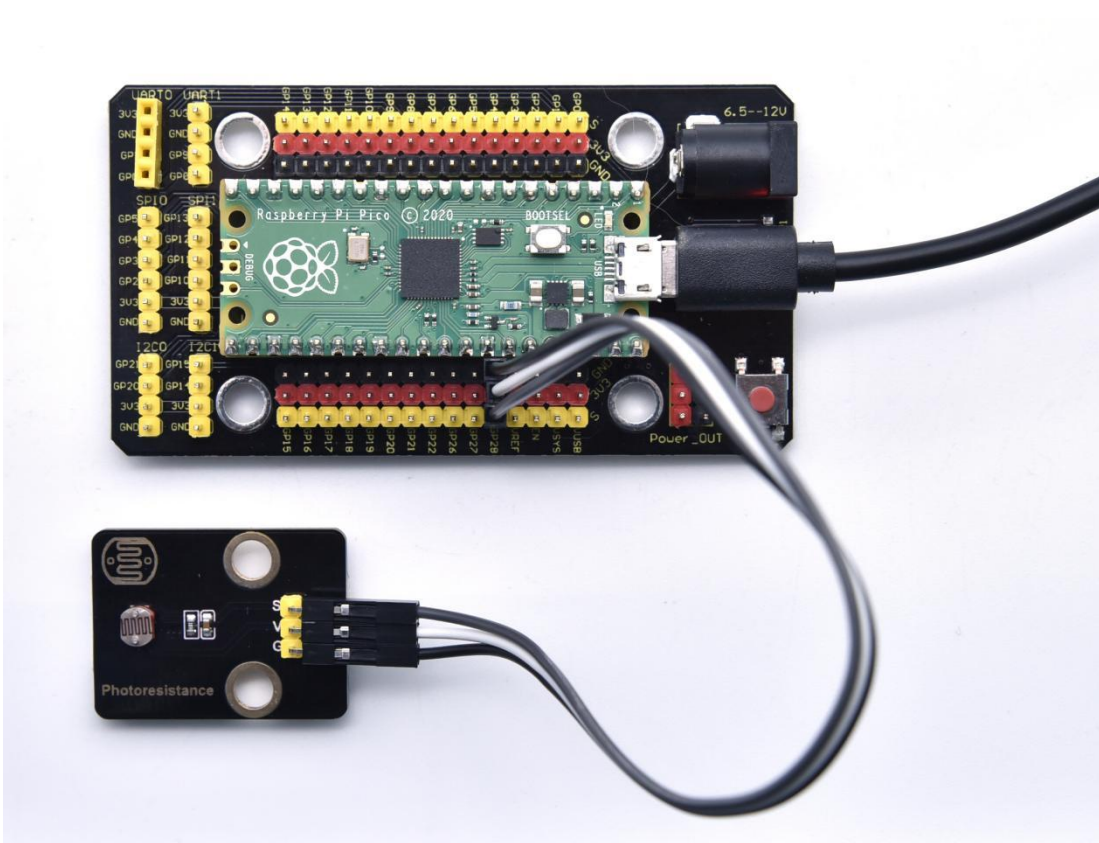
## Code Explanation

The setting method is similar to the project 11. **ADC(28)** is channel **2(ADC(2))**

## Test Result

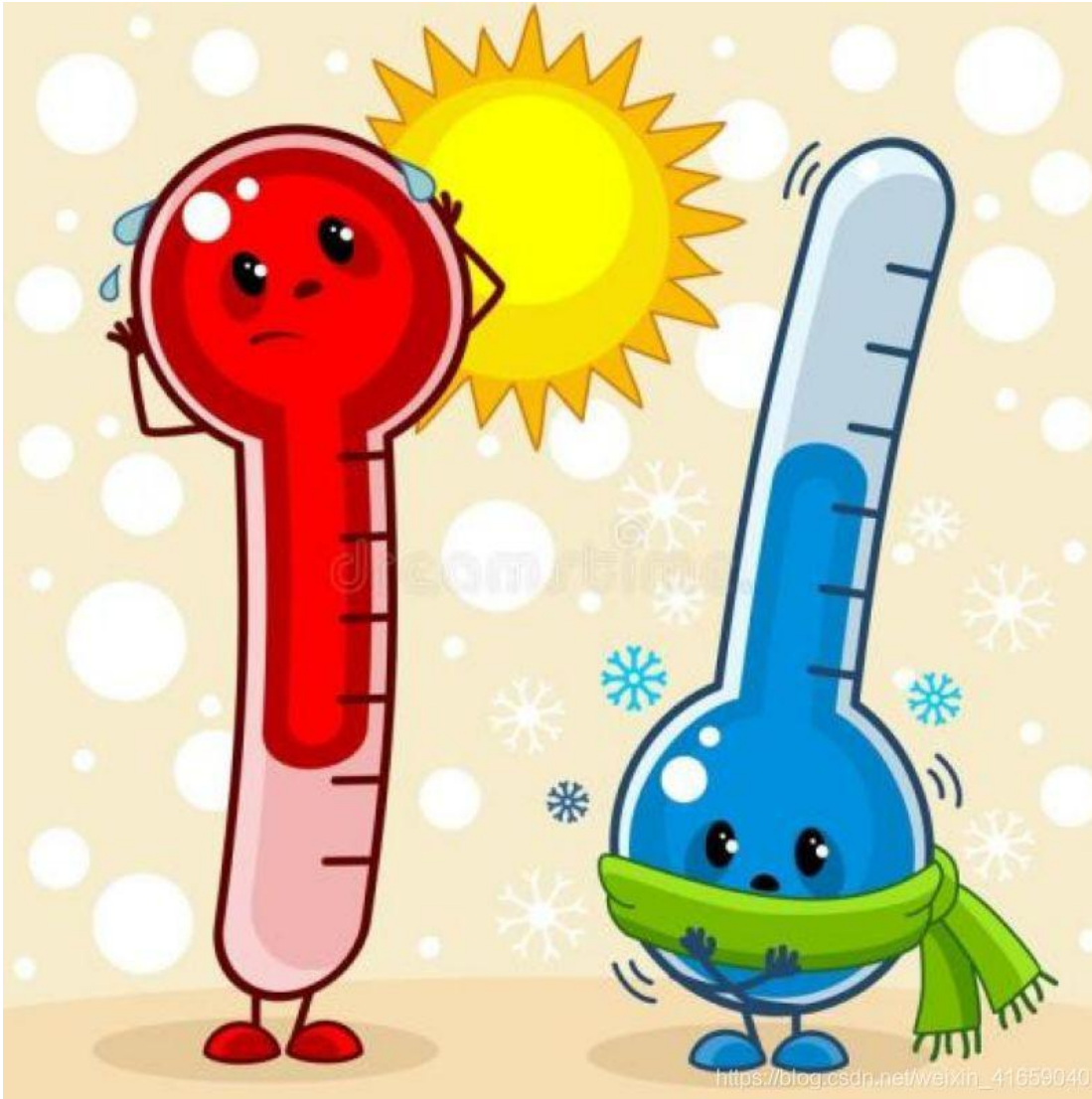
Wire up and upload the test code.

When the light intensity gets stronger, the analog value will get larger, as shown below;





## Project 14: NTC-MF52AT Thermistor



### Overview

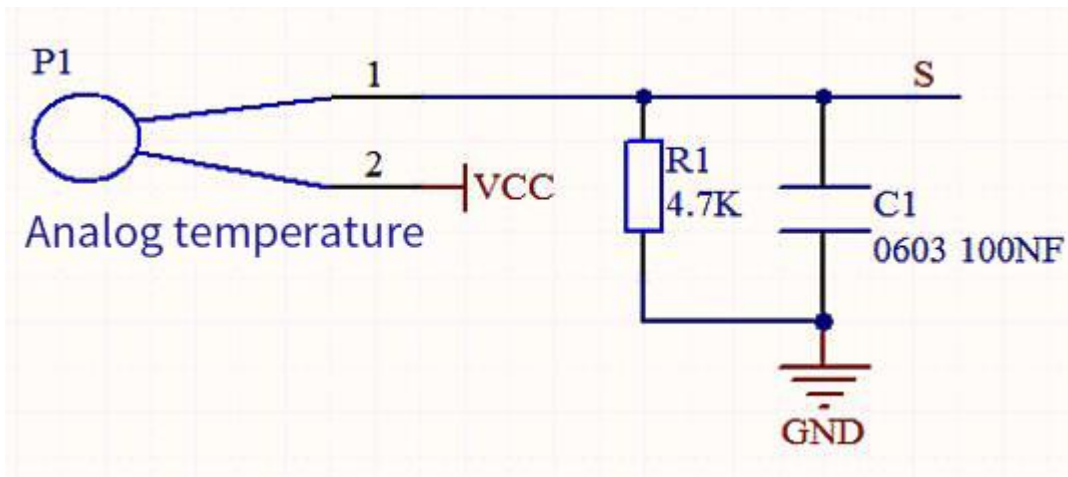
In the experiment, there is a NTC-MF52AT analog thermistor. We connect its signal terminal to the analog port of the Raspberry Pi Pico Board and read the corresponding analog value.

We can use analog values to calculate the temperature of the current



environment through specific formulas. Since the temperature calculation formula is more complicated, we only read the corresponding analog value.

## Working Principle



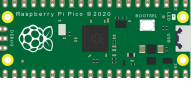
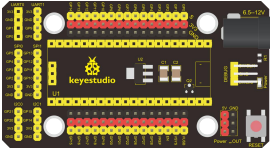
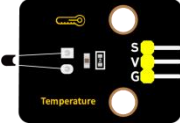


This module mainly uses NTC-MF52AT thermistor elements. The NTC-MF52AT thermistor element can sense the changes of the surrounding environment temperature. Resistance changes with the temperature, causing the voltage of the signal terminal S to change.

This sensor uses the characteristics of NTC-MF52AT thermistor element to convert resistance changes into voltage changes.

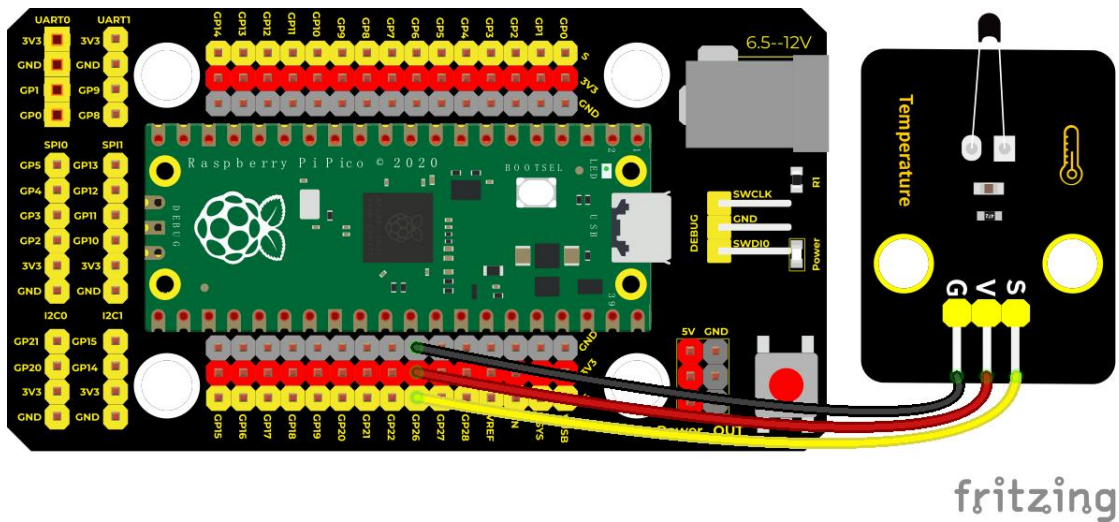




## Components

				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio NTC-MF52AT Thermistor*1	3P Dupont Wire*1	Micro USB Cable*1

## Wiring Diagram



## Test Code

...

\* [Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico](#)

\* [lesson 14](#)

\* [Temperature sensor](#)





```
* http://www.keyestudio.com
```

```
...
```

```
import machine
```

```
import utime
```

```
import math
```

```
sensor = machine.ADC(0)
```

```
while True:
```

```
    temp = sensor.read_u16()
```

```
    print("Temperature ADC: ", end = " ")
```

```
    print(temp)
```

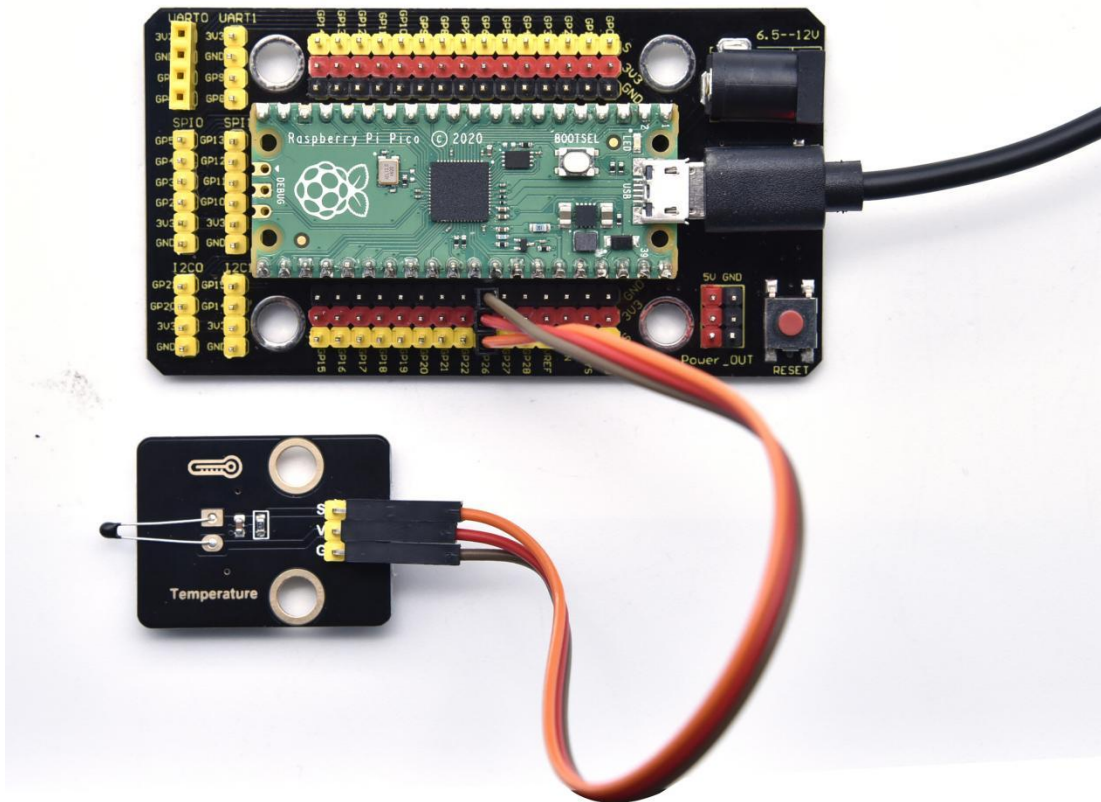
```
    utime.sleep(0.1)
```

## Code Explanation

The setting method and experiment 11 are similar.

## Test Result

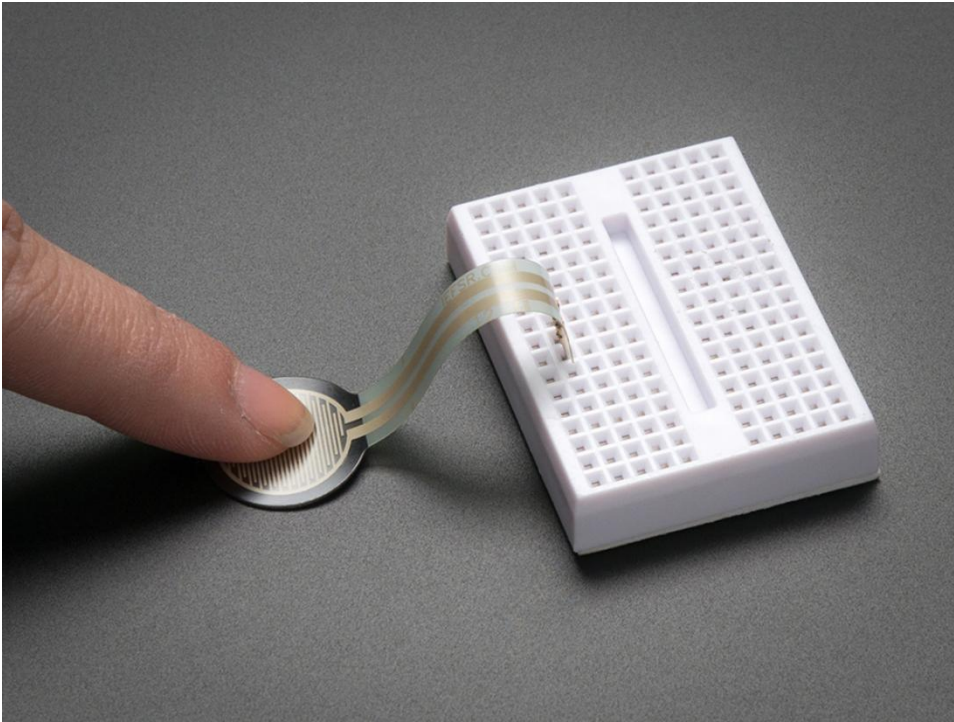
Upload the test code, the more the temperature, the larger the analog value. As shown in Shell page.



```
Shell X
Temperature ADC: 26134
Temperature ADC: 26166
Temperature ADC: 26182
Temperature ADC: 26198
Temperature ADC: 26198
Temperature ADC: 26262
Temperature ADC: 26262
Temperature ADC: 26262
Temperature ADC: 26294
Temperature ADC: 26278
Temperature ADC: 26278
Temperature ADC: 26294
Temperature ADC: 26294
Temperature ADC: 26310
```



## Project 15 Thin-film Pressure Sensor



### Overview

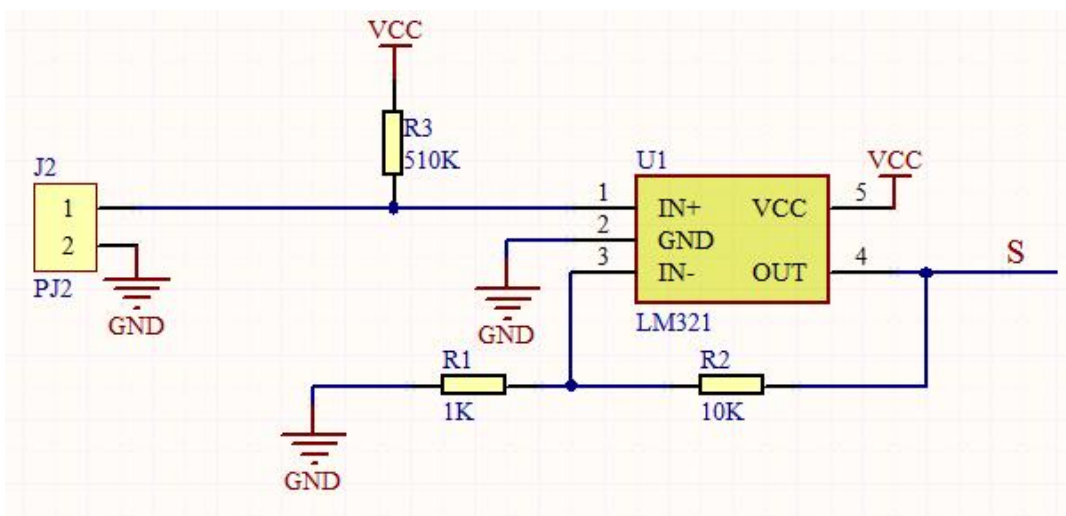
In this kit, there is a Keyestudio thin-film pressure sensor. The thin-film pressure sensor composed of a new type of nano pressure-sensitive material and a comfortable ultra-thin film substrate, has waterproof and pressure-sensitive functions.

In the experiment, we determine the pressure by collecting the analog signal on the S end of the module. The smaller the analog value, the greater the pressure; and the displayed results will shown on the Shell




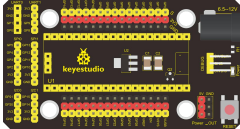



## Working Principle

When the sensor is pressed by external forces, the resistance value of sensor will vary. We convert the pressure signals detected by the sensor into the electric signals through a circuit. Then we can obtain the pressure changes by detecting voltage signal changes.

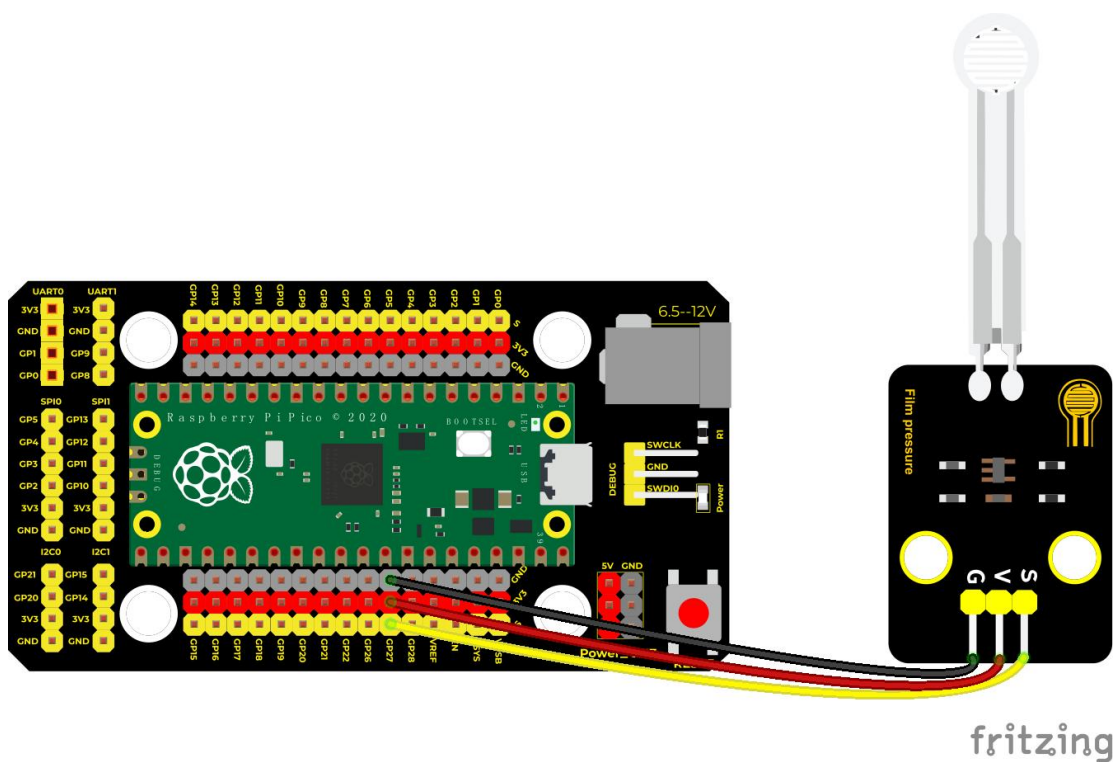




## Components

				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio Thin-film Pressure Sensor*1	3P Dupont Wire*1	Micro USB Cable*1

## Wiring Diagram



## Test Code

...



**\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico**

**\* lesson 15**

**\* Film pressure sensor**

**\* <http://www.keyestudio.com>**

...

```
import machine
```

```
import utime
```

```
film = machine.ADC(1)
```

```
while True:
```

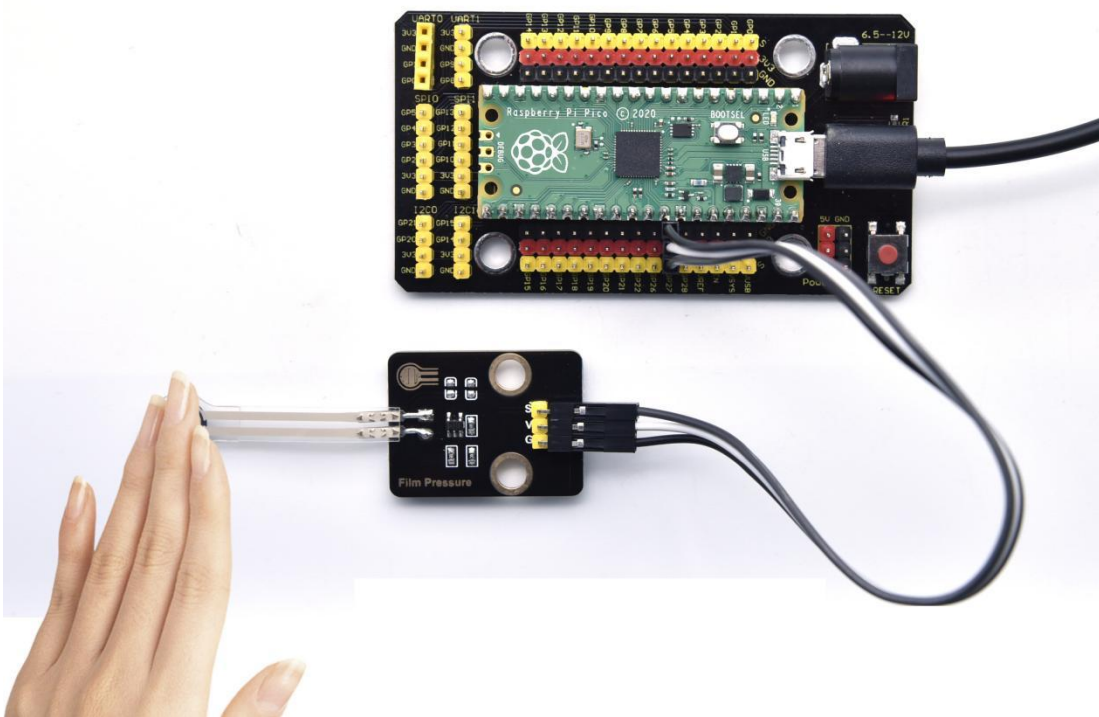
```
    value = film.read_u16()
```

```
    print(value)
```

```
    utime.sleep(0.1)
```

## **Test Result**

Upload code, when the thin-film is pressed by fingers, the analog value will decrease, as shown below;



```
Shell X
65535
65535
65535
21205
6369
8562
6289
5633
7249
6657
6785
9394
8994
5457
```





## Project 16: Joystick Module



### Overview

Game handle controllers are ubiquitous.

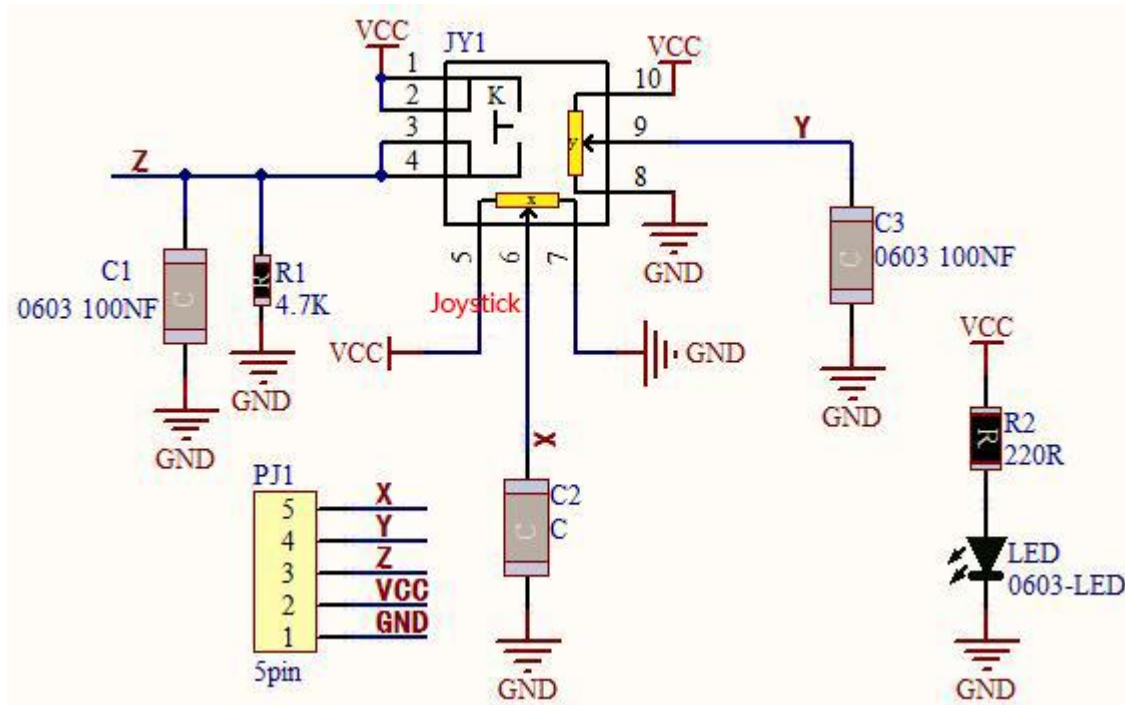
It mainly uses PS2 joysticks. When controlling it, we need to connect the X and Y ports of the module to the analog port of the single-chip microcomputer, port B to the digital port of the single-chip microcomputer, VCC to the power output port(3.3-5V), and GND to the GND of the MCU. We can read the high and low levels of two analog values and one digital port) to determine the working status of the joystick on the module.

In the experiment, two analog values(x axis and y axis) will be shown on Shell.



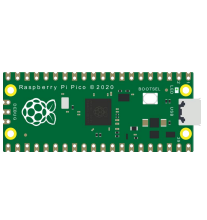
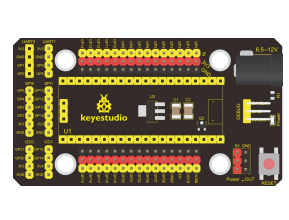





## Working Principle

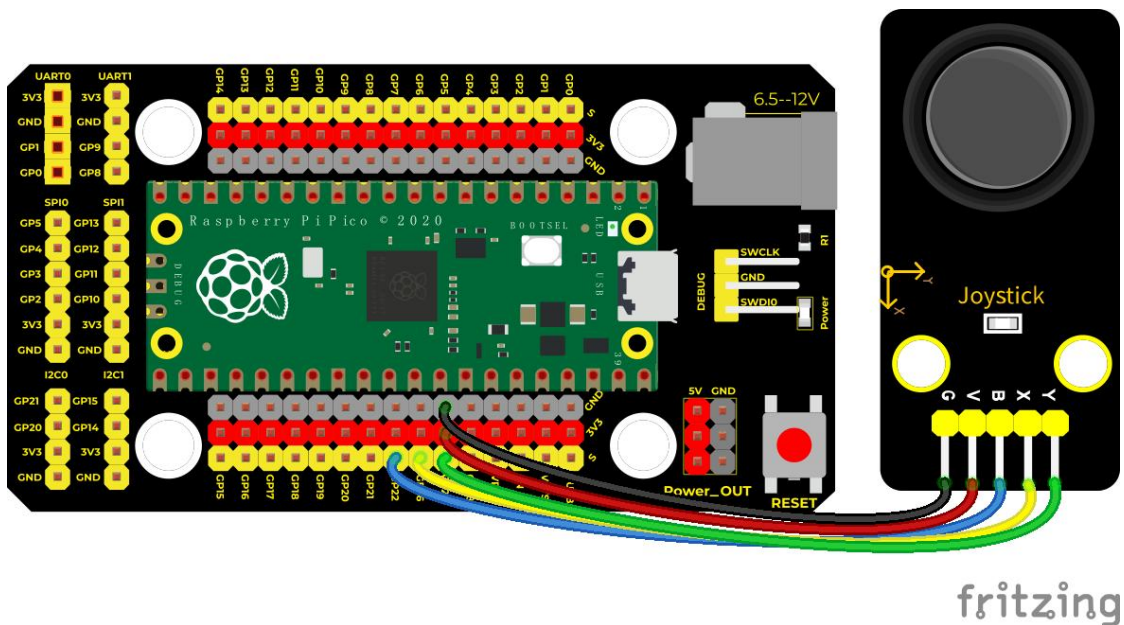


In fact, its working principle is very simple. Its inside structure is equivalent to two adjustable potentiometers and a button. When this button is not pressed and the module is pulled down by R1, low levels will be output ; on the contrary, when the button is pressed, VCC will be connected (high levels), When we move the joystick, the internal potentiometer will adjust to output different voltages, and we can read the analog value.

## Components

				
<p>Raspberry Pi Pico Board*1</p>	<p>Raspberry Pi Pico Shield*1</p>	<p>Keystudio Joystick Module*1</p>	<p>5P Dupont Wire*1</p>	<p>Micro USB Cable*1</p>

## Wiring Diagram



## Test Code

...

### \* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico



**\* lesson 16**

**\* Joystick**

**\* <http://www.keyestudio.com>**

**...**

**import machine**

**import utime**

**B = machine.Pin(22, machine.Pin.IN)**

**X = machine.ADC(26)**

**Y = machine.ADC(27)**

**while True:**

**B\_value = B.value()**

**X\_value = X.read\_u16()**

**Y\_value = Y.read\_u16()**

**print("button:", end = " ")**

**print(B\_value, end = " ")**

**print("X:", end = " ")**

**print(X\_value, end = " ")**

**print("Y:", end = " ")**

**print(Y\_value)**

**utime.sleep(0.1)**

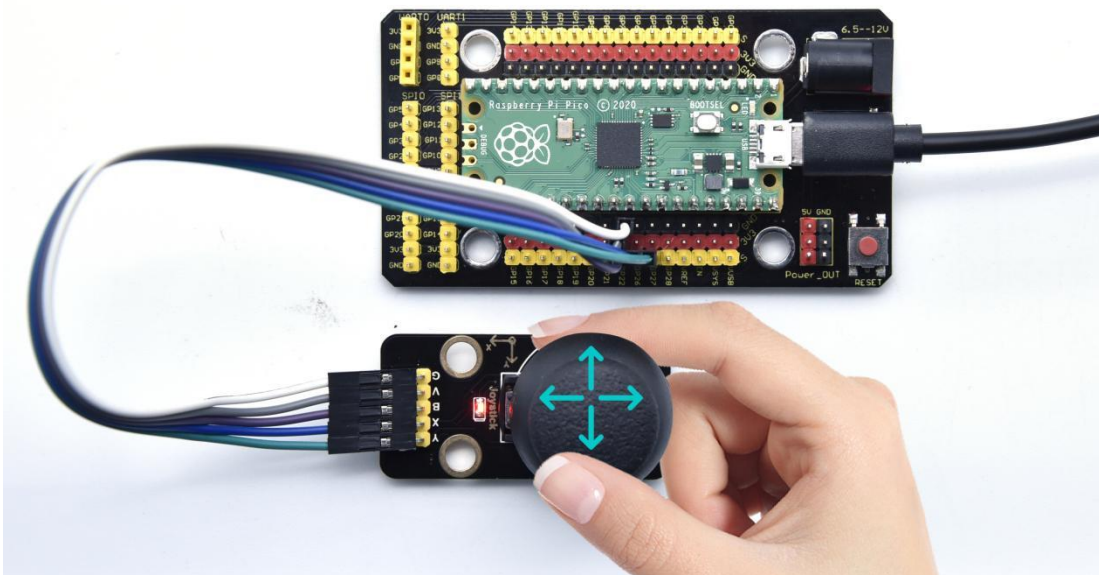


## Code Explanation

In the experiment, X is set to **ADC(26)**, Y is set to **ADC(27)** and the pin of the button is set to **GP22(input mode)**. When displaying data, we can add **end = " "** behind the function `print()` so as to not enter a new line while printing data.

## Test Result

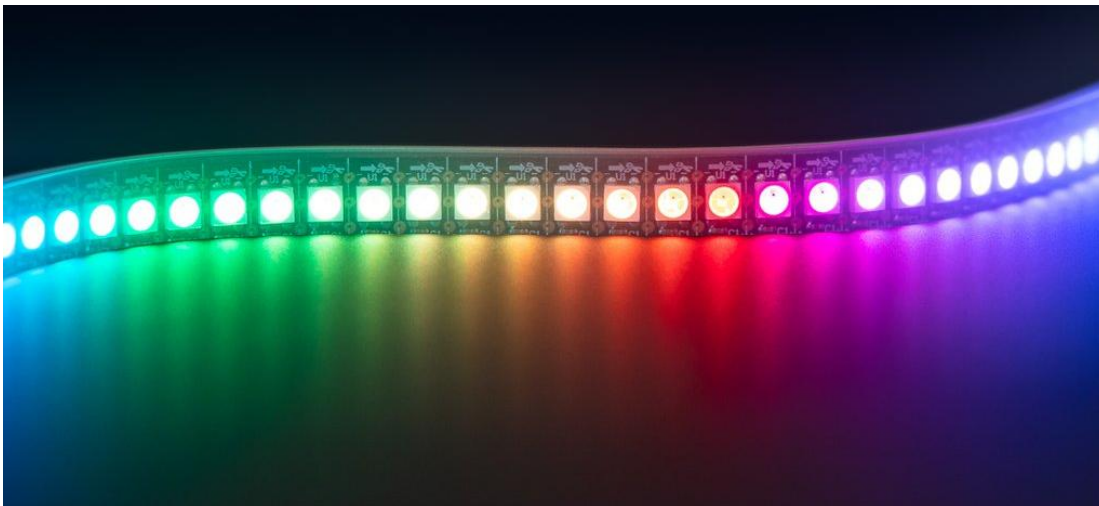
Upload the test code, move the joystick, then the value of x axis and y axis will change; press the thumb button, the value is 1; in contrast, the value is 0 as shown below;





```
Shell X
button: 0 X: 33048 Y: 33000
button: 0 X: 33016 Y: 33048
button: 0 X: 33080 Y: 33048
button: 0 X: 33016 Y: 33000
button: 0 X: 33048 Y: 33032
button: 0 X: 33048 Y: 33176
button: 0 X: 33112 Y: 33048
button: 0 X: 33080 Y: 33048
button: 1 X: 33064 Y: 33032
button: 1 X: 33032 Y: 33016
button: 1 X: 33080 Y: 33032
button: 1 X: 33048 Y: 33064
button: 1 X: 33048 Y: 32808
button: 1 X: 33048 Y: 33032
```

## Project 17: SK6812 RGB Module



### Overview

In previous lessons, we learned about the plug-in RGB module and used PWM signals to color the three pins of the module.

There is a Keyestudio 6812 RGB module whose the driving principle is different from the plug-in RGB module. It can only control with one pin. This is a set. It is an intelligent externally controlled LED light source with the control circuit and the light-emitting circuit. Each LED element is the



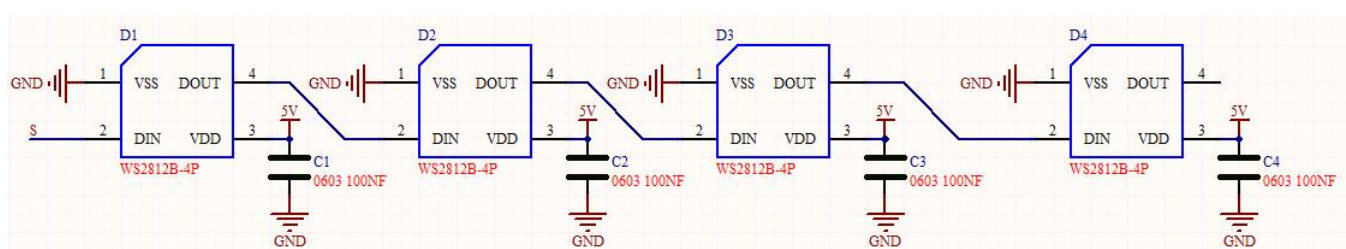
same as a 5050 LED lamp bead, and each component is a pixel. There are four lamp beads on the module, which indicates four pixels

In the experiment, we make different lights show different colors.

## Working Principle

From the schematic diagram, we can see that these four pixel lighting beads are all connected in series. In fact, no matter how many they are, we can use a pin to control a light and let it display any color. The pixel point contains a data latch signal shaping amplifier drive circuit, a high-precision internal oscillator and a 12V high-voltage programmable constant current control part, which effectively ensures the color of the pixel point light is highly consistent.

The data protocol adopts a single-wire zero-code communication method. After the pixel is powered up and reset, the S terminal receives the data transmitted from the controller. The first 24bit data sent is extracted by the first pixel and sent to the data latch of the pixel.





## Wiring Diagram



- 131





**\* 6812 RGB LED**

**\* <http://www.keyestudio.com>**

**...**

**import array, time**

**from machine import Pin**

**import rp2**

**# Configure the number of sk6812 LEDs, pins and brightness.**

**NUM\_LEDS = 4**

**PIN\_NUM = 16**

**brightness = 0.1**

**@rp2.asm\_pio(sideset\_init=rp2.PIO.OUT\_LOW,**

**out\_shiftdir=rp2.PIO.SHIFT\_LEFT, autopull=True, pull\_thresh=24)**

**def sk6812():**

**T1 = 2**

**T2 = 5**

**T3 = 3**

**wrap\_target()**

**label("bitloop")**

**out(x, 1) .side(0) [T3 - 1]**



```
jmp(not_x, "do_zero")    .side(1)    [T1 - 1]
jmp("bitloop")          .side(1)    [T2 - 1]
label("do_zero")
nop()                    .side(0)    [T2 - 1]
wrap()
```

**# Create the StateMachine with the sk6812 program, outputting on Pin(16).**

```
sm      =      rp2.StateMachine(0,      sk6812,      freq=8_000_000,
sideset_base=Pin(PIN_NUM))
```

**# Start the StateMachine, it will wait for data on its FIFO.**

```
sm.active(1)
```

**# Display a pattern on the LEDs via an array of LED RGB values.**

```
ar = array.array("I", [0 for _ in range(NUM_LEDS)])
```

```
def pixels_show():
```

```
    dimmer_ar = array.array("I", [0 for _ in range(NUM_LEDS)])
```

```
    for i,c in enumerate(ar):
```

```
        r = int(((c >> 8) & 0xFF) * brightness)
```



```
g = int(((c >> 16) & 0xFF) * brightness)
```

```
b = int((c & 0xFF) * brightness)
```

```
dimmer_ar[i] = (g<<16) + (r<<8) + b
```

```
sm.put(dimmer_ar, 8)
```

```
time.sleep_ms(10)
```

```
def pixels_set(i, color):
```

```
    ar[i] = (color[1]<<16) + (color[0]<<8) + color[2]
```

```
def pixels_fill(color):
```

```
    for i in range(len(ar)):
```

```
        pixels_set(i, color)
```

```
RED = (255, 0, 0)
```

```
GREEN = (0, 255, 0)
```

```
BLUE = (0, 0, 255)
```

```
WHITE = (255, 255, 255)
```

```
BLACK = (0, 0, 0)
```

```
pixels_set(0, RED)
```

```
pixels_set(1, GREEN)
```

```
pixels_set(2, BLUE)
```



```
pixels_set(3, WHITE)
pixels_show()
time.sleep(5)
...

for i in range(len(ar)):
    pixels_set(i, BLACK)
pixels_show()
...
```

### Code Explanation

**NUM\_LEDS = 4**, there are four light beads, therefore, we set to 4

**PIN\_NUM = 16**, this is the pin number, we connect to GP16, can be changeable

**brightness = 0.1**, this is the brightness setting, number 1 is the brightest

**pixels\_show()**, this function is used to refresh and display

**pixels\_set(i, color)**, this function is used to set up the location of 6812RGB

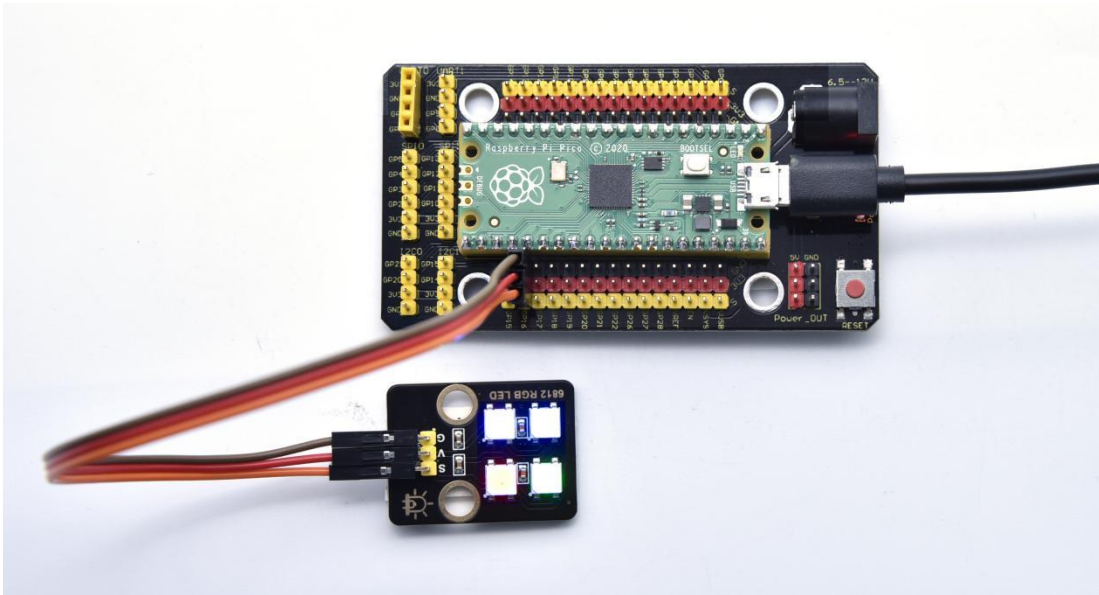
**pixels\_fill(color)**, display colors of all light beads

### Test Result

Upload the code, wire up according to connection diagrams and power on. Then we can see the light beads on the module show red, green, blue and

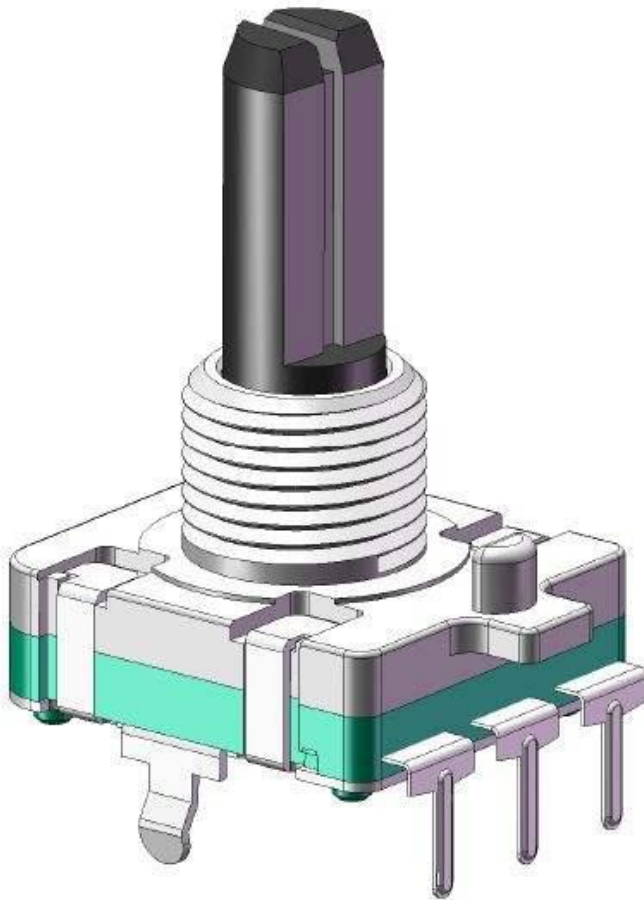


white color, as shown below;





## Project 18: Rotary Encoder



### Overview

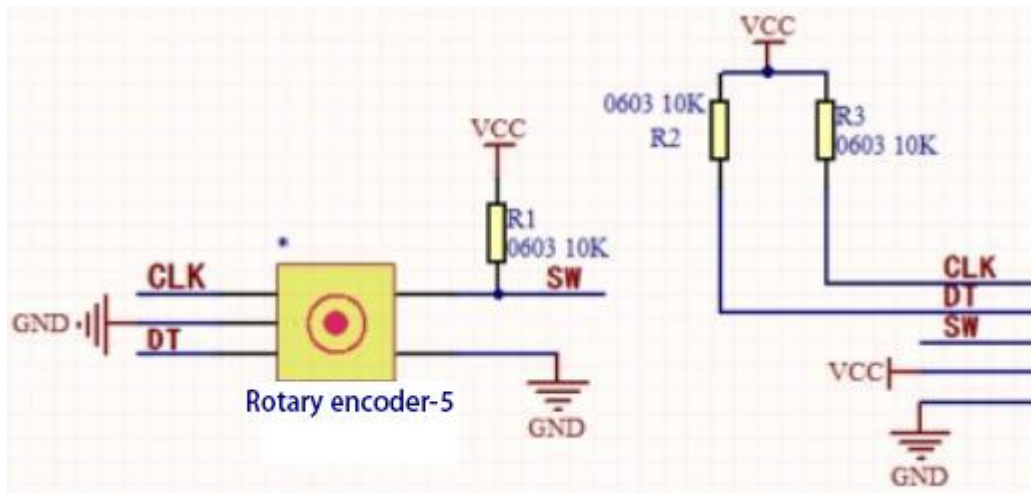
In this kit, there is a Keyestudio rotary encoder, dubbed as switch encoder. It is applied to automotive electronics, multimedia audio, instrumentation, household appliances, smart home, medical equipment and so on.

In the experiment, it is used for counting. When we rotate the rotary encoder clockwise, the set data falls by 1; if you rotate it anticlockwise, the set data is up 1; and when the middle button is pressed, the value will be



show on Shell.

## Working Principle

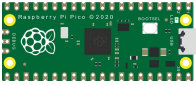
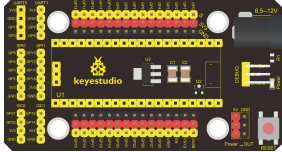
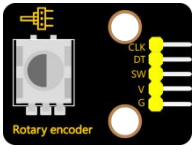




The incremental encoder converts the displacement into a periodic electric signal, and then converts this signal into a counting pulse, and the number of pulses indicates the size of the displacement. This module mainly uses 20-pulse rotary encoder components. It can calculate the number of pulses output during clockwise and reverse rotation. There is no limit to count rotation. It resets to the initial state, that is, starts counting from 0.

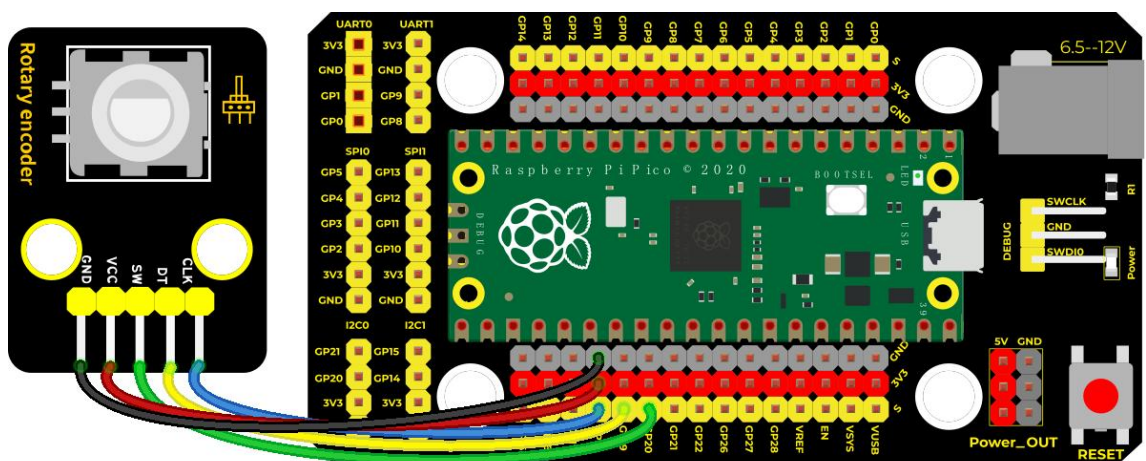




## Components

				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio Rotary Encoder*1	5P Dupont Wire*1	Micro USB Cable*1

## Wiring Diagram



fritzing

## Test Code

...



\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 18

\* Encoder

\* <http://www.keyestudio.com>

...

```
import time
```

```
from rotary_irq_rp2 import RotaryIRQ
```

```
from machine import Pin
```

```
SW=Pin(20,Pin.IN,Pin.PULL_UP)
```

```
r = RotaryIRQ(pin_num_clk=18,  
               pin_num_dt=19,  
               min_val=0,  
               reverse=False,  
               range_mode=RotaryIRQ.RANGE_UNBOUNDED)
```

```
val_old = r.value()
```

```
while True:
```

```
    try:
```

```
        val_new = r.value()
```

```
        if SW.value()==0 and n==0:
```

```
            print("Button Pressed")
```

```
            print("Selected Number is : ",val_new)
```

```
            n=1
```



```
while SW.value()==0:

    continue

n=0

if val_old != val_new:

    val_old = val_new

    print('result =', val_new)

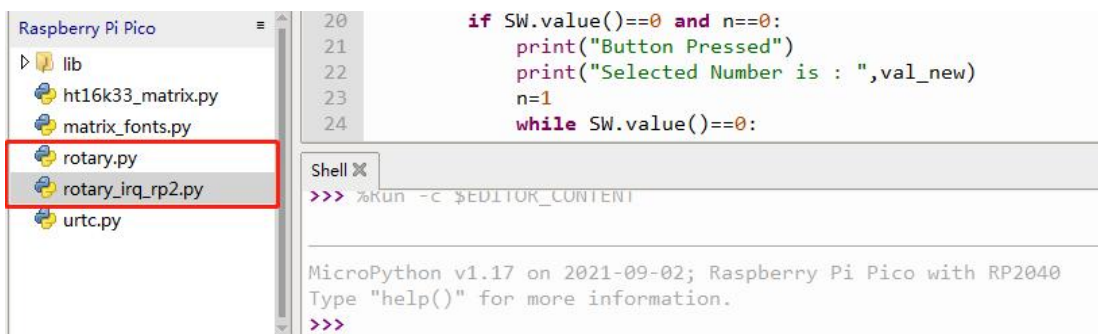
time.sleep_ms(50)

except KeyboardInterrupt:

    break
```

## Code Explanation

1. In the experiment, we need to add the encoder module to pico first, then import the module. You just need to save the .py file on the pico.
2. Add the encoder module and click file as follows;



The above picture shows that we saved them in the pico successfully. Then



we can use **from rotary\_irq\_rp2 import RotaryIRQ**

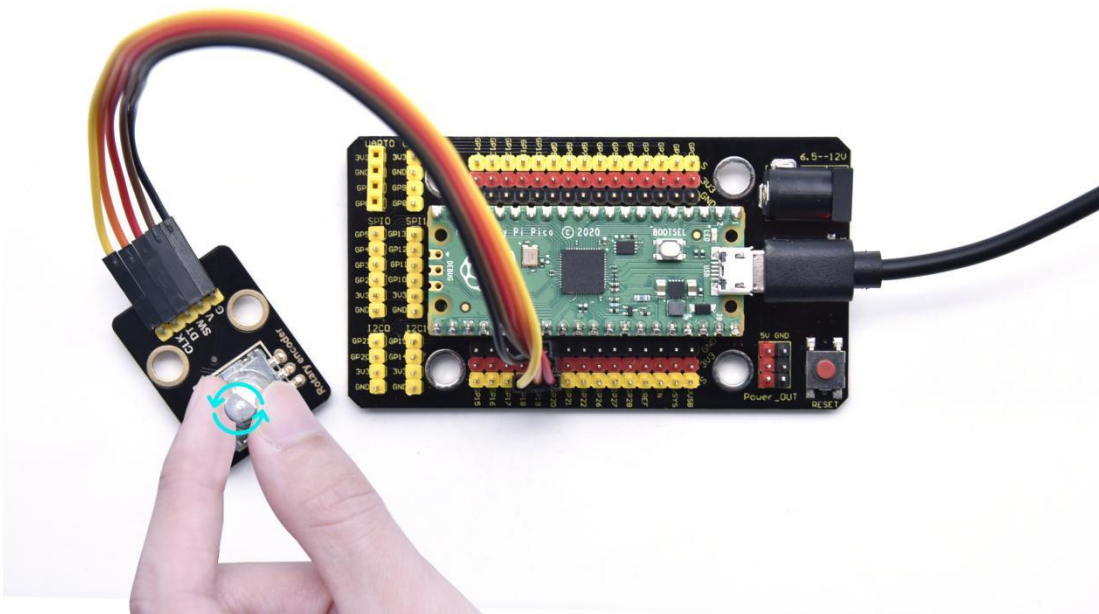
Next, we can see the pin port, **SW=Pin(20,Pin.IN,Pin.PULL\_UP)** means the pin of SW is connected to GP20, **pin\_num\_clk=18** shows that CLK is connected to GP18. **pin\_num\_dt=19** indicates that DT is linked with GP19. These pins can be changed.

1. **try/except** is used to process the abnormal language of Python, try is the executable code. Press Ctrl+C to exit program.

2. **r.value()** value returning to the encoder: the value returning the encoder

## Test Result

Upload the code, rotate the knob on the rotary encoder clockwise, the displayed data will decrease; in contrast, rotate the knob anticlockwise, the data will rise. Equally, press the button on the rotary encoder, the value is shown.



```
Shell X
type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

Button Pressed
Selected Number is : 0
result = 1
result = 2
result = 3
result = 2
result = 1
result = 0
result = -1
Button Pressed
Selected Number is : -1
```



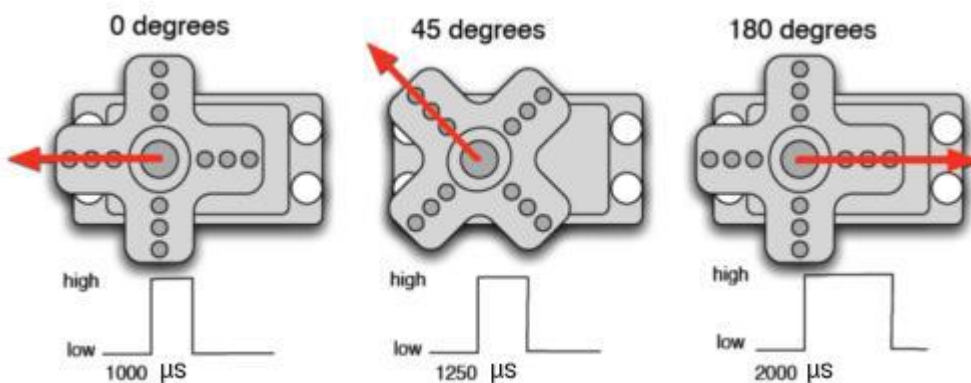
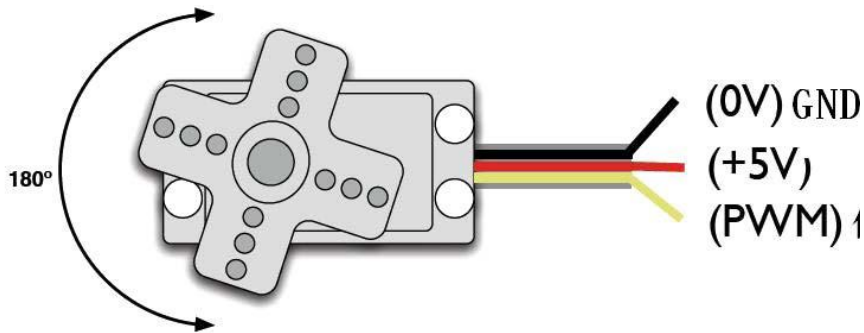
## Project 19: Servo Control



### Overview

Servo motor is a position control rotary actuator. It mainly consists of a housing, a circuit board, a core-less motor, a gear and a position sensor. Its working principle is that the servo receives the signal sent by MCU or receiver and produces a reference signal with a period of 20ms and width of 1.5ms, then compares the acquired DC bias voltage to the voltage of the potentiometer and obtain the voltage difference output.

In general, servo has three lines in brown, red and orange. The brown wire is grounded, the red one is a positive pole line and the orange one is a signal line.



## Working Principle

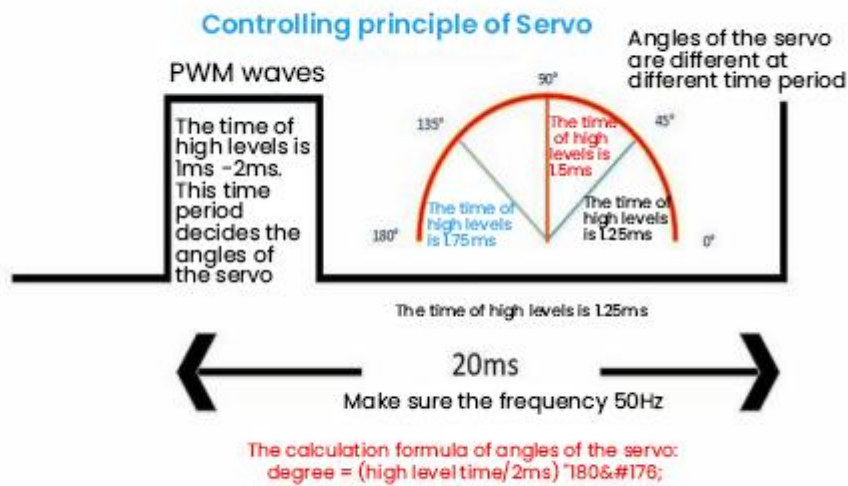
When the motor speed is constant, the potentiometer is driven to rotate through the cascade reduction gear, which leads that the voltage difference is 0, and the motor stops rotating. Generally, the angle range of servo rotation is  $0^{\circ}$  --  $180^{\circ}$

The rotation angle of servo motor is controlled by regulating the duty cycle of PWM (Pulse-Width Modulation) signal. The standard cycle of PWM signal is 20ms (50Hz). Theoretically, the width is distributed between 1ms-2ms, but in fact, it's between 0.5ms-2.5ms. The width

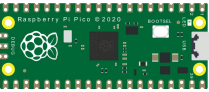
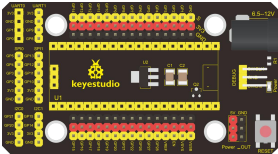






corresponds the rotation angle from 0° to 180°. But note that for different brand motors, the same signal may have different rotation angles.

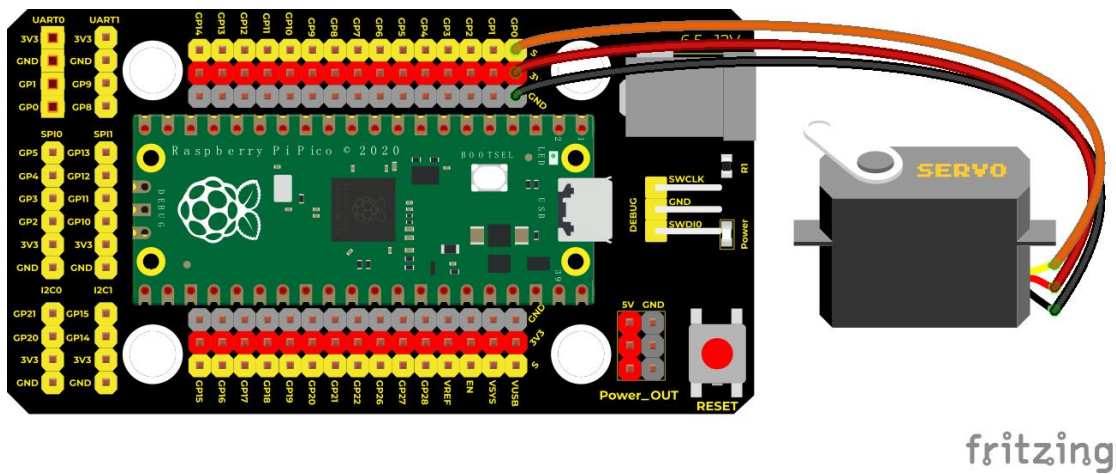


## Components

			
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Servo*1	Micro USB Cable*1



## Wiring Diagram



### Test Code//Code 1:

...

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 19.1

\* Servo test 1

\* <http://www.keyestudio.com>

...

```
from machine import Pin, PWM
```

```
import time
```

```
pwm = PWM(Pin(0))
```

```
pwm.freq(50)
```

...



## Angles correspond to duty cycle

0°----2.5%----1638

45°----5%----3276

90°----7.5%----4915

135°----10%----6553

180°----12.5%----8192

...

angle\_0 = 1638

angle\_90 = 4915

angle\_180 = 8192

while True:

    pwm.duty\_u16(angle\_0)

    time.sleep(1)

    pwm.duty\_u16(angle\_90)

    time.sleep(1)

    pwm.duty\_u16(angle\_180)

    time.sleep(1)

Code 2:

...

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico



**\* lesson 19.2**

**\* Servo test 2**

**\* <http://www.keyestudio.com>**

**...**

**from utime import sleep**

**from machine import Pin**

**from machine import PWM**

**pwm = PWM(Pin(0))#Pins of servo is connected to GP0**

**pwm.freq(50)#the cycle of 20ms, frequency is 50Hz**

**...**

**Angles correspond to duty cycle**

**0°----2.5%----1638**

**45°----5%----3276**

**90°----7.5%----4915**

**135°----10%----6553**

**180°----12.5%----8192**

**considering the error, set the duty cycle in the range of 1000~9000 to rotate 0~180°**

**...**

**# set the rotation angles of servo**

**def setServoCycle (position):**



```
pwm.duty_u16(position)
```

```
sleep(0.01)
```

```
# Convert the angle of rotation to duty cycle
```

```
def convert(x, i_m, i_M, o_m, o_M):
```

```
    return max(min(o_M, (x - i_m) * (o_M - o_m) // (i_M - i_m) + o_m),
```

```
o_m)
```

```
while True:
```

```
    for degree in range(0, 180, 1):#rotate from 0° to 180°
```

```
        pos = convert(degree, 0, 180, 1000, 9000)
```

```
        setServoCycle(pos)
```

```
    for degree in range(180, 0, -1):#rotate from 180° to 0°
```

```
        pos = convert(degree, 0, 180, 1000, 9000)
```

```
        setServoCycle(pos)
```

### Code explanation for code 1:

Convert to duty cycle according to the angle of the signal pulse width, the formula is:  $2.5 + \text{angle} / 180 * 10$ , taking the pin resolution of PWM of Pico as



an example,  $2^{16} = 65535$ , when converted to 0 degree, the duty cycle value is  $65535 * 2.5\% = 1638.375$ , when the angle is 180 degrees, its duty cycle value is  $65535 * 12.5\% = 8191.875$ , these two values will be related to the program, considering the error and rotation angle, I will set the duty cycle at 1000 and 9000 to make servo rotate by 0~180 degrees.

### Code explanation for code 2:

1. **convert(x, i\_m, i\_M, o\_m, o\_M):** X is the value we will map.

i\_m, i\_M is the lower limit and upper limit of the current value ; o\_m, o\_M is the lower limit and upper limit of the object range.

For instance, **convert(degree, 0, 180, 1000, 9000)**

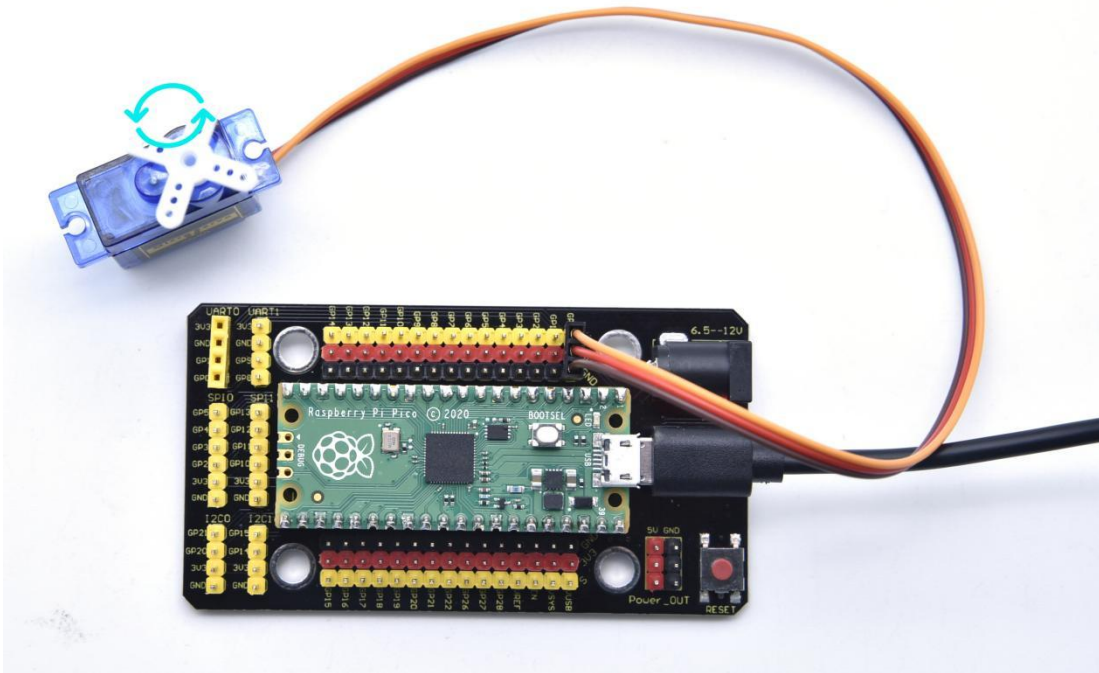
Rotation angle degree is in the range of 0° and 180°. The duty cycle we will map is in the range of 1000 and 9000

### Test Result 1:

Upload the code, the servo will rotate 0°, 90° and 180°.

### Test Result 2:

Upload the code, the servo will rotate from 0° to 180° and move 1° for each 10ms.



## Project 20: Ultrasonic Sensor

### Overview

In this kit, there is a keyes HC-SR04 ultrasonic sensor, which can detect obstacles in front and the detailed distance between the sensor and the obstacle. Its principle is the same as that of bat flying. It can emit the ultrasonic signals that cannot be heard by humans. When these signals hit an obstacle and come back immediately. The distance between the sensor and the obstacle can be calculated by the time gap of emitting signals and receiving signals.

In the experiment, we use the sensor to detect the distance between the





sensor and the obstacle, and print the test result.

Ultrasonic detector module can provide 2cm-450cm non-contact sensing distance, and its ranging accuracy is up to 3mm, very good to meet the normal requirements. The module includes an ultrasonic transmitter and receiver as well as the corresponding control circuit.

## **Working Principle**

The most common ultrasonic ranging method is the echo detection. As shown below; when the ultrasonic emitter emits the ultrasonic waves towards certain direction, the counter will count. The ultrasonic waves travel and reflect back once encountering the obstacle. Then the counter will stop counting when the receiver receives the ultrasonic waves coming back.

The ultrasonic wave is also sound wave, and its speed of sound  $V$  is related to temperature. Generally, it travels 340m/s in the air. According to time  $t$ , we can calculate the distance  $s$  from the emitting spot to the obstacle.

$$s = 340t/2.$$

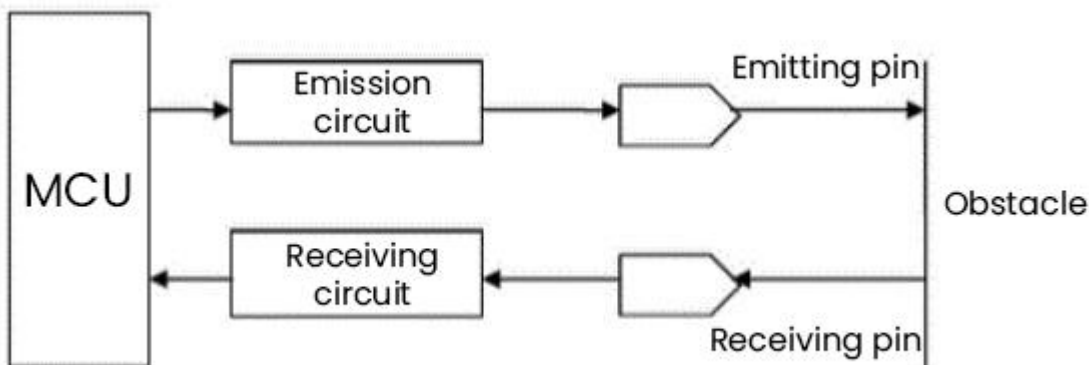
The HC-SR04 ultrasonic ranging module can provide a non-contact distance sensing function of 2cm-400cm, and the ranging accuracy can



reach as high as 3mm; the module includes an ultrasonic transmitter, receiver and control circuit. Basic working principle:

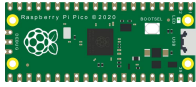
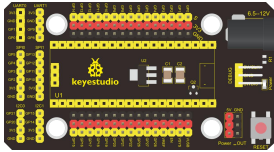



1. First pull down the TRIG, and then trigger it with at least 10us high level signal;
2. After triggering, the module will automatically transmit eight 40KHZ square waves, and automatically detect whether there is a signal to return.
3. If there is a signal returned back, through the ECHO to output a high level, the duration time of high level is actually the time from emission to reception of ultrasonic.

Test distance = high level duration \* 340m/s \* 0.5.

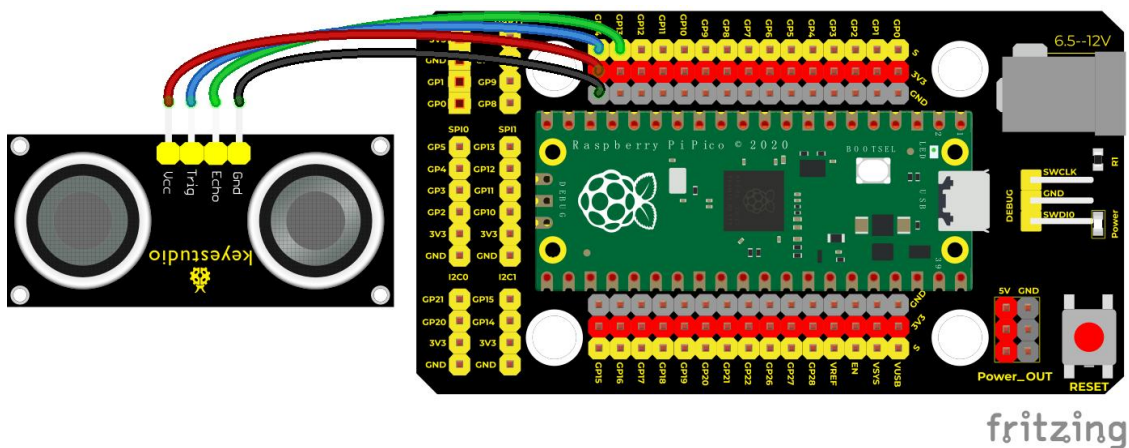


## Components



				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	keyestudio SR01 Ultrasonic Sensor*1	4P Dupont Wire*1	Micro USB Cable*1

## Wiring Diagram



## Test Code'''

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 20

\* Ultrasonic

\* <http://www.keyestudio.com>

'''

from machine import Pin



**import utime**

**# ultrasonic ranging, unit: cm**

**def getDistance(trigger, echo):**

**# produce the square wave of 10us**

**trigger.low() #give a short low level and ensure a high pulse:**

**utime.sleep\_us(2)**

**trigger.high()**

**utime.sleep\_us(10)#pull up the high levels, wait for 10ms and set to low level**

**trigger.low()**

**while echo.value() == 0: #build a while loop to detect if the pin is 0,  
record the current time**

**start = utime.ticks\_us()**

**while echo.value() == 1: #build a while loop to detect if the pin is 1,  
record the current time**

**end = utime.ticks\_us()**

**d = (end - start) \* 0.0343 / 2 # travelling time x speed of  
sound(343.2 m/s, 0.0343 for each ms), the total distance is divided by**

**1**

**return d**



```
# set pins

trigger = Pin(14, Pin.OUT)

echo = Pin(13, Pin.IN)

# main program

while True:

    distance = getDistance(trigger, echo)

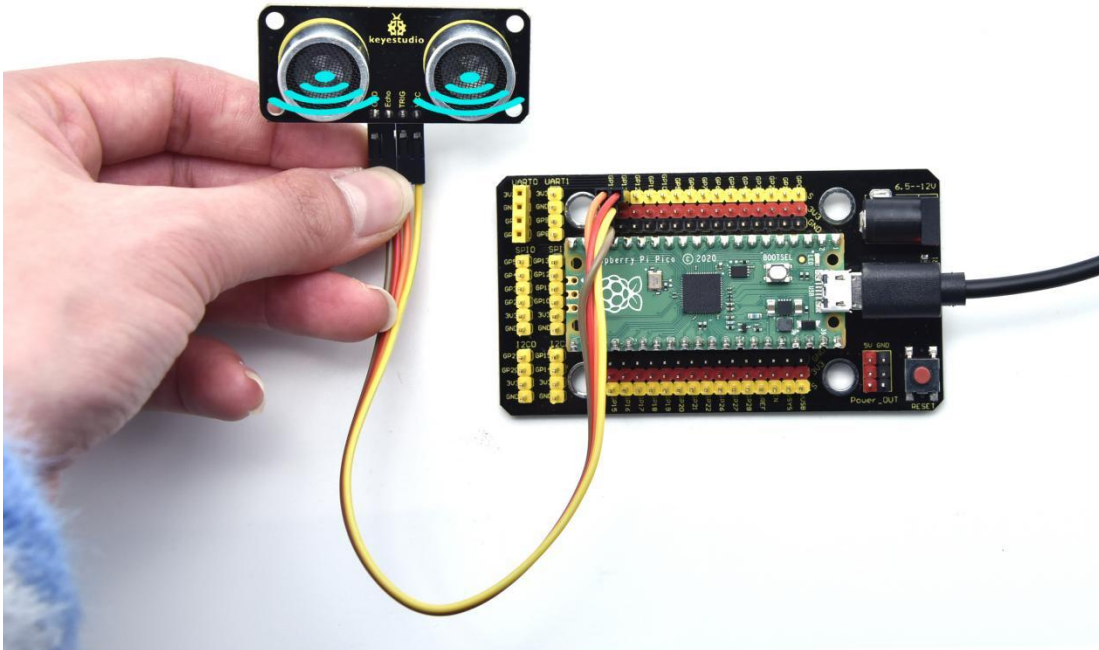
    print("The distance is : {:.2f} cm".format(distance))

    utime.sleep(0.1)
```

## Test Result

Upload the code. The distance between the ultrasonic sensor and the obstacle is shown on Shell, as shown below;

The maximum detection distance of the HC-SR04 ultrasonic sensor is 3-4m, the minimum detection distance is 2cm.



```
Shell X
The distance is : 12.76 cm
The distance is : 12.49 cm
The distance is : 11.51 cm
The distance is : 11.44 cm
The distance is : 8.71 cm
The distance is : 6.96 cm
The distance is : 6.29 cm
The distance is : 5.73 cm
The distance is : 5.20 cm
The distance is : 4.96 cm
The distance is : 4.46 cm
The distance is : 4.17 cm
The distance is : 4.18 cm
The distance is : 4.18 cm
```



## Project 21: IR Receiver Module



### Overview

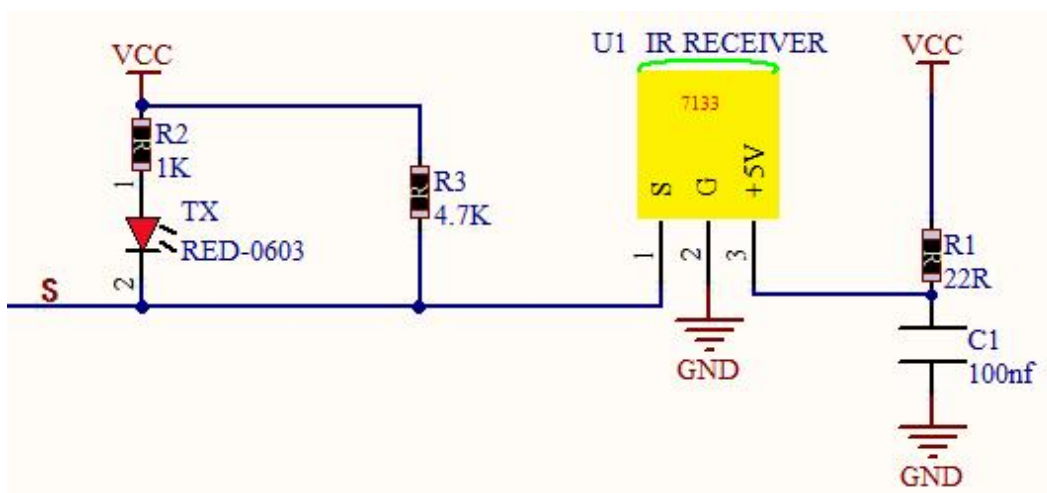
There is no doubt that infrared remote control is ubiquitous in daily life. It is used to control various household appliances, such as TVs, stereos, video recorders and satellite signal receivers. Infrared remote control is composed of infrared transmitting and infrared receiving systems, that is, an infrared remote control and infrared receiving module and a single-chip microcomputer capable of decoding.

In this experiment, we need to know how to use the infrared receiving sensor. The infrared receiving sensor mainly uses the VS1838B infrared



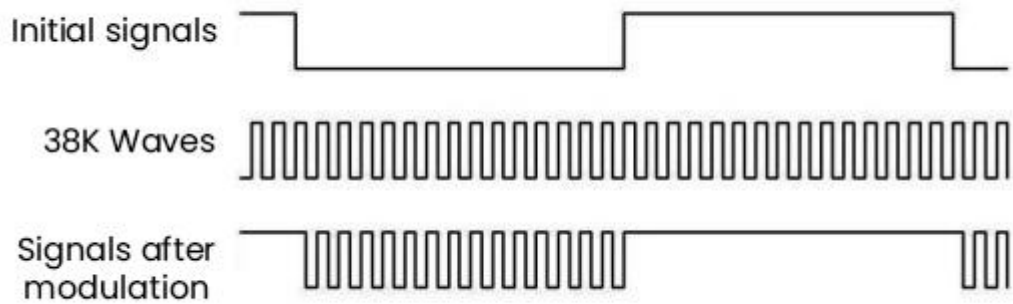
receiving sensor element. It integrates receiving, amplifying, and demodulating. The internal IC has already completed the demodulation, and the output is a digital signal. It can receive 38KHz modulated remote control signal. In the experiment, we use the IR receiver to receive the infrared signal emitted by the external infrared transmitting device, and display the received signal in the shell.

## Working Principle



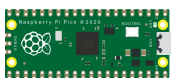
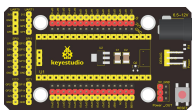




The main part of the IR remote control system is modulation, transmission and reception. The modulated carrier frequency is generally between 30khz and 60khz, and most of them use a square wave of 38kHz and a duty ratio of 1/3. A 4.7K pull-up resistor R3 is added to the signal end of the infrared receiver.





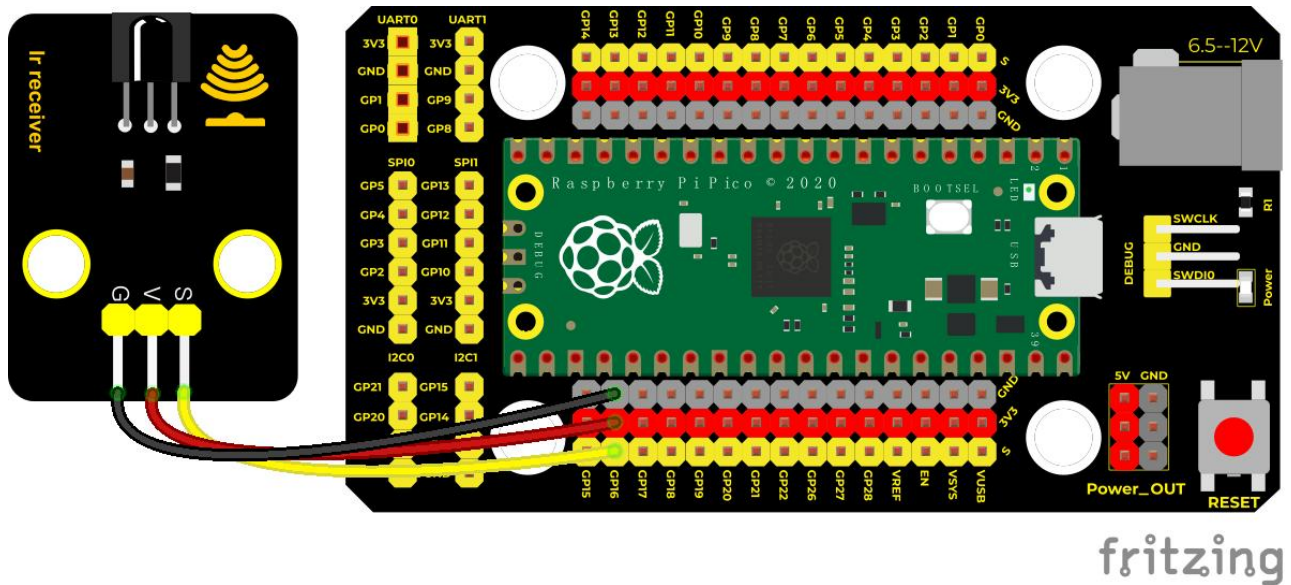
[https://blog.csdn.net/weixin\\_41655040/](https://blog.csdn.net/weixin_41655040/)

## Components

					
Raspberr y Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestu dio DIY IR Receiver *1	3P Dupont Wire*1	Micro USB Cable*1	Remote Control* 1



## Wiring Diagram



## Test Code

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 21

\* IR Receiver

\* <http://www.keyestudio.com>

...

```
import utime
```

```
from machine import Pin
```

```
ird = Pin(16,Pin.IN)
```

```
act = {"1": "LLLLLLLLHHHHHHHHHLHHLHLLHLLHLLHHH","2":
```



```
"LLLLLLLLHHHHHHHHHHHLLHHLLLLHHLLHHH","3":  
"LLLLLLLLHHHHHHHHHHHLLHHLLLLHHLLHHH",  
    "4":      "LLLLLLLLHHHHHHHHHHHLLHHLLLLHHLLHHH","5":  
"LLLLLLLLHHHHHHHHHHHLLHHLLLLHHLLHHH","6":  
"LLLLLLLLHHHHHHHHHHHLLHHHLLHLLHLLHLLH",  
    "7":      "LLLLLLLLHHHHHHHHHHHLLHHLLLLHHLLHHH","8":  
"LLLLLLLLHHHHHHHHHHHLLHHHLLHLLHLLHHH","9":  
"LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLHLLH",  
    "0":      "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLHLLH","Up":  
"LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLHLLH","Down":  
"LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLHLLH",  
    "Left":   "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLHLLH","Right":  
"LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLHLLH","Ok":  
"LLLLLLLLHHHHHHHHHHHLLLLLLHLLHHHHHLLH",  
    "*":      "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLHLLH","#":  
"LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLHLLH"}  
}
```

```
def read_irdcode(ird):
```

```
    wait = 1
```

```
    complete = 0
```

```
    seq0 = []
```

```
    seq1 = []
```



```
while wait == 1:
    if ird.value() == 0:
        wait = 0
while wait == 0 and complete == 0:
    start = utime.ticks_us()
    while ird.value() == 0:
        ms1 = utime.ticks_us()
        diff = utime.ticks_diff(ms1, start)
        seq0.append(diff)
    while ird.value() == 1 and complete == 0:
        ms2 = utime.ticks_us()
        diff = utime.ticks_diff(ms2, ms1)
        if diff > 10000:
            complete = 1
        seq1.append(diff)

code = ""
for val in seq1:
    if val < 2000:
        if val < 700:
            code += "L"
```



**else:**

**code += "H"**

**# print(code)**

**command = ""**

**for k,v in act.items():**

**if code == v:**

**command = k**

**if command == "":**

**command = code**

**return command**

**while True:**

**command = read\_ircode(ird)**

**print(command)**

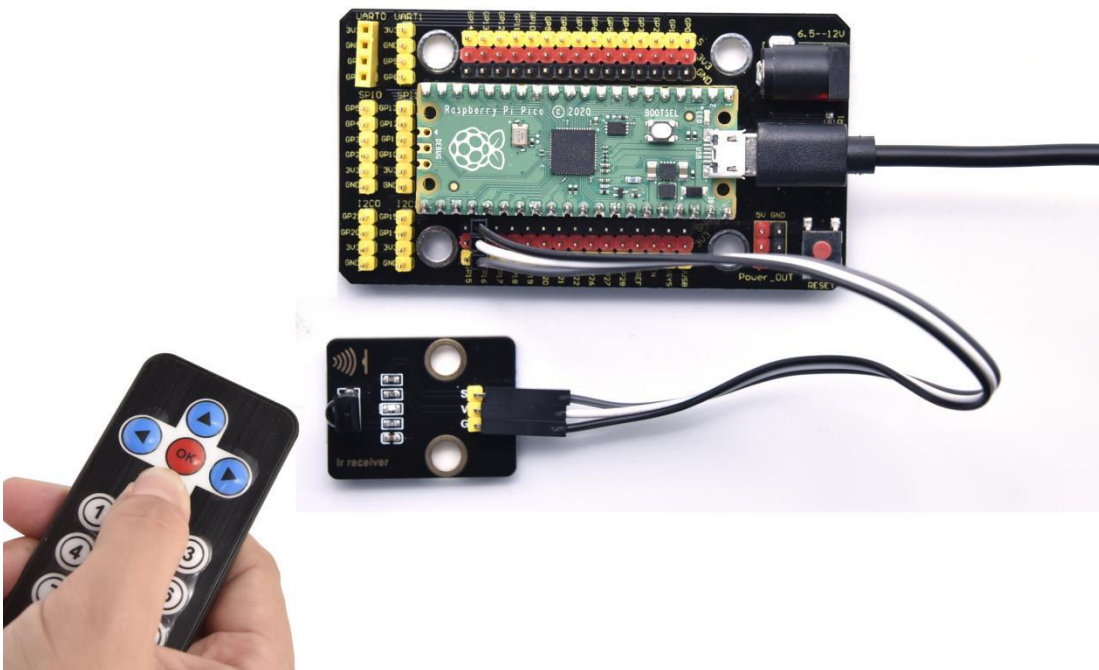
**utime.sleep(0.5)**

## **Code Explanation**

**read\_ircode(ird)** corresponds to the key symbols

## Test Result

Get an IR remote control and pull out the insulation chip. Point at the IR receiver and press keys on the IR remote control. Then the LED on the IR receiver will flash, as shown below;



```
Shell X
```

```
>>> %Run -c $EDITOR_CONTENT  
  
Left  
Ok  
LLLLLHHHHHHHHHLLHLLLHLHLLHHHLLH  
Right  
Down  
1  
LLHHHLLHHH  
2  
  
3  
4
```



## Project 22: DS1307 Clock Module



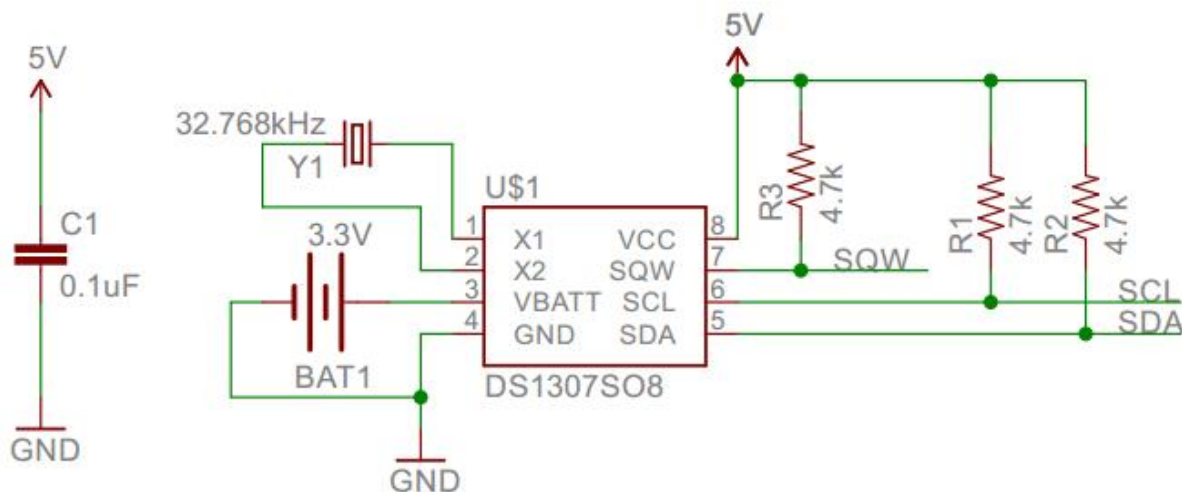
### Overview

The DS1307 serial real-time clock (RTC) is a low-power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. Address and data are transferred serially through an I2C, bidirectional bus.



The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power-sense circuit that detects power failures and automatically switches to the backup supply. Timekeeping operation continues while the part operates from the backup supply.

## Working Principle







## Detailed address and data:

Serial real-time clock records year, month, day, hour, minute, second and week; AM and PM indicate morning and afternoon respectively; 56 bytes of NVRAM store data; 2-wire serial port; programmable square wave output; power failure detection and automatic switching circuit; battery current is less than 500nA.

Pins description: X1, 32.768kHz crystal terminal ;

VBAT:X2: +3V input;

SDA: serial data;

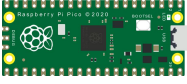
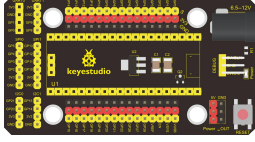
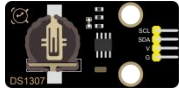


SCL: serial clock;

SQW/OUT: square waves/output drivers

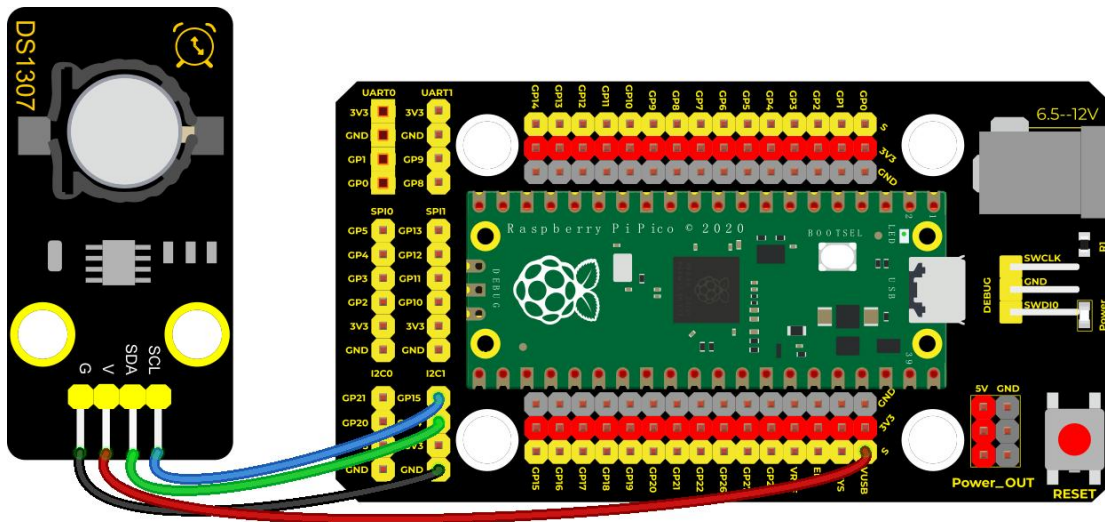
ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM 00–23
		24	PM/ AM							
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date		Date				Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year				Year				Year	00–99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh



## Components

				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio DS1307 Clock Module*1	4P Dupont Wire*1	Micro USB Cable*1

## Wiring Diagram



fritzing

VUSB is 5V, hence the power we use can be connected to VUSB



## Test Code

...

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 22

\* DS1307 Real Time Clock

\* <http://www.keyestudio.com>

...

```
from machine import I2C, Pin
```

```
from urtc import DS1307
```

```
import utime
```

```
i2c = I2C(1,scl = Pin(15),sda = Pin(14),freq = 400000)
```

```
rtc = DS1307(i2c)
```

```
year = int(input("Year : "))
```

```
month = int(input("month (Jan --> 1 , Dec --> 12): "))
```

```
date = int(input("date : "))
```

```
day = int(input("day (1 --> monday , 2 --> Tuesday ... 0 --> Sunday): "))
```

```
hour = int(input("hour (24 Hour format): "))
```

```
minute = int(input("minute : "))
```

```
second = int(input("second : "))
```



```
now = (year,month,date,day,hour,minute,second,0)

rtc.datetime(now)

#(year,month,date,day,hour,minute,second,p1) = rtc.datetime()

while True:

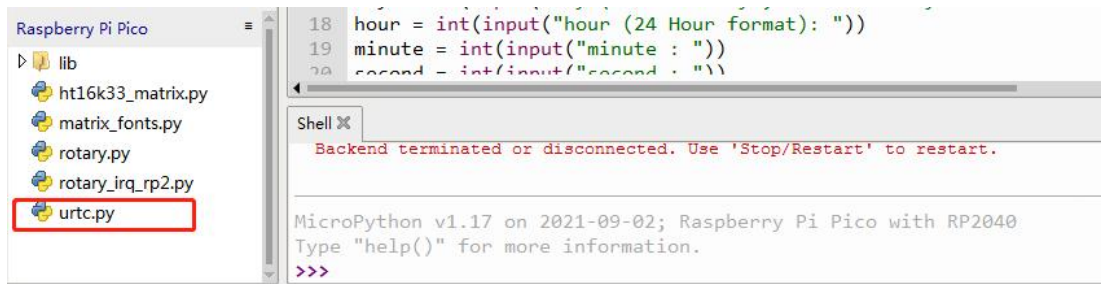
    DateTimeTuple = rtc.datetime()

    print(DateTimeTuple[0], end = '-')
    print(DateTimeTuple[1], end = '-')
    print(DateTimeTuple[2], end = ' ')
    print(DateTimeTuple[4], end = ':')
    print(DateTimeTuple[5], end = ':')
    print(DateTimeTuple[6], end = '  week:')
    print(DateTimeTuple[3])

    utime.sleep(1)
```

## Code Explanation

In the experiment, we need to import the urtc module.



```
18 hour = int(input("hour (24 Hour format): "))
19 minute = int(input("minute : "))
20 second = int(input("second : "))

Shell X
Backend terminated or disconnected. Use 'Stop/Restart' to restart.

MicroPython v1.17 on 2021-09-02; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
```

**rtc.datetime()**, when running the program, we set "input please" program, run the code, we need to input time and date, after inputting, the data will be displayed each second.

**DateTimeTuple[0]** save years

**DateTimeTuple[1]** save months

**DateTimeTuple[2]** save days

**DateTimeTuple[3]** Save weeks

**Rtc.GetDateTime().Month()** return months

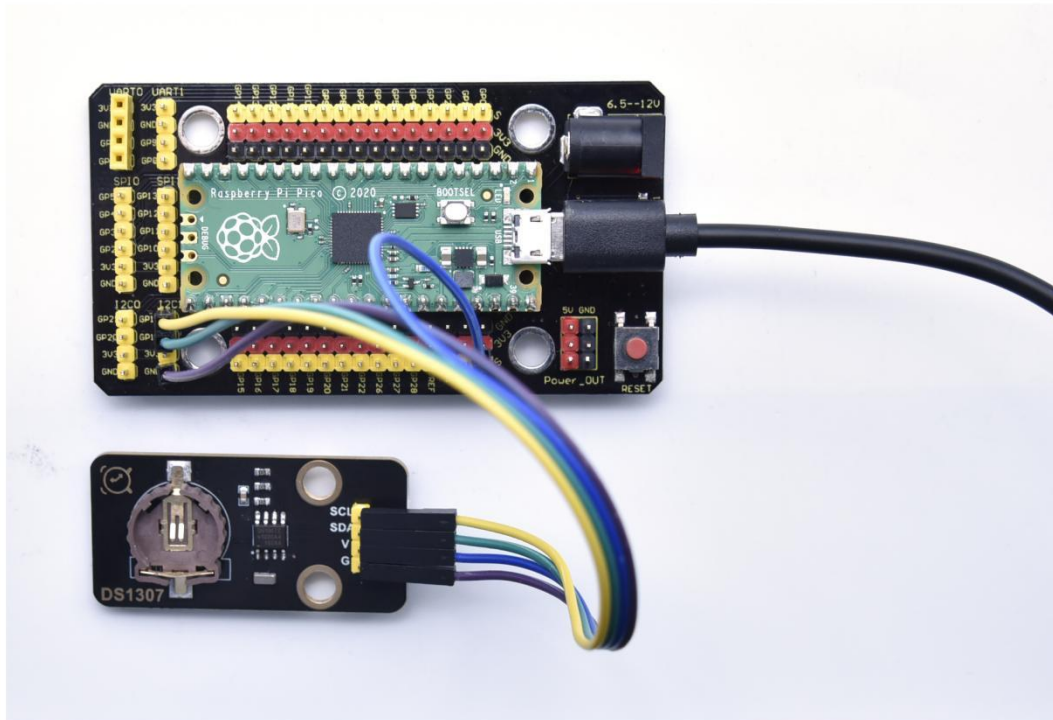
**DateTimeTuple[4]** save hours

**DateTimeTuple[5]** save minutes

**DateTimeTuple[6]** save seconds

## Test Result

Upload the test code, we can see the displayed year, month, day, hour, minute, second and week on the shell, as shown below;



```
Shell %  
date : 11  
day (1 --> monday , 2 --> Tuesday ... 0 --> Sunday): 4  
hour (24 Hour Format): 18  
minute : 56  
second : 50  
2021-11-11 18:56:50 week:4  
2021-11-11 18:56:51 week:4  
2021-11-11 18:56:52 week:4  
2021-11-11 18:56:53 week:4  
2021-11-11 18:56:54 week:4  
2021-11-11 18:56:55 week:4  
2021-11-11 18:56:56 week:4  
2021-11-11 18:56:57 week:4  
2021-11-11 18:56:58 week:4
```



## Project 23: TM1650 4-Digit Tube Display



### Overview

This module is mainly composed of a 0.36 inch red common anode 4-digit digital tube, and its driver chip is TM1650. When using it, we only need two signal lines to make the single-chip microcomputer control a 4-bit digit tube, which greatly saves the IO port resources of the control board.

TM1650 is a special circuit for LED (light emitting diode display) drive control. It integrates MCU input and output control digital interface, data latch, LED drivers, keyboard scanning, brightness adjustment and other circuits.

TM1650 has stable performance, reliable quality and strong





anti-interference ability.

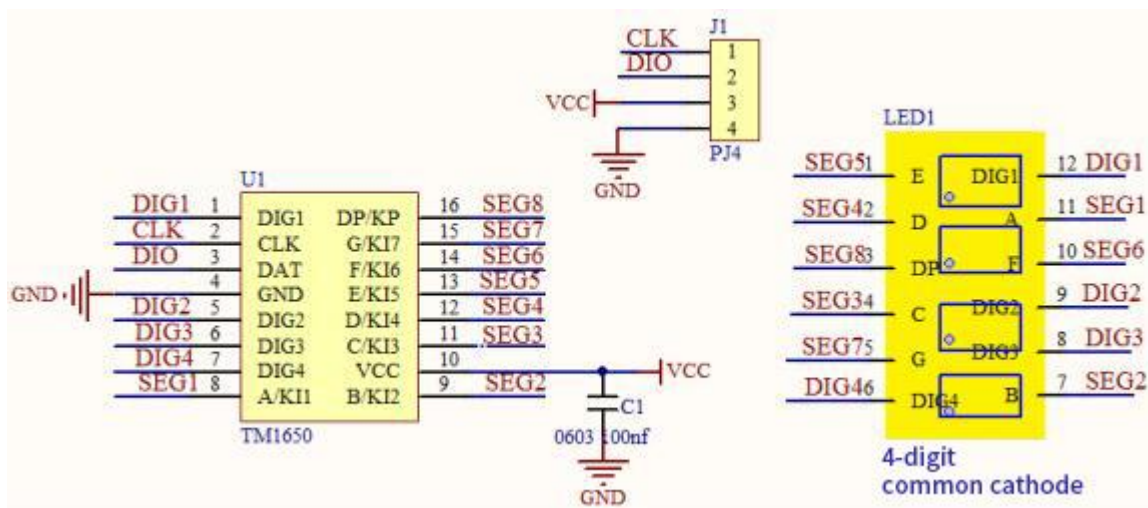
It can be applied to the application of long-term continuous working for 24 hours.

TM1650 uses 2-wire serial transmission protocol for communication (note that this data transmission protocol is not a standard I2C protocol). The chip can drive the digital tube and save MCU pin resources through two pins and MCU communication.

## Working Principle

TM1650 adopts IIC treaty and SDA and SCL wire

Data command setting is 0x48. This means that lighting up the tube display not perform its button scanning function.







**Data command setting:** 0x48 means that we light up the digital tube, instead of enable the function of key scanning

B7	B6	B5	B4	B3	B2	B1	B0	Function	Description
×	0	0	0		×	×		Brightness setting	Eight-level brightness
×	0	0	1		×	×			One-level brightness
×	0	1	0		×	×			Two-level brightness
×	0	1	1		×	×			Three-level brightness
×	1	0	0		×	×			Four-level brightness
×	1	0	1		×	×			Five-level brightness
×	1	1	0		×	×			Six-level brightness
×	1	1	1		×	×			Seven-level brightness
×				0	×	×		7/8 segment display control bit	8-segment display way
×				1	×	×			7-segment display way
×					×	×	0	ON/OFF display bit	Off display
×					×	×	1		On display

## Command display setting:

bit[6:4]: set the brightness of tube display, and 000 is brightest

bit[3]: set to show decimal points

bit[0]: start the display of the tube display

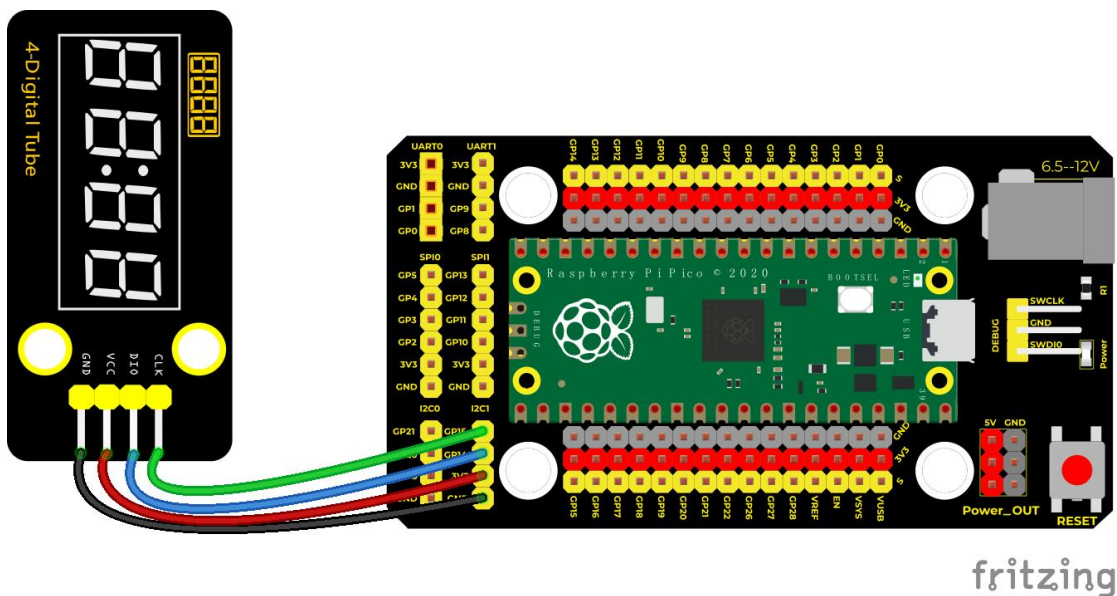
## Components





Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio TM1650 4-Digit Tube Display*1	4P Dupont Wire*1	Micro USB Cable*1
------------------------------	-------------------------------	---	---------------------	-------------------------

## Wiring Diagram



fritzing

## Test Code'''

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 23

\* TM1650 Four digital tube

\* <http://www.keyestudio.com>

'''

from machine import Pin

import time



**# definitions for TM1650**

**ADDR\_DIS = 0x48 #mode command**

**ADDR\_KEY = 0x49 #read key value command**

**# definitions for brightness**

**BRIGHT\_DARKEST = 0**

**BRIGHT\_TYPICAL = 2**

**BRIGHTTEST = 7**

**on = 1**

**off = 0**

**# number:0~9**

**NUM = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f]**

**# DIG = [0x68,0x6a,0x6c,0x6e]**

**DIG = [0x6e,0x6c,0x6a,0x68]**

**DOT = [0,0,0,0]**

**clkPin = 15**

**dioPin = 14**

**clk = machine.Pin(clkPin, machine.Pin.OUT)**



```
dio = machine.Pin(dioPin, machine.Pin.OUT)
```

```
DisplayCommand = 0
```

```
def writeByte(wr_data):
```

```
    global clk,dio
```

```
    for i in range(8):
```

```
        if(wr_data & 0x80 == 0x80):
```

```
            dio.value(1)
```

```
        else:
```

```
            dio.value(0)
```

```
        clk.value(0)
```

```
        time.sleep(0.0001)
```

```
        clk.value(1)
```

```
        time.sleep(0.0001)
```

```
        clk.value(0)
```

```
        wr_data <<= 1
```

```
    return
```

```
def start():
```

```
    global clk,dio
```

```
    dio.value(1)
```



**clk.value(1)**

**time.sleep(0.0001)**

**dio.value(0)**

**return**

**def ack():**

**global clk,dio**

**dy = 0**

**clk.value(0)**

**time.sleep(0.0001)**

**dio = Pin(dioPin, machine.Pin.IN)**

**while(dio.value() == 1):**

**time.sleep(0.0001)**

**dy += 1**

**if(dy>5000):**

**break**

**clk.value(1)**

**time.sleep(0.0001)**

**clk.value(0)**

**dio = Pin(dioPin, machine.Pin.OUT)**

**return**



**def stop():**

**global clk,dio**

**dio.value(0)**

**clk.value(1)**

**time.sleep(0.0001)**

**dio.value(1)**

**return**

**def displayBit(bit, num):**

**global ADDR\_DIS**

**if(num > 9 and bit > 4):**

**return**

**start()**

**writeByte(ADDR\_DIS)**

**ack()**

**writeByte(DisplayCommand)**

**ack()**

**stop()**

**start()**

**writeByte(DIG[bit-1])**

**ack()**

**if(DOT[bit-1] == 1):**



**writeByte(NUM[num] | 0x80)**

**else:**

**writeByte(NUM[num])**

**ack()**

**stop()**

**return**

**def clearBit(bit):**

**if(bit > 4):**

**return**

**start()**

**writeByte(ADDR\_DIS)**

**ack()**

**writeByte(DisplayCommand)**

**ack()**

**stop()**

**start()**

**writeByte(DIG[bit-1])**

**ack()**

**writeByte(0x00)**

**ack()**

**stop()**



**return**

**def setBrightness(b = BRIGHT\_TYPICAL):**

**global DisplayCommand,brightness**

**DisplayCommand = (DisplayCommand & 0x0f)+(b<<4)**

**return**

**def setMode(segment = 0):**

**global DisplayCommand**

**DisplayCommand = (DisplayCommand & 0xf7)+(segment<<3)**

**return**

**def displayOnOFF(OnOff = 1):**

**global DisplayCommand**

**DisplayCommand = (DisplayCommand & 0xfe)+OnOff**

**return**

**def displayDot(bit, OnOff):**

**if(bit > 4):**

**return**

**if(OnOff == 1):**





**DOT[bit-1] = 1;**

**else:**

**DOT[bit-1] = 0;**

**return**

**def InitDigitalTube():**

**setBrightness(2)**

**setMode(0)**

**displayOnOFF(1)**

**for \_ in range(4):**

**clearBit(\_)**

**return**

**def ShowNum(num): #0~9999**

**displayBit(1,num%10)**

**if(num < 10):**

**clearBit(2)**

**clearBit(3)**

**clearBit(4)**

**if(num > 9 and num < 100):**

**displayBit(2,num//10%10)**

**clearBit(3)**



```
clearBit(4)
```

```
if(num > 99 and num < 1000):
```

```
    displayBit(2,num//10%10)
```

```
    displayBit(3,num//100%10)
```

```
clearBit(4)
```

```
if(num > 999 and num < 10000):
```

```
    displayBit(2,num//10%10)
```

```
    displayBit(3,num//100%10)
```

```
    displayBit(4,num//1000)
```

```
InitDigitalTube()
```

```
while True:
```

```
    #displayDot(1,on)      # on or off, DigitalTube.Display(bit,number);
```

```
bit=1---4  number=0---9
```

```
for i in range(0,999):
```

```
    ShowNum(i)
```

```
    time.sleep(0.01)
```

**Code Explanation**  
**clkPin = 15、 dioPin = 14**

Set pins, that is, CLK is connected to GP15, DIO is linked with GOP14 and



we can set any pins.

**displayBit(bit, num):** show bit(1~4) and display number 显示数字  
num(0~9)

**clearBit(bit)** clear up bit(1~4) display

**setBrightness():** brightness setting

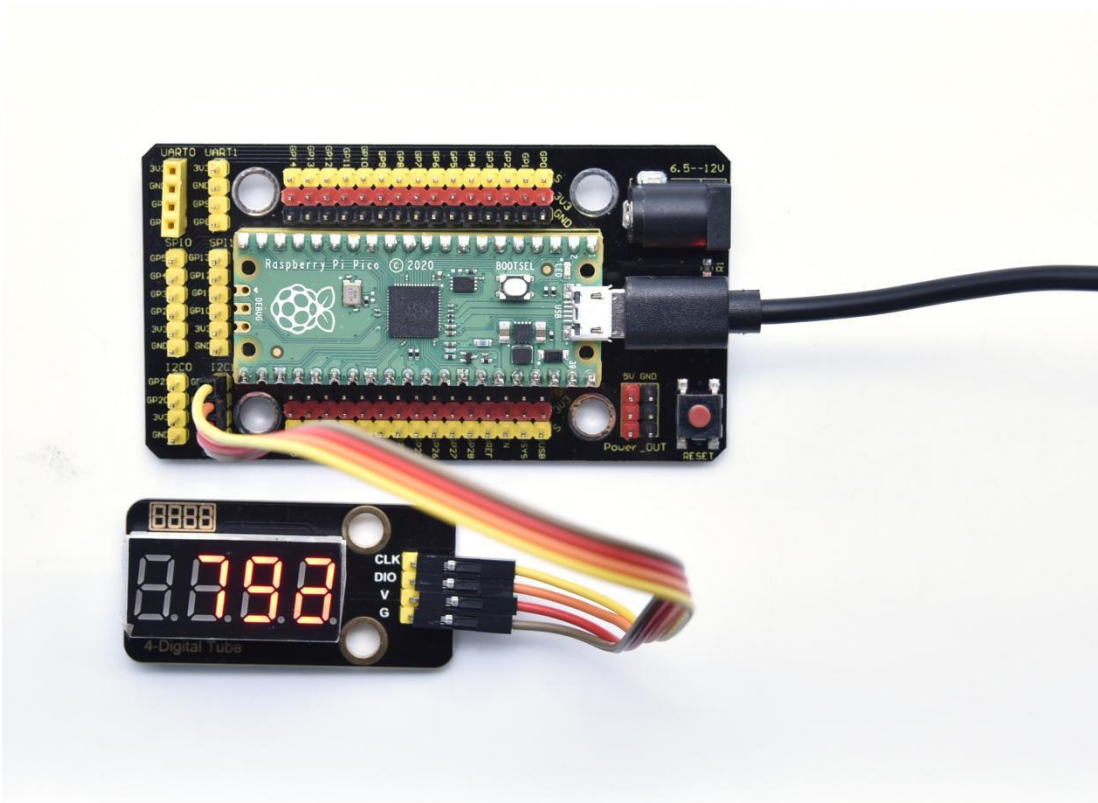
**displayOnOFF()** 0 means OFF, 1 stands for ON

**displayDot(bit, OnOff):** show bit, 0 is OFF and 1 is ON.

**ShowNum(num):** show integer num, in the range of 0~9999

## Test Result

Run the test code, wire up and power on. 4-digit tube display will show numbers from 0 to 99999 then from 9999 to 0



## Project 24: HT16K33\_8X8 Dot Matrix Module





## Overview

What is the dot matrix display?

The 8X8 dot matrix is composed of 64 light-emitting diodes, and each light-emitting diode is placed at the intersection of the row line and the column line. When the corresponding row is set to 1 level, and a certain column is set to 0 level, the corresponding diode will light up.

## Working Principle

As the schematic diagram shown, to light up the LED at the first row and column, we only need to set C1 to high level and R1 to low level. To turn on LEDs at the first row, we set R1 to low level and C1-C8 to high level.

16 IO ports are needed, which will highly waste the MCU resources.

Therefore, we designed this module, using the HT16K33 chip to drive an 8\*8 dot matrix, which greatly saves the resources of the single-chip microcomputer.

There are three DIP switches on the module, all of which are set to I2C communication address. The setting method is shown below.

A0, A1 and A2 are grounded, that is, the address is 0x70


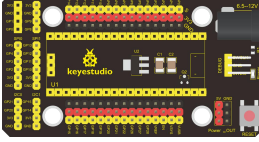
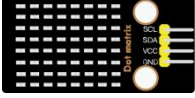


A	A	A	A	A	A	A	A	A
0 (1)	1 (2)	2 (3)	0 (1)	1 (2)	2 (3)	0 (1)	1 (2)	2 (3)
0	0	0	1	0	0	0	1	0



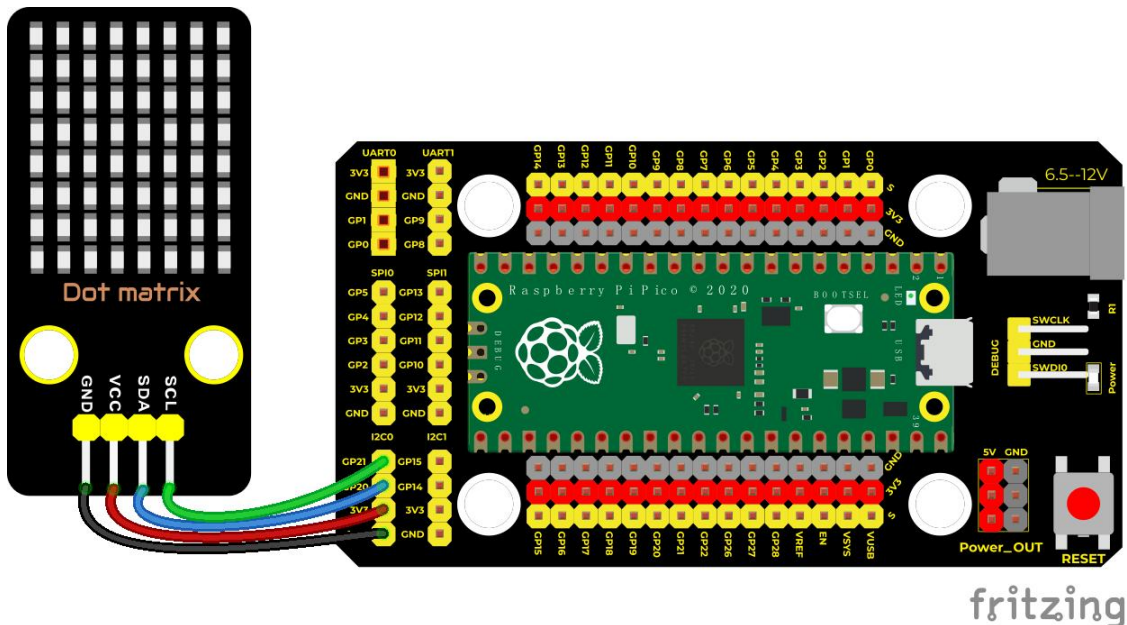
( OF F)	( OF F)	( OF F)	( O N)	( OF F)	( OF F)	( OF F)	( O N)	( OF F)
OX70			OX71			OX72		
A 0 (1)	A 1 (2)	A 2 (3)	A 0 (1)	A 1 (2)	A 2 (3)	A 0 (1)	A 1 (2)	A 2 (3)
1 ( O N)	1 ( O N)	0 ( OF F)	0 ( OF F)	0 ( OF F)	1 ( O N)	1 ( O N)	0 ( OF F)	1 ( O N)
OX73			OX74			OX75		
A 0 (1)	A 1 (2)	A 2 (3)	A 0 (1)	A 1 (2)	A 2 (3)			
0 ( OF F)	1 ( O N)	1 ( O N)	1 ( O N)	1 ( O N)	1 ( O N)			
OX76			OX77					



## Components

				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio HT16K33_ 8X8 Dot Matrix*1	4P Dupont Wire*1	Micro USB Cable*1

## Wiring Diagram



## Test Code'''

\* [Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico](#)

\* [lesson 24](#)



\* HT16K33 8\*8 dot matrix

\* <http://www.keyestudio.com>

...

import machine

import time

import json

import matrix\_fonts

from ht16k33\_matrix import ht16k33\_matrix

## Tool To Make Sprites <https://gurgleapps.com/tools/matrix>

#i2c config

clock\_pin = 21

data\_pin = 20

bus = 0

i2c\_addr\_left = 0x70

use\_i2c = True

def scan\_for\_devices():

    i2c = machine.I2C()

    machine.I2C(bus,sda=machine.Pin(data\_pin),scl=machine.Pin(clock\_pin))

    devices = i2c.scan()

    if devices:





```
    for d in devices:
```

```
        print(hex(d))
```

```
    else:
```

```
        print('no i2c devices')
```

```
if use_i2c:
```

```
    scan_for_devices()
```

```
    left_eye = ht16k33_matrix(data_pin, clock_pin, bus, i2c_addr_left)
```

```
def show_char(left):
```

```
    if use_i2c:
```

```
        left_eye.show_char(left)
```

```
def scroll_message(font,message='hello',delay=0.05):
```

```
    left_message = ' ' + message
```

```
    right_message = message + ' '
```

```
    length=len(right_message)
```

```
    char_range=range(length-1)
```

```
    for char_pos in char_range:
```

```
        right_left_char=font[right_message[char_pos]]
```

```
        right_right_char=font[right_message[char_pos+1]]
```

```
        left_left_char=font[left_message[char_pos]]
```



```
left_right_char=font[left_message[char_pos+1]]
```

```
for shift in range(8):
```

```
    left_bytes=[0,0,0,0,0,0,0,0]
```

```
    right_bytes=[0,0,0,0,0,0,0,0]
```

```
    for col in range(8):
```

```
        left_bytes[col]=left_bytes[col]|left_left_char[col]<<shift
```

```
        left_bytes[col]=left_bytes[col]|left_right_char[col]>>8-shift;
```

```
        right_bytes[col]=right_bytes[col]|right_left_char[col]<<shift
```

```
right_bytes[col]=right_bytes[col]|right_right_char[col]>>8-shift;
```

```
    if use_i2c:
```

```
        left_eye.show_char(left_bytes)
```

```
    time.sleep(delay)
```

```
while True:
```

```
    show_char(matrix_fonts.textFont1['A'])
```

```
    time.sleep(1)
```

```
    show_char(matrix_fonts.textFont1['B'])
```

```
    time.sleep(1)
```

```
    show_char(matrix_fonts.textFont1['C'])
```

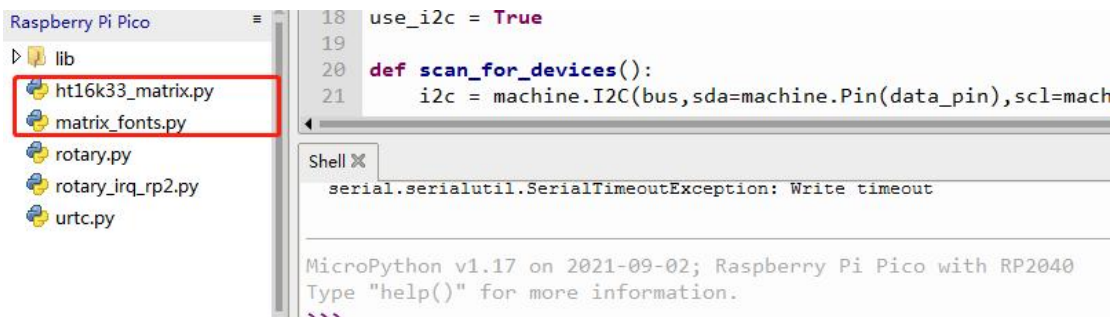
```
    time.sleep(1)
```



```
scroll_message(matrix_fonts.textFont1, ' Hello World ')
```

## Code Explanation

Firstly we need to import the dot matrix module. `matrix_fonts` is the module file for all kinds of characters.

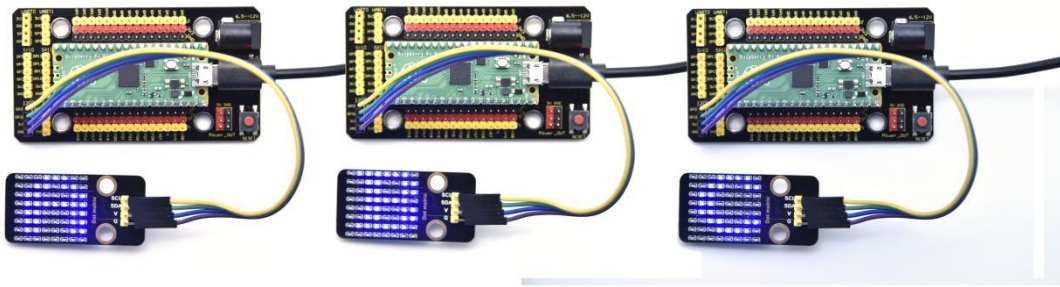


`show_char()` is a displayed character, for instance, `show_char(matrix_fonts.textFont1['A'])` represents displaying A.

`scroll_message(font,message='hello',delay=0.05):` scroll to display, 0.05 is the speed, message means displayed character strings, font is a model file.

## Test Result

Wire up and upload the test code. Then the dot matrix display will show "A" , "B" and "C" then "Hello World" .



## 5. Comprehensive Experiments

The previous projects are related to single sensor or module. In the following part, we will combine various sensors and modules to create some comprehensive experiments to perform special functions.

### Project 25: Breathing LED

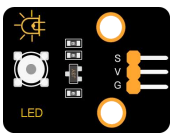





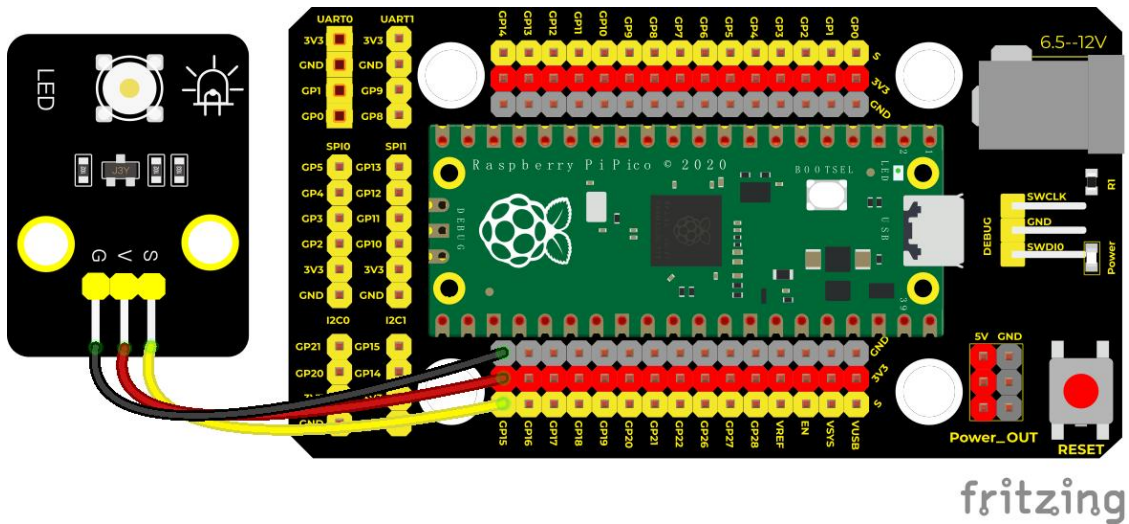
## Overview

A “breathing LED” is a phenomenon where an LED's brightness smoothly changes from dark to bright and back to dark, continuing to do so and giving the illusion of an LED “breathing.” This phenomenon is similar to a lung breathing in and out. So how to control LED’ s brightness? We need to take advantage of PWM.

## Components

				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio White LED Module*1	3P Dupont Wire*1	Micro USB Cable*1

## Wiring Diagram



## Test Code

...

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 25

\* Breath

\* <http://www.keyestudio.com>

...

```
import machine
```

```
import time
```

```
pwm = machine.PWM(machine.Pin(15))
```

```
pwm.freq(1000)
```

```
duty = 0
```



```
direction = 1
```

```
while True:
```

```
    duty += direction
```

```
    if duty > 255:
```

```
        duty = 255
```

```
        direction = -1
```

```
    elif duty < 0:
```

```
        duty = 0
```

```
        direction = 1
```

```
    pwm.duty_u16(duty * duty)
```

```
    time.sleep(0.01)
```

## **Code Explanation**

The bigger the duty cycle is set, the brighter the LED. The maximum is 65535. When duty increases from 0 to 255, up 1 and delay 10ms for each time, then LED will gradually get bright. When PWM is 255\*255, it decreases from 255 to 0, down by 1 and delay 10ms for each time, then the LED will get dimmer, like human breathe.

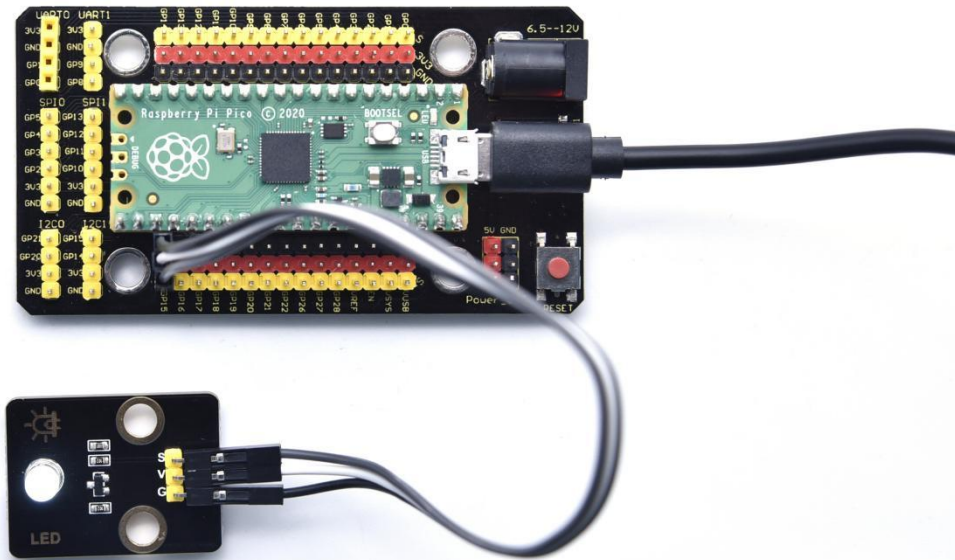
Also, we can change the time of getting dimmer or brighter in the code.

Another way is changing step length like `direction = -2` or `direction = 2`.



## Test Result

Run the test code, the LED on the module gradually gets dimmer then brighter, cyclically, like human breathe







## Project 26: Button-controlled LED



### Overview

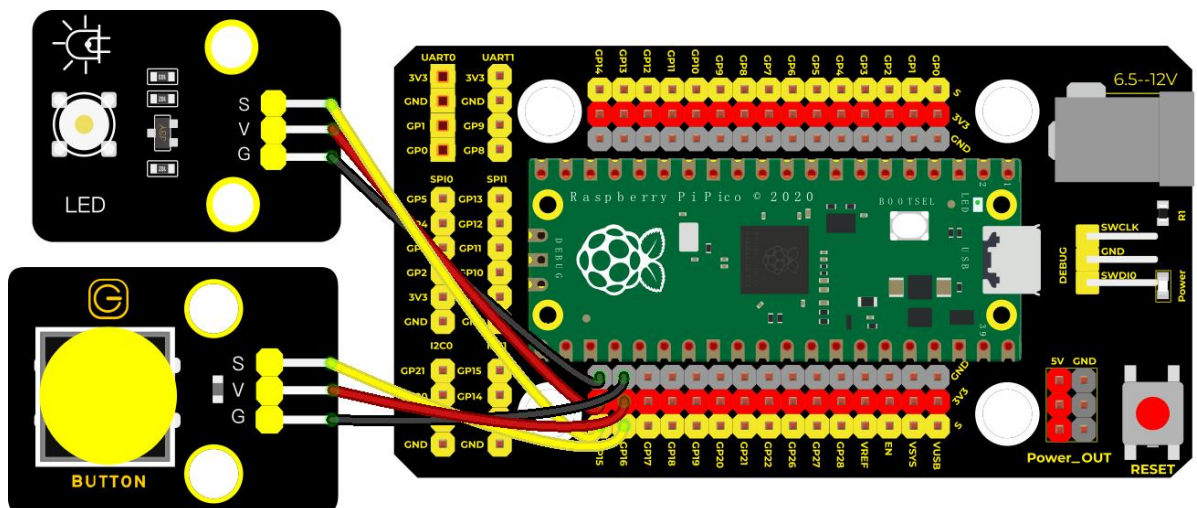
In this lesson, we will make an extension experiment with a button and an LED. When the button is pressed and low levels are output, the LED will light up; when the button is released, the LED will go off. Then we can control a module with another module.



## Components

					
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio White LED Module*1	Keyestudio DIY Button Module*1	3P Dupont Wire*2	Micro USB Cable*1

## Wiring Diagram



fritzing



## Test Code

...

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 26

\* button control LED

\* <http://www.keyestudio.com>

...

```
from machine import Pin
```

```
import time
```

```
button = Pin(16, Pin.IN)
```

```
LED = Pin(15, Pin.OUT)
```

```
touch = False
```

```
def toggle_handle(pin):
```

```
    global touch
```

```
    touch = not touch
```

```
button.irq(trigger = Pin.IRQ_FALLING, handler = toggle_handle)
```

```
while True:
```

```
    LED.value(touch)
```



**time.sleep(0.01)**

## Code Explanation

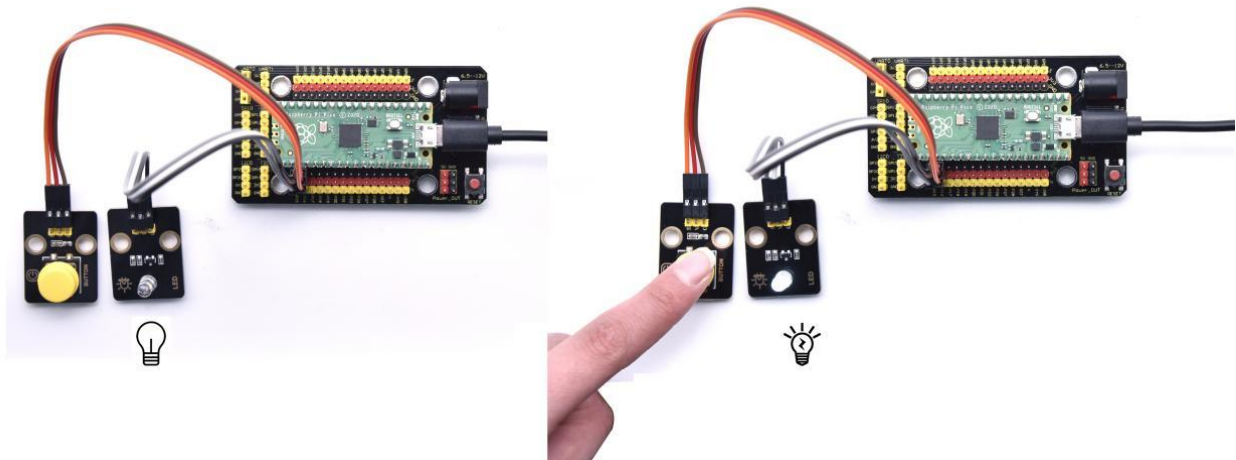
Configure pins mode according to IO ports connected to sensors and modules.

**button.irq(trigger = Pin.IRQ\_FALLING, handler = toggle\_handle)**

The trigger means that high levels turn into low levels and the trigger interrupt then use the interrupt function **toggle\_handle**.

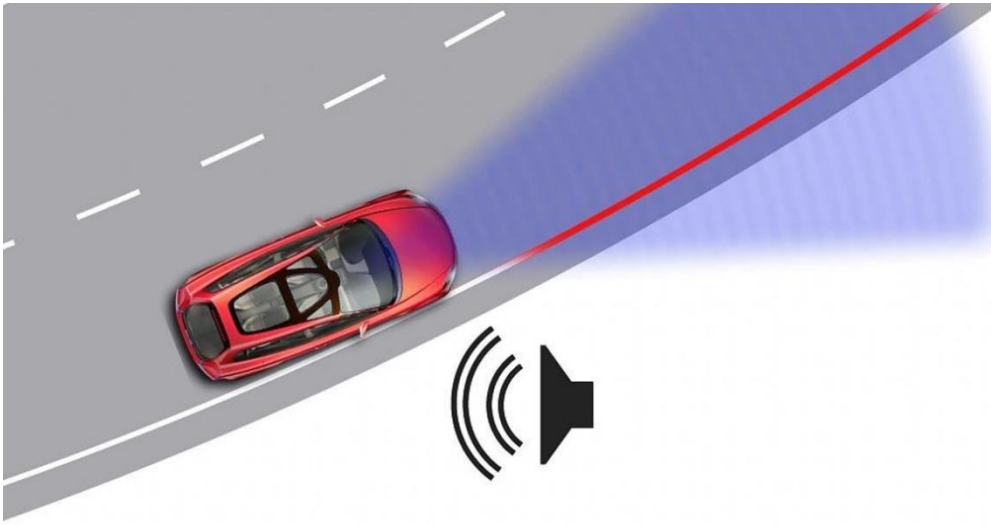
## Test Result

Upload the code, when the button is pressed, the LED will light up; when pressed again, the LED will go off





## Project 27: Alarm Experiment

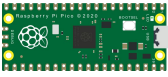
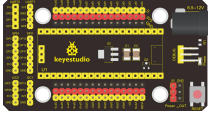
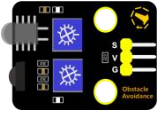





### Overview

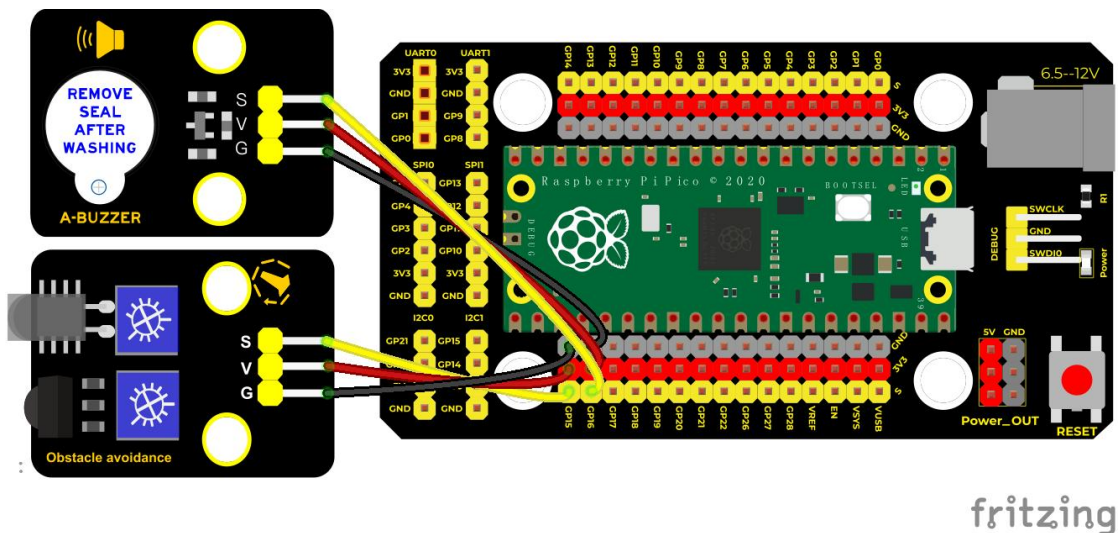
In the previous experiment, we control an output module through an input module. In this lesson, we will make an experiment that the active buzzer will emit sounds once an obstacle appears.



## Components

					
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio Obstacle Avoidance Sensor*1	Keyestudio Active Buzzer*1	3P Dupont Wire*2	Micro USB Cable*1

## Wiring Diagram



## Test Code

...



\* **Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico**

\* **lesson 27**

\* **Avoiding alarm**

\* **<http://www.keyestudio.com>**

...

```
from machine import Pin
```

```
import time
```

```
buzzer = Pin(16, Pin.OUT)
```

```
sensor = Pin(15, Pin.IN)
```

```
while True:
```

```
    buzzer.value(not(sensor.value()))
```

```
    time.sleep(0.01)
```

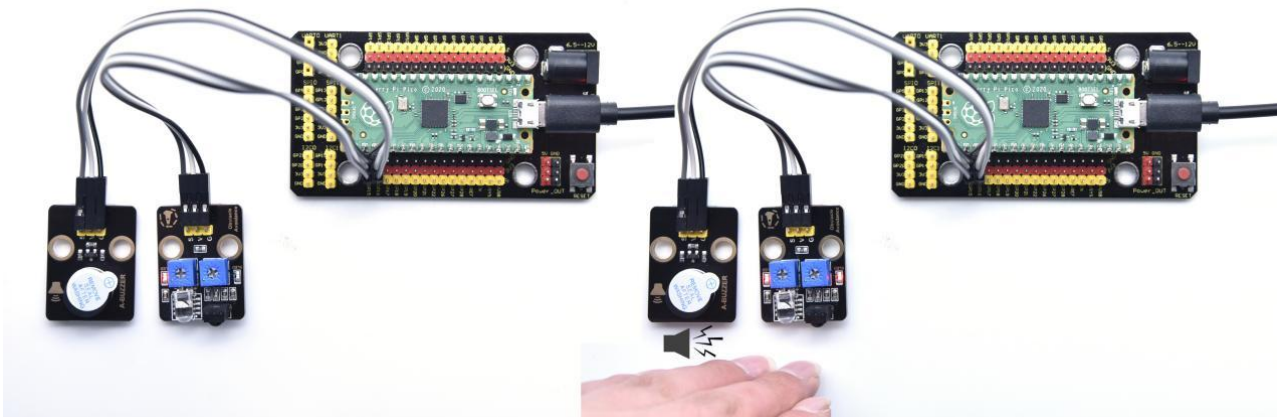
## **Code Explanation**

When detecting the obstacle, **sensor.value()** will return a low level signal. The pin GP16 of the buzzer will output high levels and the buzzer will emit sounds.



## Test Result

Upload the test code, if the obstacle is detected, the external active buzzer will chime; if not, it won' t beep








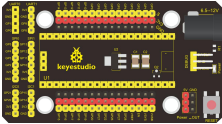
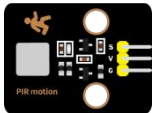



## Project 28: PIR Motion Sensor



### Introduction

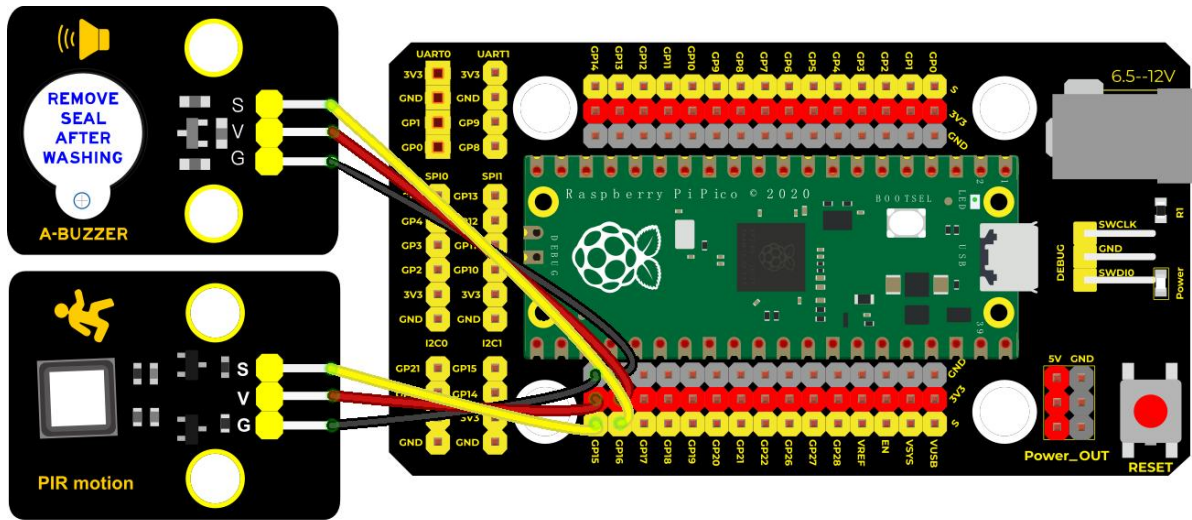
In this experiment, we will control an active buzzer and an on-board LED through a PIR motion sensor.

### Components

					
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio PIR Motion Sensor*1	Keyestudio Active Buzzer*1	3P Dupont Wire*2	MicroUSB Cable*1



## Connection Diagram



fritzing

## Test Code

...

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 28

\* PIR motion sensor

\* <http://www.keyestudio.com>

...

```
import machine
```

```
import utime
```

```
sensor_pir = machine.Pin(15, machine.Pin.IN,  
machine.Pin.PULL_DOWN)
```



```
led = machine.Pin(25, machine.Pin.OUT)
```

```
buzzer = machine.Pin(16, machine.Pin.OUT)
```

```
def pir_handler(pin):
```

```
    utime.sleep_ms(100)
```

```
    if pin.value():
```

```
        print("Warning! Intrusion detected! ")
```

```
        buzzer.value(1)
```

```
        for i in range(20):
```

```
            led.toggle()
```

```
            utime.sleep_ms(100)
```

```
sensor_pir.irq(trigger=machine.Pin.IRQ_RISING, handler=pir_handler)
```

```
while True:
```

```
    led.toggle()
```

```
    buzzer.value(0)
```

```
    utime.sleep(2)
```

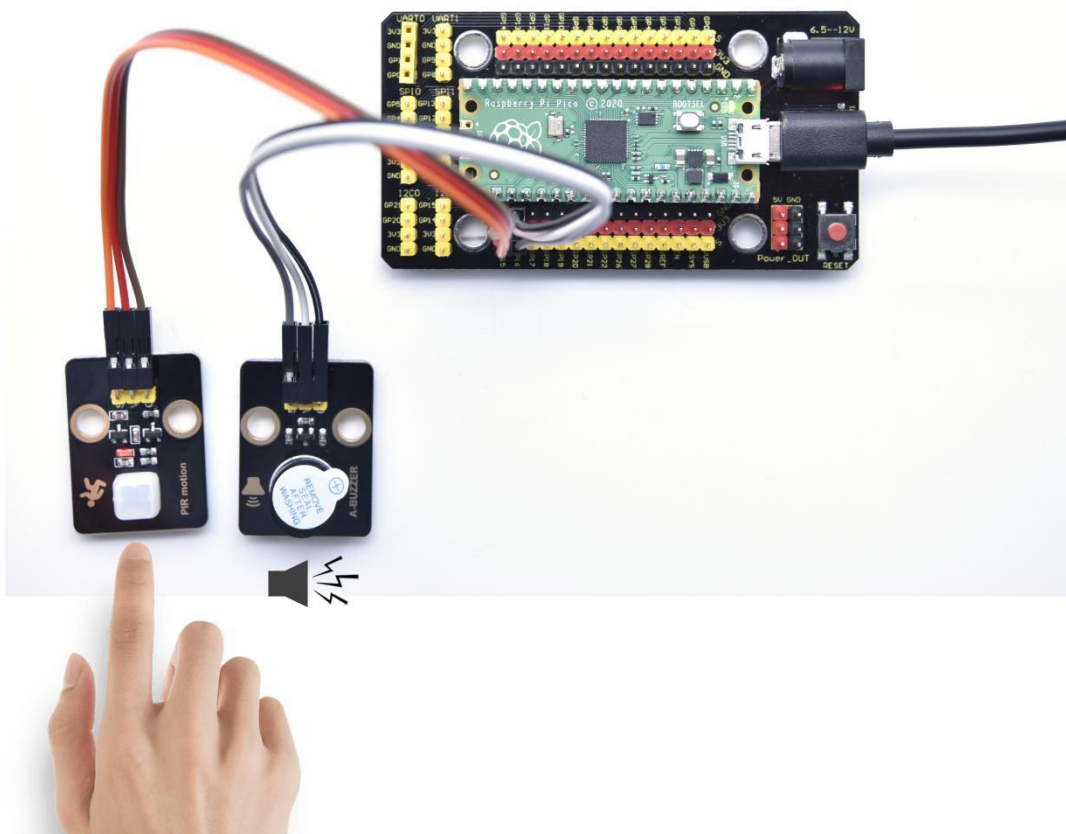
## Code Explanation



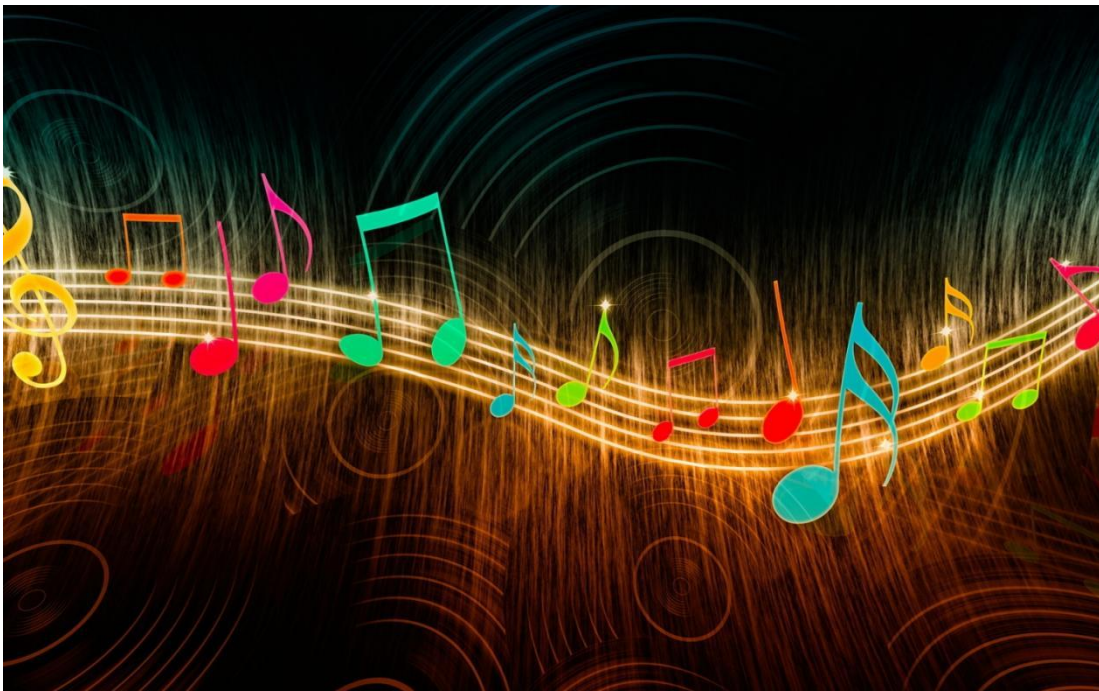
We use **sensor\_pir.irq(trigger=machine.Pin.IRQ\_RISING, handler=pir\_handler)** to trigger on a rising edge (when it goes low to high). **pir\_handler** is an interrupt handler used to control the buzzer and the LED.

## Test Result

After running the program, the LED will blink slowly and the detector starts to work. **IRQ\_RISING** is used when the interrupt triggers. When the motion is detected, the level on the PIR output will change from 0 to 1. Then the **pir\_handler()** function is called, the buzzer generates sounds and the LED flashes quickly.



## Project 29: Speaker Module





## Introduction

We learned about controlling the speaker module to make sounds, play beats and adjust its volume. In fact, each song is a combination of specific beats and tones (frequencies). In this experiment, we use this speaker module to play a song.

The frequency of each tone is shown below.

Bass:

Key Note	1 <sub>#</sub>	2 <sub>#</sub>	3 <sub>#</sub>	4 <sub>#</sub>	5 <sub>#</sub>	6 <sub>#</sub>	7 <sub>#</sub>
A	221	248	278	294	330	371	416
B	248	278	294	330	371	416	467
C	131	147	165	175	196	221	248
D	147	165	175	196	221	248	278
E	165	175	196	221	248	278	312
F	175	196	221	234	262	294	330



G	196	221	234	262	294	330	371
---	-----	-----	-----	-----	-----	-----	-----

Midrange :

Key Note	1	2	3	4	5	6	7
A	441	495	556	589	661	724	833
B	495	556	624	661	724	833	935
C	262	294	330	350	393	441	495
D	294	330	350	393	441	495	556
E	330	350	393	441	495	556	624
F	350	393	441	495	556	624	661
G	393	441	495	556	624	661	724



Treble:

Key Note	1 <sup>#</sup>	2 <sup>#</sup>	3 <sup>#</sup>	4 <sup>#</sup>	5 <sup>#</sup>	6 <sup>#</sup>	7 <sup>#</sup>
A	882	990	1112	1178	1322	1484	1665
B	990	1112	1178	1322	1484	1665	1869
C	525	589	661	700	786	882	990
D	589	661	700	786	882	990	1112
E	661	700	786	882	990	1112	1248
F	700	786	882	935	1049	1178	1322
G	786	882	990	1049	1178	1322	1484


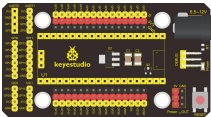
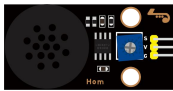


Beats are the time delay for each note. The larger the number, the longer the delay time. A note without a line in the spectrum is a beat, with a delay of 1s. while a beat with an underline is 1/2 of a beat without a line, with a delay of 0.5s, and a beat with two underlines is 1/4 of a beat without a line, with a delay of 0.25s. The 1/8 of a beat is with a delay of 0.125s.

We will take Happy Birthday Song as an example.

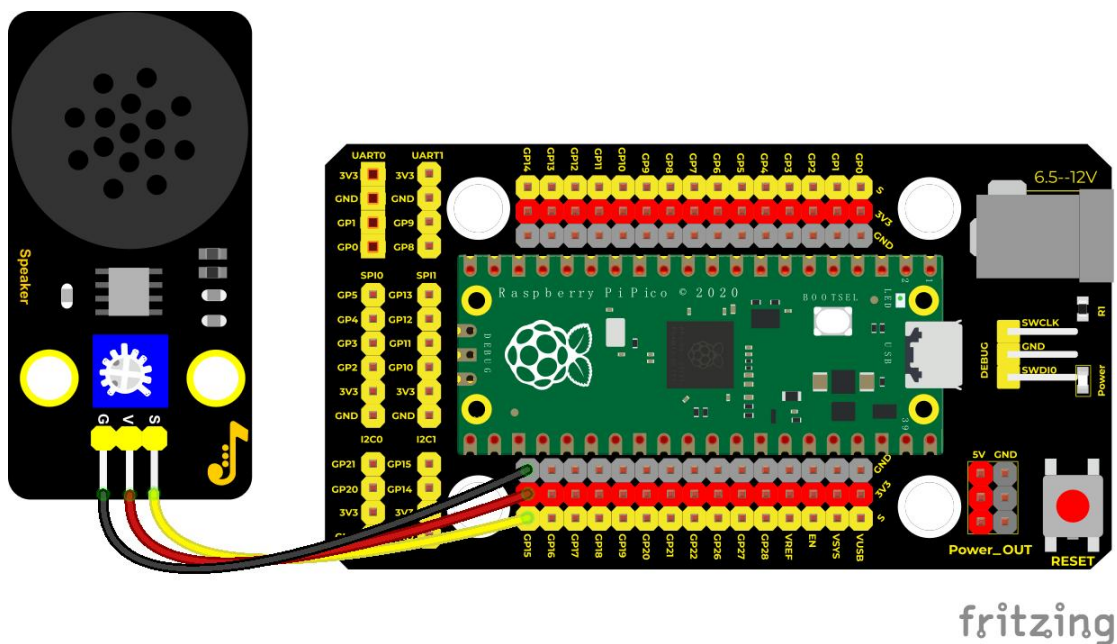




## Components

				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio Speaker Module*1	3P Dupont Wire*1	MicroUSB Cable*1

## Connection Diagram



## Test Code

'''

- \* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico
- \* lesson 29
- \* Speaker module



```
* http://www.keyestudio.com
```

```
'''
```

```
from machine import Pin, PWM
```

```
from utime import sleep
```

```
buzzer = PWM(Pin(15))
```

```
tones = {
```

```
"D1": 262,
```

```
"D2": 293,
```

```
"D3": 329,
```

```
"D4": 349,
```

```
"D5": 392,
```

```
"D6": 440,
```

```
"D7": 494,
```

```
"M1": 523,
```

```
"M2": 586,
```

```
"M3": 658,
```

```
"M4": 697,
```

```
"M5": 783,
```

```
"M6": 879,
```

```
"M7": 987,
```

```
"H1": 1045,
```

```
"H2": 1171,
```

```
"H3": 1316,
```

```
"H4": 1393,
```

```
"H5": 1563,
```

```
"H6": 1755,
```

```
"H7": 1971
```

```
}
```

```
song = ["D5", "D5", "D6", "D5", "M1", "D7",
```

```
        "D5", "D5", "D6", "D5", "M2", "M1",
```

```
        "D5", "D5", "M5", "M3", "M1", "D7", "D6",
```

```
        "M4", "M4", "M3", "M1", "M2", "M1"
```

```
]
```

```
durt = [0.25, 0.25, 0.5, 0.5, 0.5, 1,
```

```
        0.25, 0.25, 0.5, 0.5, 0.5, 1,
```

```
        0.25, 0.25, 0.5, 0.5, 0.5, 0.5, 0.5,
```

```
        0.25, 0.25, 0.5, 0.5, 0.5, 1
```

```
]
```

```
def playtone(frequency):
```

```
    buzzer.duty_u16(1000)
```



```
buzzer.freq(frequency)
```

```
def bequiet():  
    buzzer.duty_u16(0)
```

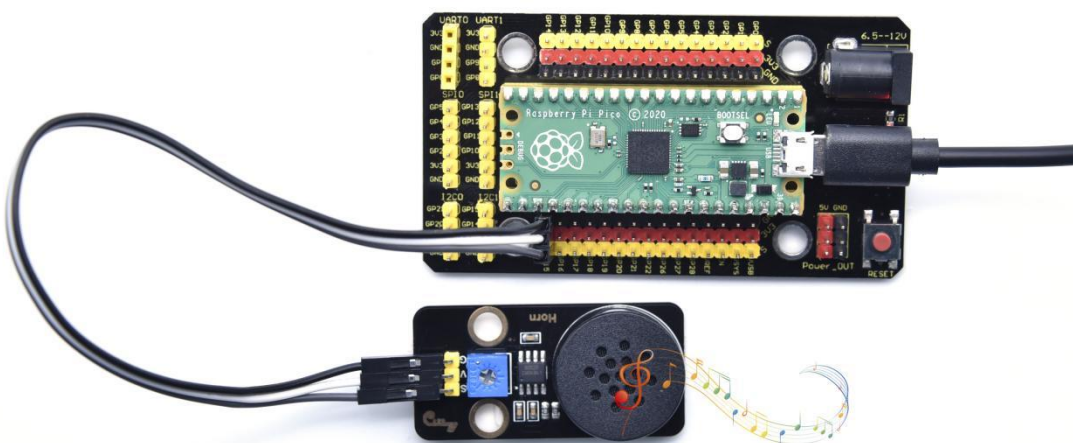
```
def playsong(my song):  
    for i in range(len(my song)):  
        playtone(tones[my song[i]])  
        sleep(durt[i])  
    bequiet()  
playsong(song)
```

## Code Explanation

We first list all the frequencies in D, then we list the frequencies and the beats according to the numbered musical notation. We use a beat of 500ms, which can be adjusted by yourself.

## Test Result

Connect the components according to the connection diagram and run the test code, the speaker module will play a song.





## Project 30: Rotary Encoder



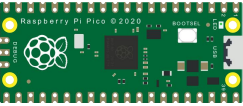

### Introduction

In this lesson, we will control the LED on the RGB module to show different colors through a rotary encoder.

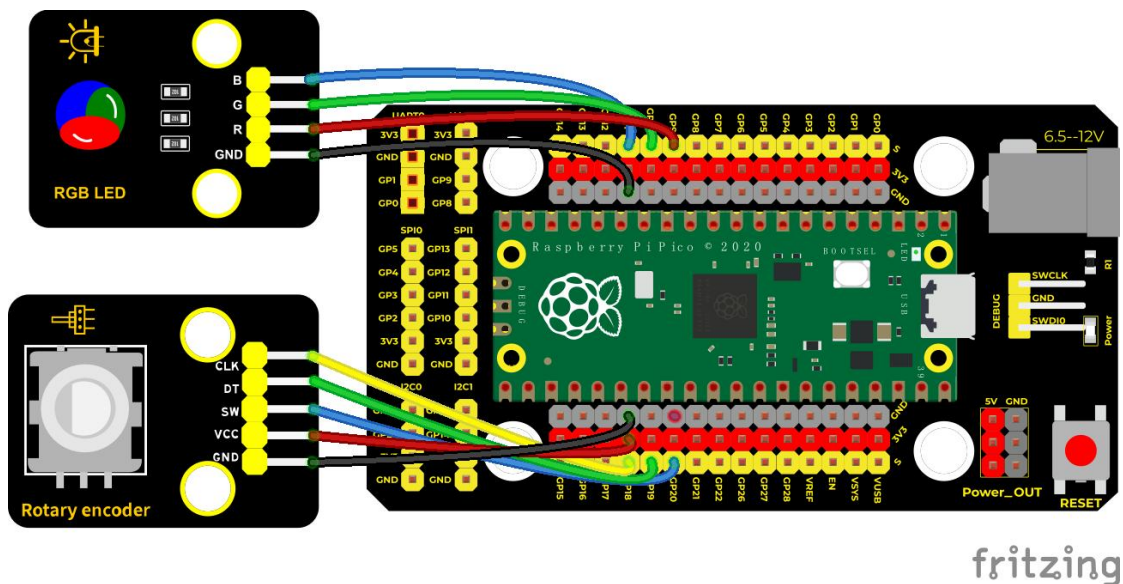
When designing the code, we need to divide the obtained values by 3 to get the remainders. The remainder is 0 and the LED will become red. The remainder is 1, the LED will become green. The remainder is 2, the LED will turn blue.

### Components



			
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio Common Cathode RGB Module*1	Keyestudio Rotary Encoder Module*1
			
5P Dupont Wire*1	4P Dupont Wire*1	Micro USB Cable*1	

## Connection Diagram



## Test Code

...



\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico

\* lesson 30

\* Rotary encoder

\* <http://www.keyestudio.com>

'''

```
import time
```

```
from rotary_irq_rp2 import RotaryIRQ
```

```
from machine import Pin, PWM
```

```
pwm_r = PWM(Pin(9))
```

```
pwm_g = PWM(Pin(10))
```

```
pwm_b = PWM(Pin(11))
```

```
pwm_r.freq(1000)
```

```
pwm_g.freq(1000)
```

```
pwm_b.freq(1000)
```

```
def light(red, green, blue):
```

```
    pwm_r.duty_u16(red)
```

```
    pwm_g.duty_u16(green)
```

```
    pwm_b.duty_u16(blue)
```

```
SW=Pin(20,Pin.IN,Pin.PULL_UP)
```

```
r = RotaryIRQ(pin_num_clk=18,  
              pin_num_dt=19,  
              min_val=0,  
              reverse=False,  
              range_mode=RotaryIRQ.RANGE_UNBOUNDED)
```

```
while True:
```

```
    val = r.value()
```

```
    print(val%3)
```

```
    if val%3 == 0:
```

```
        light(65535, 0, 0)
```

```
    elif val%3 == 1:
```

```
        light(0, 65535, 0)
```

```
    elif val%3 == 2:
```

```
        light(0, 0, 65535)
```

```
    time.sleep(0.1)
```

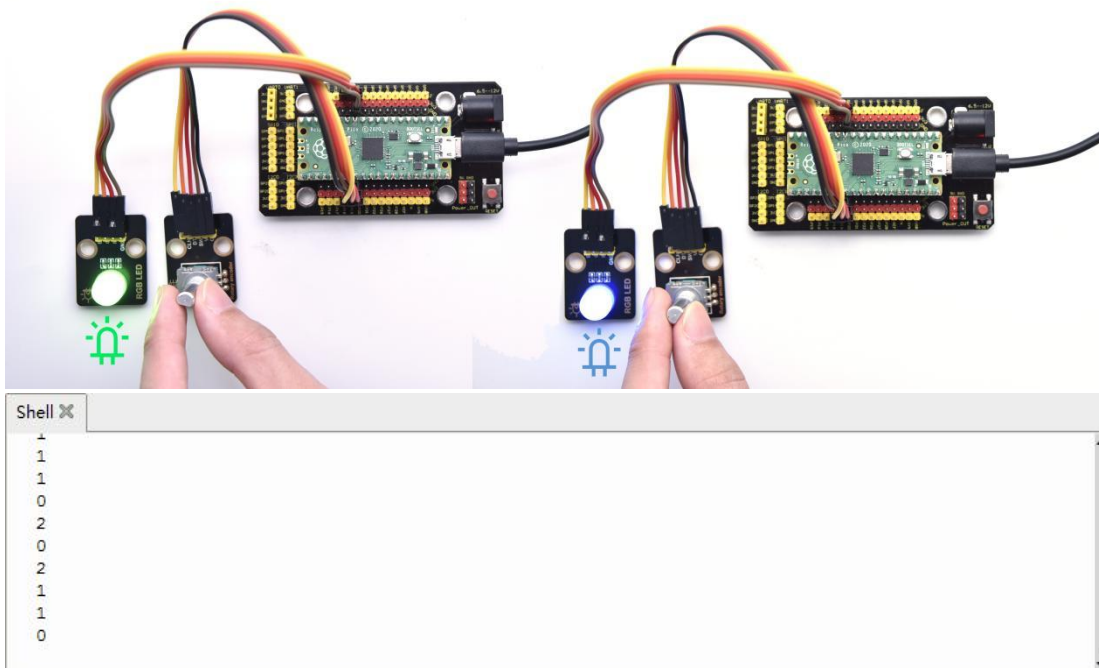
## Code Explanation



Any number divided by 3, the remainder obtained is 0, or 1, or 2, we can use these three values to determine the status of the LED. Set the pins of reminders to GP9 (red), GP10 (green) and GP11 (blue), respectively. Refer to the previous method to control the LED to show the corresponding colors.

## Test Result

Wire up the components, run the code and look at the Shell. Turn the encoder to display the reminders, then the LED will show different colors.







## Project 31: Rotary Potentiometer



### Introduction

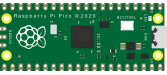
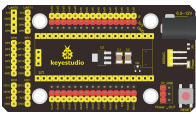
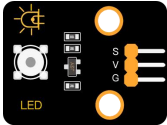
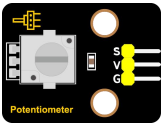


In the previous courses, we did experiments of breathing light and controlling LED with button. In this course, we do these two experiments by controlling the brightness of LED through an adjustable potentiometer. The brightness of LED is controlled by PWM values, and the range of analog values is the same as the PWM' s, from 0 to 65535.

After the code is set successfully, we can control the brightness of the LED on the module by rotating the potentiometer.

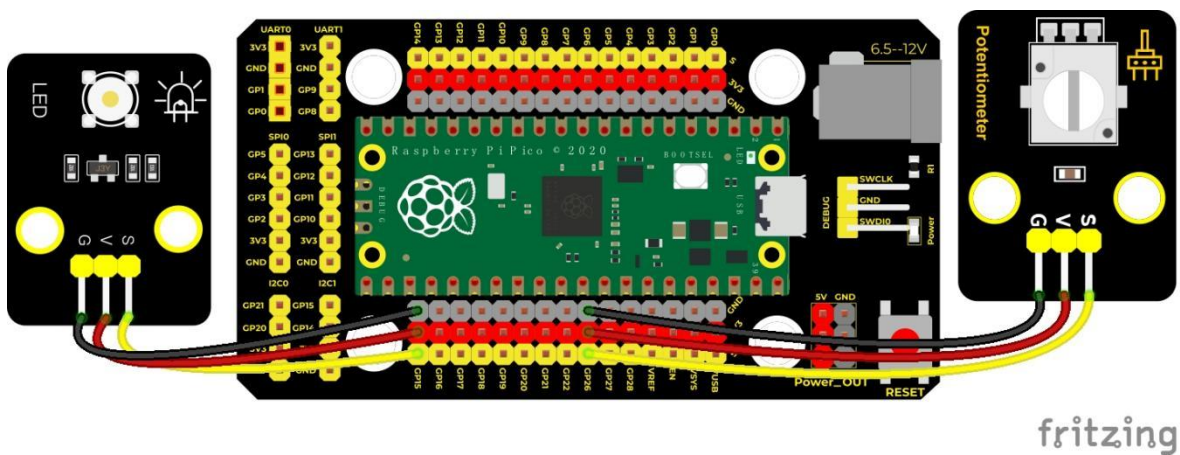
### Components





					
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio White LED Module*1	Keyestudio Rotary Potentiometer*1	3P Dupont Wire*2	MicroUSB Cable*1

## Connection Diagram



fritzing

## Test Code

```
'''
* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico
* lesson 31
* Rotary potentiometer
* http://www.keyestudio.com
'''
```

```
import machine
import utime
```



```
potentiometer = machine.ADC(26)
```

```
pwm = machine.PWM(machine.Pin(15))
```

```
pwm.freq(1000)
```

```
while True:
```

```
    pot_value = potentiometer.read_u16()
```

```
    pwm.duty_u16(pot_value)
```

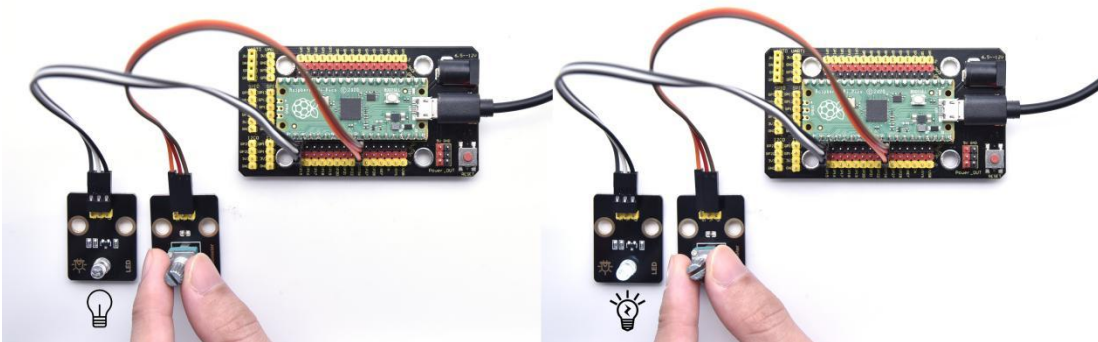
```
    utime.sleep(0.1)
```

## Code Explanation

It is easier to control the brightness of the LED with a potentiometer. In MicroPython, the ADC values range from 0 to 65535, just assign values directly, which is simple and convenient.

## Test Result

Run the code, turn the potentiometer to adjust the brightness of the LED.





## Project 32: Sound Activated Light



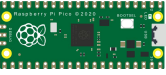
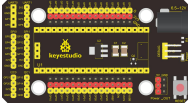

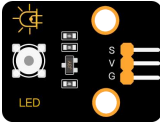


### Introduction

In this lesson, we will make a smart sound activated light using a sound sensor and an LED module. When we make a sound, the light will automatically turn on; when there is no sound, the lights will automatically turn off. How it works? Because the sound-controlled light is equipped with a sound sensor, and this sensor converts the intensity of external sound into a corresponding value. Then set a threshold, when the threshold is exceeded, the light will turn on, and when it is not exceeded, the light will

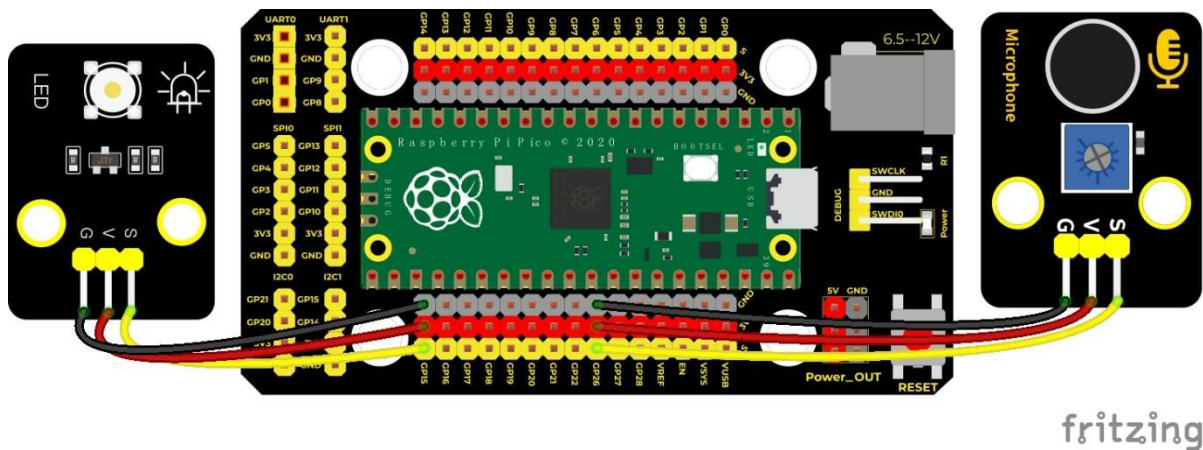


go out.

## Components

					
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio Sound Sensor*1	Keyestudio White LED Module*1	3P Dupont Wire*2	MicroUSB Cable*1

## Connection Diagram



## Test Code

!!!

\* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico



```
* lesson 32
* sound-controlled lights
* http://www.keyestudio.com
'''
```

```
import machine
import time

MicroPhone = machine.ADC(26)

led = machine.Pin(15,machine.Pin.OUT)

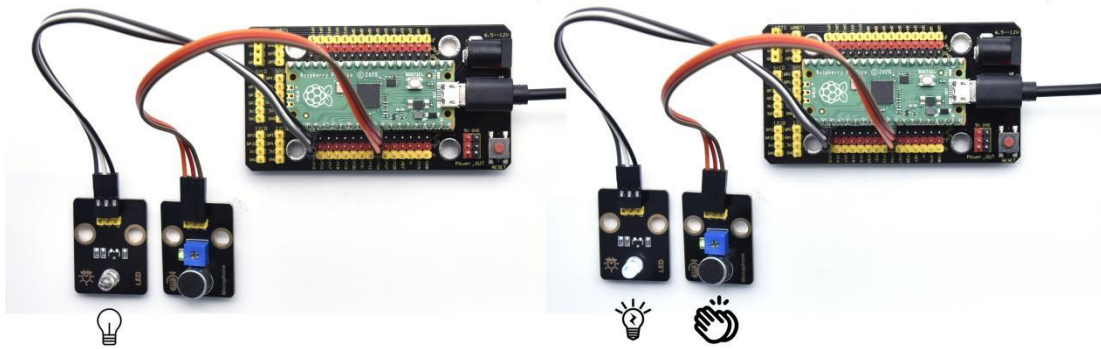
while True:
    value = MicroPhone.read_u16()
    print(value)
    if value > 5000:
        led.value(1)
        time.sleep(3)
    else:
        led.value(0)
        time.sleep(0.1)
```

## Code Explanation

In this experiment, we set the threshold to 5000. If it exceeds 5000, the LED will light up, otherwise it will be turned off.

## Test Result

Run the code, the Shell will display analog values. When we make a sound, the value will increase. If the value exceeds 5000, the LED will light up.



```
Shell %  
110  
720  
384  
0  
0  
1072  
448  
0  
0  
10242
```

## Project 33: RGB Module



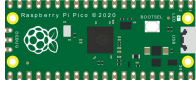
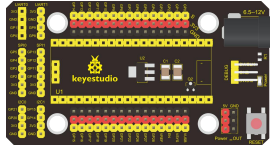
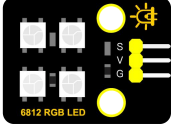


### Introduction

We learned how to use the 6812 RGB module, we knew that this module can light up each LED through a pin. In this experiment, we will control the RGB module to display different colors. (Note: do not look directly at the LEDs for a long time to avoid damage to our eyes.)

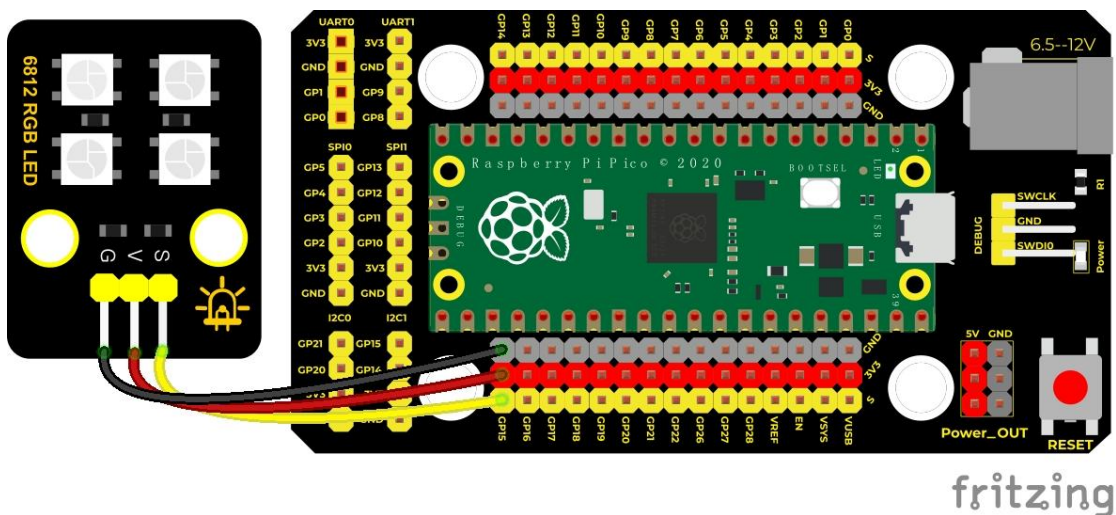




## Components

				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio 6812 RGB Module*1	3P Dupont Wire*1	MicroUSB Cable*1

## Connection Diagram



## Test Code

```

'''
* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico
* lesson 33
* RGB Module
* http://www.keyestudio.com
'''

# Example using PIO to drive a set of WS2812 LEDs.

import array, time
from machine import Pin
import rp2

```



```
# Configure the number of WS2812 LEDs.
```

```
NUM_LEDS = 4
```

```
PIN_NUM = 15
```

```
brightness = 0.2
```

```
@rp2.asm_pio(sideset_init=rp2.PIO.OUT_LOW, out_shiftdir=rp2.PIO.SHIFT_LEFT, autopull=True, pull_thresh=24)
```

```
def ws2812():
```

```
    T1 = 2
```

```
    T2 = 5
```

```
    T3 = 3
```

```
    wrap_target()
```

```
    label("bitloop")
```

```
    out(x, 1)                .side(0)    [T3 - 1]
```

```
    jmp(not_x, "do_zero")    .side(1)    [T1 - 1]
```

```
    jmp("bitloop")          .side(1)    [T2 - 1]
```

```
    label("do_zero")
```

```
    nop()                    .side(0)    [T2 - 1]
```

```
    wrap()
```

```
# Create the StateMachine with the ws2812 program, outputting on pin
```

```
sm = rp2.StateMachine(0, ws2812, freq=8_000_000, sideset_base=Pin(PIN_NUM))
```

```
# Start the StateMachine, it will wait for data on its FIFO.
```

```
sm.active(1)
```

```
# Display a pattern on the LEDs via an array of LED RGB values.
```

```
ar = array.array("I", [0 for _ in range(NUM_LEDS)])
```

```
#####
```

```
def pixels_show():
```

```
    dimmer_ar = array.array("I", [0 for _ in range(NUM_LEDS)])
```

```
    for i,c in enumerate(ar):
```

```
        r = int(((c >> 8) & 0xFF) * brightness)
```

```
        g = int(((c >> 16) & 0xFF) * brightness)
```

```
        b = int((c & 0xFF) * brightness)
```

```
        dimmer_ar[i] = (g<<16) + (r<<8) + b
```

```
    sm.put(dimmer_ar, 8)
```

```
    time.sleep_ms(10)
```

```
def pixels_set(i, color):
```

```
    ar[i] = (color[1]<<16) + (color[0]<<8) + color[2]
```





```
def color_chase(color, wait):
    for i in range(NUM_LEDS):
        pixels_set(i, color)
        time.sleep(wait)
        pixels_show()
    time.sleep(0.2)

def wheel(pos):
    # Input a value 0 to 255 to get a color value.
    # The colours are a transition r - g - b - back to r.
    if pos < 0 or pos > 255:
        return (0, 0, 0)
    if pos < 85:
        return (255 - pos * 3, pos * 3, 0)
    if pos < 170:
        pos -= 85
        return (0, 255 - pos * 3, pos * 3)
    pos -= 170
    return (pos * 3, 0, 255 - pos * 3)

def rainbow_cycle(wait):
    for j in range(255):
        for i in range(NUM_LEDS):
            rc_index = (i * 256 // NUM_LEDS) + j
            pixels_set(i, wheel(rc_index & 255))
        pixels_show()
        time.sleep(wait)

BLACK = (0, 0, 0)
RED = (255, 0, 0)
YELLOW = (255, 150, 0)
GREEN = (0, 255, 0)
CYAN = (0, 255, 255)
BLUE = (0, 0, 255)
PURPLE = (180, 0, 255)
WHITE = (255, 255, 255)
COLORS = (BLACK, RED, YELLOW, GREEN, CYAN, BLUE, PURPLE, WHITE)

print("chases")
for color in COLORS:
    color_chase(color, 0.05)

print("rainbow")
```



[rainbow\\_cycle\(0\)](#)

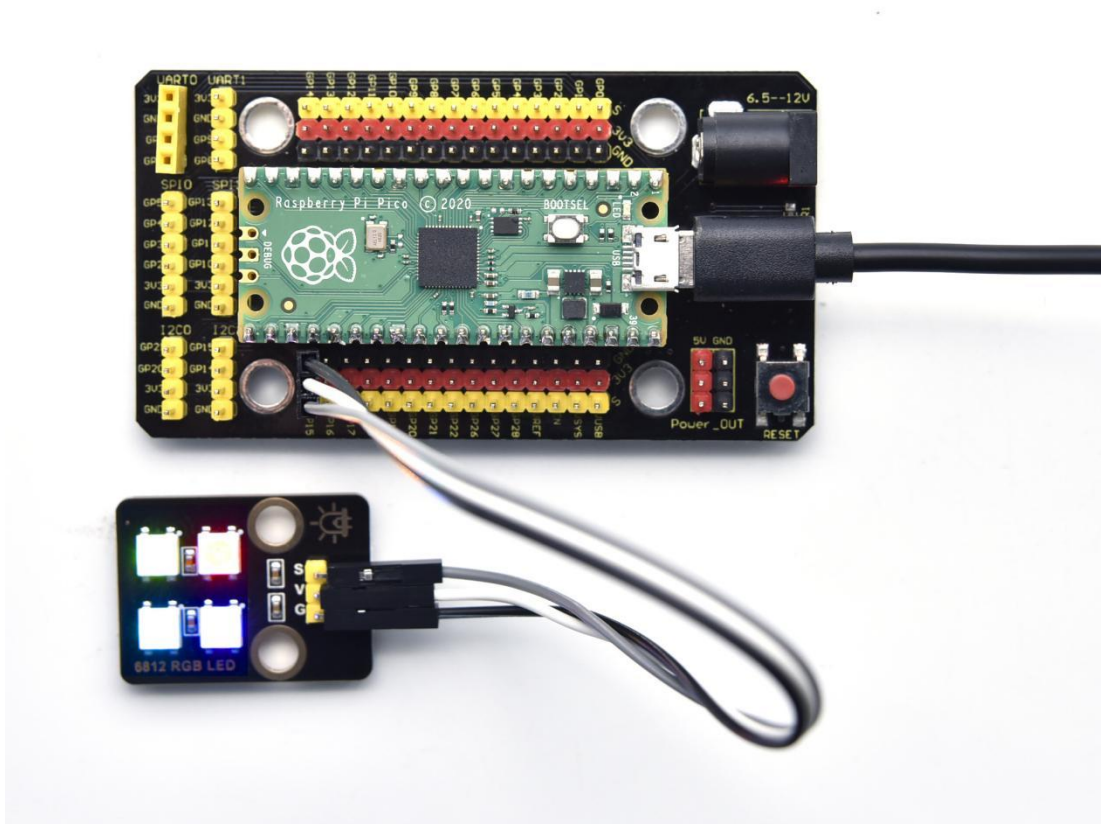
## Code Explanation

**color\_chase(color, wait):** "color" is used to control the LEDs to display corresponding color, "wait" is used to control the time that the LEDs change to another color.

**rainbow\_cycle(0):** to control the LEDs to show different colors repeatedly.

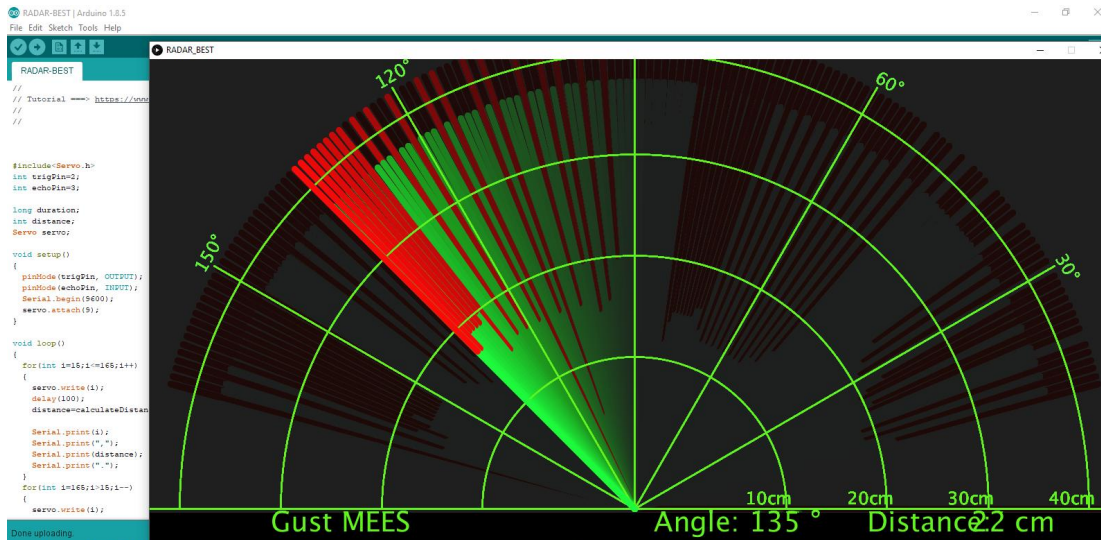
## Test Result

Wire up the components and run the code. We will see the LEDs shows different colors.

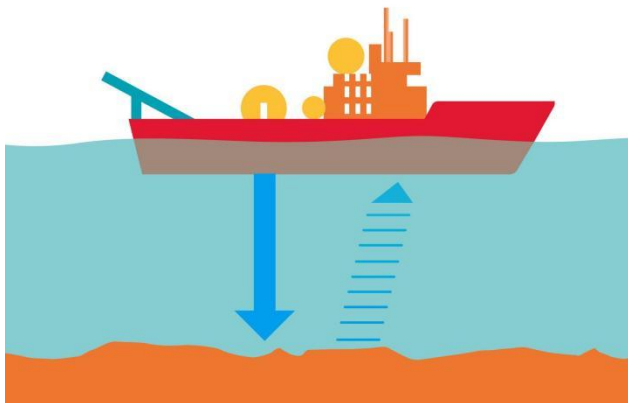




## Project 34: Ultrasonic Sensor



### Introduction



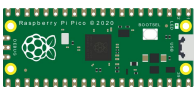
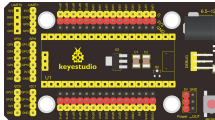

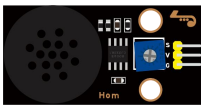
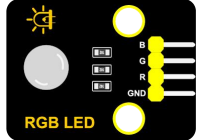




We know that bats use echoes to determine the direction and the location of their preys. In real life, sonar is used to detect sounds in the water. Since the attenuation rate of electromagnetic waves in water is very high, it cannot be used to detect signals, however, the attenuation rate of sound waves in the water is much smaller, so sound waves are most commonly used underwater for observation and measurement. In this experiment, we will use a speaker module, an RGB module and a 4-digit tube display to



make a device for detection through ultrasonic.

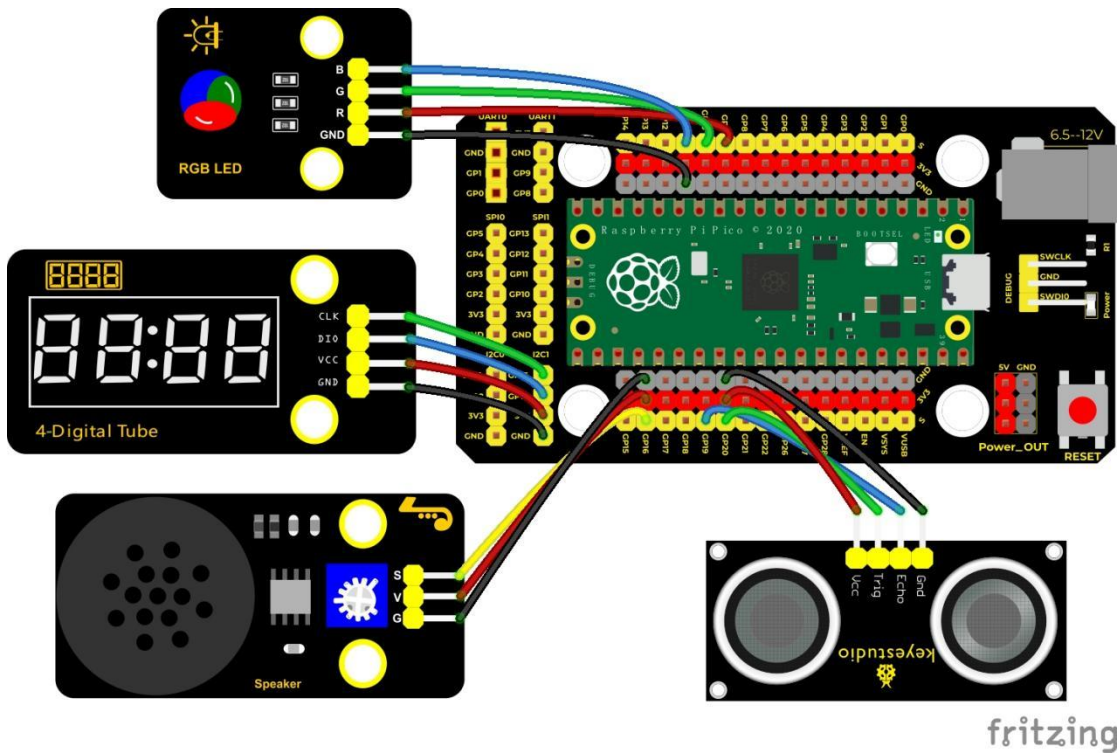


## Components

				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	keyes brick HC-SR04 Ultrasonic Sensor*1	Keyestudio Speaker Module*1	Keyestudio Common Cathode RGB Module*1
				
Keyestudio TM1650 4-Digit Tube Display*1	4P Dupont Wire*3	3P Dupont Wire*1	MicroUSB Cable*1	



## Connection Diagram



## Test Code

```
'''
* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico
* lesson 34
* Ultrasonic Sensor
* http://www.keyestudio.com
'''
```

```
from machine import Pin, PWM
import utime
```

```
# definitions for TM1650
ADDR_DIS = 0x48 #mode command
ADDR_KEY = 0x49 #read key value command
```

```
# definitions for brightness
BRIGHT_DARKEST = 0
BRIGHT_TYPICAL = 2
BRIGHTTEST      = 7
```

```
on  = 1
off = 0
```



```
# number:0~9
NUM = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f]
# DIG = [0x68,0x6a,0x6c,0x6e]
DIG = [0x6e,0x6c,0x6a,0x68]
DOT = [0,0,0,0]
```

```
clkPin = 15
dioPin = 14
clk = machine.Pin(clkPin, machine.Pin.OUT)
dio = machine.Pin(dioPin, machine.Pin.OUT)
```

```
DisplayCommand = 0
```

```
def writeByte(wr_data):
    global clk,dio
    for i in range(8):
        if(wr_data & 0x80 == 0x80):
            dio.value(1)
        else:
            dio.value(0)
        clk.value(0)
        utime.sleep(0.0001)
        clk.value(1)
        utime.sleep(0.0001)
        clk.value(0)
        wr_data <<= 1
    return
```

```
def start():
    global clk,dio
    dio.value(1)
    clk.value(1)
    utime.sleep(0.0001)
    dio.value(0)
    return
```

```
def ack():
    global clk,dio
    dy = 0
    clk.value(0)
    utime.sleep(0.0001)
    dio = Pin(dioPin, machine.Pin.IN)
    while(dio.value() == 1):
```



```
        utime.sleep(0.0001)
        dy += 1
        if(dy>5000):
            break
    clk.value(1)
    utime.sleep(0.0001)
    clk.value(0)
    dio = Pin(dioPin, machine.Pin.OUT)
    return
```

```
def stop():
    global clk,dio
    dio.value(0)
    clk.value(1)
    utime.sleep(0.0001)
    dio.value(1)
    return
```

```
def displayBit(bit, num):
    global ADDR_DIS
    if(num > 9 and bit > 4):
        return
    start()
    writeByte(ADDR_DIS)
    ack()
    writeByte(DisplayCommand)
    ack()
    stop()
    start()
    writeByte(DIG[bit-1])
    ack()
    if(DOT[bit-1] == 1):
        writeByte(NUM[num] | 0x80)
    else:
        writeByte(NUM[num])
    ack()
    stop()
    return
```

```
def clearBit(bit):
    if(bit > 4):
        return
    start()
    writeByte(ADDR_DIS)
```



```
ack()
writeByte(DisplayCommand)
ack()
stop()
start()
writeByte(DIG[bit-1])
ack()
writeByte(0x00)
ack()
stop()
return
```

```
def setBrightness(b = BRIGHT_TYPICAL):
    global DisplayCommand,brightness
    DisplayCommand = (DisplayCommand & 0x0f)+(b<<4)
    return
```

```
def setMode(segment = 0):
    global DisplayCommand
    DisplayCommand = (DisplayCommand & 0xf7)+(segment<<3)
    return
```

```
def displayOnOFF(OnOff = 1):
    global DisplayCommand
    DisplayCommand = (DisplayCommand & 0xfe)+OnOff
    return
```

```
def displayDot(bit, OnOff):
    if(bit > 4):
        return
    if(OnOff == 1):
        DOT[bit-1] = 1;
    else:
        DOT[bit-1] = 0;
    return
```

```
def InitDigitalTube():
    setBrightness(2)
    setMode(0)
    displayOnOFF(1)
    for _ in range(4):
        clearBit(_)
    return
```





```
def ShowNum(num): #0~9999
    displayBit(1,num%10)
    if(num < 10):
        clearBit(2)
        clearBit(3)
        clearBit(4)
    if(num > 9 and num < 100):
        displayBit(2,num//10%10)
        clearBit(3)
        clearBit(4)
    if(num > 99 and num < 1000):
        displayBit(2,num//10%10)
        displayBit(3,num//100%10)
        clearBit(4)
    if(num > 999 and num < 10000):
        displayBit(2,num//10%10)
        displayBit(3,num//100%10)
        displayBit(4,num//1000)

pwm_r = PWM(Pin(9))
pwm_g = PWM(Pin(10))
pwm_b = PWM(Pin(11))

pwm_r.freq(1000)
pwm_g.freq(1000)
pwm_b.freq(1000)

def light(red, green, blue):
    pwm_r.duty_u16(red)
    pwm_g.duty_u16(green)
    pwm_b.duty_u16(blue)

# Ultrasonic sensor detects distances, the unit is cm
def getDistance(trigger, echo):
    # Produces 10us square wave
    trigger.low()    #Pull the trigger pin low
    utime.sleep_us(2)
    trigger.high()
    utime.sleep_us(10)#Pull the trigger pin high for 10us before pulling the trigger pin low
    trigger.low()

    while echo.value() == 0: #Create a while loop to check whether the value of echo pin is 0 or not, and record
the time
```



```
        start = utime.ticks_us()
        while echo.value() == 1: #Create a while loop to check whether the value of echo pin is 1 or not, and record
the time
            end = utime.ticks_us()
            d = (end - start) * 0.0343 / 2 #Multiply the journey time (end-start) by the speed of sound (343.2 m/s, which
is 0.0343 cm per microsecond), the product of that equation is divided by two
            return d

# Configure the pins
trigger = Pin(20, Pin.OUT)
echo = Pin(19, Pin.IN)

buzzer = PWM(Pin(16))

def playtone(frequency):
    buzzer.duty_u16(1000)
    buzzer.freq(frequency)

def bequiet():
    buzzer.duty_u16(0)

# Main program
InitDigitalTube()
while True:
    distance = int(getDistance(trigger, echo))
    ShowNum(distance)
    if distance <= 10:
        playtone(880)
        utime.sleep(0.1)
        bequiet()
        light(65535, 0, 0)
    elif distance <= 20:
        playtone(532)
        utime.sleep(0.2)
        bequiet()
        light(0, 0, 65535)
    else:
        light(0, 65535, 0)
```

## Code Explanation

1. Set the frequencies of sound and the color of the LED by changing

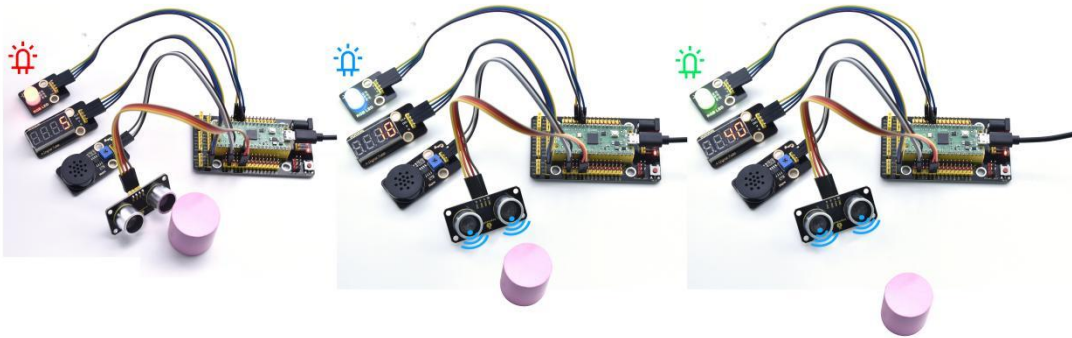


distances.

2. To facilitate the control of the distances of the obstacle, we can adjust the range of distance in the above code according to the actual situation.

## Test Result

Connect the components according to the connection diagram and run the code. When the ultrasonic sensor detects an obstacle at different distances, the buzzer on the speaker module will produce different frequencies of sound, the RGB will show different colors, and the measured distances are displayed on the 4-digit tube display.





## Project 35: IR Remote Control



### Introduction

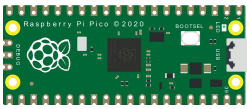
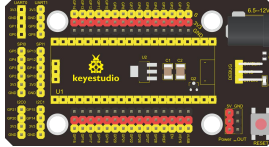
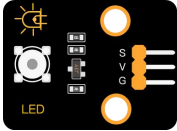
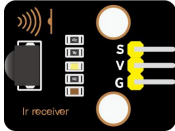

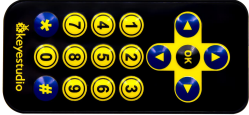

In the previous experiments, we learned to turn on or turn off the LED, adjust the brightness of a light through PWM, and how to use the infrared receiver module. So in this experiment, we use an infrared remote control to control an LED module.

When we receive a value, we set the PWM value by the corresponding button value, thus you can adjust the brightness. Control the LED to turn

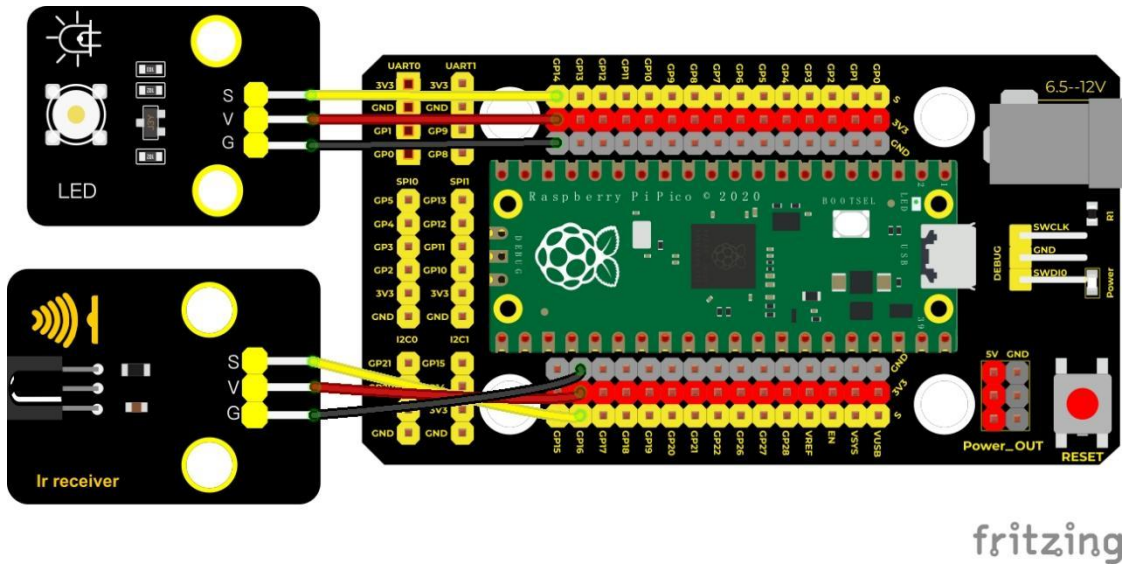


on or turn off is in the same way. If we want to use the same button to control the LED to turn on or turn off, we can achieve it through the code.

## Components

			
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio White LED Module*1	Keyestudio IR Receiver*1
			
MicroUSB Cable*1	Remote Control*1	3P Dupont Wire*2	

## Connection Diagram



## Test Code

```
'''
* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico
* lesson 35
* IR remote control
* http://www.keyestudio.com
'''

import time
from machine import Pin

led = Pin(14, Pin.OUT)
ird = Pin(16, Pin.IN)

act = {"1": "LLLLLLLLHHHHHHHHHLLHLLHLLHLLHLLH", "2": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLH", "3": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLH",
      "4": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLH", "5": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLH", "6": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLH",
      "7": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLH", "8": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLH", "9": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLH",
      "0": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLH", "Up": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLH", "Down": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLH",
      "Left": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLH", "Right": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLH",
      "Ok": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLH",
      "*": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLH", "#": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLH"}

def read_ircode(ird):
    wait = 1
    complete = 0
    seq0 = []
```



```
seq1 = []

while wait == 1:
    if ird.value() == 0:
        wait = 0
while wait == 0 and complete == 0:
    start = time.ticks_us()
    while ird.value() == 0:
        ms1 = time.ticks_us()
        diff = time.ticks_diff(ms1, start)
        seq0.append(diff)
    while ird.value() == 1 and complete == 0:
        ms2 = time.ticks_us()
        diff = time.ticks_diff(ms2, ms1)
        if diff > 10000:
            complete = 1
    seq1.append(diff)

code = ""
for val in seq1:
    if val < 2000:
        if val < 700:
            code += "L"
        else:
            code += "H"
# print(code)
command = ""
for k,v in act.items():
    if code == v:
        command = k
if command == "":
    command = code
return command
```

```
flag = False
while True:
    # global flag
    command = read_ircode(ird)
    print(command, end = " ")
    print(flag, end = " ")
    if command == "Ok":
        if flag == True:
            led.value(1)
            flag = False
```



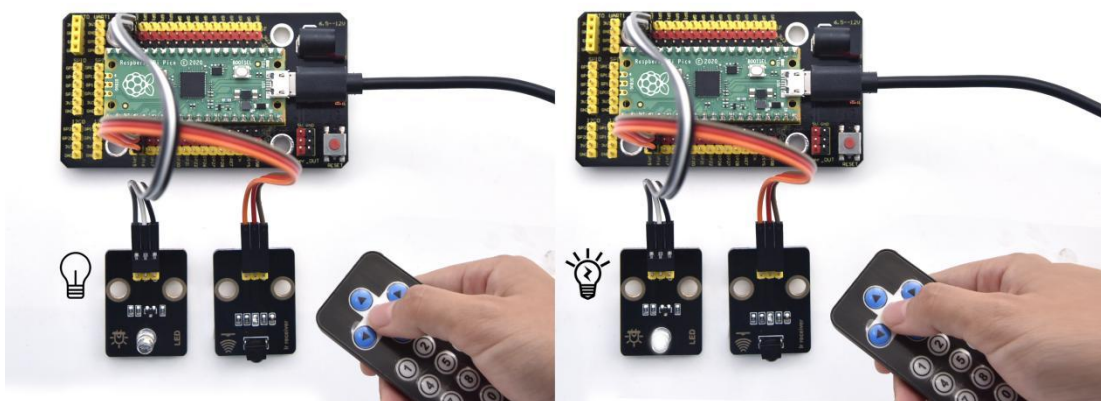
```
        print("led on")
    else:
        led.value(0)
        flag = True
        print("led off")
time.sleep(0.1)
```

## Code Explanation

1. We set a Boolean variable here, it has only two possible values: True or False.
2. When we press the "OK" button, the Shell will show "OK" . Then we set a Boolean variable (flag) to True, the LED will light up. If it is False, the LED will go off. If we set it to False after lighting up the LED, press "OK" again to turn off the LED.

## Test Result

Wire up, run the code and look at the Shell. Press keys on the IR remote control, the Shell will show values. Press "OK" to turn on the LED, and press it again to turn off the LED.

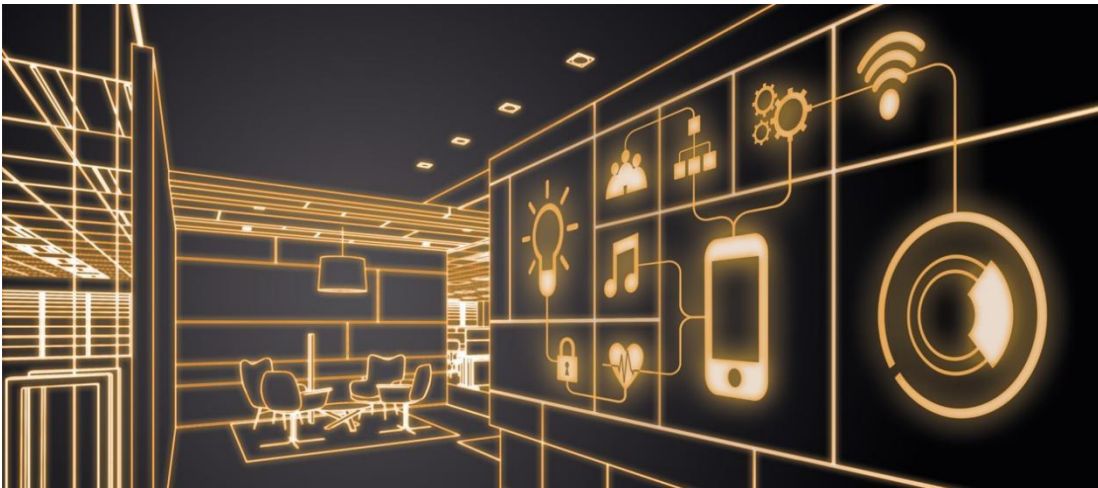






```
Shell %  
type help() for more information.  
>>> %Run -c $EDITOR_CONTENT  
  
Ok False led off  
Ok True led on  
Ok False led off  
Ok True led on  
Ok False led off  
Ok True led on  
Ok False led off
```

## Project 36: Comprehensive Experiment


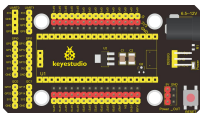
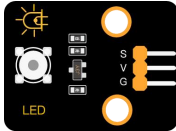
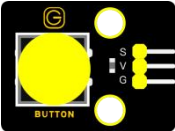
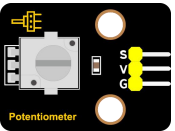
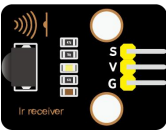


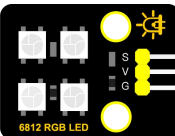







### Introduction

We did a lot of experiments, and for each one we needed to re-upload the code, so can we achieve different functions through an experiment? In this experiment, we will use an external button module to achieve different functions.

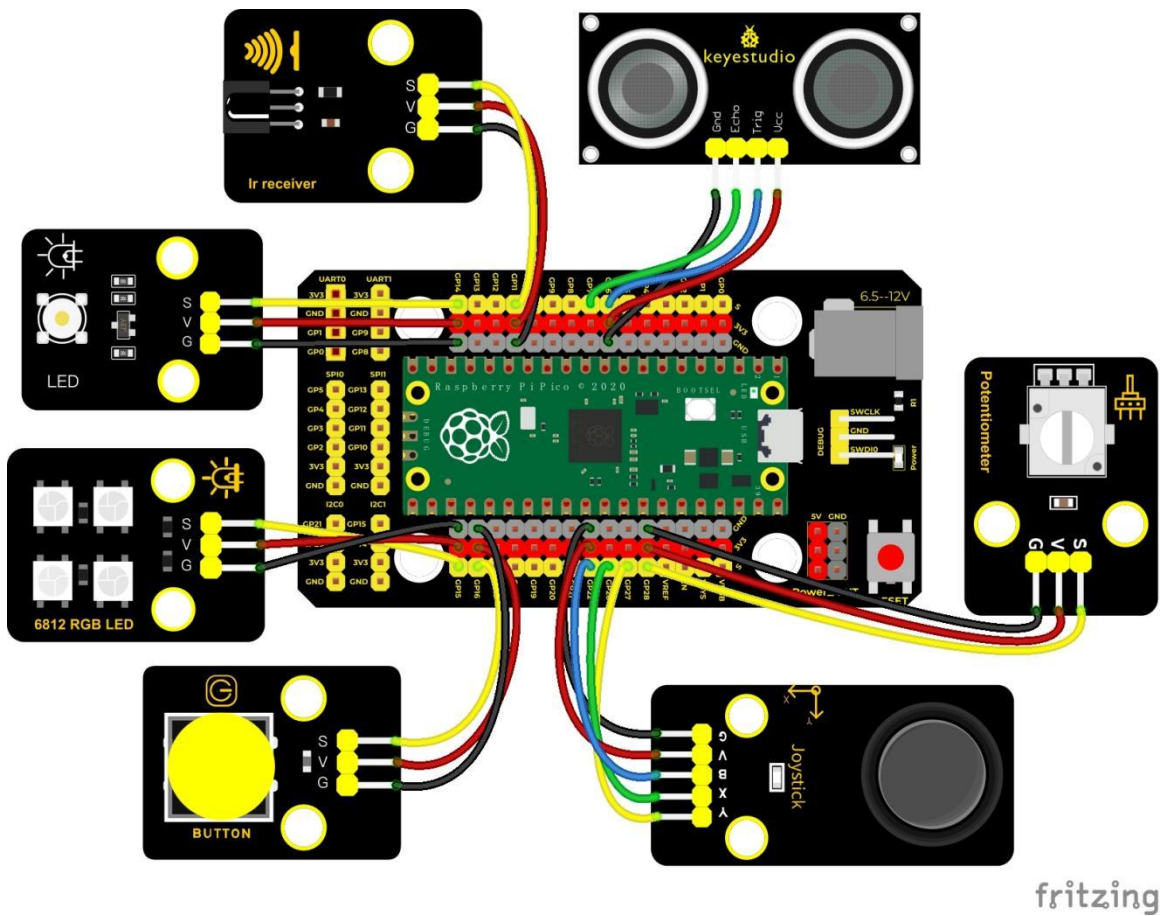


## Components

				
Raspberry Pi Pico Board*1	Raspberry Pi Pico Shield*1	Keyestudio White LED Module*1	Keyestudio Button Module*1	Keyestudio Rotary Potentiometer*1
				
Keyestudio IR Receiver*1	Keyestudio Joystick Module*1	HC-SR04 Ultrasonic Sensor*1	Keyestudio 6812 RGB Module*1	MicroUSB Cable*1
				
3P Dupont Wire*5	4P Dupont Wire*1	5P Dupont Wire*1	Remote Control*1	



## Connection Diagram



## Test Code

```
'''  
* Keyestudio 24 in 1 Starter Kit for Raspberry Pi Pico  
* lesson 36  
* Comprehensive experiment  
* http://www.keyestudio.com  
'''
```

```
from machine import Pin, PWM  
import array, time  
import random  
import rp2
```

```
potentiometer = machine.ADC(28)  
button = Pin(16, Pin.IN)  
led = PWM(Pin(14))  
led.freq(1000)
```



```
ird = Pin(11,Pin.IN)
B = machine.Pin(22, machine.Pin.IN)
X = machine.ADC(26)
Y = machine.ADC(27)
# configure the pins used with the ultrasonic sensor
trigger = Pin(6, Pin.OUT)
echo = Pin(7, Pin.IN)
# Configure the number of sk6812 LEDs, pins and brightness.
NUM_LEDS = 4
PIN_NUM = 15
brightness = 0.2

act = {"1": "LLLLLLLLHHHHHHHHHLLHHLHLLHLLHLLHHH","2": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHHH","3":
"LLLLLLLLHHHHHHHHHHHLLHHLHLLHLLHLLHHH",
      "4": "LLLLLLLLHHHHHHHHHHHLLHHLHLLHLLHLLHHH","5": "LLLLLLLLHHHHHHHHHHHLLHHLHLLHLLHLLHHH","6":
"LLLLLLLLHHHHHHHHHHHLLHHLHLLHLLHLLHHH",
      "7": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLHHH","8": "LLLLLLLLHHHHHHHHHHHLLHHLHLLHLLHLLHHH","9":
"LLLLLLLLHHHHHHHHHHHLLHHLHLLHLLHLLHHH",
      "0": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLHHH","Up": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLHHH","Down":
"LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLHHH",
      "Left": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLHHH","Right":
"LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLHHH","Ok": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLHHH",
      "*": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLHHH","#": "LLLLLLLLHHHHHHHHHHHLLHLLHLLHLLHLLHHH"}

def read_irdcode(ird):
    wait = 1
    complete = 0
    seq0 = []
    seq1 = []

    while wait == 1:
        if ird.value() == 0:
            wait = 0
    while wait == 0 and complete == 0:
        start = time.ticks_us()
        while ird.value() == 0:
            ms1 = time.ticks_us()
        diff = time.ticks_diff(ms1,start)
        seq0.append(diff)
        while ird.value() == 1 and complete == 0:
            ms2 = time.ticks_us()
            diff = time.ticks_diff(ms2,ms1)
            if diff > 10000:
                complete = 1
```



```
seq1.append(diff)
```

```
code = ""
for val in seq1:
    if val < 2000:
        if val < 700:
            code += "L"
        else:
            code += "H"
# print(code)
command = ""
for k,v in act.items():
    if code == v:
        command = k
if command == "":
    command = code
return command
```

```
@rp2.asm_pio(sideset_init=rp2.PIO.OUT_LOW, out_shiftdir=rp2.PIO.SHIFT_LEFT, autopull=True, pull_thresh=24)
def sk6812():
```

```
    T1 = 2
    T2 = 5
    T3 = 3
    wrap_target()
    label("bitloop")
    out(x, 1)                .side(0)    [T3 - 1]
    jmp(not_x, "do_zero")    .side(1)    [T1 - 1]
    jmp("bitloop")          .side(1)    [T2 - 1]
    label("do_zero")
    nop()                    .side(0)    [T2 - 1]
    wrap()
```

```
# Create the StateMachine with the sk6812 program, outputting on Pin(16).
```

```
sm = rp2.StateMachine(0, sk6812, freq=8_000_000, sideset_base=Pin(PIN_NUM))
```

```
# Start the StateMachine, it will wait for data on its FIFO.
```

```
sm.active(1)
```

```
# Display a pattern on the LEDs via an array of LED RGB values.
```

```
ar = array.array("I", [0 for _ in range(NUM_LEDS)])
```

```
def pixels_show():
```

```
    dimmer_ar = array.array("I", [0 for _ in range(NUM_LEDS)])
```



```
for i,c in enumerate(ar):
    r = int(((c >> 8) & 0xFF) * brightness)
    g = int(((c >> 16) & 0xFF) * brightness)
    b = int((c & 0xFF) * brightness)
    dimmer_ar[i] = (g<<16) + (r<<8) + b
sm.put(dimmer_ar, 8)
time.sleep_ms(10)

def pixels_set(i, color):
    ar[i] = (color[1]<<16) + (color[0]<<8) + color[2]

# Ultrasonic sensor detects distances, the unit is cm
def getDistance(trigger, echo):
    # Produces 10us square wave
    trigger.low()    # Pull the trigger pin low
    time.sleep_us(2)
    trigger.high()
    time.sleep_us(10)# Pull the trigger pin high for 10us before pulling the trigger pin low
    trigger.low()

    while echo.value() == 0: # Create a while loop to check whether the value of echo pin is 0 or not, and record
the time
        start = time.ticks_us()
        while echo.value() == 1: # Create a while loop to check whether the value of echo pin is 1 or not, and record
the time
            end = time.ticks_us()
            d = (end - start) * 0.0343 / 2 # Multiply the journey time (end-start) by the speed of sound (343.2 m/s, which
is 0.0343 cm per microsecond), the product of that equation is divided by two
            return d

keys = 0
nums = 0

def toggle_handle(pin):
    global keys
    keys += 1

button.irq(trigger = Pin.IRQ_FALLING, handler = toggle_handle)

def show6812():
    R = random.randint(0,255)
    G = random.randint(0,255)
    B = random.randint(0,255)
```



```
for i in range(NUM_LEDS):  
    pixels_set(i, (R, G, B))  
    pixels_show()  
    time.sleep(0.3)
```

```
def IRreceive():  
    command = read_ircode(ird)  
    print(command)
```

```
def showJoystick():  
    B_value = B.value()  
    X_value = X.read_u16()  
    Y_value = Y.read_u16()  
    print("button:", end = " ")  
    print(B_value, end = " ")  
    print("X:", end = " ")  
    print(X_value, end = " ")  
    print("Y:", end = " ")  
    print(Y_value)  
    time.sleep(0.1)
```

```
def adjustLight():  
    pot_value = potentiometer.read_u16()  
    print(pot_value)  
    led.duty_u16(pot_value)  
    time.sleep(0.1)
```

```
def showDistance():  
    distance = getDistance(trigger, echo)  
    print("The distance is : {:.2f} cm".format(distance))  
    time.sleep(0.1)
```

```
while True:  
    nums = keys % 5  
    print(nums)  
    if nums == 0:  
        show6812()  
    elif nums == 1:  
        IRreceive()  
    elif nums == 2:  
        showJoystick()  
    elif nums == 3:  
        adjustLight()  
    elif nums == 4:
```

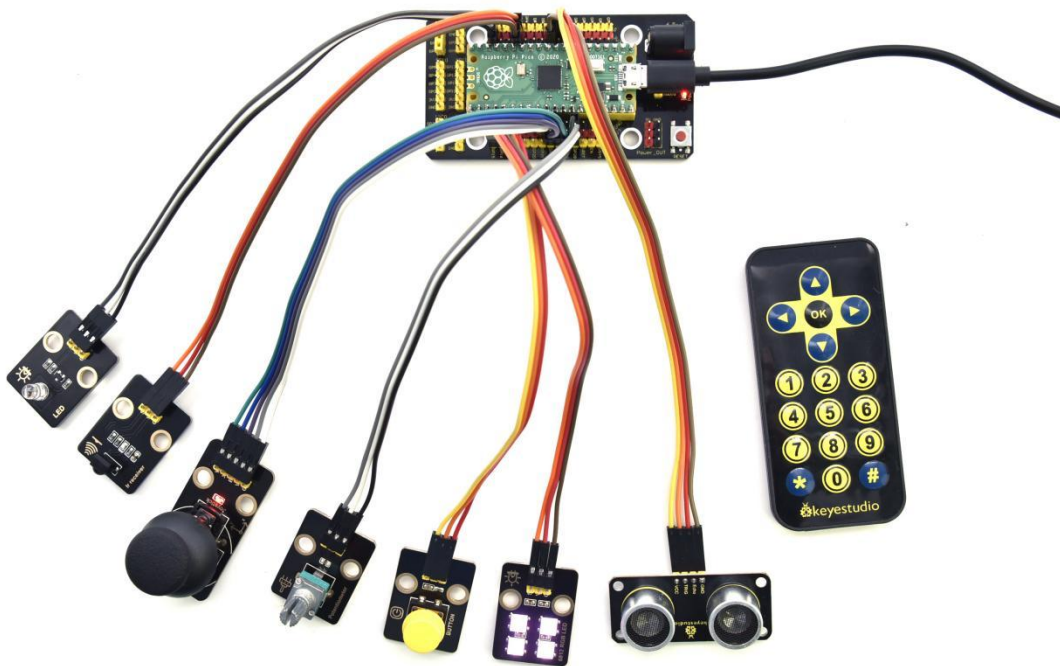


`showDistance()`

## Code Explanation

1. Each time the button on the button module is pressed, the original value of the "keys +" will plus 1. The value divided by 5, it will get a remainder (0, 1, 2, 3, 4). Different remainders correspond to different functions. We will create 5 functions to achieve them.
2. Add or reduce sensors or modules when wiring, then change the code to achieve the function that we want.

## Test Result



Wire up the components, supply power via a USB cable and run the code. In the beginning, the values of "keys +" and the remainder are 0, the LEDs





on the RGB module randomly displays colors.

```
Shell X
0
0
0
0
0
0
0
0
0
0
0
0
```



Press the button, the LEDs will go off, the values of "keys +" and the remainder are 1. This implements the function of sending information from the infrared receiver module. If we put the infrared remote control towards the receiver module, press a button, the receiver module receives information, as illustrated below.

```
Shell X
1
Left
1

1
Down
1
Ok
1
```

Press the button again, the values of "keys +" and the remainder are 2, which can read the analog values of X axis and Y axis of the joystick module. (Note: because there is no signal to return to the "IRreceive()" function, we need to press any button on the remote control again at this point). The



values of the KEY interface (Z axis) are shown below.

```
Shell X
4
button: 0 X: 32920 Y: 32856
2
button: 1 X: 32840 Y: 32968
2
button: 1 X: 32904 Y: 32792
2
button: 1 X: 32824 Y: 32808
2
button: 1 X: 32840 Y: 33000
```

As shown in the figure, press the button again, the values of "keys +" and the remainder are 3. We can use the external adjustable potentiometer module to adjust the PWM values of the LED (GP14) interface, then the brightness of the LED on the white LED module will change.

```
Shell X
3
65535
3
56973
3
46139
3
35288
3
21477
```

Press the button again, the values of "keys +" and the remainder are 4. We can use the ultrasonic sensor to detect distances, they are displayed on the Shell.

```
Shell X
4
The distance is : 6.50 cm
4
The distance is : 6.35 cm
4
The distance is : 6.29 cm
4
The distance is : 6.26 cm
4
The distance is : 6.29 cm
```

Press the button again, the value of "keys +" is 5 and the remainder is 0, the LEDs on the RGB module will blink again.



## 6. Resources:

**Download test code:**

<https://fs.keyestudio.com/KS3021>