

Welcome

Thank you for choosing Freenove products!

Getting Started

When reading this, you should have downloaded the ZIP file for this product.

Unzip it and you will get a folder containing tutorials and related files. Please start with this PDF tutorial.

- ! Unzip the ZIP file instead of opening the file in the ZIP file directly.
- ! Do not move, delete or rename files in the folder just unzipped.

Get Support

Encounter problems? Don't worry! Refer to "TroubleShooting.pdf" or contact us.

When there are packaging damage, quality problems, questions encountering in use, etc., just send us an email. We will reply to you within one working day and provide a solution.

support@freenove.com

Safety and Precautions

Please follow the following safety precautions when using or storing this product:

- Keep this product out of the reach of children under 6 years old.
- This product should be used only when there is adult supervision present as young children lack necessary judgment regarding safety and the consequences of product misuse.
- This product contains small parts and parts, which are sharp. This product contains electrically conductive parts. Use caution with electrically conductive parts near or around power supplies, batteries and powered (live) circuits.
- When the product is turned ON, activated or tested, some parts will move or rotate. To avoid injuries to hands and fingers, keep them away from any moving parts!
- It is possible that an improperly connected or shorted circuit may cause overheating. Should this happen, immediately disconnect the power supply or remove the batteries and do not touch anything until it cools down! When everything is safe and cool, review the product tutorial to identify the cause.
- Only operate the product in accordance with the instructions and guidelines of this tutorial, otherwise parts may be damaged or you could be injured.
- Store the product in a cool dry place and avoid exposing the product to direct sunlight.
- After use, always turn the power OFF and remove or unplug the batteries before storing.

Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

About Freenove

Freenove provides open source electronic products and services worldwide.

Freenove is committed to assist customers in their education of robotics, programming and electronic circuits so that they may transform their creative ideas into prototypes and new and innovative products. To this end, our services include but are not limited to:

- Educational and Entertaining Project Kits for Robots, Smart Cars and Drones
- Educational Kits to Learn Robotic Software Systems for Arduino, Raspberry Pi and micro:bit
- Electronic Component Assortments, Electronic Modules and Specialized Tools
- **Product Development and Customization Services**

You can find more about Freenove and get our latest news and updates through our website:

<http://www.freenove.com>

Copyright

All the files, materials and instructional guides provided are released under [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License](#). A copy of this license can be found in the folder containing the Tutorial and software files associated with this product.



This means you can use these resources in your own derived works, in part or completely, but **NOT for the intent or purpose of commercial use.**

Freenove brand and logo are copyright of Freenove Creative Technology Co., Ltd. and cannot be used without written permission.



Other registered trademarks and their owners appearing in this document:

Arduino® is a trademark of Arduino LLC (<https://www.arduino.cc/>).

Raspberry Pi® is a trademark of Raspberry Pi Foundation (<https://www.raspberrypi.org/>).

Raspberry Pi Pico® is a trademark of Raspberry Pi Foundation (<https://www.raspberrypi.org/>).

micro:bit® is a trademark of Micro:bit Educational Foundation (<https://www.microbit.org/>).

ESPRESSIF® and ESP32® are trademarks of ESPRESSIF Systems (Shanghai) Co, Ltd (<https://www.espressif.com/>).

Any concerns? ✉ support@freenove.com

Contents

Welcome.....	1
Contents	1
Preface.....	4
Raspberry Pi Pico.....	5
Chapter 0 Getting Ready (Important).....	9
0.1 Installing Thonny (Important)	9
0.2 Basic Configuration of Thonny	14
0.3 Burning Micropython Firmware (Important).....	16
0.4 Thonny Connected to Raspberry Pi Pico	19
0.5 Testing codes (Important).....	23
0.6 Thonny Common Operation.....	34
07. Paste the Sticker on the Breadboard	39
Chapter 1 LED (Important).....	40
Project 1.1 Blink	40
Project 1.2 Blink	49
Chapter 2 Button & LED	57
Project 2.1 Button & LED.....	58
Project 2.2 MINI table lamp.....	62
Chapter 3 LED Bar	65
Project 3.1 Flowing Light	65
Chapter 4 Analog & PWM	71
Project 4.1 Breathing LED.....	71
Project 4.2 Meteor Flowing Light	77
How to import a custom python module.....	81
Chapter 5 RGBLED	83
Project 5.1 Random Color Light.....	83
Project 5.2 Gradient Color Light.....	89
Chapter 6 NeoPixel	92
Project 6.1 NeoPixel	92
Project 6.2 Rainbow Light.....	98

Chapter 7 Buzzer	101
Project 7.1 Doorbell.....	101
Project 7.2 Alertor	107
Chapter 8 Serial Communication.....	111
Project 8.1 Serial Print.....	111
Project 8.2 Serial Read and Write	116
Chapter 9 AD Converter.....	120
Project 9.1 Read the Voltage of Potentiometer.....	120
Chapter 10 Potentiometer & LED	126
Project 10.1 Soft Light	126
Project 10.2 Soft Colorful Light	130
Project 10.3 Soft Rainbow Light.....	134
Chapter 11 Photoresistor & LED	138
Project 11.1 Control LED through Photoresistor.....	138
Chapter 12 Thermistor	143
Project 12.1 Thermometer	143
Chapter 13 Joystick	148
Project 13.1 Joystick	148
Chapter 14 74HC595 & LED Bar Graph.....	153
Project 14.1 Flowing Water Light.....	153
Chapter 15 74HC595 & 7-Segment Display.....	159
Project 15.1 7-Segment Display.....	159
Project 15.2 4-Digit 7-Segment Display.....	165
Chapter 16 74HC595 & LED Matrix	171
Project 16.1 LED Matrix.....	171
Chapter 17 Relay & Motor.....	179
Project 17.1 Relay & Motor	179
Chapter 18 L293D & Motor.....	187
Project 18.1 Control Motor with Potentiometer.....	187
Chapter 19 Servo	196
Project 19.1 Servo Sweep.....	196

Project 19.2 Servo Knob	201
Chapter 20 Stepper Motor.....	205
Project 20.1 Stepper Motor	205
Chapter 21 LCD1602	214
Project 21.1 LCD1602	214
Chapter 22 Ultrasonic Ranging	221
Project 22.1 Ultrasonic Ranging	221
Project 22.2 Ultrasonic Ranging	227
Chapter 23 Matrix Keypad	229
Project 23.1 Matrix Keypad.....	229
Project 23.2 Keypad Door	234
Chapter 24 Infrared Remote	239
Project 24.1 Infrared Remote Control.....	239
Project 24.2 Control LED through Infrared Remote	245
Chapter 25 Hygrothermograph DHT11.....	250
Project 25.1 Hygrothermograph.....	250
Project 25.2 Hygrothermograph.....	256
Chapter 26 Infrared Motion Sensor	261
Project 26.1 Infrared Motion Detector with LED Indicator.....	261
Chapter 27 Attitude Sensor MPU6050.....	267
Project 27.1 Read a MPU6050 Sensor Module	267
Chapter 28 RFID	273
Project 28.1 RFID read UID	273
Project 28.2 RFID Read and Write	283
What's Next?	288

Preface

Raspberry Pi Pico is a tiny, fast, and versatile board built using RP2040, a brand new microcontroller chip designed by Raspberry Pi in the UK. Getting started is as easy as dragging and dropping a file, it is suitable for beginners and makers to develop, design and research.

Raspberry Pi Pico is programmable in C/C++ and MicroPython. In this tutorial, we use Micropython to develop, which is a very easy to learn language with lean and simple code, hence it is very suitable for beginners to learn and for secondary development.

If you haven't downloaded the related material for Raspberry Pi Pico tutorial, you can download it from this link:

https://github.com/Freenove/Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico

In this tutorial, we devide each project into 4 sections:

- 1, Component list: helps users to learn and find what components needed in each project.
- 2, Component knowledge: allows you to learn the features and usage of the components.
- 3, Circuit: assists to build circuit for each project.
- 4, Code and annotation: makes it easier for users to learn to use Raspberry Pi Pico and make secondary development.

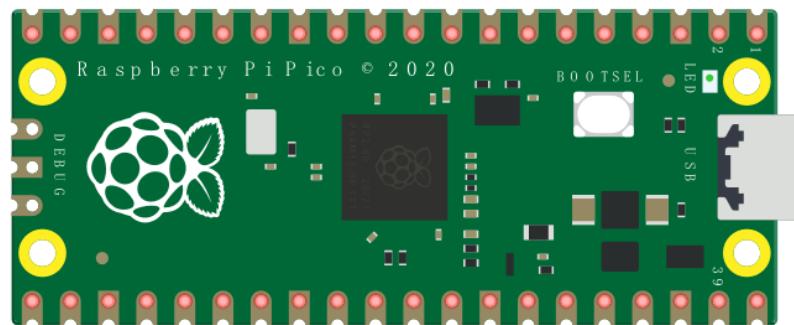
After completing the projects in this tutorial, you can also combine the components in different projects to make your own smart homes, smart car, robot, etc., bringing your imagination and creativity to life with Raspberry Pi Pico.

If you have any problems or difficulties using this product, please contact us for quick and free technical support: support@freenove.com

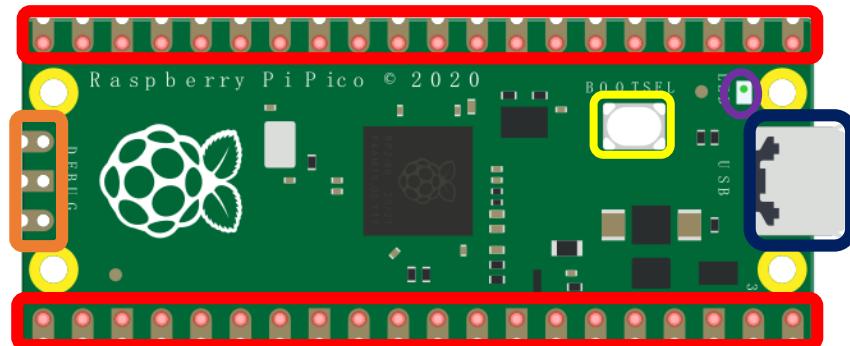
Any concerns? ✉ support@freenove.com

Raspberry Pi Pico

Raspberry Pi Pico is a light-weight electronic product with tiny size and low price. From the picture below we can see that its onboard resources have been connected to the edge interface, which is very suitable for electronic enthusiasts to use in DIY.

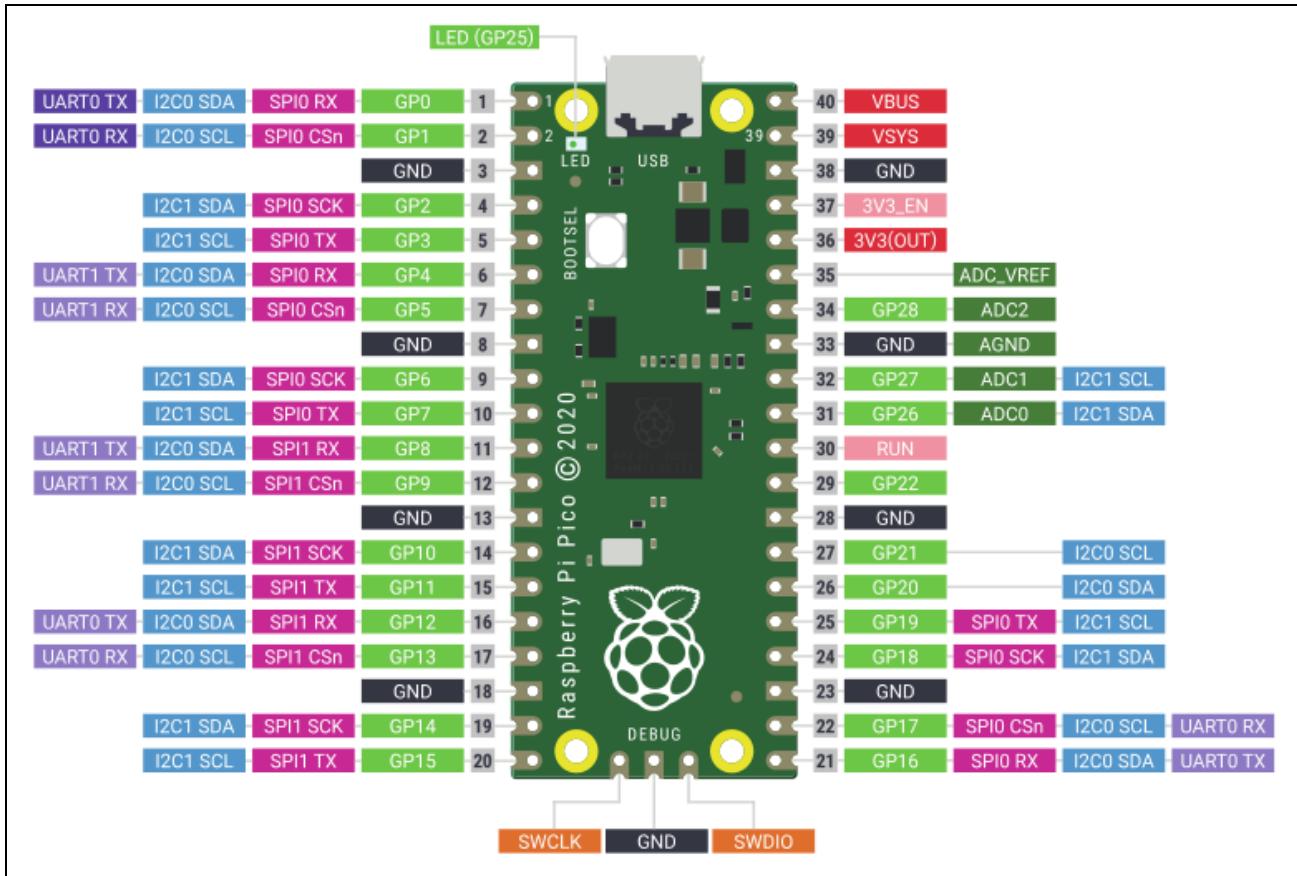


The hardware interfaces are distributed as follows:



Frame color	Description
	Pins
	BOOTSEL button
	USB port
	LED
	Debugging

Function definition of pins:



Color	Pins	Color	Pins
Black	GND	Red	Power
Green	GPIO	Dark Green	ADC
Pink	UART(defualt)	Lavender	UART
Magenta	SPI	Light Blue	I2C
Light Red	System Control	Orange	Debugging

For details: <https://datasheets.raspberrypi.org/pico/pico-datasheet.pdf>

GND	Ground Pin
Power	VBUS(microUSB Voltage)、VSYS(2-5VDC Input)、3V3(3.3V OUT)、3V3_EN(Enables Pico)
System Control	Run(Start or disable RP2040 microcontroller or reset)
ADC	Raspberry Pi Pico has a total of 5 ADC with a resolution of 12 bits, which are ADC0(GP26), ADC1(GP27), ADC2(GP28), ADC3(GP29), ADC4 respectively. Among them, ADC3(GP29) is used to measure the VSYS on Pico board; ADC4 is directly connected to the RP2040's built-in temperature sensor. ADC_VREF can connect to external accurate voltmeter as ADC reference. ADC_GND pin is used as the reference point for grounding.
PWM	There are 16 PWM channels on Raspberry Pi Pico, each of which can control frequency and duty cycle independently. The GPIO pins are switched to PWM function.
UART	There are 2 UART: UART0, UART1.
SPI	There are 2 SPI: SPI0, SPI1.
I2C	2 I2C: I2C0, I2C1.
Debugging	It is used when debugging code.

UART, I2C, SPI Default Pin

UART

Function	Default
UART_BAUDRATE	115200
UART_BITS	8
UART_STOP	1
UART0_TX	Pin 0
UART0_RX	Pin 1
UART1_TX	Pin 4
UART1_RX	Pin 5

I2C

Function	Default
I2C Frequency	400000
I2C0 SCL	Pin 9
I2C0 SDA	Pin 8
I2C1 SCL	Pin 7
I2C1 SDA	Pin 6

SPI

Function	Default
SPI_BAUDRATE	1000000
SPI_POLARITY	0
SPI_PHASE	0
SPI_BITS	8
SPI_FIRSTBIT	MSB
SPI0_SCK	Pin 6
SPI0_MOSI	Pin 7
SPI0_MISO	Pin 4
SPI1_SCK	Pin 10
SPI1_MOSI	Pin 11
SPI1_MISO	Pin 8

For more detailed information, please refer to:

<https://datasheets.raspberrypi.org/pico/raspberry-pi-pico-python-sdk.pdf>

Chapter 0 Getting Ready (Important)

Before starting building the projects, you need to make some preparation first, which is so crucial that you must not skip.

0.1 Installing Thonny (Important)

Thonny is a free, open-source software platform with compact size, simple interface, simple operation and rich functions, making it a Python IDE for beginners. In this tutorial, we use this IDE to develop Raspberry Pi Pico during the whole process.

Thonny supports various operating system, including Windows、Mac OS、Linux.

Downloading Thonny

Official website of Thonny: <https://thonny.org>

Open-source code repositories of Thonny: <https://github.com/thonny/thonny>

Follow the instruction of official website to install Thonny or click the links below to download and install.
(Select the appropriate one based on your operating system.)

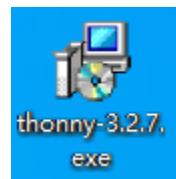
Operating System	Download links/methods
Windows	https://github.com/thonny/thonny/releases/download/v3.2.7/thonny-3.2.7.exe
Mac OS	https://github.com/thonny/thonny/releases/download/v3.2.7/thonny-3.2.7.pkg
Linux	The latest version: Binary bundle for PC (Thonny+Python): bash <(wget -O - https://thonny.org/installer-for-linux) With pip: pip3 install thonny Distro packages (may not be the latest version): Debian, Raspbian, Ubuntu, Mint and others: sudo apt install thonny Fedora: sudo dnf install thonny

You can also open “[Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico/Python_Software](#)”, we have prepared it in advance.

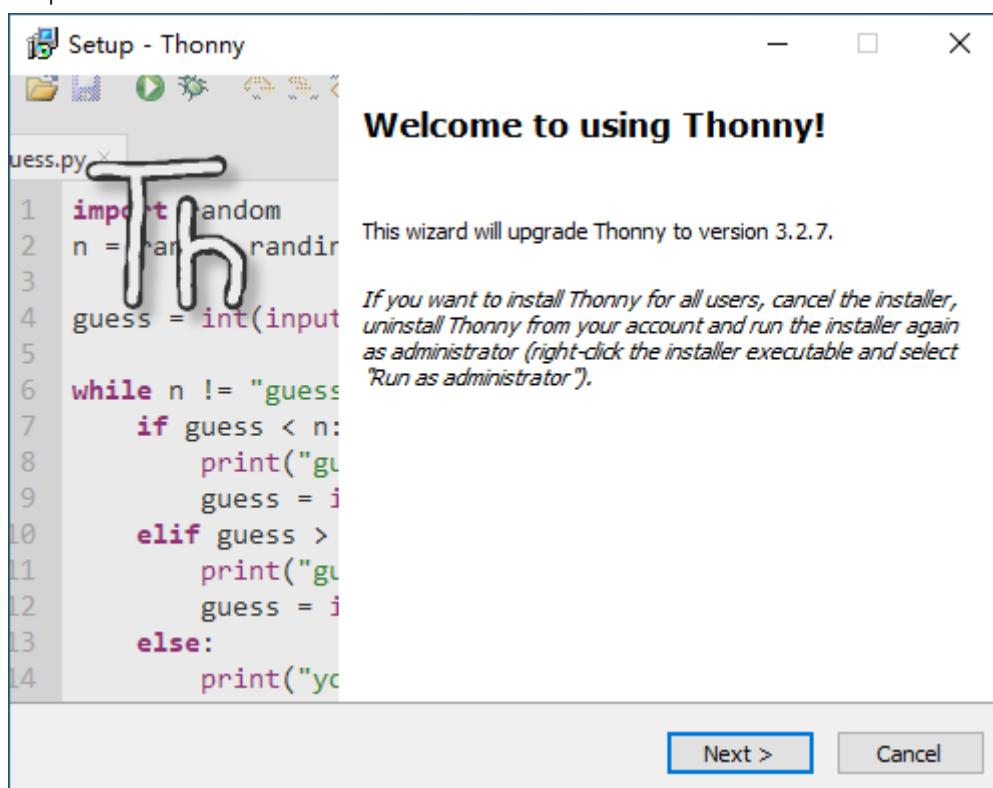


Installing on Windows

The icon of Thonny after downloading is as below. Double click “thonny-3.2.7.exe”.

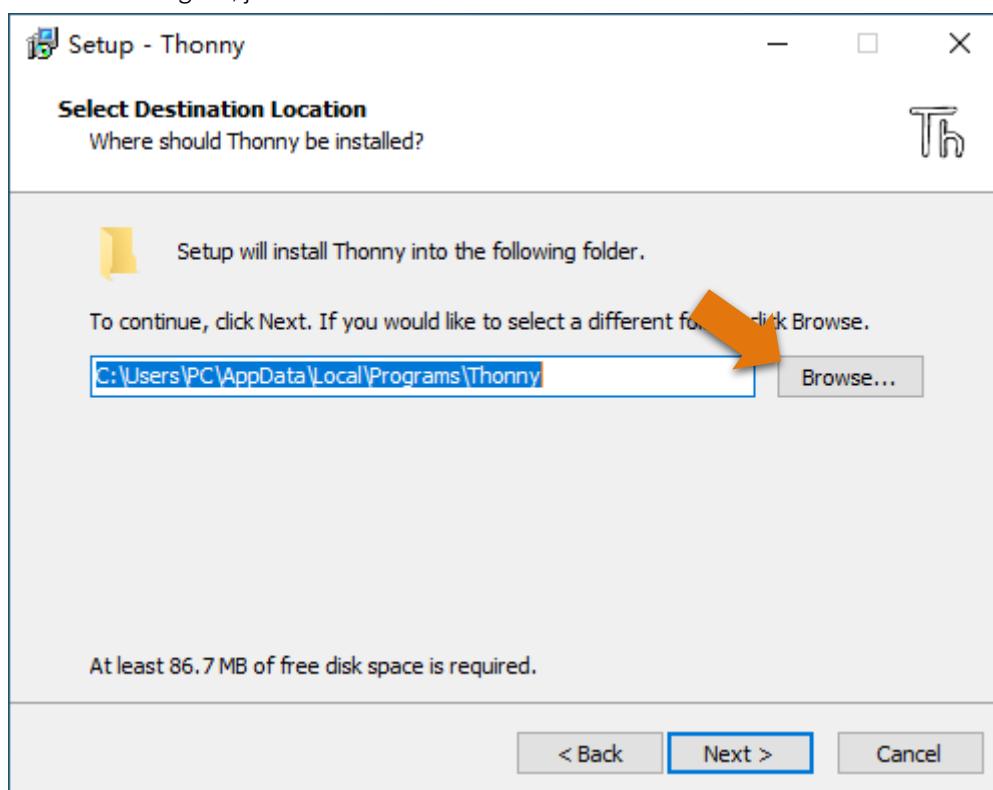


If you're not familiar with computer software installation, you can simply keep clicking "Next" until the installation completes.



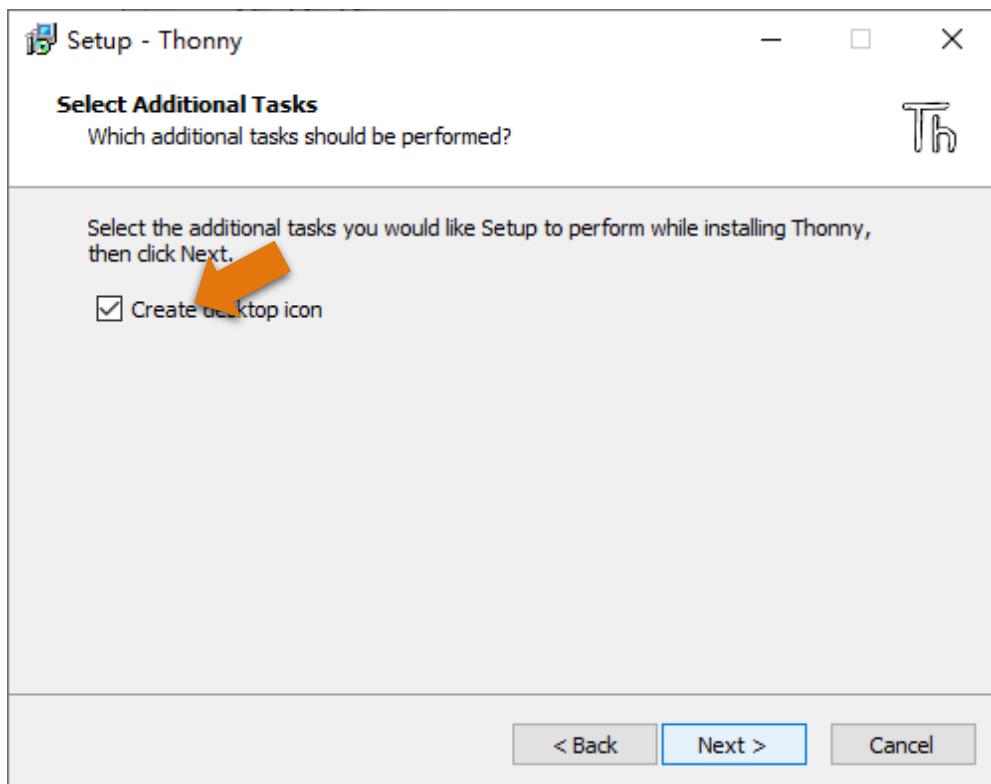
If you want to change Thonny's installation path, you can click "Browse" to modify it. After selecting installation path, click "OK".

If you do not want to change it, just click "Next".

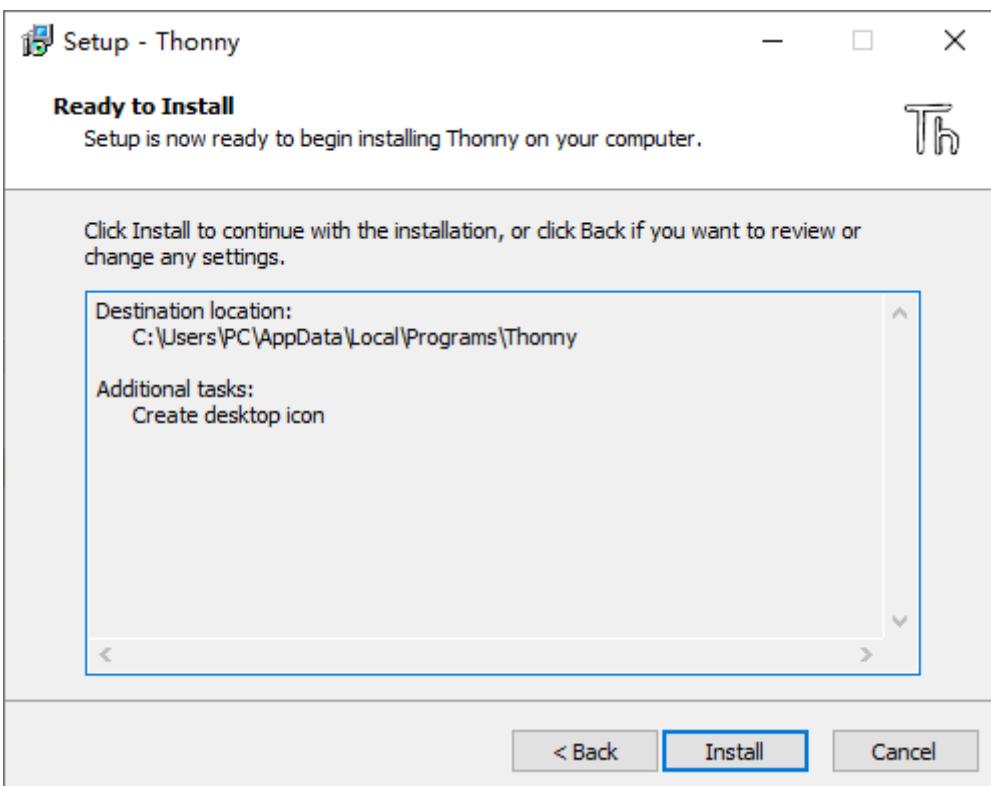




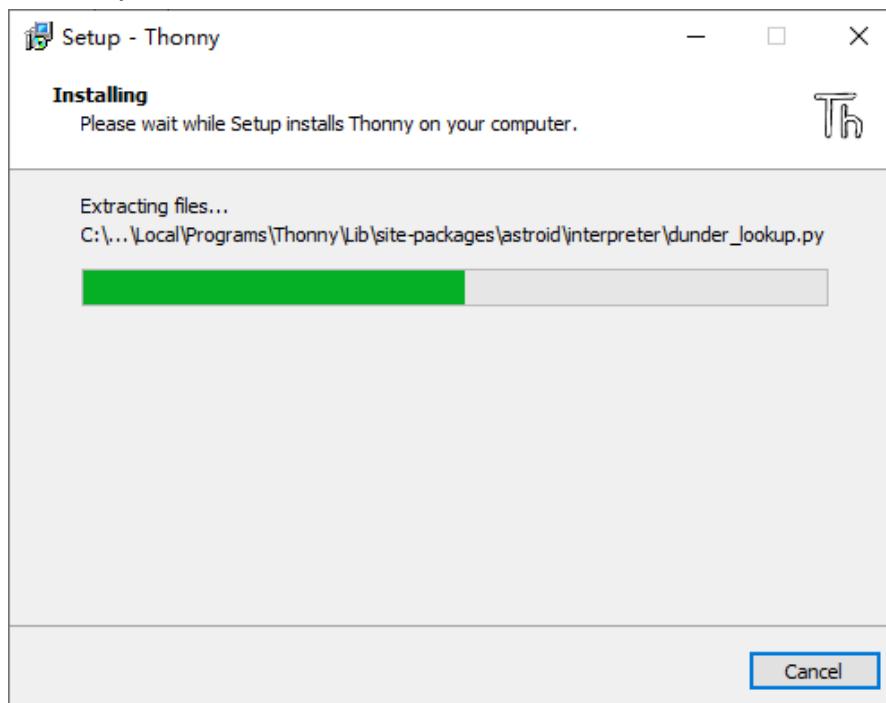
Check “Create desktop icon” and then it will generate a shortcut on your desktop to facilitate you to open Thonny later.



Click “install” to install the software.



During the installation process, you only need to wait for the installation to complete, and you must not click "Cancel", otherwise Thonny will fail to be installed.



Once you see the interface as below, Thonny has been installed successfully.



If you've checked "Create desktop icon" during the installation process, you can see the below icon on your desktop.



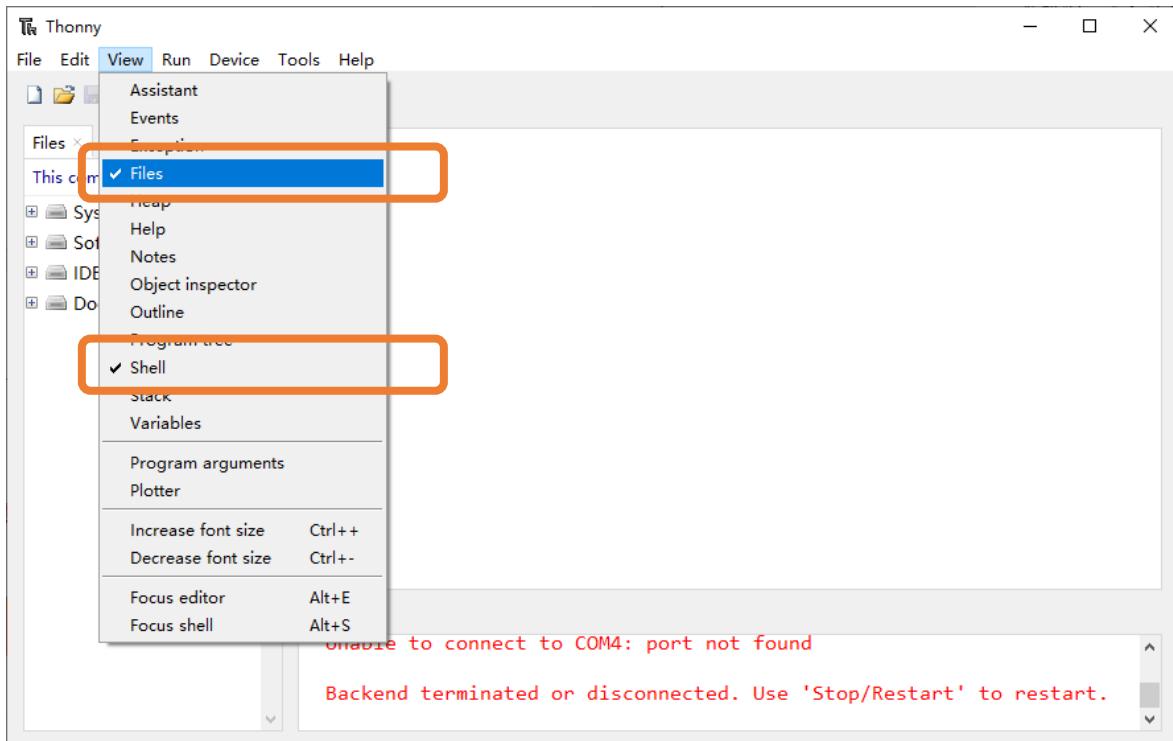


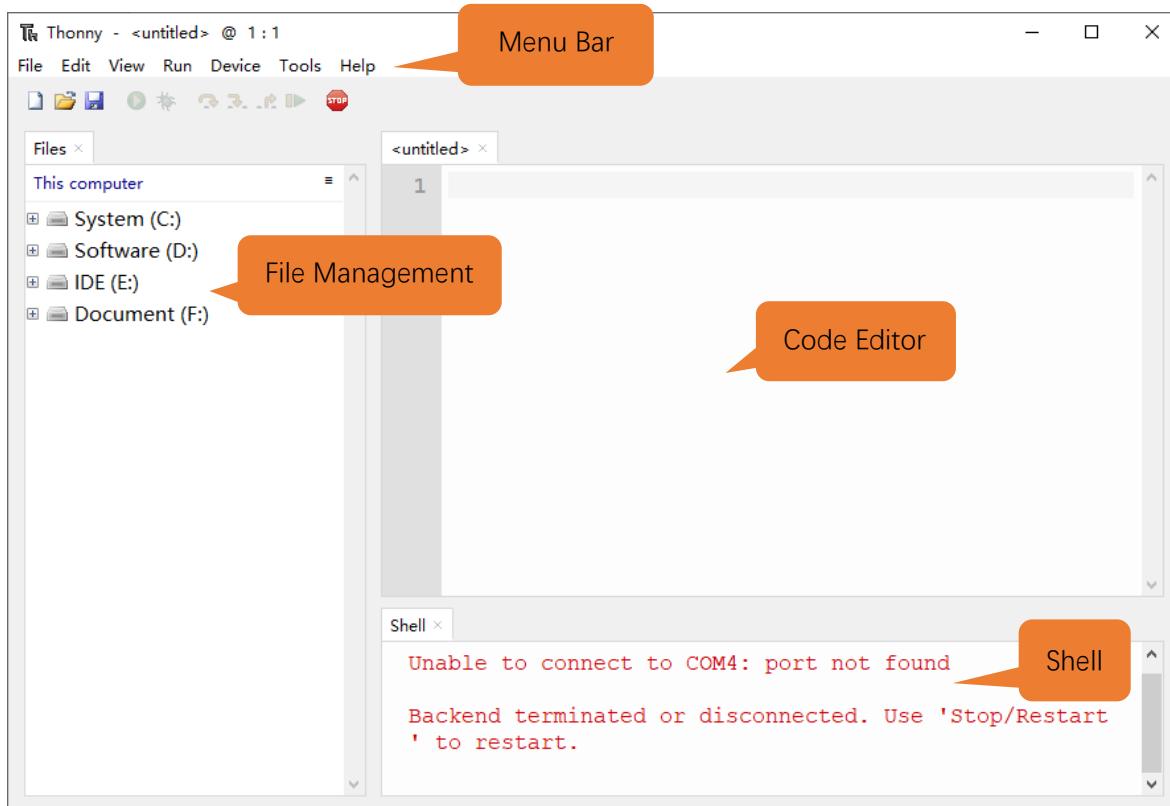
0.2 Basic Configuration of Thonny

Click the desktop icon of Thonny and you can see the interface of it as follows:



Select "View" → "Files" and "Shell".







0.3 Burning Micropython Firmware (Important)

To run Python programs on Raspberry Pi Pico, we need to burn a firmware to Raspberry Pi Pico first.

Downloading Micropython Firmware

Option 1:

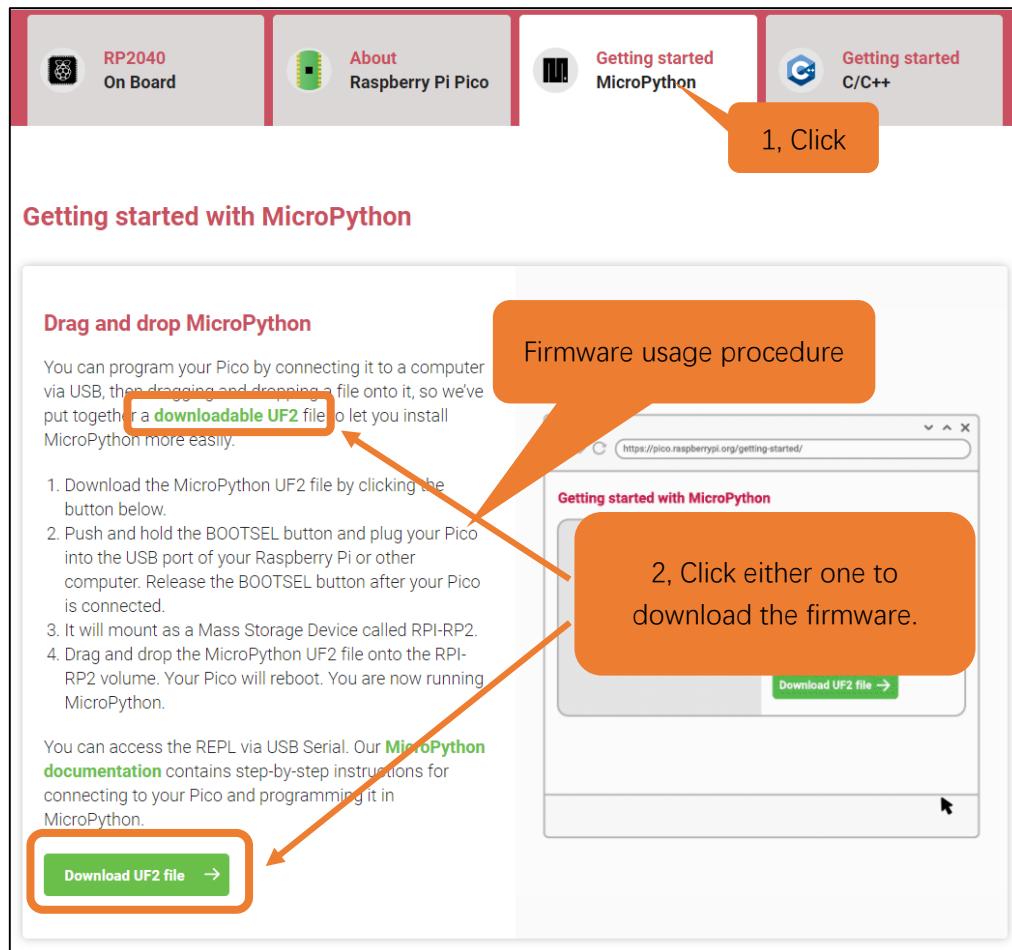
Raspberry Pi Pico official website: <https://www.raspberrypi.org/documentation/rp2040/getting-started/>

1, Click the link above and you can see the following interface

2, Roll the mouse and you can see the links for RP2040 documents.

Any concerns? ✉ support@freenove.com

3, Click "Getting started MicroPython" to enter the firmware downloading page.



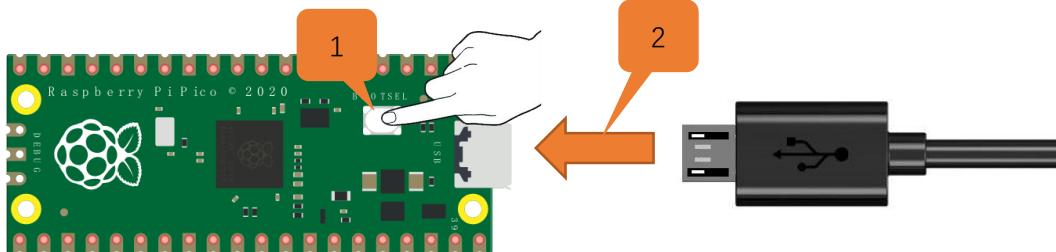
Option 2: Directly download via [rp2-pico-20210618-v1.16.uf2](https://rp2-pico.readthedocs.io/en/latest/getting-started/)

Option 3: If you cannot download it due to network issue or other reasons, you can use the one we have prepared, which locates at the following file path:

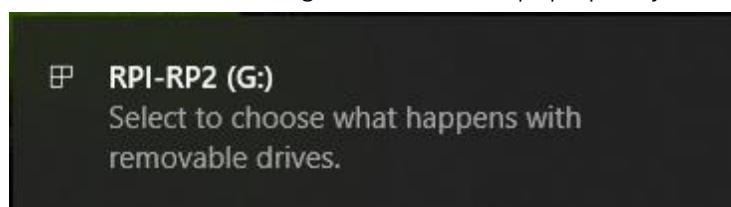
Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico/Python_Firmware

Burning a Micropython Firmware

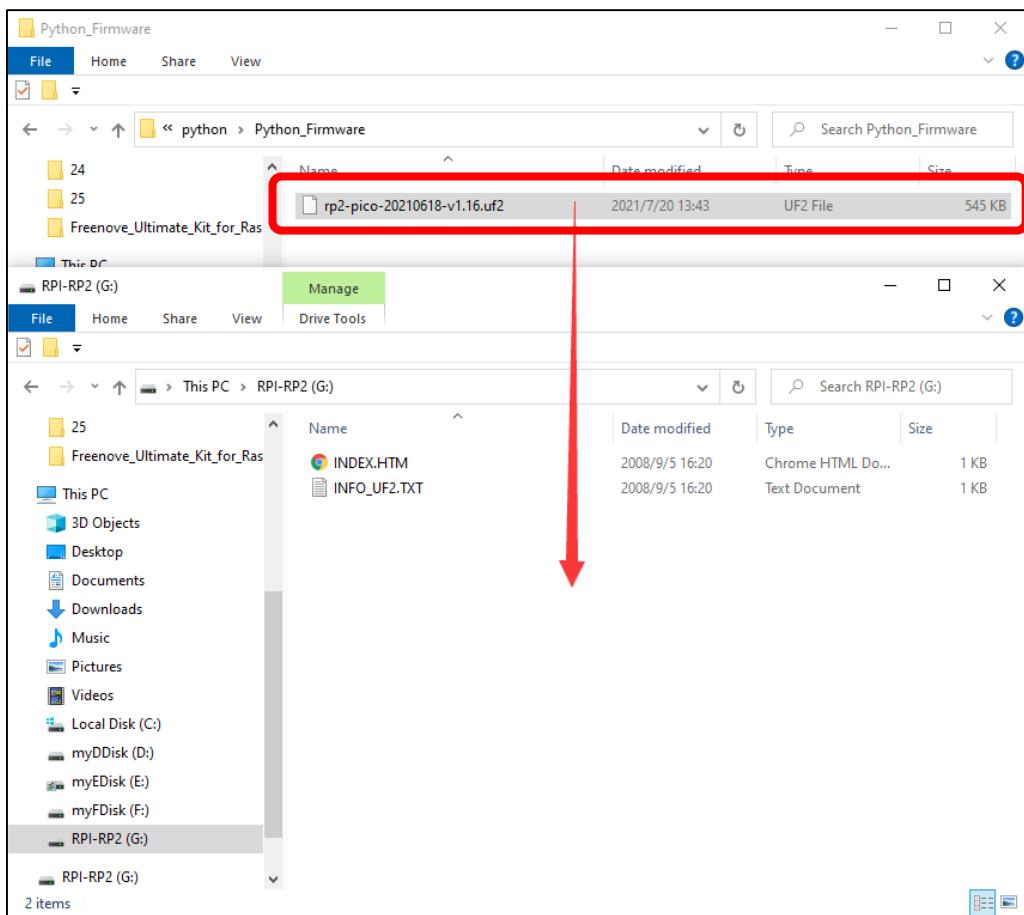
1. Connect a USB cable to your computer.
2. Long press BOOTSEL button on Raspberry Pi Pico and connect it to your computer with the USB cable.



3. When the connection succeeds, the following information will pop up on your computer.



4. Copy the file(**rp2-pico-20210618-v1.16.uf2**) to RPI-RP2 and wait for it to finish, just like copy file to a U disk.

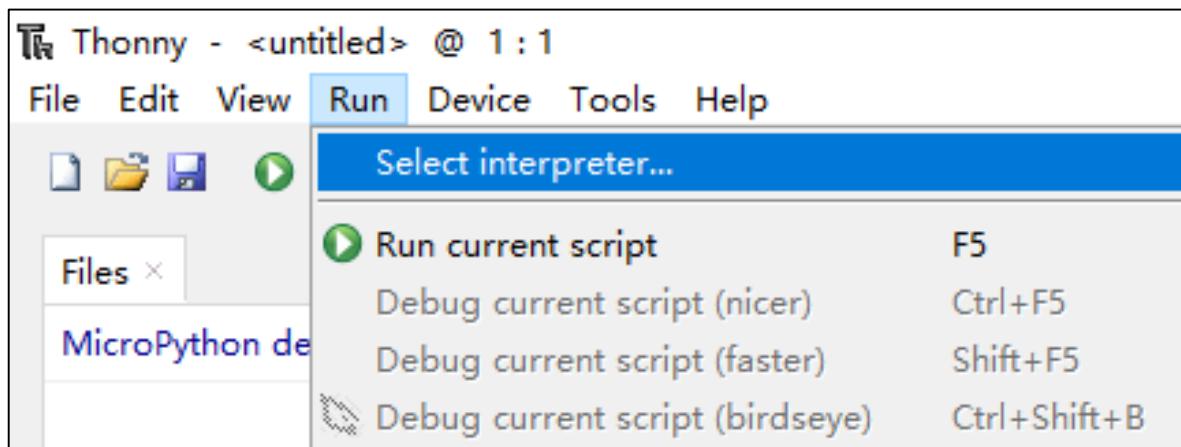


5. When the firmware finishes programming, Raspberry Pi Pico will reboot automatically. After that, you can run Micropython.

Any concerns? ✉ support@freenove.com

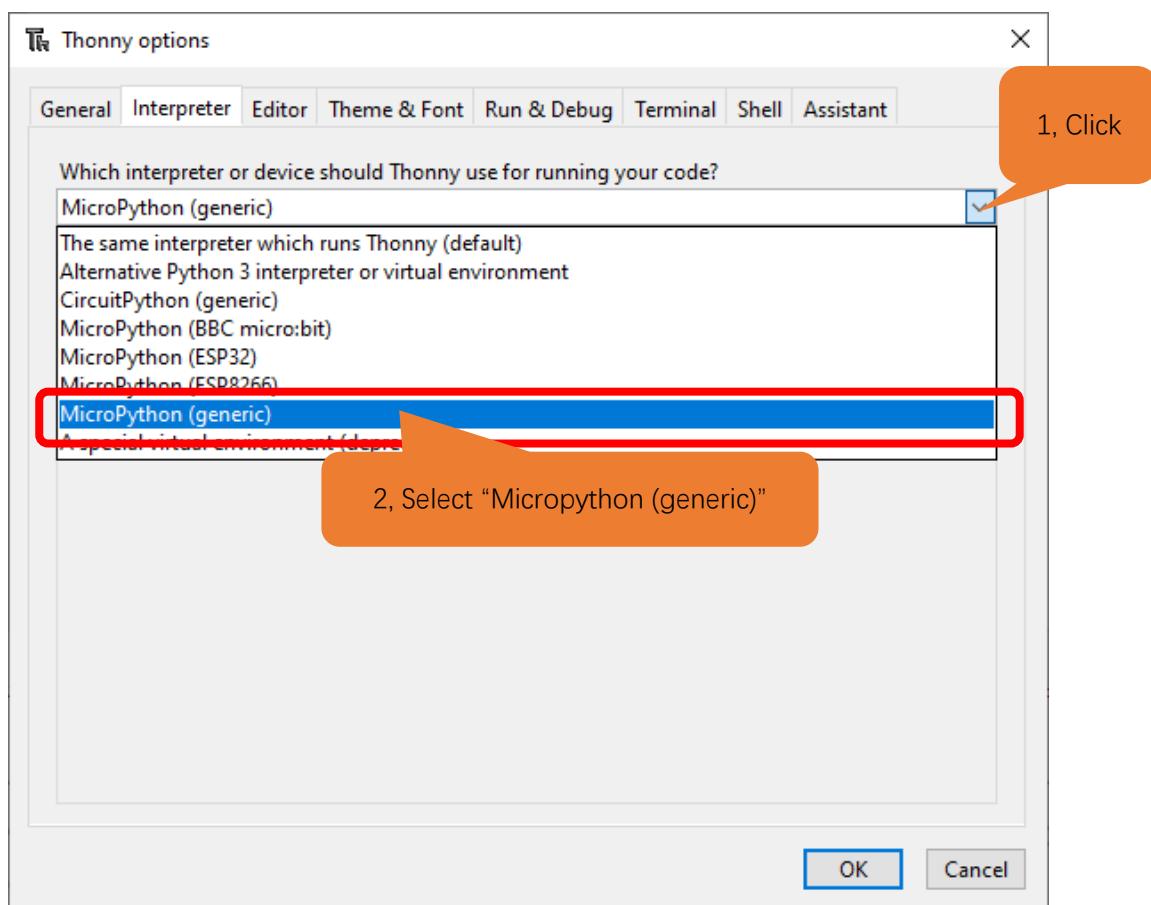
0.4 Thonny Connected to Raspberry Pi Pico

1. Open Thonny, click "run" and select "Select interpreter..."



2. Select "Micropython (generic)" or "Micropython (Raspberry Pi Pico)".

How to select "Micropython (generic)"? As shown below:

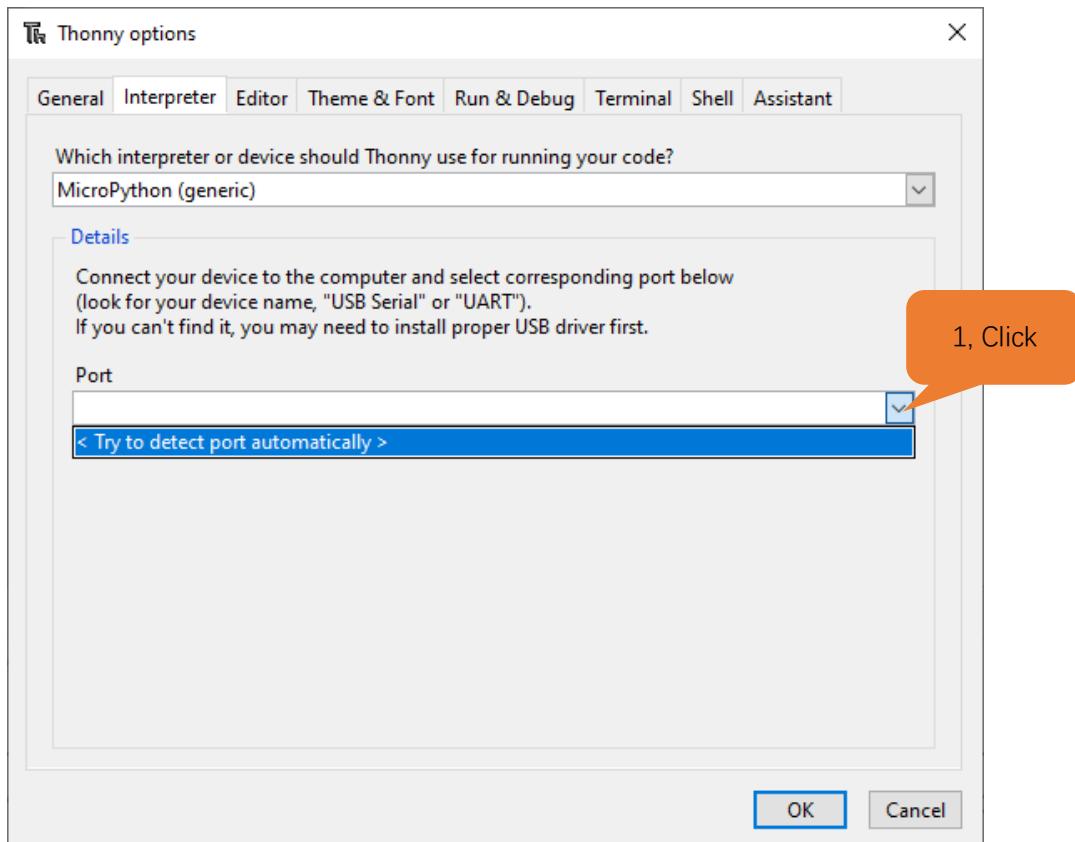




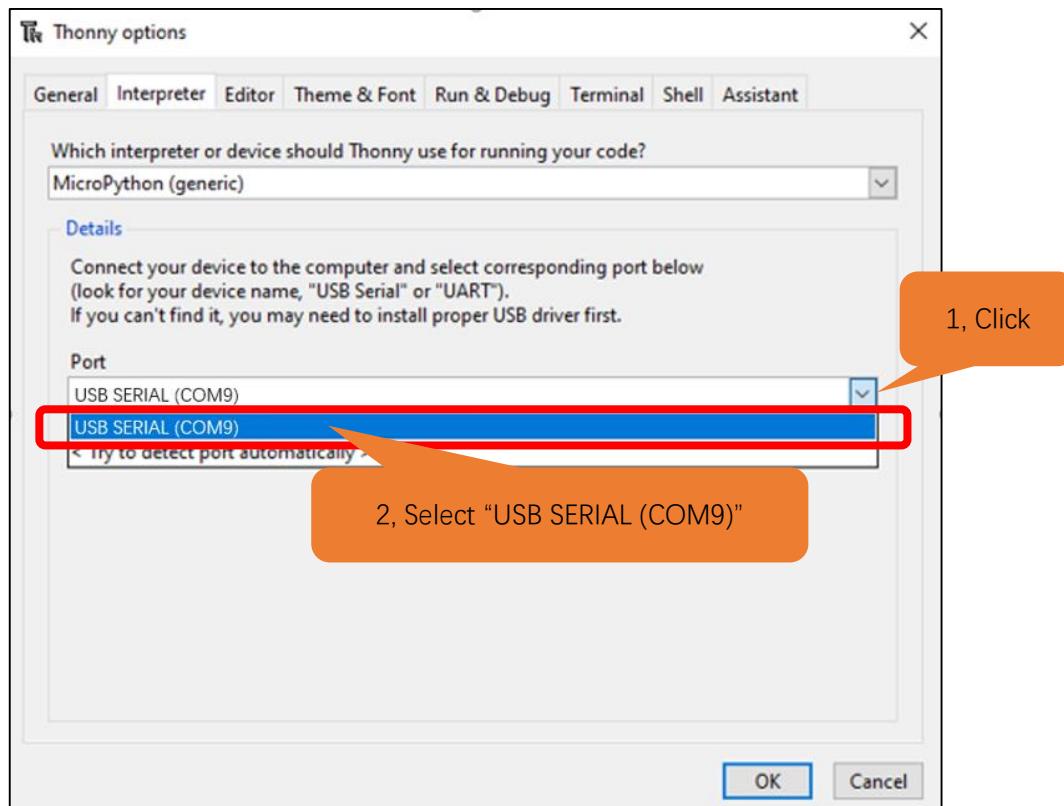
3. Select “USB-SERIAL (COMx)”, The number x of COMx may varies among different computers. You only need to make sure selecting USB-SERIAL (COMx).

How to determine the port on which your Raspberry Pi Pico communicates with your computer?

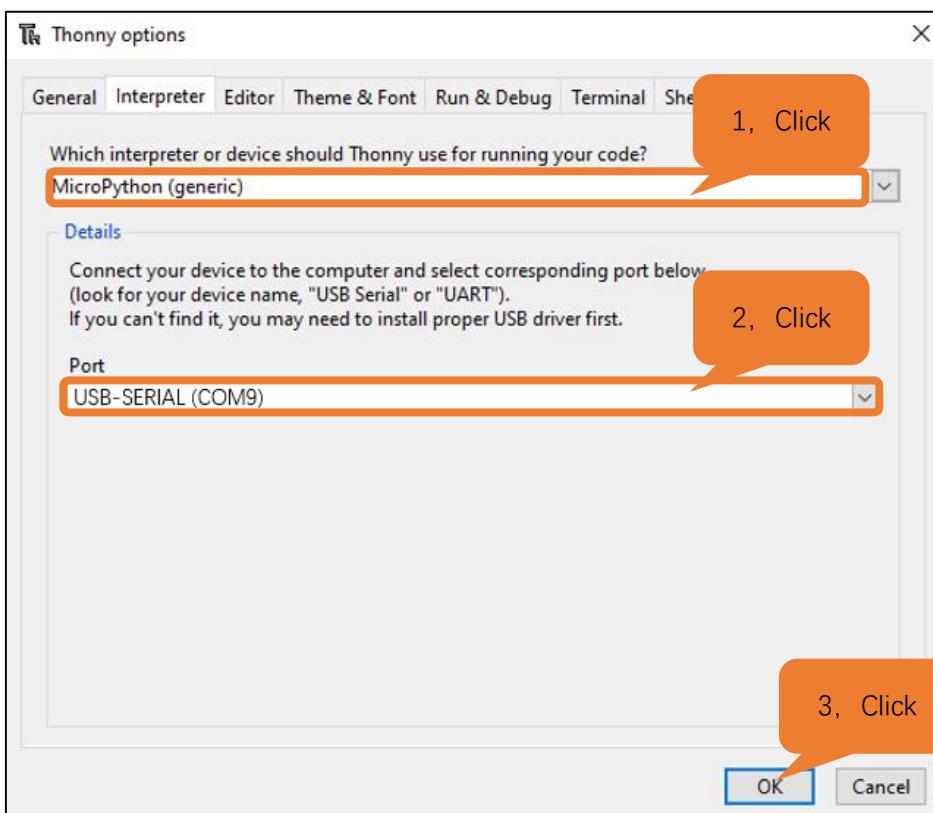
Step 1: When Pico **doesn't** connect to computer, open Thonny, click “Run”, select “Select interpreter” and then a dialog box will pop up, click “Port” and you can check the ports currently connected to your computer, as shown below:



Step 2: Close the dialog box. Connect Pico to your computer, click "Run" again and select "Select interpreter". Click "Port" on the pop-up window and check the current ports. Now there is a newly added port, with which Pico communicates with the computer.

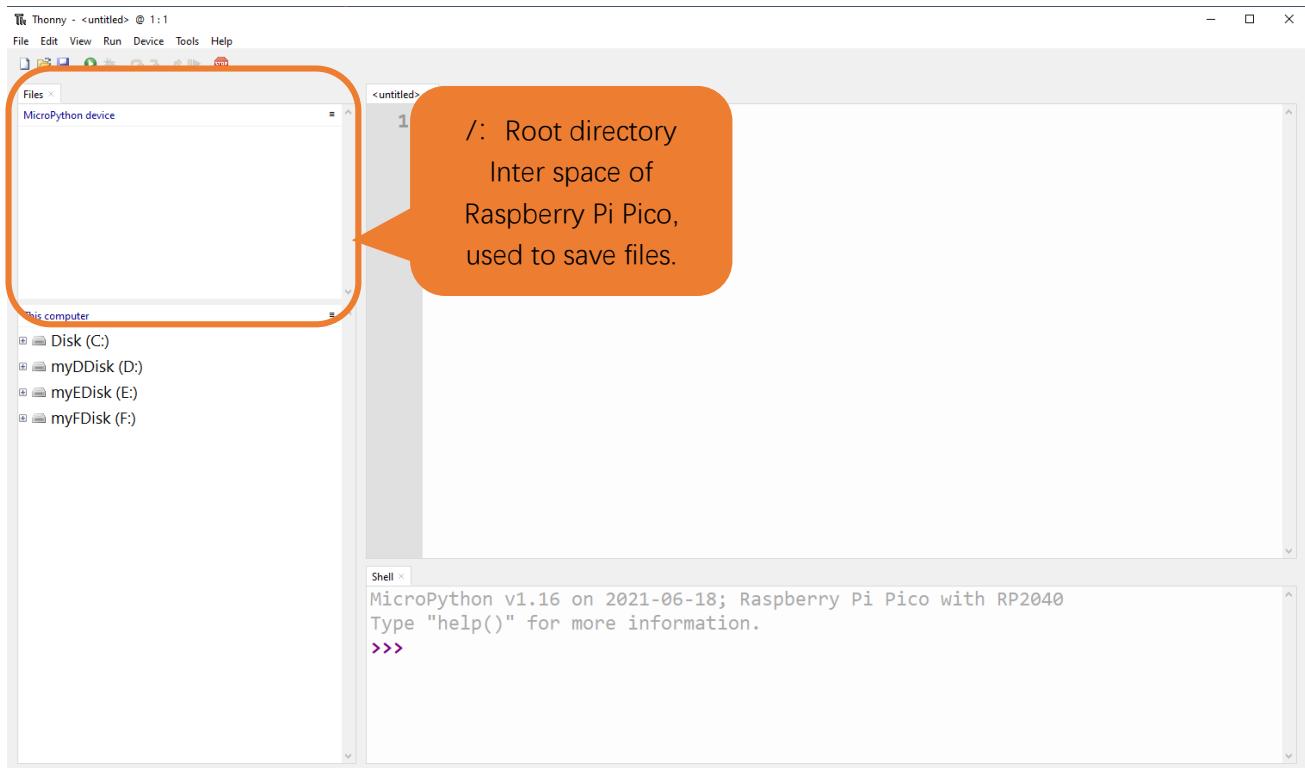


4. After selecting "Micropython (generic)" and port, click "OK"





5. When the following message displays on Thonny, it indicates Thonny has successfully connected to Pico.



So far, all the preparations have been made.

0.5 Testing codes (Important)

Testing Shell Command

Enter "print("hello world!")" in "Shell" and press Enter.

The screenshot shows the Thonny IDE interface. On the left is a file browser titled 'Files' showing a directory structure under 'MicroPython device'. A file named '00.0_HelloWorld' is selected. On the right is a 'Shell' window titled 'RP2040'. It displays the MicroPython version and help information, followed by the command 'print("hello world!")' and its output 'hello world!'. An orange callout bubble points to the output text with the text 'Successfully display'.

```
MicroPython v1.16 on 2021-06-18; Python v3.7.9, RTSPIB, WiFi
Type "help()" for more information
>>> print("hello world!")
hello world!
>>>
```

Successfully display

Running Online

To run Raspberry Pi Pico online, you need to connect it to computer. Users can use Thonny to compile or debug programs.

Advantages:

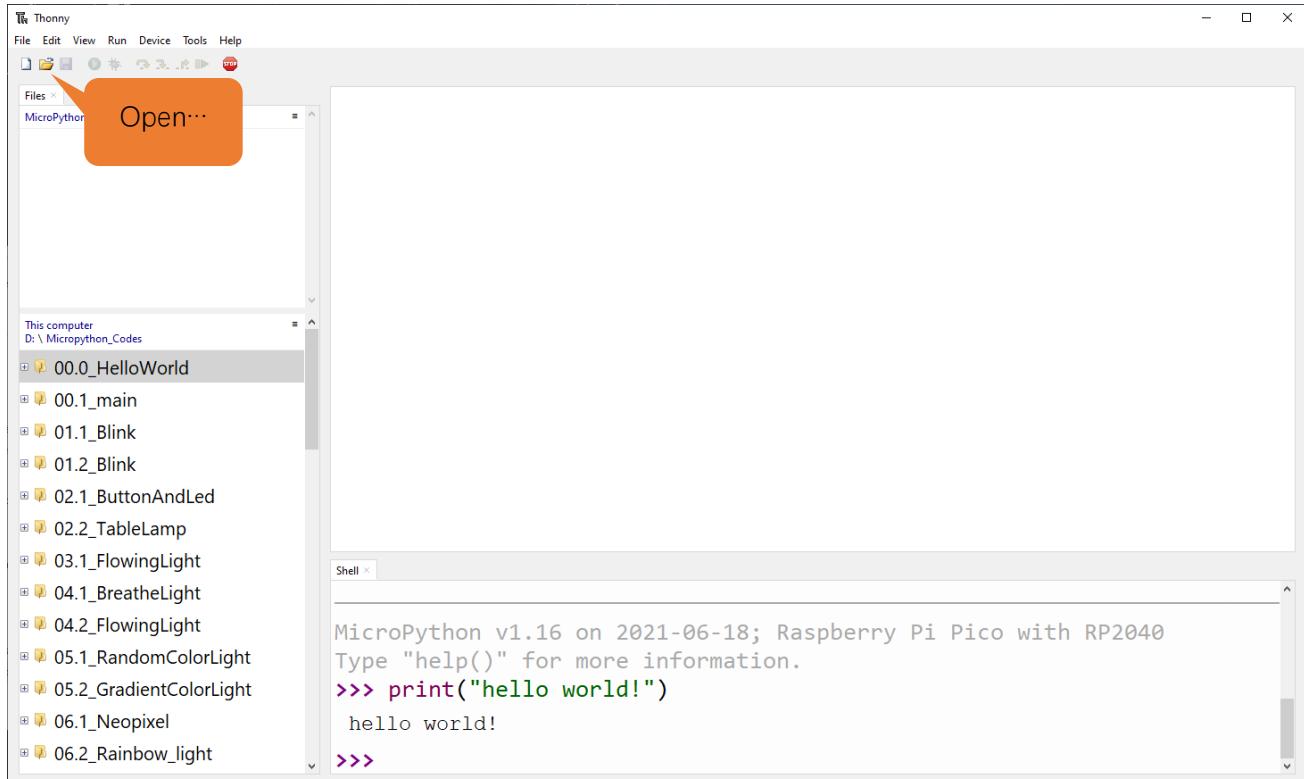
1. Users can use Thonny to compile or debug programs.
2. Through the "Shell" window, users can read the error information and output results generated during the running of the program and query related function information online to help improve the program.

Disvantages:

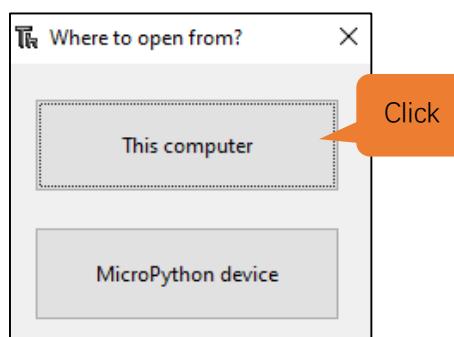
1. To run Raspberry Pi Pico online, you have to be connected to a computer and run with Thonny.
2. If Raspberry Pi Pico disconnects from computer, the program won't run again when they reconnect to each other.

Operatation

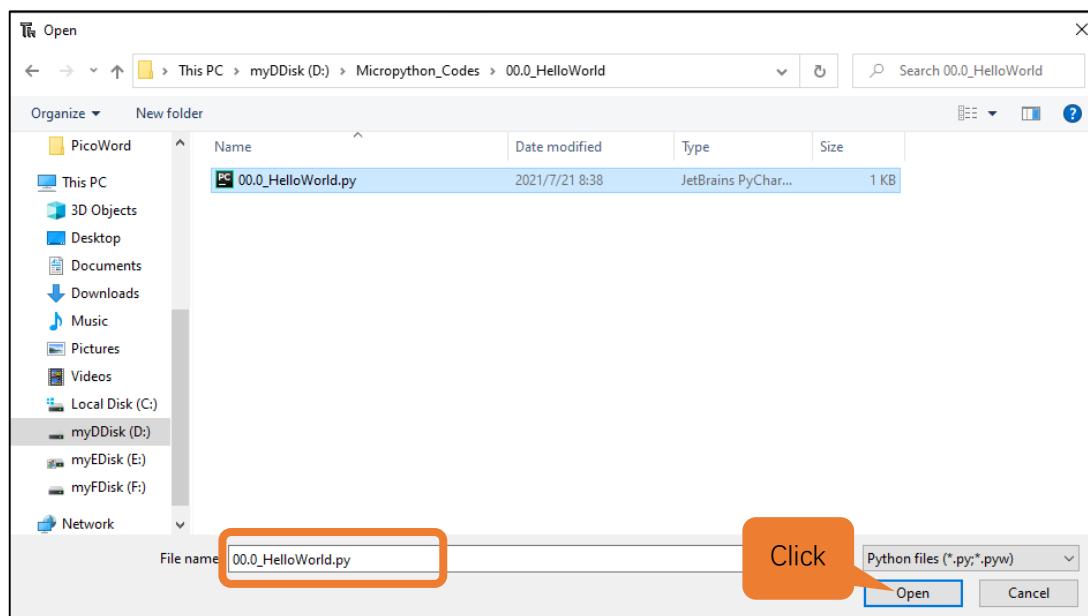
1. Open Thonny and click "Open…".



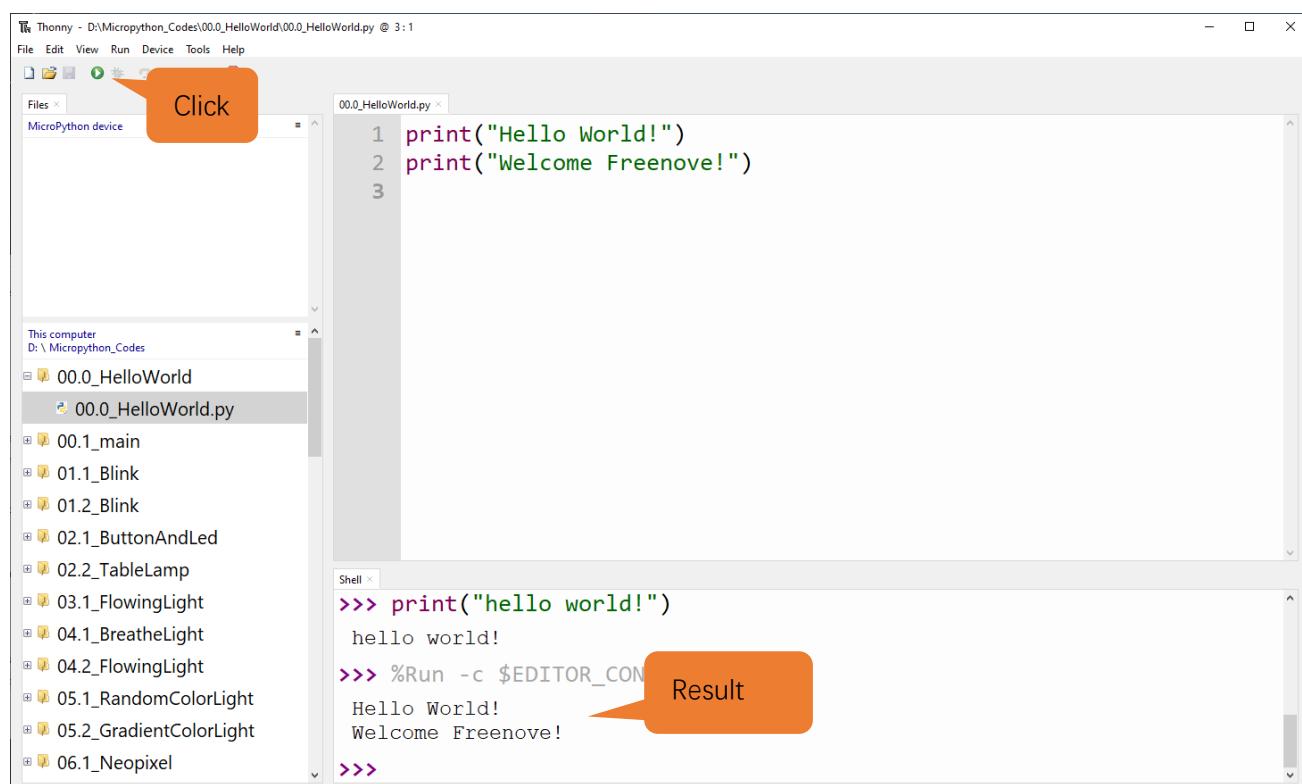
2. On the newly pop-up window, click "This computer".



In the new dialog box, select “**00.0_HelloWorld.py**” in “**Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico/00.0_HelloWorld**” folder.

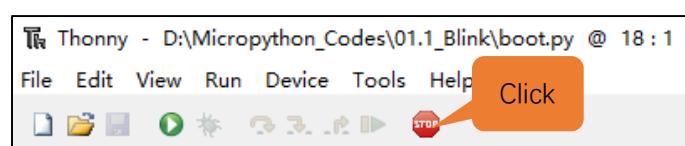


Click “Run current script” to execute the program and “Hello World!”, “Welcome Freenove” will be printed in “Shell”.



Exiting Running Online

When running online, click “Stop /Restart backend” on Thonny or press Ctrl+C to exit the program.



Running Offline

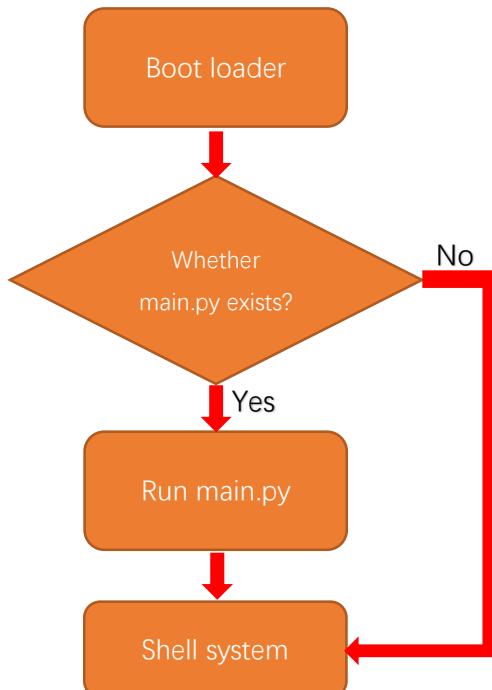
When running offline, Raspberry Pi Pico doesn't need to connect to computer and Thonny. It can run the programs stored in main.py on the device once powered up.

Advantage: It can run programs when powered up without connected to computer and Thonny.

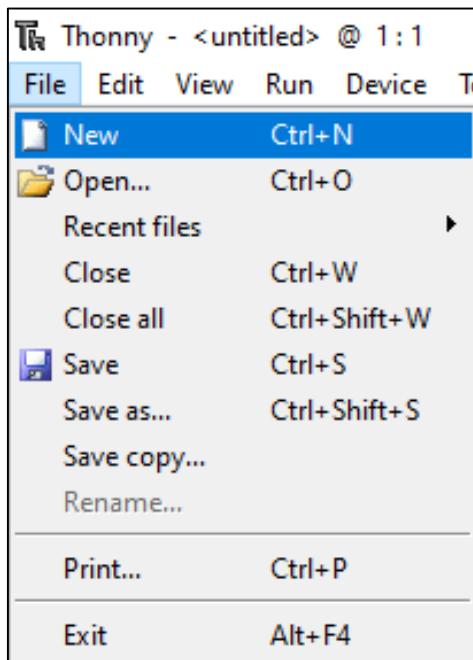
Disvantage: The program will stop automatically when error occurs or Raspberry Pi Pico is out of power. Code cannot be changed easily.

Operation

Once powered up, Raspberry Pi Pico will automatically check whether there is main.py existing on the device. If there is, it runs the programs in main.py and then enter shell command system. (If you want the code to run offline, you can save it as main.py); If main.py doesn't exist, it will enter shell command system directly.



1. Click “File”→“New” to create and write codes.



2. Enter codes in the newly opened file. Here we use codes of “01.1_Blink.py” as an example.

The screenshot shows the Thonny IDE interface. On the left, there's a file tree titled "Files" showing various MicroPython code files. In the center, a code editor window titled "<untitled> * x" contains the following Python code:

```
from machine import Pin
import time

led = Pin(25, Pin.OUT)      # create LED object from Pin 25, Set Pin

try:
    while True:
        led.value(1)      # Set led turn on
        time.sleep(0.5)   # Sleep 0.5s
        led.value(0)      # Set led turn off
        time.sleep(0.5)   # Sleep 0.5s
except:
    pass
```

Below the code editor is a "Shell" window showing the output of running the code:

```
>>> print("hello world!")
hello world!
>>> %Run -c $EDITOR_CONTENT
Hello World!
Welcome Freenove!
```



3. Click “Save” on the menu bar. You can save the codes either to your computer or to Raspberry Pi Pico.

```

from machine import Pin
import time

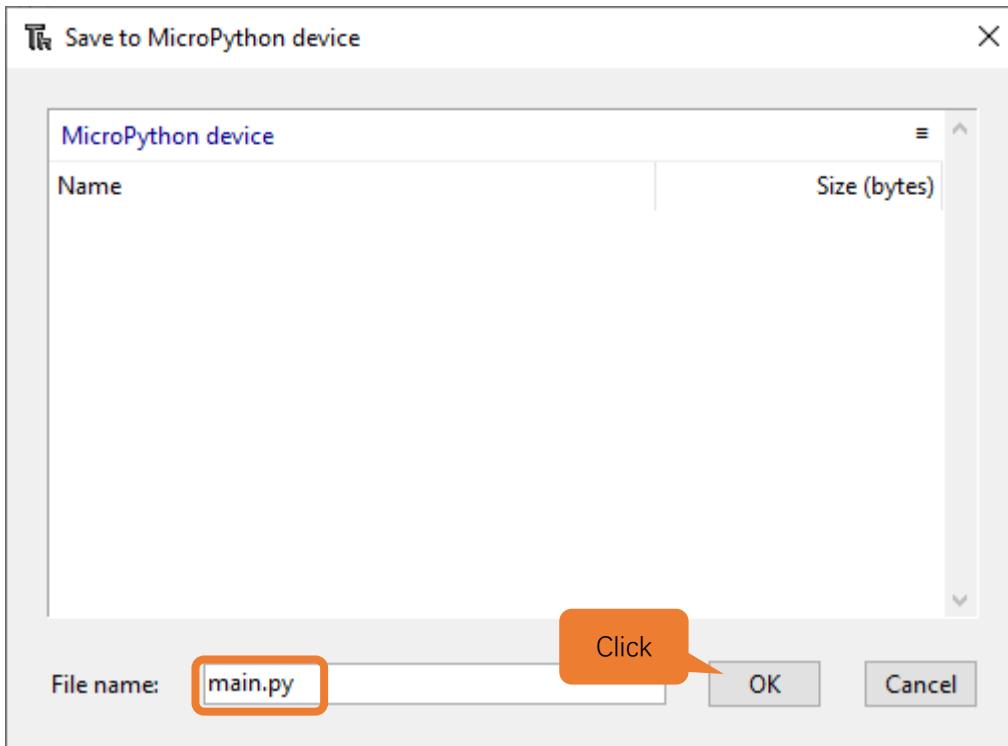
led = Pin(25, Pin.OUT)      # create LED object from Pin 25, Set Pin

try:
    while True:
        led.value(1)      # Set led turn on
        time.sleep(0.5)
        led.value(0)      # Set led turn off
        time.sleep(0.5)
except:
    pass

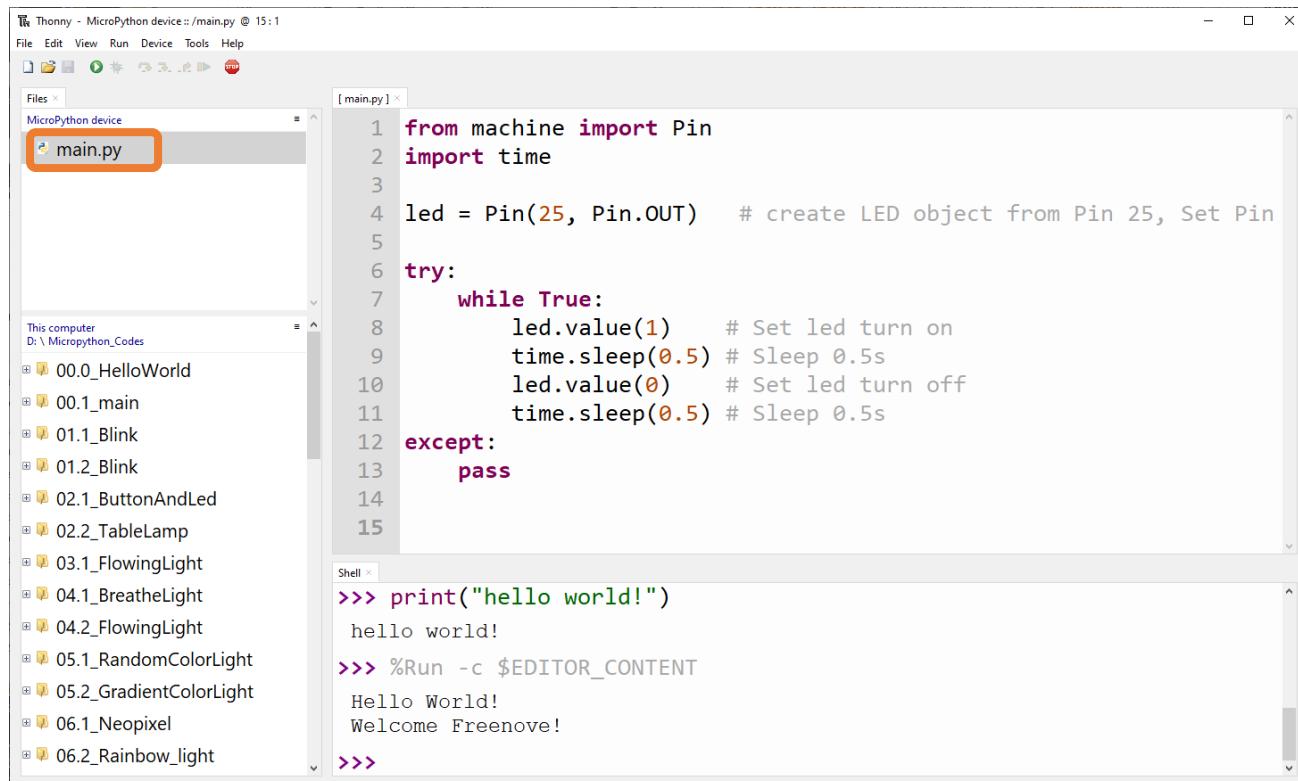
```

>>> print("hello world!")
hello world!
>>> %Run -c \$EDITOR_CONTENT
Hello World!
Welcome Freenove!
>>>

4. Select “MicroPython device”, enter “main.py” in the newly pop-up window and click “OK”.



5. You can see that codes have been uploaded to Raspberry Pi Pico.



The screenshot shows the Thonny IDE interface. The left sidebar lists files under 'MicroPython device' and 'This computer'. The main area shows the code for 'main.py':

```
from machine import Pin
import time

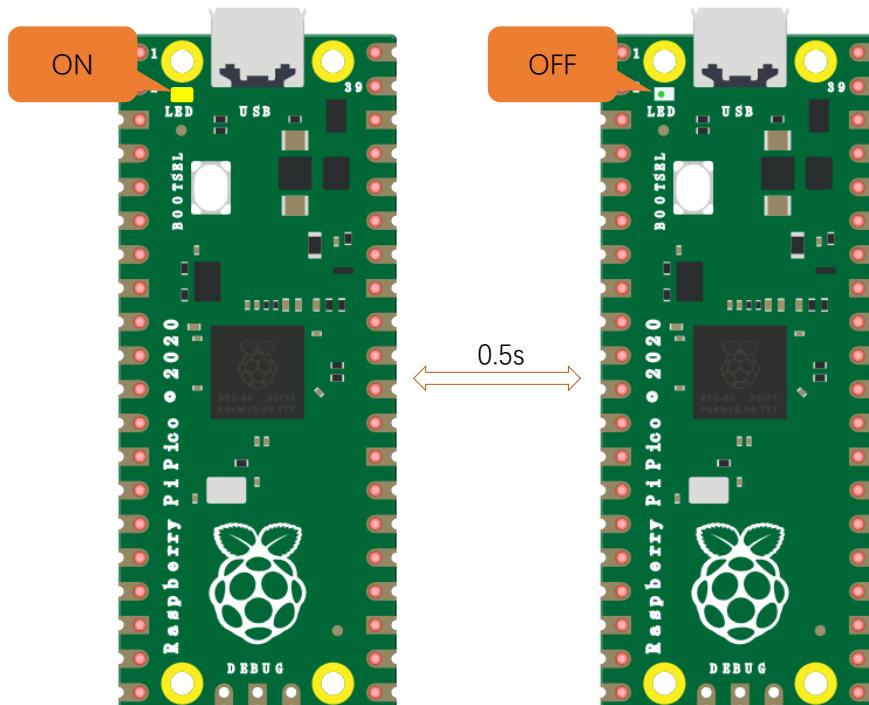
led = Pin(25, Pin.OUT) # create LED object from Pin 25, Set Pin

try:
    while True:
        led.value(1) # Set led turn on
        time.sleep(0.5) # Sleep 0.5s
        led.value(0) # Set led turn off
        time.sleep(0.5) # Sleep 0.5s
except:
    pass
```

The 'Shell' tab at the bottom shows the output of running the code:

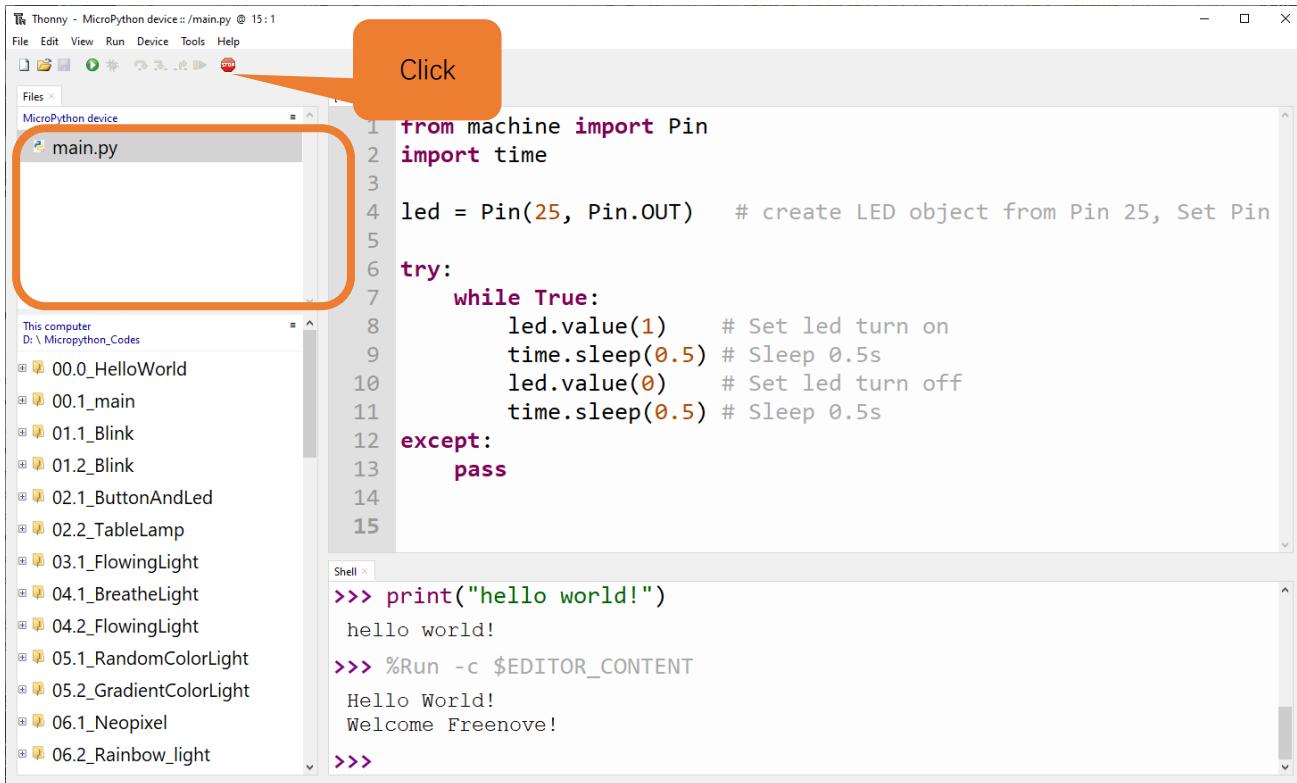
```
>>> print("hello world!")
hello world!
>>> %Run -c $EDITOR_CONTENT
Hello World!
Welcome Freenove!
>>>
```

6. Disconnect Raspberry Pi Pico USB cable and then reconnect it, the LED on Raspberry Pi Pico will blink repeatedly.



Exiting Offline Running

Connect Raspberry Pi Pico to computer, click “stop/restart backend” on Thonny to end running offline.



```

from machine import Pin
import time

led = Pin(25, Pin.OUT) # create LED object from Pin 25, Set Pin

try:
    while True:
        led.value(1) # Set led turn on
        time.sleep(0.5)
        led.value(0) # Set led turn off
        time.sleep(0.5)
except:
    pass

```

Shell

```

>>> print("hello world!")
hello world!
>>> %Run -c $EDITOR_CONTENT
Hello World!
Welcome Freenove!
>>>

```

If it doesn't work, please click on “stop/restart backend” for more times or reconnect Pico.



Computer cannot read
Pico internal files.

Shell

```

Type "help()" for more information.
>>> D<thonny>('': {'19.1_IIC_LCD1602.py': {'size': 428, 'kind': 'file', 'time': 2556144116}, 'main.py': {'size': 617, 'kind': 'file', 'time': 2556144135}, 'I2C_LCD.py': {'size': 3160, 'kind': 'file', 'time': 2556144118}, 'LCD_API.py': {'size': 6725, 'kind': 'file', 'time': 2556144121}})</thonny>

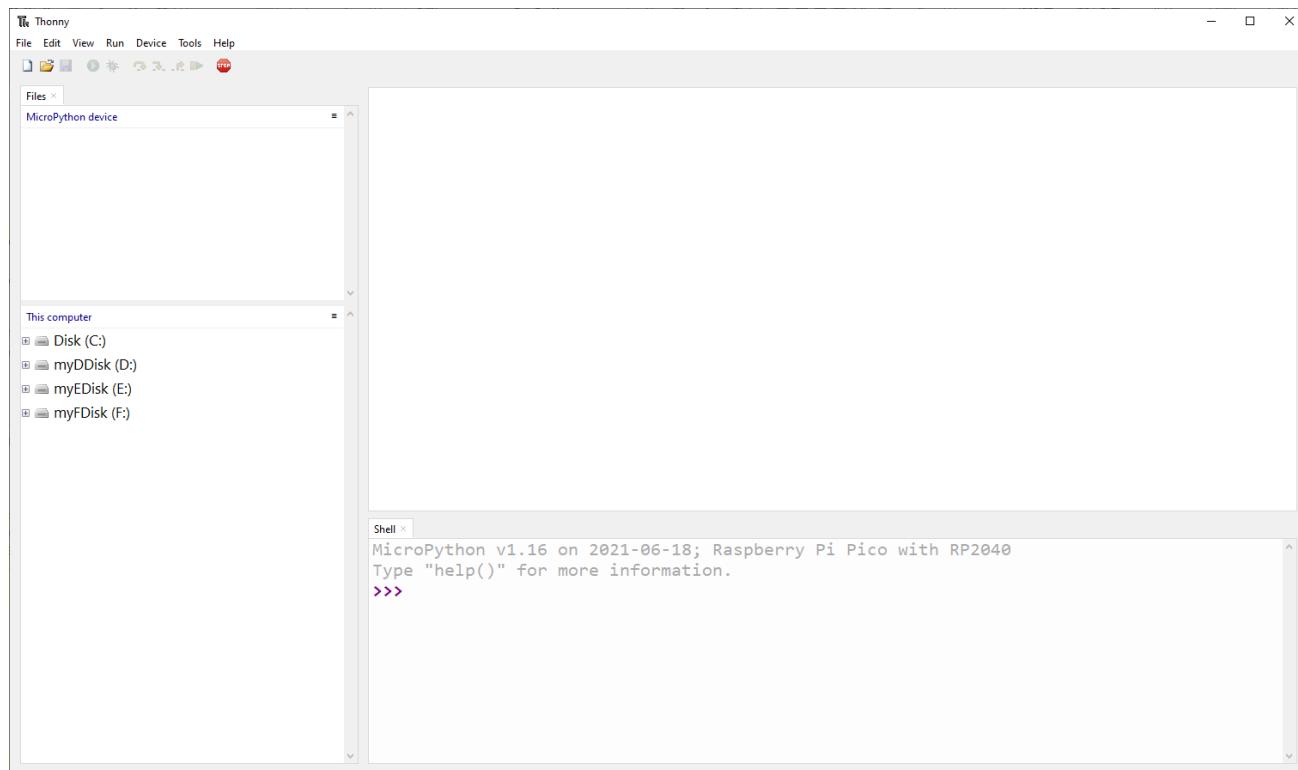
MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> D<thonny>('': {'19.1_IIC_LCD1602.py': {'size': 428, 'kind': 'file', 'time': 2556144116}, 'main.py': {'size': 617, 'kind': 'file', 'time': 2556144135}, 'I2C_LCD.py': {'size': 3160, 'kind': 'file', 'time': 2556144118}, 'LCD_API.py': {'size': 6725, 'kind': 'file', 'time': 2556144121}})</thonny>

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> D<thonny>('': {'19.1_IIC_LCD1602.py': {'size': 428, 'kind': 'file', 'time': 2556144116}, 'main.py': {'size': 617, 'kind': 'file', 'time': 2556144135}, 'I2C_LCD.py': {'size': 3160, 'kind': 'file', 'time': 2556144118}, 'LCD_API.py': {'size': 6725, 'kind': 'file', 'time': 2556144121}})</thonny>

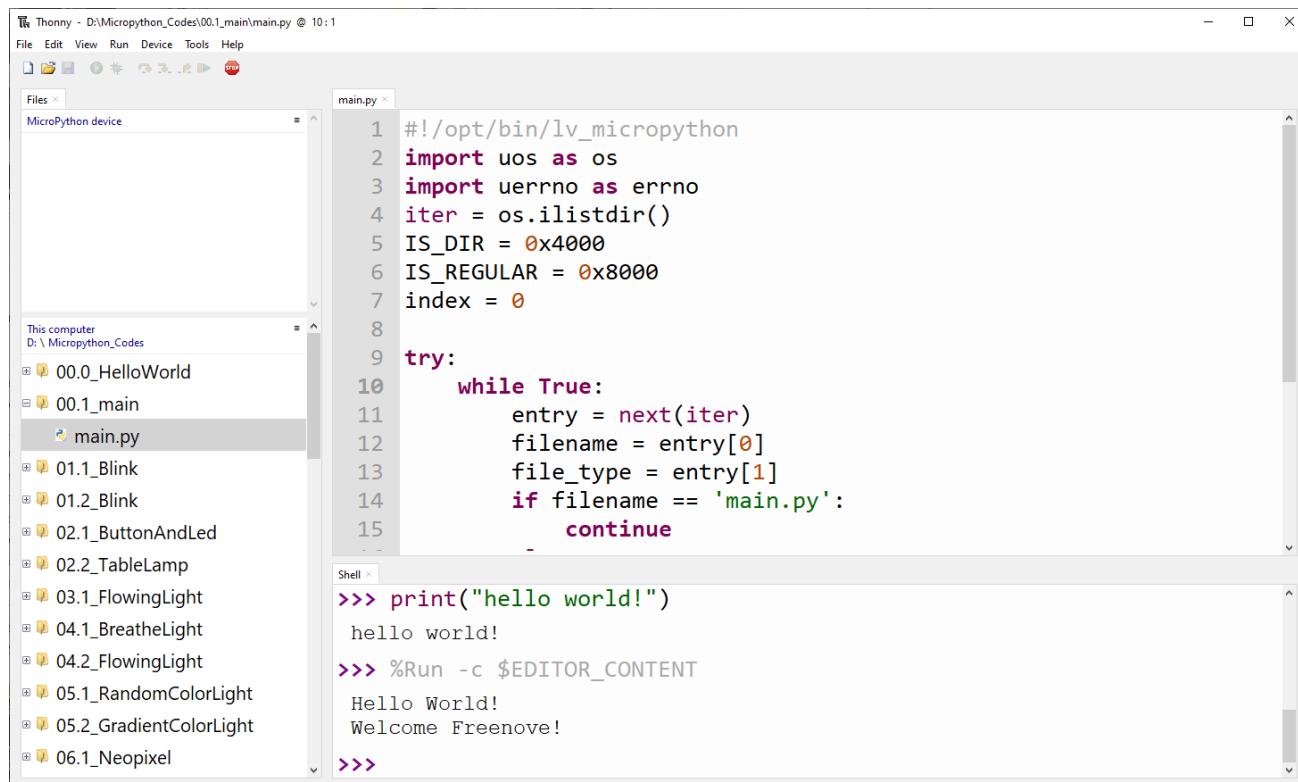
```

We provide a main.py file for running offline. The code added to main.py is a bootstrap that executes the user's code file. All you need to do is upload the offline project's code file (.py) to the Raspberry Pi Pico device.

- Move the program folder “**Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico/Python_Codes**” to disk(D) in advance with the path of “**D:/Micropython_Codes**”. Open “Thonny”.



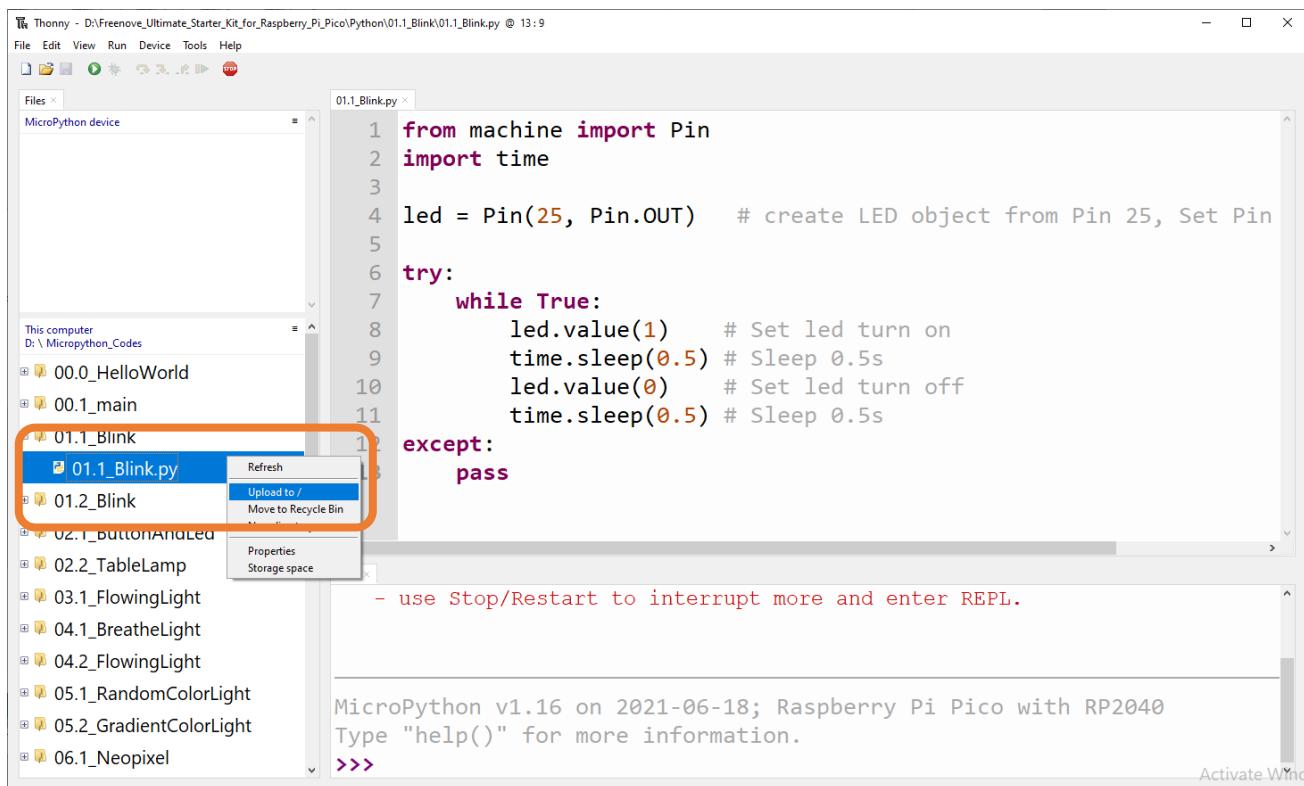
- Expand “00.1_main” in the “Micropython_Codes” in the directory of disk(D), and double-click main.py, which is provided by us to enable programs in “MicroPython device” to run offline.



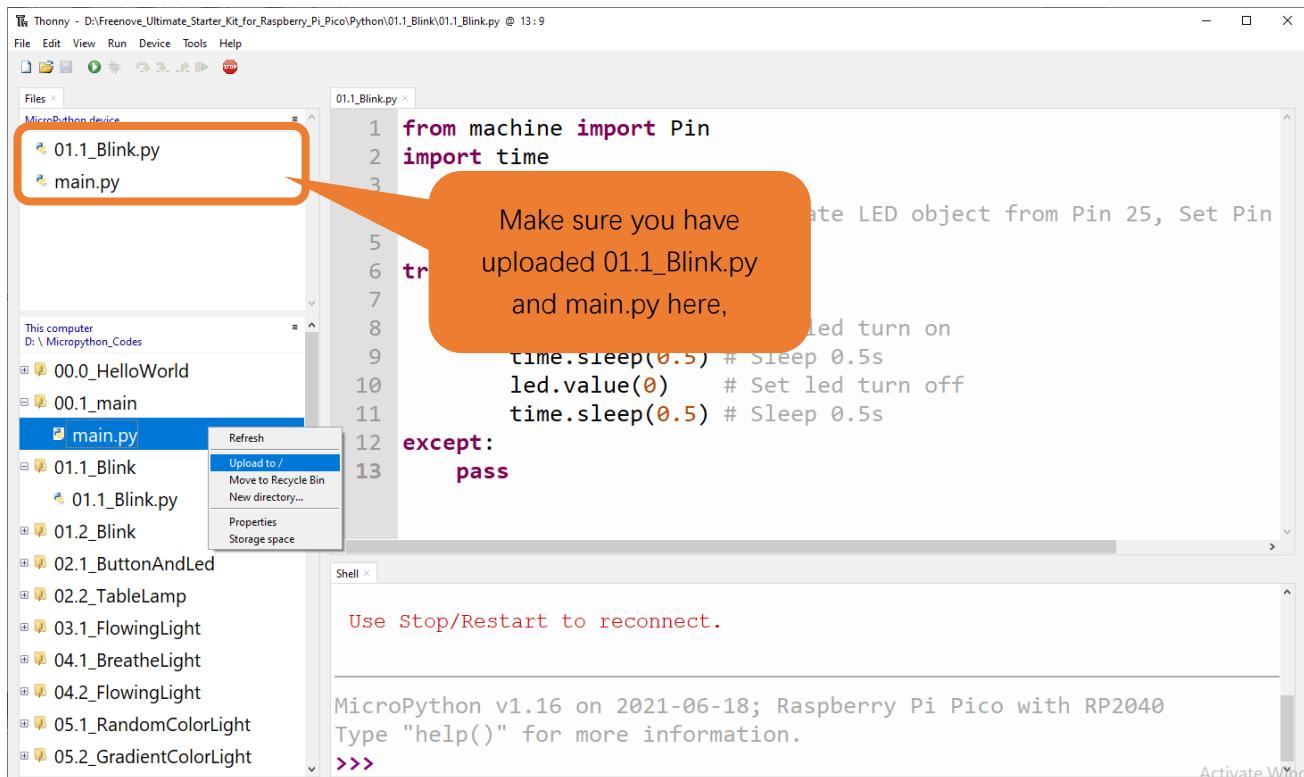


Here we use 00.1 and 01.1 cases as demonstration. The LED on Raspberry Pi Pico is used to show the result, which uses GP25 pin. If you have modified 01.1_Blink.py file, you need to change it accordingly.

As shown in the following illustration, right-click the file 01.1_Blink.py and select “Upload to /” to upload code to Raspberry Pi Pico.



Upload main.py in the same way.

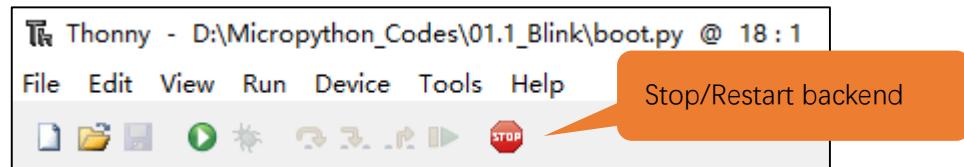


Disconnect Raspberry Pi Pico USB cable and reconnect it, the LED on pico will blink repeatedly.

Any concerns? ✉ support@freenove.com

Note:

Codes here are run offline. If you want to stop running offline and enter Shell, just click "Stop" in Thonny.

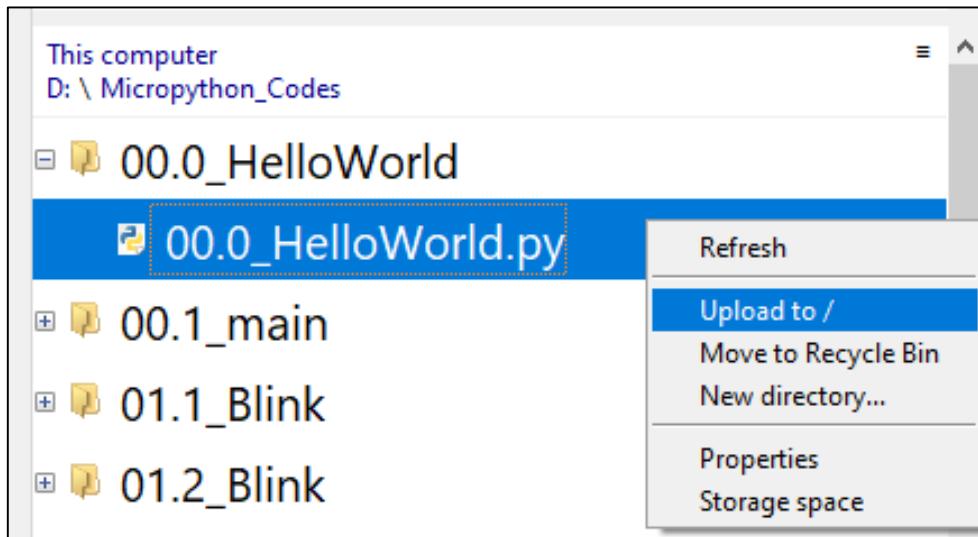




0.6 Thonny Common Operation

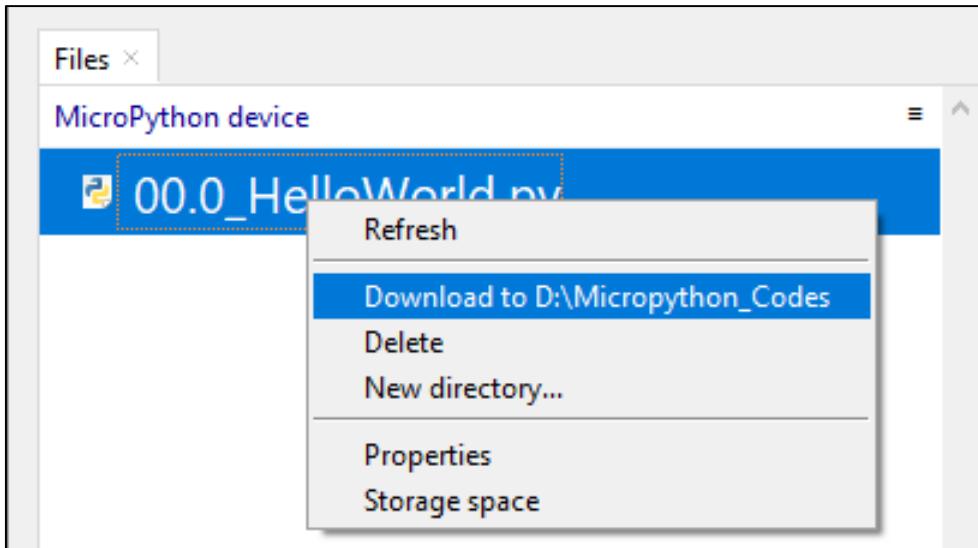
Uploading Code to Raspberry Pi Pico

Select “00.0_HelloWorld.py” in “00.0_HelloWorld”, right-click your mouse and select “Upload to /” to upload code to Raspberry Pi Pico’s root directory.



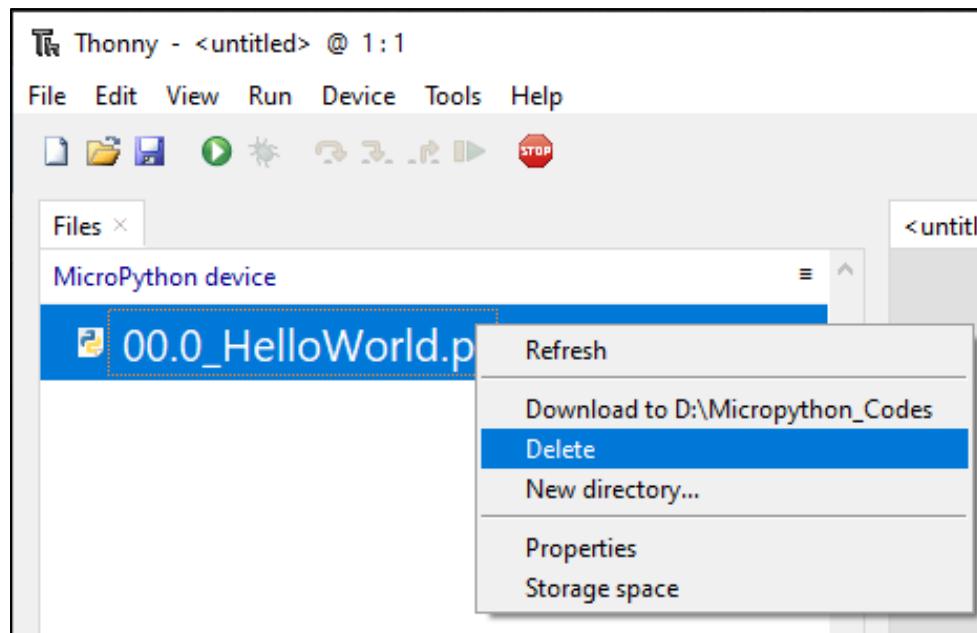
Downloading Code to Computer

Select “00.0_HelloWorld.py” in “MicroPython device”, right-click to select “Download to ...” to download the code to your computer.



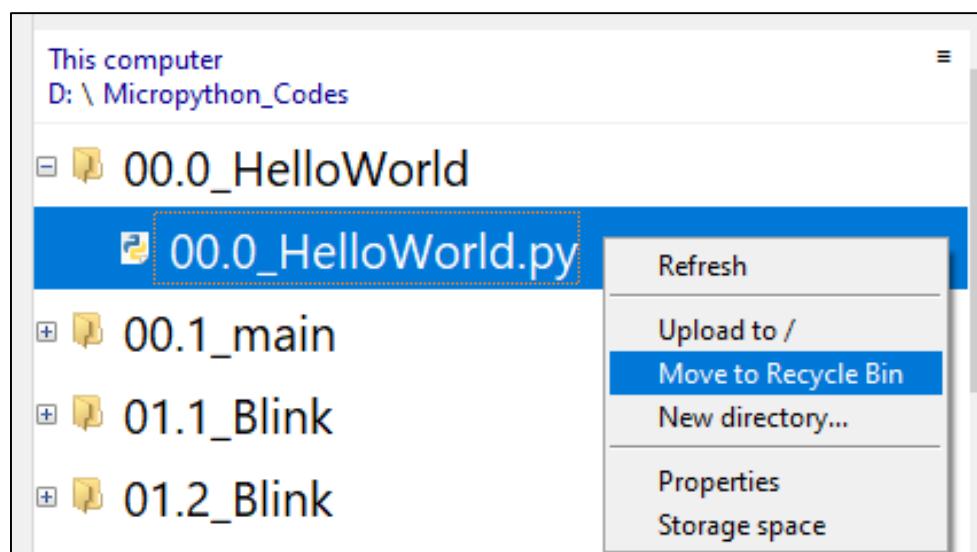
Deleting Files from Raspberry Pi Pico's Root Directory

Select “00.0_HelloWorld.py” in “MicroPython device”, right-click it and select “Delete” to delete “00.0_HelloWorld.py” from Raspberry Pi Pico’s root directory.



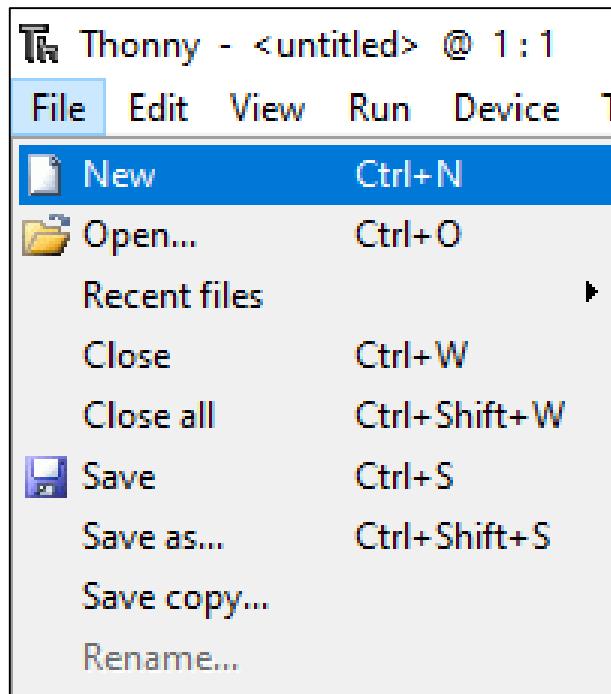
Deleting Files from your Computer Directory

Select “00.0_HelloWorld.py” in “00.0_HelloWorld”, right-click it and select “Move to Recycle Bin” to delete it from “00.0_HelloWorld”.



Creating and Saving the code

Click “File”→“New” to create and write codes.



Enter codes in the newly opened file. Here we use codes of “01.1_Blink.py” as an example.

```

from machine import Pin
import time

led = Pin(25, Pin.OUT) # create LED object from Pin 25, Set Pin

try:
    while True:
        led.value(1) # Set led turn on
        time.sleep(0.5) # Sleep 0.5s
        led.value(0) # Set led turn off
        time.sleep(0.5) # Sleep 0.5s
except:
    pass

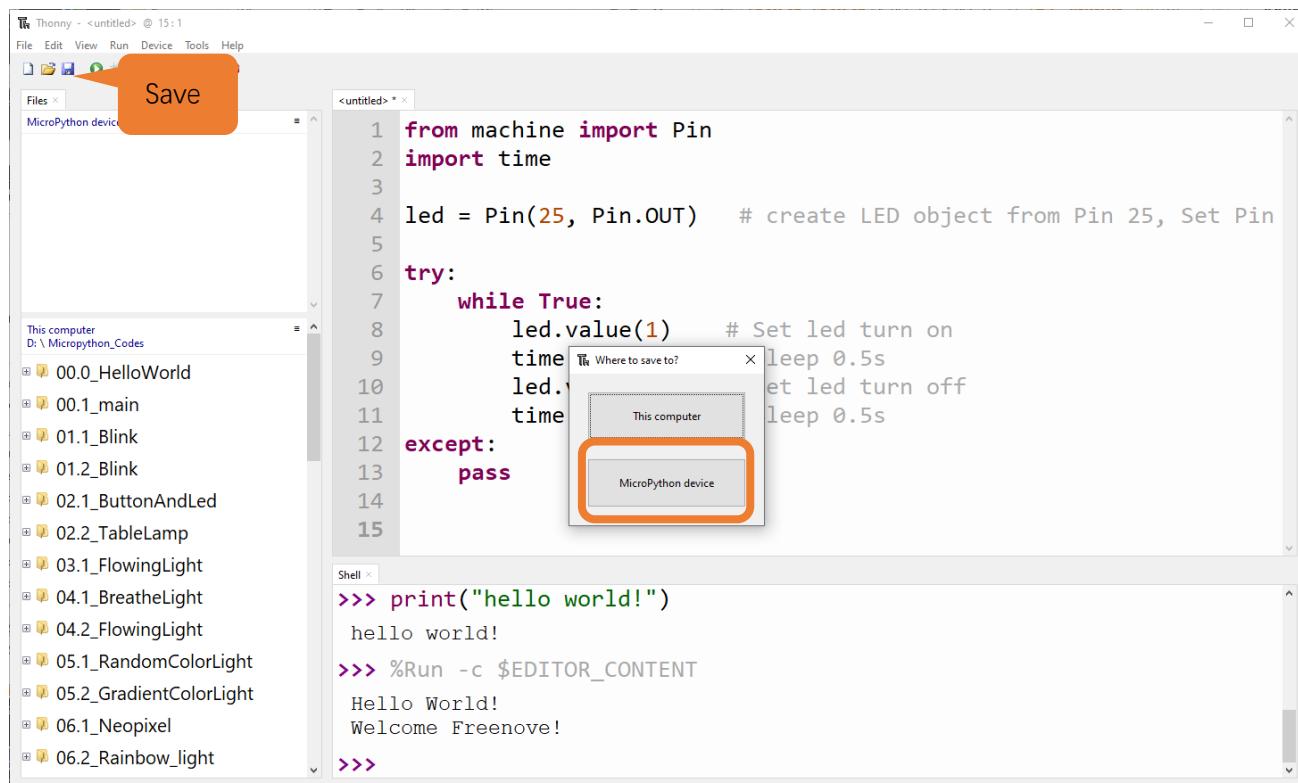
```

```

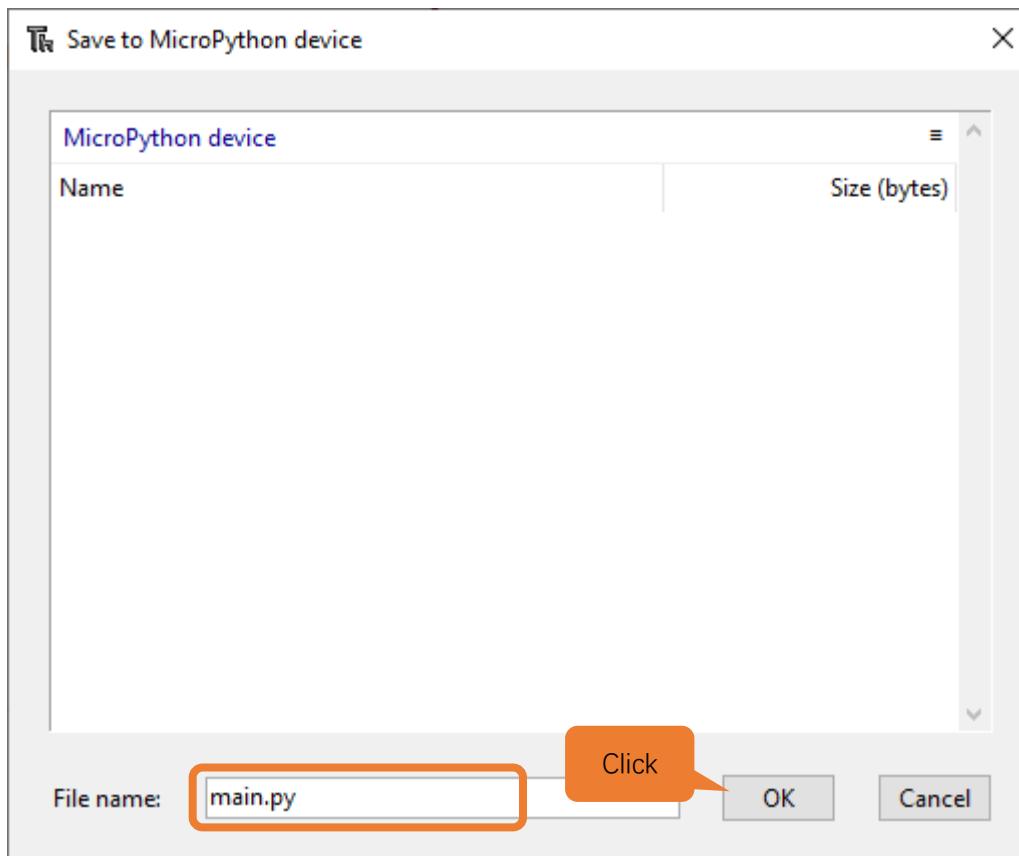
>>> print("hello world!")
hello world!
>>> %Run -c $EDITOR_CONTENT
Hello World!
Welcome Freenove!
>>>

```

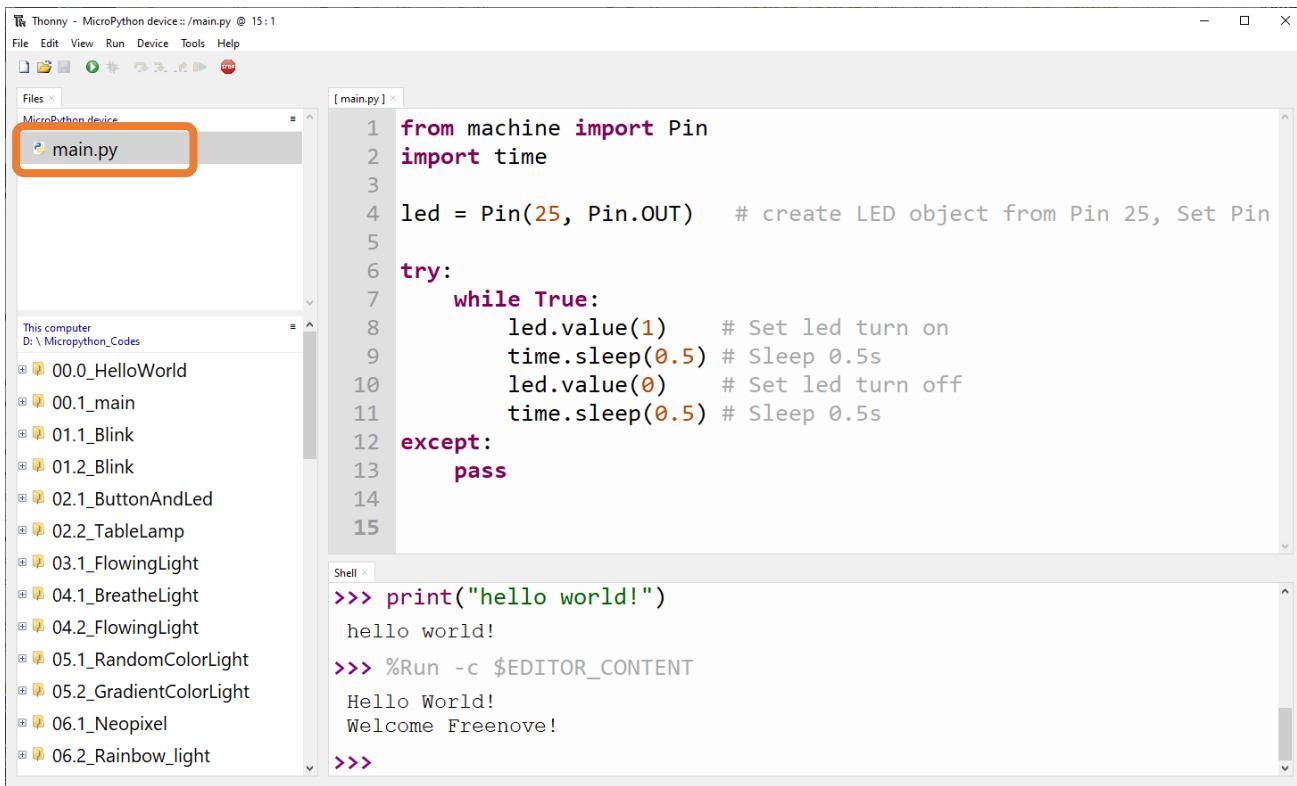
Click “Save” on the menu bar. You can save the codes either to your computer or to Raspberry Pi Pico.



Select “MicroPython device”, enter “main.py” in the newly pop-up window and click “OK”.



You can see that codes have been uploaded to Raspberry Pi Pico.



```

from machine import Pin
import time

led = Pin(25, Pin.OUT) # create LED object from Pin 25, Set Pin

try:
    while True:
        led.value(1) # Set led turn on
        time.sleep(0.5) # Sleep 0.5s
        led.value(0) # Set led turn off
        time.sleep(0.5) # Sleep 0.5s
except:
    pass

```

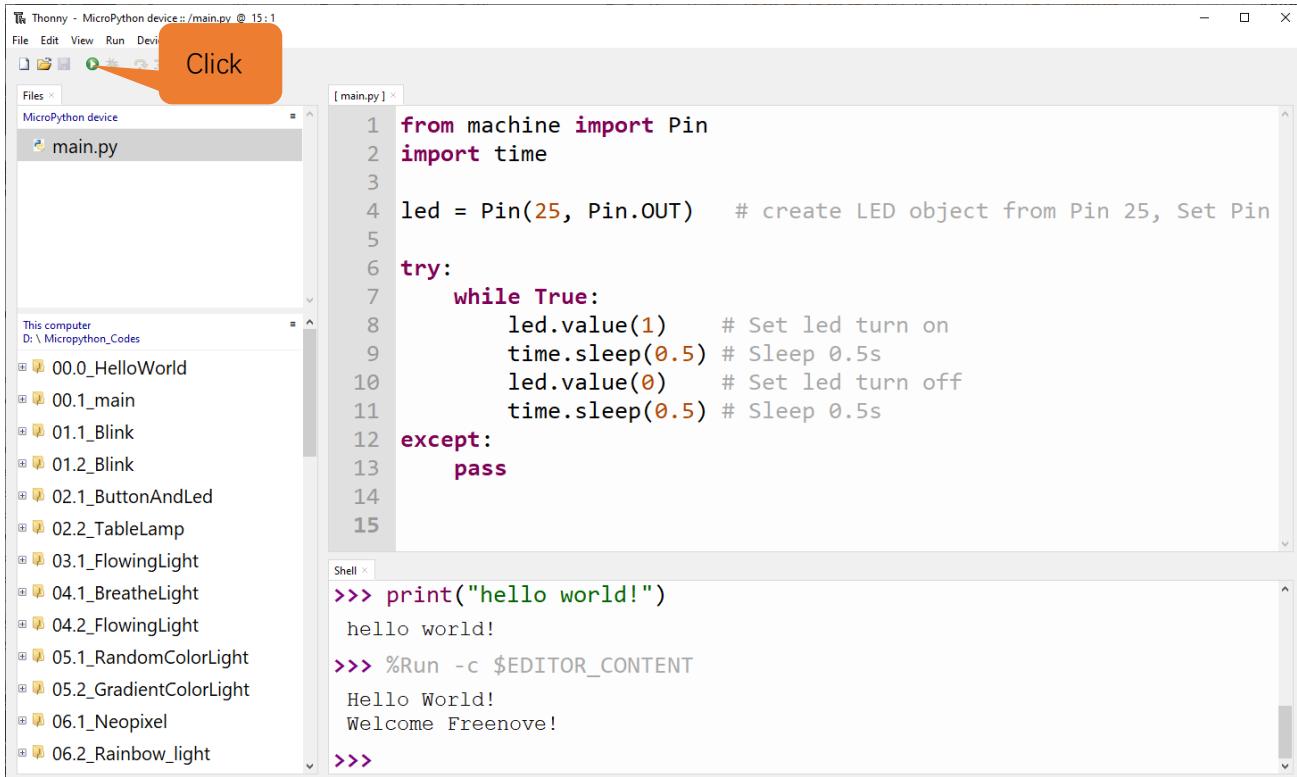
Shell

```

>>> print("hello world!")
hello world!
>>> %Run -c $EDITOR_CONTENT
Hello World!
Welcome Freenove!
>>>

```

Click "Run" and the LED on Raspberry Pi Pico will blink periodically.



Click

```

from machine import Pin
import time

led = Pin(25, Pin.OUT) # create LED object from Pin 25, Set Pin

try:
    while True:
        led.value(1) # Set led turn on
        time.sleep(0.5) # Sleep 0.5s
        led.value(0) # Set led turn off
        time.sleep(0.5) # Sleep 0.5s
except:
    pass

```

Shell

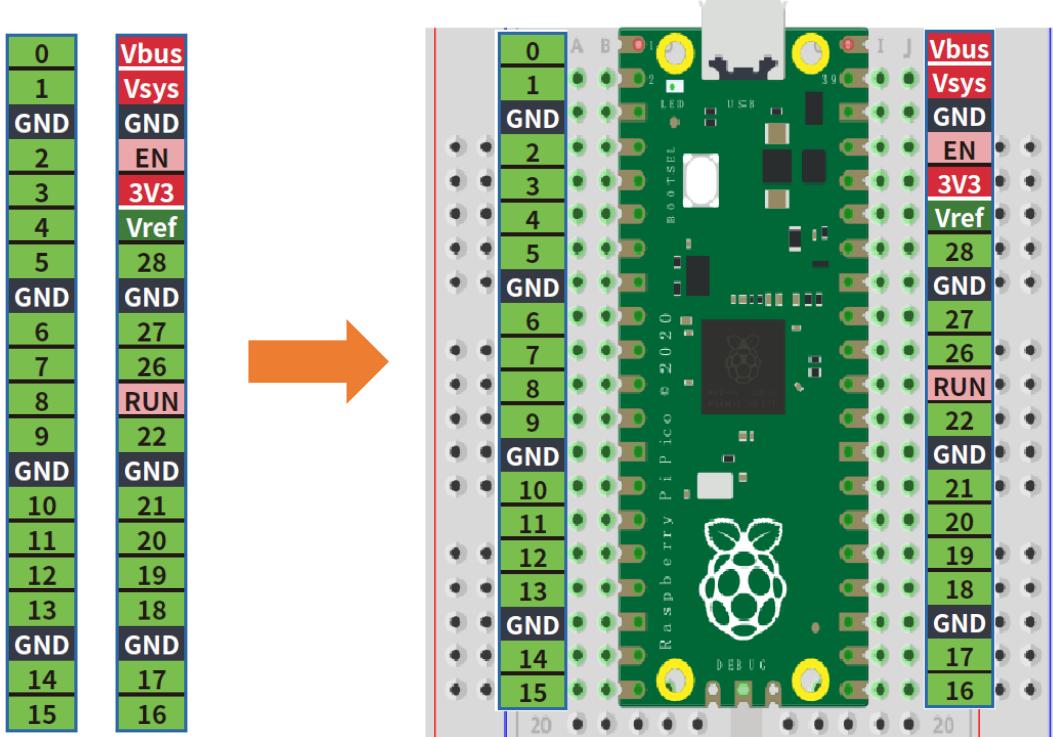
```

>>> print("hello world!")
hello world!
>>> %Run -c $EDITOR_CONTENT
Hello World!
Welcome Freenove!
>>>

```

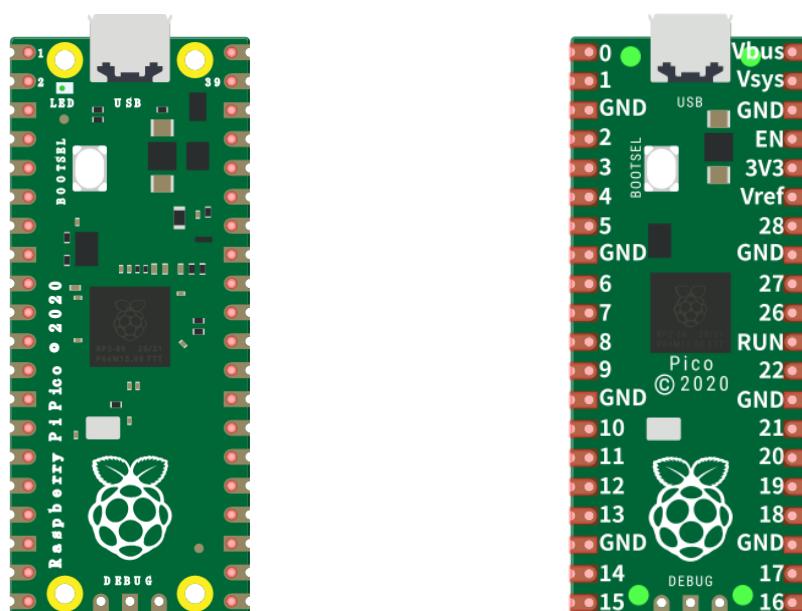
07. Paste the Sticker on the Breadboard

It is not difficult to use the Pico. However, officially, the pin functions are printed on the back of the board, which makes it inconvenient to use. To help users finish each project in the tutorial faster and easier, we provide stickers of the pin functions as follows:



You can paste the sticker on the blank area of the breadboard as above.

To make the tutorial more intuitive, we've made some changes to the simulation diagram as below. The left one is the actual Pico and the right one is its simulation diagram. Please note that to avoid misunderstanding.





Chapter 1 LED (Important)

This chapter is the Start Point in the journey to build and explore Raspberry Pi Pico electronic projects. We will start with simple “Blink” project.

Project 1.1 Blink

In this project, we will use Raspberry Pi Pico to control blinking a common LED.

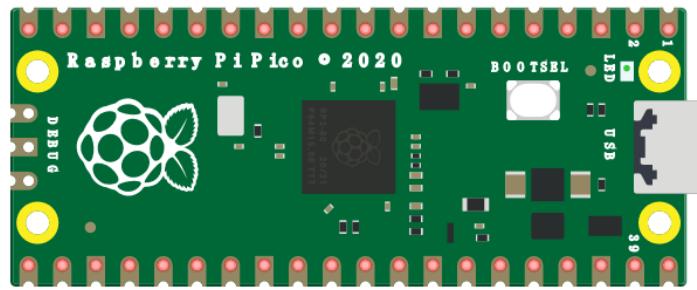
If you have not yet installed Thonny, click [here](#).

If you have not yet downloaded MicroPython Firmware, click [here](#).

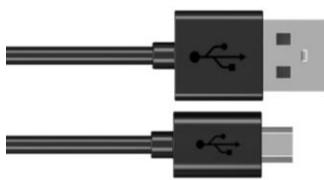
If you have not yet loaded MicroPython Firmware, click [here](#).

Component List

Raspberry Pi Pico x1

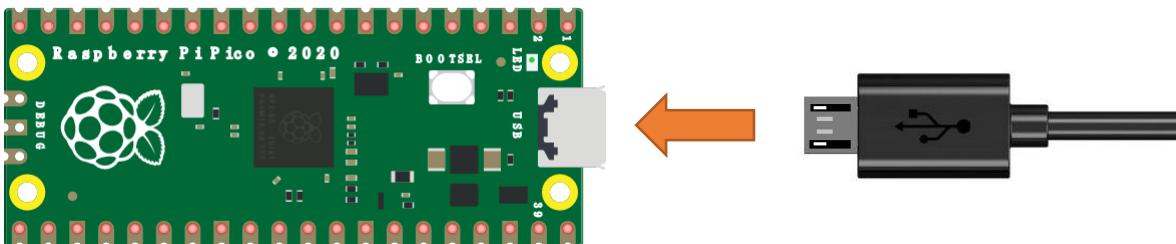


USB cable x1



Power

In this tutorial, we connect Raspberry Pi Pico and computer with a USB cable.



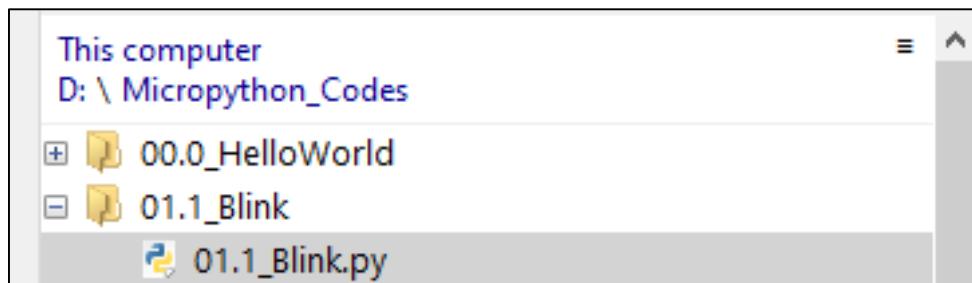
Code

Codes used in this tutorial are saved in

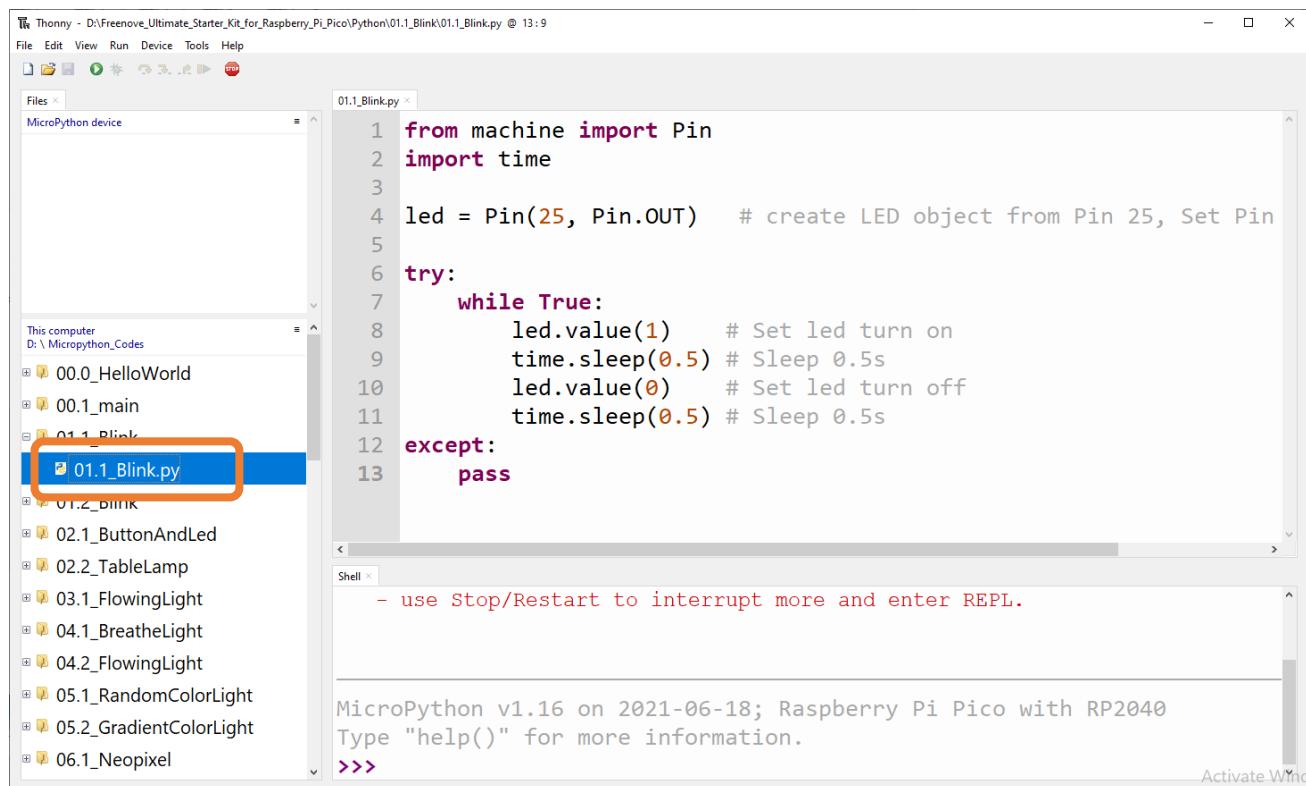
"Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico/Python_Codes". You can move the codes to any location. For example, we save the codes in Disk(D) with the path of **"D:/Micropython_Codes"**.

01.1_Blink

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes".



Expand folder "01.1_Blink" and double click "01.1_Blink.py" to open it. As shown in the illustration below.





Make sure Raspberry Pi Pico has been connected with the computer. Click “Stop/Restart backend”, and then wait to see what interface will show up.

```

from machine import Pin
import time

led = Pin(25, Pin.OUT) # create LED object from Pin 25, Set Pin

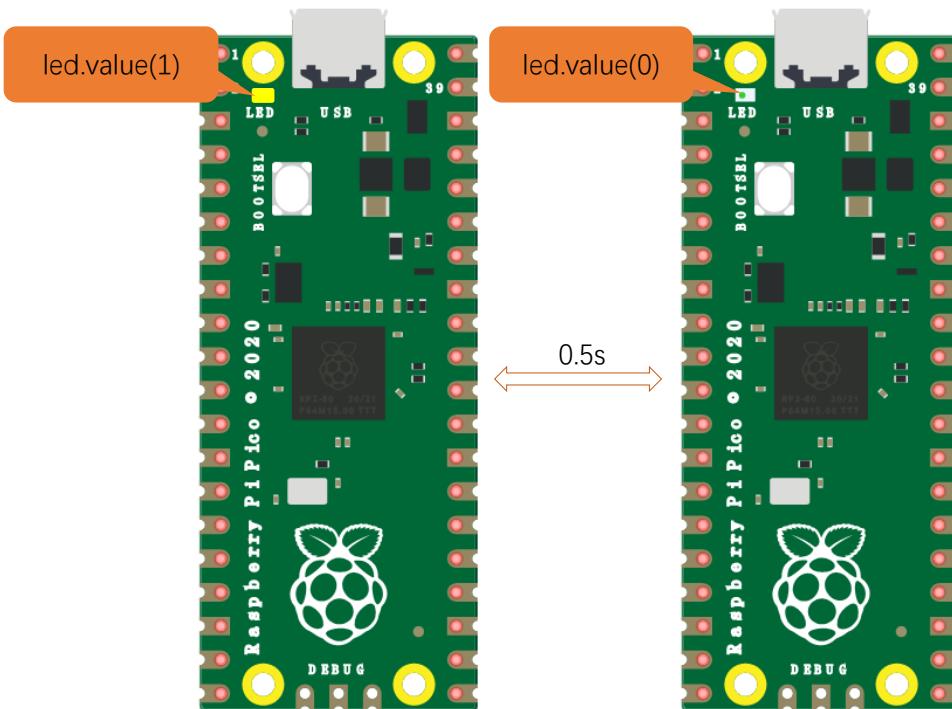
try:
    while True:
        led.value(1)      # Set led turn on
        time.sleep(0.5)   # Sleep 0.5s
        led.value(0)      # Set led turn off
        time.sleep(0.5)   # Sleep 0.5s
except:
    pass

```

Use Stop/Restart to reconnect.

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico
Type "help()" for more information.
>>>

Click “Run current script” shown in the box above, the code starts to be executed and the LED in the circuit starts to blink. Press Ctrl+C or click “Stop/Restart backend” to exit the program.



Note:

This is the code [running online](#). If you disconnect USB cable and repower Raspberry Pi Pico, LED stops blinking and the following messages will display in Thonny.

```
from machine import Pin
import time

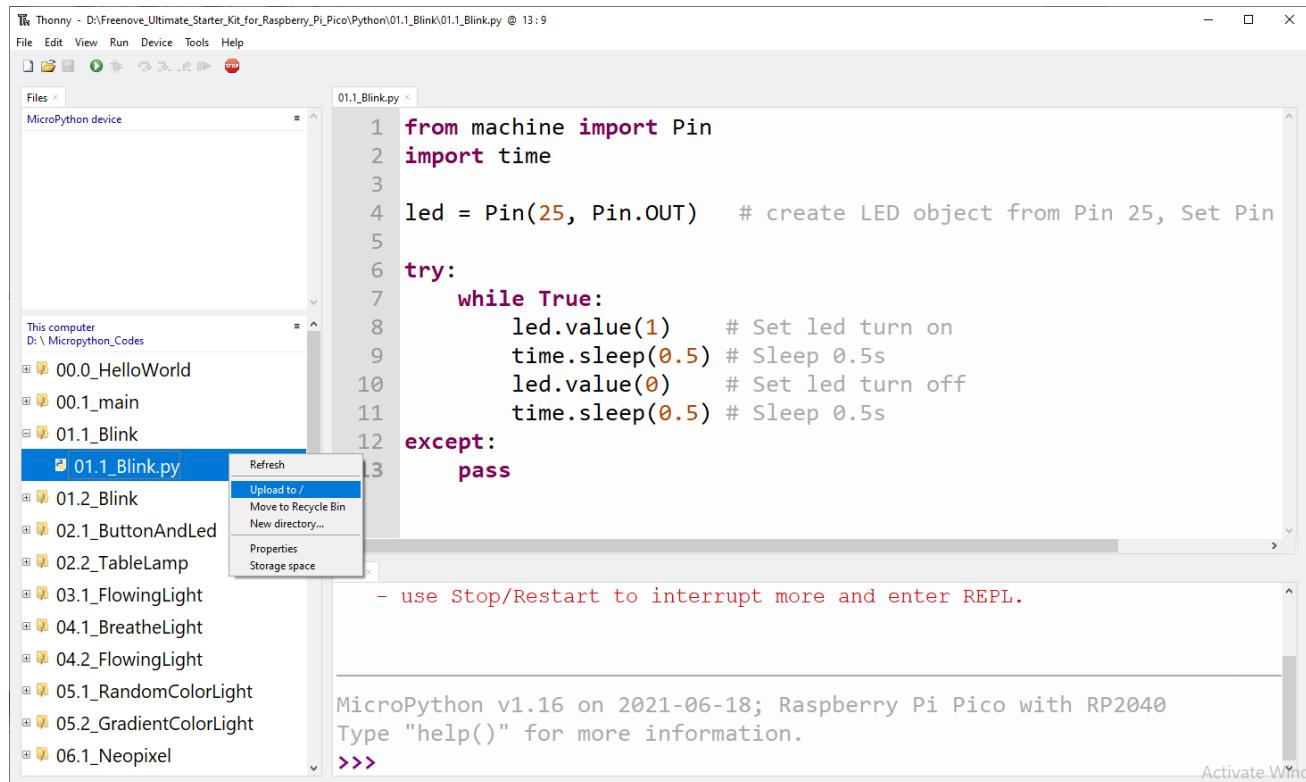
led = Pin(25, Pin.OUT) # create LED object from Pin 25, Set Pin

try:
    while True:
        led.value(1) # Set led turn on
        time.sleep(0.5) # Sleep 0.5s
        led.value(0) # Set led turn off
        time.sleep(0.5) # Sleep 0.5s
except:
    pass
```

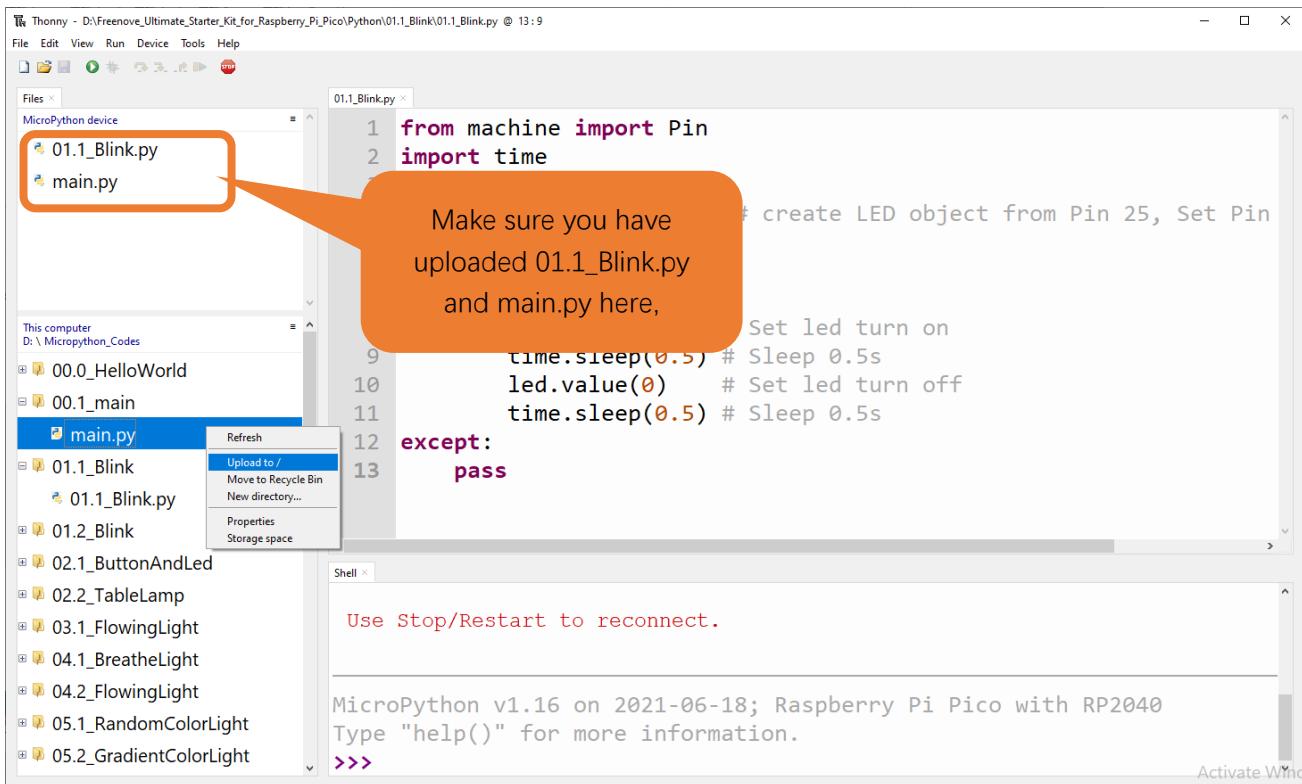
MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
->
Connection lost (EOF)
Use Stop/Restart to reconnect.

Uploading code to Raspberry Pi Pico

As shown in the following illustration, right-click the file 01.1_Blink.py and select “Upload to /” to upload code to Raspberry Pi Pico.



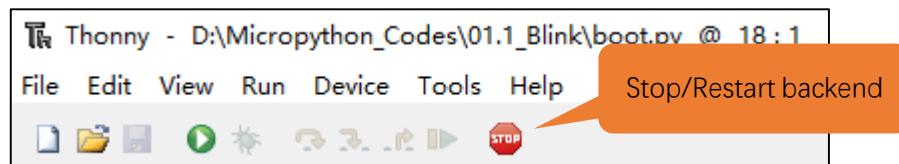
Upload main.py in the same way.



Disconnect Raspberry Pi Pico USB cable and reconnect it, LED on Pico will blink repeatedly.

Note:

Codes here is run offline. If you want to stop running offline and enter Shell, just click "Stop" in Thonny.



If you have any concerns, please contact us via: support@freenove.com

Any concerns? ✉ support@freenove.com

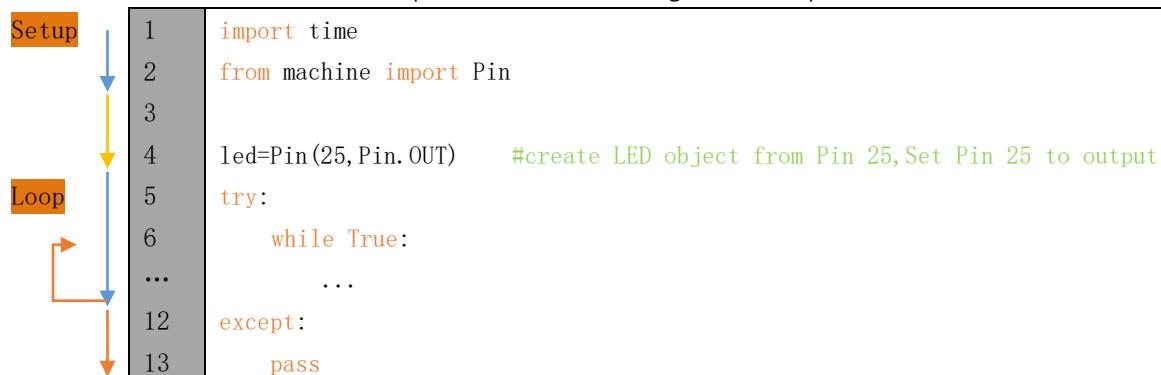
The following is the program code:

```

1 import time
2 from machine import Pin
3
4 led=Pin(25,Pin.OUT)      #create LED object from Pin 25, Set Pin 25 to output
5
6 try:
7     while True:
8         led.value(1)      #Set led turn on
9         time.sleep(0.5)   #Sleep 0.5s
10        led.value(0)     #Set led turn off
11        time.sleep(0.5)   #Sleep 0.5s
12 except:
13     pass

```

Each time a new file is opened, the program will be executed from top to bottom. When encountering a loop construction, it will execute the loop statement according to the loop condition.



`Print()` function is used to print data to Terminal. It can be executed in Terminal directly or be written in a Python file and executed by running the file.

```
print("Hello world!")
```

Each time when using the functions of Raspberry Pi Pico, you need to import modules corresponding to those functions: Import `time` module and `Pin` module of `machine` module.

```

1 import time
2 from machine import Pin

```

Configure GP25 of Raspberry Pi Pico to output mode and assign it to an object named "led".

```
4 led=Pin(25,Pin.OUT) #create LED object from Pin 25, Set Pin 25 to output
```

It means that from now on, LED representing GP25 is in output mode.

Set the value of LED to 1 and GP25 will output high level.

```
8 led.value(1) #Set led turn on
```

Set the value of LED to 0 and GP25 will output low level.

```
10 led.value(0) #Set led turn on
```

Execute codes in a while loop.

```

7 while True:
...

```



Put statements that may cause an error in “try” block and the executing statements when an error occurs in “except” block. In general, when the program executes statements, it will execute those in “try” block. However, when an error occurs to Raspberry Pi Pico due to some interference or other reasons, it will execute statements in “except” block.

“Pass” is an empty statement. When it is executed, nothing happens. It is useful as a placeholder to make the structure of a program look better.

```
6   try:  
...  
12  ...  
13  except:  
    pass
```

The single-line comment of Micropython starts with a “#” and continues to the end of the line. Comments help us to understand code. When programs are running, Thonny will skip comments.

```
8 #Set led turn on
```

MicroPython uses indentations to distinguish different blocks of code instead of braces. The number of indentations is changeable, but it must be consistent throughout one block. If the indentation of the same code block is inconsistent, it will cause errors when the program runs.

```
7 while True:  
8     led.value(1)      #Set led turn on  
9     time.sleep(0.5)  #Sleep 0.5s  
10    led.value(0)     #Set led turn off  
11    time.sleep(0.5)  #Sleep 0.5s
```

How to import python files

Whether to import the built-in python module or to import that written by users, the command “import” is needed.

If you import the module directly you should indicate the module to which the function or attribute belongs when using the function or attribute (constant, variable) in the module. The format should be: <module name>.<function or attribute>, otherwise an error will occur.

```
import random  
  
num = random.randint(1, 100)  
print(num)
```

If you only want to import a certain function or attribute in the module, use the from...import statement. The format is as follows

```
from random import randint  
num = randint(1, 100)  
print(num)
```

When using “from...import” statement to import function, to avoid conflicts and for easy understanding, you can use “as” statement to rename the imported function, as follows

```
from random import randint as rand  
num = rand(1, 100)  
print(num)
```

Reference

Class machine

Before each use of the **machine** module, please add the statement “**import machine**” to the top of python file.

machine.freq(freq_val): When “freq_val” is not specified, it is to return to the current CPU frequency; Otherwise, it is to set the current CPU frequency.

freq_val: 125000000Hz(125MHz).

machine.reset(): A reset function. When it is called, the program will be reset.

machine.unique_id(): Obtains MAC address of the device.

machine.idle(): Turns off any temporarily unused functions on the chip and its clock, which is useful to reduce power consumption at any time during short or long periods.

machine.disable_irq(): Disables interrupt requests and return the previous IRQ state. The disable_irq () function and enable_irq () function need to be used together; Otherwise the machine will crash and restart.

machine.enable_irq(state): To re-enable interrupt requests. The parameter **state** should be the value that was returned from the most recent call to the disable_irq() function.

machine.time_pulse_us(pin, pulse_level, timeout_us=1000000):

Tests the duration of the external pulse level on the given pin and returns the duration of the external pulse level in microseconds. When pulse level = 1, it tests the high level duration; When pulse level = 0, it tests the low level duration.

If the setting level is not consistent with the current pulse level, it will wait until they are consistent, and then start timing. If the set level is consistent with the current pulse level, it will start timing immediately.

When the pin level is opposite to the set level, it will wait for timeout and return “-2”. When the pin level and the set level is the same, it will also wait timeout but return “-1”. **timeout_us** is the duration of timeout.

For more information about class and function, please refer to:

<https://docs.micropython.org/en/latest/rp2/quickref.html>

Class time

Before each use of the **time** module, please add the statement “**import time**” to the top of python file

time.sleep(sec): Sleeps for the given number of seconds.

sec: This argument should be either an int or a float.

time.sleep_ms(ms): Sleeps for the given number of milliseconds, ms should be an int.

time.sleep_us(us): Sleeps for the given number of microseconds, us should be an int.

time.time(): Obtains the timestamp of CPU, with second as its unit.

time.ticks_ms(): Returns the incrementing millisecond counter value, which recounts after some values.

time.ticks_us(): Returns microsecond.

time.ticks_cpu(): Similar to ticks_ms() and ticks_us(), but it is more accurate(return clock of CPU).

time.ticks_add(ticks, delta): Gets the timestamp after the offset.

ticks: ticks_ms()、ticks_us()、ticks_cpu().

delta: Delta can be an arbitrary integer number or numeric expression.

time.ticks_diff(old_t, new_t): Calculates the interval between two timestamps, such as ticks_ms(), ticks_us() or ticks_cpu().

old_t: Starting time.

new_t: Ending time.

Class Pin(id, mode, pull, value)

Before each use of the **Pin** module, please add the statement “**from machine import Pin**” to the top of python file.

id: Arbitrary pin number.

mode: Mode of pins.

Pin.IN: Input Mode.

Pin.OUT: Output Mode.

Pin.OPEN_DRAIN: Open-drain Mode.

Pull: Whether to enable the internal pull up and down mode.

None: No pull up or pull down resistors.

Pin.PULL_UP: Pull-up Mode, outputting high level by default.

Pin.PULL_DOWN: Pull-down Mode, outputting low level by default.

Value: State of the pin level, 0/1.

Pin.init(mode, pull): Initialize pins.

Pin.value([value]): Obtain or set state of the pin level, return 0 or 1 according to the logic level of pins.

Without parameter, it reads input level. With parameter given, it is to set output level.

value: It can be either True/False or 1/0.

Pin.irq(trigger, handler): Configures an interrupt handler to be called when the pin level meets a condition.

trigger:

Pin.IRQ_FALLING: interrupt on falling edge.

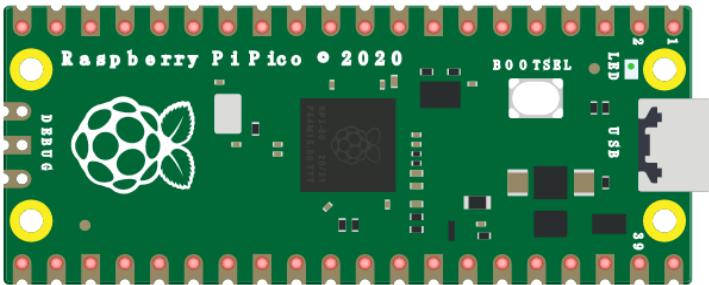
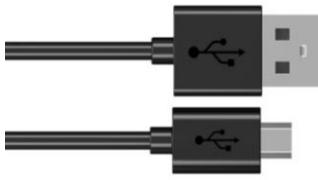
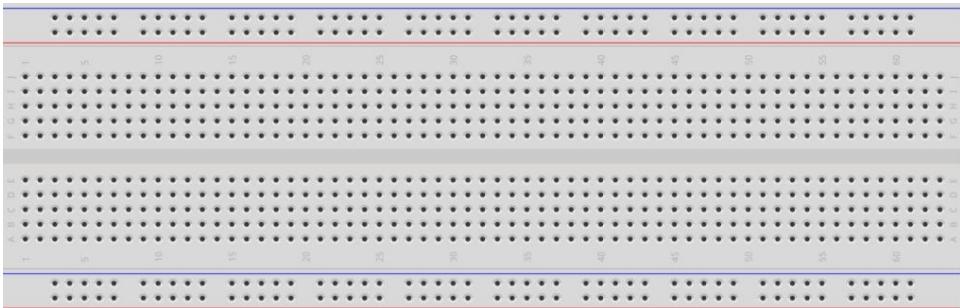
Pin.IRQ_RISING: interrupt on rising edge.

Handler: callback function.

Project 1.2 Blink

In this project, we will use Raspberry Pi Pico to control blinking a common LED.

Component List

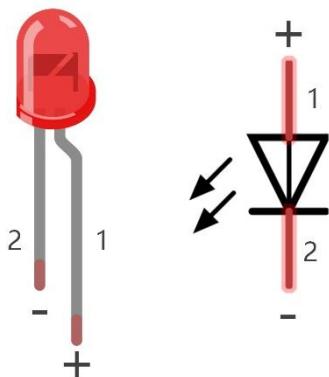
Raspberry Pi Pico x1	USB Cable x1	
		
Breadboard x1		
		
LED x1	Resistor 220Ω x1	Jumper
		

Component knowledge

LED

An LED is a type of diode. All diodes only work if current is flowing in the correct direction and have two Poles. An LED will only work (light up) if the longer pin (+) of LED is connected to the positive output from a power source and the shorter pin is connected to the negative (-). Negative output is also referred to as Ground (GND). This type of component is known as “Polar” (think One-Way Street).

All common 2 lead diodes are the same in this respect. Diodes work only if the voltage of its positive electrode is higher than its negative electrode and there is a narrow range of operating voltage for most all common diodes of 1.9 and 3.4V. If you use much more than 3.3V the LED will be damaged and burn out.



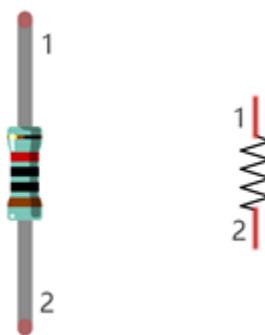
LED	Voltage	Maximum current	Recommended current
Red	1.9 - 2.2V	20mA	10mA
Green	2.9 - 3.4V	10mA	5mA
Blue	2.9 - 3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

Note: LEDs cannot be directly connected to a power supply, which usually ends in a damaged component. A resistor with a specified resistance value must be connected in series to the LED you plan to use.

Resistor

Resistors use Ohms (Ω) as the unit of measurement of their resistance (R). $1M\Omega=1000k\Omega$, $1k\Omega=1000\Omega$.

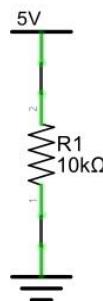
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



The bands of color on a resistor is a shorthand code used to identify its resistance value. For more details of resistor color codes, please refer to the appendix of this tutorial.

With a fixed voltage, there will be less current output with greater resistance added to the circuit. The relationship between Current, Voltage and Resistance can be expressed by this formula: $I=V/R$ known as Ohm's Law where I = Current, V = Voltage and R = Resistance. Knowing the values of any two of these allows you to solve the value of the third.

In the following diagram, the current through R1 is: $I=U/R=5V/10k\Omega=0.0005A=0.5mA$.



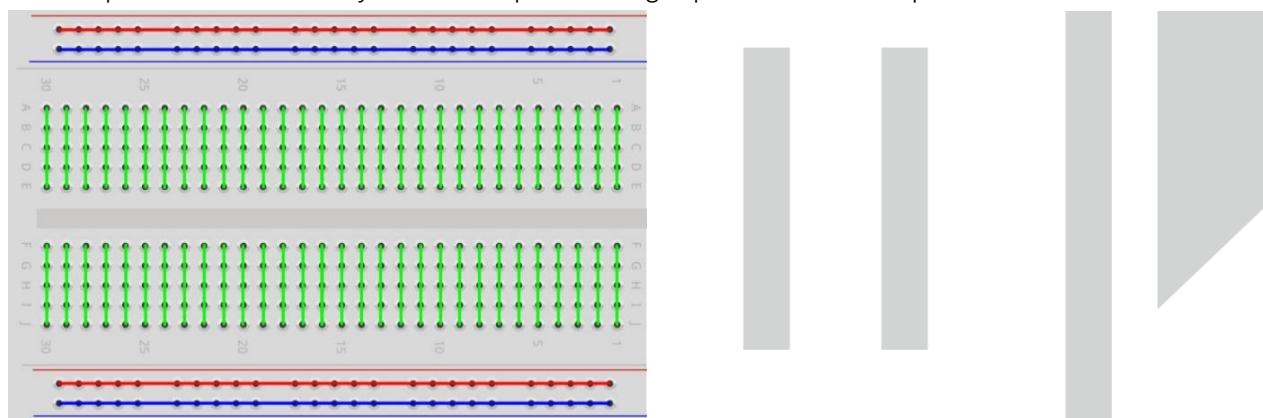
WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

Note: Unlike LEDs and Diodes, Resistors have no poles and are non-polar (it does not matter which direction you insert them into a circuit, it will work the same)

Breadboard

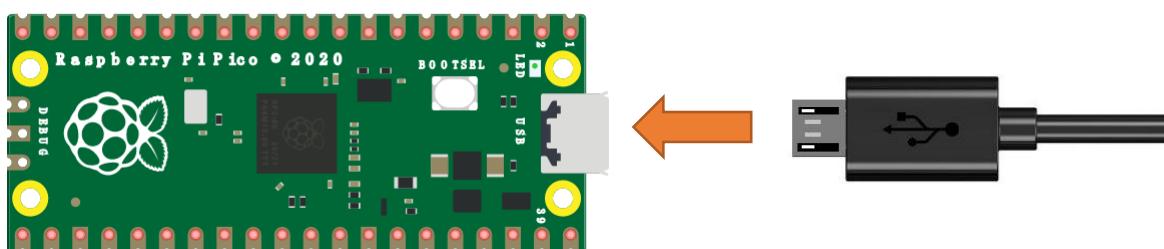
Here we have a small breadboard as an example of how the rows of holes (sockets) are electrically attached.

The left picture shows the way to connect pins. The right picture shows the practical internal structure.



Power

In this tutorial, we connect Raspberry Pi Pico and computer with a USB cable.

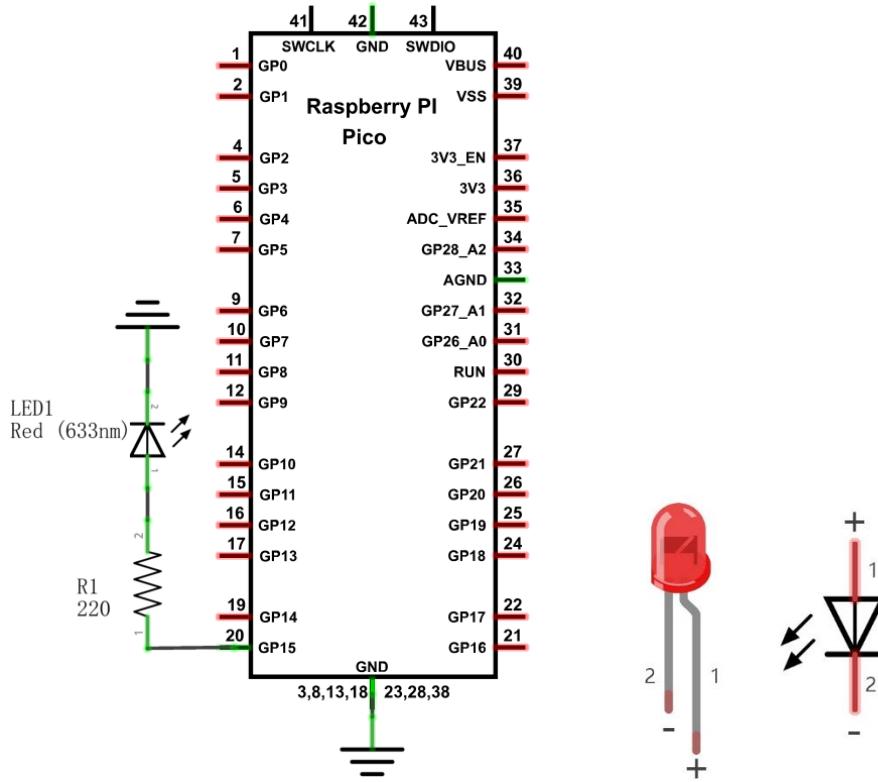


Circuit

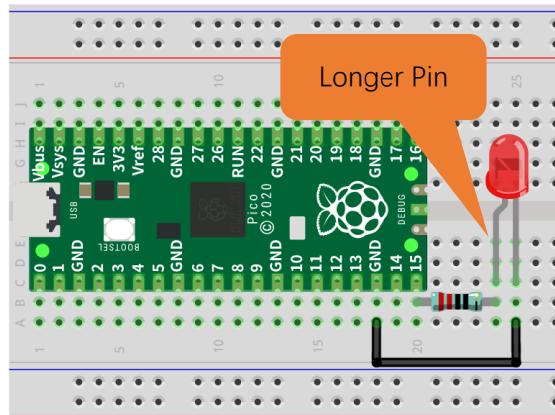
First, disconnect all power from the Raspberry Pi Pico. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the PC to Raspberry Pi Pico.

CAUTION: Avoid any possible short circuits (especially connecting 3.3V and GND)! **WARNING:** A short circuit can cause high current in your circuit, create excessive component heat and cause permanent damage to your hardware!

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

Any concerns? ✉ support@freenove.com

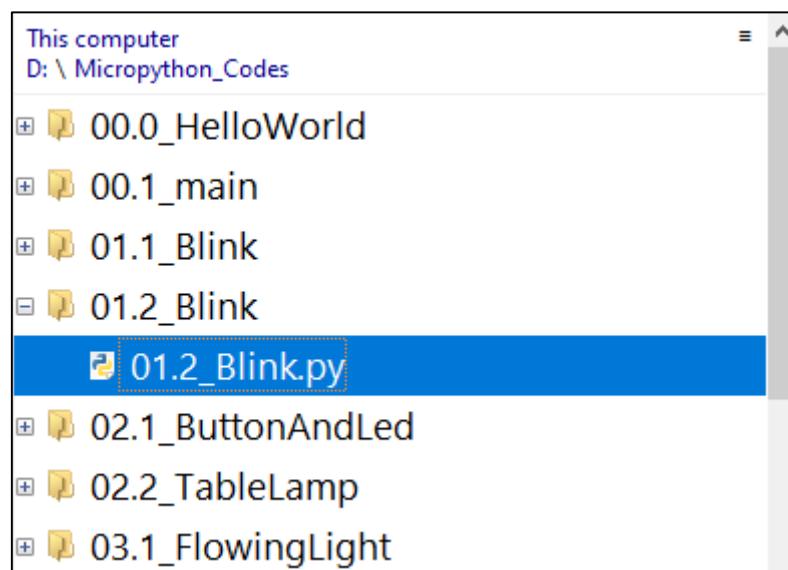
Code

Codes used in this tutorial are saved in

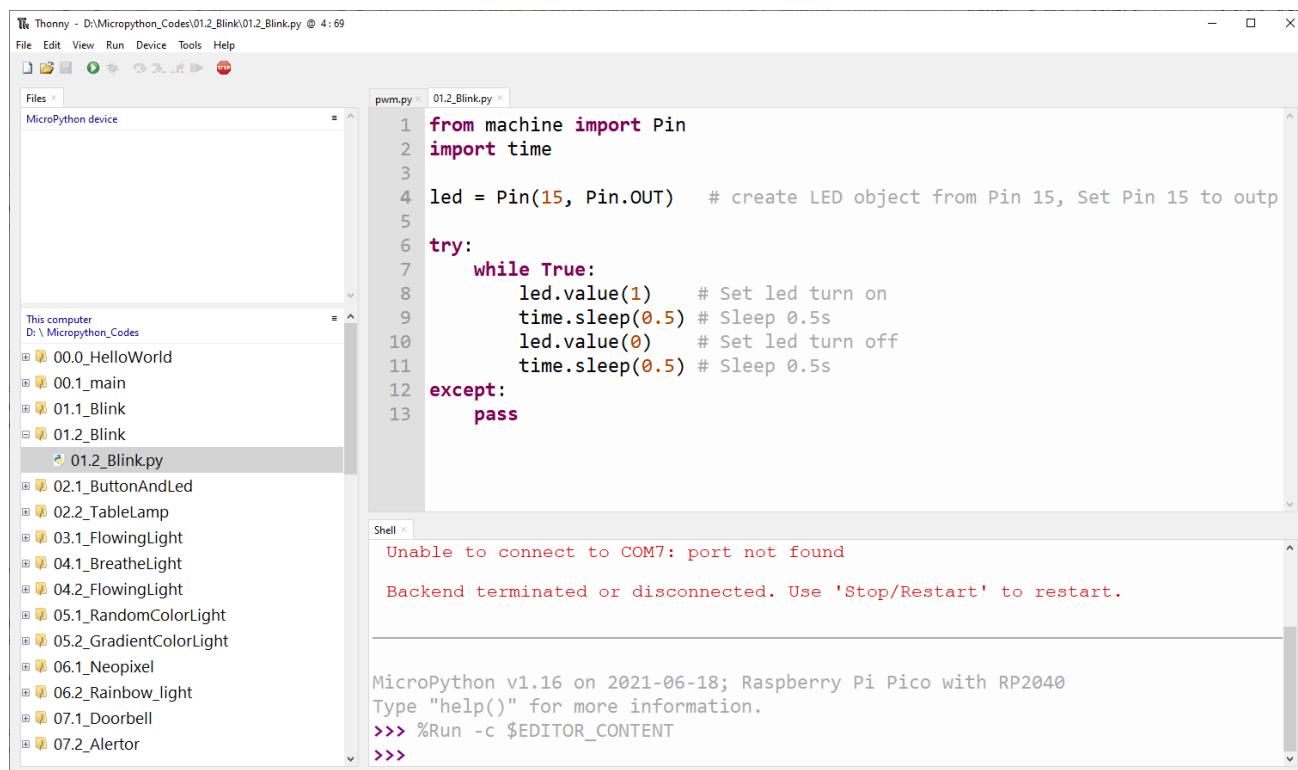
"Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico/Python_Codes". You can move the codes to any location. For example, we save the codes in Disk(D) with the path of "**D:/Micropython_Codes**".

01.2_Blink

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes".



Expand folder "01.2_Blink" and double click "01.2_Blink.py" to open it. As shown in the illustration below.





Make sure Raspberry Pi Pico has been connected with the computer. Click “Stop/Restart backend”, and then wait to see what interface will show up.

1, Stop/Restart backend

2, Run current script

This indicates that the connection is successful.

```

from machine import Pin
import time

led = Pin(15, Pin.OUT) # create LED object from Pin 15, Set Pin 15 to output

try:
    while True:
        led.value(1) # Set led turn on
        time.sleep(0.5) # Sleep 0.5s
        led.value(0) # Set led turn off
        time.sleep(0.5) # Sleep 0.5s
except:
    pass

```

Shell x
Unable to connect to COM7: port not found
Backend terminated or disconnected. Use 'Stop/Restart' to restart

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c \$EDITOR_CONTENT
>>>

Click “Run current script” shown in the box above, the code starts to be executed and the LED in the circuit starts to blink. Press Ctrl+C or click “Stop/Restart backend” to exit the program.



Note:

This is the code [running online](#). If you disconnect USB cable and repower Raspberry Pi Pico, LED stops blinking and the following messages will be displayed in Thonny.

The screenshot shows the Thonny IDE interface. On the left, the file tree displays various Python scripts under 'D:\Micropython_Codes'. The script '01.2_Blink.py' is selected and highlighted in blue. The main window shows the code content:

```

from machine import Pin
import time

led = Pin(15, Pin.OUT) # create LED object from Pin 15, Set Pin 15 to output
try:
    while True:
        led.value(1) # Set led turn on
        time.sleep(0.5)
        led.value(0) # Set led turn off
        time.sleep(0.5)
except:
    pass

```

Below the code, the 'Shell' tab shows the MicroPython environment and the error message:

```

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
>>>
Connection lost (EOF)

Use Stop/Restart to reconnect.

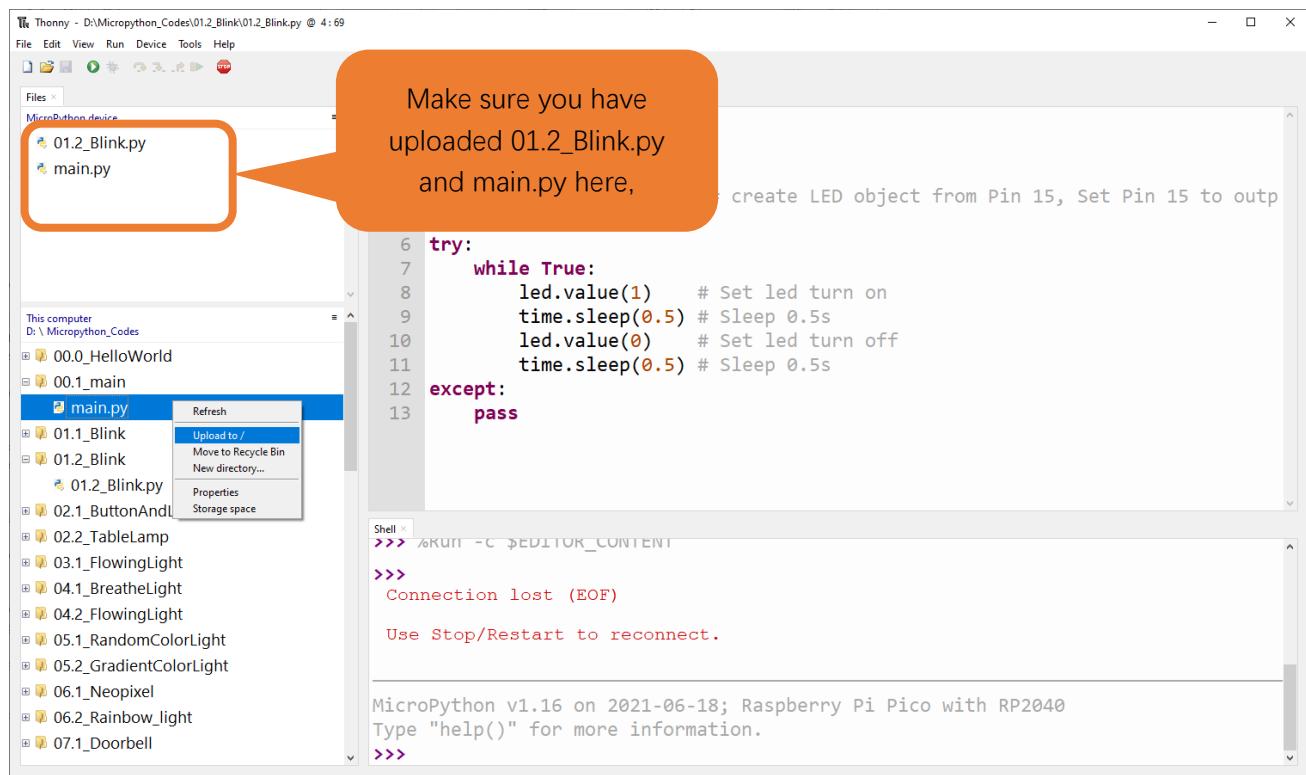
```

Uploading code to Raspberry Pi Pico

As shown in the following illustration, right-click the file 01.2_Blink.py and select “Upload to /” to upload code to Raspberry Pi Pico.

The screenshot shows the Thonny IDE interface with the same setup as the previous one. The '01.2_Blink.py' file is selected. A context menu is open over the file, with the 'Upload to /' option highlighted in blue. The menu also includes other options like 'Refresh', 'Move to Recycle Bin', 'New directory...', 'Properties', and 'Storage space'. The main window shows the code and the shell output, which includes the connection loss message and the MicroPython environment details.

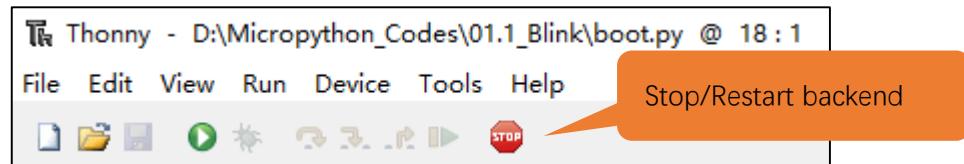
Upload main.py in the same way.



Disconnect Raspberry Pi Pico USB cable and reconnect it, LED on Pico will blink repeatedly.

Note:

Codes here is run offline. If you want to stop running offline and enter Shell, just click "Stop" in Thonny.

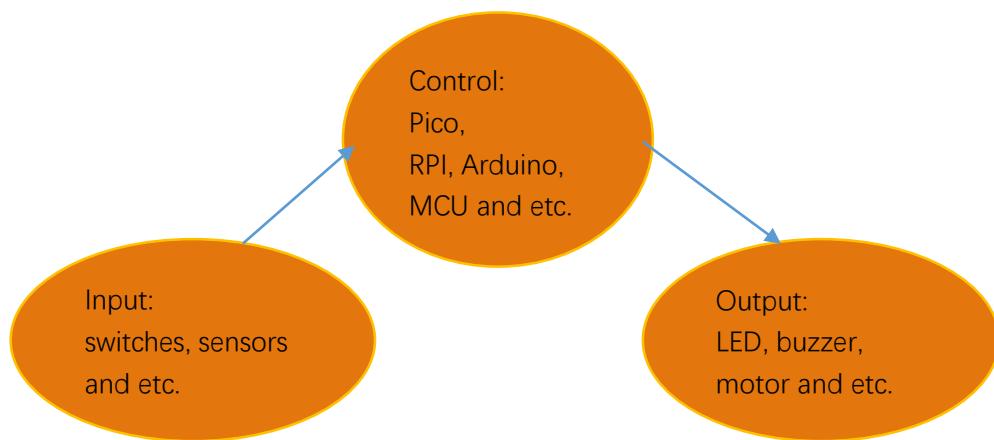


If you have any concerns, please contact us via: support@freenove.com

Any concerns? ✉ support@freenove.com

Chapter 2 Button & LED

Usually, there are three essential parts in a complete automatic control device: INPUT, OUTPUT, and CONTROL. In last section, the LED module was the output part and Raspberry Pi Pico was the control part. In practical applications, we not only make LEDs blink, but also make a device sense the surrounding environment, receive instructions and then take the appropriate action such as LEDs light up, make a buzzer turn ON and so on.



Next we make a simple project: build a control system with button, LED and Raspberry Pi Pico.

Input: Button

Control: Raspberry Pi Pico

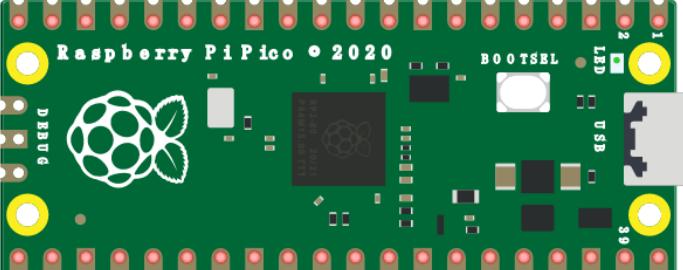
Output: LED



Project 2.1 Button & LED

In the project, we will control the LED state through a Push Button Switch. When the button is pressed, our LED will turn ON, and when it is released, the LED will turn OFF. This describes a Momentary Switch.

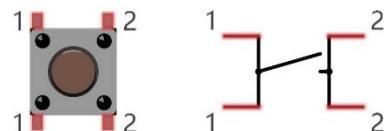
Component List

Raspberry Pi Pico x1		USB cable x1
Breadboard x1		
Jumper	LED x1	Resistor 220Ω x1
		Resistor 10kΩ x2
		Push button x1

Component knowledge

Push button

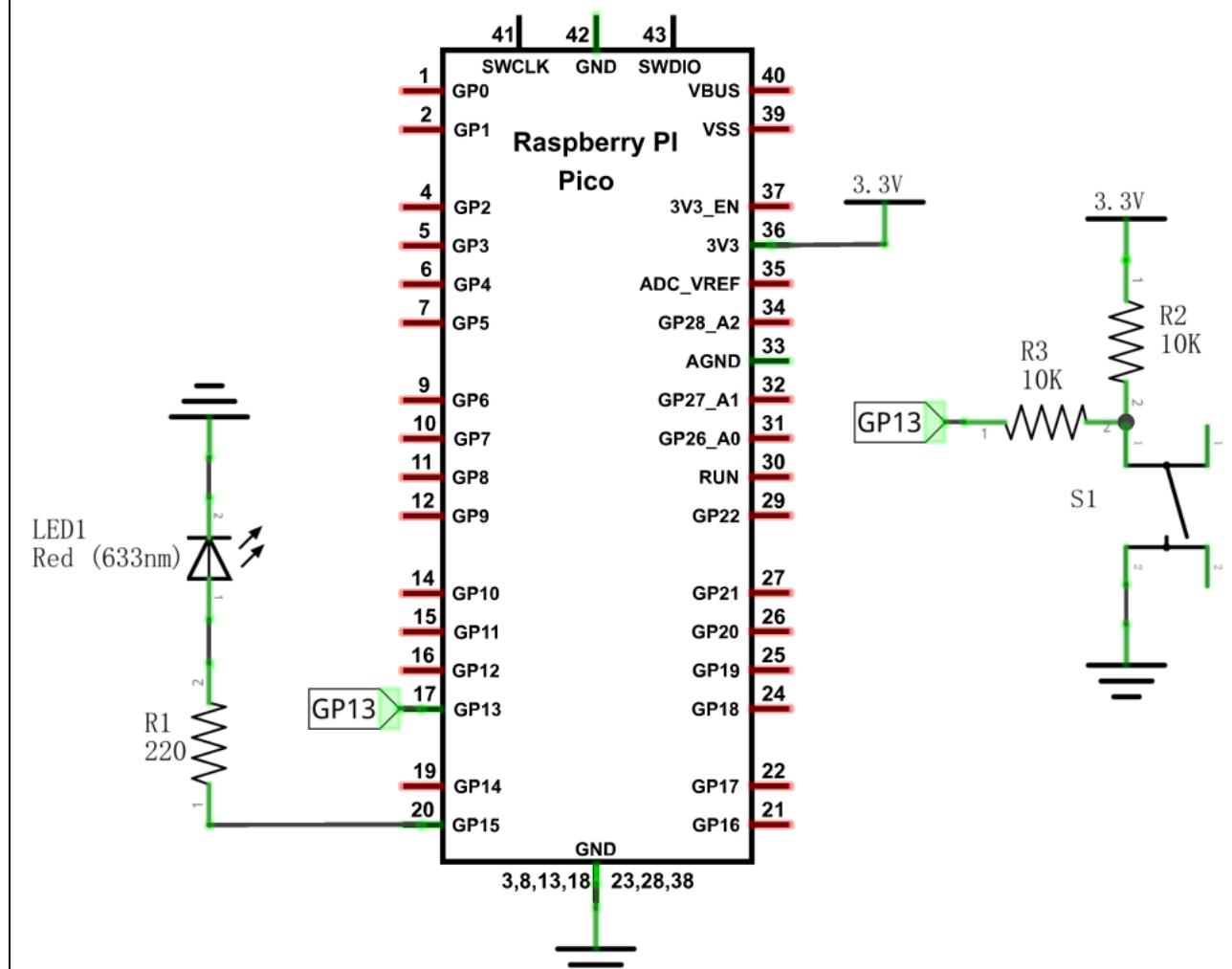
This type of Push Button Switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



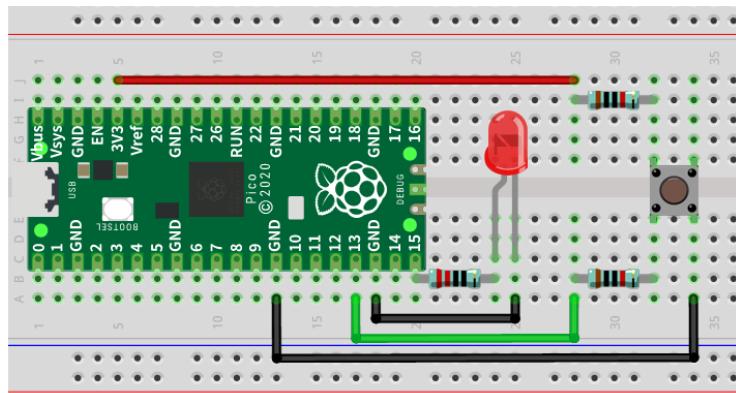
When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

Circuit

Schematic diagram



Hardware connection. If you need any support, please free to contact us via: support@freenove.com



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

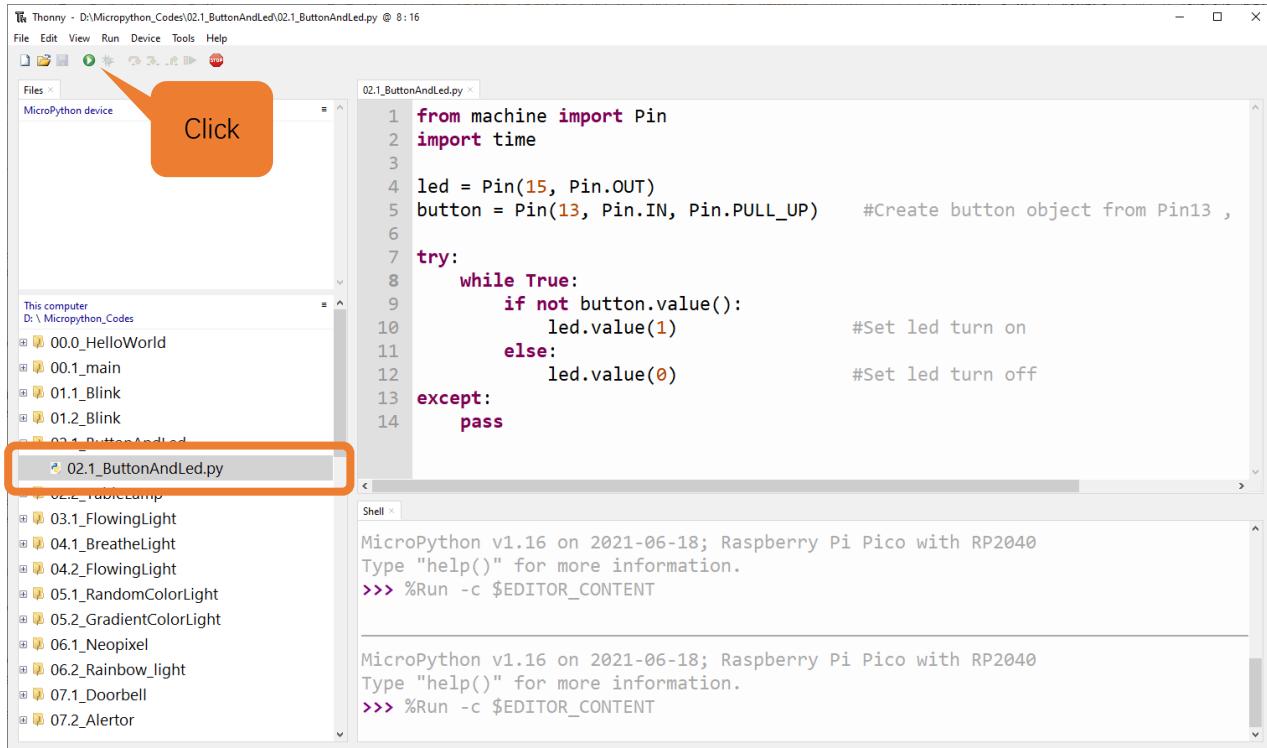
Code

This project is designed to learn to control an LED with a push button switch. First, we need to read the state of the switch and then decide whether the LED is turned on or not based on it.

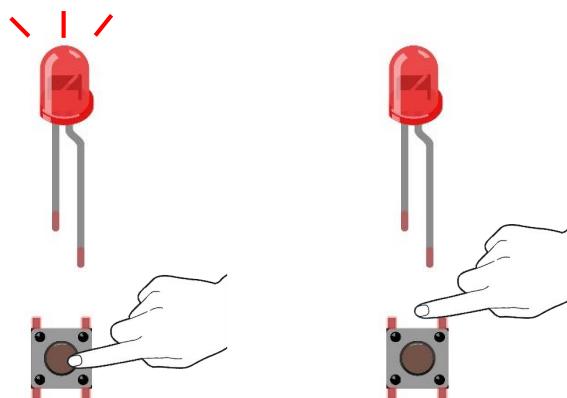
Move the program folder “Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico/Python_Codes” to disk(D) in advance with the path of “D:/Micropython_Codes”.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “02.1_ButtonAndLed” and double click “02.1_ButtonAndLed.py”.

02.1_ButtonAndLed



Click “Run current script” shown in the box of the above illustration, press the push button switch, LED turns ON; release the switch, LED turns OFF. Press Ctrl+C or click “Stop/Restart backend” to exit program.



The following is the program code:

```

1  from machine import Pin
2  import time
3
4  led = Pin(15, Pin.OUT)
5  button = Pin(13, Pin.IN, Pin.PULL_UP)      #Create button object from Pin13 , Pin13 to input
6
7  try:
8      while True:
9          if not button.value():
10             led.value(1)                  #Set led turn on
11         else:
12             led.value(0)                  #Set led turn off
13     except:
14         pass

```

In this project, we use the Pin module of the machine, so before initializing the Pin, we need to import this module first.

```
1  from machine import Pin
```

In the circuit connection, LED and Button are connected with GP15 and GP13 respectively, so define LED and button as 0 and 3 respectively.

```

4  led = Pin(15, Pin.OUT)
5  button = Pin(13, Pin.IN, Pin.PULL_UP)      #Create button object from Pin13, Set Pin13 to input

```

Read the pin state of button with value() function. Press the button switch, the function returns low level and the result of “if” is true, and then LED will be turned ON; Otherwise, LED is turned OFF.

```

9      if not button.value():
10         led.value(1)                  #Set led turn on
11     else:
12         led.value(0)                  #Set led turn off

```

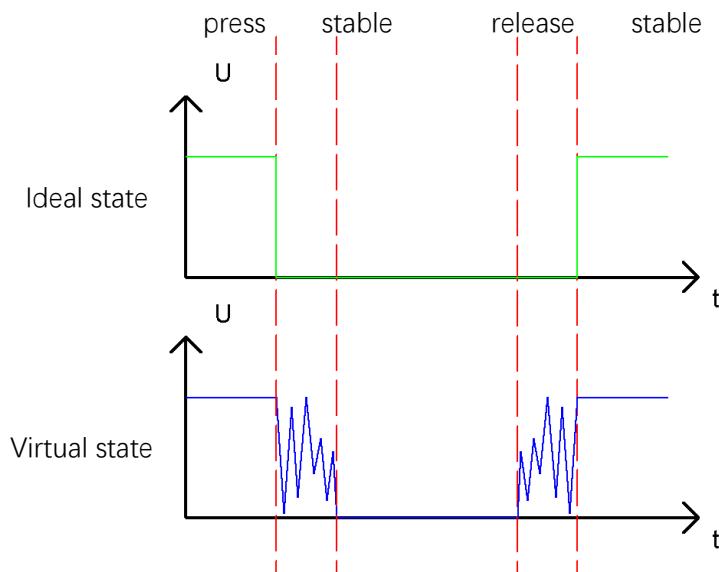
Project 2.2 MINI table lamp

We will also use a Push Button Switch, LED and Raspberry Pi Pico to make a MINI Table Lamp but this will function differently: Press the button, the LED will turn ON, and pressing the button again, the LED turns OFF. The ON switch action is no longer momentary (like a door bell) but remains ON without needing to continually press on the Button Switch.

First, let us learn something about the push button switch.

Debounce for Push Button

When a Momentary Push Button Switch is pressed, it will not change from one state to another state immediately. Due to tiny mechanical vibrations, there will be a short period of continuous buffeting before it completely reaches another state too fast for Humans to detect but not for computer microcontrollers. The same is true when the push button switch is released. This unwanted phenomenon is known as “bounce”.



Therefore, if we can directly detect the state of the Push Button Switch, there are multiple pressing and releasing actions in one pressing cycle. This buffeting will mislead the high-speed operation of the microcontroller to cause many false decisions. Therefore, we need to eliminate the impact of buffeting. Our solution: to judge the state of the button multiple times. Only when the button state is stable (consistent) over a period of time, can it indicate that the button is actually in the ON state (being pressed).

This project needs the same components and circuits as we used in the previous section.

Code

02.2_Tablelamp

Move the program folder “Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico/Python_Codes” to disk(D) in advance with the path of “D:/Micropython_Codes”.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “02.2_TableLamp” and double click “02.2_TableLamp.py”.

```

from machine import Pin
import time

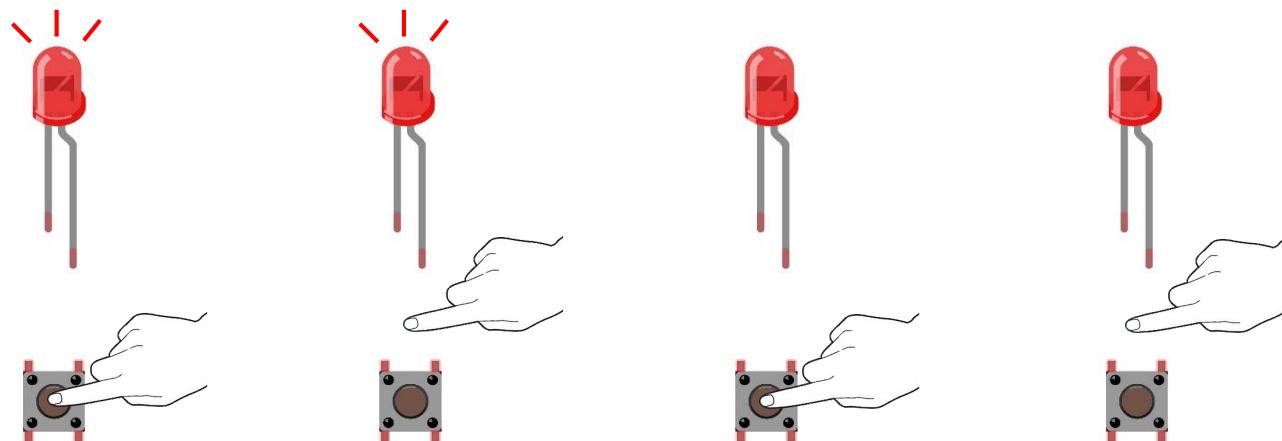
led = Pin(15, Pin.OUT)
button = Pin(13, Pin.IN, Pin.PULL_UP) #Create button object from Pin13 ,

def reverseGPIO():
    if led.value():
        led.value(0) #Set led turn on
    else:
        led.value(1) #Set led turn off

try:
    while True:
        if not button.value():
            time.sleep_ms(20)

```

Click “Run current script” shown in the box of the above illustration, press the push button switch, LED turns ON ; press it again, LED turns OFF. Press Ctrl+C or click “Stop/Restart backend” to exit file.



If you have any concerns, please contact us via: support@freenove.com

The following is the program code:

```

1  from machine import Pin
2  import time
3
4  led = Pin(15, Pin.OUT)
5  button = Pin(13, Pin.IN, Pin.PULL_UP)      #Create button object from Pin13 , Set Pin13 to input
6
7  def reverseGPIO():
8      if led.value():
9          led.value(0)                      #Set led turn on
10     else:
11         led.value(1)                      #Set led turn off
12
13 try:
14     while True:
15         if not button.value():
16             time.sleep_ms(20)
17             if not button.value():
18                 reverseGPIO()
19                 while not button.value():
20                     time.sleep_ms(20)
21 except:
22     pass

```

When the button is detected to be pressed, delay 20ms to avoid the effect of bounce, and then check whether the button has been pressed again. If so, the conditional statement will be executed, otherwise it will not be executed.

```

14 while True:
15     if not button.value():
16         time.sleep_ms(20)
17         if not button.value():
18             reverseGPIO()
19             while not button.value():
20                 time.sleep_ms(20)

```

Customize a function and name it reverseGPIO(), which reverses the output level of the LED.

```

7  def reverseGPIO():
8      if led.value():
9          led.value(0)                      #Set led turn on
10     else:
11         led.value(1)                      #Set led turn off

```

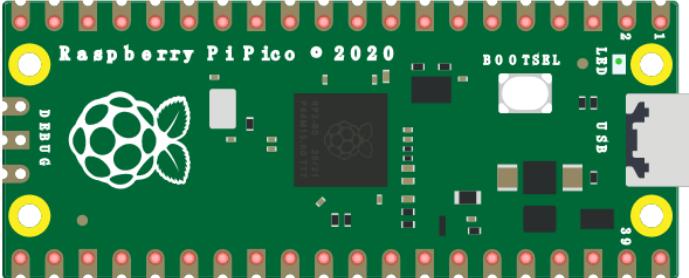
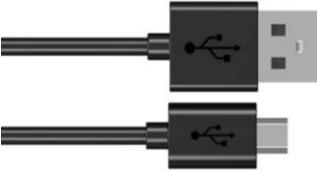
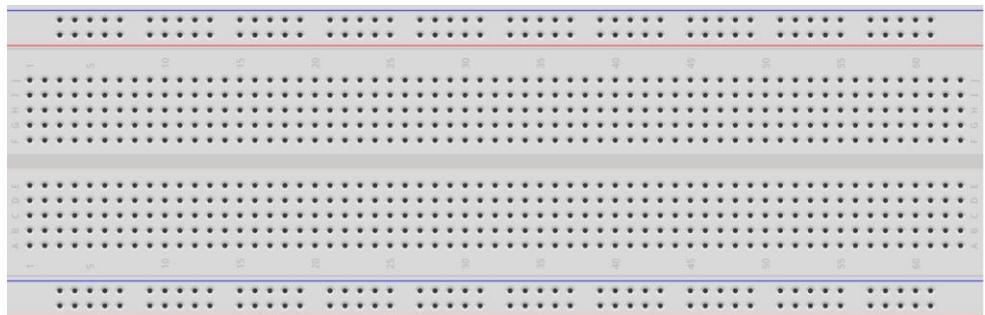
Chapter 3 LED Bar

We have learned how to control an LED blinking, next we will learn how to control a number of LEDs.

Project 3.1 Flowing Light

In this project, we use a number of LEDs to make a flowing light.

Component List

Raspberry Pi Pico x1	 A green printed circuit board (PCB) for the Raspberry Pi Pico. It features a central Broadcom SoC, a USB-C port, a micro-USB port, and several pins along the edges.	USB cable x1	 Two standard black USB cables, one with a Type-A connector and one with a Type-B connector.
Breadboard x1	 A breadboard with three horizontal rows of 40 pins each. The columns are labeled from 1 to 36 on the left and 40 to 56 on the right. The rows are color-coded: blue, red, yellow, green, and orange.		
Jumper	 A long, thin, grey jumper wire with black caps at both ends.	LED bar graph x1	 A black rectangular component with a grid of 16 white LED lights arranged in four rows of four.
		Resistor 220Ω x10	 A cylindrical resistor with a grey body, a red band, a black band, and a gold band.

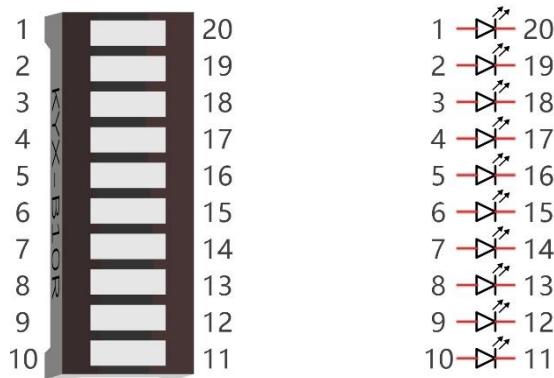


Component knowledge

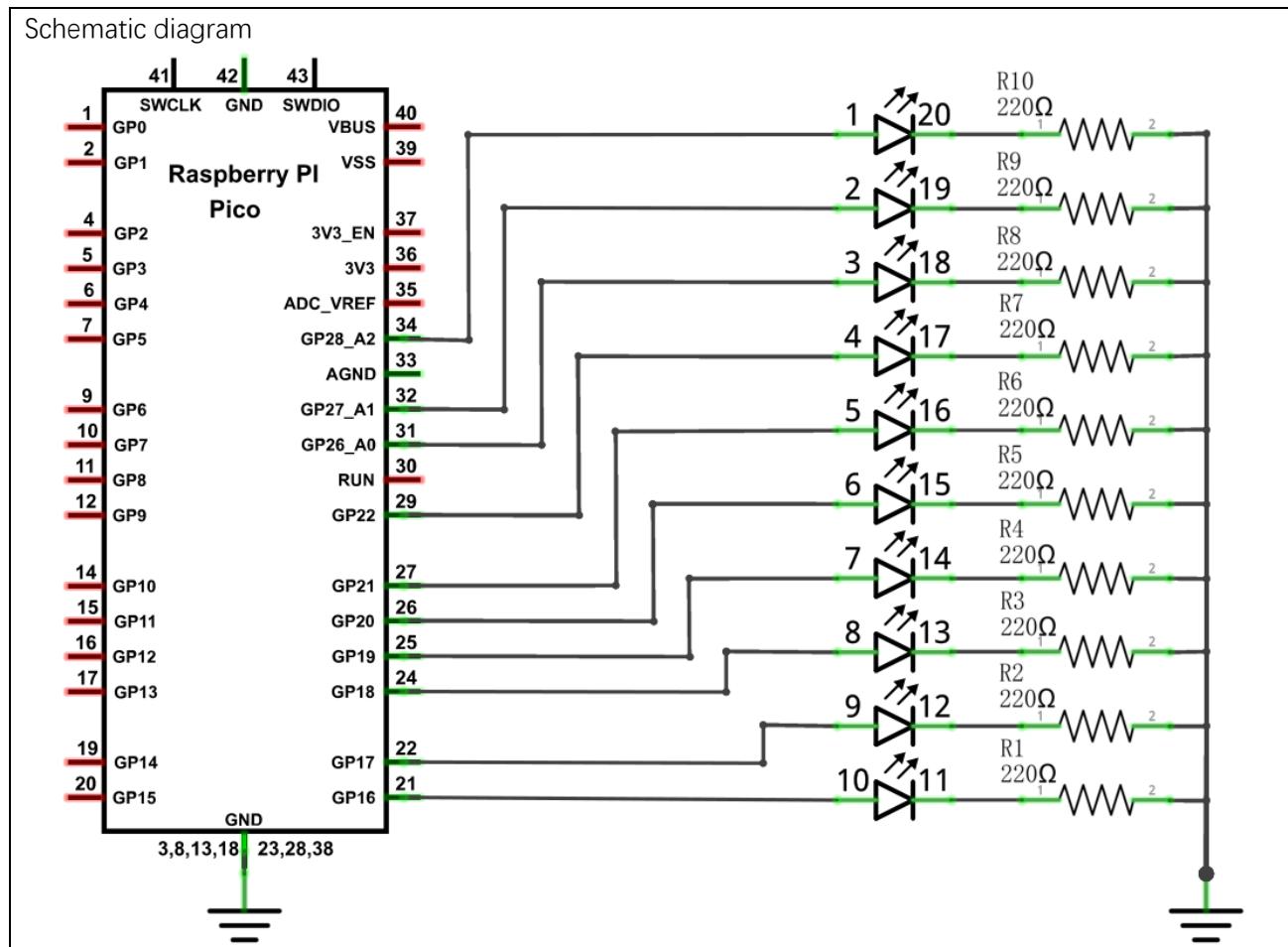
Let us learn about the basic features of these components to use and understand them better.

LED bar

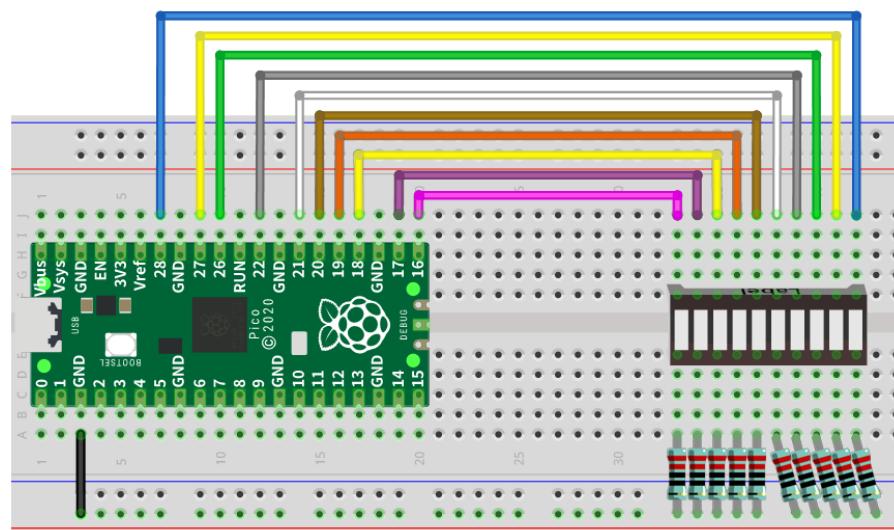
A Bar Graph LED has 10 LEDs integrated into one compact component. The two rows of pins at its bottom are paired to identify each LED like the single LED used earlier.



Circuit



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

If LEDbar doesn't work, try to rotate LEDbar for 180°. The label is random.

Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)



Code

This project is designed to make a flowing water lamp. Which are these actions: First turn LED #1 ON, then turn it OFF. Then turn LED #2 ON, and then turn it OFF... and repeat the same to all 10 LEDs until the last LED is turns OFF. This process is repeated to achieve the “movements” of flowing water.

03.1_FlowingLight

Move the program folder “**Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico/Python_Codes**” to disk(D) in advance with the path of “**D:/Micropython_Codes**”.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “03.1_FlowingLight” and double click “03.1_FlowingLight.py”.

```

from machine import Pin
import time

pins = [16, 17, 18, 19, 20, 21, 22, 26, 27, 28]
def showLed():
    for pin in pins:
        print(pin)
        led = Pin(pin, Pin.OUT)
        led.value(1)
        time.sleep_ms(100)
        led.value(0)
        time.sleep_ms(100)
    for pin in reversed(pins):
        print(pin)
        led = Pin(pin, Pin.OUT)
        led.value(1)

```

Use Stop/Restart to reconnect.

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>

Click “Run current script” shown in the box above, LED Bar Graph will light up from left to right and then back from right to left. Press Ctrl+C or click “Stop/Rotate backend” to exit the program.



If you have any concerns, please contact us via: support@freenove.com

Any concerns? ✉ support@freenove.com

The following is the program code:

```
1 from machine import Pin
2 import time
3
4 pins = [16, 17, 18, 19, 20, 21, 22, 26, 27, 28]
5 def showLed():
6     for pin in pins:
7         led = Pin(pin, Pin.OUT)
8         led.value(1)
9         time.sleep_ms(100)
10        led.value(0)
11        time.sleep_ms(100)
12    for pin in reversed(pins):
13        led = Pin(pin, Pin.OUT)
14        led.value(1)
15        time.sleep_ms(100)
16        led.value(0)
17        time.sleep_ms(100)
18
19 while True:
20     showLed()
```

Use an array to define 10 GPIO ports connected to LED Bar Graph for easier operation.

```
4 pins = [16, 17, 18, 19, 20, 21, 22, 26, 27, 28]
```

Use two for loops to turn on LEDs separately from left to right and then back from right to left.

```
5 def showLed():
6     for pin in pins:
7         led = Pin(pin, Pin.OUT)
8         led.value(1)
9         time.sleep_ms(100)
10        led.value(0)
11        time.sleep_ms(100)
12    for pin in reversed(pins):
13        led = Pin(pin, Pin.OUT)
14        led.value(1)
15        time.sleep_ms(100)
16        led.value(0)
17        time.sleep_ms(100)
```

Reference

for

For loop is used to execute a program endlessly and iterate in the order of items (a list or a string) in the sequence.

Commonly used:

“for pin in pins”

Pins is a list of elements that are iterated over by a for loop and assigned to pin each time.

“for i in range(start, end, num: 1)”

start: initial value, from which the for loop starts counting. The default initial value is 0. For example, range(5) equals to range(0, 5).

end: the value with which the function ends. For loop counts till it arrives at this value, but this value isn't included in the counting.

num: Num is automatically added each time to the data. The default value is 1.

range() function returns a sequence number which is assigned to I by for loop.

Chapter 4 Analog & PWM

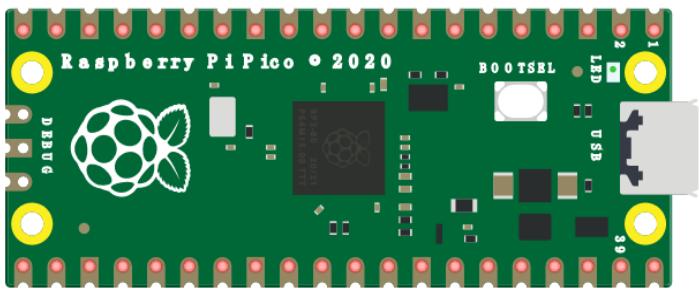
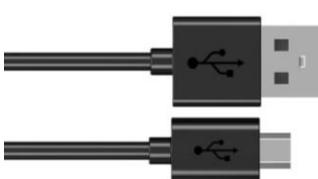
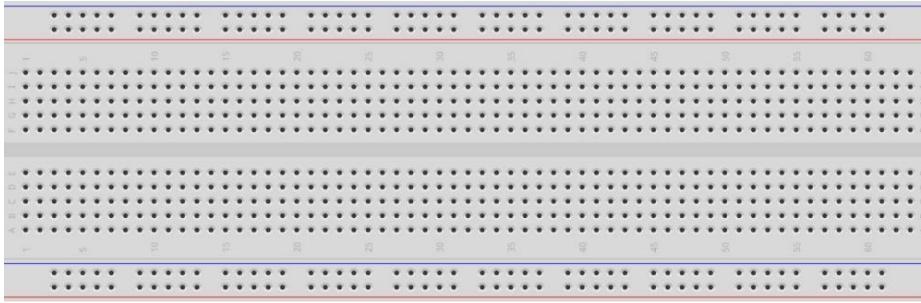
In previous study, we have known that one button has two states: pressed and released, and LED has light-on/off state, then how to enter a middle state? How to output an intermediate state to let LED "semi bright"? That's what we're going to learn.

First, let's learn how to control the brightness of an LED.

Project 4.1 Breathing LED

Breathing light, that is, LED is turned from off to on gradually, and gradually from on to off, just like "breathing". So, how to control the brightness of an LED? We will use PWM to achieve this target.

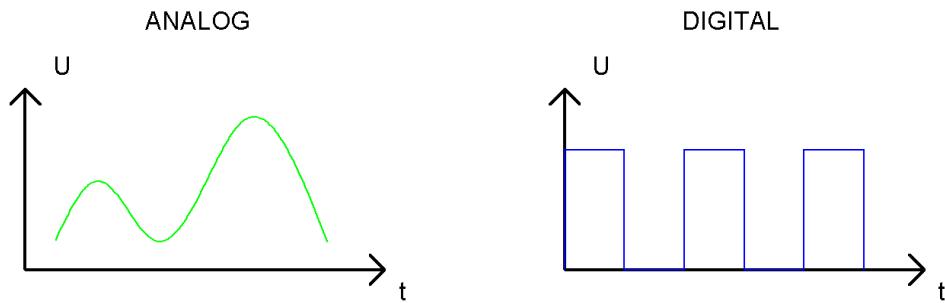
Component List

Raspberry Pi Pico x1	USB cable x1	
		
Breadboard x1		
LED x1	Resistor 220Ω x1	Jumper

Related knowledge

Analog & Digital

An Analog Signal is a continuous signal in both time and value. On the contrary, a Digital Signal or discrete-time signal is a time series consisting of a sequence of quantities. Most signals in life are analog signals. A familiar example of an Analog Signal would be how the temperature throughout the day is continuously changing and could not suddenly change instantaneously from 0°C to 10°C. However, Digital Signals can instantaneously change in value. This change is expressed in numbers as 1 and 0 (the basis of binary code). Their differences can more easily be seen when compared when graphed as below.



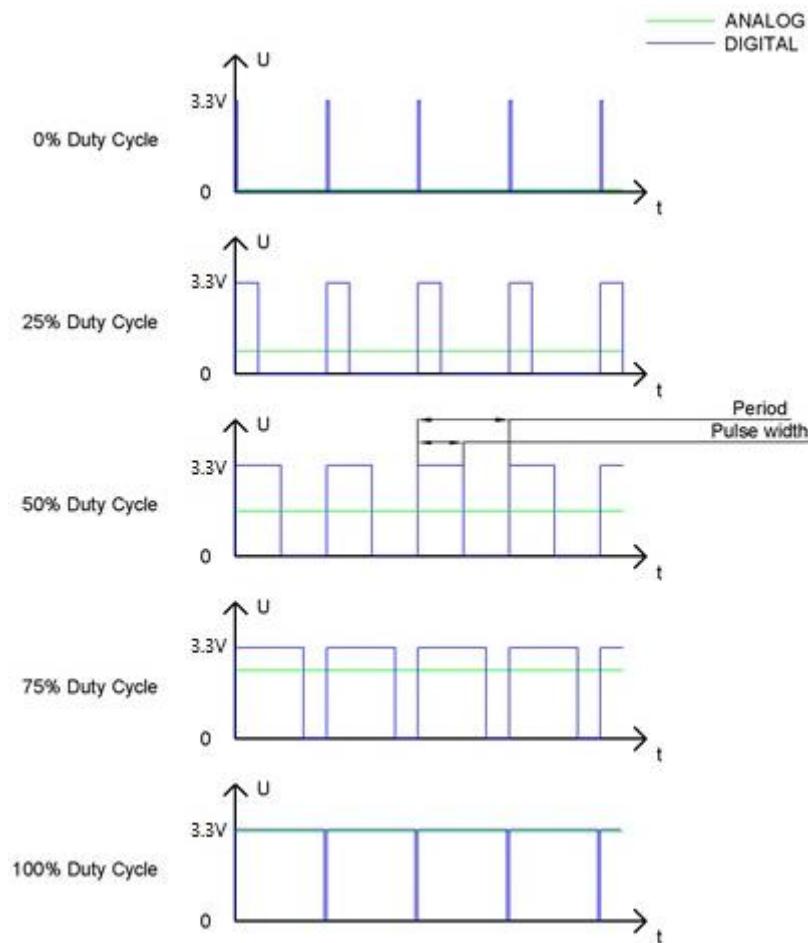
In practical application, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1), it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

PWM

PWM, Pulse-Width Modulation, is a very effective method for using digital signals to control analog circuits. Common processors cannot directly output analog signals. PWM technology makes it very convenient to achieve this conversion (translation of digital to analog signals).

PWM technology uses digital pins to send certain frequencies of square waves, that is, the output of high levels and low levels, which alternately last for a while. The total time for each set of high levels and low levels is generally fixed, which is called the period (Note: the reciprocal of the period is frequency). The time of high level outputs are generally called "pulse width", and the duty cycle is the percentage of the ratio of pulse duration, or pulse width (PW) to the total period (T) of the waveform.

The longer the output of high levels last, the longer the duty cycle and the higher the corresponding voltage in the analog signal will be. The following figures show how the analog signal voltages vary between 0V-5V (high level is 5V) corresponding to the pulse width 0%-100%:



The longer the PWM duty cycle is, the higher the output power will be. Now that we understand this relationship, we can use PWM to control the brightness of an LED or the speed of DC motor and so on. It is evident from the above that PWM is not real analog, and the effective value of the voltage is equivalent to the corresponding analog. so, we can control the output power of the LED and other output modules to achieve different effects.

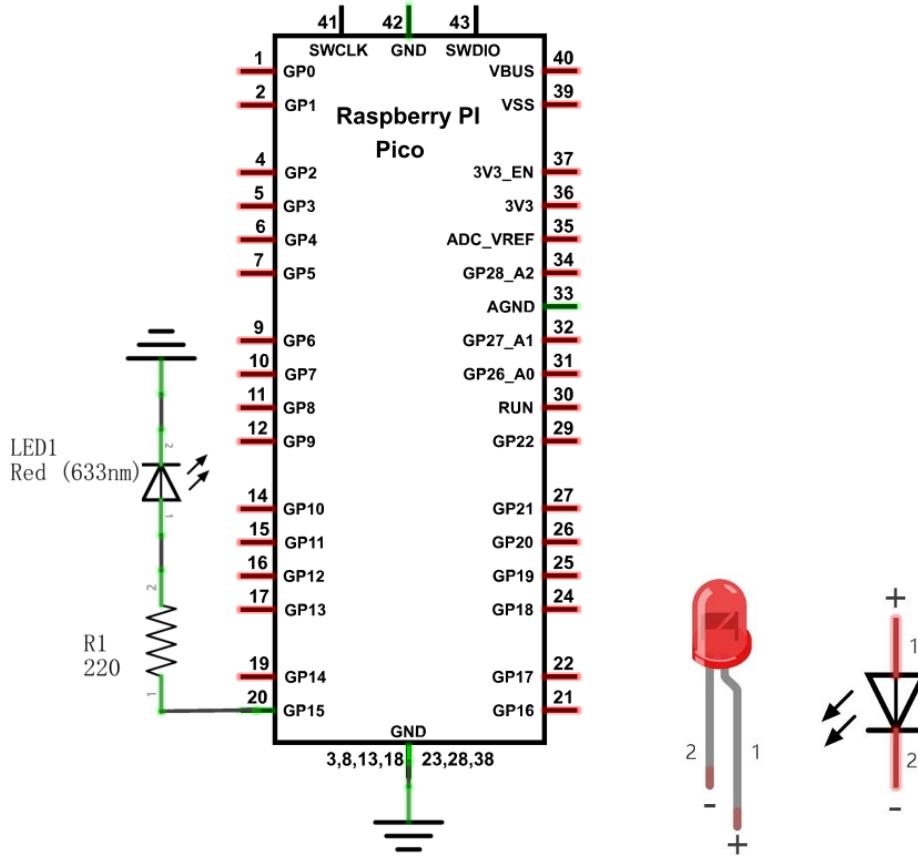
Raspberry Pi Pico and PWM

Raspberry Pi Pico has 16 PWM channels, each of which can control frequency and duty cycle independently, with the clock frequency ranges from 7Hz to 125MHz. Every pin on Raspberry Pi Pico can be configured as PWM output.

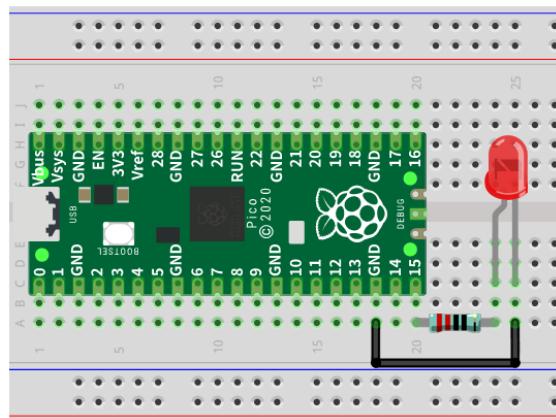
Circuit

This circuit is the same as the one in project Blink.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



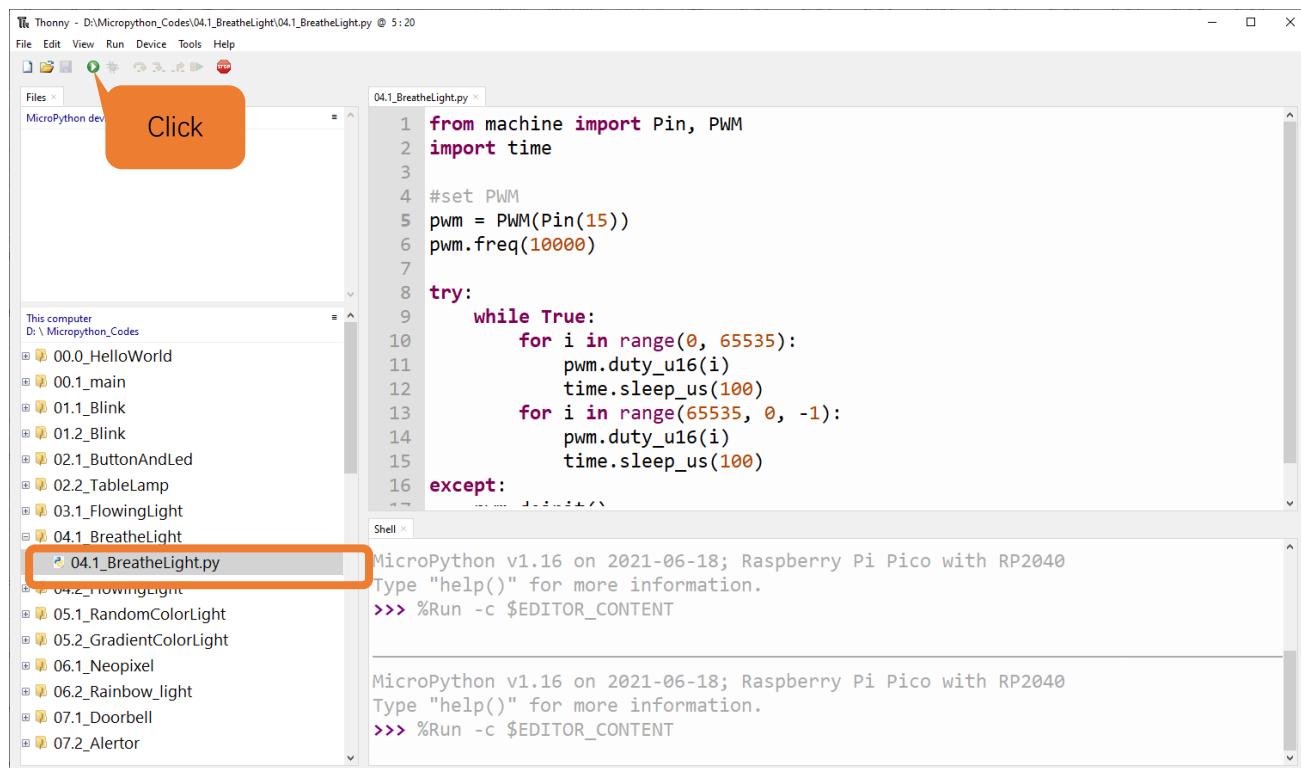
Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

Code

This project is designed to make PWM output GP15 with pulse width increasing from 0% to 100%, and then reducing from 100% to 0% gradually.

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "04.1_BreatheLight" and double click "04.1_BreatheLight.py".

04.1_BreatheLight



```

from machine import Pin, PWM
import time

#set PWM
pwm = PWM(Pin(15))
pwm.freq(10000)

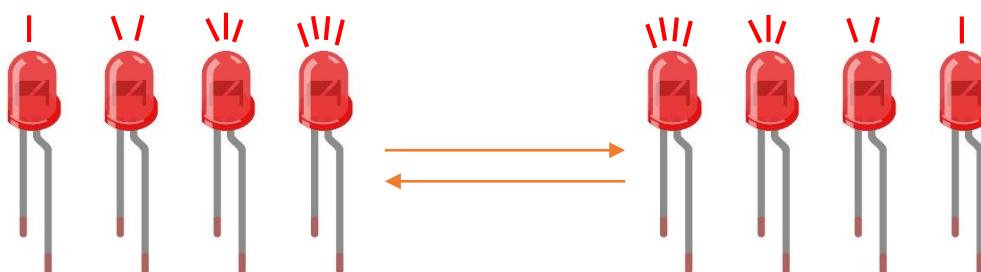
try:
    while True:
        for i in range(0, 65535):
            pwm.duty_u16(i)
            time.sleep_us(100)
        for i in range(65535, 0, -1):
            pwm.duty_u16(i)
            time.sleep_us(100)
except:
    ...

```

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
=>>> %Run -c \$EDITOR_CONTENT

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
=>>> %Run -c \$EDITOR_CONTENT

Click "Run current script", and you'll see that LED is turned from ON to OFF and then back from OFF to ON gradually like breathing. Press Ctrl+C or click "Stop/Restart backend" to exit program.



The following is the program code:

```

1  from machine import Pin, PWM
2  import time
3
4  #set PWM
5  pwm = PWM(Pin(15))
6  pwm.freq(10000)
7
8  try:
9      while True:
10         for i in range(0, 65535):
11             pwm.duty_u16(i)
12             time.sleep_us(100)
13         for i in range(65535, 0, -1):
14             pwm.duty_u16(i)
15             time.sleep_us(100)
16     except:
17         pwm.deinit()

```

Create a PWM object and configure GP15 as PWM output pin. Call freq() to set PWM output frequency of GP15 to 10000Hz.

```

5  pwm = PWM(Pin(15))
6  pwm.freq(10000)

```

The range of duty cycle is 0-65535, so we use the first for loop to control PWM to change the duty cycle value, making PWM output 0% - 100%; Use the second for loop to make PWM output 100%-0%.

```

10     for i in range(0, 65535):
11         pwm.duty_u16(i)
12         time.sleep_us(100)
13     for i in range(65535, 0, -1):
14         pwm.duty_u16(i)
15         time.sleep_us(100)

```

Each time PWM is used, the hardware Timer will be turned ON to cooperate it. Therefore, after each use of PWM, deinit() needs to be called to turned OFF the timer. Otherwise, the PWM may fail to work next time.

```

17     pwm.deinit()

```

Reference

Class PWM(pin)

Before each use of PWM module, please add the statement “**from machine import PWM**” to the top of the python file.

pin: PWM pins are supported, such as GP(0~22)、GP(25)、GP(26~28).

PWM.freq(freq_val): the function is used to set PWM frequency and returns nothing; when there is no parameter, the function obtains and returns PWM frequency.

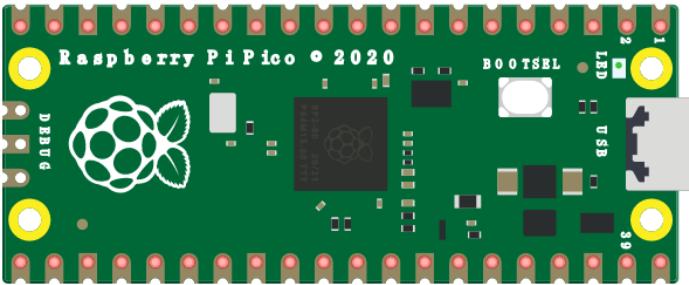
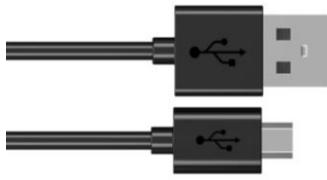
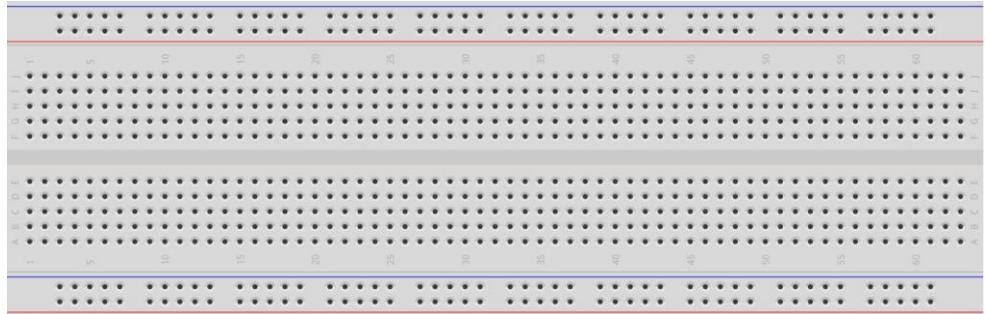
PWM.duty_u16(duty_val): the function is used to set PWM duty cycle, among which, duty_val ranges from 0 to 65535. If there is no parameter, the function returns to currently set duty cycle. If duty cycle has not yet been set, it returns 0.

PWM.deinit(): Turn OFF PWM.

Project 4.2 Meteor Flowing Light

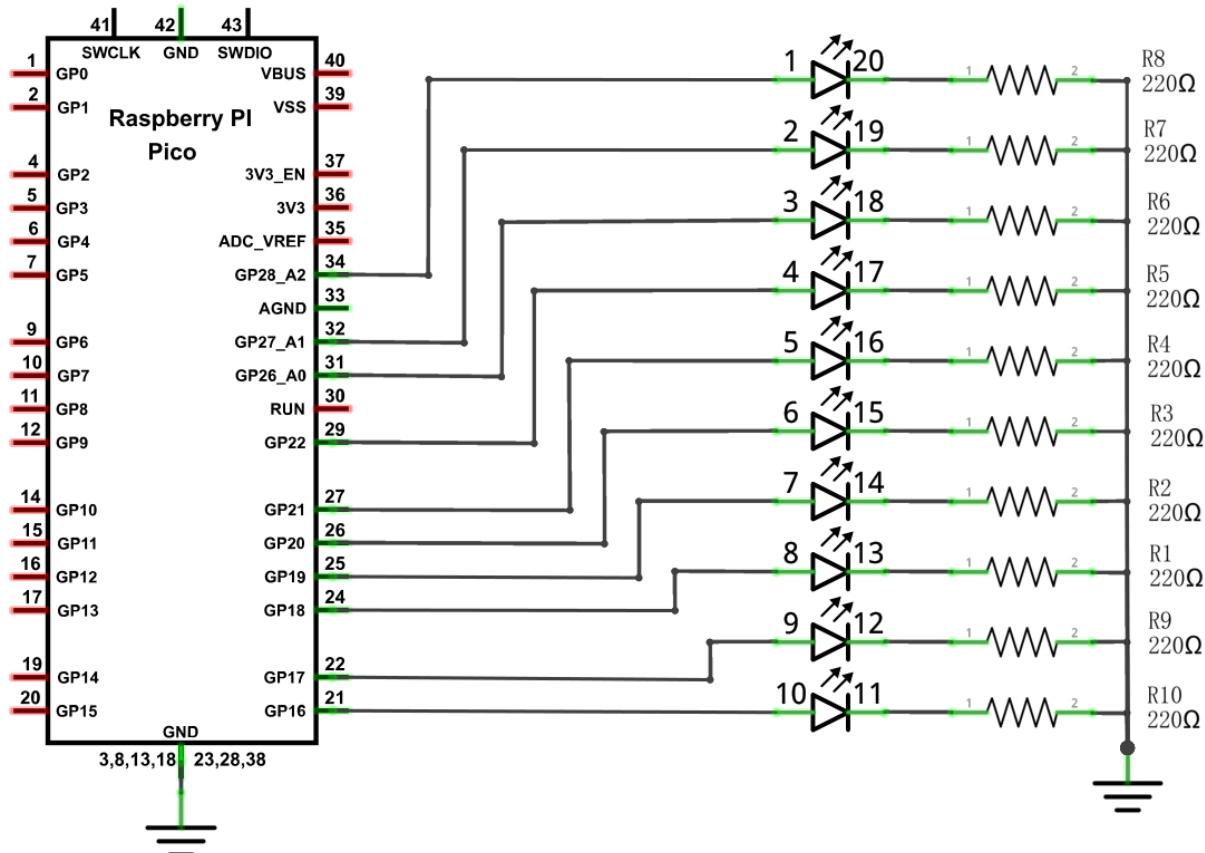
Having learned about PWM, we can use it to control LED Bar Graph and realize a cooler Flowing Light.

Component List

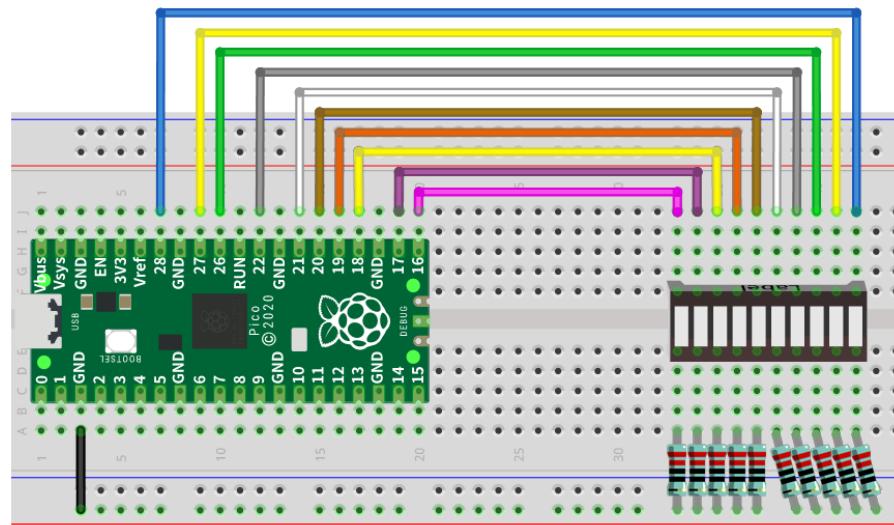
Raspberry Pi Pico x1	USB cable x1	
		
Breadboard x1		
Jumper	LED bar graph x1	Resistor 220Ω x10

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

If LEDbar doesn't work, try to rotate LEDbar for 180°. The label is random.

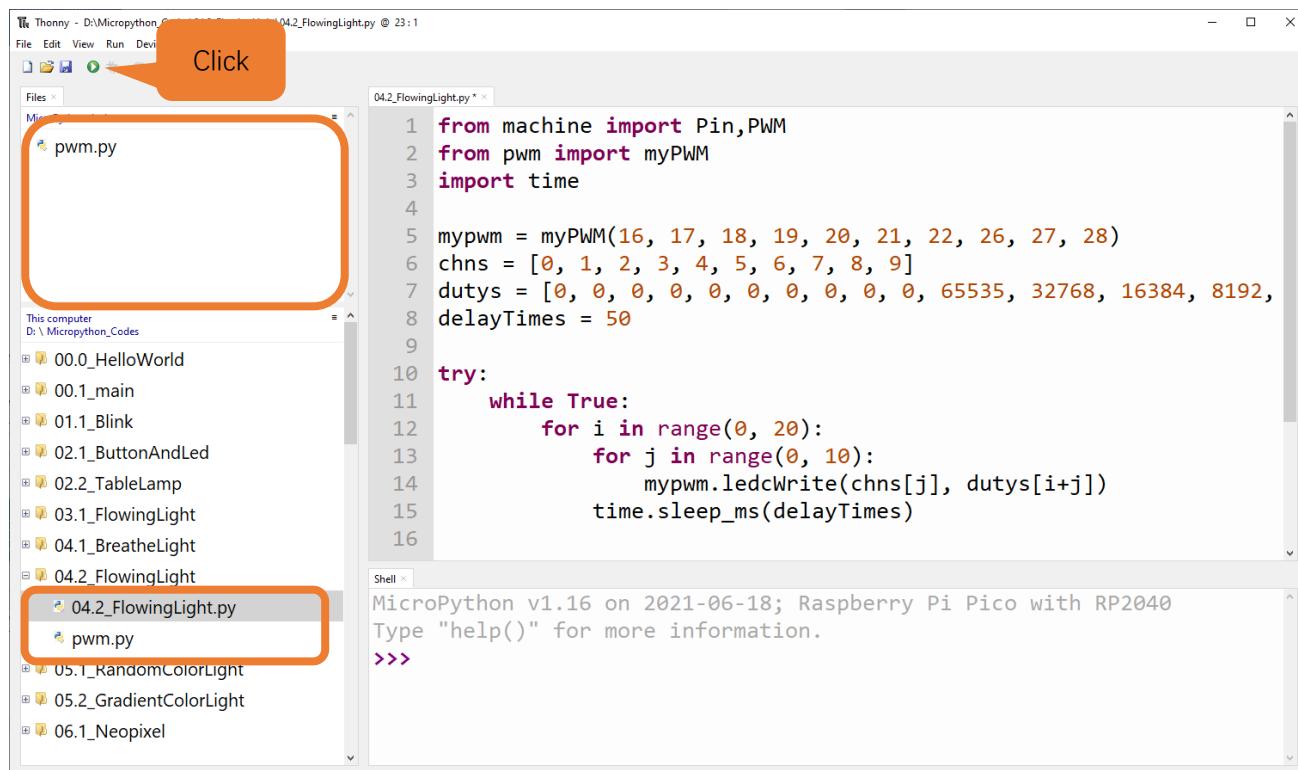
Any concerns? ✉ support@freenove.com

Code

Flowing Light with tail was implemented with PWM.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “04.2_FlowingLight”. Select “pwm.py”, right click to select “Upload to /”, wait for “pwm.py” to be uploaded to Raspberry Pi Pico and then double click “04.2_FlowingLight.py”.

04.2_FlowingLight



Click “Run current script”, and LED Bar Graph will gradually light up and out from left to right, then light up and out from right to left. Press Ctrl+C or click “Stop/Restart backend” to exit program.

The following is the program code:

```

from machine import Pin, PWM
from pwm import myPWM
import time

mypwm = myPWM(16, 17, 18, 19, 20, 21, 22, 26, 27, 28)
chns = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
dutys = [0, 0, 0, 0, 0, 0, 0, 0, 0, 65535, 32768, 16384, 8192, 4096, 2048, 1024, 512, 256, 128, 0, 0, 0, 0, 0, 0, 0, 0]
delayTimes = 50

try:
    while True:
        for i in range(0, 20):
            for j in range(0, 10):
                mypwms.ledcWrite(chns[j], dutys[i+j])
            time.sleep_ms(delayTimes)

```

```

13     for j in range(0, 10):
14         mypwm.ledcWrite(chns[j], dutys[i+j])
15         time.sleep_ms(delayTimes)
16
17     for i in range(0, 20):
18         for j in range(0, 10):
19             mypwm.ledcWrite(chns[9-j], dutys[i+j])
20             time.sleep_ms(delayTimes)
21     except:
22         mypwm.deinit()

```

Import the object myPWM from pwm.py and set corresponding pins for PWM channel.

```

2  from pwm import myPWM
...
5  mypwm = myPWM(16, 17, 18, 19, 20, 21, 22, 26, 27, 28)

```

Create an object for myPWM and configure 10 PWM output pins. Define 10 PWM channels and 30 pulse width values.

```

5  mypwm = myPWM(16, 17, 18, 19, 20, 21, 22, 26, 27, 28)
6  chns = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
7  dutys = [0, 0, 0, 0, 0, 0, 0, 0, 65535, 32768, 16384, 8192, 4096, 2048, 1024, 512, 256,
128, 0, 0, 0, 0, 0, 0, 0, 0, 0]

```

Call ledcWrite() to set duty cycle dutys[i+j] for the chns[j] channel of PWM.

```

14  mypwm.ledcWrite(chns[j], dutys[i+j])

```

Turn OFF the PWM of the object myPWM.

```

22  mypwm.deinit()

```

In the code, a nesting of two for loops are used to achieve this effect.

```

12  for i in range(0, 20):
13      for j in range(0, 10):
14          mypwm.ledcWrite(chns[j], dutys[i+j])
15          time.sleep_ms(delayTimes)
16
17      for i in range(0, 20):
18          for j in range(0, 10):
19              mypwm.ledcWrite(chns[9 -j], dutys[i+j])
20              time.sleep_ms(delayTimes)

```

In the main function, a nested for loop is used to control the pulse width of the PWM. Every time i in the first for loop increases by 1, the LED Bar Graph will move one grid, and gradually change according to the value in the array dutys. As shown in the following table, the value in the second row is the value of the array dutys, and the 10 green grids in each row below represent the 10 LEDs on the LED Bar Graph. Each time i increases by 1, the value of the LED Bar Graph will move to the right by one grid, and when it reaches the end, it will move from the end to the starting point, achieving the desired effect.

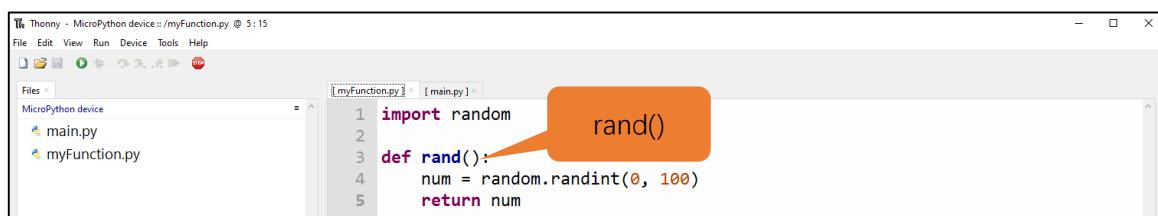
0	1	2	3	...	7	8	9	1	1	1	1	1	1	1	1	1	1	2	2	2	...	2	2	3
d	0	0	0	...	0	0	0	0	6	3	1	8	4	2	1	5	2	1	0	0	0	0	0	0
i									5	2	6	1	0	0	0	1	5	2						
									5	7	3	9	9	4	2	2	6	8						
0									3	6	8	2	6	8	4									
1									5	8	4													
...																								
18																								
19																								
20																								

How to import a custom python module

Each Python file, as long as it's stored on the file system of Raspberry Pi Pico, is a module. To import a custom module, the module file needs to be located in the MicroPython environment variable path or in the same path as the currently running program.

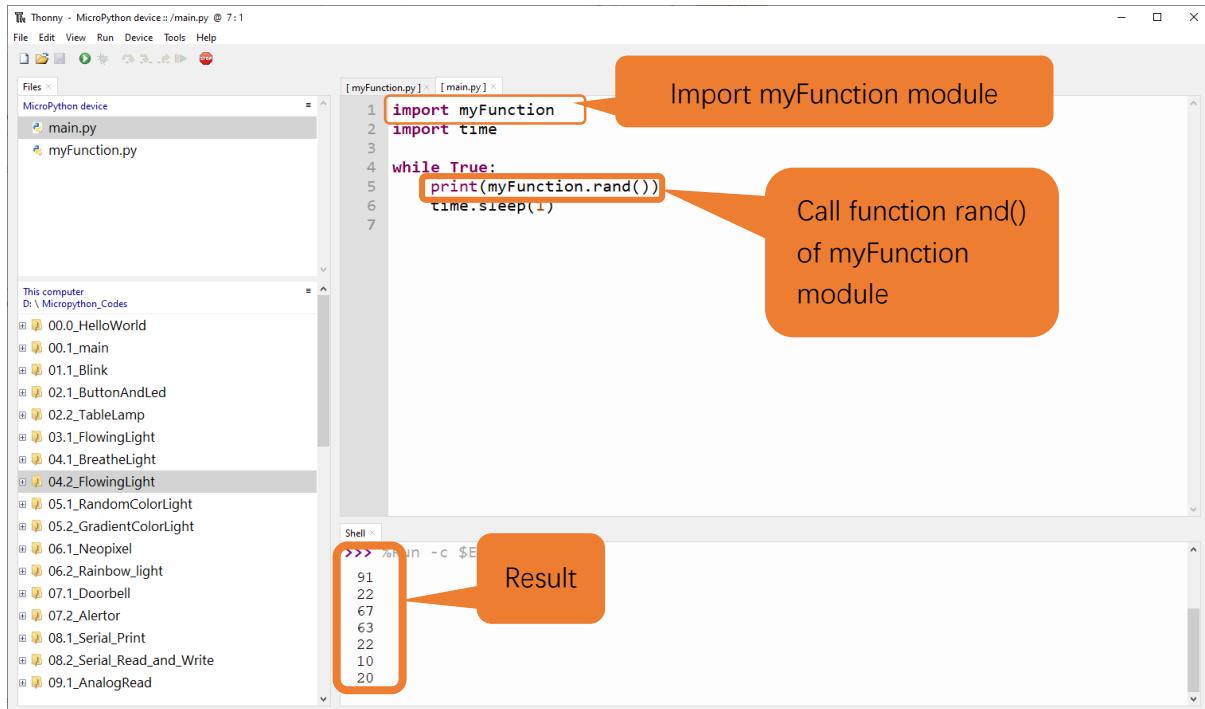
Code

First, customize a python module "myFunction.py". Create a new py file and name it "myFunction.py". Write code to it and save it to Raspberry Pi Pico.



rand() function randomly creates an integer ranging from 0 to 99.

Second, import myFunction module "myFunction" to main.py.



The following is the program code:

1. myFunction.py

```

1 import random
2
3 def rand():
4     num = random.randint(0, 100)
5     return num

```

Import random module.

```
1 import random
```

Call randint() function in random module to randomly generate an integer at the range of 0-99 and assign it to num variable.

```
4 num = random.randint(0, 100)
```

2. main.py

```

1 import myFunction
2 import time
3
4 while True:
5     print(myFunction.rand())
6     time.sleep(1)

```

Import myFunction module "myFunction" to main.py.

```
1 import myFunction
```

Call rand() in myFunction module.

```
5     print(myFunction.rand())
```

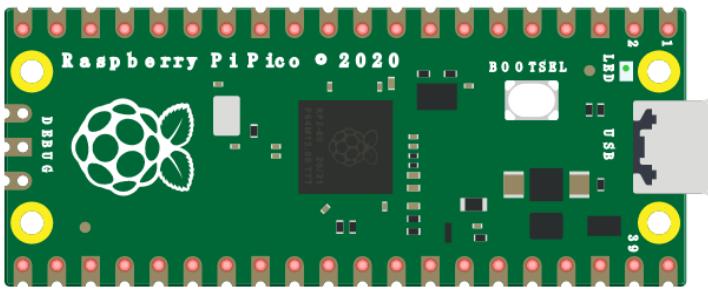
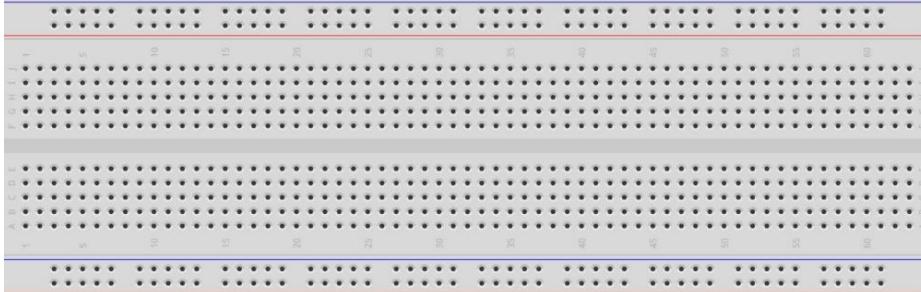
Chapter 5 RGBLED

In this chapter, we will learn how to control an RGBLED. It can emit different colors of light. Next, we will use RGBLED to make a multicolored light.

Project 5.1 Random Color Light

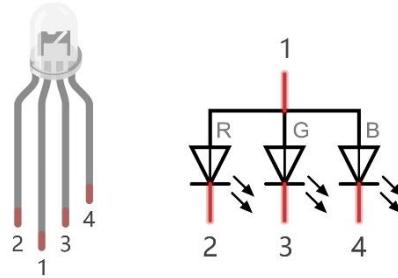
In this project, we will make a multicolored LED. And we can control RGBLED to switch different colors automatically.

Component List

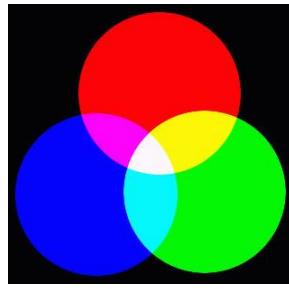
Raspberry Pi Pico x1	USB cable x1	
Breadboard x1		
RGBLED x1	Resistor 220Ω x3	Jumper
		

Related knowledge

RGB LED has integrated 3 LEDs that can respectively emit red, green and blue light. And it has 4 pins. The long pin (1) is the common port, that is, 3 LED's positive or negative port. The RGB LED with common positive port and its symbol is shown below. We can make RGB LED emit various colors of light by controlling these 3 LEDs to emit light with different brightness.



Red, green, and blue light are known as three primary colors. When you combine these three primary-color lights with different brightness, it can produce almost all kinds of visible lights. Computer screens, single pixel of cell phone screen, neon, and etc. are working under this principle.

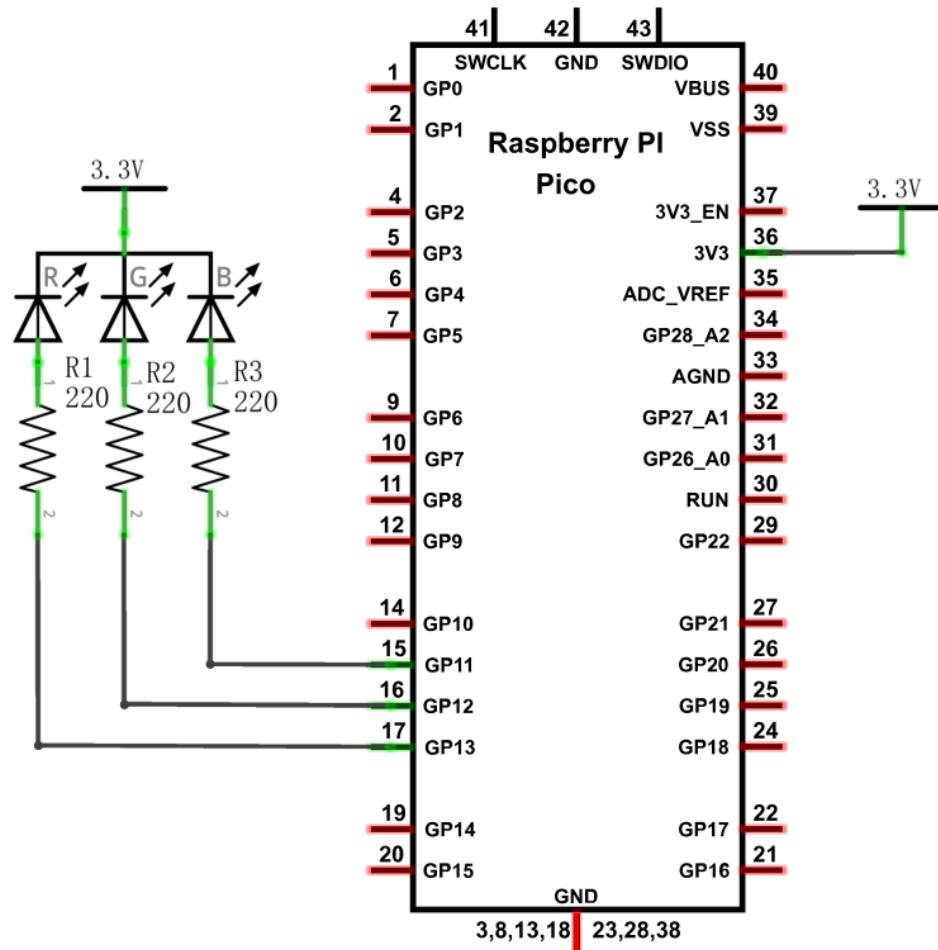


RGB

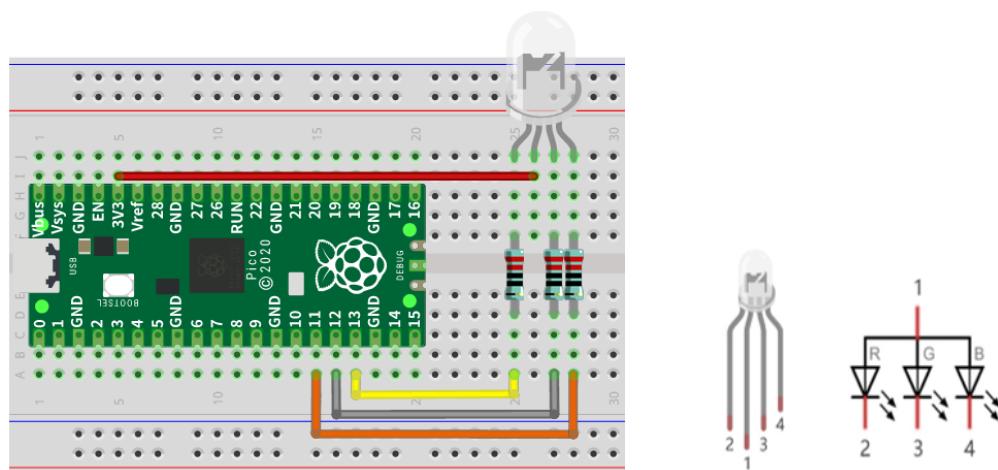
If we use three 10-bit PWM to control the RGBLED, in theory, we can create $2^{10} * 2^{10} * 2^{10} = 1,073,741,824$ (1 billion) colors through different combinations.

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

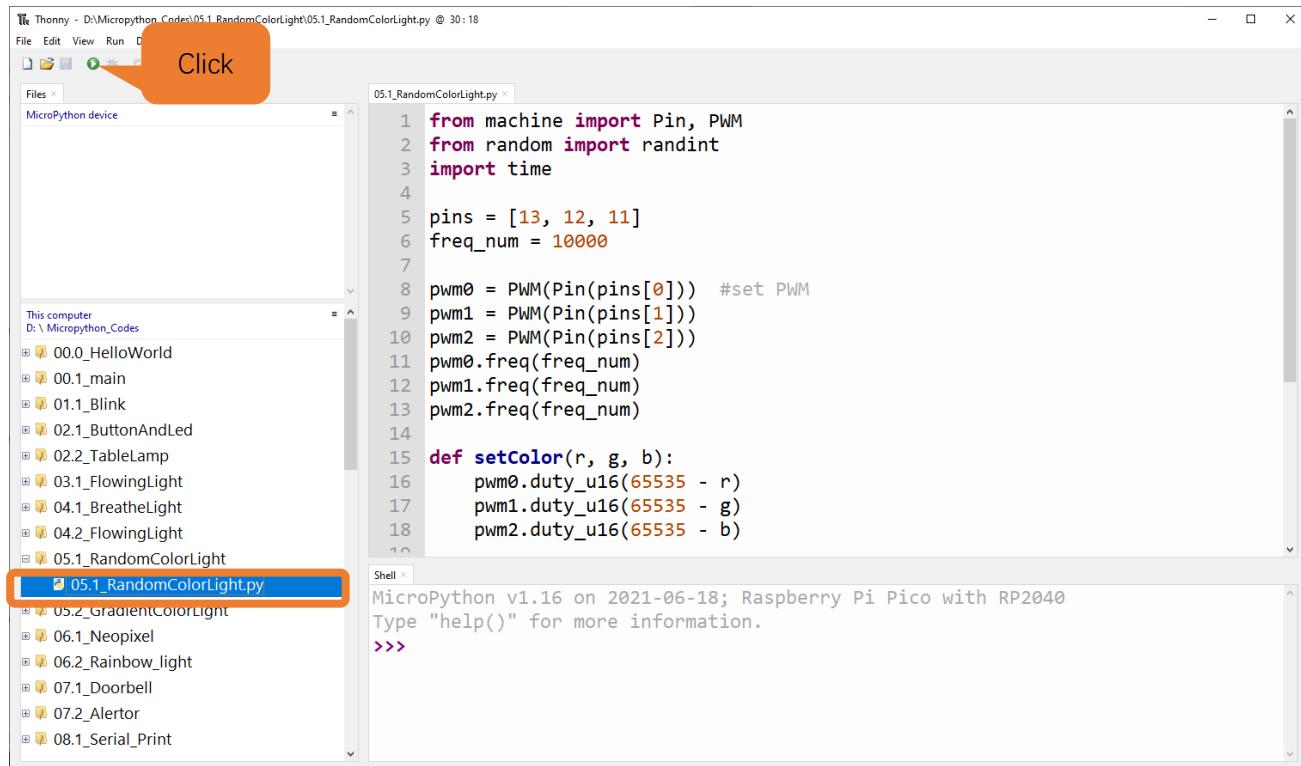
Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

Code

We need to create three PWM channels and use random duty cycle to make random RGBLED color.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “05.1_RandomColorLight” and double click “05.1_RandomColorLight.py”.

05.1_RandomColorLight



```

from machine import Pin, PWM
from random import randint
import time

pins = [13, 12, 11]
freq_num = 10000

pwm0 = PWM(Pin(pins[0])) #set PWM
pwm1 = PWM(Pin(pins[1]))
pwm2 = PWM(Pin(pins[2]))
pwm0.freq(freq_num)
pwm1.freq(freq_num)
pwm2.freq(freq_num)

def setColor(r, g, b):
    pwm0.duty_u16(65535 - r)
    pwm1.duty_u16(65535 - g)
    pwm2.duty_u16(65535 - b)

```

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>

Click “Run current script”, RGBLED begins to display random colors. Press Ctrl+C or click “Stop/Restart backend” to exit program.

If you have any concerns, please contact us via: support@freenove.com

Any concerns? ✉ support@freenove.com

The following is the program code:

```

1  from machine import Pin, PWM
2  from random import randint
3  import time
4
5  pins = [13, 12, 11]
6  freq_num = 10000
7
8  pwm0 = PWM(Pin(pins[0])) #set PWM
9  pwm1 = PWM(Pin(pins[1]))
10 pwm2 = PWM(Pin(pins[2]))
11 pwm0.freq(freq_num)
12 pwm1.freq(freq_num)
13 pwm2.freq(freq_num)
14
15 def setColor(r, g, b):
16     pwm0.duty_u16(65535 - r)
17     pwm1.duty_u16(65535 - g)
18     pwm2.duty_u16(65535 - b)
19
20 try:
21     while True:
22         red   = randint(0, 65535)
23         green = randint(0, 65535)
24         blue  = randint(0, 65535)
25         setColor(red, green, blue)
26         time.sleep_ms(200)
27 except:
28     pwm0.deinit()
29     pwm1.deinit()
30     pwm2.deinit()
```

Import Pin, PWM and Random function modules.

```

1  from machine import Pin, PWM
2  from random import randint
3  import time
```

Configure ouput mode of GP11, GP12 and GP13 as PWM output and PWM frequency as 10000Hz.

```

5  pins = [13, 12, 11]
6  freq_num = 10000
7
8  pwm0 = PWM(Pin(pins[0])) #set PWM
9  pwm1 = PWM(Pin(pins[1]))
10 pwm2 = PWM(Pin(pins[2]))
11 pwm0.freq(freq_num)
12 pwm1.freq(freq_num)
```

```
13    pwm2.freq(freq_num)
```

Define a function to set the color of RGBLED.

```
15    def setColor(r, g, b):
16        pwm0.duty_u16(65535 - r)
17        pwm1.duty_u16(65535 - g)
18        pwm2.duty_u16(65535 - b)
```

Call random function randint() to generate a random number in the range of 0-65535 and assign the value to red.

```
22    red = randint(0, 65535)
```

Obtain 3 random numbers every 200 milliseconds and call function setColor to make RGBLED display dazzling colors.

```
21    while True:
22        red = randint(0, 65535)
23        green = randint(0, 65535)
24        blue = randint(0, 65535)
25        setColor(red, green, blue)
26        time.sleep_ms(200)
```

Reference

Class random

Before each use of the module **random**, please add the statement “**import random**” to the top of Python file.

randint(start, end): Randomly generates an integer between the value of start and end.

start: Starting value in the specified range, which would be included in the range.

end: Ending value in the specified range, which would be included in the range.

random(): Randomly generates a floating point number between 0 and 1.

random.uniform(start, end): Randomly generates a floating point number between the value of start and end.

start: Starting value in the specified range, which would be included in the range.

end: Ending value in the specified range, which would be included in the range.

random.getrandbits(size): Generates an integer with size random bits.

For example:

size = 4, it generates an integer in the range of 0 to 0b1111.

size = 8, it generates an integer in the range of 0 to 0b11111111.

random.randrange(start, end, step): Randomly generates a positive integer in the range from start to end and increment to step.

start: Starting value in the specified range, which would be included in the range.

end: Ending value in the specified range, which would be included in the range.

step: An integer specifying the incrementation.

random.seed(sed): Specifies a random seed, usually being applied in conjunction with other random number generators.

sed: Random seed, a starting point in generating random numbers.

random.choice(obj): Randomly generates an element from the object obj.

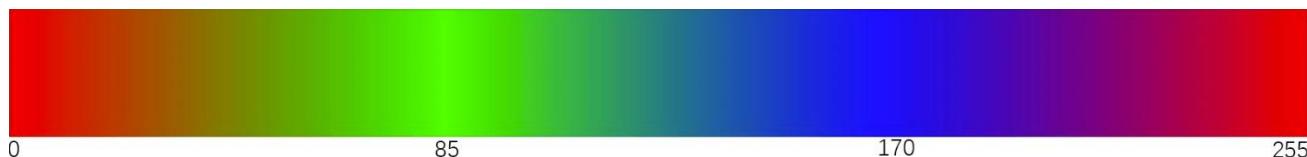
obj: list of elements.

Project 5.2 Gradient Color Light

In the previous project, we have mastered the usage of RGBLED, but the random color display is rather stiff. This project will realize a fashionable Light with soft color changes.

Component list, the circuit is exactly the same as the project random color light.

Using a color model, the color changes from 0 to 255 as shown below.



Code

In this code, the color model will be implemented and RGBLED will change colors along the model.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “05.2_GradientColorLight” and double click “05.2_GradientColorLight.py”.

05.2_GradientColorLight

```

from machine import Pin, PWM
import time

pins = [13, 12, 11]

#set pwm
pwm0 = PWM(Pin(pins[0]))
pwm1 = PWM(Pin(pins[1]))
pwm2 = PWM(Pin(pins[2]))
pwm0.freq(1000)
pwm1.freq(1000)
pwm2.freq(1000)

def setColor(rgb):
    pwm0.duty_u16(65535 - (rgb >> 4))

```

The screenshot shows the Thonny IDE interface. On the left, the file tree displays several projects and scripts, with "05.2_GradientColorLight.py" selected. On the right, the main window shows the Python code for the script. The code imports the necessary modules, initializes three PWM pins (pins 13, 12, and 11), and defines a function "setColor" that takes an RGB value and sets the duty cycle of the PWM pins accordingly. Below the code editor is a terminal window showing the MicroPython version and a prompt for commands.

Click “Run current script”, and the light emitted by RGBLED will change gradually. Press Ctrl+C or click “Stop/Restart backend” to exit program.

The following is the program code:

```
1  from machine import Pin, PWM
2  import time
3
4  pins = [13, 12, 11]
5
6  #set pwm
7  pwm0 = PWM(Pin(pins[0]))
8  pwm1 = PWM(Pin(pins[1]))
9  pwm2 = PWM(Pin(pins[2]))
10 pwm0.freq(1000)
11 pwm1.freq(1000)
12 pwm2.freq(1000)
13
14 def setColor(rgb):
15     pwm0.duty_u16(65535 - (rgb >> 4))
16     pwm1.duty_u16(65535 - (rgb >> 1))
17     pwm2.duty_u16(65535 - (rgb >> 0))
18
19 def wheel(pos):
20     WheelPos = pos % 65535
21     if WheelPos < 21845:
22         return (((65535 - WheelPos*3) << 4) | ((WheelPos*3) << 1))
23     elif WheelPos >= 21845 and WheelPos < 43690:
24         WheelPos -= 21845
25         return (((65535 - WheelPos*3) << 1) | (WheelPos*3))
26     else :
27         WheelPos -= 43690
28         return (((WheelPos*3) << 4) | (65535 - WheelPos*3))
29
30 try:
31     while True:
32         for i in range(0, 65535):
33             setColor(wheel(i))
34             time.sleep_ms(10)
35     except:
36         pwm0.deinit()
37         pwm1.deinit()
38         pwm2.deinit()
```

In the function **setColor()**, we use a variable to represent the value of RGB, making it more convenient for the passing of parameters. As the range of PWM's duty cycle is 0-65535, which is 2 to the sixteen power, when split, the value of each color channel can be obtained with a simple bitwise operation.

```
14 def setColor(rgb):  
15     pwm0.duty_u16(65535 - (rgb >> 4))  
16     pwm1.duty_u16(65535 - (rgb >> 1))  
17     pwm2.duty_u16(65535 - (rgb >> 0))
```

The function **wheel()** is a color selection method of the color model introduced earlier. The value range of the parameter pos is 0-65535. The function will return a data containing the duty cycle values of 3 pins.

```
19 def wheel(pos):  
20     WheelPos = pos % 65535  
21     if WheelPos < 21845:  
22         return (((65535 - WheelPos*3) << 4) | ((WheelPos*3) << 1))  
23     elif WheelPos >= 21845 and WheelPos < 43690:  
24         WheelPos -= 21845  
25         return (((65535 - WheelPos*3) << 1) | (WheelPos*3))  
26     else :  
27         WheelPos -= 43690  
28         return (((WheelPos*3) << 4) | (65535 - WheelPos*3))
```

Chapter 6 NeoPixel

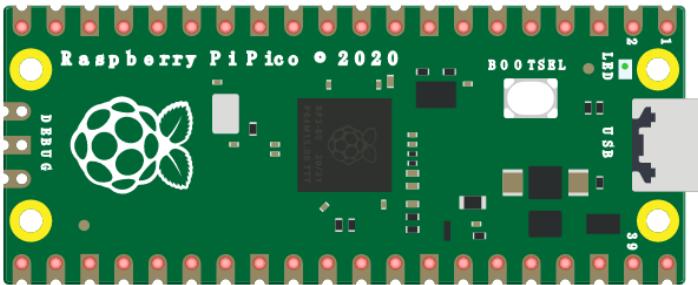
This chapter will help you learn to use a more convenient RGBLED lamp, which requires only one GPIO control and can be connected in infinite series in theory. Each LED can be controlled independently.

Project 6.1 NeoPixel

Learn the basic usage of NeoPixel and use it to blink red, green, blue and white.

Component List

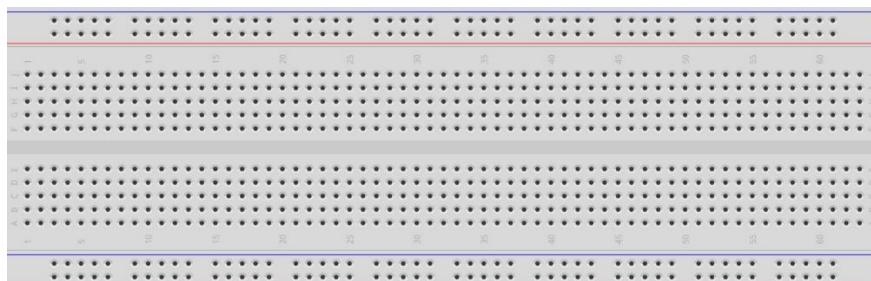
Raspberry Pi Pico x1



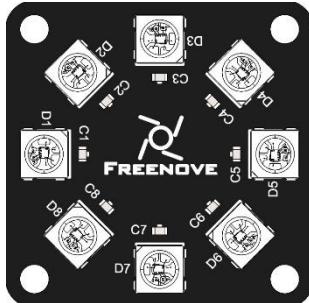
USB cable x1



Breadboard x1



Freenove 8 RGB LED Module x1



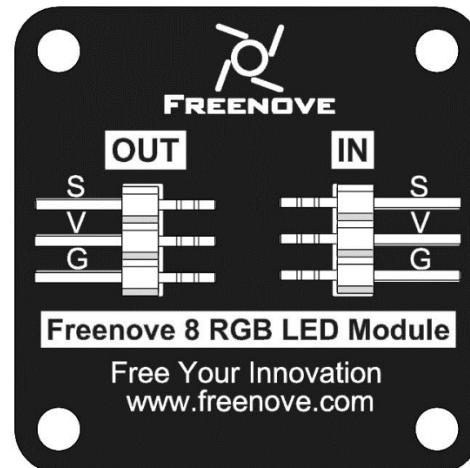
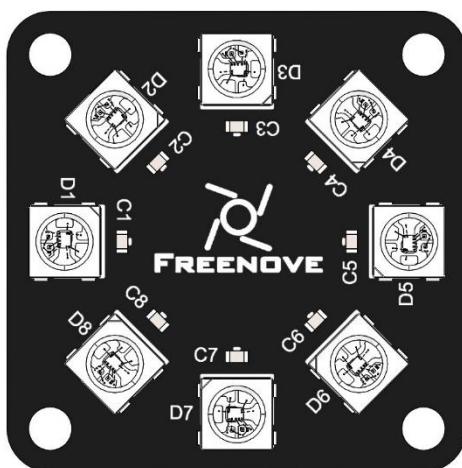
Jumper



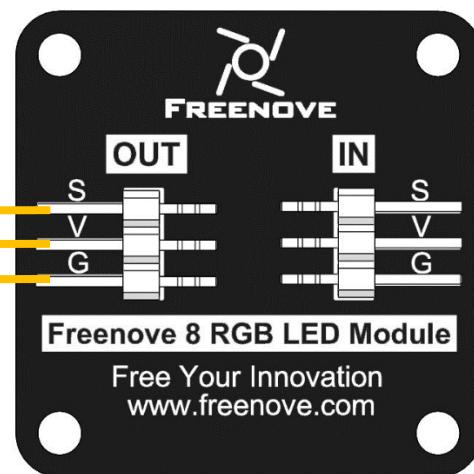
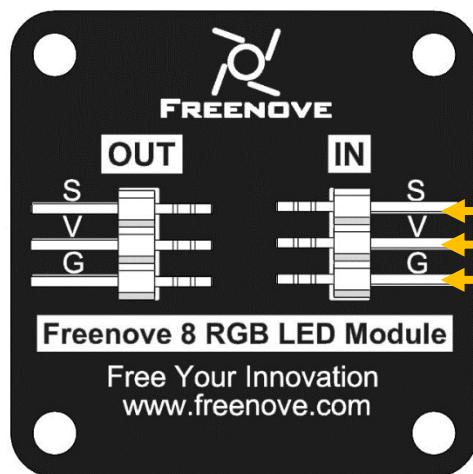
Related knowledge

Freenove 8 RGB LED Module

The Freenove 8 RGB LED Module is as below. You can use only one data pin to control eight LEDs on the module. As shown below:



And you can also control many modules at the same time. Just connect OUT pin of one module to IN pin of another module. In this way, you can use one data pin to control 8, 16, 32 ... LEDs.

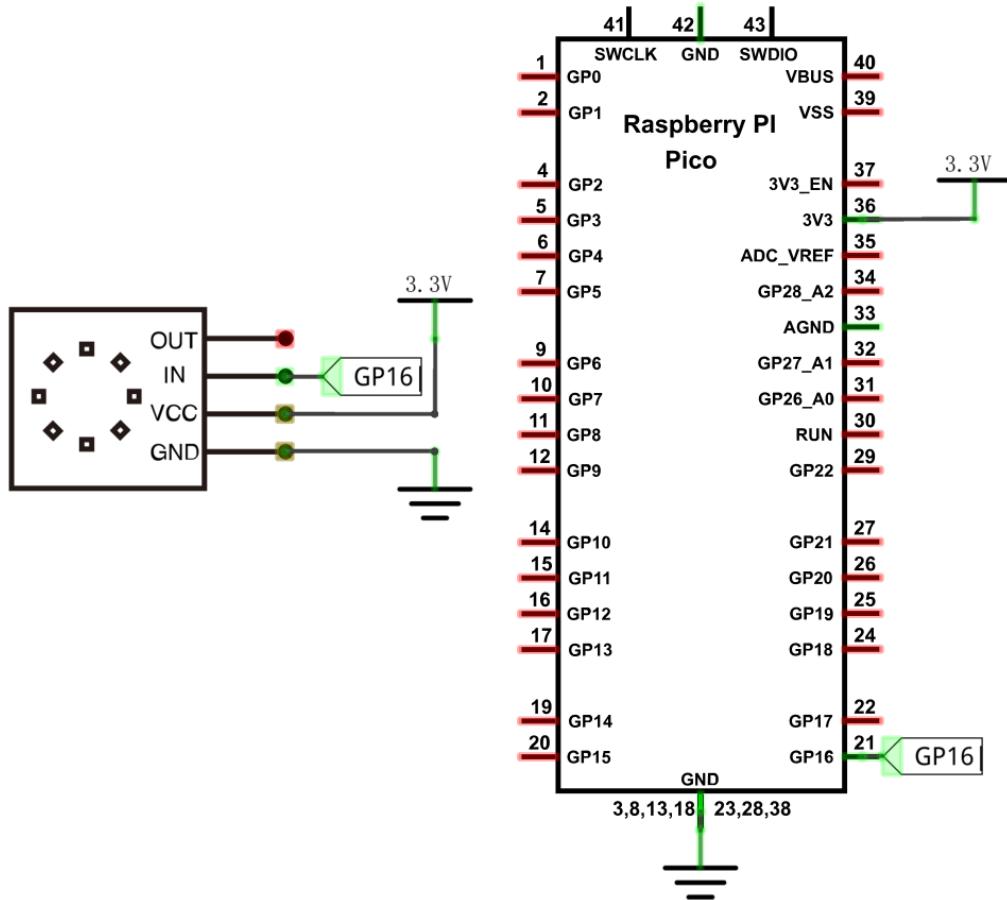


Pin description:

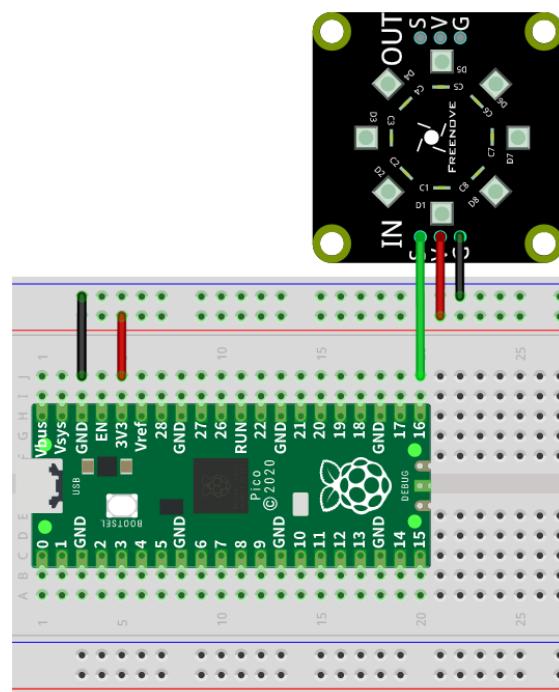
(IN)		(OUT)	
symbol	Function	symbol	Function
S	Input control signal	S	Output control signal
V	Power supply pin, +3.3V~5.5V	V	Power supply pin, +3.3V~5.5V
G	GND	G	GND

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

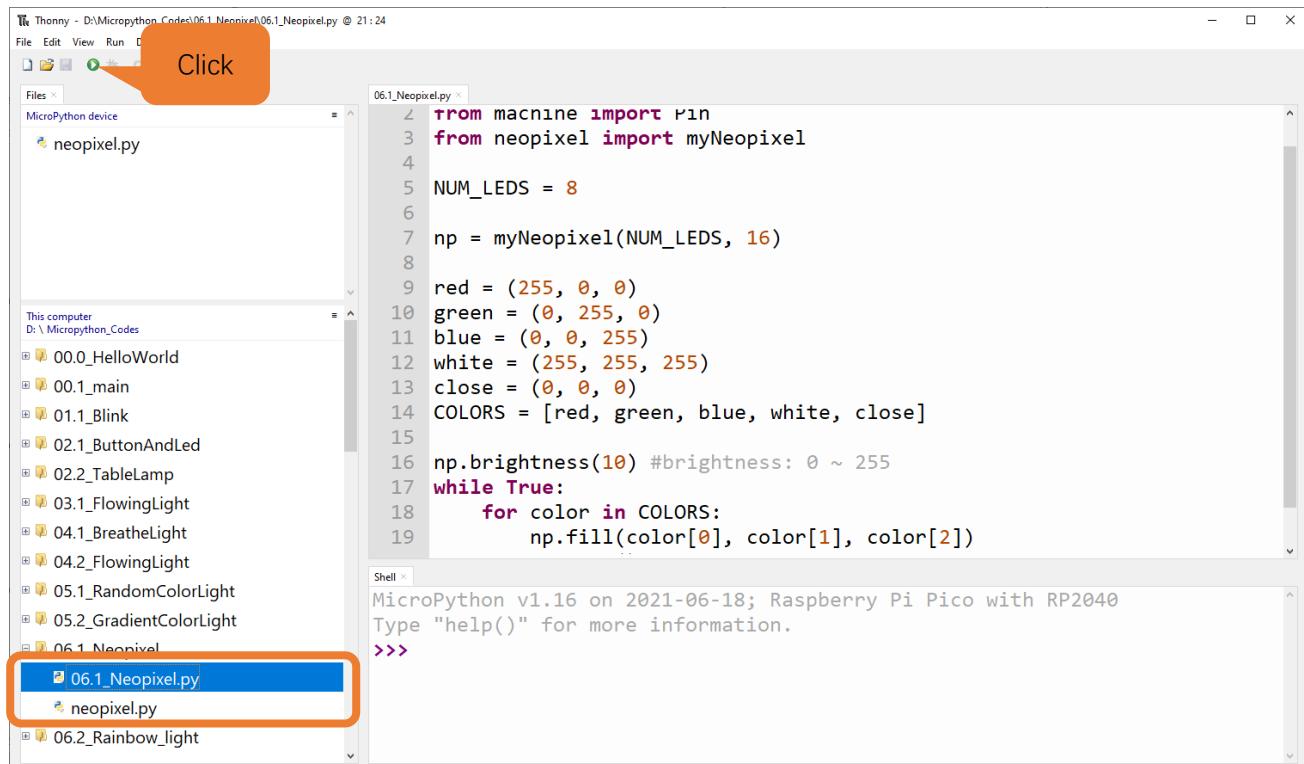


Any concerns? ✉ support@freenove.com

Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “06.1_Neopixel”. Select “neopixel.py”, right click to select “Upload to /”, wait for “neopixel.py” to be uploaded to Raspberry Pi Pico and double click “06.1_Neopixel.py”.

06.1_Neopixel

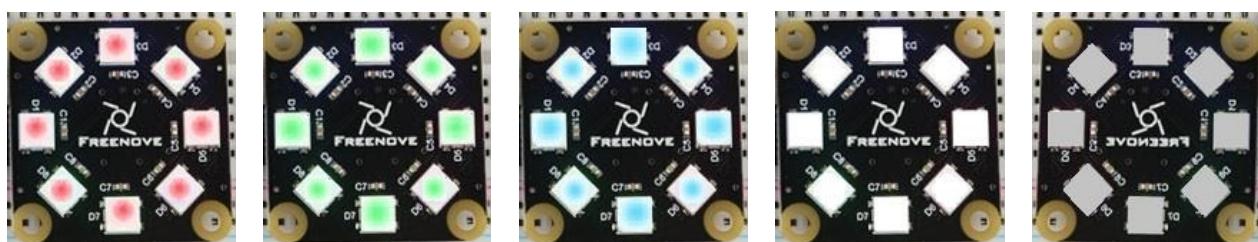


```

1 from machine import Pin
2 from neopixel import myNeopixel
3
4 NUM_LEDS = 8
5
6 np = myNeopixel(NUM_LEDS, 16)
7
8 red = (255, 0, 0)
9 green = (0, 255, 0)
10 blue = (0, 0, 255)
11 white = (255, 255, 255)
12 close = (0, 0, 0)
13 COLORS = [red, green, blue, white, close]
14
15 np.brightness(10) #brightness: 0 ~ 255
16 while True:
17     for color in COLORS:
18         np.fill(color[0], color[1], color[2])
    
```

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>

Click “Run current script”, and Neopixel begins to light up in red, green, blue, white and close. Press Ctrl+C or click “Stop/Restart backend” to end program.





The following is the program code:

```

1 import time
2 from machine import Pin
3 from neopixel import myNeopixel
4
5 NUM_LEDS = 8
6
7 np = myNeopixel(NUM_LEDS, 16)
8
9 red = (255, 0, 0)
10 green = (0, 255, 0)
11 blue = (0, 0, 255)
12 white = (255, 255, 255)
13 close = (0, 0, 0)
14 COLORS = [red, green, blue, white, close]
15
16 np.brightness(10) #brightness: 0 ~ 255
17 while True:
18     for color in COLORS:
19         np.fill(color[0], color[1], color[2])
20         np.show()
21         time.sleep(0.5)

```

Import Pin, neopixel and time modules.

```

1 import time
2 from machine import Pin
3 from neopixel import myNeopixel

```

Define the number of LEDs: NUM_LEDS = 8, create an object for myNeopixel and set GP16 pin to connect with neopixel.

```

5 NUM_LEDS = 8
6
7 np = myNeopixel(NUM_LEDS, 16)

```

Define 5 status for the LED, namely, red, green, blue, white and close and write them to COLORS list.

```

9 red = (255, 0, 0)
10 green = (0, 255, 0)
11 blue = (0, 0, 255)
12 white = (255, 255, 255)
13 close = (0, 0, 0)
14 COLORS = [red, green, blue, white, close]

```

Call brightness() function to set the brightness of LED, ranging from 0-255.

```

16 np.brightness(10) #brightness: 0 ~ 255

```

Call fill() function to set color for all LEDs at once. Only when show() function is called will the LEDs be lighted up to show the previously setting color.

```

19 np.fill(color[0], color[1], color[2])
20 np.show()

```

Any concerns? ✉ support@freenove.com

Define COLORS list through for loop to make neopixel module repeatedly emit red, green, blue, white and close, a total of five status.

```
17 while True:  
18     for color in COLORS:  
19         np.fill(color[0], color[1], color[2])  
20         np.show()  
21         time.sleep(0.5)
```

Reference

Class neopixel

Before each use of **neopixel** module, please add the statement “**import neopixel**” to the top of Python file.

myNeopixel(num_leds, pin): Define the number of output pins and LEDs of neopixel module

num_leds: The number of LEDs.

pin: Output pins.

myNeopixel.brightness(brightness): Set brightness for LEDs of neopixel module. Brightness range 0-255.

myNeopixel.set_pixel(pixel_num, r, g, b): Specify the color data of a single LED in the neopixel module.

pixel_num: neopixel: The sequence of LED in the module.

r: data of red channel.

g: data of green channel.

b: data of blue channel.

myNeopixel.fill (r, g, b): Set color data for all LEDs at once.

myNeopixel.show(): Make LED module show the setting color.

Project 6.2 Rainbow Light

In the previous project, we have mastered the usage of NeoPixel. This project will realize a slightly complicated Rainbow Light. The component list and the circuit are exactly the same as the project fashionable Light.

Code

Continue to use the following color model to equalize the color distribution of the 8 leds and gradually change.



Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “06.2_Rainbow_light”, **Select “neopixel.py”**, right click to select “Upload to /”, wait for “neopixel.py” to be uploaded to Raspberry Pi Pico and double click “06.2_Rainbow_light.py”.

06.2_Rainbow_light

```

from machine import Pin
from neopixel import myNeopixel
import time

NUM_LEDS = 8
np = myNeopixel(NUM_LEDS, 16)

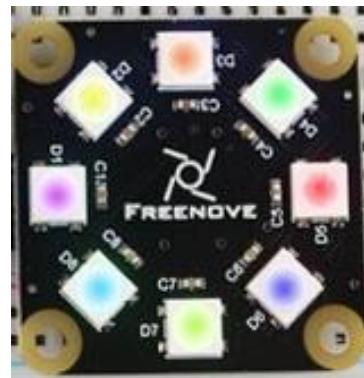
red = 0 #red
green = 0 #green
blue = 0 #blue

def wheel(pos):
    global red, green, blue
    WheelPos = pos % 255
    if WheelPos < 85:
        red = (255-WheelPos*3)
    elif WheelPos < 170:
        green = (WheelPos*3)
    else:
        blue = (255-WheelPos*3)
    return (red, green, blue)

```

The screenshot shows the Thonny IDE interface. The title bar says "Thonny - D:\Micropython_Codes\06.2_Rainbow_light\06.2_Rainbow_light.py @ 43:1". The main area has tabs for "Files" and "Shell". The "Files" tab shows a file tree with "06.2_Rainbow_light" selected. Inside "06.2_Rainbow_light", "06.2_Rainbow_light.py" is highlighted with an orange border. The "Shell" tab shows the MicroPython version and a prompt: "MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040 Type "help()" for more information. >>>". An orange callout bubble with the text "Click" points to the "Files" tab.

Click “Run current script”, and the Freenove 8 RGB LED Strip displays different colors and the color changes gradually. Press Ctrl+C or click “Stop/Restart backend” to exit program.



The following is the program code:

```
1  from machine import Pin
2  from neopixel import myNeopixel
3  import time
4
5  NUM_LEDS = 8
6  np = myNeopixel(NUM_LEDS, 16)
7
8  red = 0          #red
9  green = 0        #green
10 blue = 0         #blue
11
12 def wheel(pos):
13     global red, green, blue
14     WheelPos = pos % 255
15     if WheelPos < 85:
16         red = (255 - WheelPos*3)
17         green = (WheelPos*3)
18         blue=0
19     elif WheelPos >= 85 and WheelPos < 170:
20         WheelPos -= 85;
21         red = 0
22         green = (255 - WheelPos*3)
23         blue = (WheelPos*3)
24     else :
25         WheelPos -= 170;
26         red = (WheelPos*3)
27         green = 0
28         blue = (255 - WheelPos*3)
29
30 np.brightness(20)
```

```

31 while True:
32     for i in range(0, 255):
33         for j in range(0, 8):
34             wheel(i + j*255 // 8)
35             np.set_pixel(j, red, green, blue)
36             np.show()
37             time.sleep_ms(1)

```

Define a wheel() function to process the color data of neopixel module.

```

13 def wheel(pos):
14     global red, green, blue
15     WheelPos = pos % 255
16     if WheelPos < 85:
17         red = (255 - WheelPos*3)
18         green = (WheelPos*3)
19         blue=0
20     elif WheelPos >= 85 and WheelPos < 170:
21         WheelPos -= 85;
22         red = 0
23         green = (255 - WheelPos*3)
24         blue = (WheelPos*3)
25     else :
26         WheelPos -= 170;
27         red = (WheelPos*3)
28         green = 0
29         blue = (255 - WheelPos*3)

```

We use np.set_pixel(pixel_num, r, g, b) function to set data for each LED of neopixel separately, among which, parameter pixel_num presents the sequence of LED, parameters r, g, b indicate data of different color channels.

```
35     np.set_pixel(j, red, green, blue)
```

Use a nesting of two for loops. The first for loop makes the value of i increase from 0 to 255 automatically and the wheel() function processes the value of i into data of the module's three colors; the second for loop writes the color data to the module.

```

31 while True:
32     for i in range(0, 255):
33         for j in range(0, 8):
34             wheel(i + j*255 // 8)
35             np.set_pixel(j, red, green, blue)
36             np.show()
37             time.sleep_ms(1)

```

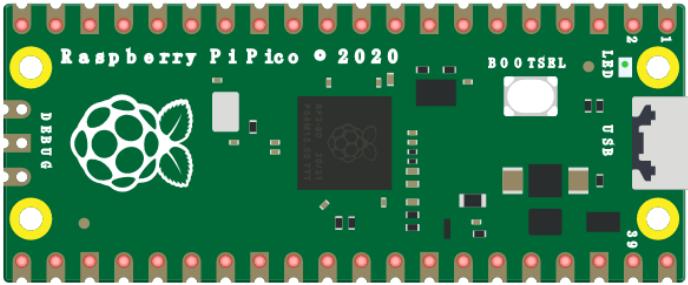
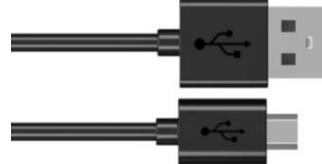
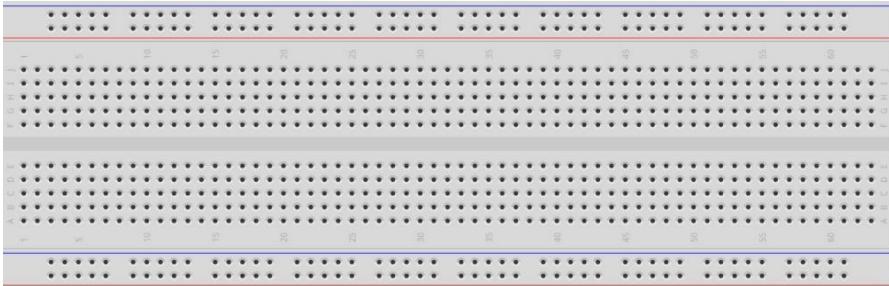
Chapter 7 Buzzer

In this chapter, we will learn about buzzers and the sounds they make.

Project 7.1 Doorbell

We will make this kind of doorbell: when the button is pressed, the buzzer sounds; and when the button is released, the buzzer stops sounding.

Component List

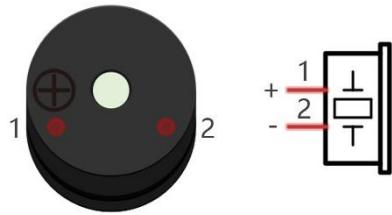
Raspberry Pi Pico x1		USB cable x1							
Breadboard x1									
Jumper									
NPN transistor x1 (S8050)		Active buzzer x1		Push button x1		Resistor 1kΩ x1		Resistor 10kΩ x2	

Component knowledge

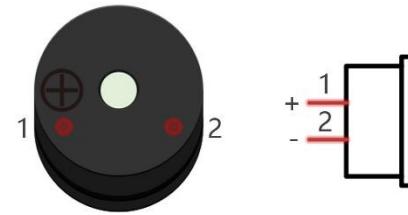
Buzzer

Buzzer is a sounding component, which is widely used in electronic devices such as calculator, electronic warning clock and alarm. Buzzer has two types: active and passive. Active buzzer has oscillator inside, which will sound as long as it is supplied with power. Passive buzzer requires external oscillator signal (generally use PWM with different frequency) to make a sound.

Active buzzer



Passive buzzer



Active buzzer is easy to use. Generally, it can only make a specific frequency of sound. Passive buzzer requires an external circuit to make a sound, but it can be controlled to make a sound with different frequency. The resonant frequency of the passive buzzer is 2kHz, which means the passive buzzer is loudest when its resonant frequency is 2kHz.

Next, we will use an active buzzer to make a doorbell and a passive buzzer to make an alarm.

How to identify active and passive buzzer?

1. Usually, there is a label on the surface of active buzzer covering the vocal hole, but this is not an absolute judgment method.
2. Active buzzers are more complex than passive buzzers in their manufacture. There are many circuits and crystal oscillator elements inside active buzzers; all of this is usually protected with a waterproof coating (and a housing) exposing only its pins from the underside. On the other hand, passive buzzers do not have protective coatings on their underside. From the pin holes viewing of a passive buzzer, you can see the circuit board, coils, and a permanent magnet (all or any combination of these components depending on the model).

Active buzzer



Passive buzzer



Transistor

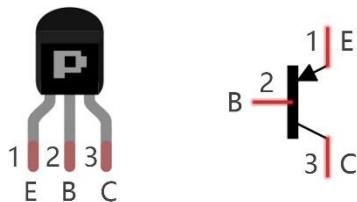
Because the buzzer requires such large current that GP of Raspberry Pi Pico output capability cannot meet the requirement, a transistor of NPN type is needed here to amplify the current.

Transistor, the full name: semiconductor transistor, is a semiconductor device that controls current. Transistor

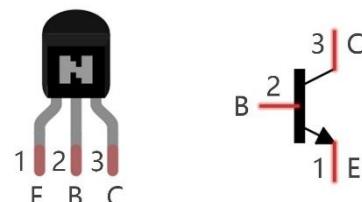
Any concerns? ✉ support@freenove.com

can be used to amplify weak signal, or works as a switch. It has three electrodes(PINs): base (b), collector (c) and emitter (e). When there is current passing between "be", "ce" will allow several-fold current (transistor magnification) pass, at this point, transistor works in the amplifying area. When current between "be" exceeds a certain value, "ce" will not allow current to increase any longer, at this point, transistor works in the saturation area. Transistor has two types as shown below: PNP and NPN.

PNP transistor



NPN transistor

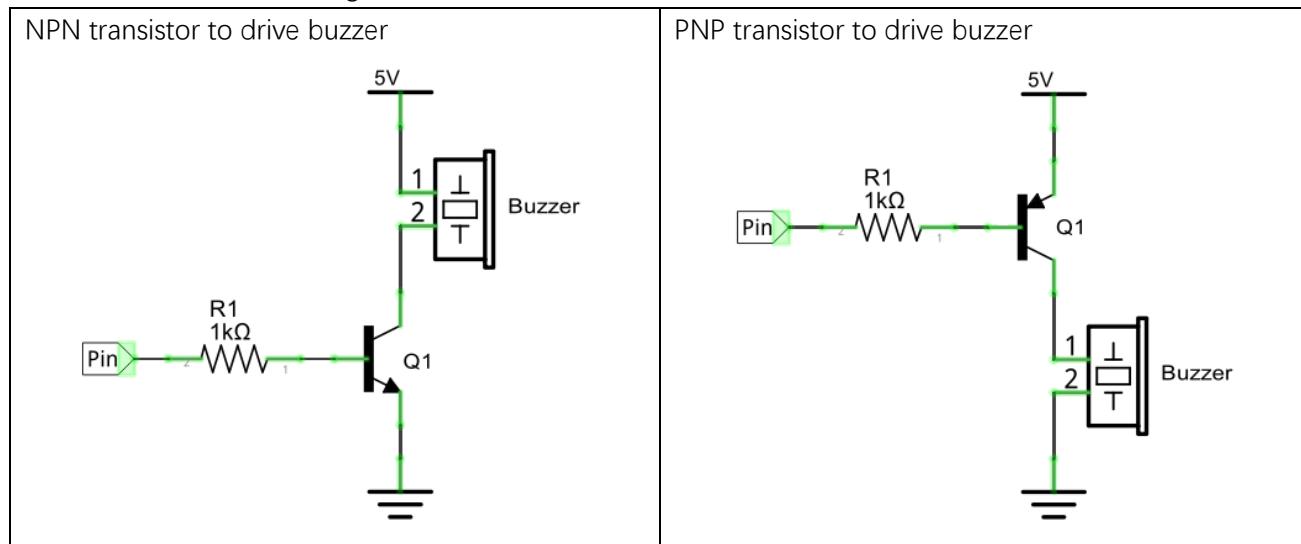


In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

Based on the transistor's characteristics, it is often used as a switch in digital circuits. As micro-controller's capacity to output current is very weak, we will use transistor to amplify current and drive large-current components.

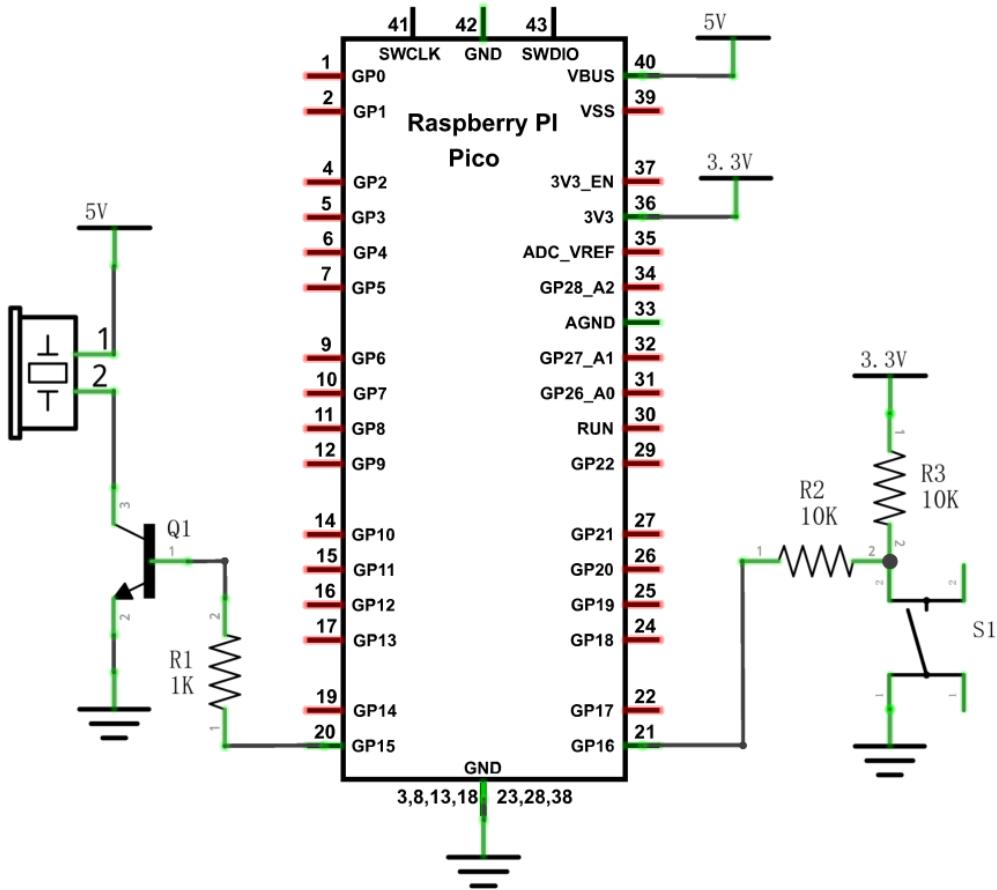
When using NPN transistor to drive buzzer, we often adopt the following method. If GP outputs high level, current will flow through R1, the transistor will get conducted, and the buzzer will sound. If GP outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer will not sound.

When using PNP transistor to drive buzzer, we often adopt the following method. If GP outputs low level, current will flow through R1, the transistor will get conducted, and the buzzer will sound. If GP outputs high level, no current flows through R1, the transistor will not be conducted, and buzzer will not sound.

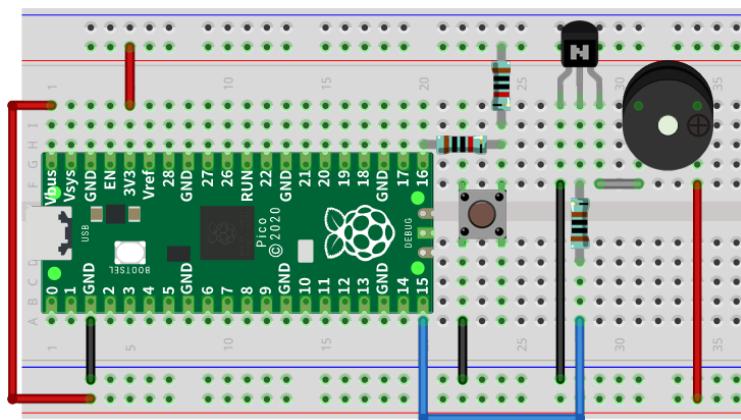


Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Note:

1. in this circuit, the power supply for buzzer is 5V, and pull-up resistor of the button connected to the power 3.3V. The buzzer can work when connected to power 3.3V, but it will reduce the loudness.
2. VBUS should be connect to the positive end of USB cable. If it connects to GND, it may burn the computer or Raspberry Pi Pico. Similarly, please be careful when wiring pins 36-40 of Pico to avoid short circuit.

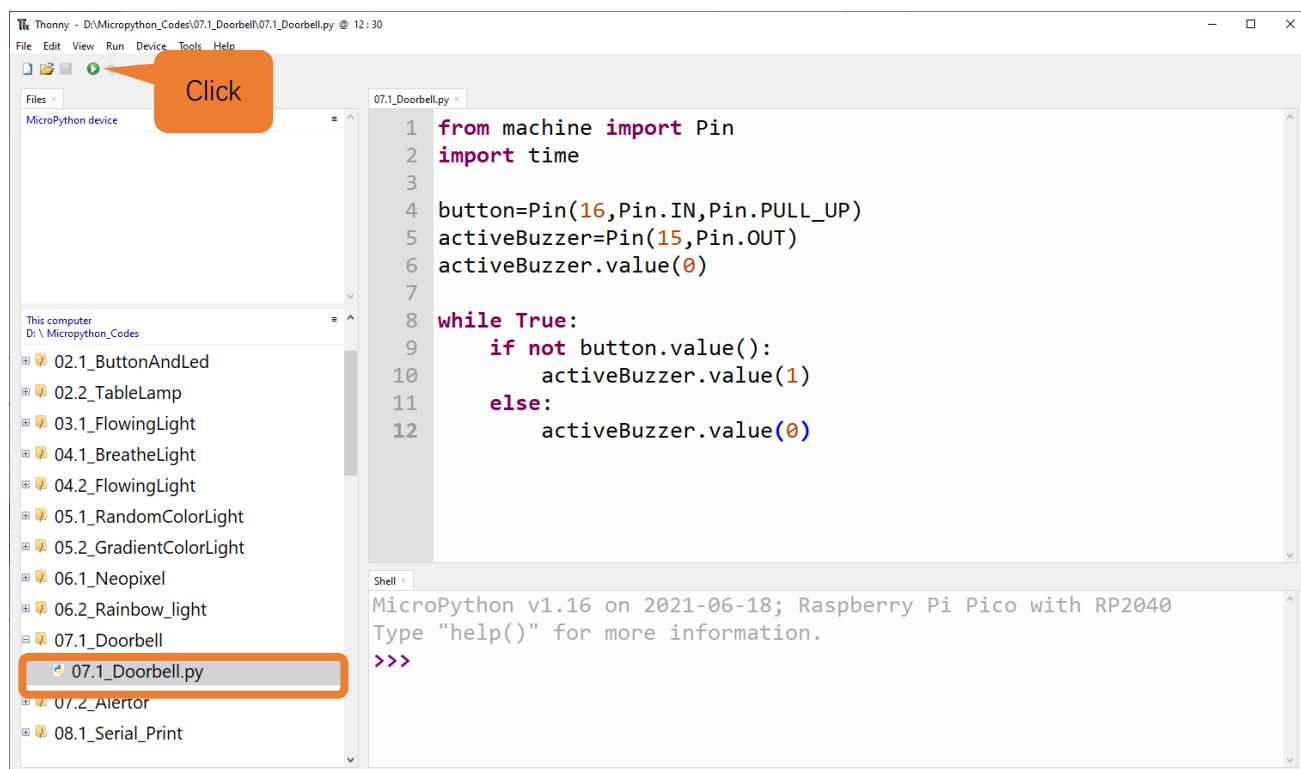
Any concerns? ✉ support@freenove.com

Code

In this project, a buzzer will be controlled by a push button switch. When the button switch is pressed, the buzzer sounds and when the button is released, the buzzer stops. It is analogous to our earlier project that controlled an LED ON and OFF.

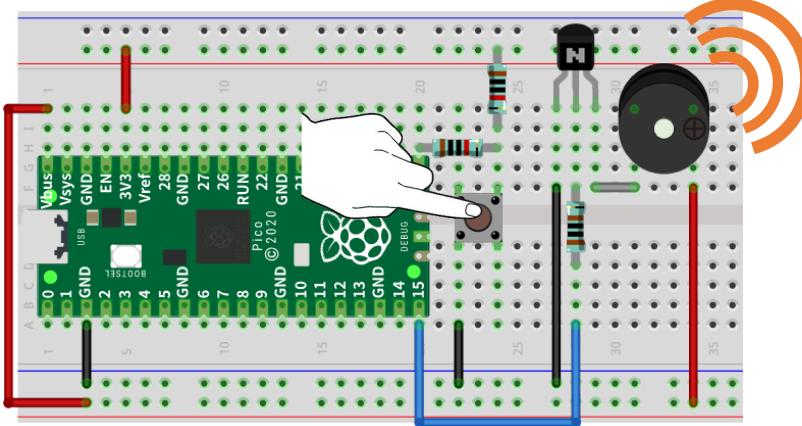
Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “07.1_Doorbell” and double click “07.1_Doorbell.py”.

07.1_Doorbell





Click “Run current script”, press the push button switch and the buzzer will sound. Release the push button switch and the buzzer will stop. Press Ctrl+C or click “Stop/Rerun backend” to exit program.



The following is the program code:

```

1 from machine import Pin
2 import time
3
4 button=Pin(16,Pin.IN,Pin.PULL_UP)
5 activeBuzzer=Pin(15,Pin.OUT)
6 activeBuzzer.value(0)
7
8 while True:
9     if not button.value():
10         activeBuzzer.value(1)
11     else:
12         activeBuzzer.value(0)

```

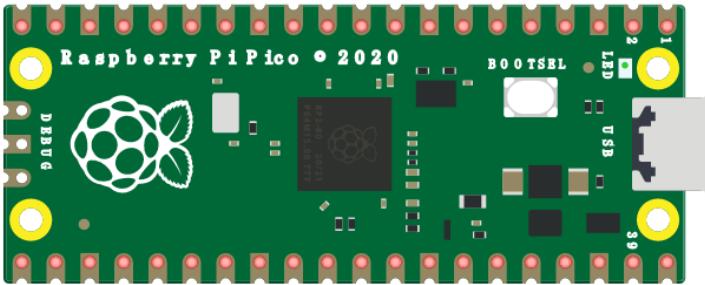
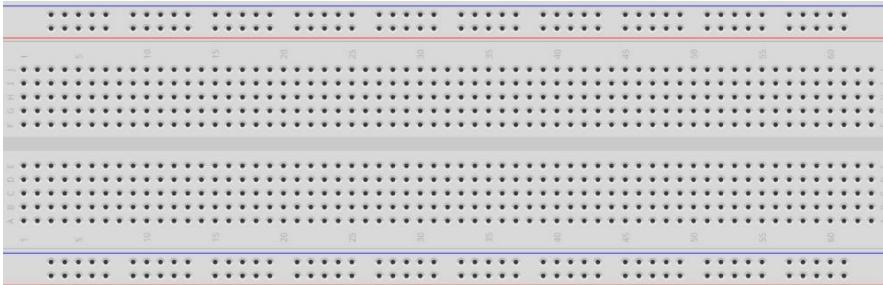
The code is logically the same as using button to control LED.

Project 7.2 Alertor

Next, we will use a passive buzzer to make an alarm.

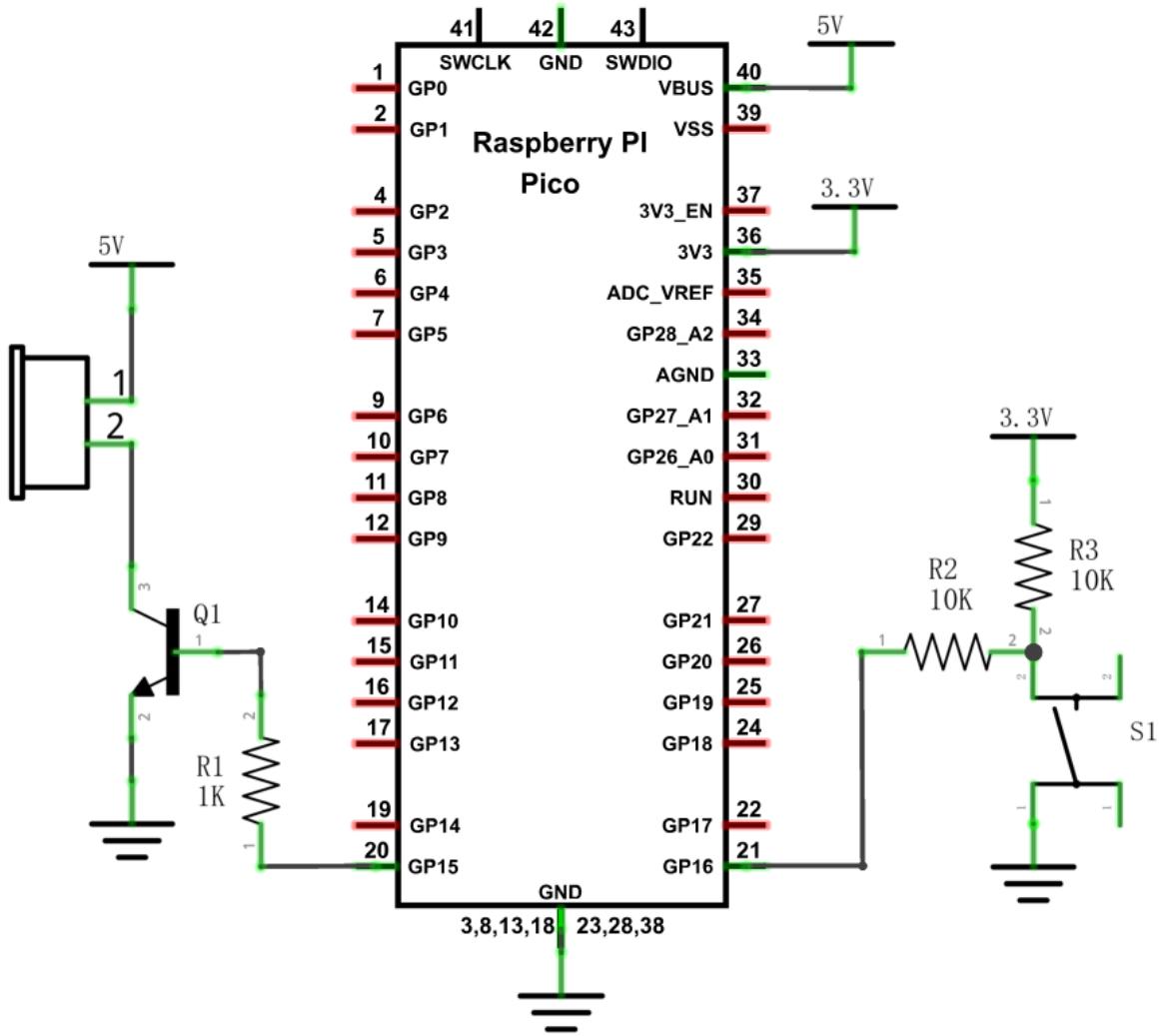
Component list and the circuit part is similar to last section. In the Doorbell circuit only the **active buzzer** needs to be **replaced** with a **passive buzzer**.

Component List

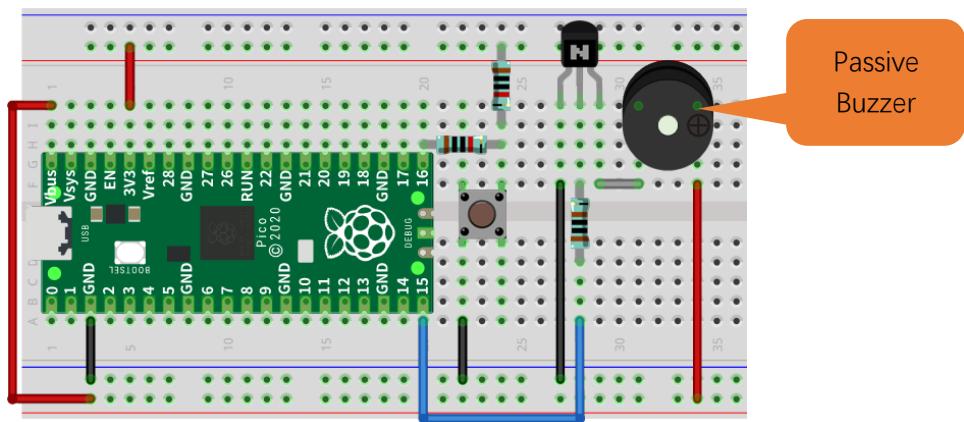
Raspberry Pi Pico x1	USB cable x1			
				
Breadboard x1				
Jumper				
NPN transistorx1 (S8050)	Passive buzzer x1	Push button x1	Resistor 1kΩ x1	Resistor 10kΩ x2
				

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

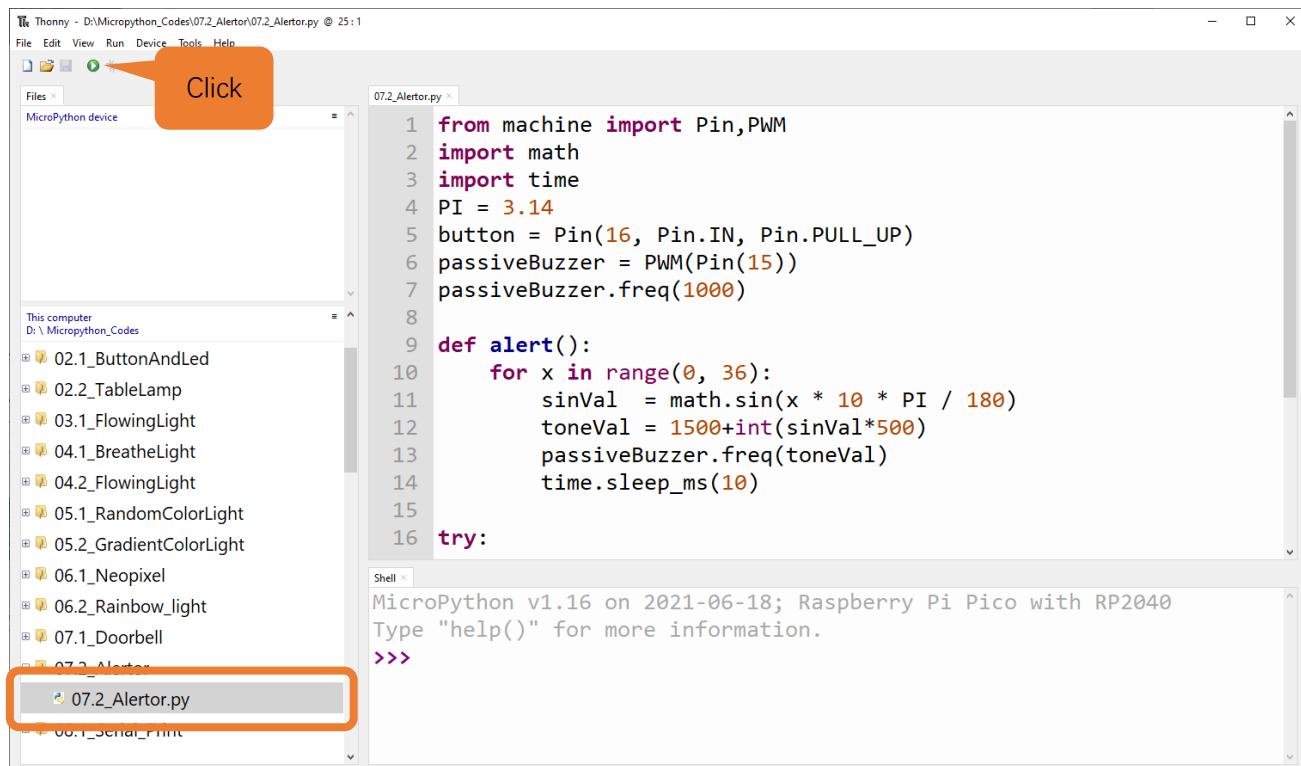


Code

In this project, the buzzer alarm is controlled by the button. Press the button, then buzzer sounds. If you release the button, the buzzer will stop sounding. Logically, it is the same as using button to control LED. As to the control method, passive buzzer requires PWM of certain frequency to sound.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “07.2_Alertor”, and double click “07.2_Alertor.py”.

07.2_Alertor



```

from machine import Pin,PWM
import math
import time
PI = 3.14
button = Pin(16, Pin.IN, Pin.PULL_UP)
passiveBuzzer = PWM(Pin(15))
passiveBuzzer.freq(1000)

def alert():
    for x in range(0, 36):
        sinVal = math.sin(x * 10 * PI / 180)
        toneVal = 1500+int(sinVal*500)
        passiveBuzzer.freq(toneVal)
        time.sleep_ms(10)

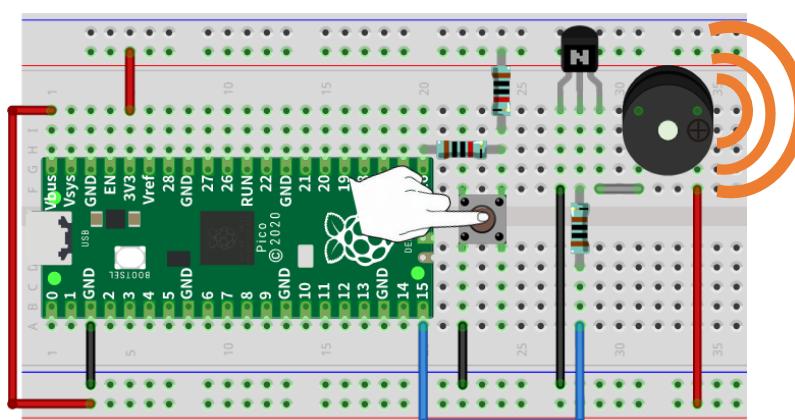
try:
    while True:
        if not button.value():
            alert()
        else:
            passiveBuzzer.deinit()
            break

```

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>

Click “Run current script”, press the button, and the alarm sounds. And when the button is released, the alarm will stop sounding. Press Ctrl+C or click “Stop/Restart backend” to exit program.

If the buzzer sound is too loud or too low for you, you can modify duty cycle or frequency of PWM.



The following is the program code:

```

1  from machine import Pin, PWM
2  import math
3  import time
4  PI = 3.14
5  button = Pin(16, Pin.IN, Pin.PULL_UP)
6  passiveBuzzer = PWM(Pin(15))
7  passiveBuzzer.freq(1000)
8
9  def alert():
10     for x in range(0, 36):
11         sinVal = math.sin(x * 10 * PI / 180)
12         toneVal = 1500 + int(sinVal*500)
13         passiveBuzzer.freq(toneVal)
14         time.sleep_ms(10)
15
16     try:
17         while True:
18             if not button.value():
19                 passiveBuzzer.duty_u16(4092*2)
20                 alert()
21             else:
22                 passiveBuzzer.duty_u16(0)
23     except:
24         passiveBuzzer.deinit()
```

Import PWM, Pin, math and time modules.

```

1  from machine import Pin, PWM
2  import math
3  import time
```

Define the pins of the button and passive buzzer.

```

4  PI = 3.14
5  button = Pin(16, Pin.IN, Pin.PULL_UP)
6  passiveBuzzer = PWM(Pin(15))
7  passiveBuzzer.freq(1000)
```

Call sin function of math module to create the frequency data of the passive buzzer.

```

9  def alert():
10     for x in range(0, 36):
11         sinVal = math.sin(x * 10 * PI / 180)
12         toneVal = 1500 + int(sinVal*500)
13         passiveBuzzer.freq(toneVal)
14         time.sleep_ms(10)
```

When not using PWM, please turn it OFF in time.

```
24  passiveBuzzer.deinit()
```

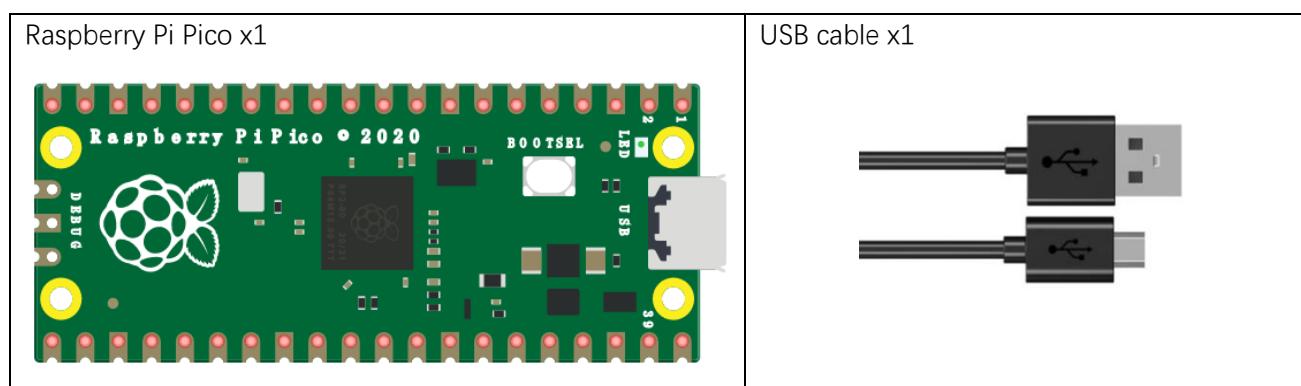
Chapter 8 Serial Communication

Serial Communication is a means of Communication between different devices. This section describes Raspberry Pi Pico Serial Communication.

Project 8.1 Serial Print

This project uses Raspberry Pi Pico serial communicator to send data to the computer and print it on the serial monitor.

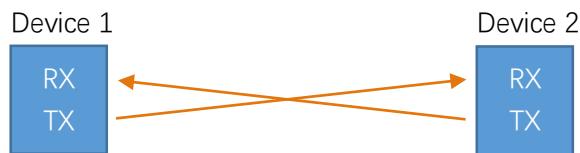
Component List



Related knowledge

Serial communication

Serial communication generally refers to the Universal Asynchronous Receiver/Transmitter (UART), which is commonly used in electronic circuit communication. It has two communication lines, one is responsible for sending data (TX line) and the other for receiving data (RX line). The serial communication connections two devices use is as follows:



Before serial communication starts, the baud rate of both sides must be the same. Communication between devices can work only if the same baud rate is used. The baud rates commonly used is 9600 and 115200.

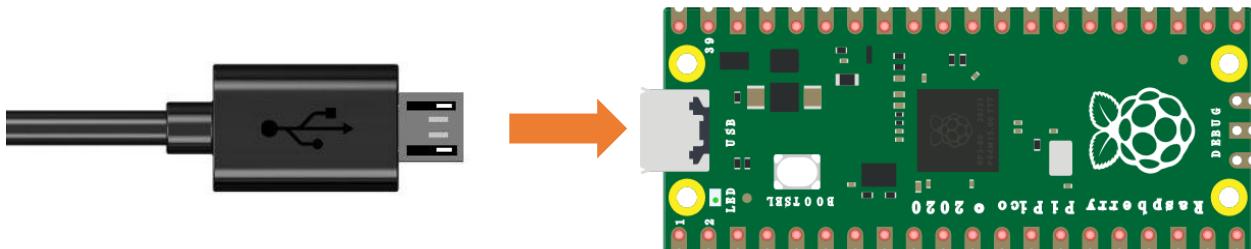
Serial port on Raspberry Pi Pico

Raspberry Pi Pico has integrated USB to serial transfer, so it could communicate with computer connecting to USB cable.



Circuit

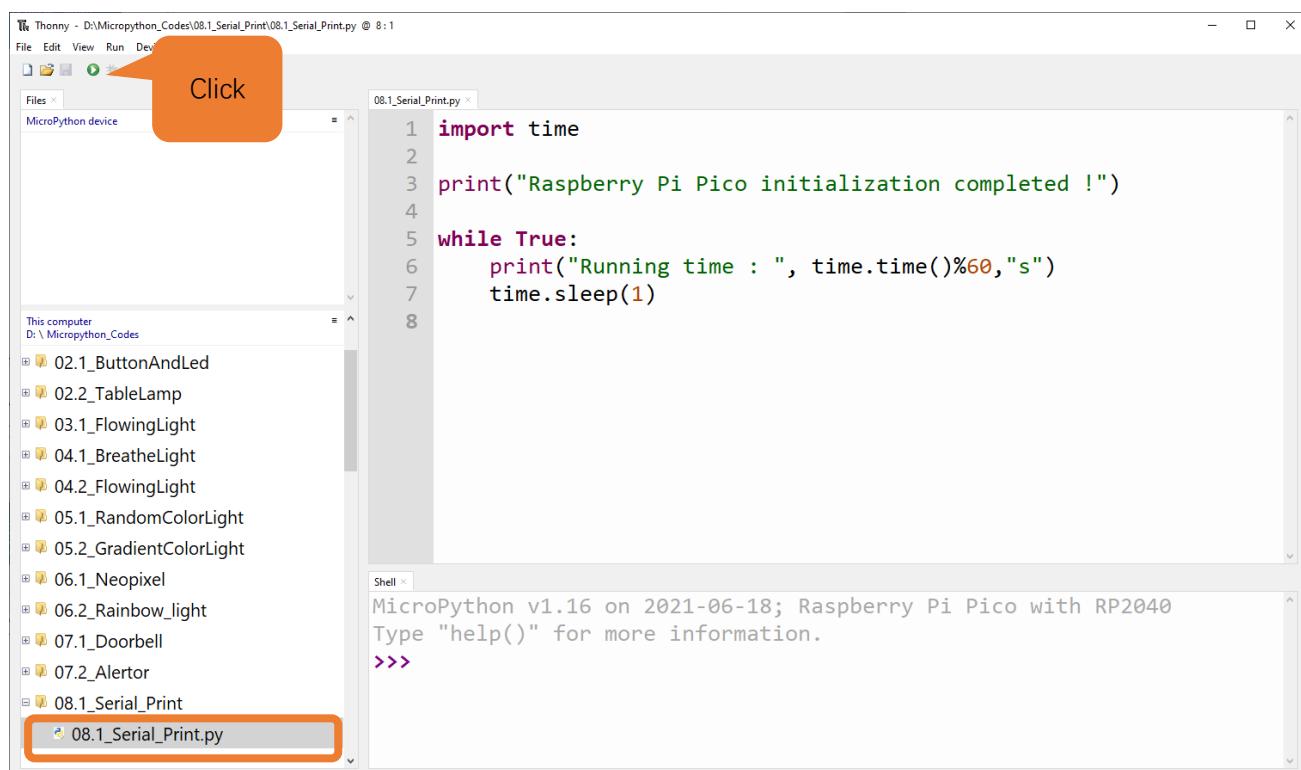
Connect Raspberry Pi Pico to the computer with USB cable.



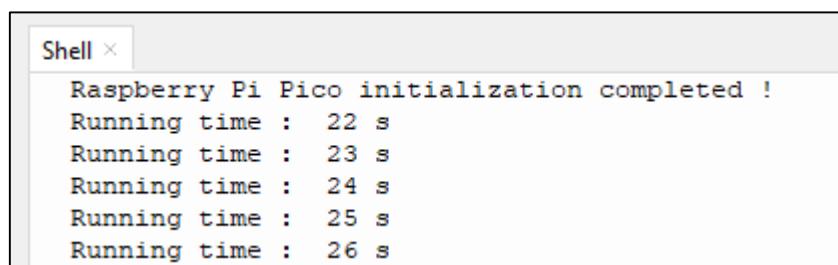
Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “08.1_Serial_Print” and double “08.1_Serial_Print.py”.

08.1_Serial_Print



Click “Run current script” and observe the changes of “Shell”, which will display the time when Raspberry Pi Pico is powered on once per second.



The following is the program code:

1	import time
2	
3	print("Raspberry Pi Pico initialization completed !")
4	
5	while True:
6	print("Running time : ", time.time()%60,"s")
7	time.sleep(1)



There are two serial communications on Raspberry Pi Pico: UART0 and UART1.

Default pin for UART0

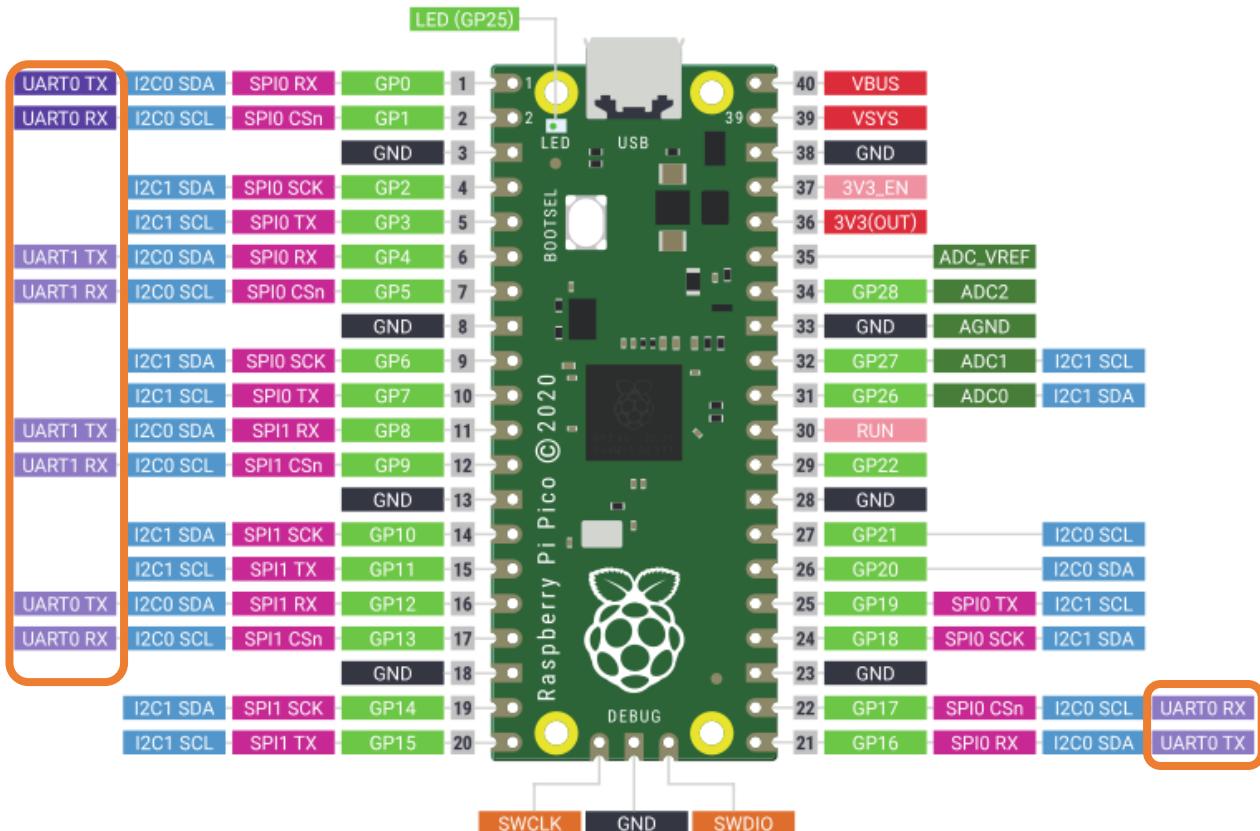
UART0_TX	Pin 0
UART0_RX	Pin 1

Default pin for UART1

UART1_TX	Pin 4
UART1_RX	Pin 5

For details, please refer to [UART, I2C, SPI default pin](#).

And you can also change settings according to the distribution of pins.



Reference

Class UART

Before each use of **UART** module, please add the statement “**from machine import UART**” to the top of python file.

UART(id, baudrate, bits, parity, rx, tx, stop, timeout): Define serial ports and configure parameters for them.

id: Serial Number. The available serial port number is 0 or 1.

baudrate: Baud rate.

bits: The number of each character.

parity: Check even or odd, with 0 for even checking and 1 for odd checking.

rx, tx: UAPT's reading and writing pins, GP0, GP1, GP4, GP5, GP8, GP9, GP12, GP13, GP16, GP17.

stop: The number of stop bits, and the stop bit is 1 or 2.

timeout: timeout period (Unit: millisecond).

$0 < \text{timeout} \leq 0x7FFF FFFF$ (decimal: $0 < \text{timeout} \leq 2147483647$).

UART.read(nbytes): Read nbytes bytes.

UART.read(): Read data.

UART.write(buf): Write byte buffer to UART bus.

UART.readline(): Read a line of data, ending with a newline character.

UART.readinto(buf): Read and write data into buffer.

UART.readinto(buf, nbytes): Read and write data into buffer.

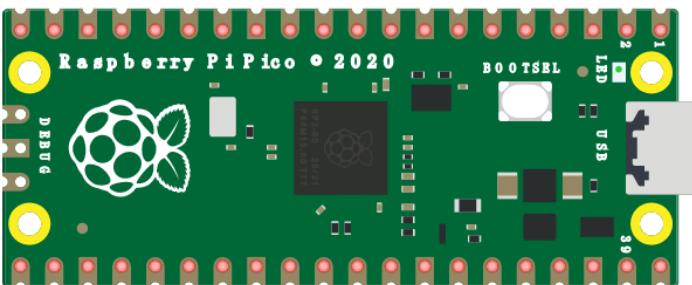
UART.any(): Determine whether there is data in serial ports. If there is, return the number of bytes; Otherwise, return 0.

Project 8.2 Serial Read and Write

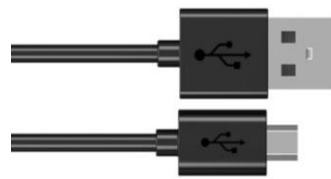
In the following example, we use Raspberry Pi Pico's UART1 to send data to UART0, and read the data received by UART0 and print it out through the "Shell".

Component List

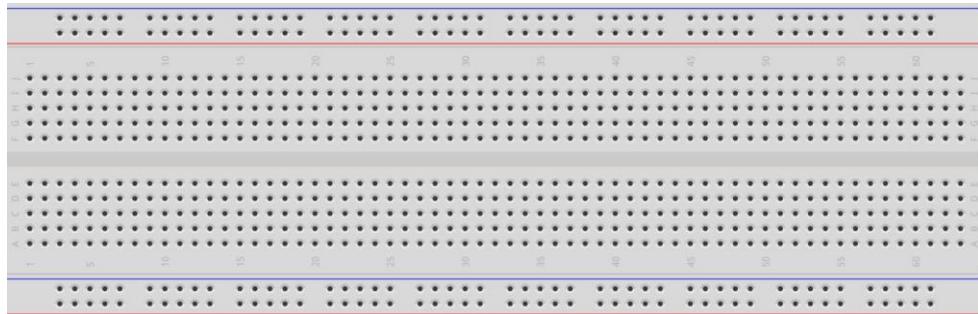
Raspberry Pi Pico x1



USB cable x1



Breadboard x1

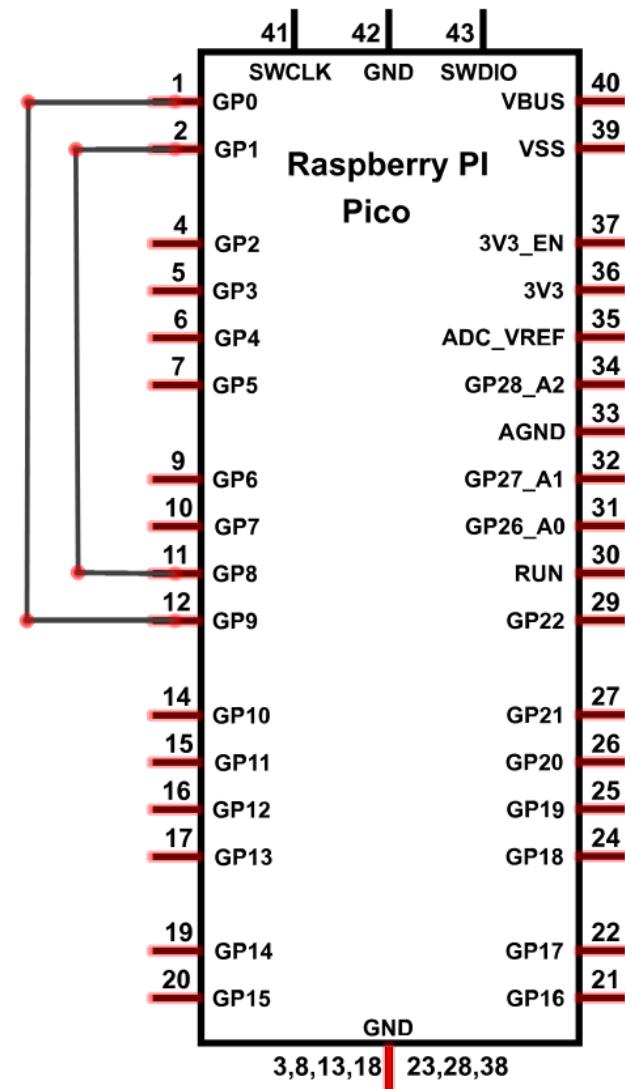


Jumper

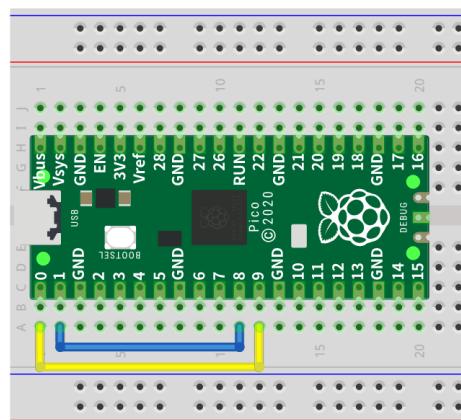


Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “08.2_Serial_Read_and_Write” and double click “08.2_Serial_Read_and_Write_UART1_to_UART0.py”.

08.2_Serial_Read_and_Write_UART1_to_UART0

The screenshot shows the Thonny IDE interface. On the left, the file tree displays a folder structure under "This computer" and "D:\Micropython_Codes". A red box highlights the file "08.2_Serial_Read_and_Write_UART1_to_UART0.py" in the "08.2_Serial_Read_and_Write" folder. An orange callout bubble labeled "Click" points to the file icon in the tree. The main window shows the Python code for the script:

```

from machine import UART, Pin
import time

myUsart0 = UART(0, baudrate=9600, bits=8, tx=Pin(0), rx=Pin(1), timeout=10)
myUsart1 = UART(1, baudrate=9600, bits=8, tx=Pin(8), rx=Pin(9), timeout=10)

while True:
    rxData = bytes()
    input_cnt = str(input("myUsart1: "))
    myUsart1.write(input_cnt)
    time.sleep(0.1)
    while myUsart0.any() > 0:
        rxData += myUsart0.read(1)
    print("myUsart0: ", rxData.decode('utf-8'))

```

The bottom right pane is the "Shell" window, which outputs:

```

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>

```

Click “Run current script”. Users can enter any data in “Shell” and press Enter. The input data will be written to UART1 and sent to UART0 to read, and printed out on “Shell”. Press “Ctrl+C” or click “Stop/Restart backend” to exit the program.

The screenshot shows the Thonny Shell window with the following interaction:

```

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_C...
myUsart1: 4252
myUsart0: 4252
myUsart1: 415
myUsart0: 415
myUsart1: 5275275
myUsart0: 5275275
myUsart1: 5757857
myUsart0: 5757857
myUsart1: 752752752
myUsart0: 752752752
myUsart1: 444asdfasd
myUsart0: 444asdfasd
myUsart1: dfadfa
myUsart0: dfadfa
myUsart1:

```

A red callout bubble labeled "Data that users input." points to the first line of input ("myUsart1: 4252"). Another red callout bubble labeled "Data read by UART0" points to the second line of output ("myUsart0: 4252").

The following is the program code:

```

1  from machine import UART, Pin
2  import time
3
4  myUsart0 = UART(0, baudrate=9600, bits=8, tx=Pin(0), rx=Pin(1), timeout=10)
5  myUsart1 = UART(1, baudrate=9600, bits=8, tx=Pin(8), rx=Pin(9), timeout=10)
6
7  while True:
8      rxData = bytes()
9      input_cnt = str(input("myUsart1: "))
10     myUsart1.write(input_cnt)
11     time.sleep(0.1)
12     while myUsart0.any() > 0:
13         rxData += myUsart0.read(1)
14     print("myUsart0: " , rxData.decode('utf-8'))

```

Import UART、Pin and time modules.

```

1  from machine import UART, Pin
2  import time

```

Create two UART objects and configure them as the parameters of UART0 and UART1.

```

4  myUsart0 = UART(0, baudrate=9600, bits=8, tx=Pin(0), rx=Pin(1), timeout=10)
5  myUsart1 = UART(1, baudrate=9600, bits=8, tx=Pin(8), rx=Pin(9), timeout=10)

```

Define a bytes value and assign it to rxDate.

```
8     rxData = bytes()
```

Define input_cnt to receive user input and convert it to a string format.

```
9     input_cnt = str(input("myUsart1: "))
```

myUsart1 calls write() function and writes the user input to UART1.

```
10    myUsart1.write(input_cnt)
```

myUsart0 calls the read() function to read the data sent by UART1 bit by bit and save the rxData in the received variable. When myUsart0 calls any() to determine whether UART0 has read the data, when any() returns 0, UART0 has read the data sent by UART1.

```

12     while myUsart0.any() > 0:
13         rxData += myUsart0.read(1)

```

The decode() function is called to decode the data and print it out to "Shell."

```
14     print("myUsart0: " , rxData.decode('utf-8'))
```



Chapter 9 AD Converter

This chapter we learn to use the ADC function of Rasepberry Pi Pico.

Project 9.1 Read the Voltage of Potentiometer

In this chapter, we use ADC function of Pico to read the voltage output by potentiometer.

Component List

Raspberry Pi Pico x1	A green printed circuit board (PCB) with a central Broadcom SoC chip. It features several pins along the top and bottom edges, and various components like resistors, capacitors, and a crystal oscillator.	USB cable x1	Two standard black USB-A to USB-B cables, each consisting of a male connector on one end and a female connector on the other.
Breadboard x1			A schematic diagram of a breadboard, showing its grid of 40 columns and 24 rows of connection points. The columns are labeled with letters A through P at the top, and the rows are numbered 1 through 24 on the left side.
Rotary potentiometer x1	A physical component with a brown plastic housing and three metal pins. It has a central shaft with a metal wiper that can slide along a resistive track, allowing it to provide a variable analog signal.	Jumper	A short, thin black wire with two small black plastic caps on the ends, used for connecting components.

Related knowledge

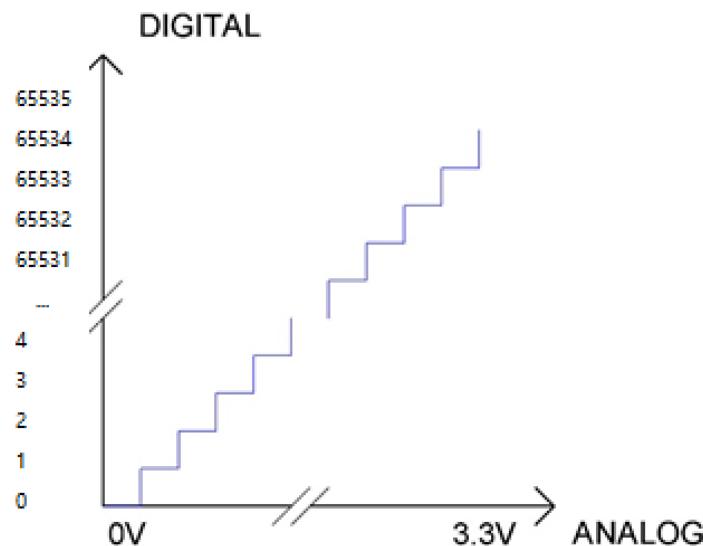
ADC

An analog-to-digital converter(ADC) converts a measured analog signal into a digital code. ADC has two key features: resolution and channels.

ADC resolution of Raspberry Pi Pico

Raspberry Pi Pico uses RP2040 chip. With a 12-bit ADC resolution, it can convert any analog signal to digital signal, ranging from 0-4095. For example, if the analog voltage range it measures is 0-3.3V, the ADC can divide it into 4096 equal parts.

However, when using Micropython firmware to call Raspberry Pi Pico ADC, the digital signal it obtains ranges from 0 to 65535. This is because Micropython internally processes Pico's ADC resolution to 16 bits, and the values is also changed to 0-65535, so as to make it the same as the ADC of other Micropython microcontrollers. The way that ADC converts doesn't change but only the resolution changes. So if the measured analog voltage ranges from 0-3.3V, ADC can devide it into 65536 equal parts.



Subsection 1: the analog in a range of 0V---3.3/65535 V corresponds to digital 0;

Subsection 2: the analog in a range of 3.3/65535 V---2*3.3 /65535 V corresponds to digital 1;

...

Subsection 65535: the analog in range of 65534*3.3/65535 V---65535*3.3 /65535 V corresponds to digital 65534;

The following analog will be divided accordingly.

The conversion formula is as follows:

$$ADCValue = \frac{\text{Analog Voltage}}{3.3} * 65535$$

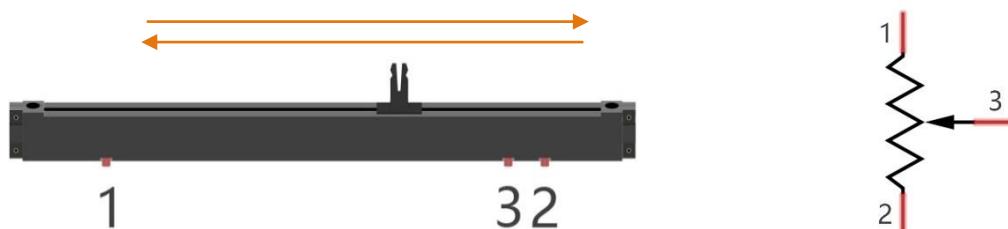
ADC ChannelsRaspberry Pi Pico

Raspberry Pi Pico has 5 ADC channels, which are ADC0(GP26), ADC1(GP27), ADC2(GP28), ADC3(GP29): used to measure VSYS on Pico board, and ADC4, which directly connects to the built-in temperature sensor of RP2040 chip. Therefore, there are only three generic ADC channels that can be directly used, namely, ADC0, ADC1 and ADC2.

Component knowledge

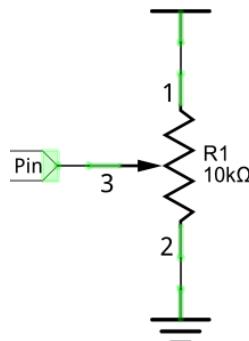
Potentiometer

Potentiometer is a resistive element with three Terminal parts. Unlike the resistors that we have used thus far in our project which have a fixed resistance value, the resistance value of a potentiometer can be adjusted. A potentiometer is often made up by a resistive substance (a wire or carbon element) and movable contact brush. When the brush moves along the resistor element, there will be a change in the resistance of the potentiometer's output side (3) (or change in the voltage of the circuit that is a part). The illustration below represents a linear sliding potentiometer and its electronic symbol on the right.



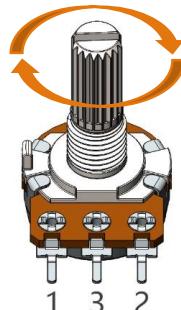
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly (until it reaches the highest value of the resistive element) and at the same time the resistance between pin 2 and pin 3 will decrease linearly and conversely down to zero. At the midpoint of the slider the measured resistance values between pin 1 and 3 and between pin 2 and 3 will be the same.

In a circuit, both sides of resistive element are often connected to the positive and negative electrodes of power. When you slide the brush "pin 3", you can get variable voltage within the range of the power supply.



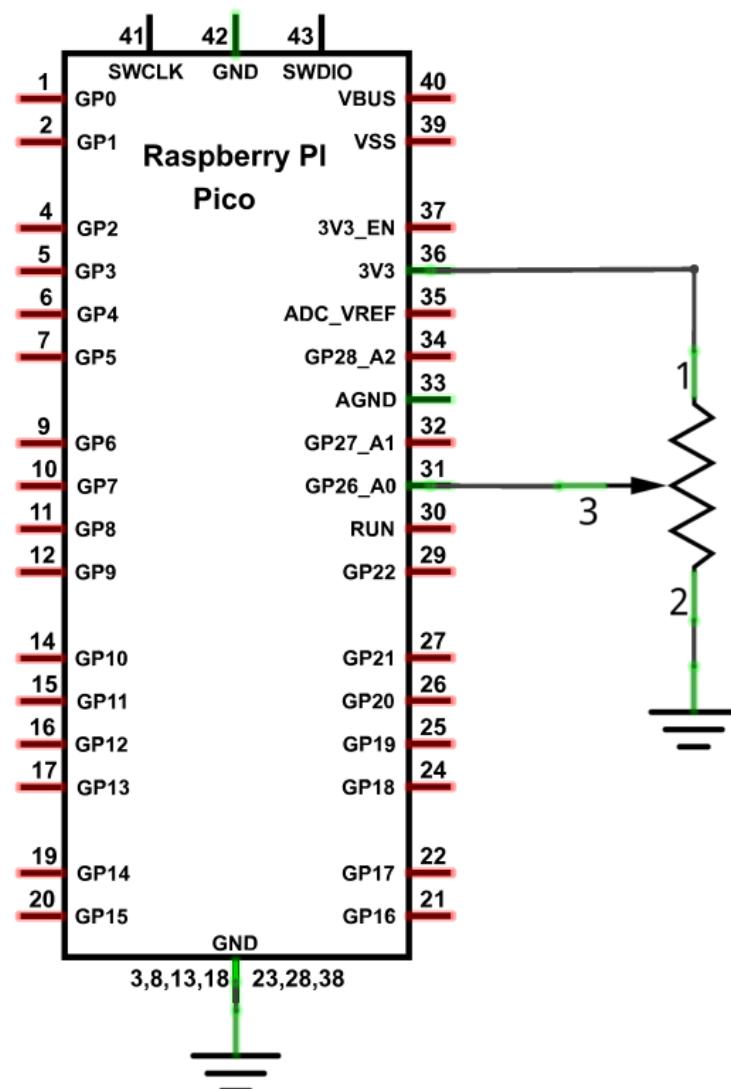
Rotary potentiometer

Rotary potentiometers and linear potentiometers have the same function; the only difference being the physical action being a rotational rather than a sliding movement.

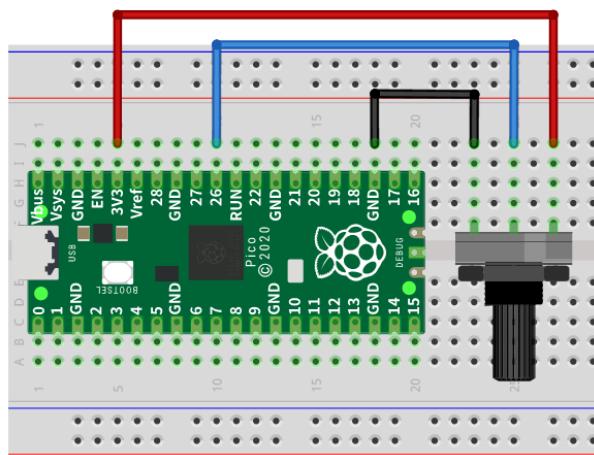


Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

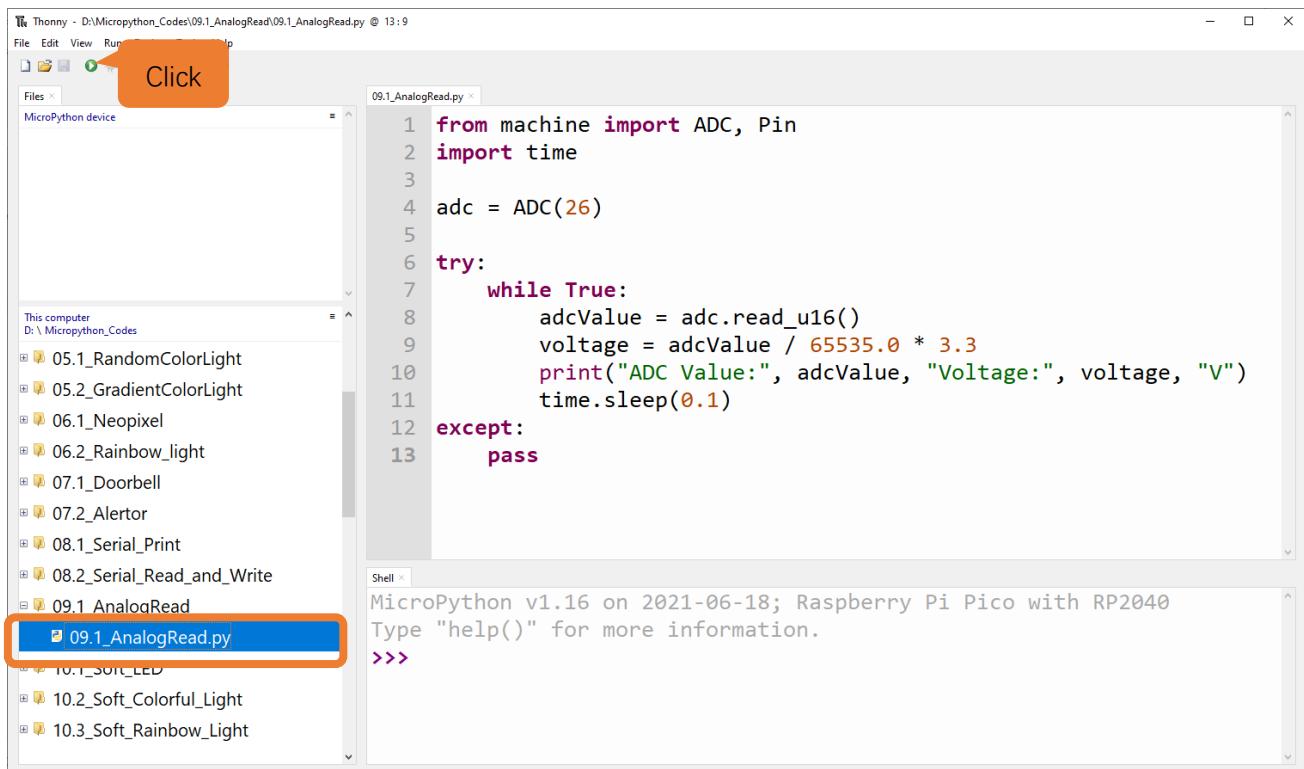


Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “09.1_AnalogRead” and then click “09.1_AnalogRead.py”.

09.1_AnalogRead



```

from machine import ADC, Pin
import time

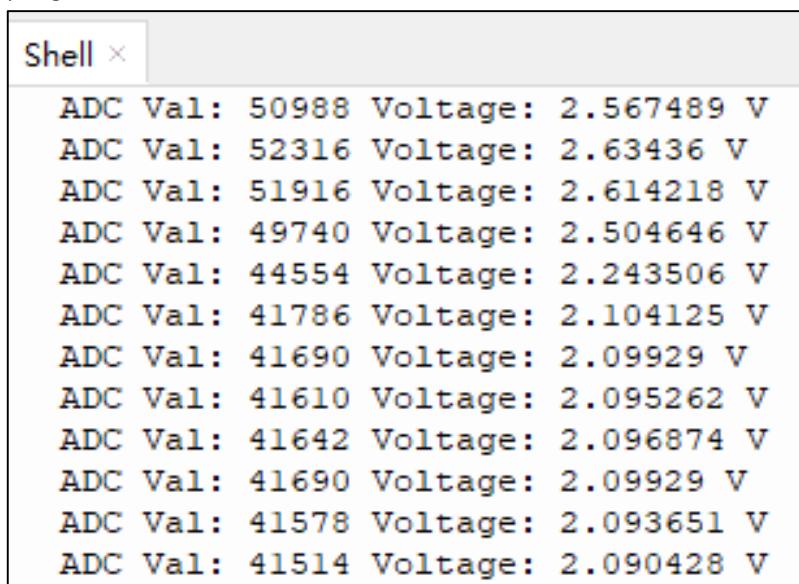
adc = ADC(26)

try:
    while True:
        adcValue = adc.read_u16()
        voltage = adcValue / 65535.0 * 3.3
        print("ADC Value:", adcValue, "Voltage:", voltage, "V")
        time.sleep(0.1)
except:
    pass

```

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>

Click “Run current script” and observe the message printed in “Shell”. Press Ctrl+C or click “Stop/Restart backend” to exit the program.



ADC Val	Voltage	V
50988	2.567489	V
52316	2.63436	V
51916	2.614218	V
49740	2.504646	V
44554	2.243506	V
41786	2.104125	V
41690	2.09929	V
41610	2.095262	V
41642	2.096874	V
41690	2.09929	V
41578	2.093651	V
41514	2.090428	V

The following is the code:

```

1  from machine import ADC, Pin
2  import time
3
4  adc = ADC(26)
5
6  try:
7      while True:
8          adcValue = adc.read_u16()
9          voltage = adcValue / 65535.0 * 3.3
10         print("ADC Value:", adcValue, "Voltage:", voltage, "V")
11         time.sleep(0.1)
12     except:
13         pass

```

Import Pin, ADC and time modules.

```

1  from machine import ADC, Pin
2  import time

```

Create an ADC object and connect GP26, which corresponds to ADC0 channel of Raspberry Pi Pico.

```
4  adc = ADC(26)
```

Read ADC value every 0.1 second. Calculate the current voltage based on the formula $ADCValue = (Analog\ Voltage)/3.3*65535$ and print it to "Shell".

```

7  while True:
8      adcValue = adc.read_u16()
9      voltage = adcValue / 65535.0 * 3.3
10     print("ADC Value:", adcValue, "Voltage:", voltage, "V")
11     time.sleep(0.1)

```

Reference

Class ADC

Before each use of ADC module, please add the statement “**from machine import ADC**” to the top of the python file.

machine.ADC(pin or channel_num): Create an ADC object associated with the given pin.

pin: Available pins are:GP26,GP27,GP28,GP29.

channel_num: Available channel 0, 1, 2, 3, 4.

For example:

```

machine.ADC(0) = machine.ADC(26)
machine.ADC(1) = machine.ADC(27)
machine.ADC(2) = machine.ADC(28)
machine.ADC(3) = machine.ADC(29)
machine.ADC(4) Connects to the internal temperature sensor.

```

ADC.read_16(): reads the current ADC value and returns it, with a range of 0-65535.



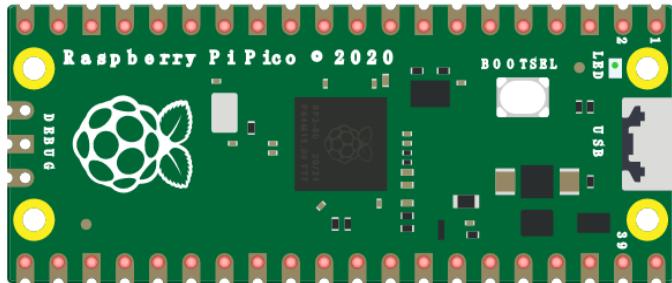
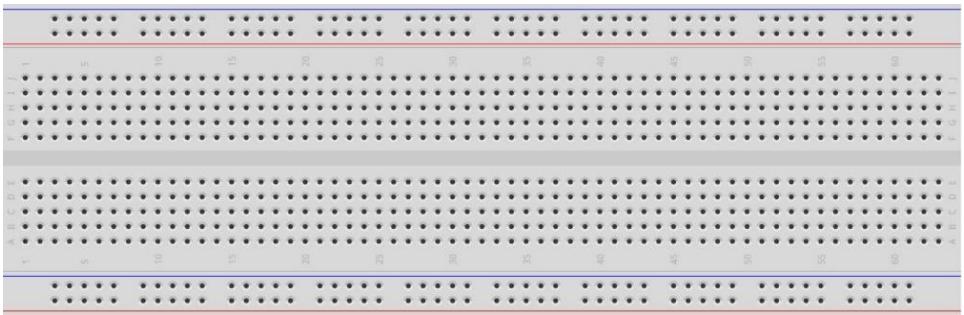
Chapter 10 Potentiometer & LED

We have learnt to use ADC in the previous chapter. In this chapter, we will combine PWM and ADC to use potentiometer to control LED, RGBLED and Neopixel.

Project 10.1 Soft Light

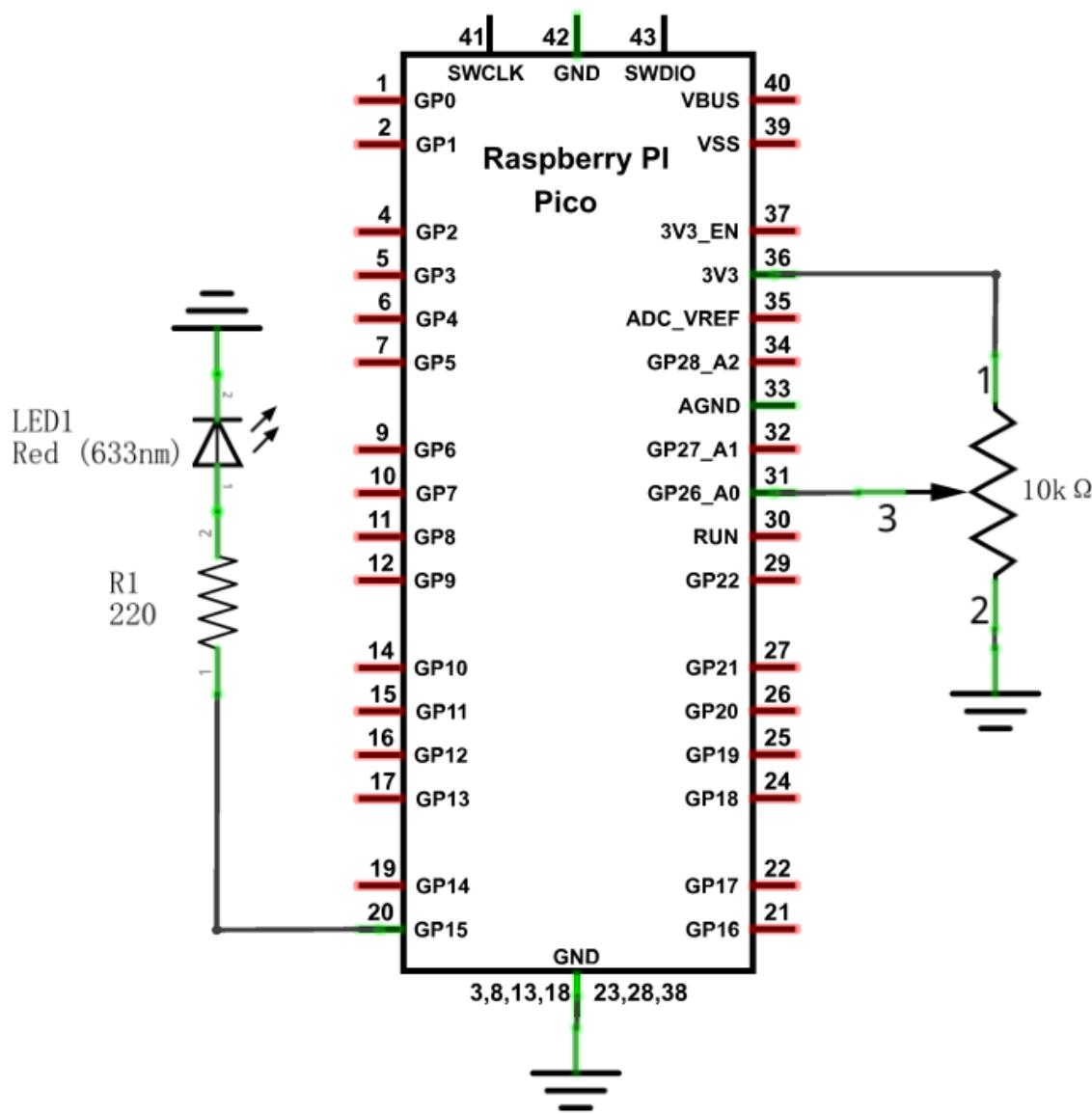
In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle of the PWM used to control the brightness of an LED. Then you can change the brightness of an LED by adjusting the potentiometer.

Component List

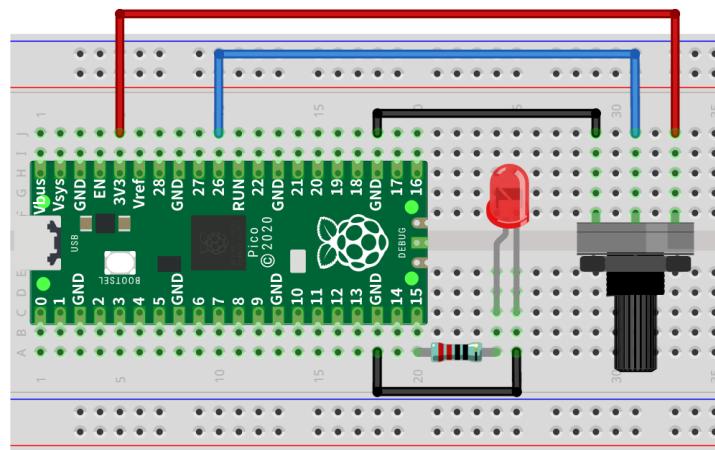
Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
			
Rotary potentiometer x1	Resistor 220Ω x1	LED x1	
			Jumper

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

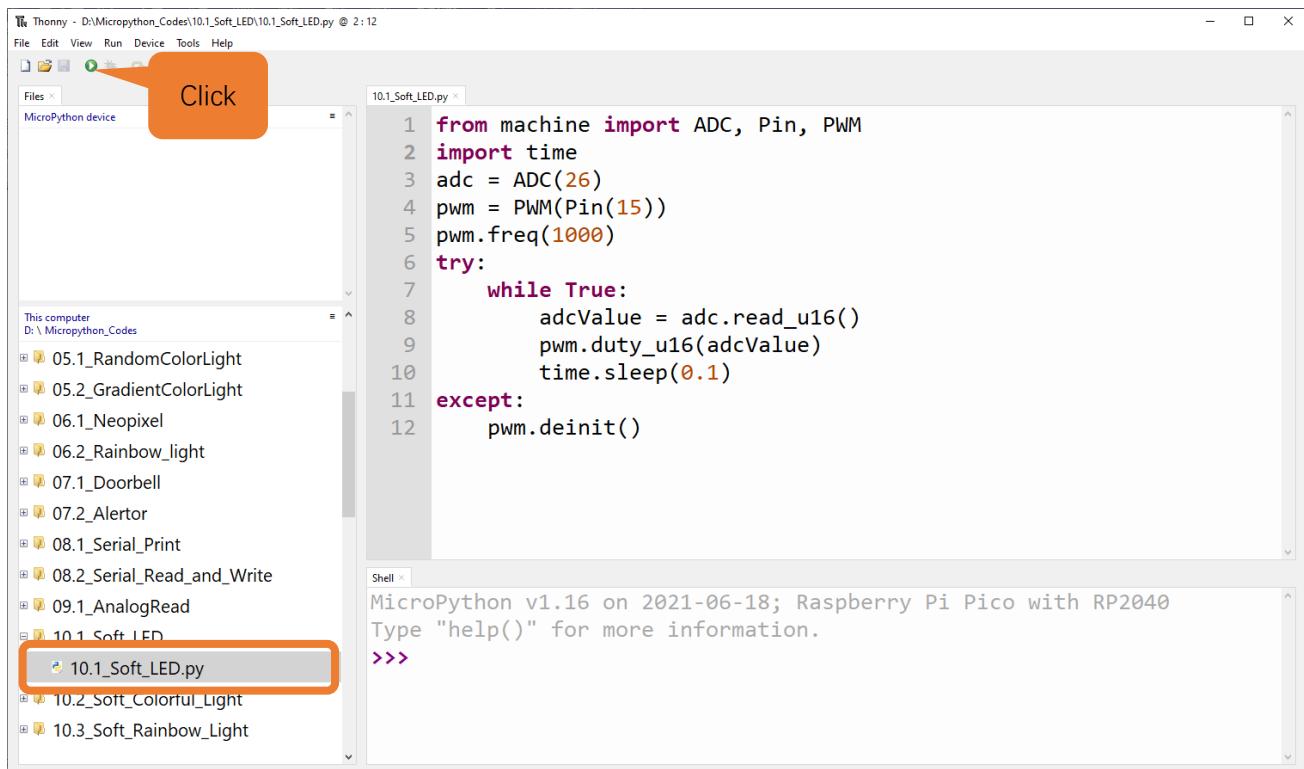


Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “10.1_Soft_LED” and double click “10.1_Soft_LED.py”.

10.1_Soft_LED



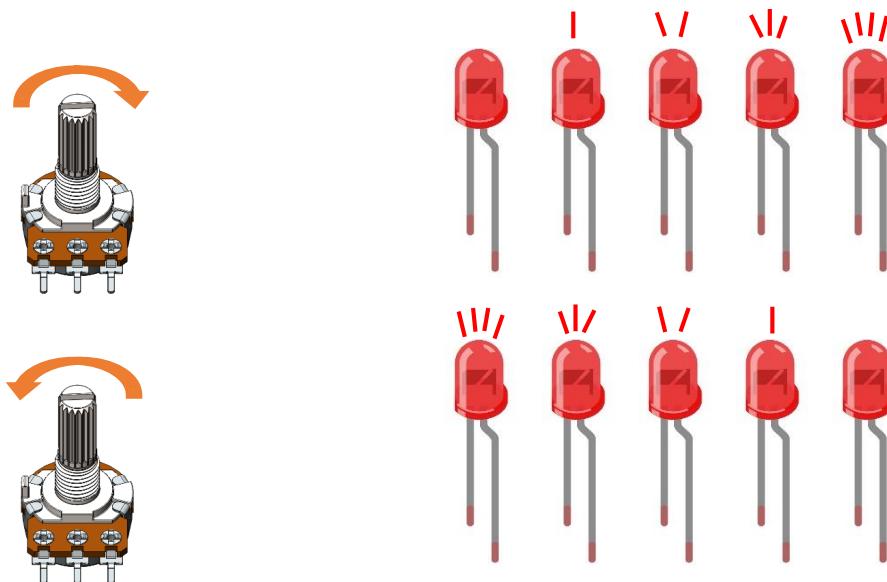
```

from machine import ADC, Pin, PWM
import time
adc = ADC(26)
pwm = PWM(Pin(15))
pwm.freq(1000)
try:
    while True:
        adcValue = adc.read_u16()
        pwm.duty_u16(adcValue)
        time.sleep(0.1)
except:
    pwm.deinit()

```

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>

Click “Run current script”. Rotate the handle of potentiometer and the brightness of LED will change correspondingly. Press Ctrl+C or click “Stop/Restart backend” to exit the program.



If you have any concerns, please contact us via: support@freenove.com

Any concerns? ✉ support@freenove.com

The following is the code:

```
1  from machine import ADC, Pin, PWM
2  import time
3  adc = ADC(26)
4  pwm = PWM(Pin(15))
5  pwm.freq(1000)
6  try:
7      while True:
8          adcValue = adc.read_u16()
9          pwm.duty_u16(adcValue)
10         time.sleep(0.1)
11     except:
12         pwm.deinit()
```

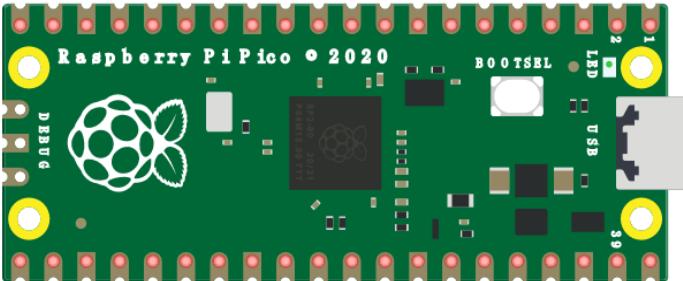
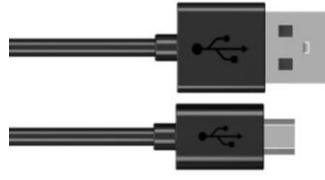
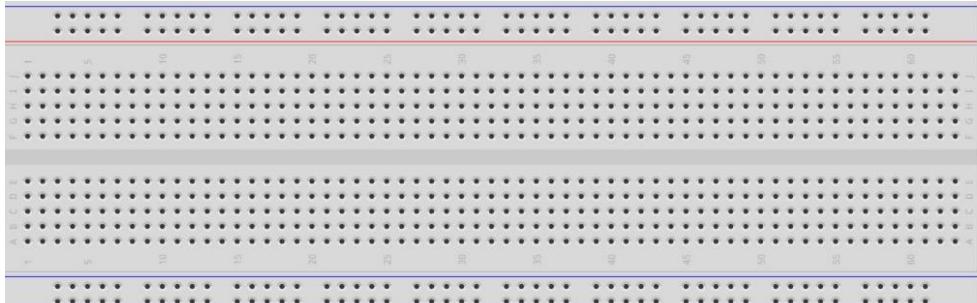
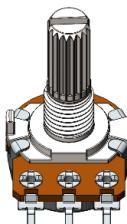
In the code, read the ADC value of potentiometer and map it to the duty cycle of PWM to control LED brightness.



Project 10.2 Soft Colorful Light

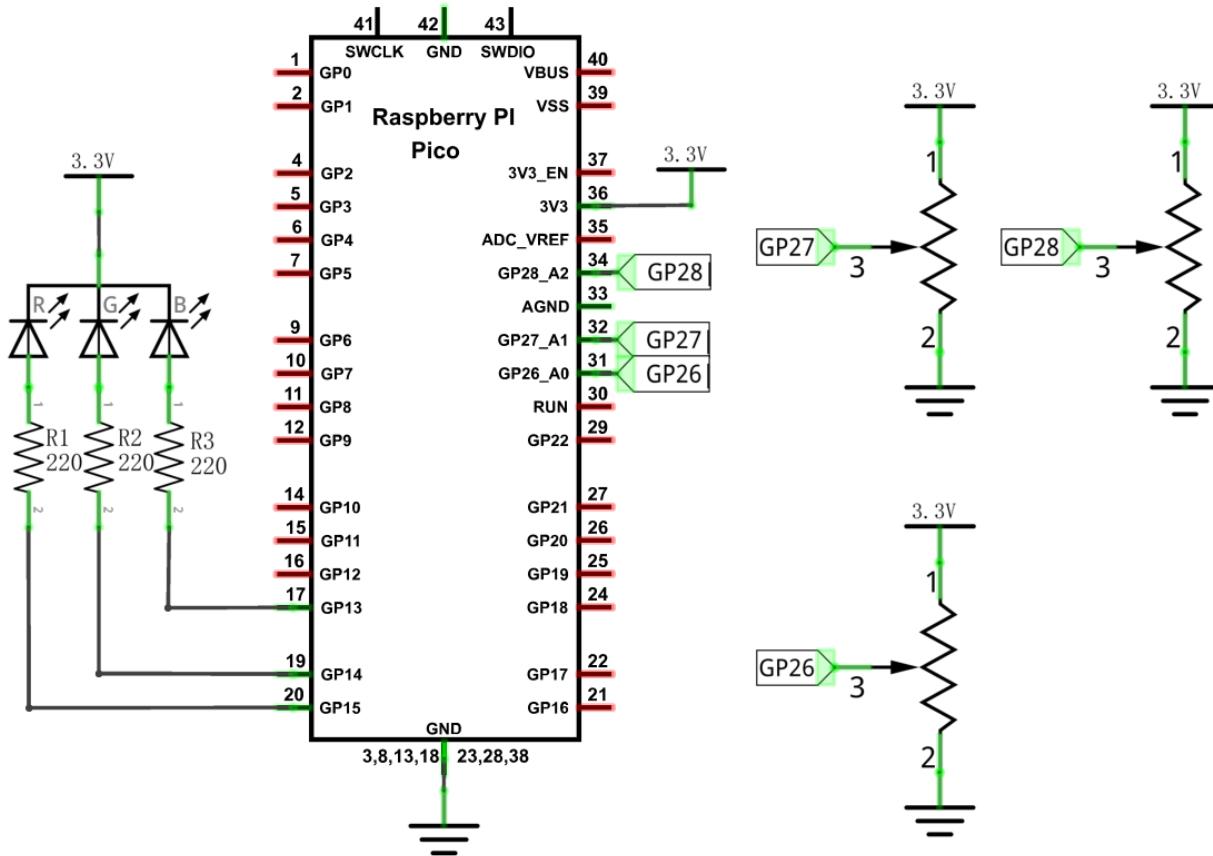
In this project, 3 potentiometers are used to control the RGB LED and in principle it is the same as the Soft Light project. Namely, read the voltage value of the potentiometer and then convert it to PWM used to control LED brightness. Difference is that the original project only controlled one LED, but this project required (3) RGB LEDs.

Component List

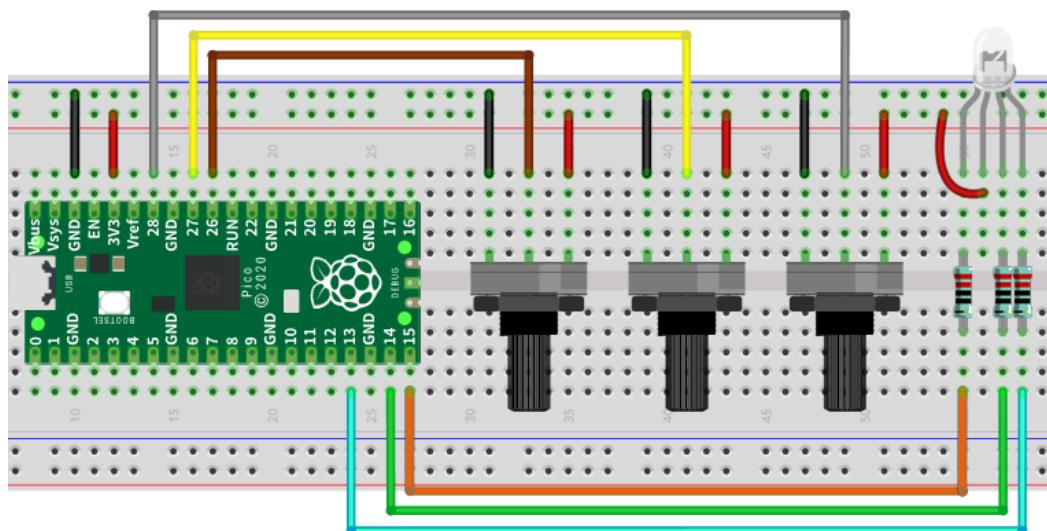
Raspberry Pi Pico x1	USB cable x1		
			
Breadboard x1			
Rotary potentiometer x3	Resistor 220Ω x3	RGBLED x1	Jumper
			

Circuit

Schematic diagram



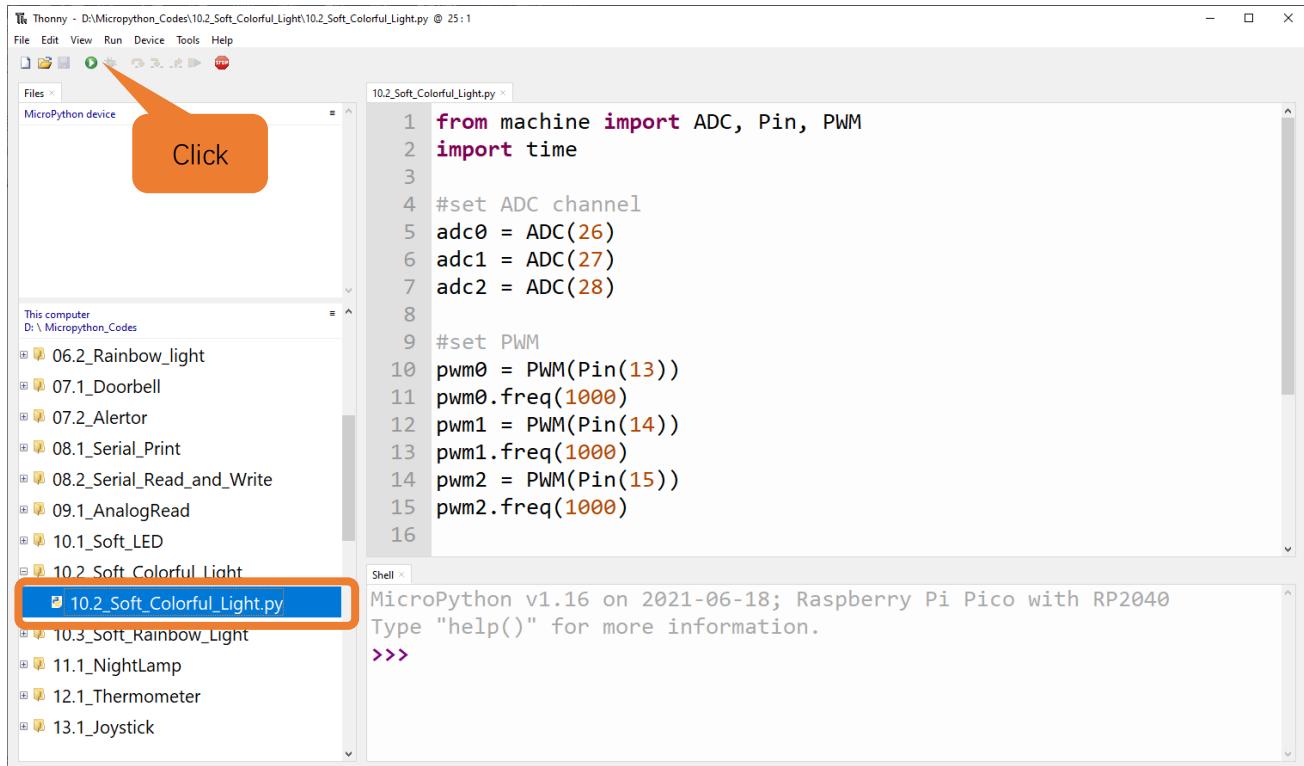
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “10.2_Soft_Colorful_Light” and double click “10.2_Soft_Colorful_Light.py”.

10.2_Soft_Colorful_Light



The screenshot shows the Thonny IDE interface. On the left, there is a file tree titled "This computer" under "D:\Micropython_Codes". The file "10.2_Soft_Colorful_Light.py" is highlighted with a blue selection bar and has an orange callout bubble pointing to it with the text "Click". The main workspace shows the code for "10.2_Soft_Colorful_Light.py". The code uses the machine module to import ADC, Pin, and PWM, and sets up three PWM channels (pwm0, pwm1, pwm2) on pins 13, 14, and 15 respectively, each at a frequency of 1000 Hz. The shell window at the bottom shows the MicroPython version and a prompt for help or commands.

```
from machine import ADC, Pin, PWM
import time

#set ADC channel
adc0 = ADC(26)
adc1 = ADC(27)
adc2 = ADC(28)

#set PWM
pwm0 = PWM(Pin(13))
pwm0.freq(1000)
pwm1 = PWM(Pin(14))
pwm1.freq(1000)
pwm2 = PWM(Pin(15))
pwm2.freq(1000)
```

Click “Run current script” and control the change of RGBLED color by rotating the handles of three rotary potentiometers. Press Ctrl+C or click “Stop/Restart backend” to exit the program.

The following is the program code:

```
1  from machine import ADC, Pin, PWM
2  import time
3
4  #set ADC channel
5  adc0 = ADC(26)
6  adc1 = ADC(27)
7  adc2 = ADC(28)
8
9  #set PWM
10 pwm0 = PWM(Pin(13))
11 pwm0.freq(1000)
12 pwm1 = PWM(Pin(14))
13 pwm1.freq(1000)
14 pwm2 = PWM(Pin(15))
15 pwm2.freq(1000)
16
17 try:
18     while True:
19         pwm0.duty_u16(65535 - adc0.read_u16())
20         pwm1.duty_u16(65535 - adc1.read_u16())
21         pwm2.duty_u16(65535 - adc2.read_u16())
22         time.sleep(0.1)
23 except:
24     pwm0.deinit()
25     pwm1.deinit()
26     pwm2.deinit()
```

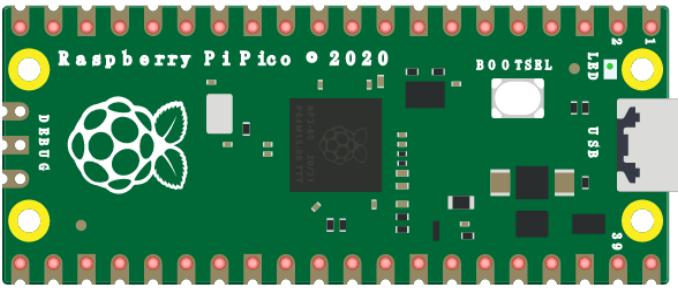
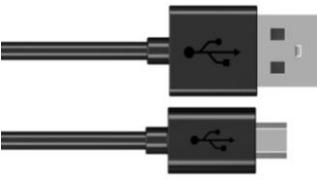
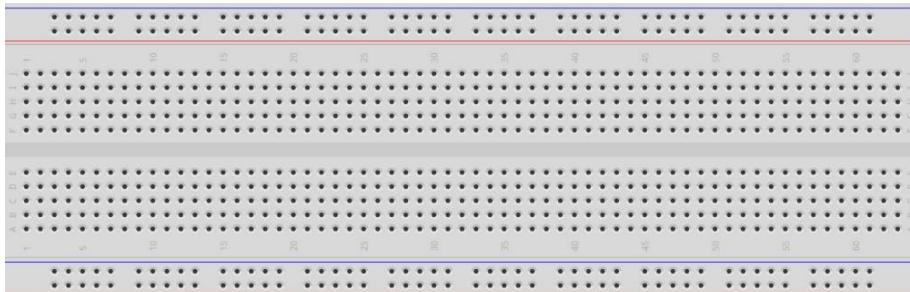
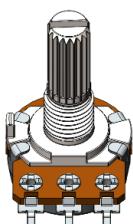
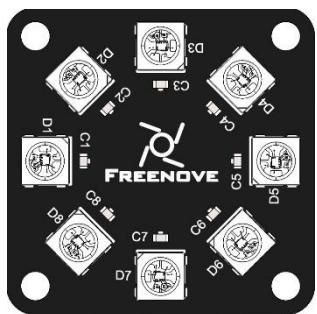
In the code you can read the ADC values of the 3 potentiometers and map it into a PWM duty cycle to control the 3 LED elements to vary the color of their respective RGB LED.



Project 10.3 Soft Rainbow Light

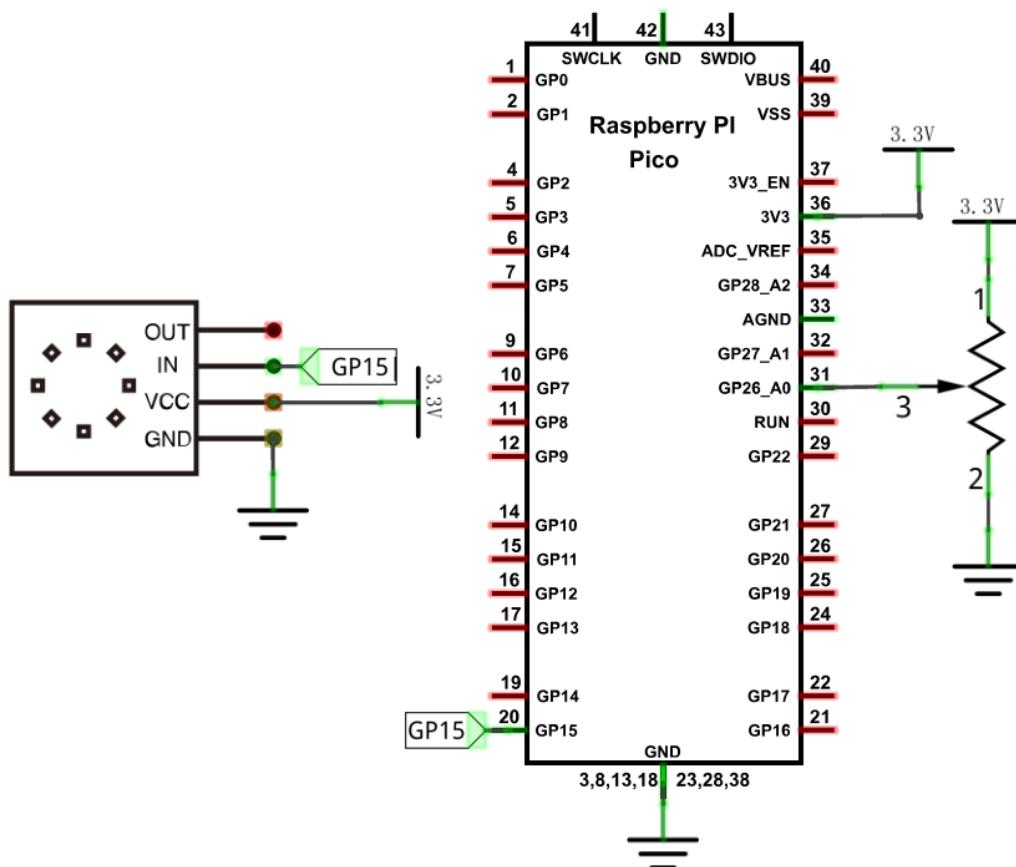
In this project, we use a potentiometer to control Freenove 8 RGBLED Module.

Component List

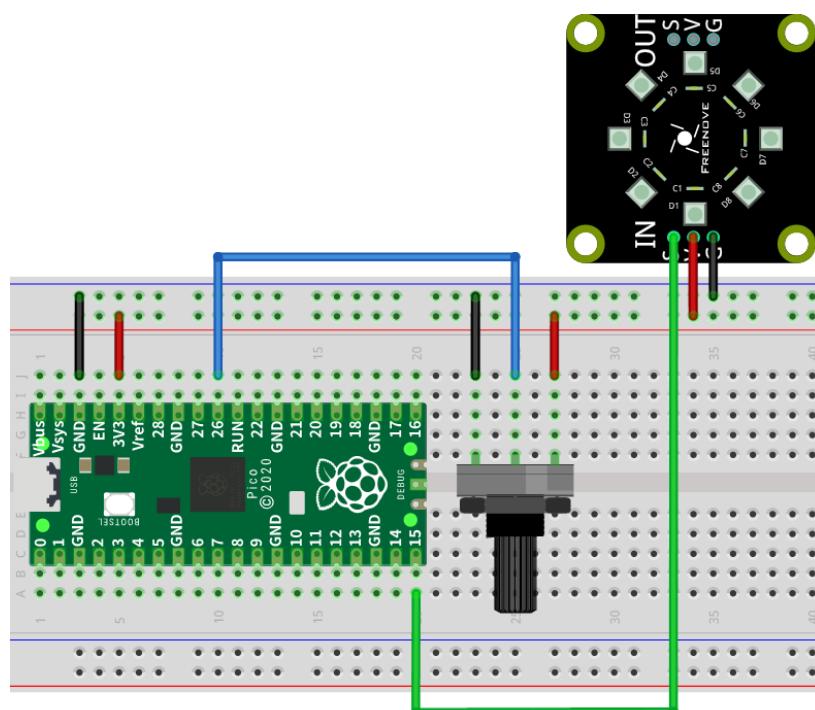
Raspberry Pi Pico x1	USB cable x1
Breadboard x1	
Rotary potentiometer x1	Freenove 8 RGB LED Module x1
	
	
	

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Any concerns? ✉ support@freenove.com



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “10.3_Soft_Rainbow_Light”. Select “neopixel.py”, right click to select “Upload to /”, wait for “neopixel.py” to be uploaded to Raspberry Pi Pico and then double click “10.3_Soft_Rainbow_Light.py”.

10.3_Soft_Rainbow_Light

```

from machine import Pin, ADC
from neopixel import myNeopixel
import time

red=0           #red
green=0          #green
blue=0           #blue
np = myNeopixel(8, 15)
adc0=ADC(26)

def wheel(pos):
    global red, green, blue
    
```

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>

Click “Run current script”. Rotate the handle of potentiometer and the color of the lights will change.



If you have any concerns, please contact us via: support@freenove.com

Any concerns? ✉ support@freenove.com

The following is the program code:

```
1  from machine import Pin, ADC
2  from neopixel import myNeopixel
3  import time
4
5  red=0          #red
6  green=0        #green
7  blue=0         #blue
8  np = myNeopixel(8, 15)
9  adc0 = ADC(26)
10
11 def wheel(pos):
12     global red, green, blue
13     WheelPos=pos%255
14     if WheelPos < 85:
15         red = (255 - WheelPos*3)
16         green = (WheelPos*3)
17         blue = 0
18     elif WheelPos >= 85 and WheelPos < 170:
19         WheelPos -= 85
20         red = 0
21         green = (255 - WheelPos*3)
22         blue = (WheelPos*3)
23     else:
24         WheelPos -= 170
25         red = (WheelPos*3)
26         green = 0
27         blue = (255 - WheelPos*3)
28
29 np.brightness(20)
30 while True:
31     for i in range(0, 255):
32         for j in range(0, 8):
33             wheel(i + j*255 // 8)
34             np.set_pixel(j, red, green, blue)
35             np.show()
36             time.sleep_ms(1)
```

The logic of the code is basically the same as the previous project [Rainbow Light](#). The difference is that in this code, the starting point of the color is controlled by the potentiometer.



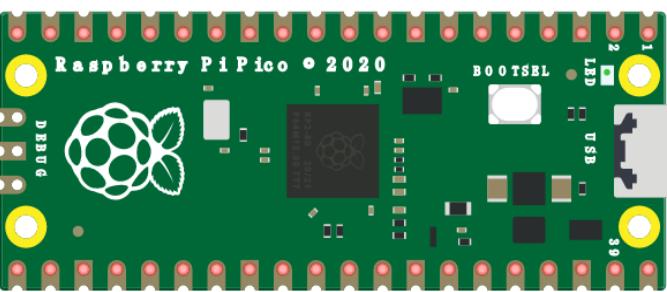
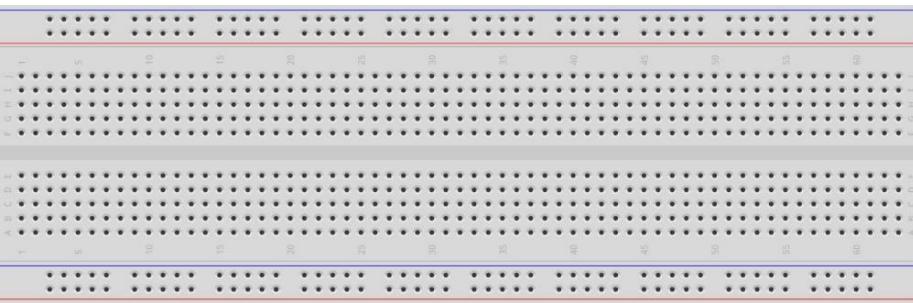
Chapter 11 Photoresistor & LED

In this chapter, we will learn how to use photoresistor.

Project 11.1 Control LED through Photoresistor

A Photoresistor is very sensitive to the amount of light present. We can take advantage of the characteristic to make a night lamp with the following function: when the ambient light is less (darker environment) the LED will automatically become brighter to compensate and when the ambient light is greater (brighter environment) the LED will automatically dim to compensate.

Component List

Raspberry Pi Pico x1		USB cable x1												
Breadboard x1														
<table border="1"> <thead> <tr> <th>Photoresistor x1</th> <th>Resistor</th> <th>LED x1</th> <th>Jumper</th> </tr> </thead> <tbody> <tr> <td></td><td>220Ω x1</td><td></td><td></td></tr> <tr> <td></td><td>10KΩ x1</td><td></td><td></td></tr> </tbody> </table>		Photoresistor x1	Resistor	LED x1	Jumper		220Ω x1				10KΩ x1			
Photoresistor x1	Resistor	LED x1	Jumper											
	220Ω x1													
	10KΩ x1													

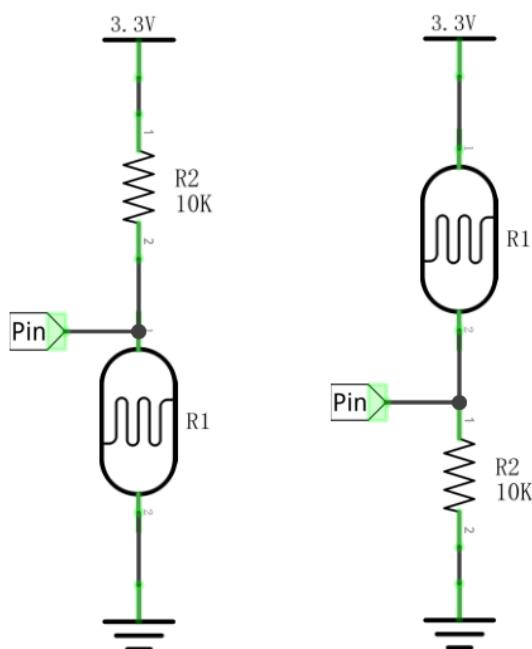
Component knowledge

Photoresistor

Photoresistor is simply a light sensitive resistor. It is an active component that decreases resistance with respect to receiving luminosity (light) on the component's light sensitive surface. Photoresistor's resistance value will change in proportion to the ambient light detected. With this characteristic, we can use a Photoresistor to detect light intensity. The Photoresistor and its electronic symbol are as follows.



The circuit below is used to detect the change of a Photoresistor's resistance value:

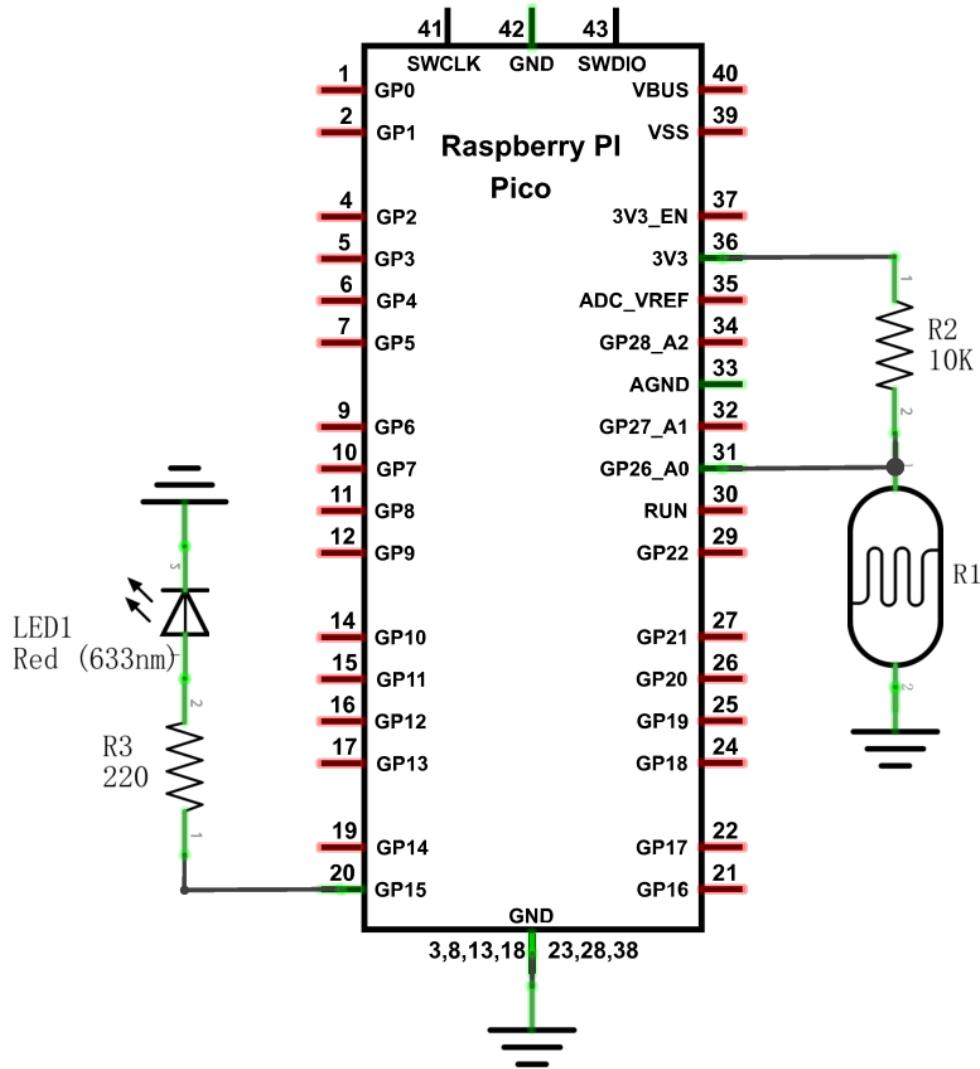


In the above circuit, when a Photoresistor's resistance value changes due to a change in light intensity, the voltage between the Photoresistor and Resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

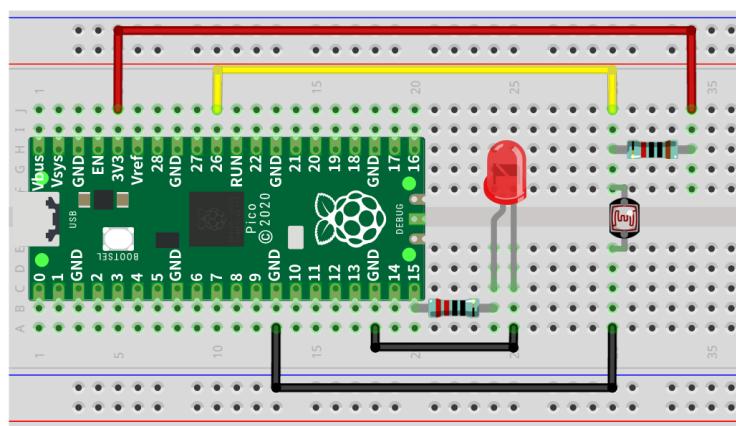
Circuit

The circuit of this project is similar to SoftLight. The only difference is that the input signal is changed from a potentiometer to a combination of a photoresistor and a resistor.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Any concerns? ✉ support@freenove.com

Code

Codes of this project is logically the same as the project [Soft Light](#).

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “11.1_Photoresistor” and double click “11.1_Photoresistor.py”.

11.1_Photoresistor



```

from machine import Pin, ADC, PWM
import time

adc = ADC(26)
pwm = PWM(Pin(15))
pwm.freq(10000)
try:
    while True:
        pwm.duty_u16(adc.read_u16())
        time.sleep(0.1)
except:
    pwm.deinit()

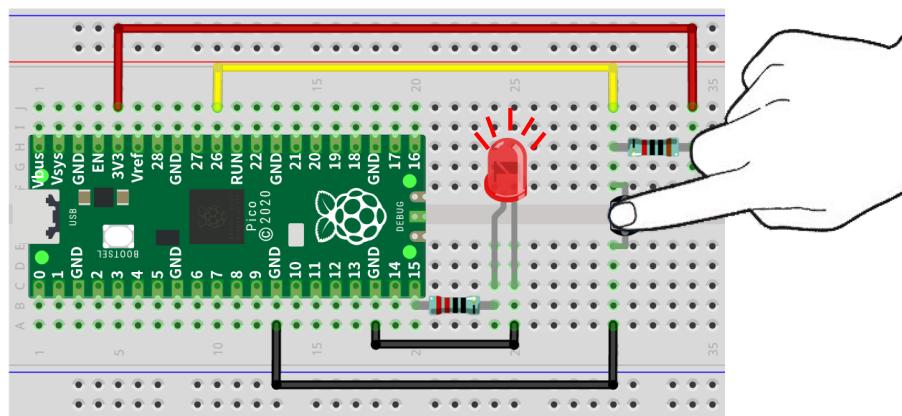
```

Shell >>>

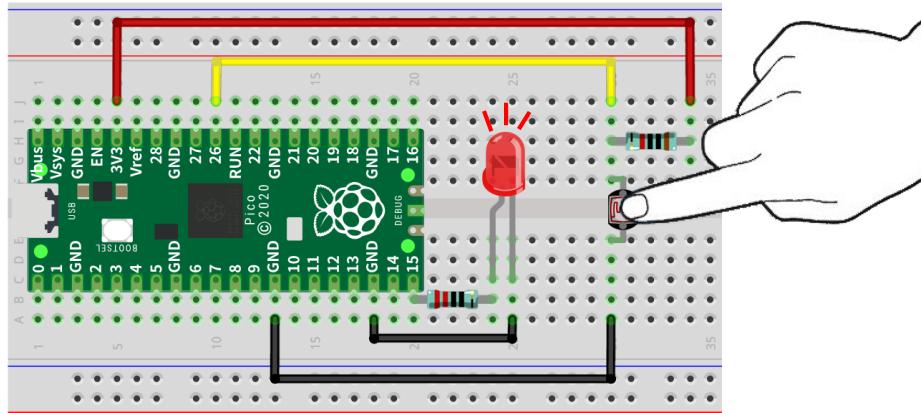
MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.

Click “Run current script”. Cover the photoresistor with your hands or illuminate it with lights, the brightness of LEDs will change.

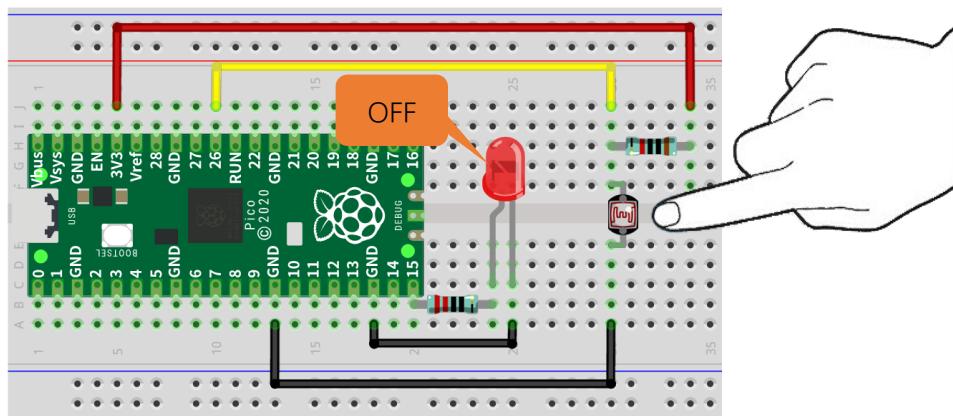
Fully cover the photoresistor:



Half cover the photoresistor:



Not cover the photoresistor:



The following is the program code:

```

1 from machine import Pin, ADC, PWM
2 import time
3
4 adc = ADC(26)
5 pwm = PWM(Pin(15))
6 pwm.freq(10000)
7 try:
8     while True:
9         pwm.duty_u16(adc.read_u16())
10        time.sleep(0.1)
11 except:
12     pwm.deinit()

```

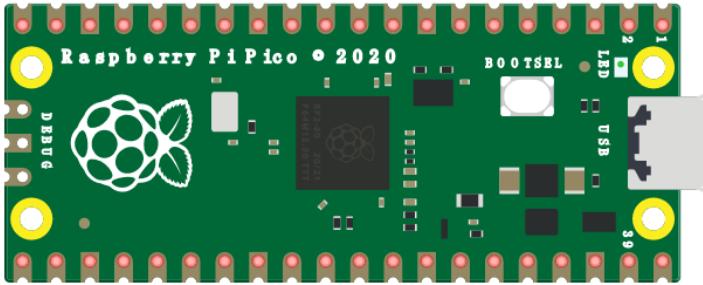
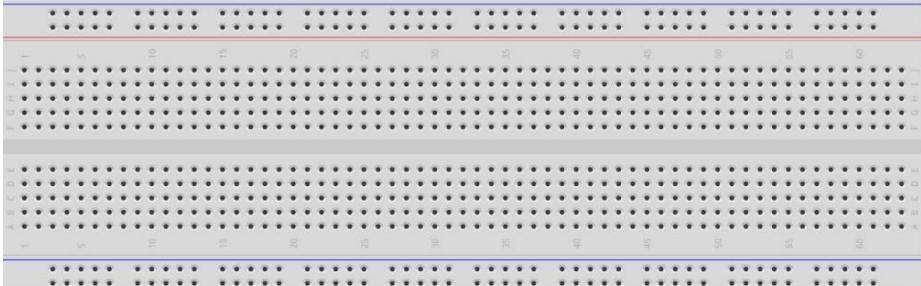
Chapter 12 Thermistor

In this chapter, we will learn about Thermistors which are another kind of Resistor.

Project 12.1 Thermometer

A Thermistor is a type of Resistor whose resistance value is dependent on temperature and changes in temperature. Therefore, we can take advantage of this characteristic to make a Thermometer.

Component List

Raspberry Pi Pico x1	USB cable x1	
		
Breadboard x1		
		
Thermistor x1	Resistor 10kΩ x1	Jumper
		

Component knowledge

Thermistor

A Thermistor is a temperature sensitive resistor. When it senses a change in temperature, the resistance of the Thermistor will change. We can take advantage of this characteristic by using a Thermistor to detect temperature intensity. A Thermistor and its electronic symbol are shown below.



The relationship between resistance value and temperature of a thermistor is:

$$R_t = R * \text{EXP} \left[B * \left(\frac{1}{T_2} - \frac{1}{T_1} \right) \right]$$

Where:

Rt is the thermistor resistance under T2 temperature;

R is the nominal resistance of thermistor under T1 temperature;

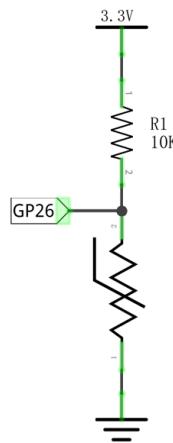
EXP[n] is nth power of e;

B is for thermal index;

T1, T2 is Kelvin temperature (absolute temperature). Kelvin temperature=273.15 + Celsius temperature.

For the parameters of the Thermistor, we use: B=3950, R=10kΩ, T1=25°C.

The circuit connection method of the Thermistor is similar to photoresistor, as the following:



We can use the value measured by the ADC converter to obtain the resistance value of Thermistor, and then we can use the formula to obtain the temperature value.

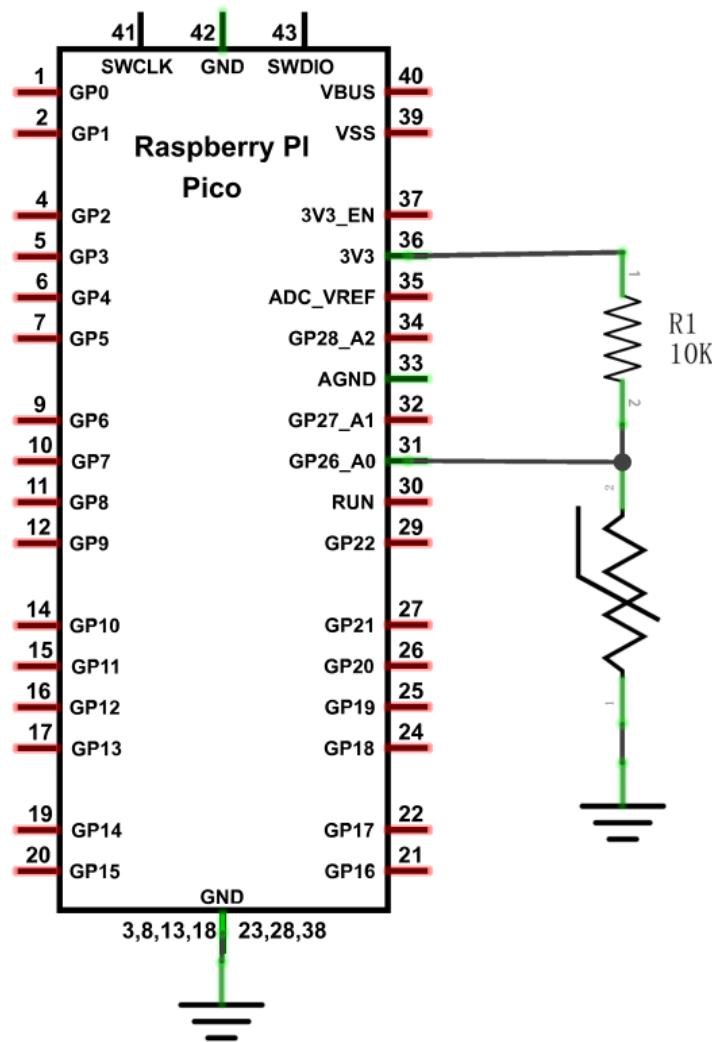
Therefore, the temperature formula can be derived as:

$$T_2 = 1 / \left(\frac{1}{T_1} + \ln \left(\frac{R_t}{R} \right) / B \right)$$

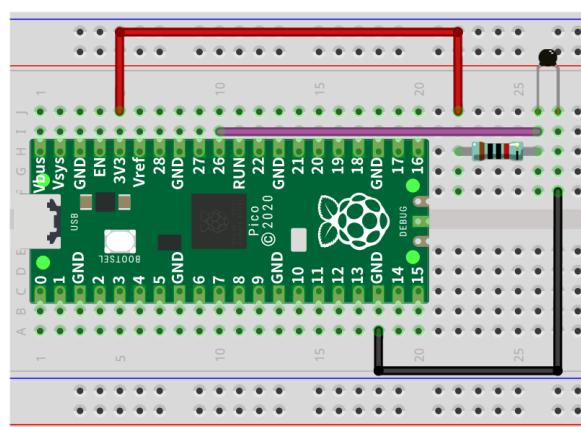
Circuit

The circuit of this project is similar to the one in the previous chapter. The only difference is that the Photoresistor is replaced by a Thermistor.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “12.1_Thermometer” and double click “12.1_Thermometer.py”.

12.1_Thermometer

```

from machine import Pin, ADC
import time
import math

#Set ADC
adc=ADC(26)

try:
    while True:
        adcValue = adc.read_u16()
        voltage = adcValue / 65535.0 * 3.3
        Rt = 10 * voltage / (3.3-voltage)
        tempK = (1 / (1 / (273.15+25) + (math.log(Rt/10)) / 3950))
        tempC = int(tempK - 273.15)
        print("ADC value:", adcValue, " Voltage: %0.2f" %voltage,
              " Temperature: %0.1fC" %tempC)

```

The Shell window shows the following output:

ADC value	Voltage	Temperature
31975	1.61	26C
31975	1.61	26C
32023	1.61	26C
32359	1.63	25C
32135	1.62	25C
32183	1.62	25C

Click “Run current script” and “Shell” will constantly display the current ADC value, voltage value and temperature value. Try to “pinch” the thermistor (without touching the leads) with your index finger and thumb for a brief time, you should see that the temperature value increases. Press Ctrl+C or click “Stop/Restart backend” to exit the program.

ADC value	Voltage	Temperature
32135	1.62	25C
32055	1.61	25C
31975	1.61	26C
31975	1.61	26C
32023	1.61	26C
32359	1.63	25C
32135	1.62	25C
32183	1.62	25C

If you have any concerns, please contact us via: support@freenove.com

Any concerns? ✉ support@freenove.com

The following is the code:

```

1  from machine import Pin, ADC
2  import time
3  import math
4
5  #Set ADC
6  adc=ADC(26)
7
8  try:
9      while True:
10         adcValue = adc.read_u16()
11         voltage = adcValue / 65535.0 * 3.3
12         Rt = 10 * voltage / (3.3-voltage)
13         tempK = (1 / (1 / (273.15+25) + (math.log(Rt/10)) / 3950))
14         tempC = int(tempK - 273.15)
15         print("ADC value:", adcValue, " Voltage: %0.2f"%voltage,
16               " Temperature: " + str(tempC) + "C")
17         time.sleep(1)
18     except:
19         pass

```

`read_u16()` function is called to read the value of ADC0.

```
10    adcValue = adc.read_u16()
```

Convert the read ADC0 value to get the current Thermistor voltage value.

```
11    voltage = adcValue / 65535.0 * 3.3
```

The current Thermistor resistance (`Rt`) is calculated by Ohm's law.

```
12    Rt = 10 * voltage / (3.3-voltage)
```

According to the formula: $T_2 = 1/\left(\frac{1}{T_1} + \ln\left(\frac{R_t}{R}\right)/B\right)$, where $T_1 = 25^\circ\text{C}$, $R = 10\text{K}\Omega$, $B = 3950$ and the `Rt` calculated

in the previous step, substitute the formula to calculate `tempK(T2)`. Get the value of the temperature unit K.

```
13    tempK = (1 / (1 / (273.15+25) + (math.log(Rt/10)) / 3950))
```

Finally, `tempK` (unit: K) is converted to `tempC` (unit: $^\circ\text{C}$). You can also convert to Fahrenheit based on your needs.

```
14    tempC = int(tempK - 273.15)
```



Chapter 13 Joystick

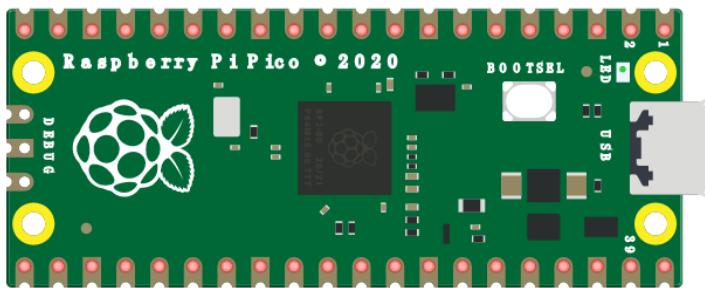
In the previous chapter, we have learned how to use rotary potentiometer. Now, let's learn a new electronic module Joystick which works on the same principle as rotary potentiometer.

Project 13.1 Joystick

In this project, we will read the output data of a Joystick and display it to the Terminal screen.

Component List

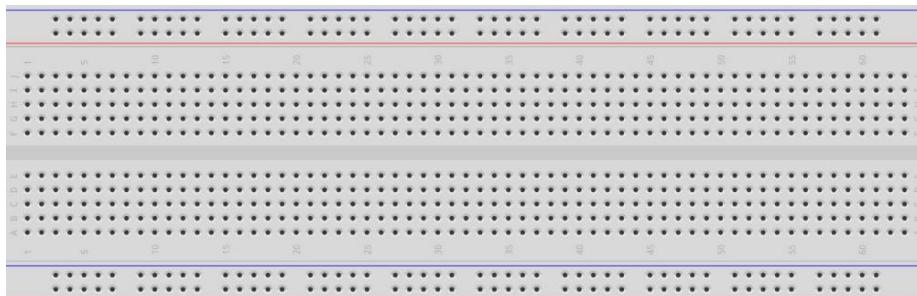
Raspberry Pi Pico x1



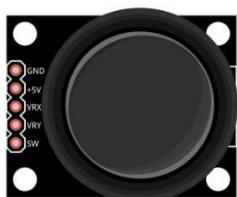
USB cable x1



Breadboard x1



Joystick x1



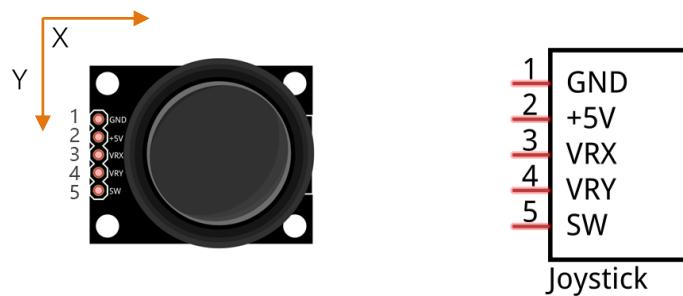
Jumper



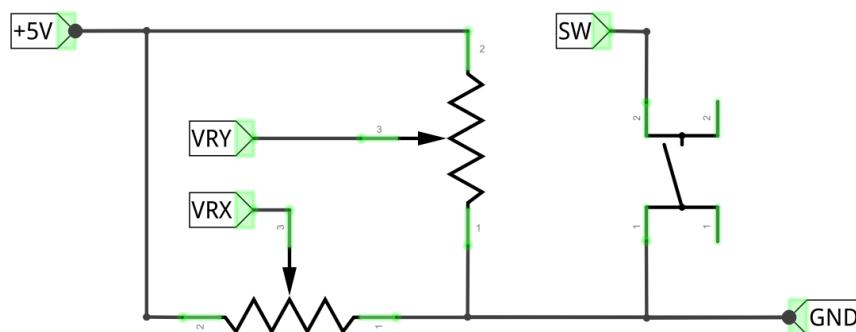
Component knowledge

Joystick

A Joystick is a kind of input sensor used with your fingers. You should be familiar with this concept already as they are widely used in gamepads and remote controls. It can receive input on two axes (Y and or X) at the same time (usually used to control direction on a two dimensional plane). And it also has a third direction capability by pressing down (Z axis/direction).



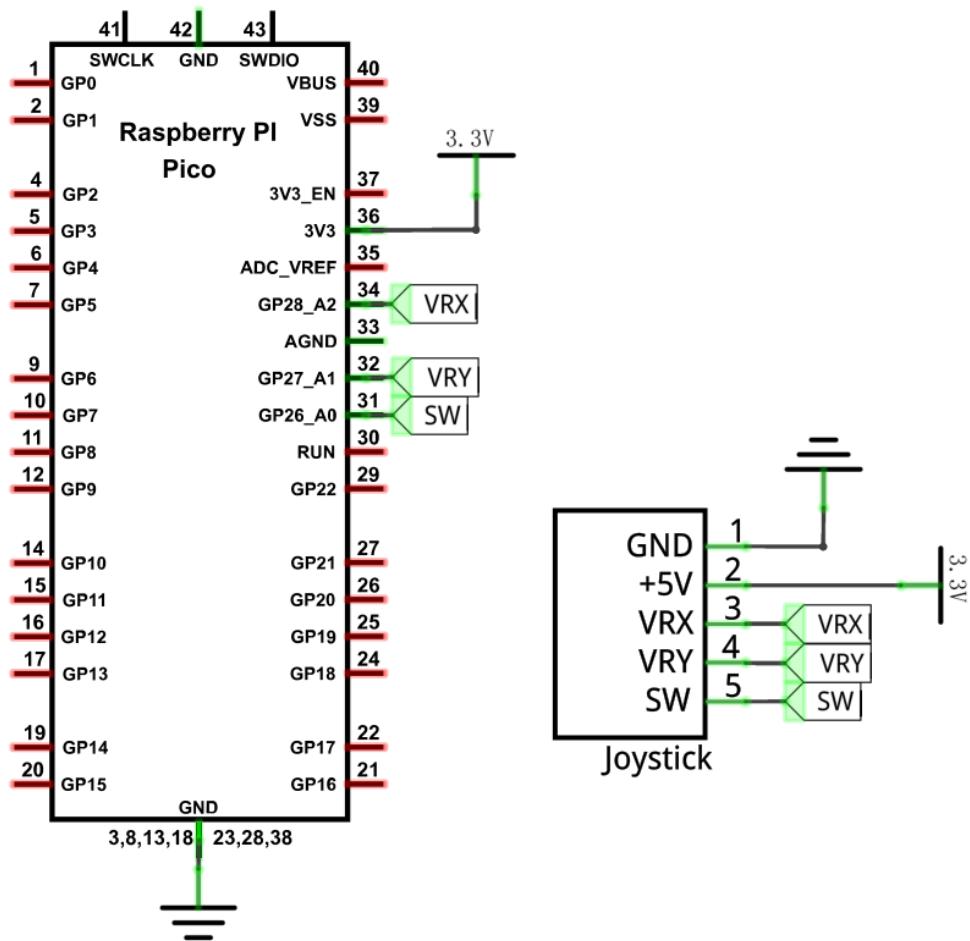
This is accomplished by incorporating two rotary potentiometers inside the Joystick Module at 90 degrees of each other, placed in such a manner as to detect shifts in direction in two directions simultaneously and with a Push Button Switch in the “vertical” axis, which can detect when a User presses on the Joystick.



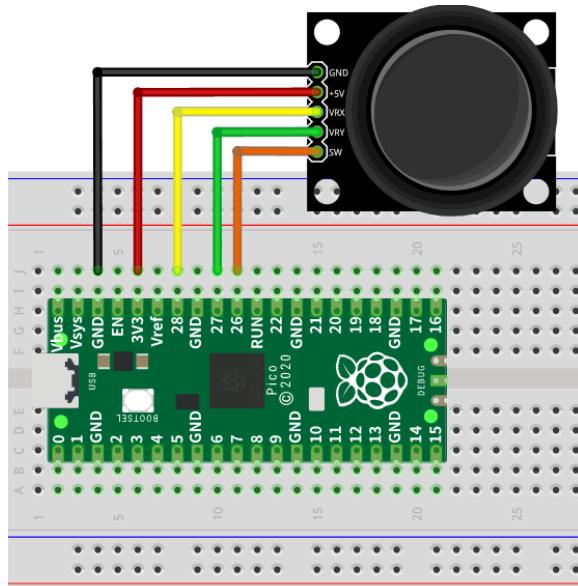
When the Joystick data is read, there are some differences between the axes: data of X and Y axes is analog, which needs to use the ADC. The data of the Z axis is digital, so you can directly use the GPIO to read this data or you have the option to use the ADC to read this.

Circuit

Schematic diagram



Hardware connection. If you need any support, please free to contact us via: support@freenove.com



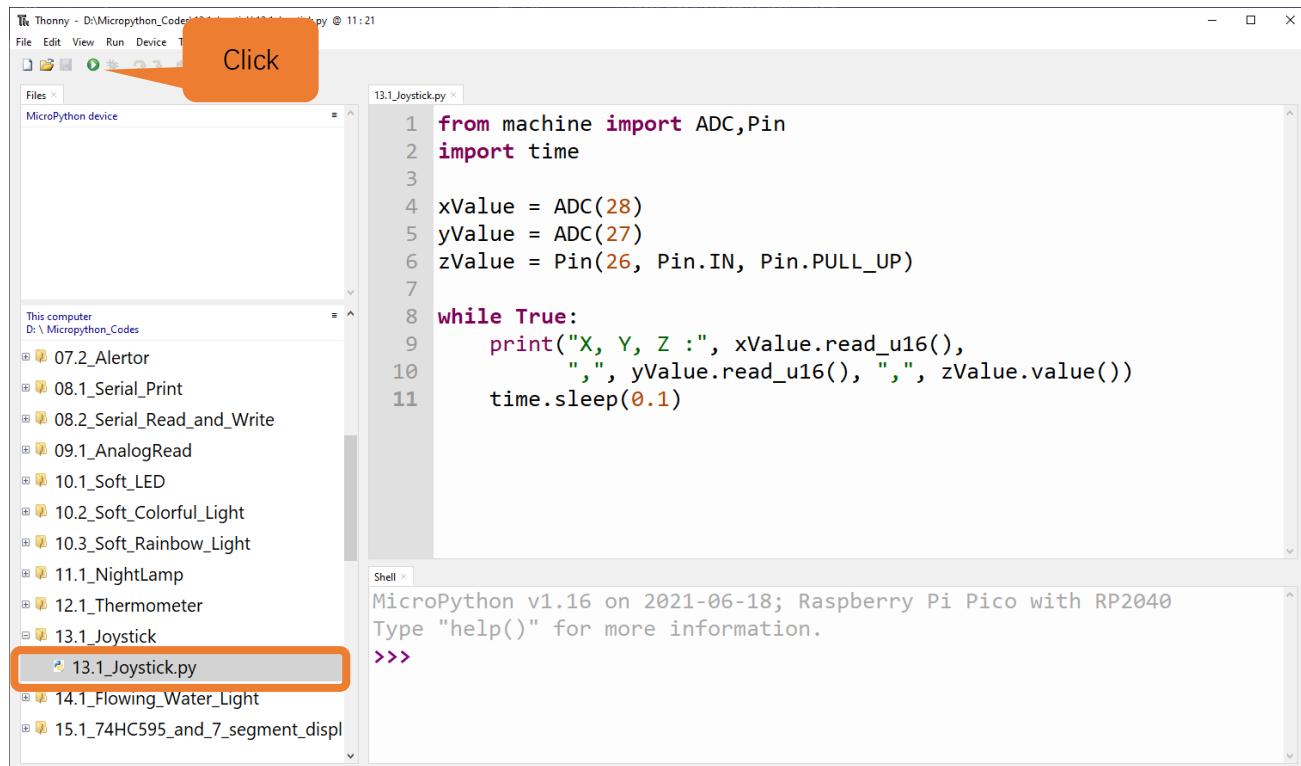
Any concerns? ✉ support@freenove.com

Code

In this project's code, we will read the ADC values of X and Y axes of the Joystick, and read digital quality of the Z axis, then display these out in Terminal.

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "13.1_Joystick" and double click "13.1_Joystick.py".

13.1_Joystick



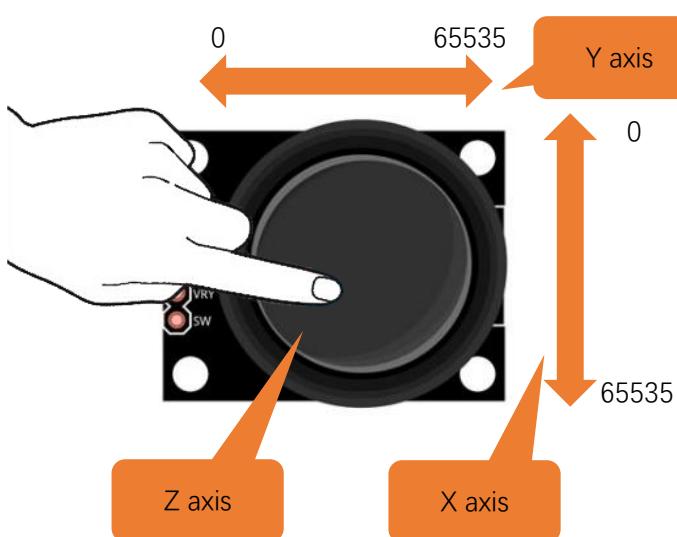
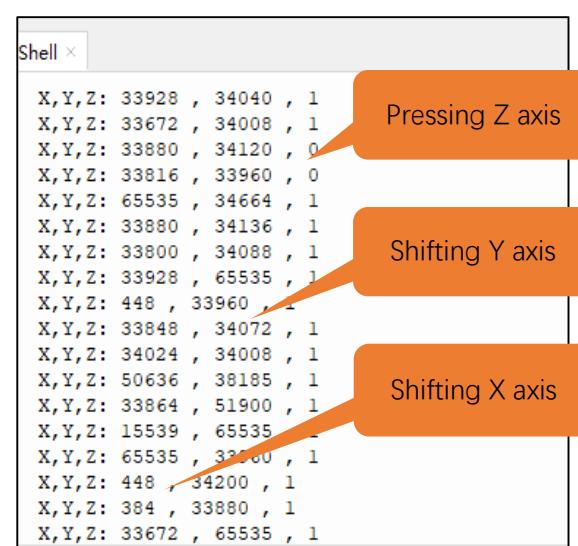
```

from machine import ADC,Pin
import time

xValue = ADC(28)
yValue = ADC(27)
zValue = Pin(26, Pin.IN, Pin.PULL_UP)

while True:
    print("X, Y, Z : ", xValue.read_u16(),
          ", ", yValue.read_u16(), ", ", zValue.value())
    time.sleep(0.1)
  
```

Click "Run current script". Shifting the Joystick or pressing it down will change the printed data in "Shell". Press Ctrl+C or click "Stop/Restart backend" to exit the program.

```

X,Y,Z: 33928 , 34040 , 1
X,Y,Z: 33672 , 34008 , 1
X,Y,Z: 33880 , 34120 , 0
X,Y,Z: 33816 , 33960 , 0
X,Y,Z: 65535 , 34664 , 1
X,Y,Z: 33880 , 34136 , 1
X,Y,Z: 33800 , 34088 , 1
X,Y,Z: 33928 , 65535 , 1
X,Y,Z: 448 , 33960 , 1
X,Y,Z: 33848 , 34072 , 1
X,Y,Z: 34024 , 34008 , 1
X,Y,Z: 50636 , 38185 , 1
X,Y,Z: 33864 , 51900 , 1
X,Y,Z: 15539 , 65535 , 1
X,Y,Z: 65535 , 32760 , 1
X,Y,Z: 448 , 34200 , 1
X,Y,Z: 384 , 33880 , 1
X,Y,Z: 33672 , 65535 , 1
  
```



The flowing is the code:

```
1 from machine import ADC, Pin  
2 import time  
3  
4 xValue = ADC(28)  
5 yValue = ADC(27)  
6 zValue = Pin(26, Pin.IN, Pin.PULL_UP)  
7  
8 while True:  
9     print("X, Y, Z :", xValue.read_u16(),  
10          ", ", yValue.read_u16(), ", ", zValue.value())  
11     time.sleep(0.1)
```

In the code, configure Z_Pin to pull-up input mode. In loop, use read_u16 () to read the value of axies X and Y and use value() to read the value of axis Z, and then display them.

```
9     print("X, Y, Z :", xValue.read_u16(),  
10           ", ", yValue.read_u16(), ", ", zValue.value())
```

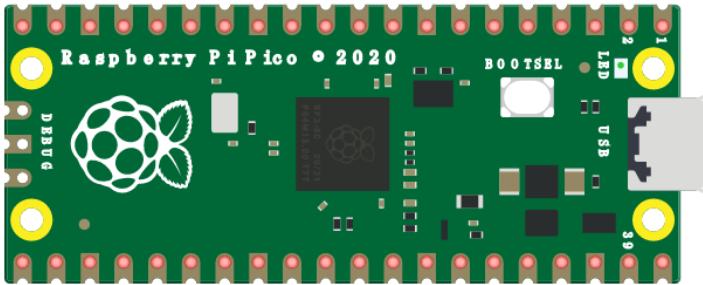
Chapter 14 74HC595 & LED Bar Graph

We have used LED Bar Graph to make a flowing water light, in which 10 GPIO ports of Raspberry Pi Pico is occupied. More GPIO ports mean that more peripherals can be connected to Raspberry Pi Pico, so GPIO resource is very precious. Can we make flowing water light with less GPIO? In this chapter, we will learn a component, 74HC595, which can achieve the target.

Project 14.1 Flowing Water Light

Now let's learn how to use the 74HC595 IC Chip to make a flowing water light using less GPIO.

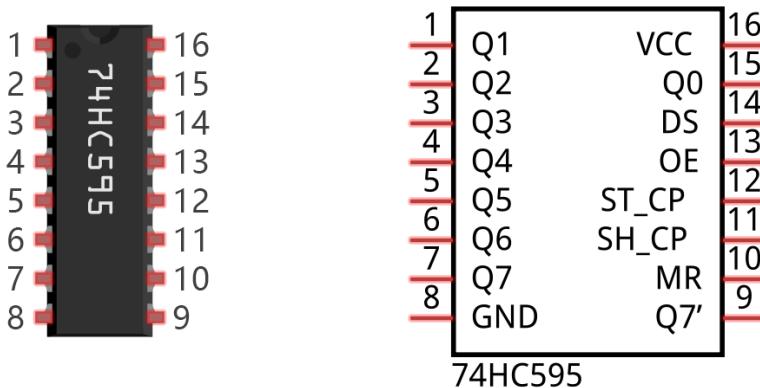
Component List

Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
74HC595 x1	LED Bar Graph x1	Resistor 220Ω x8	Jumper
			

Related knowledge

74HC595

A 74HC595 chip is used to convert serial data into parallel data. A 74HC595 chip can convert the serial data of one byte into 8 bits, and send its corresponding level to each of the 8 ports correspondingly. With this characteristic, the 74HC595 chip can be used to expand the IO ports of Raspberry Pi Pico. At least 3 ports are required to control the 8 ports of the 74HC595 chip.

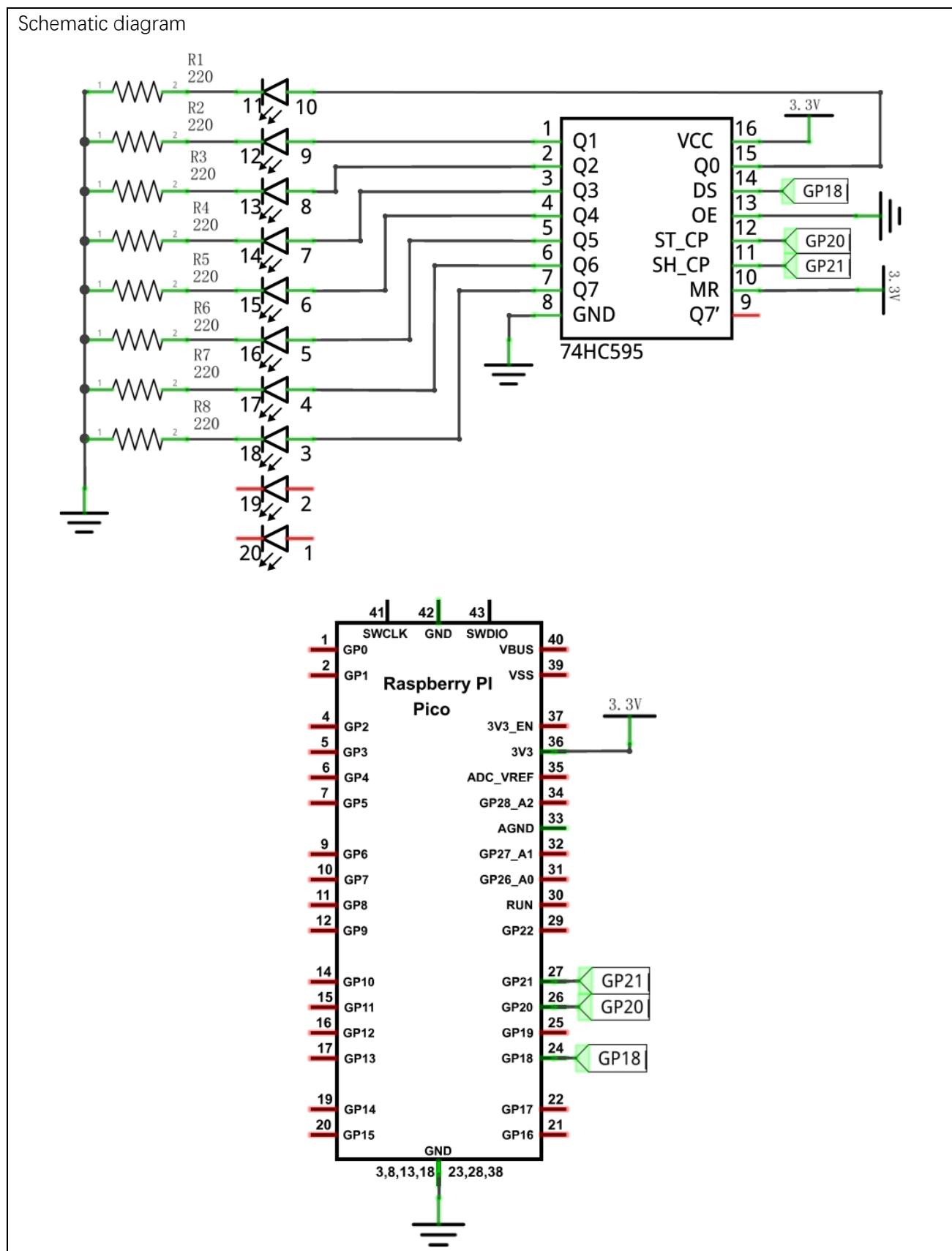


The ports of the 74HC595 chip are described as follows:

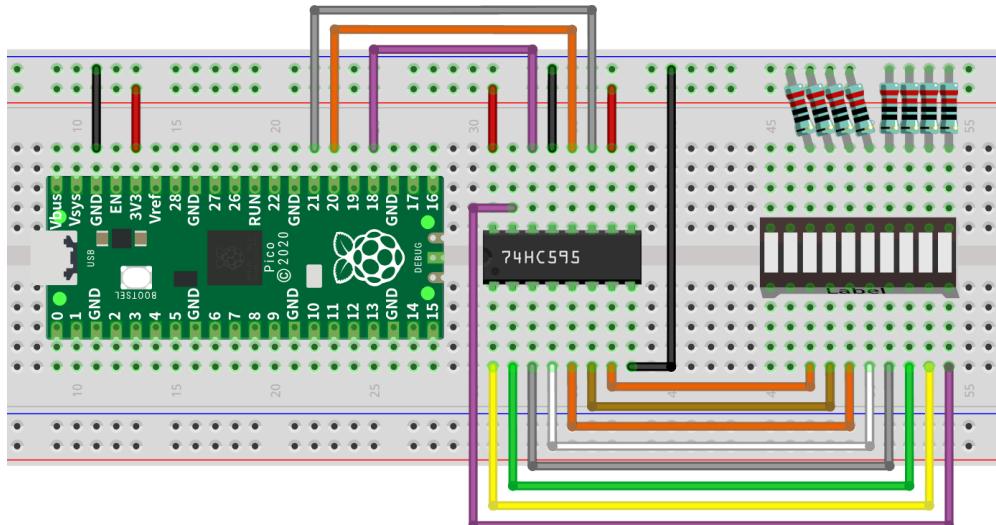
Pin name	GPIO number	Description
Q0-Q7	15, 1-7	Parallel data output
VCC	16	The positive electrode of power supply, the voltage is 2~6V
GND	8	The negative electrode of power supply
DS	14	Serial data Input
OE	13	Enable output, When this pin is in high level, Q0-Q7 is in high resistance state When this pin is in low level, Q0-Q7 is in output mode
ST_CP	12	Parallel Update Output: when its electrical level is rising, it will update the parallel data output.
SH_CP	11	Serial shift clock: when its electrical level is rising, serial data input register will do a shift.
MR	10	Remove shift register: When this pin is in low level, the content in shift register will be cleared.
Q7'	9	Serial data output: it can be connected to more 74HC595 in series.

For more detail, please refer to the datasheet on the 74HC595 chip.

Circuit



Hardware connection. If you need any support, please free to contact us via: support@freenove.com



Code

In this project, we will make a flowing water light with a 74HC595 chip to learn about its functions.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “14.1_Flowing_Water_Light”.

Select “my74HC595.py”, right click your mouse to select “Upload to /”, wait for “my74HC595.py” to be uploaded to Raspberry Pi Pico and then double click “14.1_Flowing_Water_Light.py”.

14.1_Flowing_Water_Light

```

1 import time
2 from my74HC595 import Chip74HC595
3
4 chip = Chip74HC595(18, 20, 21)
#Chip74HC595() == Chip74HC595(18, 20, 21)
5
6
7 while True:
8     x = 0x01
9     for count in range(8):
10         chip.shiftOut(1, x)
11         x = x<<1;
12         time.sleep_ms(300)
13     x = 0x01
14     for count in range(8):
15         chip.shiftOut(0, x)
16         x = x<<1

```

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>

Click “Run current script” and you will see that Bar Graph LED starts with the flowing water pattern blinking

Any concerns? ✉ support@freenove.com

from left to right and then back from right to left. If it displays nothing, maybe the LED Bar is connected upside down, please unplug it and then re-plug it reversely. Press Ctrl+C or click “Stop/Restart backend” to exit the program.



If you have any concerns, please contact us via: support@freenove.com

The following is the program code:

```

1 import time
2 from my74HC595 import Chip74HC595
3
4 chip = Chip74HC595(18, 20, 21)
5 #Chip74HC595() == Chip74HC595(18, 20, 21)
6
7 while True:
8     x=0x01
9     for count in range(8):
10         chip.shiftOut(1, x)
11         x=x<<1;
12         time.sleep_ms(300)
13     x=0x01
14     for count in range(8):
15         chip.shiftOut(0, x)
16         x=x<<1
17         time.sleep_ms(300)

```

Import time and my74HC595 modules.

```

1 import time
2 from my74HC595 import Chip74HC595

```

Create a Chip74HC595 object and configure pins, among which, the default Raspberry Pi Pico and 74HC595 bound pins are DS(GP18), STCP(GP20), SHCP(GP21), OE(GP19). If you want to use other pins, you can change the pins by changing the arguments passed to the Chip74HC595 object.

```

4 chip = Chip74HC595(18, 20, 21)
5 #Chip74HC595() == Chip74HC595(18, 20, 21)

```

The first for loop makes LED Bar display separately from left to right while the second for loop make it display separately from right to left.

```

8 x=0x01
9     for count in range(8):
10         chip.shiftOut(1, x)
11         x=x<<1;
12         time.sleep_ms(300)
13     x=0x01
14     for count in range(8):

```

Any concerns? ✉ support@freenove.com

```
15     chip.shiftOut(0, x)
16     x=x<<1
17     time.sleep_ms(300)
```

Reference

Class Chip74HC595

Before each use of the object **Chip74HC595**, make sure my74HC595.py has been uploaded to "/" of Raspberry Pi Pico, and then add the statement "**from my74HC595 import Chip74HC595**" to the top of the python file.

Chip74HC595(): An object. By default, 74HC595's DS pin is connected to GP18 of Raspberry Pi Pico, ST_CP pin is connected to Raspberry Pi Pico GP20, SH_CP pin is connected to Raspberry Pi Pico GP21, OE pin is connected to Raspberry Pi Pico GP19. If you need to modify the pins, just do the following operations.

chip=Chip74HC595() or **chip=Chip74HC595(18, 20, 21, 19)**.

shiftOut(direction, data): Write data to 74HC595.

direction: When direction=1, it indicates data is sent from the highest byte to the lowest byte (left to right) in turn; direction=0 indicates data is sent from the lowest byte to the highest byte (right to left) in turn.

data: The content that is sent, which is one-byte data.

clear(): Clear the latch data of 74HC595.

Chapter 15 74HC595 & 7-Segment Display.

In this chapter, we will introduce the 7-Segment Display.

Project 15.1 7-Segment Display.

We will use 74HC595 to control 7-segment display and make it display hexadecimal character "0-F".

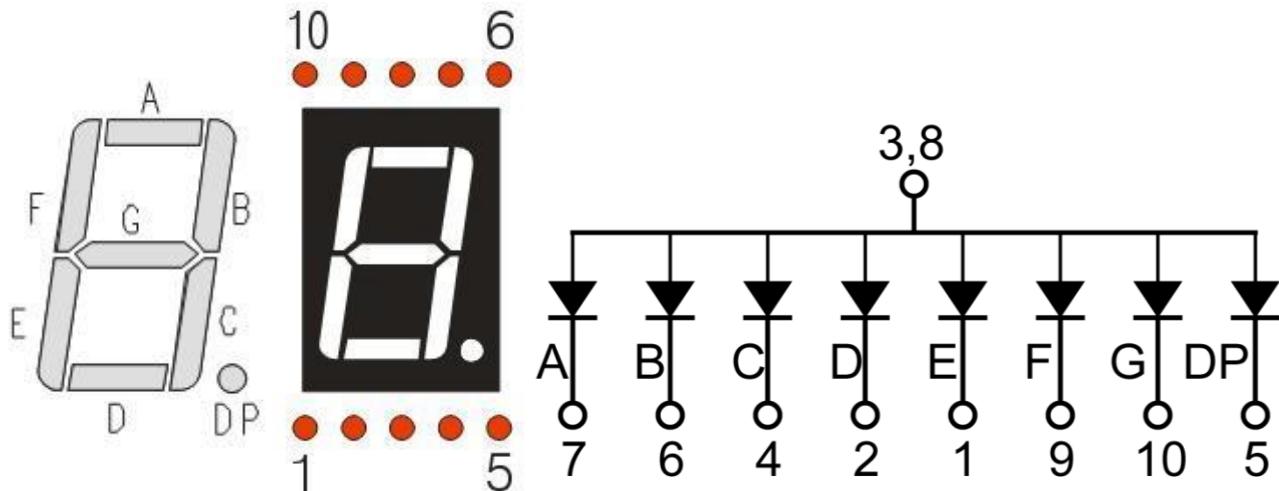
Component List

Raspberry Pi Pico x1		USB cable x1
Breadboard x1		
74HC595 x1	7-segment display x1	Resistor 220Ω x8
		
		Jumper

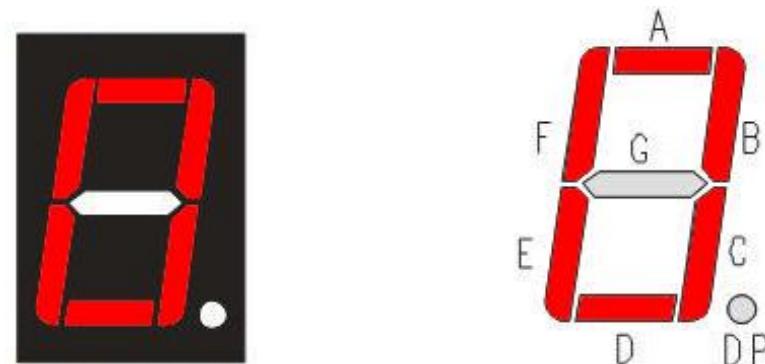
Component knowledge

7-segment display

A 7-Segment Display is a digital electronic display device. There is a figure "8" and a decimal point represented, which consists of 8 LEDs. The LEDs have a Common Anode and individual Cathodes. Its internal structure and pin designation diagram is shown below:



As we can see in the above circuit diagram, we can control the state of each LED separately. Also, by combining LEDs with different states of ON and OFF, we can display different characters (Numbers and Letters). For example, to display a "0": we need to turn ON LED segments A, B, C, D, E and F, and turn OFF LED segments G and DP.



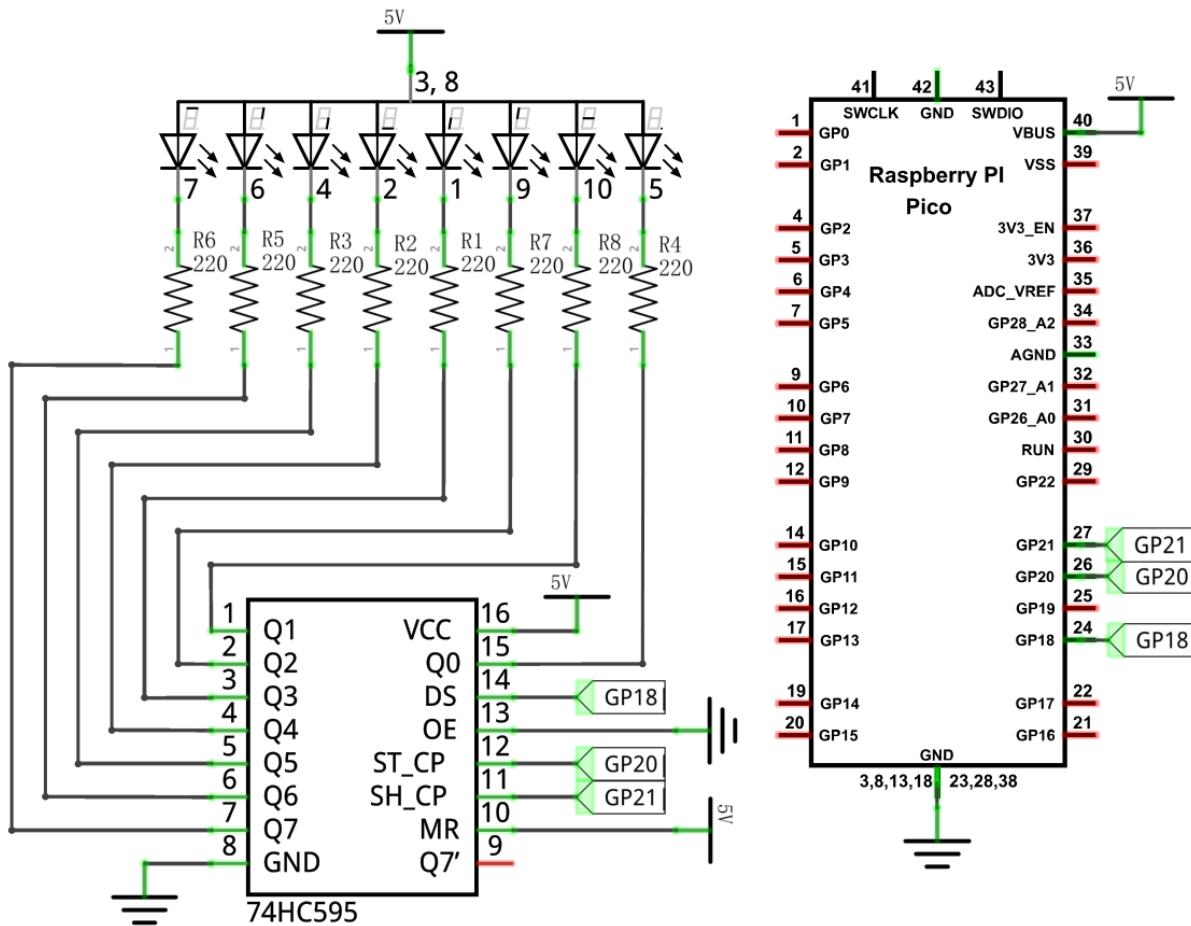
In this project, we will use a 7-Segment Display with a Common Anode. Therefore, when there is an input low level to an LED segment the LED will turn ON. Defining segment "A" as the lowest level and segment "DP" as the highest level, from high to low would look like this: "DP", "G", "F", "E", "D", "C", "B", "A". Character "0" corresponds to the code: 1100 0000b=0xc0.

For detailed code values, please refer to the following table (common anode).

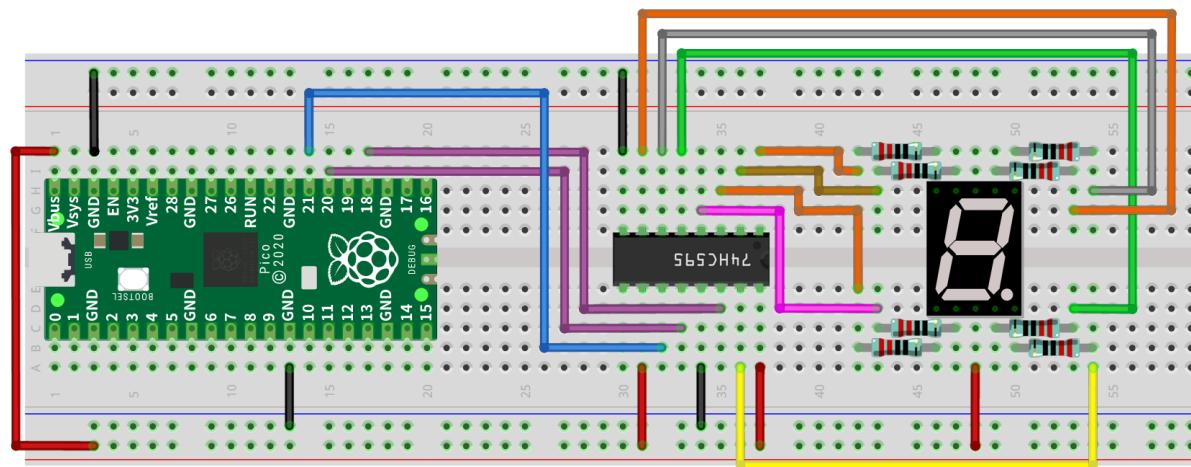
CHAR	DP	G	F	E	D	C	B	A	Hex	ASCII
0	1	1	0	0	0	0	0	0	0xc0	1100 0000
1	1	1	1	1	1	0	0	1	0xf9	1111 1001
2	1	0	1	0	0	1	0	0	0xa4	1010 0100
3	1	0	1	1	0	0	0	0	0xb0	1011 0000
4	1	0	0	1	1	0	0	1	0x99	1001 1001
5	1	0	0	1	0	0	1	0	0x92	1001 0010
6	1	0	0	0	0	0	1	0	0x82	1000 0010
7	1	1	1	1	1	0	0	0	0xf8	1111 1000
8	1	0	0	0	0	0	0	0	0x80	1000 0000
9	1	0	0	1	0	0	0	0	0x90	1001 0000
A	1	0	0	0	1	0	0	0	0x88	1000 1000
B	1	0	0	0	0	0	1	1	0x83	1000 0011
C	1	1	0	0	0	1	1	0	0xc6	1100 0110
D	1	0	1	0	0	0	0	1	0xa1	1010 0001
E	1	0	0	0	0	1	1	0	0x86	1000 0110
F	1	0	0	0	1	1	1	0	0x8e	1000 1110

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

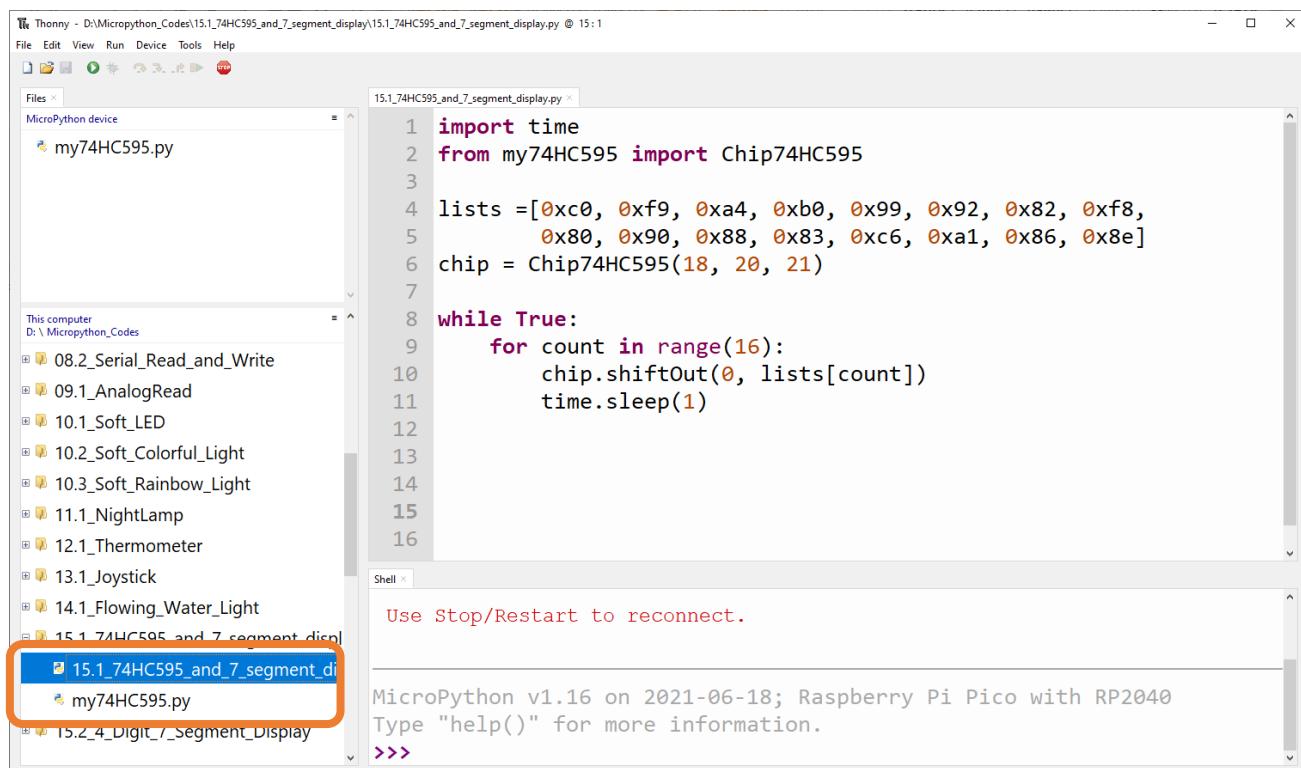


Code

In this section, the 74HC595 is used in the same way as in the previous section, but with different values transferred. We can learn how to master the digital display by sending the code value of "0" - "F".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "15.1_74HC595_and_7_segment_display". Select "my74HC595.py", right click your mouse to select "Upload to /", wait for "my74HC595.py" to be uploaded to Raspberry Pi Pico and then double click "15.1_74HC595_and_7_segment_display.py".

15.1_74HC595_and_7_segment_display



```

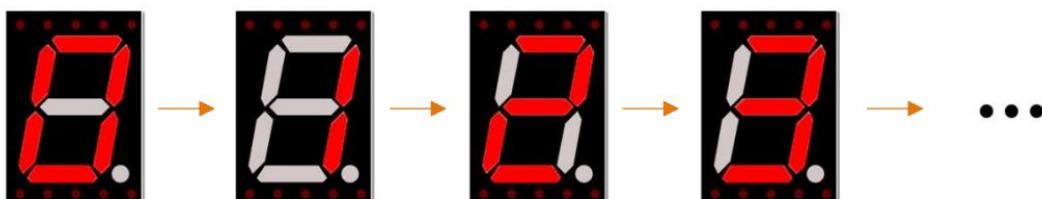
1 import time
2 from my74HC595 import Chip74HC595
3
4 lists =[0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
5         0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e]
6 chip = Chip74HC595(18, 20, 21)
7
8 while True:
9     for count in range(16):
10         chip.shiftOut(0, lists[count])
11         time.sleep(1)
12
13
14
15
16

```

Use Stop/Restart to reconnect.

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>

Click "Run current script", and you'll see a 1-bit, 7-segment display displaying 0-f in a loop. Press Ctrl+C or click "Stop/Restart backend" to exit the program.





The following is the program code:

```
1 import time
2 from my74HC595 import Chip74HC595
3
4 lists = [0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
5         0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e]
6 chip = Chip74HC595(18, 20, 21)
7
8 while True:
9     for count in range(16):
10        chip.shiftOut(0, lists[count])
11        time.sleep(1)
```

Import time and my74HC595 modules.

```
1 import time
2 from my74HC595 import Chip74HC595
```

Put the encoding "0" - "F" into the list.

```
4 lists =[0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
5       0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e]
```

Define an object, whose pins applies default configuration, to drive 74HC595.

```
6 chip = Chip74HC595(18, 20, 21)
```

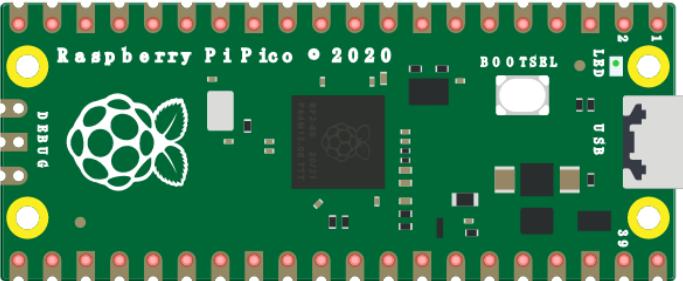
Send data of digital tube to 74HC595 chip.

```
10 chip.shiftOut(0, lists[count])
```

Project 15.2 4-Digit 7-Segment Display

Now, let's try to control a more-digit 7-segment display.

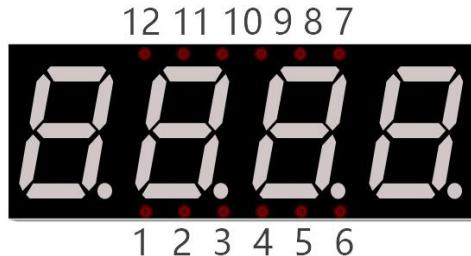
Component List

Raspberry Pi Pico x1		USB cable x1
Breadboard x1		
74HC595 x1	7-segment display x1	Resistor 220Ω x8
		
		Jumper

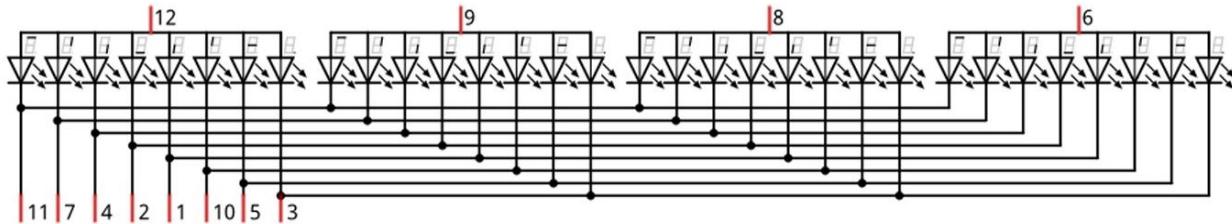
Component knowledge

4 Digit 7-Segment Display

A 4-Digit 7-segment display integrates four 7-Segment Displays into one module. Therefore, it can display more characters. All the LEDs contained have a Common Anode and individual Cathodes. Its internal structure and pin designation diagram is shown below:



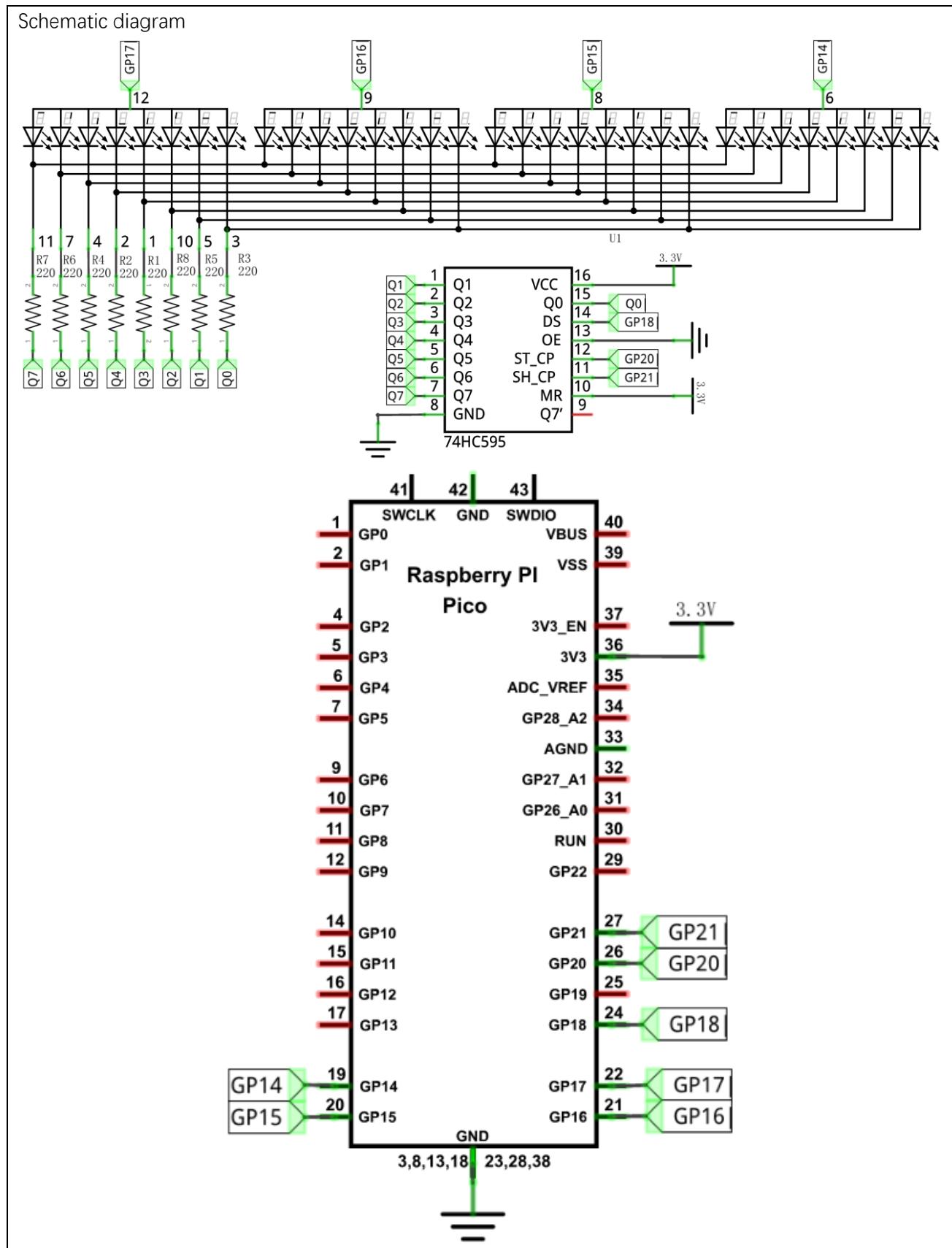
The internal electronic circuit is shown below, and all 8 LED cathode pins of each 7-Segment Display are connected together.



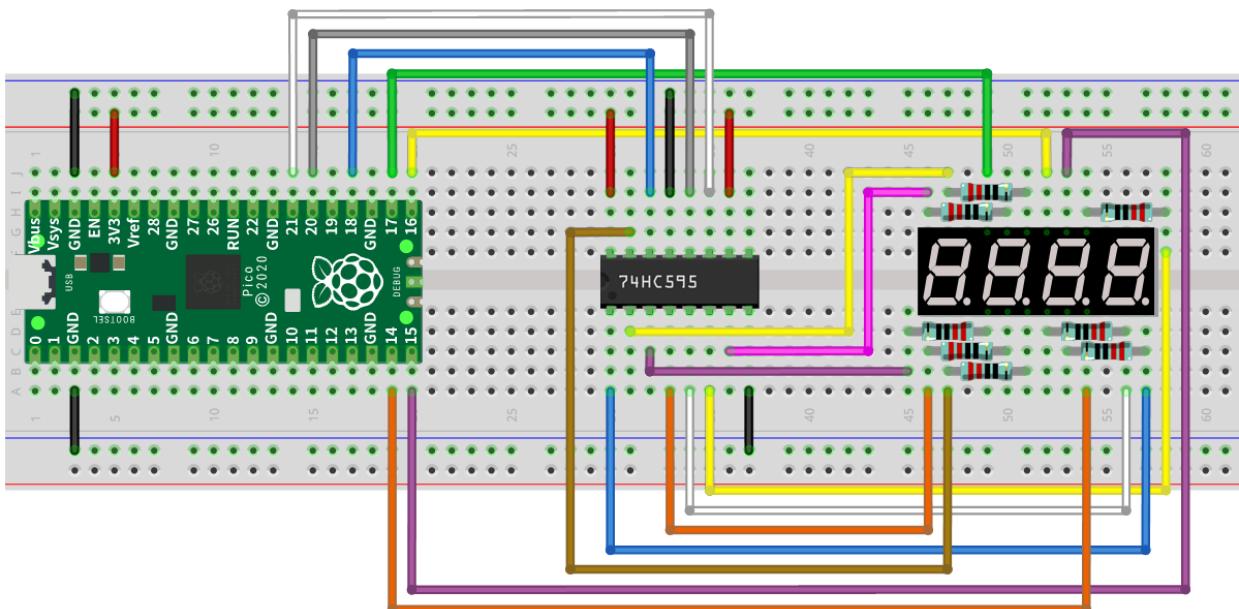
Display method of 4-Digit 7-segment display is similar to 1-Digit 7-segment display. The difference between them is that the 4-Digit displays each Digit visibly in turn, one by one and not together. We need to first send high level to the common end of the first Digit Display, and send low level to the remaining three common ends, and then send content to 8 LED cathode pins of the first Digit Display. At this time, the first 7-Segment Display will show visible content and the remaining three will be OFF.

Similarly, the second, third and fourth 7-Segment Displays will show visible content in turn by scanning the display. Although the four number characters are displayed in turn separately, this process is so fast that it is imperceptible to the naked eye. This is due to the principle of optical afterglow effect and the vision persistence effect in human sight. This is how we can see all 4 number characters at the same time. However, if each number character is displayed for a longer period, you will be able to see that the number characters are displayed separately.

Circuit



Hardware connection. If you need any support, please free to contact us via: support@freenove.com



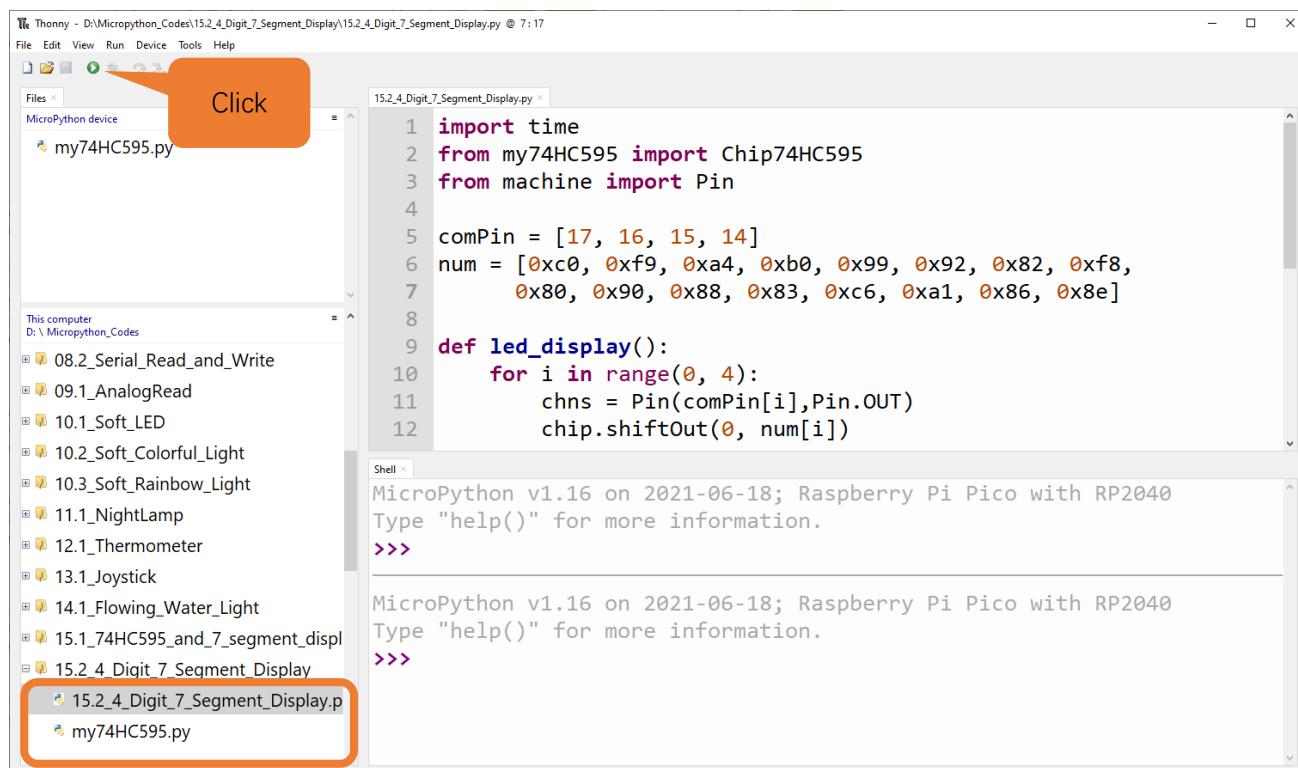
Any concerns? ✉ support@freenove.com

Code

In this code, we use the 74HC595 IC Chip to control the 4-Digit 7-Segment Display, and use the dynamic scanning method to show the changing number characters.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “15.2_4_Digit_7_Segment_Display”. Select “my74HC595.py”, right click your mouse to select “Upload to /”, wait for “my74HC595.py” to be uploaded to Raspberry Pi Pico and double click “15.2_4_Digit_7_Segment_Display.py”.

15.2_4_Digit_7_Segment_Display



Click “Run current script”, and the Nixie tube display as shown in the image below. Press Ctrl+C or click “Stop/Restart backend” to exit the program.



The following is the program code:

```

1 import time
2 from my74HC595 import Chip74HC595
3 from machine import Pin
4
5 comPin = [17, 16, 15, 14]

```

```

6     num = [0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
7         0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e]
8
9     def led_display():
10        for i in range(0, 4):
11            chns = Pin(comPin[i], Pin.OUT)
12            chip.shiftOut(0, num[i])
13            chns.value(1)
14            time.sleep_ms(1)
15            chns.value(0)
16
17    # Pico-GP 18: 74HC595-DS (18)
18    # Pico-GP 20: 74HC595-STCP (20)
19    # Pico-GP 21: 74HC595-SHCP (21)
20    chip = Chip74HC595(18, 20, 21)
21
22    try:
23        while True:
24            led_display()
25    except:
26        pass

```

Import time, my74HC595 and Pin modules.

```

1 import time
2 from my74HC595 import Chip74HC595
3 from machine import Pin

```

Define common anode pins for digital tubes and request a list to put character encodings in it.

```

5     comPin = [17, 16, 15, 14]
6     num = [0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
7         0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e]

```

Request an object to drive 74HC595 and associate pins with it.

```

17   # Pico-GP 18: 74HC595-DS (18)
18   # Pico-GP 20: 74HC595-STCP (20)
19   # Pico-GP 21: 74HC595-SHCP (21)
20   chip = Chip74HC595(18, 20, 21)

```

Make the digital tube display "0 1 2 3".

```

9     def led_display():
10        for i in range(0, 4):
11            chns = Pin(comPin[i], Pin.OUT)
12            chip.shiftOut(0, num[i])
13            chns.value(1)
14            time.sleep_ms(1)
15            chns.value(0)

```

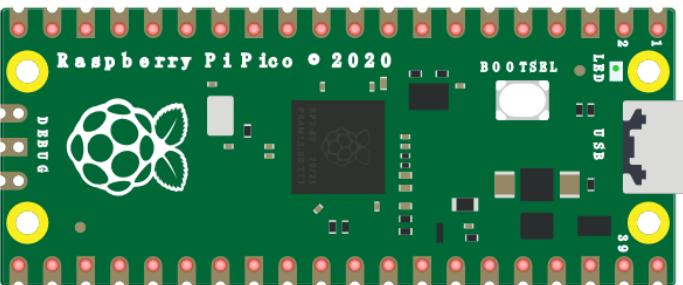
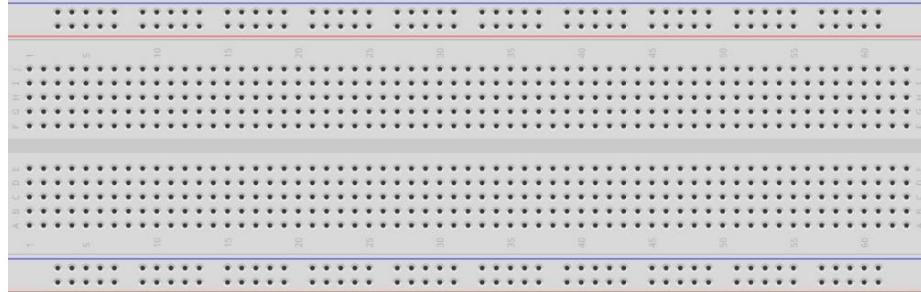
Chapter 16 74HC595 & LED Matrix

Thus far we have learned how to use the 74HC595 IC Chip to control the Bar Graph LED and the 7-Segment Display. We will now use 74HC595 IC Chips to control an LED Matrix.

Project 16.1 LED Matrix

In this project, we will use two 74HC595 IC chips to control a monochrome (one color) (8X8) LED Matrix to make it display both simple graphics and characters.

Component List

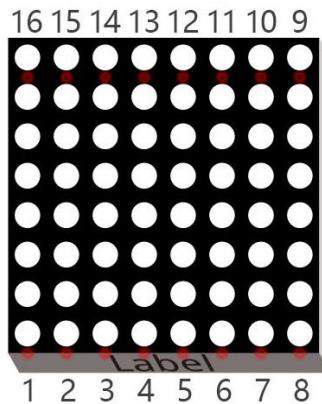
Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
74HC595 x2	8*8 LEDMatrix x1	Resistor 220Ω x8	Jumper



Component knowledge

LED matrix

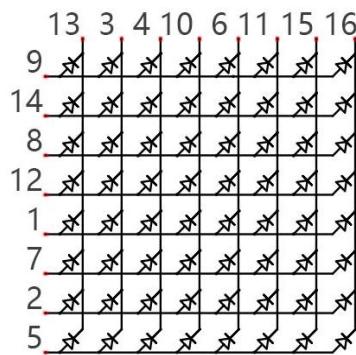
An LED Matrix is a rectangular display module that consists of a uniform grid of LEDs. The following is an 8X8 monochrome (one color) LED Matrix containing 64 LEDs (8 rows by 8 columns).



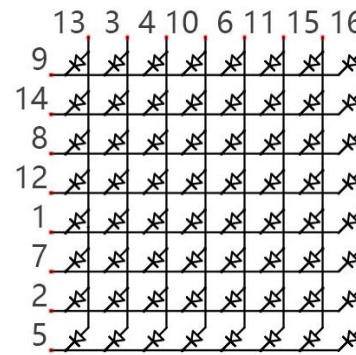
In order to facilitate the operation and reduce the number of ports required to drive this component, the Positive Poles of the LEDs in each row and Negative Poles of the LEDs in each column are respectively connected together inside the LED Matrix module, which is called a Common Anode. There is another arrangement type. Negative Poles of the LEDs in each row and the Positive Poles of the LEDs in each column are respectively connected together, which is called a Common Cathode.

The LED Matrix that we use in this project is a Common Anode LED Matrix.

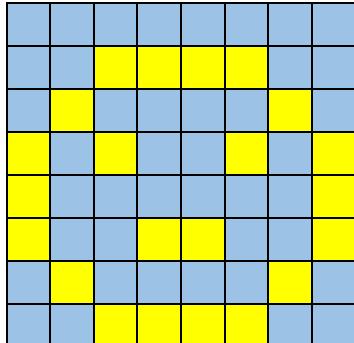
Connection mode of common anode



Connection mode of common cathode



Here is how a Common Anode LED Matrix works. First, choose 16 ports on Raspberry Pi Pico to connect to the 16 ports of LED Matrix. Configure one port in columns for low level, which makes that column the selected port. Then configure the eight port in the row to display content in the selected column. Add a delay value and then select the next column that outputs the corresponding content. This kind of operation by column is called Scan. If you want to display the following image of a smiling face, you can display it in 8 columns, and each column is represented by one byte.



1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	1	0
1	0	1	0	0	1	0	1
1	0	0	0	0	0	0	1
1	0	0	1	1	0	0	1
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0

Column	Binary	Hexadecimal
1	0001 1100	0x1c
2	0010 0010	0x22
3	0101 0001	0x51
4	0100 0101	0x45
5	0100 0101	0x45
6	0101 0001	0x51
7	0010 0010	0x22
8	0001 1100	0x1c

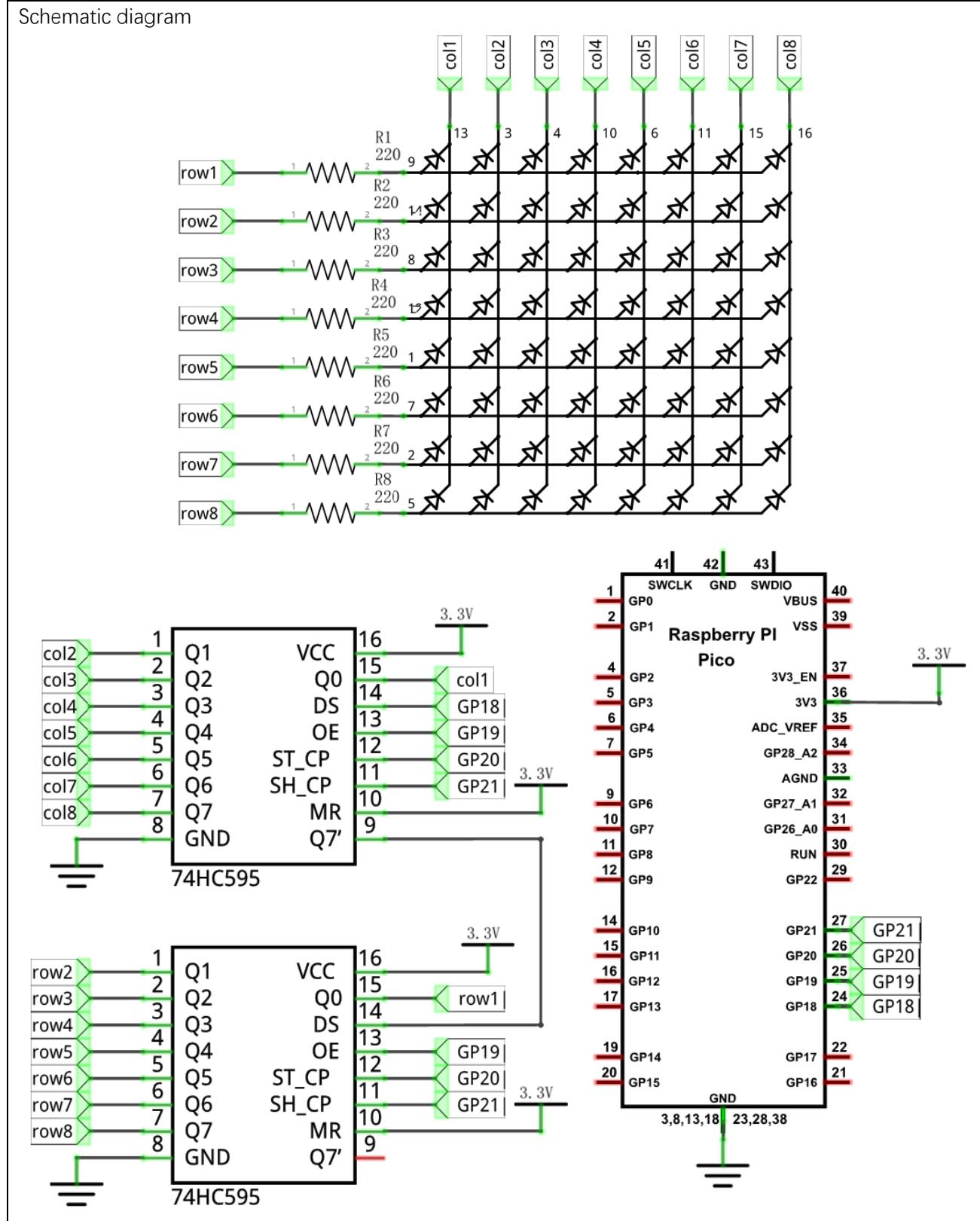
To begin, display the first column, then turn off the first column and display the second column. (and so on) turn off the seventh column and display the 8th column, and then start the process over from the first column again like the control of LED Bar Graph project. The whole process will be repeated rapidly in a loop. Due to the principle of optical afterglow effect and the vision persistence effect in human sight, we will see a picture of a smiling face directly rather than individual columns of LEDs turned ON one column at a time (although in fact this is the reality we cannot perceive).

Then, to save the number of GPIO, we use a 74HC595. When the first column is turned ON, set the lights that need to be displayed in the first column to "1", otherwise to "0", as shown in the above example, where the value of the first column is 0x1c. This value is sent to 74HC595 to control the display of the first column of the LEDMatrix. Following the above idea, turn OFF the display of the first column, then turn ON the second column, and then send the value of the second column to 74HC595 Until each column is displayed, the LEDMatrix is displayed again from the first column.

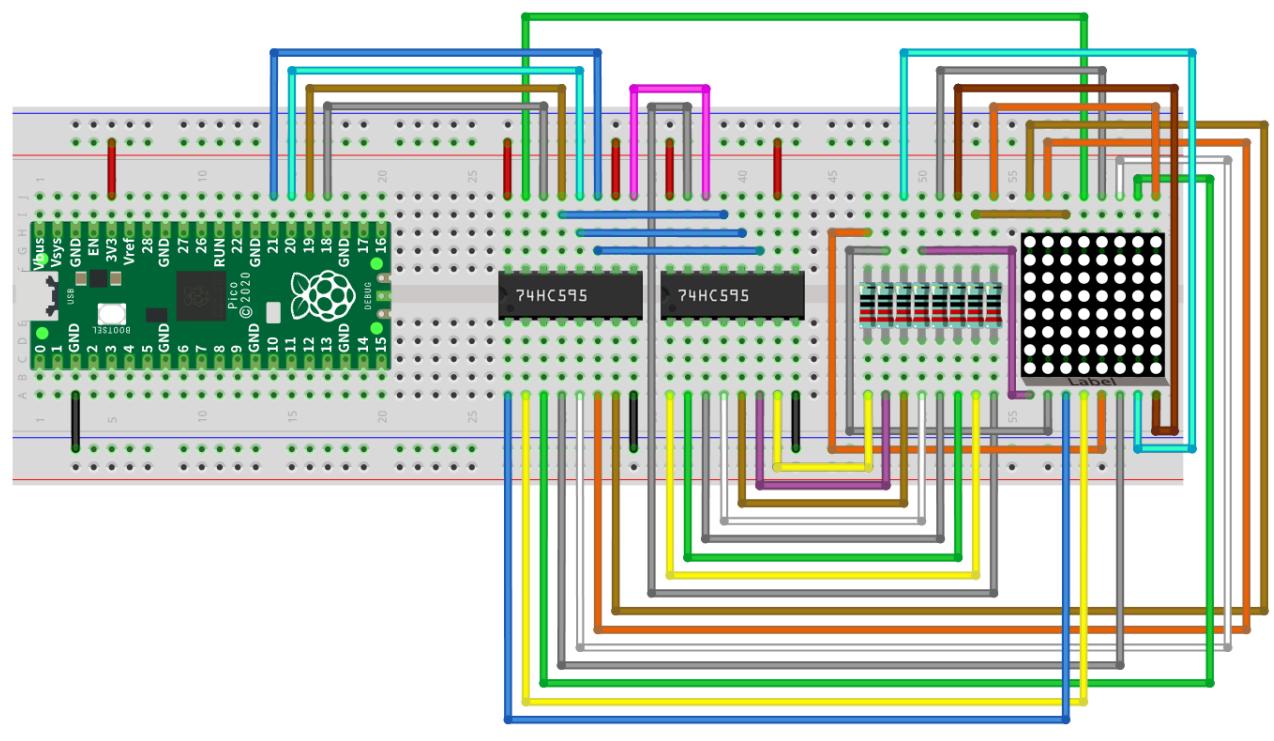
Circuit

In circuit of this project, the power pin of the 74HC595 IC Chip is connected to 3.3V. It can also be connected to 5V to make LED Matrix brighter.

Schematic diagram



Hardware connection. If you need any support, please free to contact us via:support@freenove.com

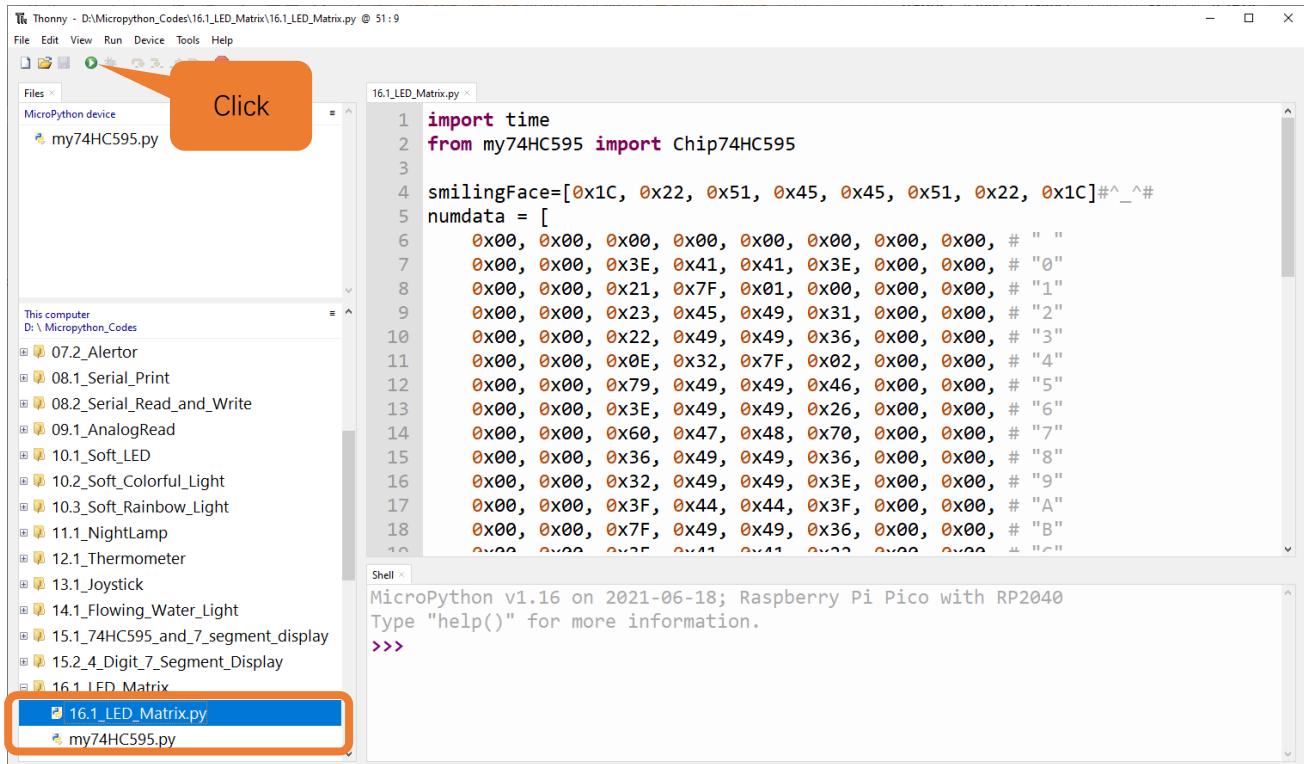


Code

The following code will make LEDMatrix display a smiling face, and then display scrolling character "0-F".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "Micropython_Codes". Select "my74HC595.py", right click your mouse to select "Upload to /", wait for "my74HC595.py" to be uploaded to Raspberry Pi Pico and double click "16.1_LED_Matrix.py".

16.1_LED_Matrix



Click "Run current script", and the LED Matrix display a smiling face, and then display characters "0 to F" scrolling in a loop on the LED Matrix.

The following is the program code:

```

1 import time
2 from my74HC595 import Chip74HC595
3
4 smilingFace=[0x1C, 0x22, 0x51, 0x45, 0x45, 0x51, 0x22, 0x1C]#^_^#
5 numdata = [
6     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, # " "
7     0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, # "0"
8     0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, # "1"
9     0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, # "2"
10    0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, # "3"
11    0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, # "4"
12    0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, # "5"
13    0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, # "6"
14    0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, # "7"
15    0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, # "8"
16    0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, # "9"
17    0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, # "A"
18    0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, # "B"
19    0x00, 0x00, 0x3E, 0x49, 0x49, 0x36, 0x00, 0x00, # "C"
20    0x00, 0x00, 0x3E, 0x49, 0x49, 0x36, 0x00, 0x00, # "D"
21    0x00, 0x00, 0x3E, 0x49, 0x49, 0x36, 0x00, 0x00, # "E"
22    0x00, 0x00, 0x3E, 0x49, 0x49, 0x36, 0x00, 0x00, # "F"

```

```
14     0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, # "7"
15     0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, # "8"
16     0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, # "9"
17     0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, # "A"
18     0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, # "B"
19     0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, # "C"
20     0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, # "D"
21     0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, # "E"
22     0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00, # "F"
23     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, # " "
24 ]
25
26 chip = Chip74HC595(18, 20, 21, 19)
27 try:
28     while True:
29         #smilingFace
30         for j in range(100):
31             cols = 0x01
32             for i in range(8):
33                 chip.disable()
34                 chip.shiftOut(1, smilingFace[i])
35                 chip.shiftOut(0, ~cols)
36                 cols <= 1
37                 chip.enable()
38                 time.sleep_us(500)
39         #numdata
40         for i in range(136):
41             for _ in range(5):
42                 cols = 0x01
43                 for j in range(i, 8+i):
44                     chip.disable()
45                     chip.shiftOut(1, numdata[j])
46                     chip.shiftOut(0, ~cols)
47                     cols <= 1
48                     chip.enable()
49                     time.sleep_us(500)
50     except:
51         pass
```



Import time and my74HC595 modules.

```
1 import time
2 from my74HC595 import Chip74HC595
```

Use a nesting of two for loops to display a smiling face.

```
30 for j in range(100):
31     cols = 0x01
32     for i in range(8):
33         chip.disable()
34         chip.shiftOut(1, smilingFace[i])
35         chip.shiftOut(0, ~cols)
36         cols <= 1
37         chip.enable()
38         time.sleep_us(500)
```

Use a nesting of two for loops to display "0"- "F".

```
40 for i in range(136):
41     for _ in range(5):
42         cols = 0x01
43         for j in range(i, 8+i):
44             chip.disable()
45             chip.shiftOut(1, numdata[j])
46             chip.shiftOut(0, ~cols)
47             cols <= 1
48             chip.enable()
49             time.sleep_us(500)
```

The amount of pins of Raspberry Pi Pico is limited, so we need to find ways to save pins. If we use Raspberry Pi Pico GPIO to control the LEDMatrix instead of 74HC595, we need 16 pins to drive LED matrix. In this example, we use two 74HC595 chips to drive the LED matrix, requiring only three pins, so that we could save the rest of 13 pins.

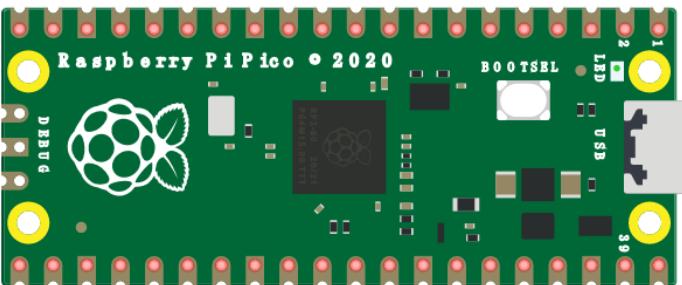
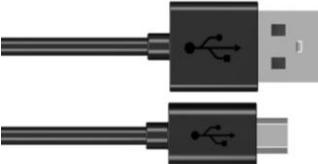
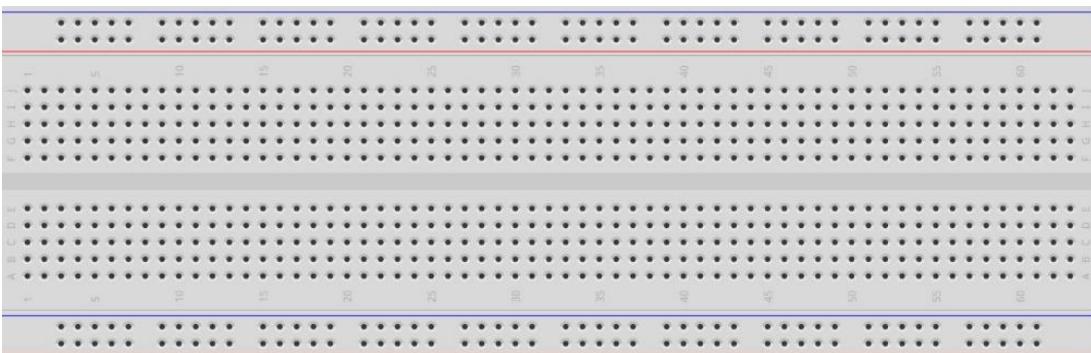
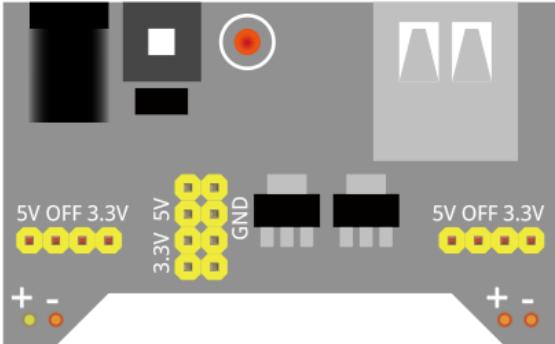
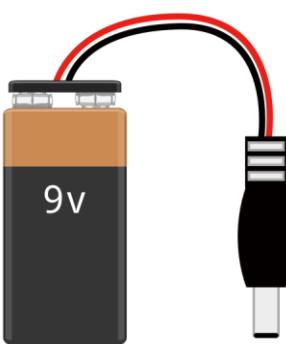
Chapter 17 Relay & Motor

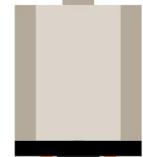
In this chapter, we will learn a kind of special switch module, Relay Module.

Project 17.1 Relay & Motor

In this project, we will use a Push Button Switch indirectly to control the motor via a Relay.

Component List

Raspberry Pi Pico x1	 A green printed circuit board (PCB) for the Raspberry Pi Pico. It features a central Broadcom SoC, four yellow push-pull buttons labeled A, B, C, and D, and a USB Type-A port at the bottom right.	USB cable x1	 Two standard black USB cables, each with a black USB-A male connector and a grey USB-B male connector.
Breadboard x1	 A photograph of a breadboard with numbered pins from 1 to 60 along its top and bottom edges. The breadboard has three main horizontal rows of 40 pins each, with vertical columns labeled A through J on both sides.		
BreadBoardPower x1	 A grey rectangular power module. It has two sets of yellow terminal blocks labeled "5V OFF 3.3V" and "GND" on the left, and two sets on the right. A red circular LED is positioned above the top-right terminal block.	9V battery (prepared by yourself) & battery line x1	 A 9V battery with a black cylindrical body and an orange top, connected to a black battery line with a multi-pin connector.

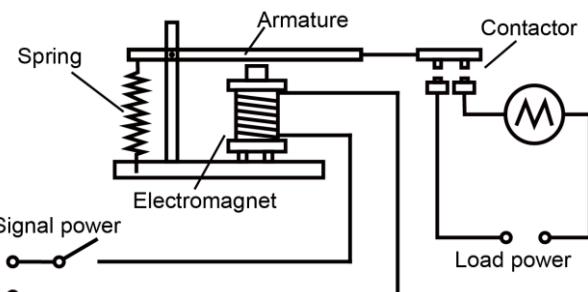
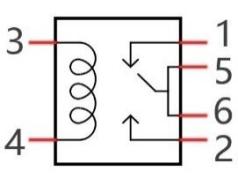
Resistor 10kΩ x2	Resistor 1kΩ x1	Resistor 220Ω x1			
					
NPN transistor x1	Relay x1	Motor x1			
					
Push button x1 LED x1 Diode x1					
Jumper					
					

Component knowledge

Relay

A relay is a safe switch which can use low power circuit to control high power circuit. It consists of electromagnet and contacts. The electromagnet is controlled by low power circuit and contacts are used in high power circuit. When the electromagnet is energized, it will attract contacts.

The following is a schematic diagram of a common relay and the feature and circuit symbol of a 5V relay used in this project:

Diagram	Feature:	Symbol
		

Pin 5 and pin 6 are connected to each other inside. When the coil pins 3 and 4 get connected to 5V power supply, pin 1 will be disconnected from pin 5&6 and pin 2 will be connected to pin 5&6. So pin 1 is called close end, pin 2 is called open end.

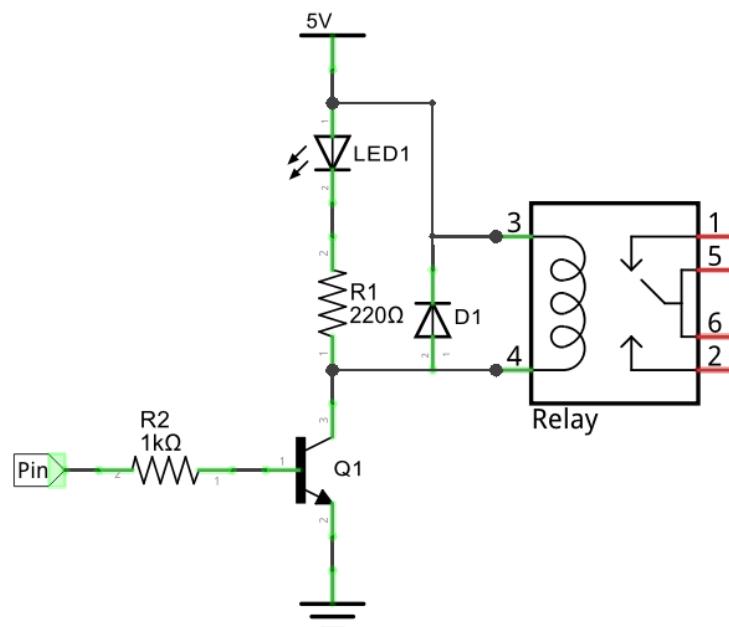
Inductor

The symbol of Inductance is "L" and the unit of inductance is the "Henry" (H). Here is an example of how this can be encountered: $1H=1000mH$, $1mH=1000\mu H$.

An inductor is an energy storage device that converts electrical energy into magnetic energy. Generally, it consists of winding coil, with a certain amount of inductance. Inductors hinder the change of current passing through it. When the current passing through it increases, it will attempt to hinder the increasing trend of current; and when the current passing through it decreases, it will attempt to hinder the decreasing trend of current. So the current passing through inductor is not transient.

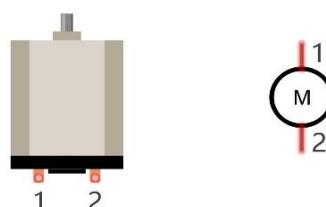


The reference circuit for relay is as follows. The coil of relays can be equivalent to that of inductors, when the transistor disconnects power supply of the relay, the current in the coil of the relay can't stop immediately, causing an impact on power supply. So a parallel diode will get connected to both ends of relay coil pin in reversing direction, then the current will pass through diode, avoiding the impact on power supply.

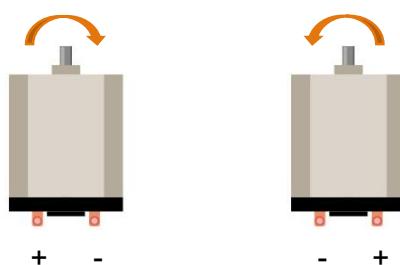


Motor

A motor is a device that converts electrical energy into mechanical energy. Motor consists of two parts: stator and rotor. When motor works, the stationary part is stator, and the rotating part is rotor. Stator is usually the outer case of motor, and it has terminals to connect to the power. Rotor is usually the shaft of motor, and can drive other mechanical devices to run. Diagram below is a small DC motor with two pins.

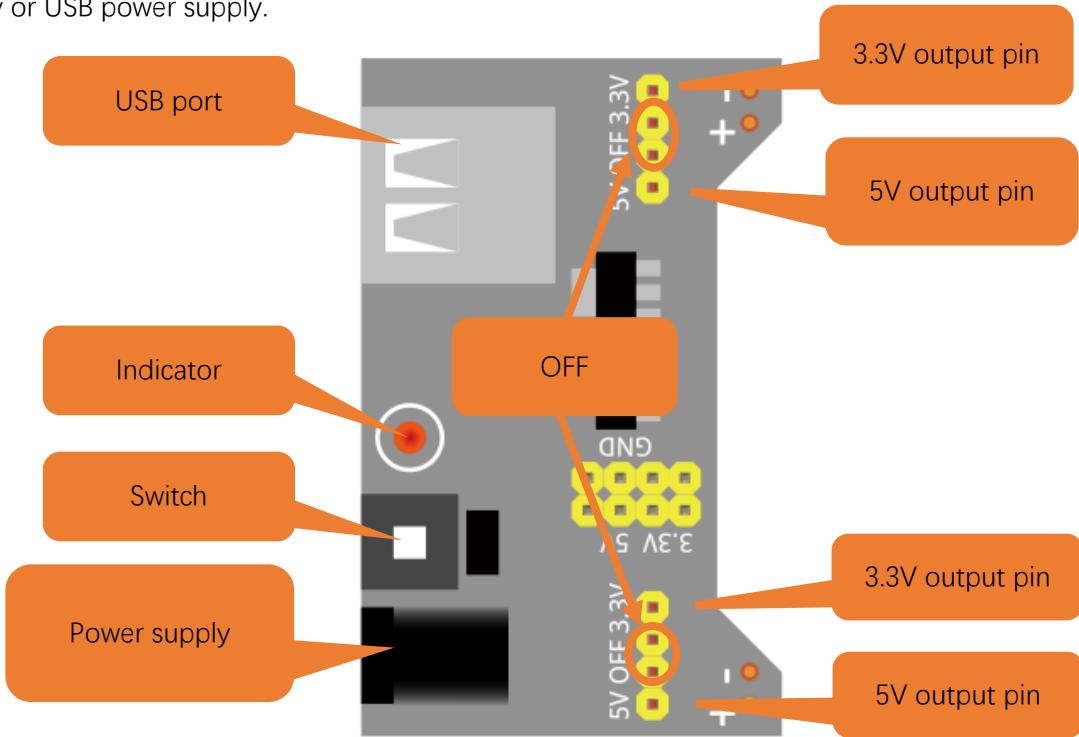


When a motor gets connected to the power supply, it will rotate in one direction. Reverse the polarity of power supply, then the motor rotates in opposite direction.



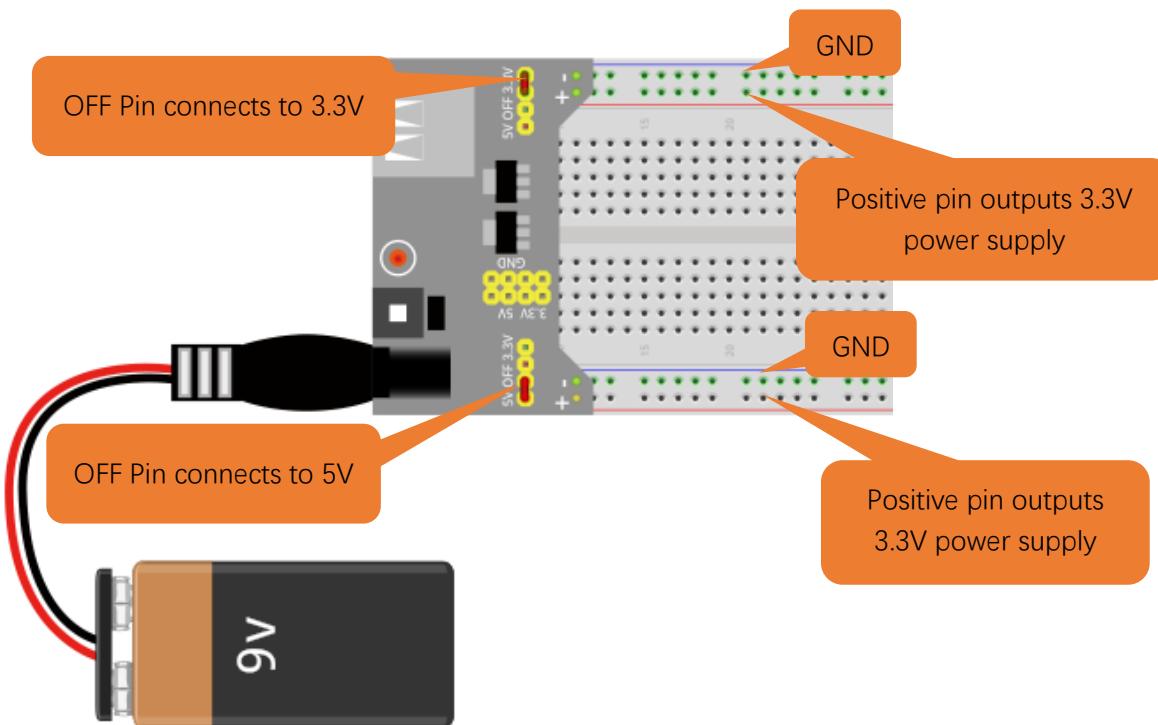
BreadBoardPower

When the Raspberry Pi Pico outputs insufficient power or the power supply voltage and power consumption required by the device or module exceeds that provided by the Raspberry Pi Pico, you can choose BreadBoardPower to either output 3.3V voltage source or 5V voltage source. Input power: DC 6~10V power supply or USB power supply.



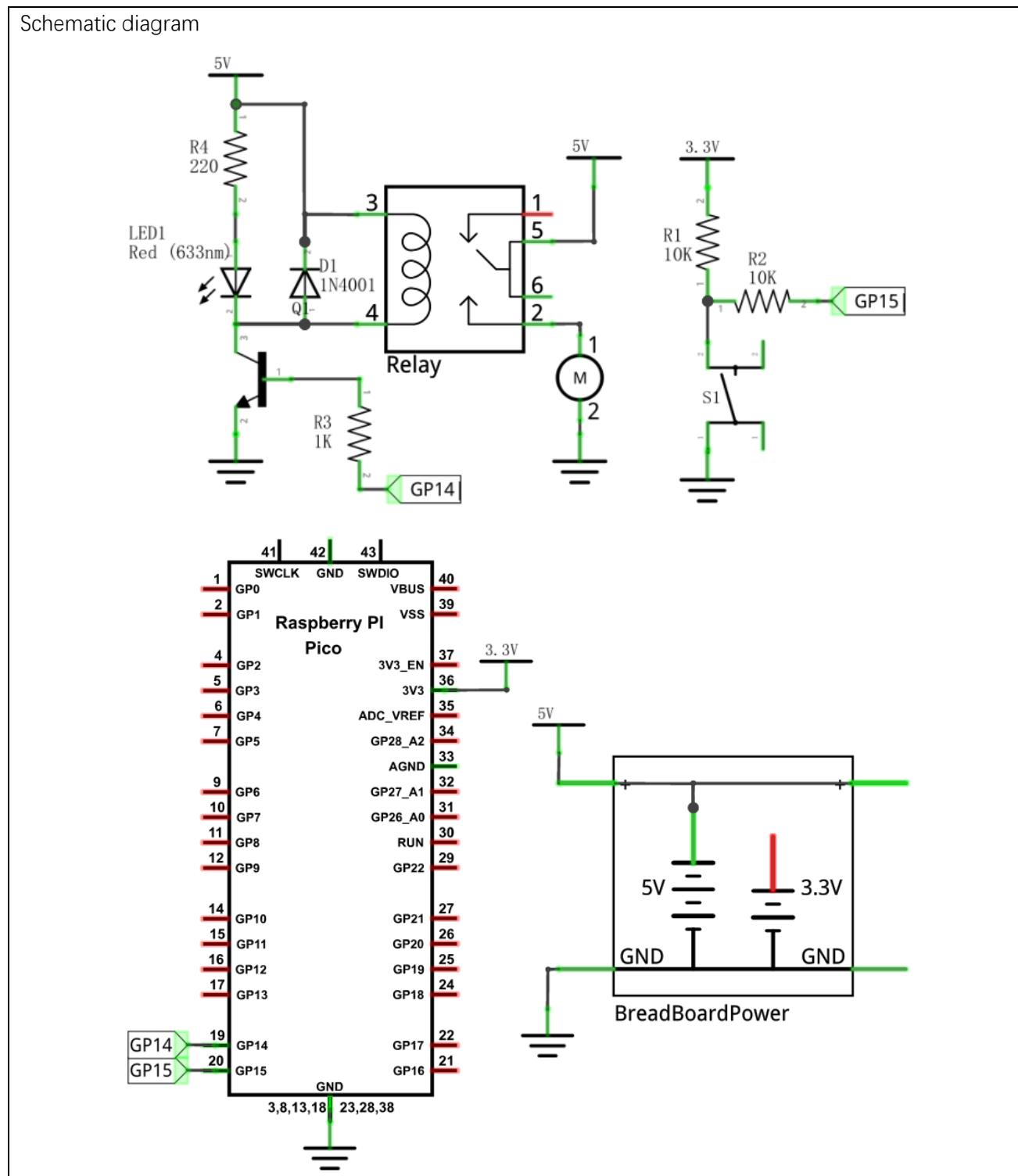
Usage:

First connect BreadBoardPower to the power supply, and then selectively choose the connection between OFF and 5V, 3V to generate different voltage source outputs. as the picture shows.

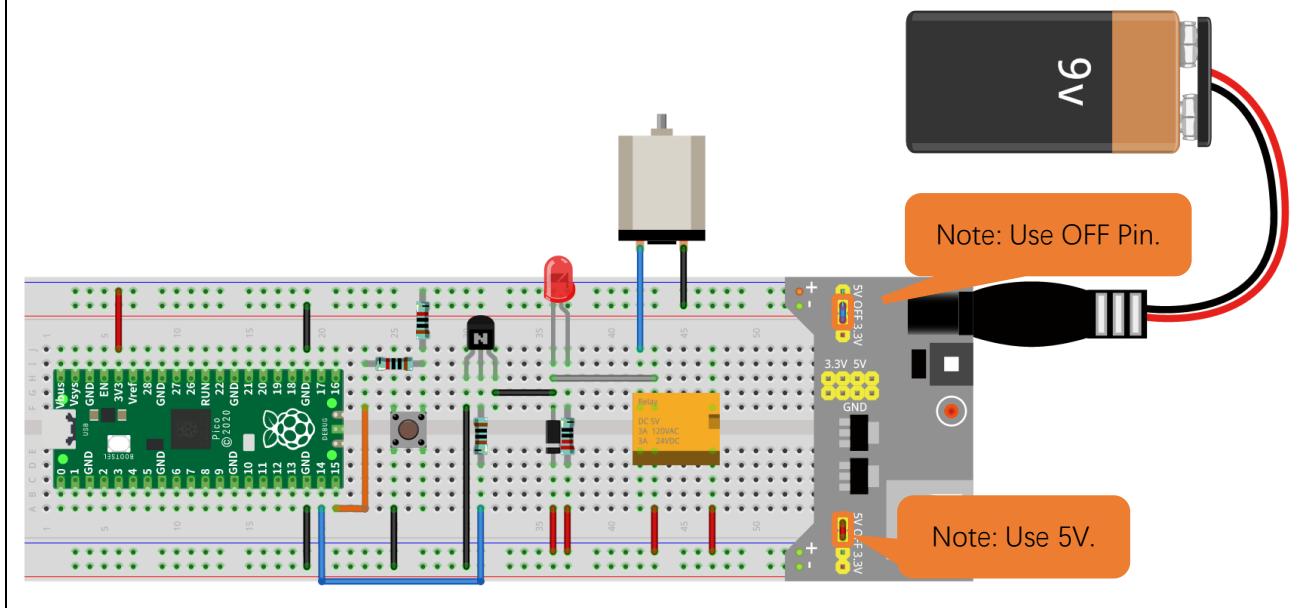


You can either select only one side of the output power supply, or choose to output 5V or 3.3V power supply at the same time on both sides, because the output circuits on both sides are separated and controlled independently and will not cause interference. Note that the GND should be connected to any GND of the Raspberry Pi Pico to form a Common ground connection.

Circuit



Hardware connection. If you need any support, please free to contact us via: support@freenove.com



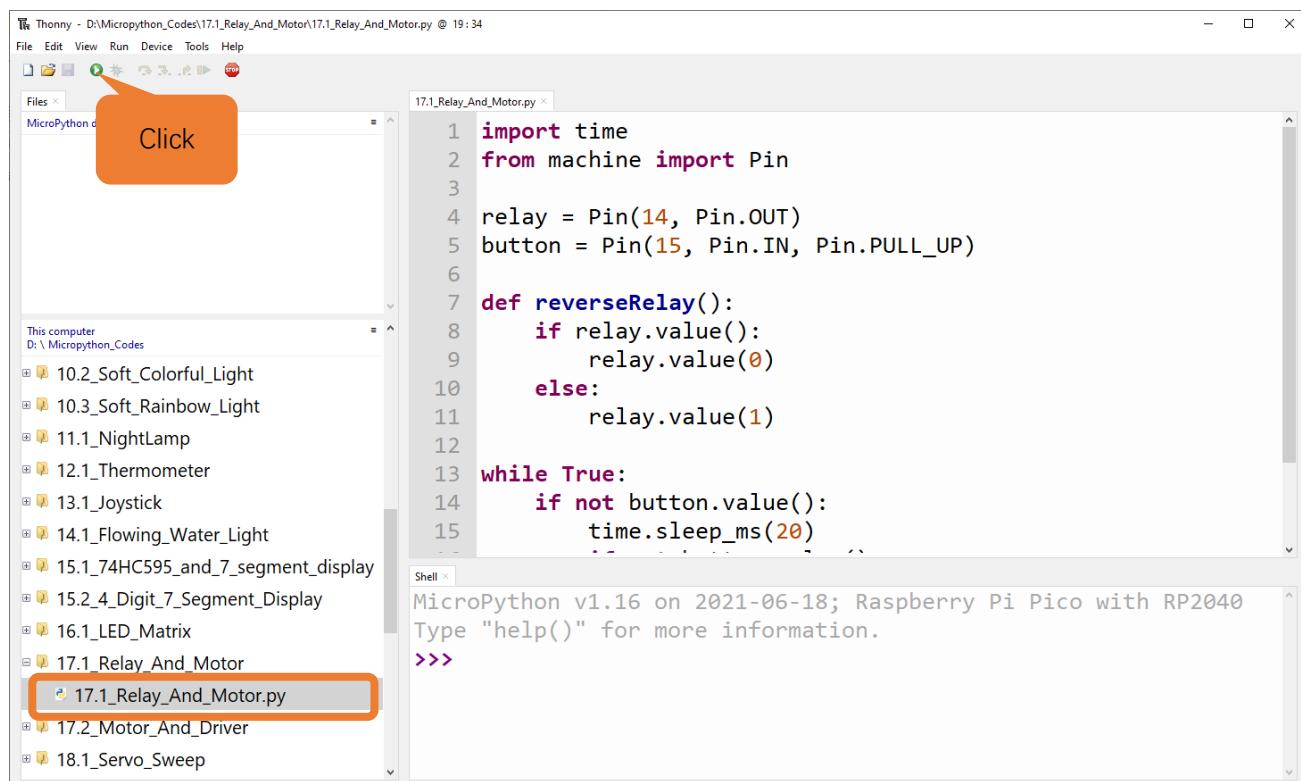
Any concerns? ✉ support@freenove.com

Code

Use buttons to control the relays and motors.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “17.1_Relay_And_Motor” and double click “17.1_Relay_And_Motor.py”.

17.1_Relay_And_Motor



```

import time
from machine import Pin

relay = Pin(14, Pin.OUT)
button = Pin(15, Pin.IN, Pin.PULL_UP)

def reverseRelay():
    if relay.value():
        relay.value(0)
    else:
        relay.value(1)

while True:
    if not button.value():
        time.sleep_ms(20)
    else:
        reverseRelay()

```

The code above is a Python script for MicroPython running on a Raspberry Pi Pico. It uses the `machine` module to control a relay connected to pin 14 and a push-button connected to pin 15. The script defines a function `reverseRelay()` that toggles the state of the relay. It then enters a loop where it checks the button's value every 20ms. If the button is pressed (value is low), it calls `time.sleep_ms(20)`. If the button is released (value is high), it calls `reverseRelay()`, which toggles the relay's state between high (0) and low (1).

Click “Run current script”. When the DC Motor is connected to a power supply, it will rotate in one direction. If you reverse the polarity of the power supply, the DC Motor will rotate in opposite direction. Press Ctrl+C or click “Stop/Restart backend” to exit the program.





The following is the program code:

```
1 import time
2 from machine import Pin
3
4 relay = Pin(14, Pin.OUT)
5 button = Pin(15, Pin.IN, Pin.PULL_UP)
6
7 def reverseRelay():
8     if relay.value():
9         relay.value(0)
10    else:
11        relay.value(1)
12
13 while True:
14     if not button.value():
15         time.sleep_ms(20)
16         if not button.value():
17             reverseRelay()
18             while not button.value():
19                 time.sleep_ms(20)
```

This section of code is basically the same as that of project Tablelamp. If you don't understand the program, you can click [here](#) to go back to the Tablelamp and study again.

Chapter 18 L293D & Motor

Project 18.1 Control Motor with Potentiometer

Control the direction and speed of the motor with a potentiometer.

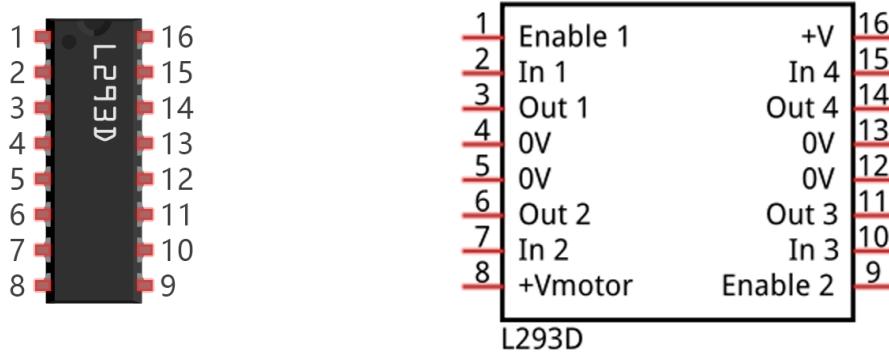
Component List

Raspberry Pi Pico x1	USB cable x1
Breadboard x1	
Rotary potentiometer x1	Motor x1
L293D x1	
Jumper	Battery box x1

Component knowledge

L293D

L293D is an IC Chip (Integrated Circuit Chip) with a 4-channel motor drive. You can drive a Unidirectional DC Motor with 4 ports or a Bi-Directional DC Motor with 2 ports or a Stepper Motor (Stepper Motors are covered later in this Tutorial).



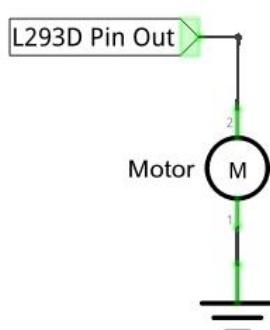
Port description of L293D module is as follows:

Pin name	Pin number	Description
In x	2, 7, 10, 15	Channel x digital signal input pin
Out x	3, 6, 11, 14	Channel x output pin, input high or low level according to In x pin, get connected to +Vmotor or 0V
Enable1	1	Channel 1 and channel 2 enable pin, high level enable
Enable2	9	Channel 3 and channel 4 enable pin, high level enable
0V	4, 5, 12, 13	Power cathode (GND)
+V	16	Positive electrode (VCC) of power supply, supply voltage 3.0~36V
+Vmotor	8	Positive electrode of load power supply, provide power supply for the Out pin x, the supply voltage is +V~36V

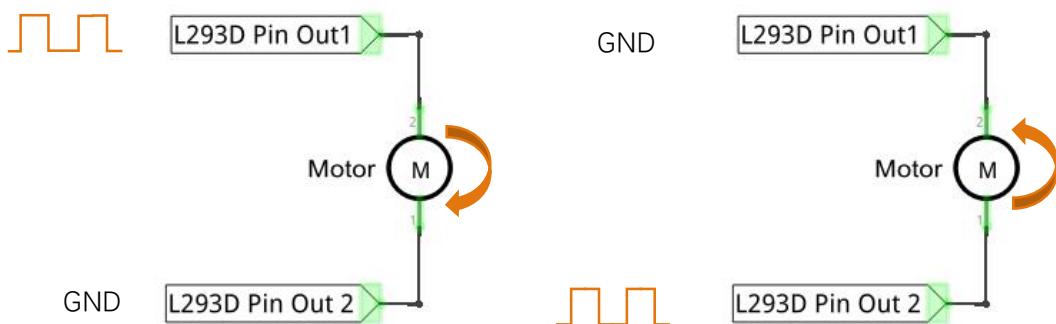
For more details, please refer to the datasheet for this IC Chip.

When using L293D to drive DC motor, there are usually two connection options.

The following connection option uses one channel of the L293D, which can control motor speed through the PWM. However the motor then can only rotate in one direction.



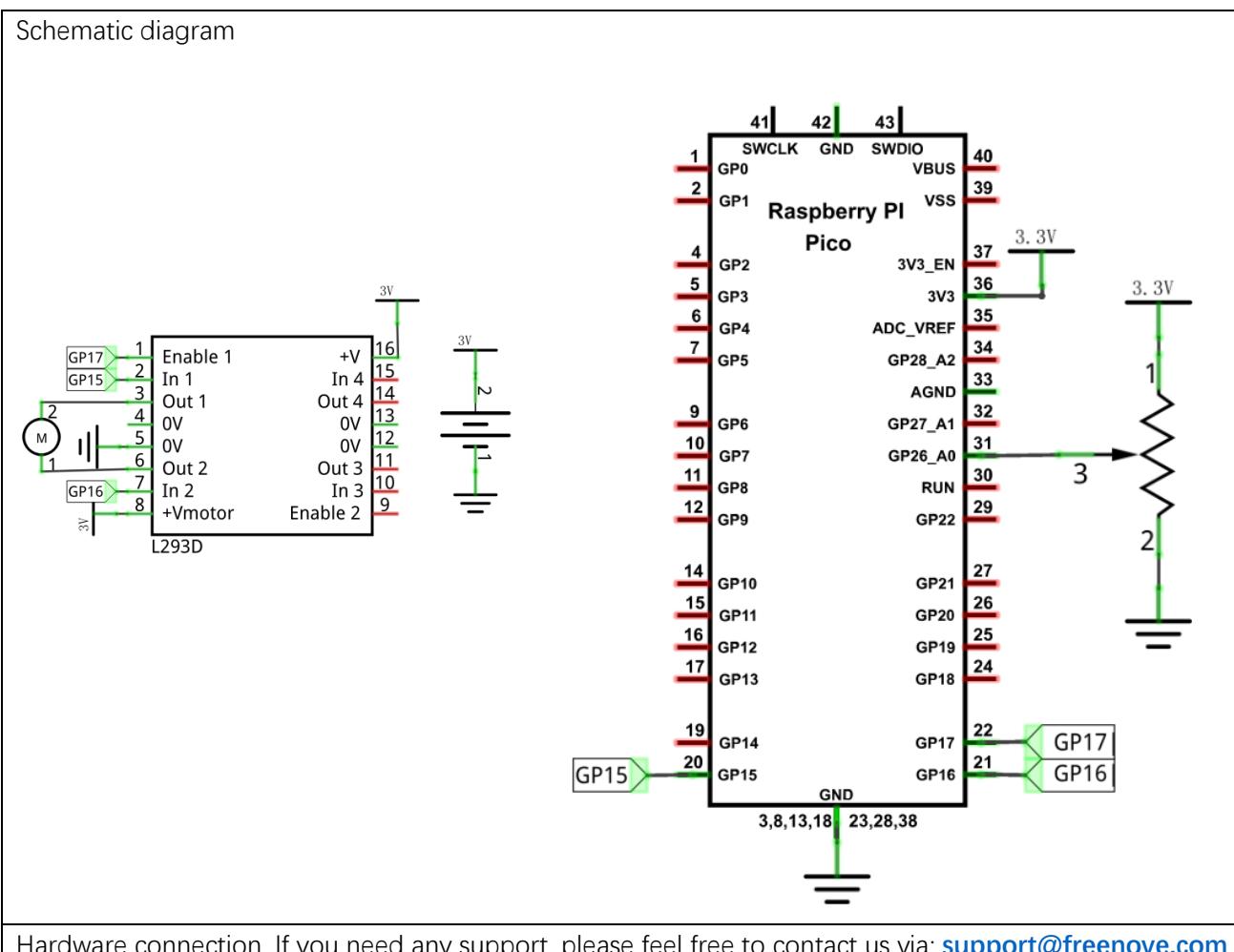
The following connection uses two channels of the L293D: one channel outputs the PWM wave, and the other channel connects to GND, therefore you can control the speed of the motor. When these two channel signals are exchanged, not only can they control the speed of motor, but also control the direction of the motor.

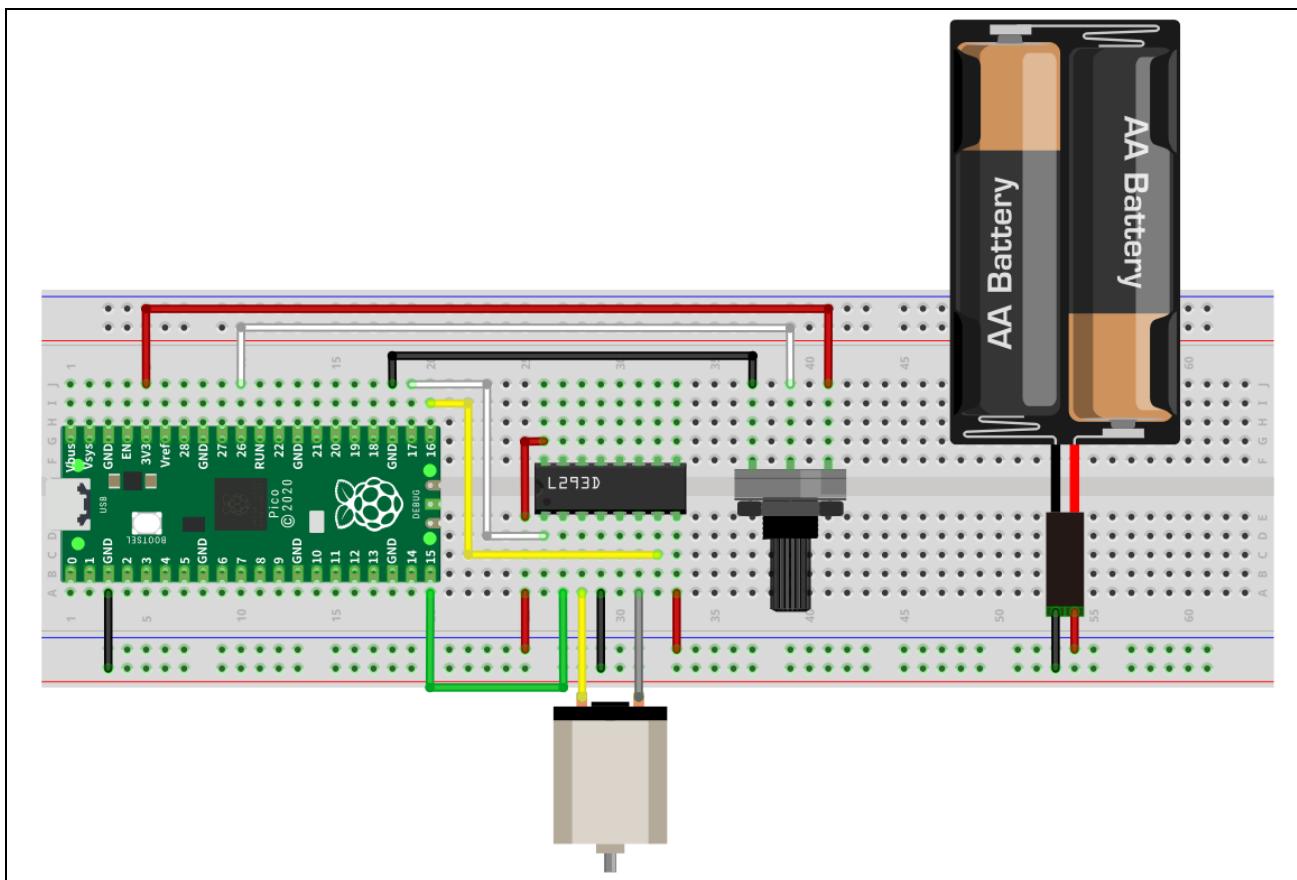


In practical use the motor is usually connected to channels 1 and 2 by outputting different levels to in1 and in2 to control the rotational direction of the motor, and output to the PWM wave to Enable1 port to control the motor's rotational speed. If the motor is connected to channel 3 and 4 by outputting different levels to in3 and in4 to control the motor's rotation direction, and output to the PWM wave to Enable2 pin to control the motor's rotational speed.

Circuit

Schematic diagram





Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “18.1_Motor_And_Driver” and double click “18.1_Motor_And_Driver.py”.

18.1_Motor_And_Driver

The screenshot shows the Thonny IDE interface. On the left, there's a tree view of files under "This computer" and "D:\Python_Codes". A red box highlights the "18.1_Motor_And_Driver" folder, and another red box highlights the "18.1_Motor_And_Driver.py" file within it. A large orange arrow points from the text "Click" to the "18.1_Motor_And_Driver.py" file. The main window displays the Python code for "18.1_Motor_And_Driver.py". The code uses the machine module to control pins 15, 16, and 17 via PWM and ADC to drive a motor based on potentiometer input. The code is as follows:

```
from machine import ADC, Pin, PWM
import time
import math

in1Pin = Pin(15, Pin.OUT)
in2Pin = Pin(16, Pin.OUT)
pwm = PWM(Pin(17)) #enable(GP17)
pwm.freq(1000)
pwm.duty_u16(512)
adc = ADC(26)

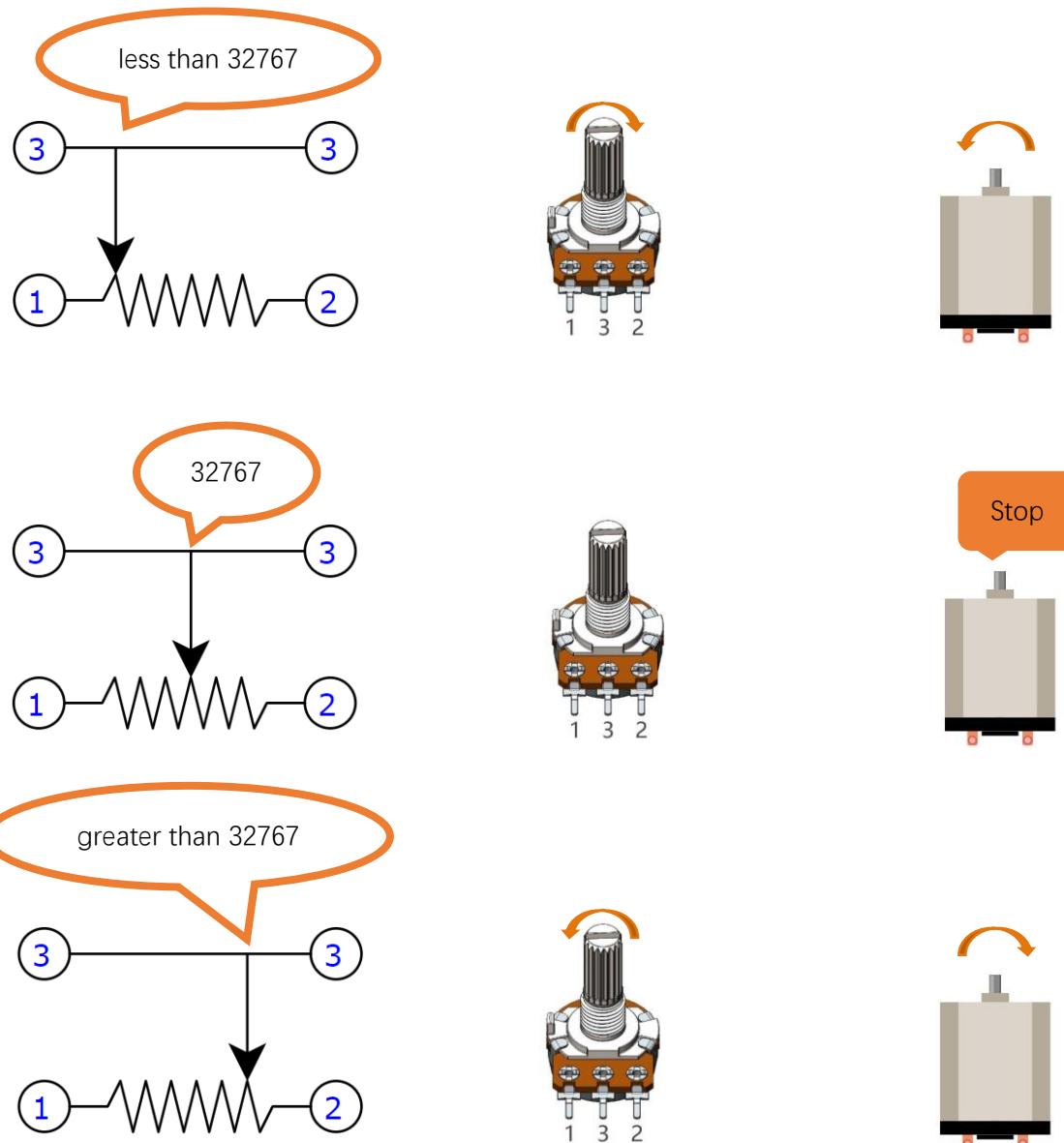
def driveMotor(dir, spd):
    if dir:
        in1Pin.value(1)
        in2Pin.value(0)
    else :
        in1Pin.value(0)
        in2Pin.value(1)
    pwm.duty_u16(spd)

try:
    while True:
        potenVal = adc.read_u16()
        rotationSpeed = potenVal - 32767
        if (potenVal > 32767):
            rotationDir = 1
        else:
            rotationDir = -1
        driveMotor(rotationDir, rotationSpeed)
        time.sleep(0.01)

```

The bottom shell window shows the MicroPython version and hardware information: "MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040".

Click “Run current script”, rotate the potentiometer in one direction and the motor speeds up slowly in one direction. Rotate the potentiometer in the other direction and the motor will slow down to stop. And then rotate it in the original direction to accelerate the motor. Press Ctrl+C or click “Stop/Restart backend” to exit the program.



The following is the Code:

```

1  from machine import ADC, Pin, PWM
2  import time
3  import math
4
5  in1Pin = Pin(15, Pin.OUT)
6  in2Pin = Pin(16, Pin.OUT)
7  pwm = PWM(Pin(17)) #enable(GP17)
8  pwm.freq(1000)
9  pwm.duty_u16(512)
10 adc = ADC(26)
11
12 def driveMotor(dir, spd):
13     if dir:
14         in1Pin.value(1)
15         in2Pin.value(0)
16     else :
17         in1Pin.value(0)
18         in2Pin.value(1)
19     pwm.duty_u16(spd)
20
21 try:
22     while True:
23         potenVal = adc.read_u16()
24         rotationSpeed = potenVal - 32767
25         if (potenVal > 32767):
26             rotationDir = 1;
27         else:
28             rotationDir = 0;
29         rotationSpeed = int(math.fabs((potenVal-32767) * 2) - 1)
30         driveMotor(rotationDir, rotationSpeed)
31         time.sleep_ms(10)
32     except:
33         pwm.deinit()

```

In the program, we will define 32767 as the intermediate point to adjust the potentiometer. When the ADC value is less than 32767, the motor rotates in one direction; when the ADC value is greater than 32767, the motor rotates in the opposite direction. The speed of the motor will change with the adjustment of the potentiometer. When the potentiometer is adjusted, the closer the ADC value is to the middle point, the slower the motor speed; the closer the ADC value is to 0 or 65535, the faster the motor speed.

```

22     while True:
23         potenVal = adc.read_u16()
24         rotationSpeed = potenVal - 32767
25         if (potenVal > 32767):
26             rotationDir = 1;

```

Any concerns? ✉ support@freenove.com

```
27     else:  
28         rotationDir = 0;  
29         rotationSpeed = int(math.fabs((potenVal-32767) * 2) - 1)  
30         driveMotor(rotationDir, rotationSpeed)  
31         time.sleep_ms(10)
```

Initialize pins of L293D chip.

```
5     in1Pin = Pin(15, Pin.OUT)  
6     in2Pin = Pin(16, Pin.OUT)  
7     pwm = PWM(Pin(17)) #enable -> GP 17  
8     pwm.freq(1000)  
9     pwm.duty_u16(512)
```

Function `driveMotor` is used to control the rotation direction and speed of the motor. The `dir` represents direction while `spd` refers to speed.

```
12 def driveMotor(dir, spd):  
13     if dir:  
14         in1Pin.value(1)  
15         in2Pin.value(0)  
16     else :  
17         in1Pin.value(0)  
18         in2Pin.value(1)  
19     pwm.duty_u16(spd)
```



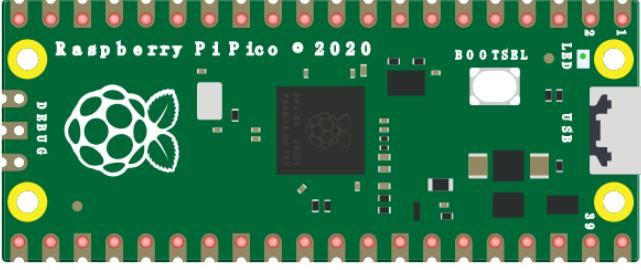
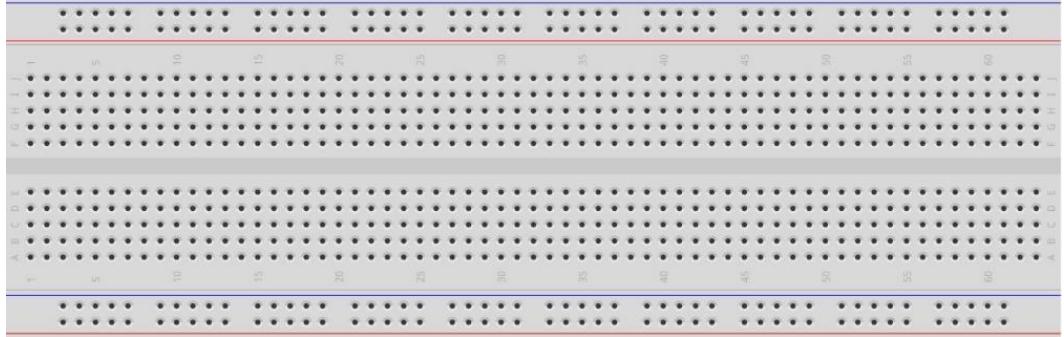
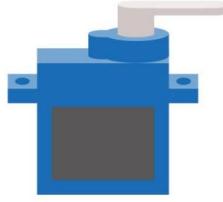
Chapter 19 Servo

Previously, we learned how to control the speed and rotational direction of a Motor. In this chapter, we will learn about Servos which are a rotary actuator type motor that can be controlled to rotate to specific angles.

Project 19.1 Servo Sweep

First, we need to learn how to make a Servo rotate.

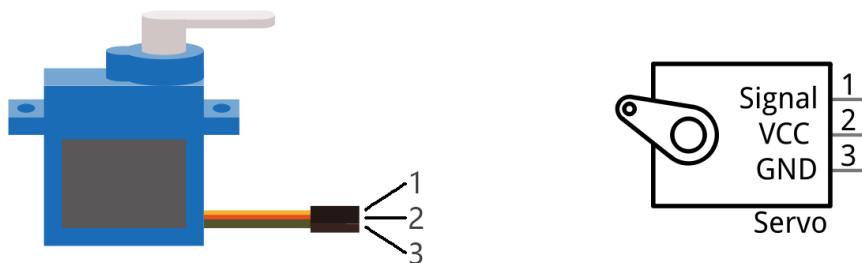
Component List

Raspberry Pi Pico x1	 A green printed circuit board (PCB) for the Raspberry Pi Pico. It features a central Broadcom SoC, a USB-C port, and various pins and connectors. The text "Raspberry Pi Pico • 2020" is printed on the board.	USB cable x1	 Two standard black USB-A to USB-A cables, each with a grey plastic housing.
Breadboard x1	 A schematic diagram of a breadboard. It shows a grid of 40 columns and 6 rows of holes. The columns are labeled from 1 to 40 along the bottom. The rows are labeled A, B, C, D, E, F, G, H, I, J along the left side. The center row is red, and the other five rows are grey.		
Servo x1	 A blue servo motor with a black rectangular body and a light-colored plastic horn at the end of the shaft.	Jumper	 A single black jumper wire with two small black plastic caps on the ends.

Component knowledge

Servo

Servo is a compact package which consists of a DC Motor, a set of reduction gears to provide torque, a sensor and control circuit board. Most Servos only have a 180-degree range of motion via their "horn". Servos can output higher torque than a simple DC Motor alone and they are widely used to control motion in model cars, model airplanes, robots, etc. Servos have three wire leads which usually terminate to a male or female 3-pin plug. Two leads are for electric power: Positive (2-VCC, Red wire), Negative (3-GND, Brown wire), and the signal line (1-Signal, Orange wire) as represented in the Servo provided in your Kit.



We will use a 50Hz PWM signal with a duty cycle in a certain range to drive the Servo. The time interval of 0.5ms-2.5ms of PWM single cycle high level corresponds to the Servo angle 0 degrees - 180 degrees linearly. Part of the corresponding values are as follows:

High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	0 degree
2ms	45 degree
2.5ms	180 degree

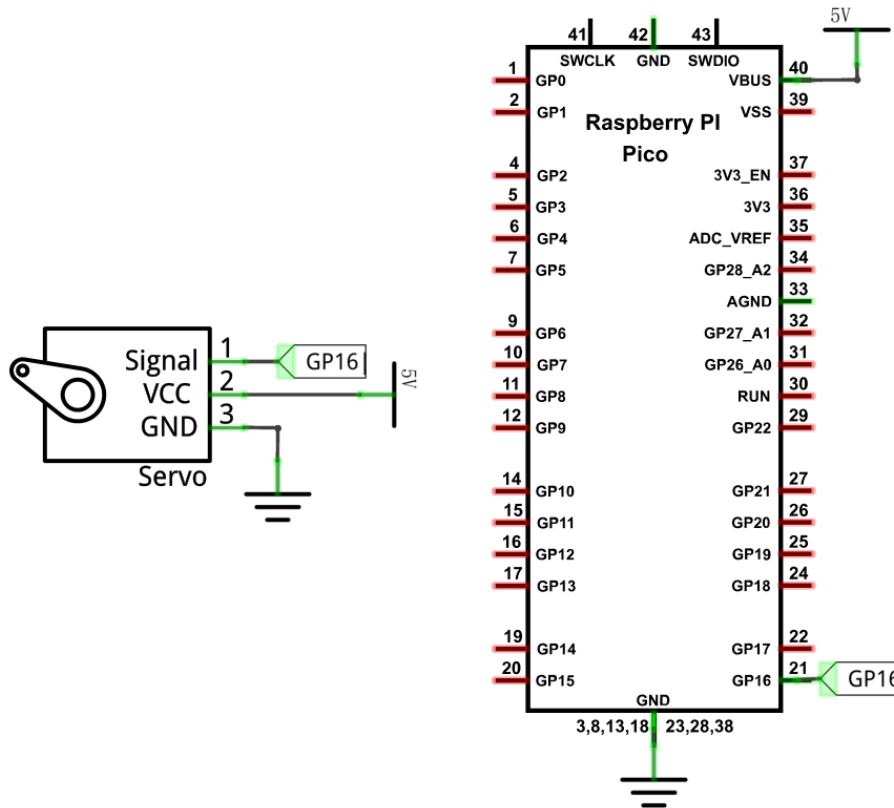
When you change the Servo signal value, the Servo will rotate to the designated angle.



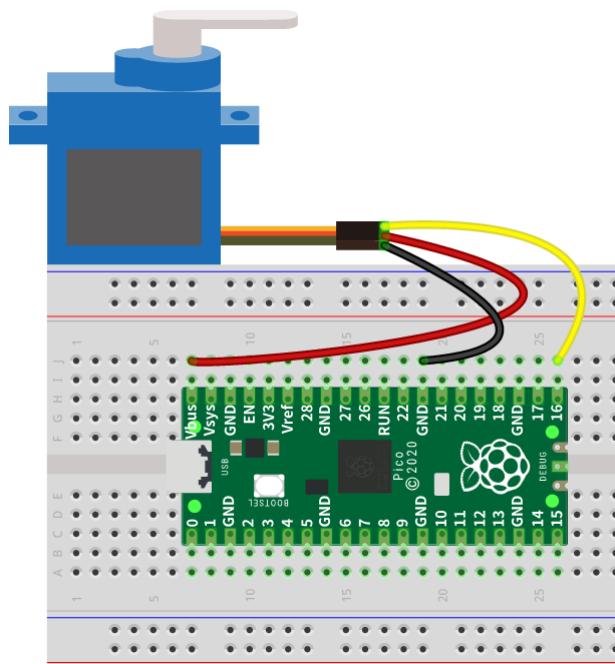
Circuit

Use caution when supplying power to the Servo, it should be 5V. Make sure you do not make any errors when connecting the Servo to the power supply.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Any concerns? ✉ support@freenove.com

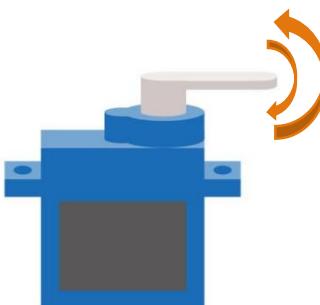
Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “19.1_Servo_Sweep”. Select “myservo.py”, right click your mouse to select “Upload to /”, wait for “myservo.py” to be uploaded to Raspberry Pi Pico and then double click “19.1_Servo_Sweep.py”.

19.1_Servo_Sweep



Click “Run current script”, the Servo will rotate from 0 degrees to 180 degrees and then reverse the direction to make it rotate from 180 degrees to 0 degrees and repeat these actions in an endless loop. Press Ctrl+C or click “Stop/Restart backend” to exit the program.



The following is the program code:

```

1 from myservo import Servo
2 import time
3
4 servo=Servo(16)
5 servo.ServoAngle(0)
6 time.sleep_ms(1000)
7
8 try:

```

```

9     while True:
10    for i in range(0, 180, 1):
11        servo.ServoAngle(i)
12        time.sleep_ms(15)
13    for i in range(180, 0, -1):
14        servo.ServoAngle(i)
15        time.sleep_ms(15)
16 except:
17     servo.deinit()

```

Import myservo module.

```
1 from myservo import Servo
```

Initialize pins of the servo and set the starting point of the servo to 0 degree.

```

4 servo=Servo(16)
5 servo.ServoAngle(0)
6 time.sleep_ms(1000)

```

Control the servo to rotate to a specified angle within the range of 0-180 degrees.

```
9 servo.ServoAngle(i)
```

Use two for loops. The first one controls the servo to rotate from 0 degree to 180 degrees while the other controls it to rotate back from 180 degrees to 0 degree.

```

10   for i in range(0, 180, 1):
11       servo.ServoAngle(i)
12       time.sleep_ms(15)
13   for i in range(180, 0, -1):
14       servo.ServoAngle(i)
15       time.sleep_ms(15)

```

Reference

```
class myServo
```

Before each use of **Servo**, please make sure myservo.py has been uploaded to “/” of Raspberry Pi Pico, and then add the statement “**from myservo import Servo**” to the top of the python file.

Servo(): The object that controls the servo, with the default pin GP15, default frequency 50Hz.

ServoDuty(duty): The function controls the servo's rotating angle through the duty cycle.

duty: Ranges from 1638 to 8190, with 1638 corresponding to the servo's 0 degree and 8190 corresponding to 180 degrees.

ServoAngle(pos): The function passes in pos(angle) value to control the servo's rotating angle.

pos: Ranging from 0-180, corresponding the 0-180 degrees of the servo.

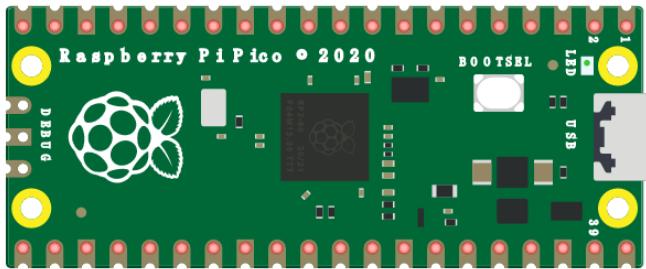
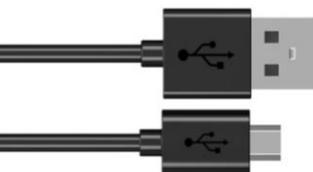
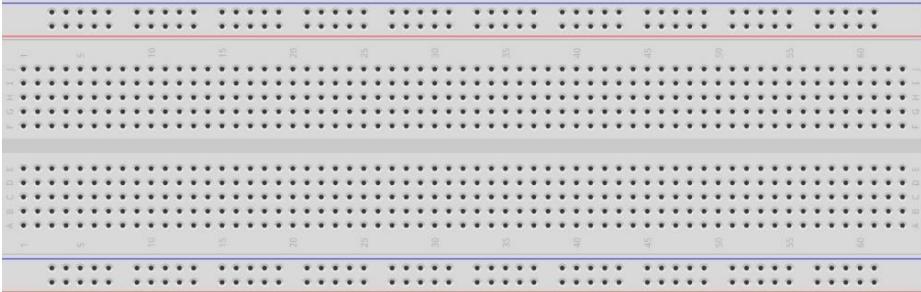
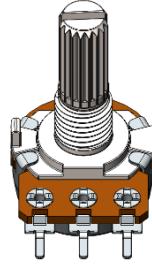
ServoTime(us): The function passes in us(time) to control the servo's rotating angle.

us: Ranges from 500-2500, with 500 corresponding to the servo's 0 degree and 2500 corresponding to 180 degrees.

Project 19.2 Servo Knob

Use a potentiometer to control the servo motor to rotate at any angle.

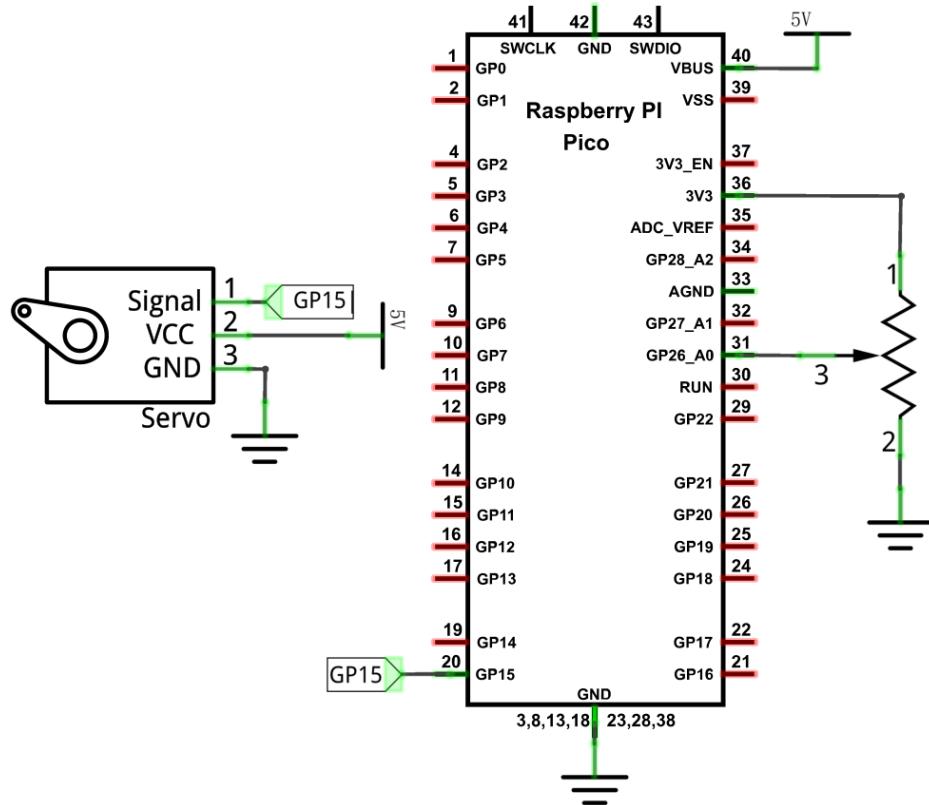
Component List

Raspberry Pi Pico x1	USB cable x1	
		
Breadboard x1		
		
Servo x1	Jumper	Rotary potentiometer x1
		

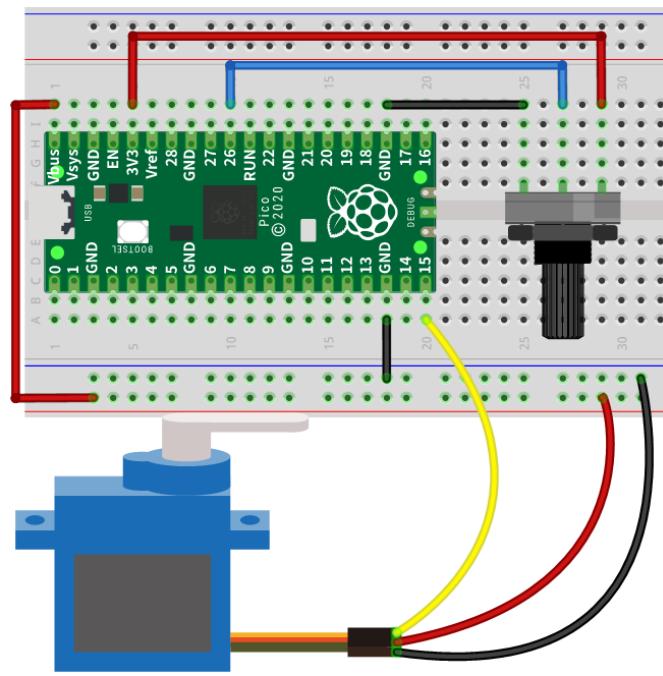
Circuit

Use caution when supplying power to the Servo, it should be 5V. Make sure you do not make any errors when connecting the Servo to the power supply.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

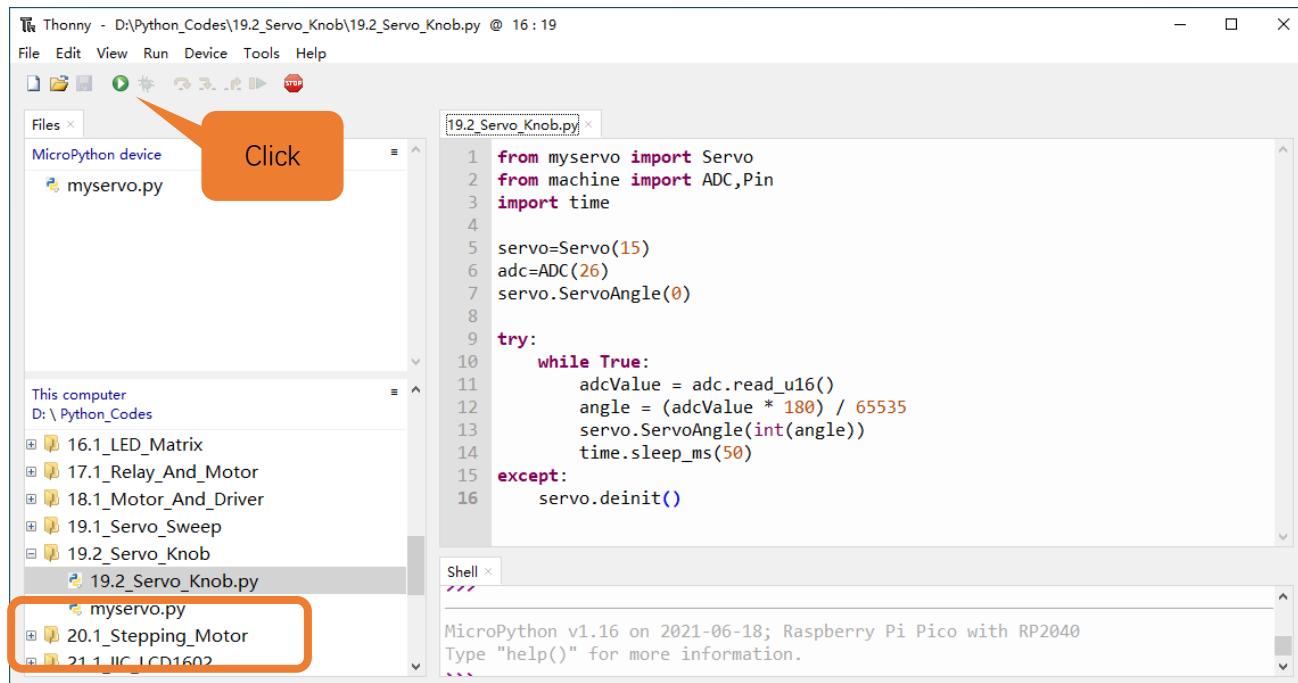


Any concerns? ✉ support@freenove.com

Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “19.2_Servo_Knob”. Select “myservo.py”, right click your mouse to select “Upload to /”, wait for “myservo.py” to be uploaded to Raspberry Pi Pico and then double click “19.2_Servo_Knob.py”.

19.2_Servo_Knob



The screenshot shows the Thonny IDE interface. The title bar says "Thonny - D:\Python_Codes\19.2_Servo_Knob\19.2_Servo_Knob.py @ 16 : 19". The menu bar includes File, Edit, View, Run, Device, Tools, and Help. The toolbar has icons for file operations like Open, Save, and Run. The left sidebar shows a tree view of the project structure under "This computer" and "D:\ Python_Codes". A red box highlights the "19.2_Servo_Knob" folder, and an orange callout bubble with the text "Click" points to the "Run current script" button in the toolbar. The main area shows the code for "19.2_Servo_Knob.py":

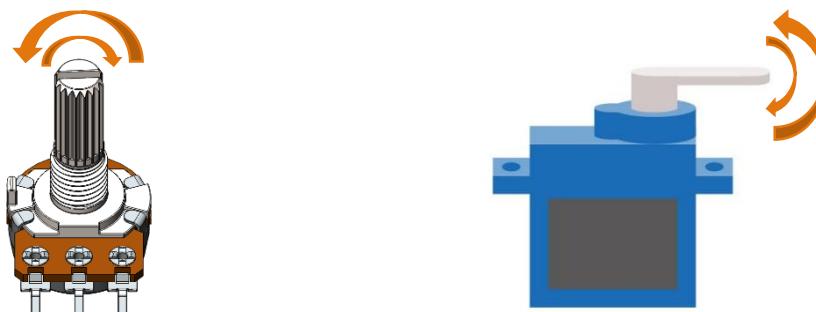
```

1 from myservo import Servo
2 from machine import ADC,Pin
3 import time
4
5 servo=Servo(15)
6 adc=ADC(26)
7 servo.ServoAngle(0)
8
9 try:
10     while True:
11         adcValue = adc.read_u16()
12         angle = (adcValue * 180) / 65535
13         servo.ServoAngle(int(angle))
14         time.sleep_ms(50)
15     except:
16         servo.deinit()

```

The bottom pane is a "Shell" window displaying the message: "MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040 Type "help()" for more information."

Click “Run current script”, twist the potentiometer back and forth, and the servo motor rotates accordingly.





The following is the program code:

```
1  from myservo import Servo
2  from machine import ADC, Pin
3  import time
4
5  servo=Servo(15)
6  adc=ADC(26)
7  servo.ServoAngle(0)
8
9  try:
10     while True:
11         adcValue = adc.read_u16()
12         angle = (adcValue * 180) / 65535
13         servo.ServoAngle(int(angle))
14         time.sleep_ms(50)
15     except:
16         servo.deinit()
```

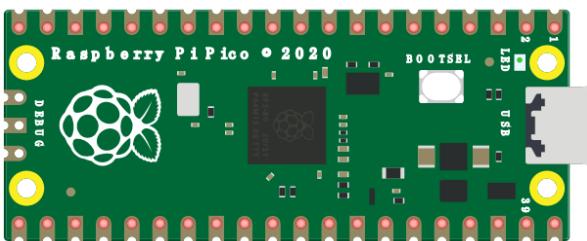
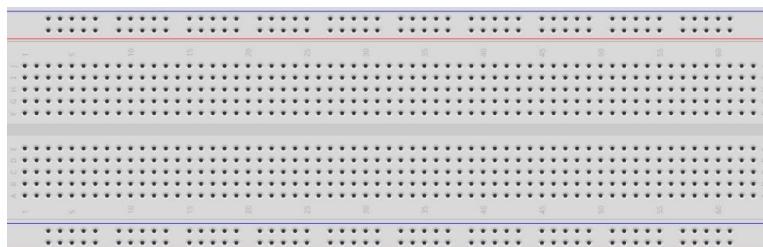
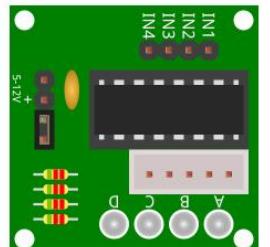
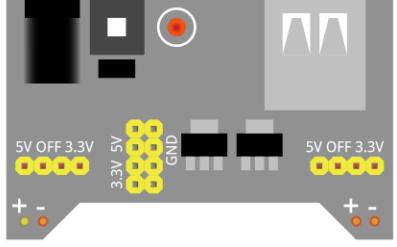
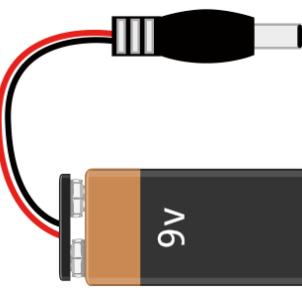
In this project, we will use GP26 of Raspberry Pi Pico to read the ADC value of the rotary potentiometer and then convert it to the angle value required by the servo and control the servo to rotate to the corresponding angle.

Chapter 20 Stepper Motor

In this project, we will learn how to drive a Stepper Motor, and understand its working principle.

Project 20.1 Stepper Motor

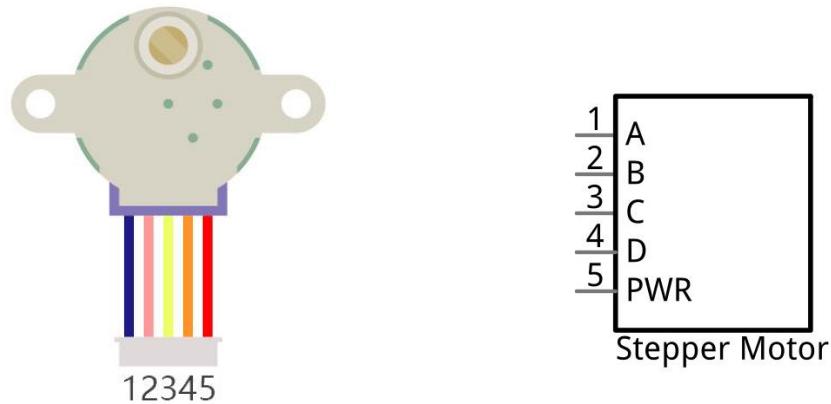
Component List

Raspberry Pi Pico x1		USB cable x1
Breadboard x1		
Stepper Motor x1		ULN2003 Stepper motorDriver x1
		Jumper
BreadBoardPower x1		9V battery (prepared by yourself) & battery line x1
		

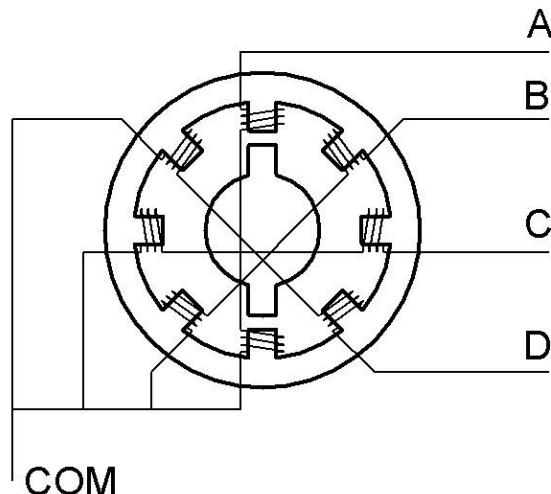
Component knowledge

Stepper Motor x1

Stepper Motors are an open-loop control device, which converts an electronic pulse signal into angular displacement or linear displacement. In a non-overload condition, the speed of the motor and the location of the stops depend only on the pulse signal frequency as well as the number of pulses and are not affected by changes in load as with a DC Motor. A small Four-Phase Deceleration Stepper Motor is shown here:

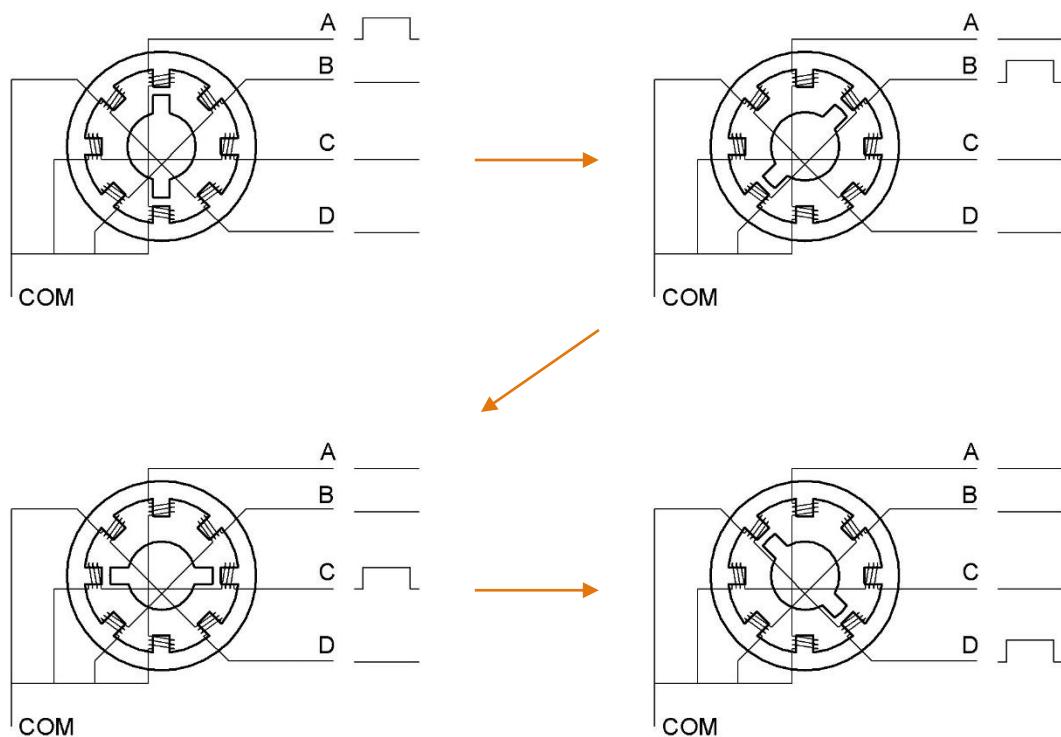


The electronic schematic diagram of a Four-Phase Stepper Motor is shown below:



The outside case or housing of the Stepper Motor is the Stator and inside the Stator is the Rotor. There are a specific number of individual coils, usually an integer multiple of the number of phases the motor has, when the Stator is powered ON, an electromagnetic field will be formed to attract a corresponding convex diagonal groove or indentation in the Rotor's surface. The Rotor is usually made of iron or a permanent magnet. Therefore, the Stepper Motor can be driven by powering the coils on the Stator in an ordered sequence (producing a series of "steps" or stepped movements).

A common driving process is as follows:



In the course above, the stepping motor rotates a certain angle once, which is called a step. By controlling the number of rotation steps, you can control the stepping motor rotation angle. By controlling the time between two steps, you can control the stepping motor rotation speed. When rotating clockwise, the order of coil powered on is: A→B→C→D→A→…… . And the rotor will rotate in accordance with the order, step by step down, called four steps four pats. If the coils is powered on in the reverse order, D→C→B→A→D→… , the rotor will rotate in anti-clockwise direction.

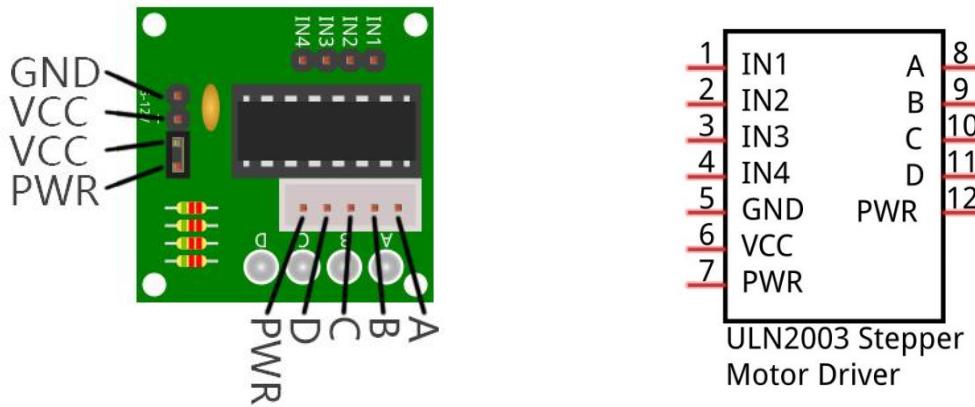
There are other methods to control Stepper Motors, such as: connect A phase, then connect A B phase, the stator will be located in the center of A B, which is called a half-step. This method can improve the stability of the Stepper Motor and reduces noise. The sequence of powering the coils looks like this: A→ AB→B →BC→C →CD→D→DA→A→……, the rotor will rotate in accordance to this sequence at a half-step at a time, called four-steps, eight-part. Conversely, if the coils are powered ON in the reverse order the Stepper Motor will rotate in the opposite direction.

The stator in the Stepper Motor we have supplied has 32 magnetic poles. Therefore, to complete one full revolution requires 32 full steps. The rotor (or output shaft) of the Stepper Motor is connected to a speed reduction set of gears and the reduction ratio is 1:64. Therefore, the final output shaft (exiting the Stepper Motor's housing) requires $32 \times 64 = 2048$ steps to make one full revolution.

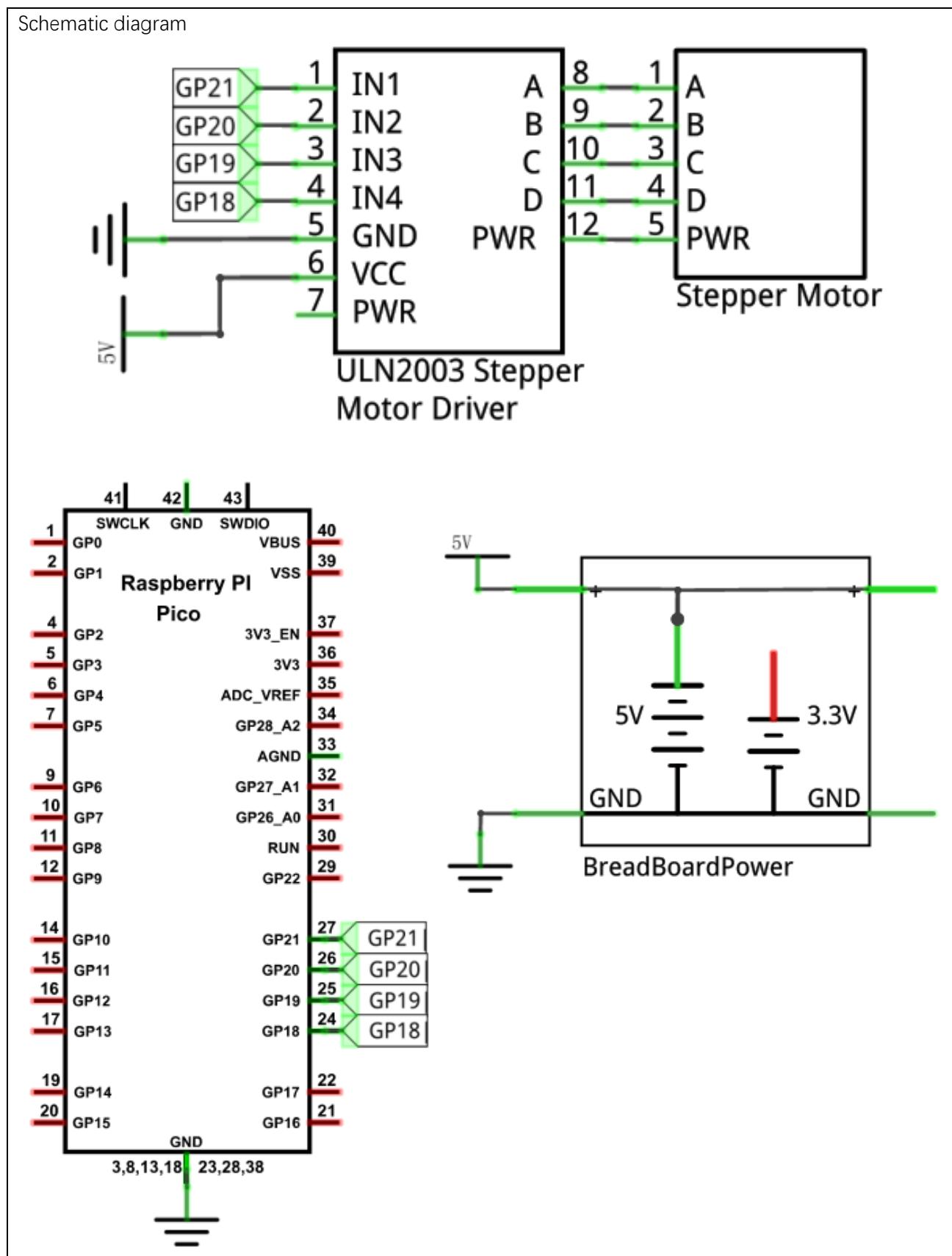
The time required for each step of the stepper motor must be greater than 2ms to operate normally.

ULN2003 Stepper motor driver

A ULN2003 Stepper Motor Driver is used to convert weak signals into more powerful control signals in order to drive the Stepper Motor. In the illustration below, the input signal IN1-IN4 corresponds to the output signal A-D, and 4 LEDs are integrated into the board to indicate the state of these signals. The PWR interface can be used as a power supply for the Stepper Motor. By default, PWR and VCC are connected.

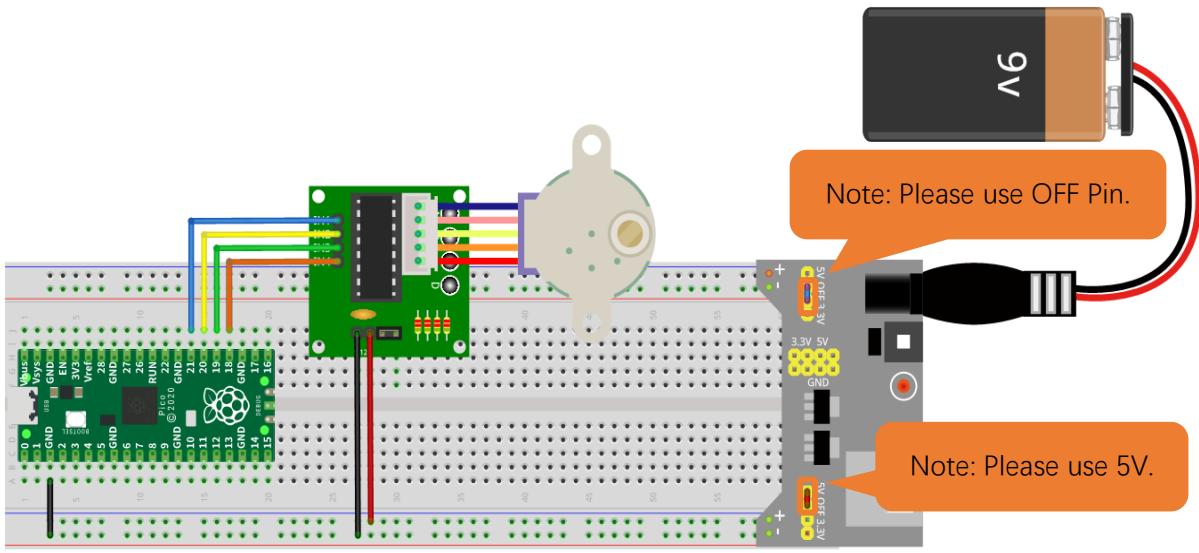


Circuit





Hardware connection. If you need any support, please free to contact us via: support@freenove.com

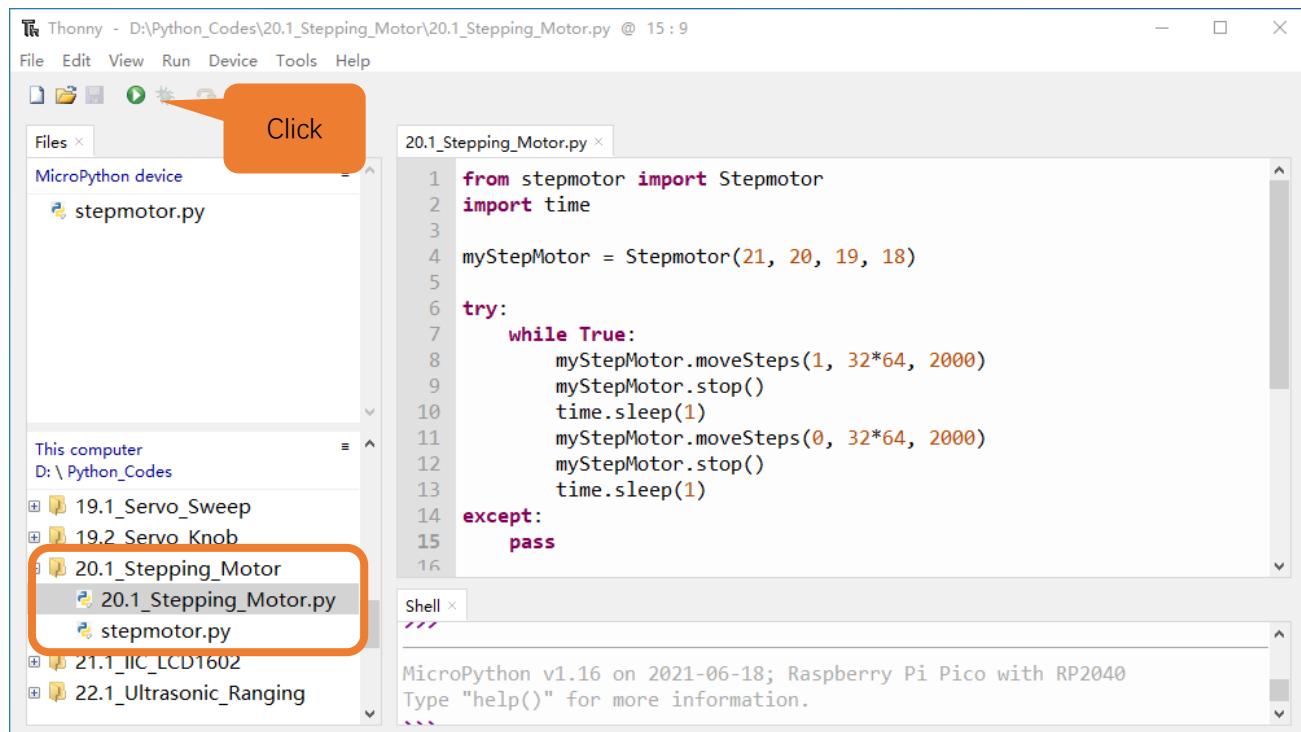


Code

This code uses the four-step, four-part mode to drive the Stepper Motor in the clockwise and anticlockwise directions.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “20.1_Stepping_Motor”. Select “stepmotor.py”, right click your mouse to select “Upload to /”, wait for “stepper.py” to be uploaded to Raspberry Pi Pico and then double click “20.1_Stepping_Motor.py”.

20.1_Stepping_Motor



```

from stepmotor import Stepmotor
import time

myStepMotor = Stepmotor(21, 20, 19, 18)

try:
    while True:
        myStepMotor.moveSteps(1, 32*64, 2000)
        myStepMotor.stop()
        time.sleep(1)
        myStepMotor.moveSteps(0, 32*64, 2000)
        myStepMotor.stop()
        time.sleep(1)
except:
    pass

```

Click “Run current script”, the stepper motor will rotate 360° clockwise and stop for 1s, and then rotate 360° anticlockwise and stop for 1s. And it will repeat this action in an endless loop. Press Ctrl+C or click “Stop/Restart backend” to exit the program.





The following is the program code:

```
1 from stepmotor import Stepmotor  
2 import time  
3  
4 myStepMotor=Stepmotor(21, 20, 19, 18)  
5  
6 try:  
7     while True:  
8         myStepMotor.moveSteps(1, 32*64, 2000)  
9         myStepMotor.stop()  
10        time.sleep(1)  
11        myStepMotor.moveSteps(0, 32*64, 2000)  
12        myStepMotor.stop()  
13        time.sleep(1)  
14 except:  
15     pass
```

Import time and stepmotor modules.

```
1 from stepmotor import Stepmotor  
2 import time
```

In this project, we define four pins to drive the stepper motor.

```
4 myStepMotor=Stepmotor(21, 20, 19, 18)
```

Call the function moveSteps to control the stepper motor to rotate for 360° and then call function stop() to stop it.

```
8     myStepMotor.moveSteps(1, 32*64, 2000)  
9     myStepMotor.stop()
```

Repeatedly control the stepmotor to rotate 360° clockwise and then rotate 360° anti-clockwise.

```
7     while True:  
8         myStepMotor.moveSteps(1, 32*64, 2000)  
9         myStepMotor.stop()  
10        time.sleep(1)  
11        myStepMotor.moveSteps(0, 32*64, 2000)  
12        myStepMotor.stop()  
13        time.sleep(1)
```

Reference

class stepmotor

Before each use of the object **Stepmotor**, please make sure that stepmotor.py has been uploaded to "/" of Raspberry Pi Pico, and then add the statement "**from stepmotor import Stepmotor**" to the top of the python file.

Stepper(): The object to control the stepper motor. The default control pins are GP21, GP20, GP19, GP18.

moveSteps(direction,steps,us): Control the stepper motor to rotate a specified number of steps.

direction: The rotation direction of stepper motor.

Steps: Rotation steps of the stepper motor.

us: The time needed for stepper motor to rotate one step, which should be greater than 2000.

moveAround(direction,turns,us): Control the stepper motor to rotate a specific number of turns.

Turns: Number of turns that the stepper motor rotates.

moveAngle(direction,angles,us): Control the stepper motor to rotate a specific angle.

Angles: Rotation angles that the stepper motor rotates.

stop(): Stop the stepper motor.



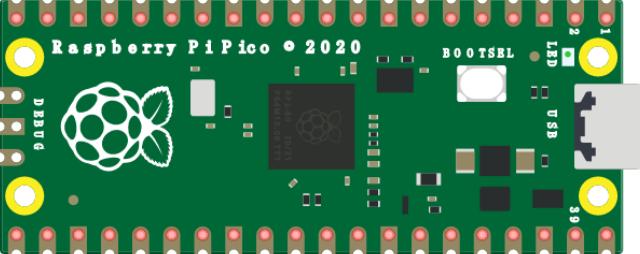
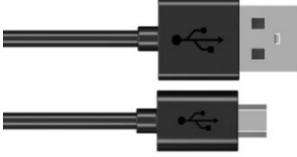
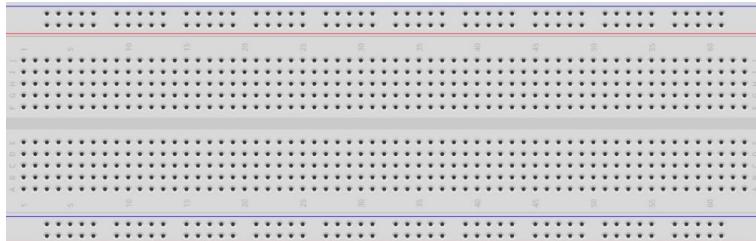
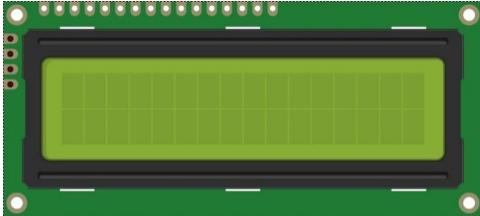
Chapter 21 LCD1602

In this chapter, we will learn about the LCD1602 Display Screen.

Project 21.1 LCD1602

In this section we learn how to use LCD1602 to display something.

Component List

Raspberry Pi Pico x1	 A green printed circuit board with a central Broadcom SoC, labeled "Raspberry Pi Pico • 2020". It has a 40-pin header at the bottom and several yellow circular pads on the top edge.	USB cable x1	 Two black USB cables, each with a standard A-type connector on one end and a micro-B connector on the other.
Breadboard x1	 A grey breadboard with two rows of 40 pins each, designed for prototyping with component pins or jumper wires.		
LCD1602 Module x1	 A green module featuring a 16x2 character LCD screen with a black frame and a small PCB underneath.	Jumper	 Two black jumper wires, each consisting of two parallel wires connected by a small bridge.

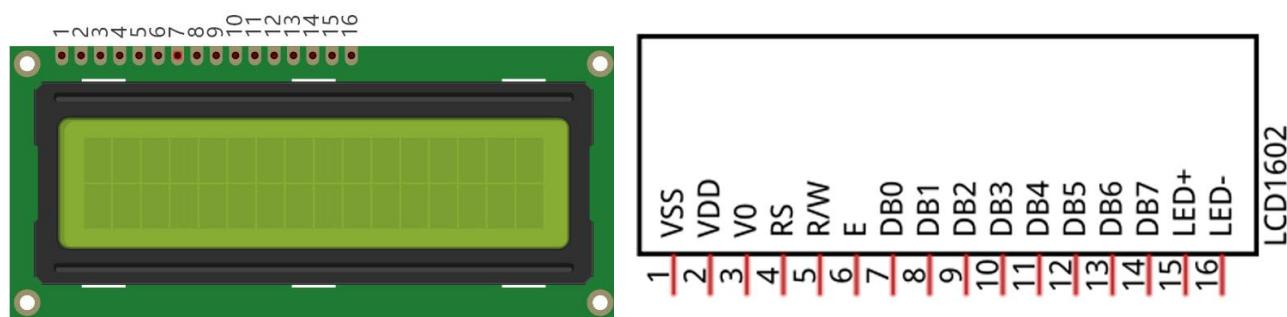
Component knowledge

I2C communication

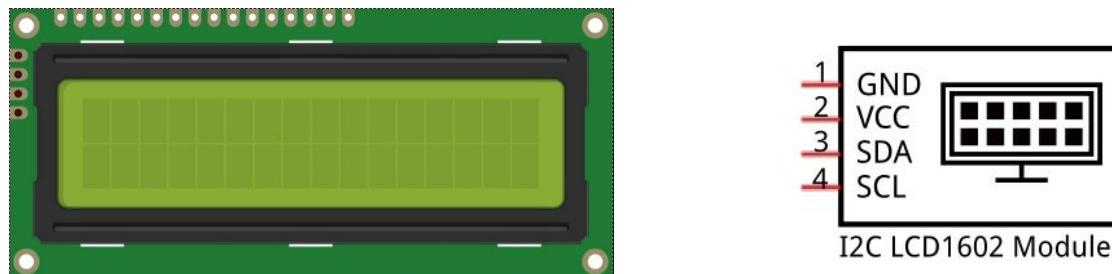
I2C (Inter-Integrated Circuit) is a two-wire serial communication mode, which can be used for the connection of micro controllers and their peripheral equipment. Devices using I2C communication must be connected to the serial data (SDA) line, and serial clock (SCL) line (called I2C bus). Each device has a unique address and can be used as a transmitter or receiver to communicate with devices connected to the bus.

LCD1602 communication

The LCD1602 Display Screen can display 2 lines of characters in 16 columns. It is capable of displaying numbers, letters, symbols, ASCII code and so on. As shown below is a monochrome LCD1602 Display Screen along with its circuit pin diagram.

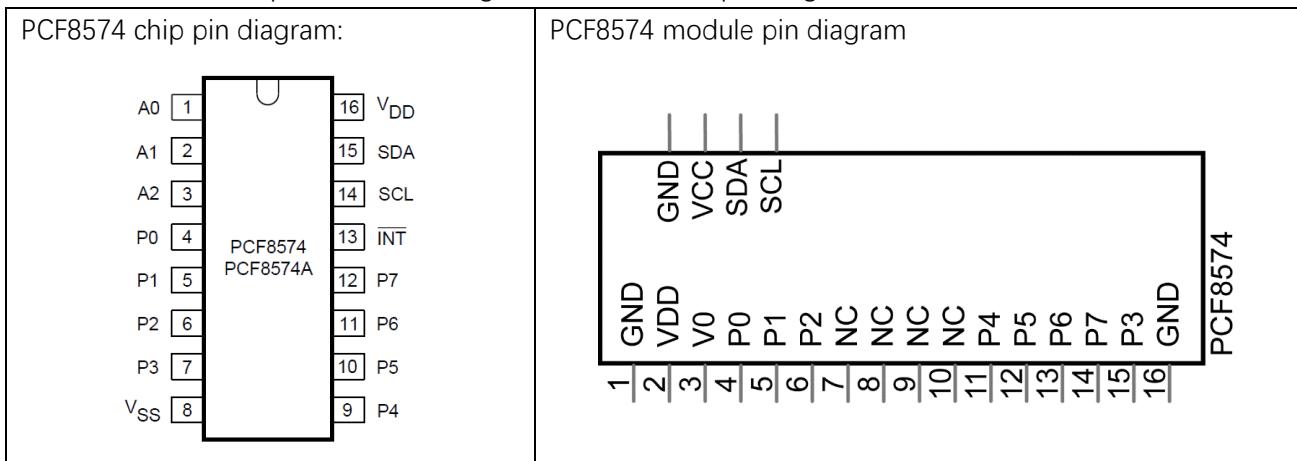


I2C LCD1602 Display Screen integrates an I2C interface, which connects the serial-input & parallel-output module to the LCD1602 Display Screen. This allows us to use only 4 lines to operate the LCD1602.

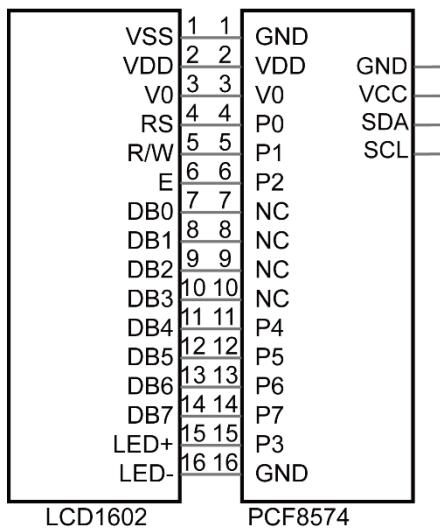


The serial-to-parallel IC chip used in this module is PCF8574T (PCF8574AT), and its default I2C address is 0x27(0x3F).

Below is the PCF8574 pin schematic diagram and the block pin diagram:



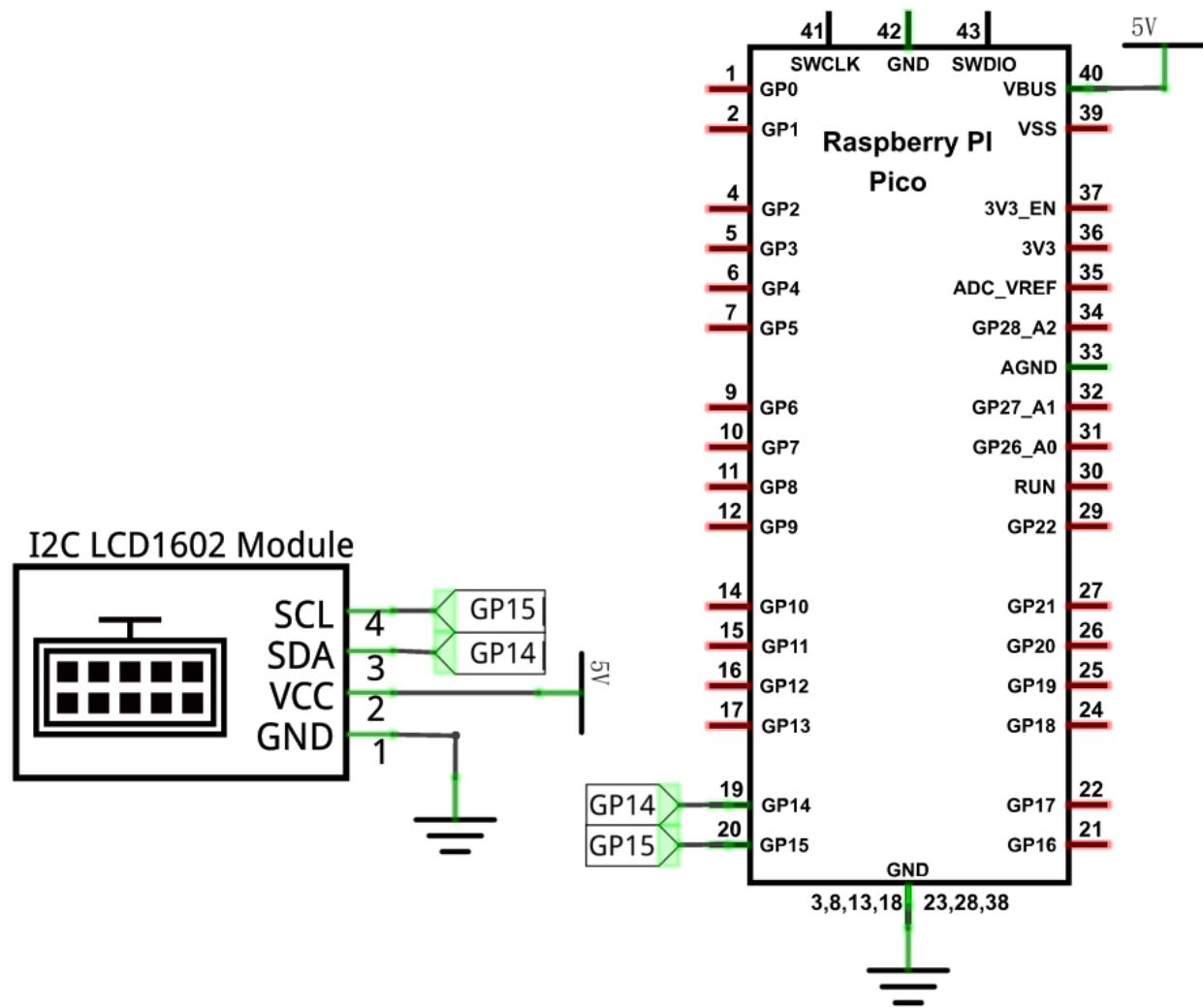
PCF8574 module pin and LCD1602 pin are corresponding to each other and connected with each other:



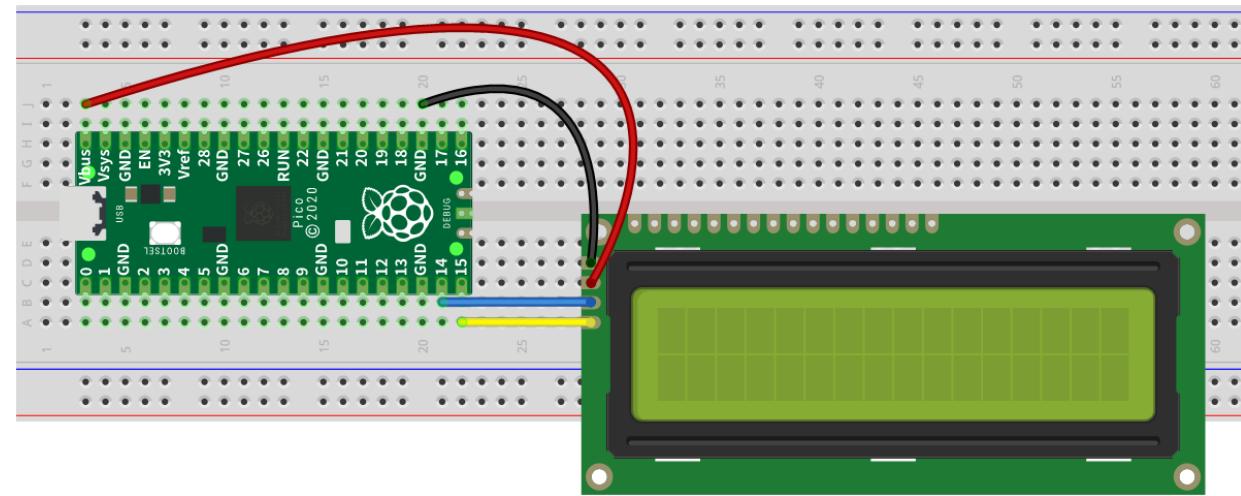
So we only need 4 pins to control the 16 pins of the LCD1602 Display Screen through the I2C interface. In this project, we will use the I2C LCD1602 to display some static characters and dynamic variables.

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Any concerns? ✉ support@freenove.com

Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “21.1_I2C_LCD1602”. Select “I2C_LCD.py” and “LCD_API.py”, right click your mouse to select “Upload to /”, wait for “I2C_LCD.py” and “LCD_API.py” to be uploaded to Raspberry Pi Pico and then double click “21.1_I2C_LCD1602.py”.

21.1_I2C_LCD1602

```

import time
from machine import I2C, Pin
from I2C_LCD import I2CLcd

i2c = I2C(1, sda=Pin(14), scl=Pin(15), freq=400000)
devices = i2c.scan()

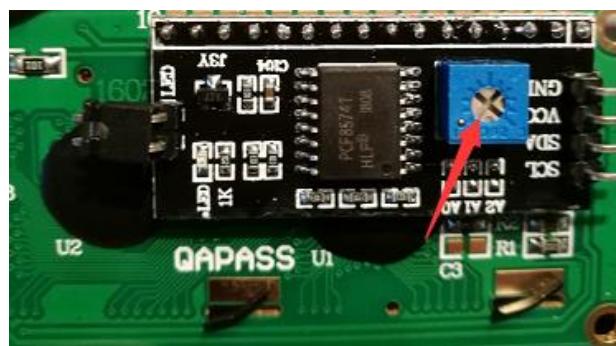
try:
    if devices != []:
        lcd = I2CLcd(i2c, devices[0], 2, 16)
        lcd.move_to(0, 0)
        lcd.putstr("Hello, world!")
        count = 0
        while True:
            lcd.move_to(0, 1)
            lcd.putstr("Counter:" + str(count))
            count += 1

```

Click “Run current script” and LCD1602 displays some characters. Press Ctrl+C or click “Stop/Restart backend” to exit the program.



Note: If you cannot see anything on the display or the display is not clear, try rotating the white knob on back of LCD1602 slowly, which adjusts the contrast, until the screen can display clearly.



The following is the program code:

```

1 import time
2 from machine import I2C, Pin
3 from I2C_LCD import I2CLcd
4
5 i2c = I2C(1, sda=Pin(14), scl=Pin(15), freq=400000)
6 devices = i2c.scan()
7
8 try:
9     if devices != []:
10         lcd = I2CLcd(i2c, devices[0], 2, 16)
11         lcd.move_to(0, 0)
12         lcd.putstr("Hello, world!")
13         count = 0
14         while True:
15             lcd.move_to(0, 1)
16             lcd.putstr("Counter:%d" %(count))
17             time.sleep(1)
18             count += 1
19     else:
20         print("No address found")
21 except:
22     pass

```

Import time, I2C and I2C_LCD modules.

```

1 import time
2 from machine import I2C, Pin
3 from I2C_LCD import I2CLcd

```

Create an I2C object, initialize the I2C parameter configuration, and associate it with the LCD1602 pin. Call the scan() function to query the LCD1602 device address.

```

4 i2c = I2C(1, sda=Pin(14), scl=Pin(15), freq=400000)
5 devices = i2c.scan()

```

Use the if statement to determine whether the queried I2C device address is empty. If it is not empty, create an I2CLcd object and set the created I2C object, I2C device address, and the number of rows and columns of LCD1602; if it is empty, print out "No address found" and the program exits.

```

9     if devices != []:
10    lcd = I2CLcd(i2c, devices[0], 2, 16)
...      ...
19    else:
20      print("No address found")

```

Move the cursor of LCD1602 to the first row, first column, and print out "Hello, world!"

```

11    lcd.move_to(0, 0)
12    lcd.putstr("Hello, world!")

```

The second line of LCD1602 continuously prints the number of seconds after the Raspberry Pi Pico program runs.

```

14    while True:
15      lcd.move_to(0, 1)
16      lcd.putstr("Counter:%d" %(count))
17      time.sleep(1)
18      count += 1

```

Reference

Class I2CLcd

Before each use of the object **I2CLcd**, please make sure that **I2C_LCD.py** and **LCD_API.py** have been uploaded to "/" of Raspberry Pi Pico, and then add the statement "**from I2C_LCD import I2CLcd**" to the top of the python file.

clear(): Clear the LCD1602 screen display.

show_cursor(): Show the cursor of LCD1602.

hide_cursor(): Hide the cursor of LCD1602.

blink_cursor_on(): Turn on cursor blinking.

blink_cursor_off(): Turn off cursor blinking.

display_on(): Turn on the display function of LCD1602.

display_off(): Turn on the display function of LCD1602.

backlight_on(): Turn on the backlight of LCD1602.

backlight_off(): Turn on the backlight of LCD1602.

move_to(cursor_x, cursor_y): Move the cursor to a specified position.

cursor_x: Column cursor_x.

cursor_y: Row cursor_y.

putchar(char): Print the character in the bracket on LCD1602.

putstr(string): Print the string in the bracket on LCD1602.

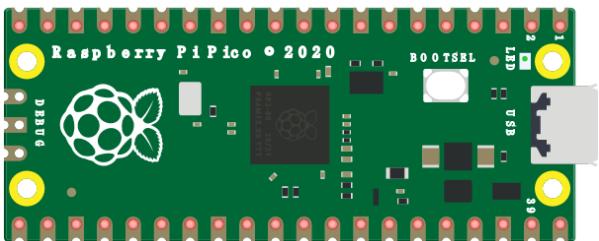
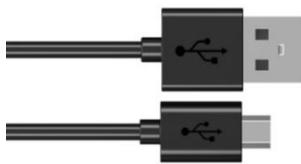
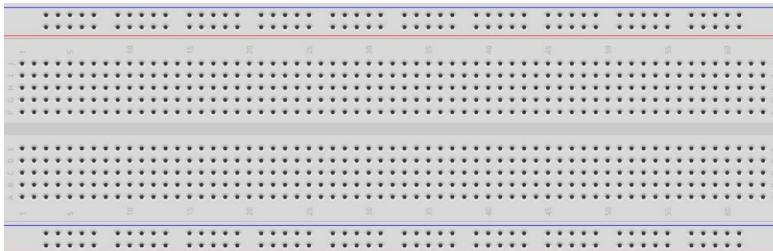
Chapter 22 Ultrasonic Ranging

In this chapter, we learn a module which use ultrasonic to measure distance, HC SR04.

Project 22.1 Ultrasonic Ranging

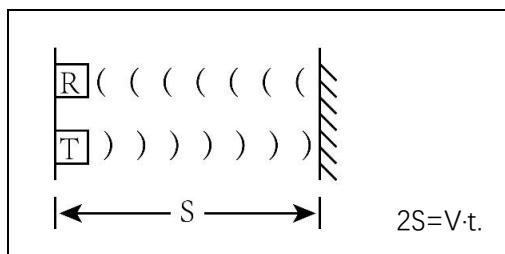
In this project, we use ultrasonic ranging module to measure distance, and print out the data in the terminal.

Component List

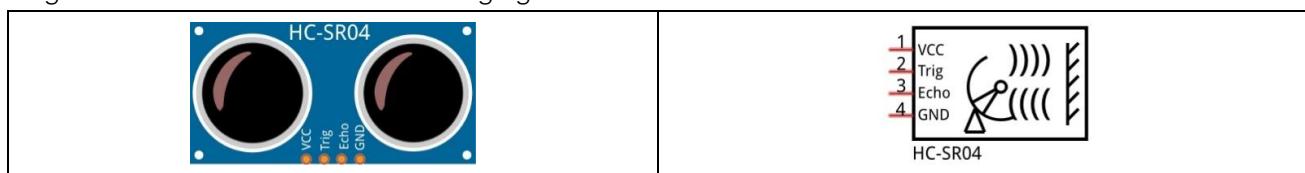
Raspberry Pi Pico x1	USB cable x1
	
Breadboard x1	
Jumper	HC SR04 x1
	

Component Knowledge

The Ultrasonic Ranging Module uses the principle that ultrasonic waves will reflect when they encounter any obstacles. This is possible by counting the time interval between when the ultrasonic wave is transmitted to when the ultrasonic wave reflects back after encountering an obstacle. Time interval counting will end after an ultrasonic wave is received, and the time difference (delta) is the total time of the ultrasonic wave's journey from being transmitted to being received. Because the speed of sound in air is a constant, and is about $v=340\text{m/s}$, we can calculate the distance between the Ultrasonic Ranging Module and the obstacle: $s=vt/2$.



The HC-SR04 Ultrasonic Ranging Module integrates both an ultrasonic transmitter and a receiver. The transmitter is used to convert electrical signals (electrical energy) into high frequency (beyond human hearing) sound waves (mechanical energy) and the function of the receiver is opposite of this. The picture and the diagram of the HC SR04 Ultrasonic Ranging Module are shown below:



Pin description:

Pin	Description
VCC	power supply pin
Trig	trigger pin
Echo	Echo pin
GND	GND

Technical specs:

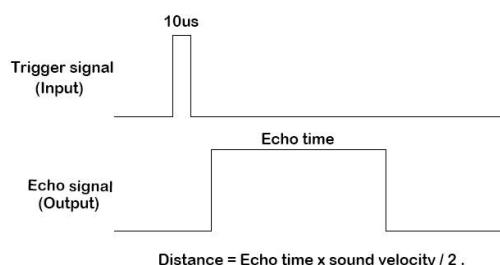
Working voltage: 5V

Working current: 12mA

Minimum measured distance: 2cm

Maximum measured distance: 200cm

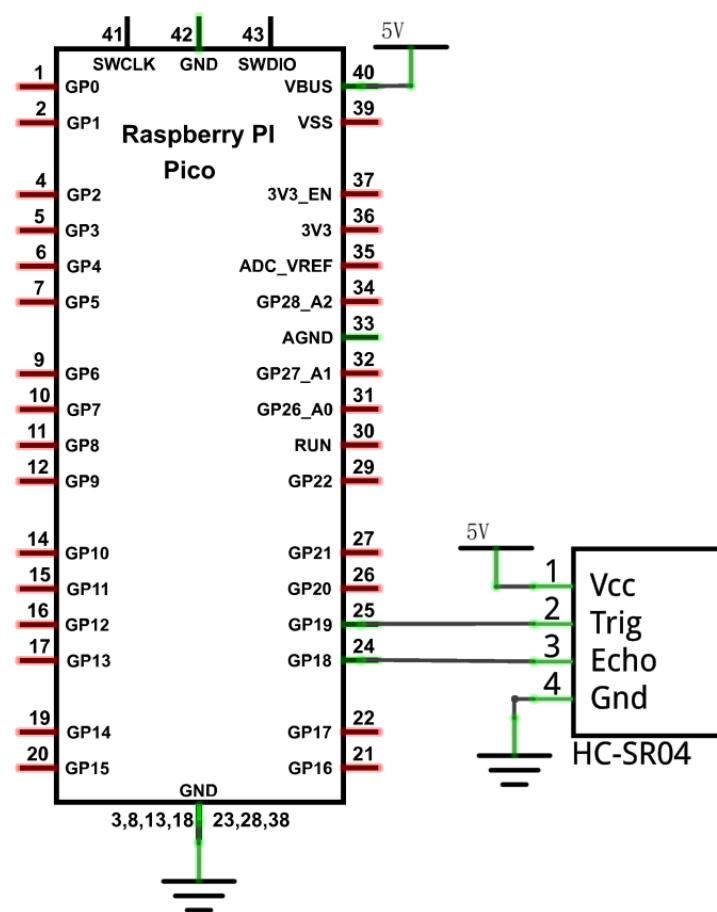
Instructions for Use: output a high-level pulse in Trig pin lasting for least 10us, the module begins to transmit ultrasonic waves. At the same time, the Echo pin is pulled up. When the module receives the returned ultrasonic waves from encountering an obstacle, the Echo pin will be pulled down. The duration of high level in the Echo pin is the total time of the ultrasonic wave from transmitting to receiving, $s=vt/2$.



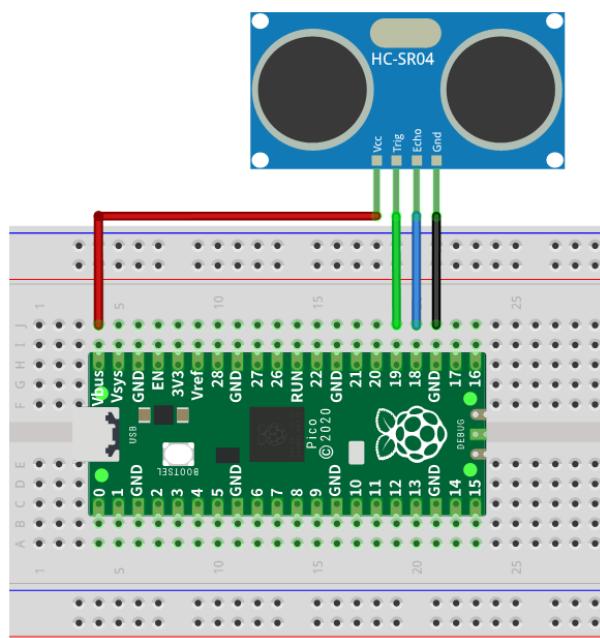
Circuit

Note that the voltage of ultrasonic module is 5V in the circuit.

Schematic diagram



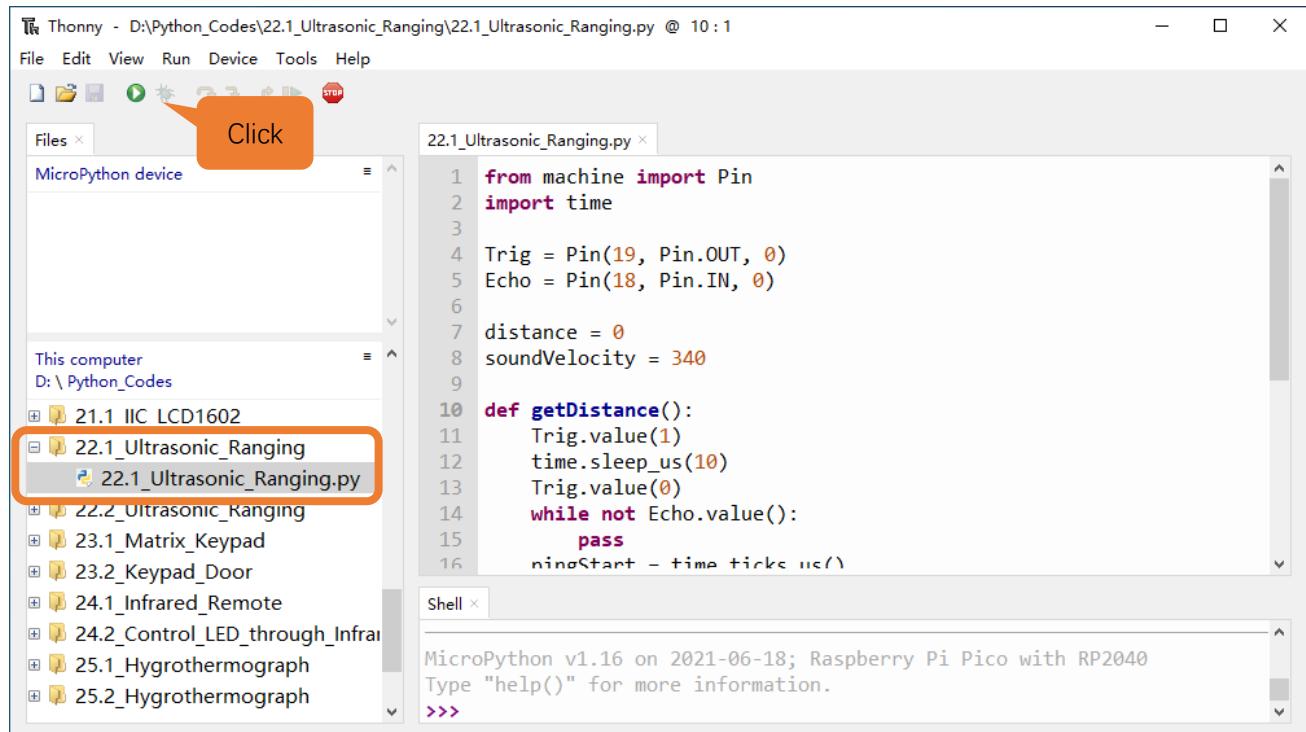
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “22.1_Ultrasonic_Ranging” and double click “22.1_Ultrasonic_Ranging.py”.

22.1_Ultrasonic_Ranging



The screenshot shows the Thonny IDE interface. The title bar says "Thonny - D:\Python_Codes\22.1_Ultrasonic_Ranging\22.1_Ultrasonic_Ranging.py @ 10 : 1". The menu bar includes File, Edit, View, Run, Device, Tools, and Help. The toolbar has icons for Open, Save, Run, Stop, and others. The left sidebar is titled "Files" and shows a tree view of files under "MicroPython device". A red box highlights the "Click" button in the toolbar, and another red box highlights the "22.1_Ultrasonic_Ranging" folder and its subfile "22.1_Ultrasonic_Ranging.py". The main editor window displays the Python code for ultrasonic ranging:

```

1 from machine import Pin
2 import time
3
4 Trig = Pin(19, Pin.OUT, 0)
5 Echo = Pin(18, Pin.IN, 0)
6
7 distance = 0
8 soundVelocity = 340
9
10 def getDistance():
11     Trig.value(1)
12     time.sleep_us(10)
13     Trig.value(0)
14     while not Echo.value():
15         pass
16     pingStart = time.ticks_us()

```

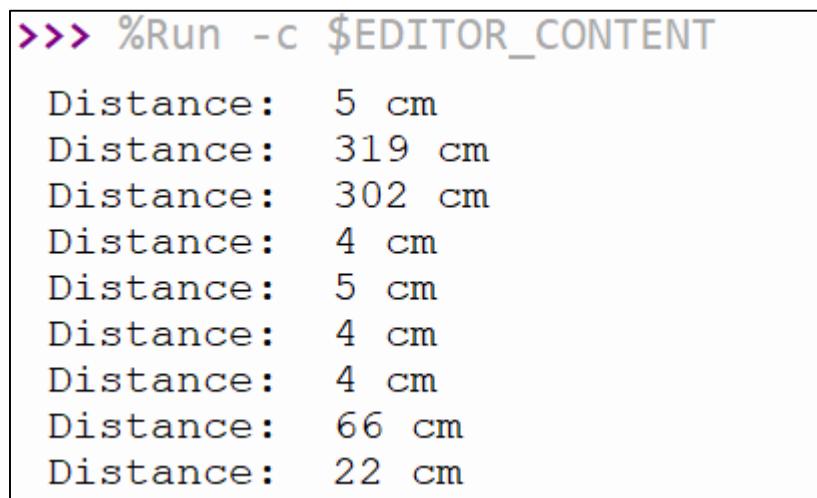
The bottom shell window shows the MicroPython version and a prompt:

```

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>

```

Click “Run current script”, you can use it to measure the distance between the ultrasonic module and the object. As shown in the following figure. Press Ctrl+C or click “Stop/Restart backend” to exit the program.



The screenshot shows the Thonny Shell window with the command ">>> %Run -c \$EDITOR_CONTENT" entered. The output shows a series of distance measurements in centimeters:

```

>>> %Run -c $EDITOR_CONTENT
Distance: 5 cm
Distance: 319 cm
Distance: 302 cm
Distance: 4 cm
Distance: 5 cm
Distance: 4 cm
Distance: 4 cm
Distance: 66 cm
Distance: 22 cm

```

The following is the program code:

```

1  from machine import Pin
2  import time
3
4  Trig = Pin(19, Pin.OUT, 0)
5  Echo = Pin(18, Pin.IN, 0)
6
7  distance = 0
8  soundVelocity = 340
9
10 def getDistance():
11     Trig.value(1)
12     time.sleep_us(10)
13     Trig.value(0)
14     while not Echo.value():
15         pass
16     pingStart = time.ticks_us()
17     while Echo.value():
18         pass
19     pingStop = time.ticks_us()
20     distanceTime = time.ticks_diff(pingStop, pingStart) // 2
21     distance = int(soundVelocity * distanceTime // 10000)
22     return distance
23
24 time.sleep(2)
25 while True:
26     time.sleep_ms(500)
27     distance = getDistance()
28     print("Distance: ", distance, "cm")

```

Define the control pins of the ultrasonic ranging module.

```

4  Trig = Pin(19, Pin.OUT, 0)
5  Echo = Pin(18, Pin.IN, 0)

```

Set the speed of sound.

```

8  soundVelocity = 340

```

The `getDistance()` function is used to drive the ultrasonic module to measure distance. In the function, after the `Trig` pin keeps at high level for 10us to start the ultrasonic module, `Echo.value()` is used to read the status of ultrasonic module's `Echo` pin, and then use timestamp function of the time module to calculate the duration of `Echo` pin's high level. Finally, calculate the measured distance based on time and return the value.

```

10 def getDistance():
11     Trig.value(1)
12     time.sleep_us(10)
13     Trig.value(0)
14     while not Echo.value():
15         pass

```

```
16 pingStart = time.ticks_us()
17 while Echo.value():
18     pass
19 pingStop = time.ticks_us()
20 distanceTime = time.ticks_diff(pingStop, pingStart) // 2
21 distance = int(soundVelocity * distanceTime // 10000)
22 return distance
```

Delay for 2 seconds and wait for the ultrasonic module to stabilize. Print data obtained from ultrasonic module every 500 milliseconds.

```
24 time.sleep(2)
25 while True:
26     time.sleep_ms(500)
27     distance = getDistance()
28     print("Distance: ", distance, "CM")
```

Project 22.2 Ultrasonic Ranging

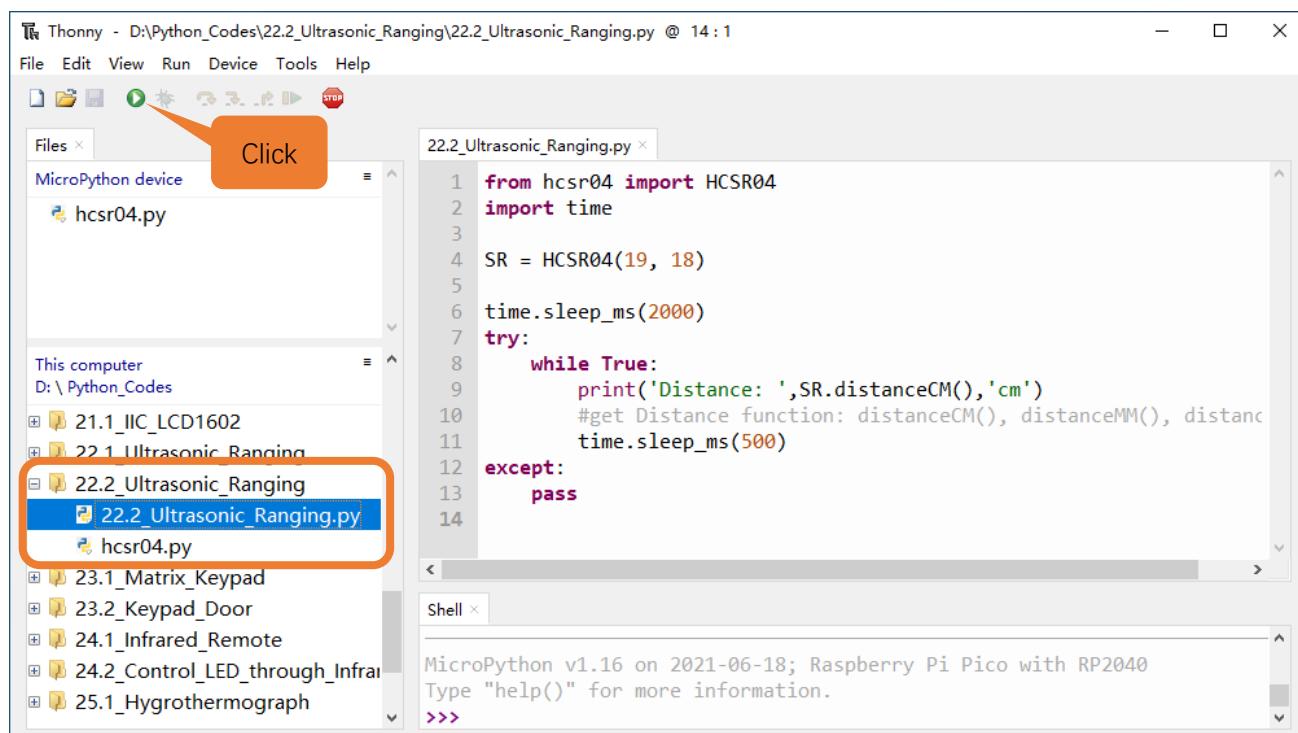
Component List and Circuit

Component List and Circuit are the same as the previous section.

Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “22.2_Ultrasonic_Ranging”. Select “hcsr04.py”, right click your mouse to select “Upload to /”, wait for “hcsr04.py” to be uploaded to Raspberry Pi Pico and then double click “22.2_Ultrasonic_Ranging.py”.

22.2_Ultrasonic_Ranging



Click “Run current script”. Use the ultrasonic module to measure distance. As shown in the following figure. Press Ctrl+C or click “Stop/Restart backend” to exit the program.

```

>>> %Run -c $EDITOR_CONTENT
Distance: 16 cm
Distance: 17 cm
Distance: 18 cm
Distance: 17 cm
Distance: 17 cm
Distance: 16 cm
Distance: 16 cm
Distance: 16 cm

```

The following is the program code:

```

1  from hcsr04 import HCSR04
2  import time
3
4  SR = HCSR04(19, 18)
5
6  time.sleep_ms(2000)
7  try:
8      while True:
9          print('Distance: ',SR.distanceCM(),'cm')
10         #get Distance function: distanceCM(), distanceMM(), distanceM()
11         time.sleep_ms(500)
12     except:
13         pass

```

Import hcsr04 module.

```
1  from hcsr04 import HCSR04
```

Define an ultrasonic object and associate with the pins.

```
4  SR= HCSR04(18, 19)
```

Obtain the distance data returned from the ultrasonic ranging module.

```
9  SR.distanceCM()
```

Obtain the ultrasonic data every 500 milliseconds and print them out in "Shell".

```

8      while True:
9          print('Distance: ',SR.distanceCM(),'cm')
10         #get Distance function: distanceCM(), distanceMM(), distanceM()
11         time.sleep_ms(500)

```

Reference

Class hcsr04

Before each use of object **HCSR04**, please add the statement “**from hcsr04 import HCSR04**” to the top of python file.

SRHC04(): Object of ultrasonic module. By default, trig pin is GP19 and echo pin is GP18.

distanceCM(): Obtain the distance from the ultrasonic to the measured object with the data type being int type, and the unit being cm.

distanceMM(): Obtain the distance from the ultrasonic to the measured object with the data type being int type, and the unit being mm.

distanceM(): Obtain the distance from the ultrasonic to the measured object with the data type being float type, and the unit being m.

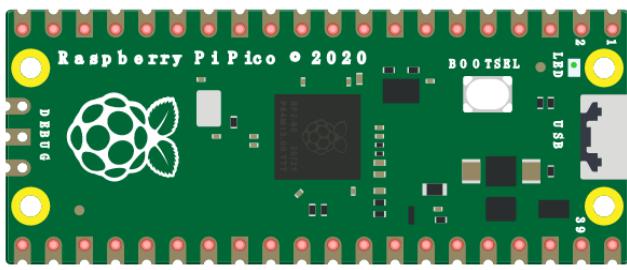
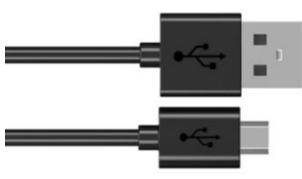
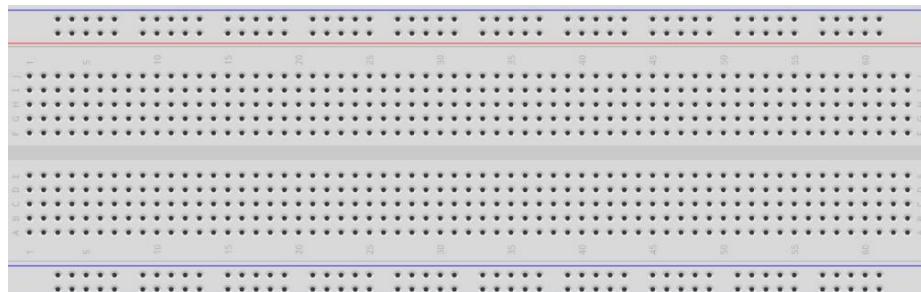
Chapter 23 Matrix Keypad

Earlier we learned about a single Push Button Switch. In this chapter, we will learn about Matrix Keyboards, which integrates a number of Push Button Switches as Keys for the purposes of Input.

Project 23.1 Matrix Keypad

In this project, we will attempt to get every key code on the Matrix Keypad to work.

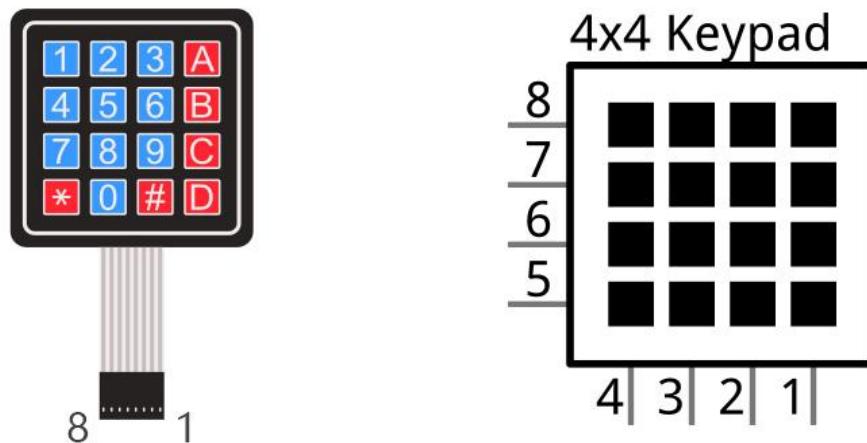
Component List

Raspberry Pi Pico x1	USB cable x1
	
Breadboard x1	
Jumper	4x4 Matrix Keypad x1 

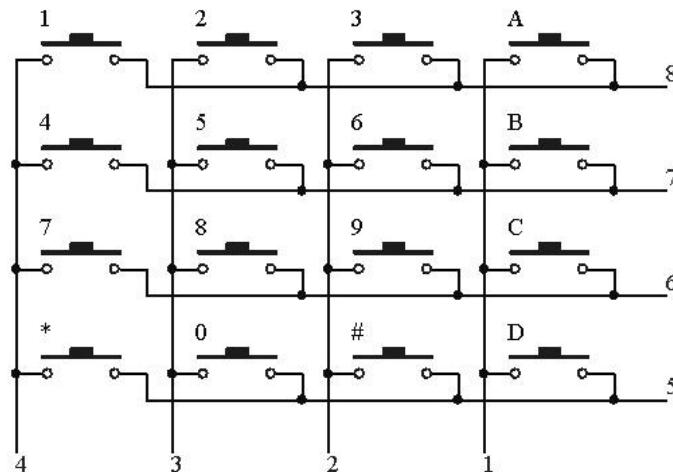
Component knowledge

4x4 Matrix Keypad

A Keypad Matrix is a device that integrates a number of keys in one package. As is shown below, a 4x4 Keypad Matrix integrates 16 keys:



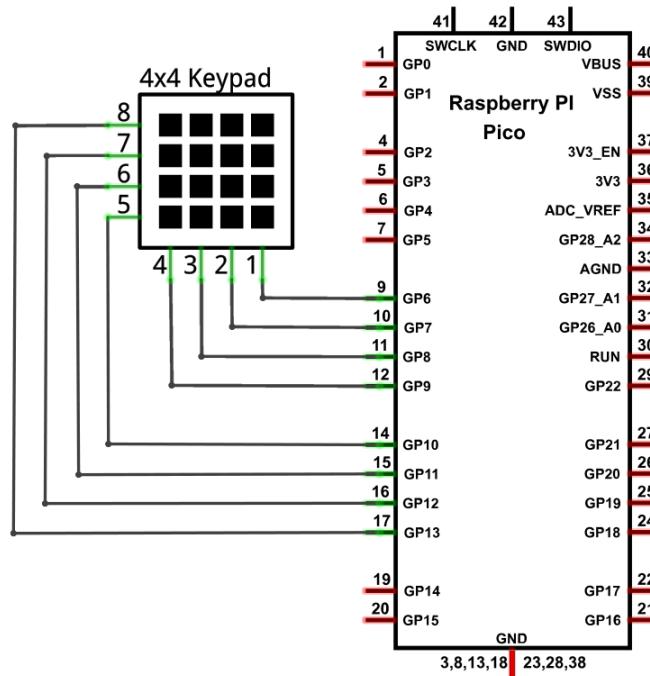
Similar to the integration of an LED Matrix, the 4x4 Keypad Matrix has each row of keys connected with one pin and this is the same for the columns. Such efficient connections reduce the number of processor ports required. The internal circuit of the Keypad Matrix is shown below.



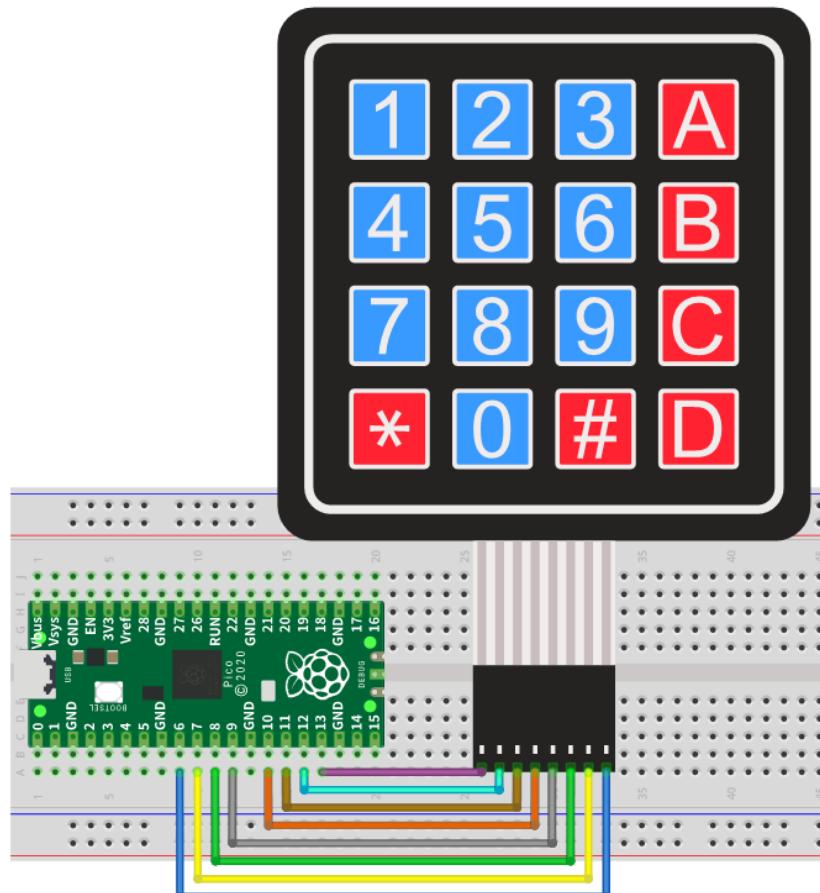
The method of usage is similar to the Matrix LED, by using a row or column scanning method to detect the state of each key's position by column and row. Take column scanning method as an example, send low level to the first 1 column (Pin1), detect level state of row 5, 6, 7, 8 to determine whether the key A, B, C, D are pressed. Then send low level to column 2, 3, 4 in turn to detect whether other keys are pressed. By this means, you can get the state of all of the keys.

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



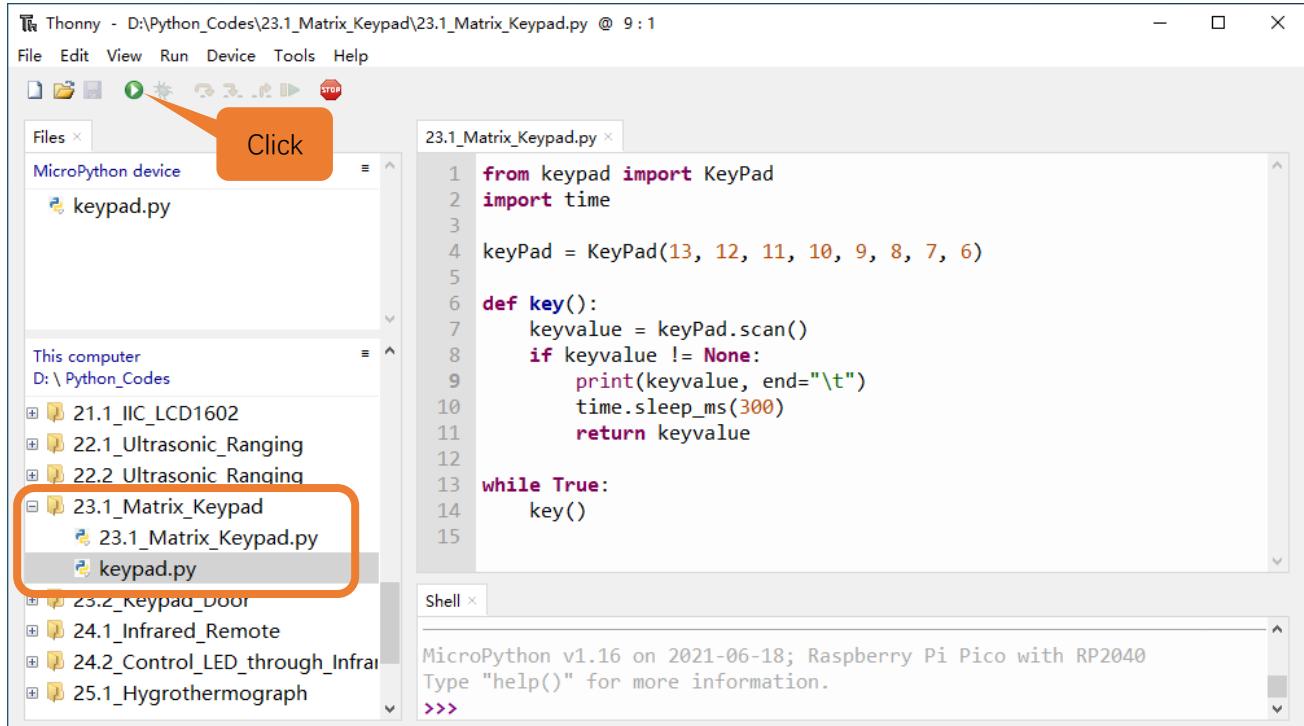
Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

Code

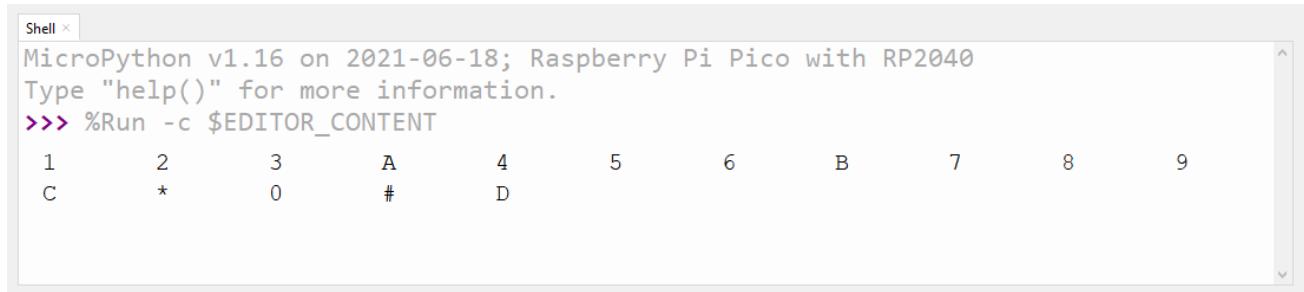
This code is used to obtain all key codes of the 4x4 Matrix Keypad, when one of the keys is pressed, the key code will be printed out via serial port.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “23.1_Matrix_Keypad”. Select “keypad.py”, right click your mouse to select “Upload to /”, wait for “keypad.py” to be uploaded to Raspberry Pi Pico and then double click “23.1_Matrix_Keypad.py”.

23.1_Matrix_Keypad



Click “Run current script”, push the key board and the key value will be printed in “Shell”, as shown in the illustration below. Press Ctrl+C or click “Stop/Restart backend” to exit the program.



The following is the program code:

```

1  from keypad import KeyPad
2  import time
3
4  keyPad = KeyPad(13, 12, 11, 10, 9, 8, 7, 6)
5  def key():
6      keyvalue = keyPad.scan()
7      if keyvalue != None:
8          print(keyvalue, end="\t")
9          time.sleep_ms(300)
10     return keyvalue
11
12 while True:
13     key()

```

Import keypad module.

```
1  from keypad import KeyPad
```

Associate the keypad module to Raspberry Pi Pico pins.

```
4  keyPad = KeyPad(13, 12, 11, 10, 9, 8, 7, 6)
```

Call function `keypad.scan()` of the keypad module. When the keypad module detects that the key is pressed, it returns the value of the pressed key; when no key is pressed, the return value is `None`.

```
6  keyPad.scan()
```

Call function `keyPan.scan()` to obtain the value of the pressed key. Once it is obtained, print it out.

```

5  def key():
6      keyvalue=keyPad.scan()
7      if keyvalue!= None:
8          print(keyvalue, end="\t")
9          time.sleep_ms(300)
10     return keyvalue

```

Reference

Class keypad

Before each use of the object **KeyPad**, please make sure **keypad.py** has been uploaded to "/" of Raspberry Pi Pico and then add the statement "**from keypad import KeyPad**" to the top of python file.

KeyPad(row1, row2, row3, row4, col1, col2, col3, col4): Initialize keypad module and associate its pins with Raspberry Pi Pico. **KeyPad() == KeyPad(13, 12, 11, 10, 9, 8, 7, 6)**.

scan(): Non-blocking keypad scan function. If no key is pressed, it returns `None`; Otherwise, it returns the valued of the pressed key.



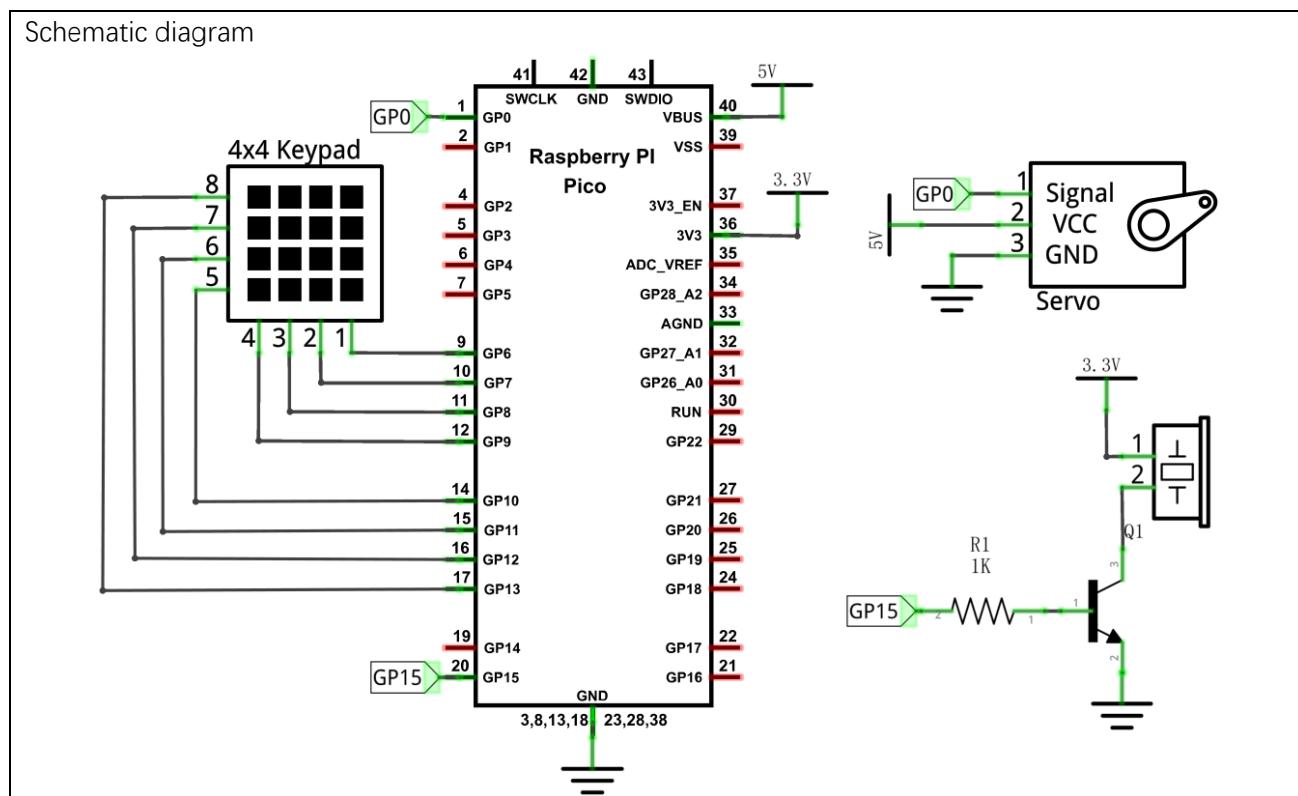
Project 23.2 Keypad Door

In this project, we use keypad as a keyboard to control the action of the servo motor.

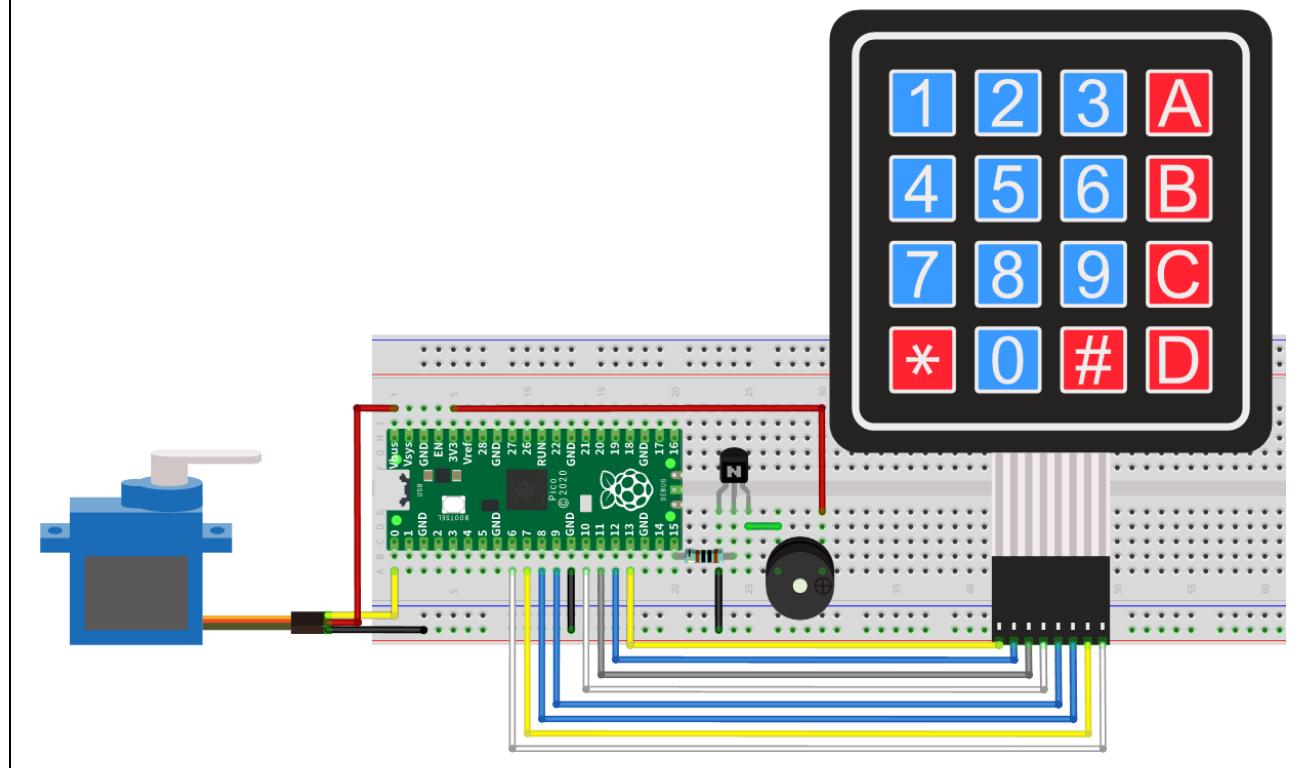
Component List

Raspberry Pi Pico x1	USB cable x1
Breadboard x1	
Jumper	Servo x1
4x4 Matrix Keypad x1	
NPN transistor x1 (S8050)	Active buzzer x1
	Resistor 1kΩ x1

Circuit



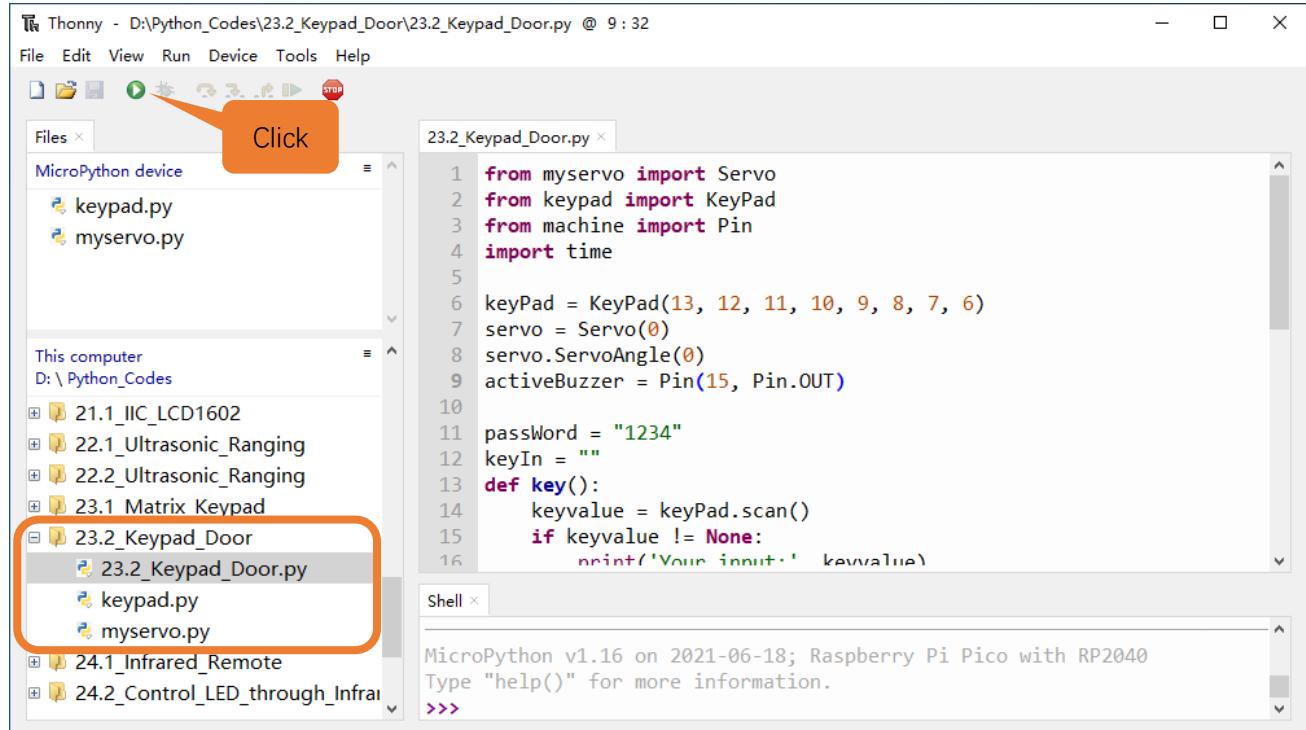
Hardware connection. If you need any support, please free to contact us via: support@freenove.com



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “23.2_Keypad_Door”. Select “keypad.py” and “myservo.py”, right click your mouse to select “Upload to /”, wait for “keypad.py” and “myservo.py” to be uploaded to Raspberry Pi Pico and then double click “23.2_Keypad_Door.py”.

23.2_Keypad_Door



The screenshot shows the Thonny IDE interface. The title bar says "Thonny - D:\Python_Codes\23.2_Keypad_Door\23.2_Keypad_Door.py @ 9 : 32". The menu bar includes File, Edit, View, Run, Device, Tools, and Help. The toolbar has icons for file operations like Open, Save, and Run. The left sidebar shows a tree view of the project structure under "MicroPython device". A red box highlights the "Click" button on the toolbar, and another red box highlights the "23.2_Keypad_Door" folder and its files ("keypad.py", "myservo.py") within the project tree. The main code editor window displays the following Python code:

```

1 from myservo import Servo
2 from keypad import KeyPad
3 from machine import Pin
4 import time
5
6 keyPad = KeyPad(13, 12, 11, 10, 9, 8, 7, 6)
7 servo = Servo(0)
8 servo.ServoAngle(0)
9 activeBuzzer = Pin(15, Pin.OUT)
10
11 passWord = "1234"
12 keyIn = ""
13 def key():
14     keyValue = keyPad.scan()
15     if keyValue != None:
16         print('Your input: ', keyValue)

```

The bottom shell window shows the MicroPython version and a prompt: "MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040 Type "help()" for more information. >>>"

Click “Run current script”, press the keypad to input password with 4 characters. If the input is correct, the servo will move to a certain degree, and then return to the original position. If the input is wrong, an input error will occur. Press Ctrl+C or click “Stop/Restart backend” to exit the program.

The following is the program code:

```
1  from myservo import Servo
2  from keypad import KeyPad
3  from machine import Pin
4  import time
5
6  keyPad = KeyPad(13, 12, 11, 10, 9, 8, 7, 6)
7  servo = Servo(0)
8  servo.ServoAngle(0)
9  activeBuzzer = Pin(15, Pin.OUT)
10
11 passWord = "1234"
12 keyIn = ""
13 def key():
14     keyvalue = keyPad.scan()
15     if keyvalue != None:
16         print(' Your input:', keyvalue)
17         time.sleep_ms(200)
18         return keyvalue
19
20 while True:
21     keydata = key()
22     if keydata != None:
23         activeBuzzer.value(1)
24         time.sleep_ms(100)
25         activeBuzzer.value(0)
26         keyIn += keydata
27
28     if len(keyIn) == 4:
29         if keyIn == passWord:
30             print("passWord right!")
31             servo.ServoAngle(90)
32             time.sleep_ms(1000)
33             servo.ServoAngle(0)
34         else:
35             print("passWord error!")
36             activeBuzzer.value(1)
37             time.sleep_ms(1000)
38             activeBuzzer.value(0)
39         keyIn = ""
```



Set the passWord data to a string and assign the value to the passWord variable.

```
11  passWord = "1234"
```

The buzzer makes a brief sound each time a key is pressed, and the key value is recorded in the keyIn string.

```
21  keydata = key()
22  if keydata != None:
23      activeBuzzer.value(1)
24      time.sleep_ms(100)
25      activeBuzzer.value(0)
26      keyIn += keydata
```

When pressed 4 times, keyIn records 4 key values and compares them with passWord. If they are the same, the motor rotates 90 degrees, and returns to its original position after one second. If they are inconsistent, the buzzer will sound a warning and clear the contents of the keyIn string and wait for the next input.

```
28  if len(keyIn) == 4:
29      if keyIn == passWord:
30          print("passWord right!")
31          servo.ServoAngle(90)
32          time.sleep_ms(1000)
33          servo.ServoAngle(0)
34      else:
35          print("passWord error!")
36          activeBuzzer.value(1)
37          time.sleep_ms(1000)
38          activeBuzzer.value(0)
39      keyIn = ""
```

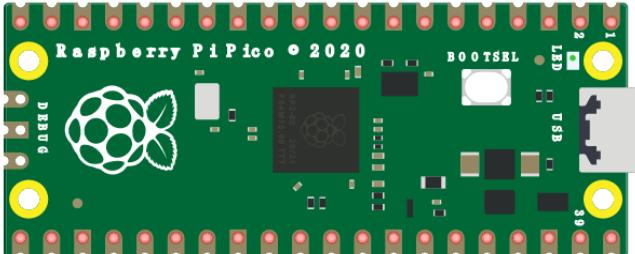
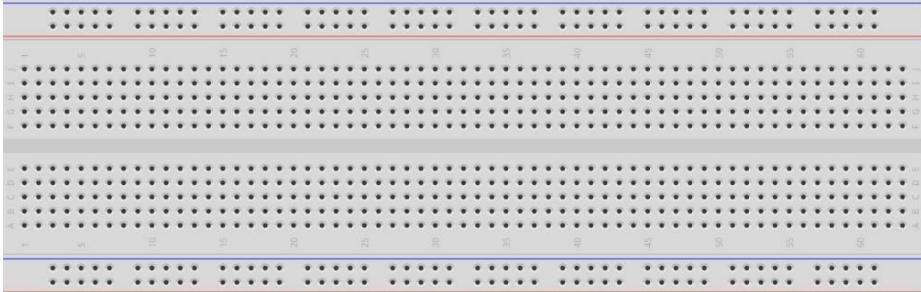
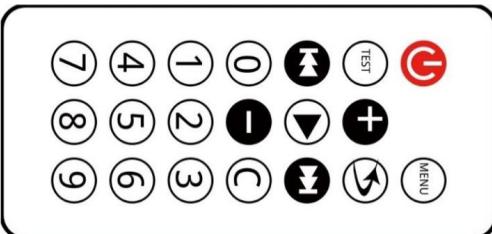
Chapter 24 Infrared Remote

In this chapter, we'll learn how to use an infrared remote control, and control an LED.

Project 24.1 Infrared Remote Control

First, we need to understand how infrared remote control works, then get the command sent from infrared remote control.

Component List

Raspberry Pi Pico x1		USB cable x1
Breadboard x1		
Jumper		Infrared Remote x1 (May need CR2025 battery x1, please check the holder)
Infrared Remote x1		Resistor 10kΩ x1
		



Component knowledge

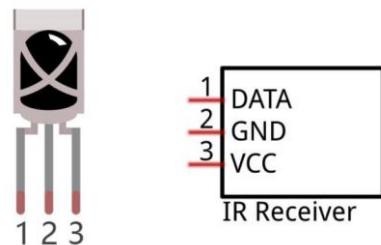
Infrared Remote

An infrared(IR) remote control is a device with a certain number of buttons. Pressing down different buttons will make the infrared emission tube, which is located in the front of the remote control, send infrared ray with different command. Infrared remote control technology is widely used in electronic products such as TV, air conditioning, etc. Thus making it possible for you to switch TV programs and adjust the temperature of the air conditioning when away from them. The remote control we use is shown below:



Infrared receiver

An infrared(IR) receiver is a component which can receive the infrared light, so we can use it to detect the signal emitted by the infrared remote control. DATA pin here outputs the received infrared signal.



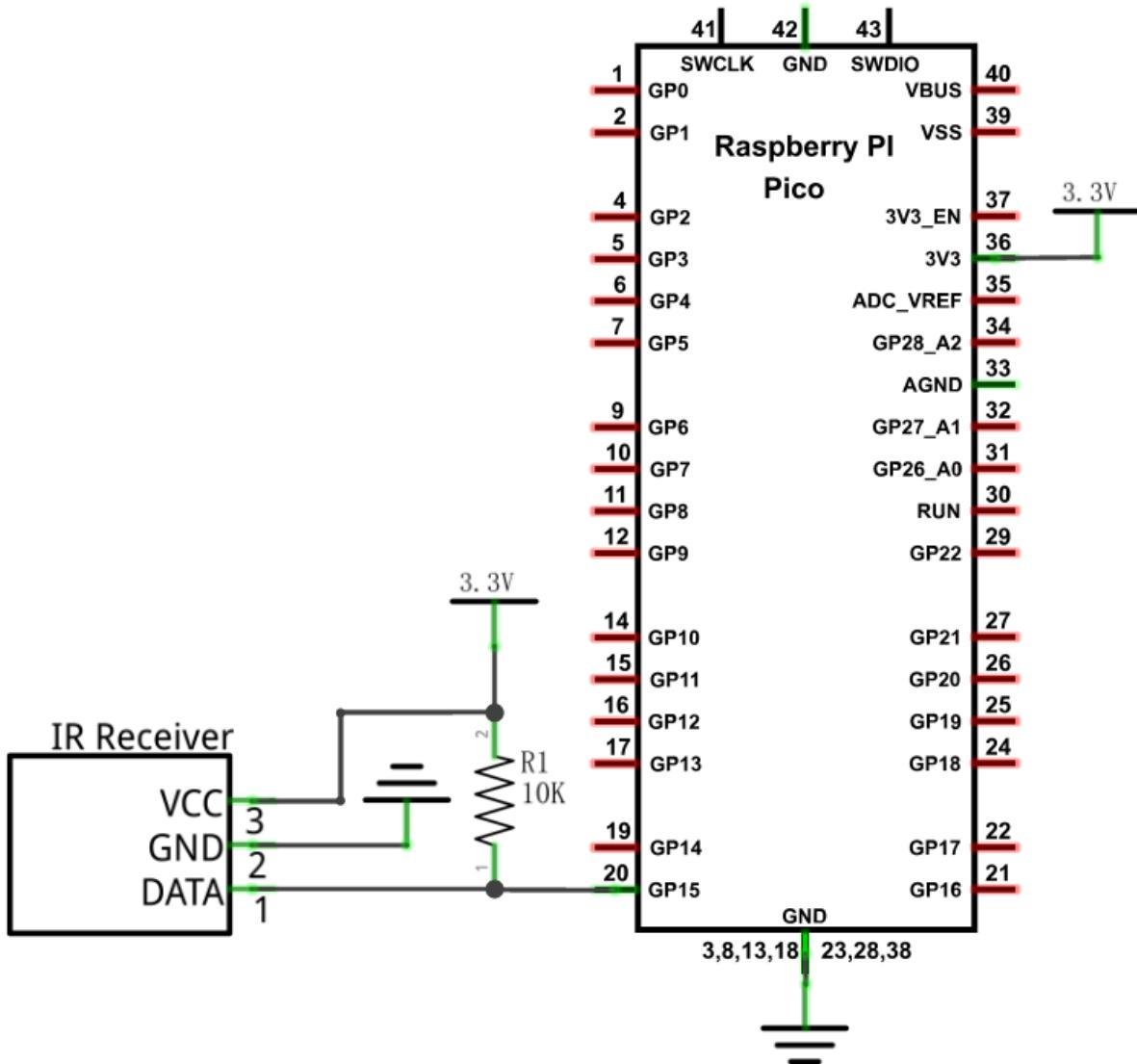
When you use the infrared remote control, the infrared remote control sends a key value to the receiving circuit according to the pressed key. We can program the Raspberry Pi Pico to do things like lighting, when a key value is received.

The following is the key value that the receiving circuit will receive when each key of the infrared remote control is pressed.

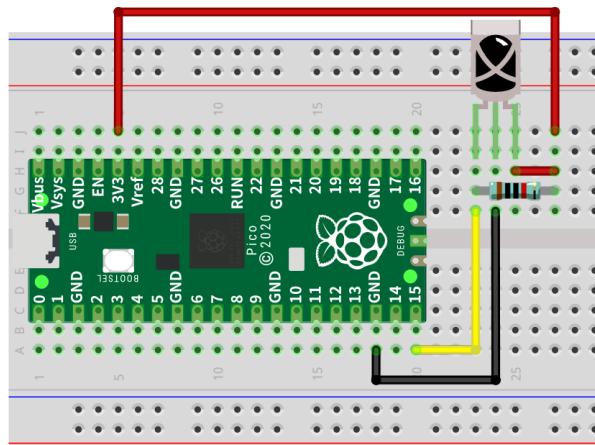
ICON	KEY Value	ICON	KEY Value
	FFA25D		FFB04F
	FFE21D		FF30CF
	FF22DD		FF18E7
	FF02FD		FF7A85
	FFC23D		FF10EF
	FFE01F		FF38C7
	FFA857		FF5AA5
	FF906F		FF42BD
	FF6897		FF4AB5
	FF9867		FF52AD

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

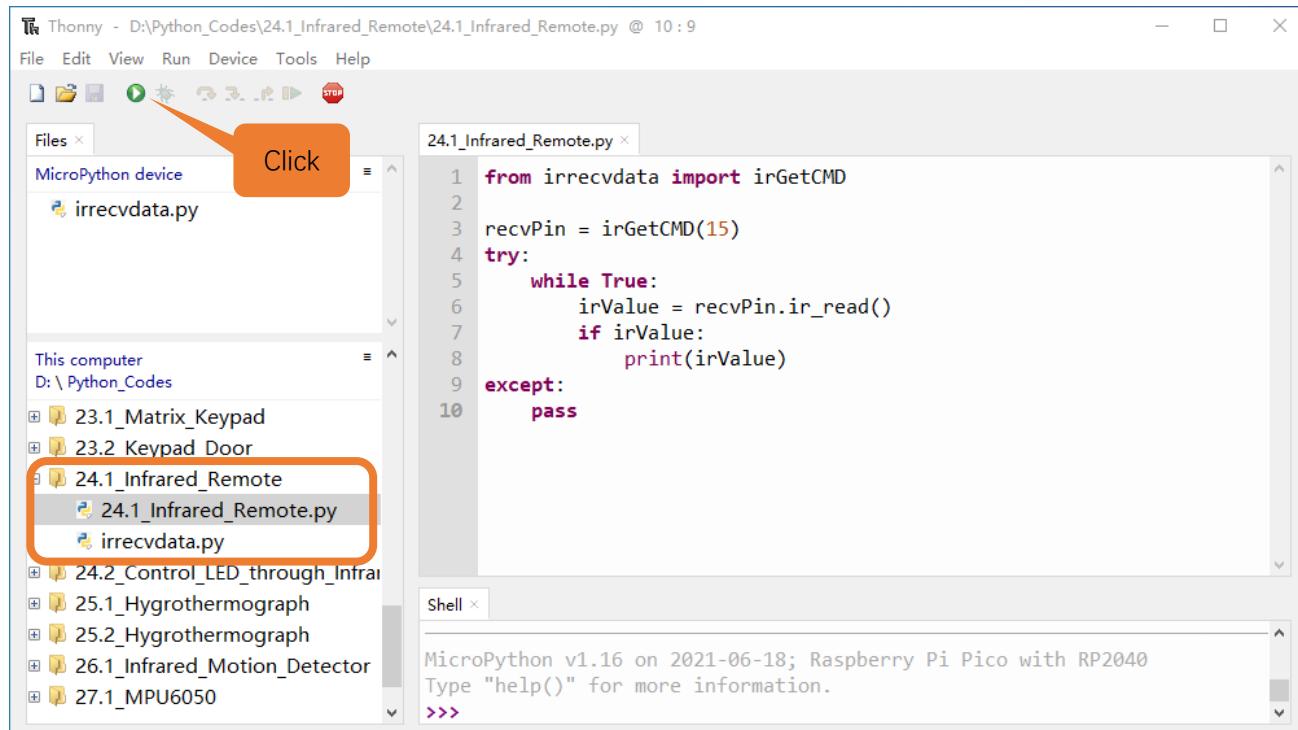


Any concerns? ✉ support@freenove.com

Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “24.1_Infrared_Remote”. Select “irrecvdata.py”, right click your mouse to select “Upload to /”, wait for “irrecvdata.py” to be uploaded to Raspberry Pi Pico and then double click “24.1_Infrared_Remote.py”.

24.1_Infrared_Remote



Click “Run current script”. Press any key of the infrared remote and the key value will be printed in “Shell”, as shown in the illustration below. Press Ctrl+C or click “Stop/Restart backend” to exit the program.

```

MicroPython v1.16 on 2021-07-20; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

0xff6897
0xfff18e7
0xfff7a85
0xfff30cf
0xfff10ef
0xfff38c7
0xfff5aa5
0xfff4ab5
0xfff42bd
0xfff4ab5
0xfff52ad

```



The following is the program code:

```
1  from irrecvdata import irGetCMD  
2  
3  recvPin = irGetCMD(15)  
4  try:  
5      while True:  
6          irValue = recvPin.ir_read()  
7          if irValue:  
8              print(irValue)  
9  except:  
10     pass
```

Import the infrared decoder.

```
1  from irrecvdata import irGetCMD
```

Associate the infrared decoder with GP15.

```
6  recvPin = irGetCMD(15)
```

Call `ir_read()` to read the value of the pressed key and assign it to `IRValue`.

```
12  irValue = recvPin.ir_read()
```

When infrared key value is obtained, print it out in "Shell".

```
5      while True:  
6          irValue = recvPin.ir_read()  
7          if irValue:  
8              print(irValue)
```

Reference

Class `irrecvdata`

Before each use of the object `irrecvdata`, please add the statement "`from irrecvdata import irGetCMD`" to the top of the python file.

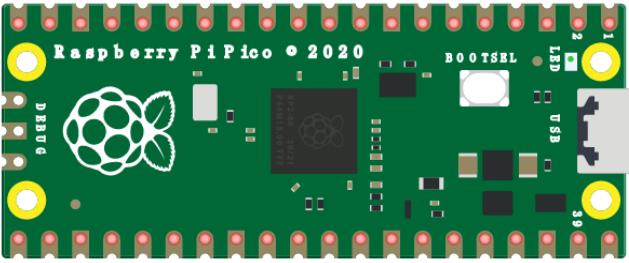
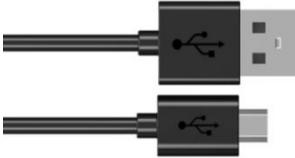
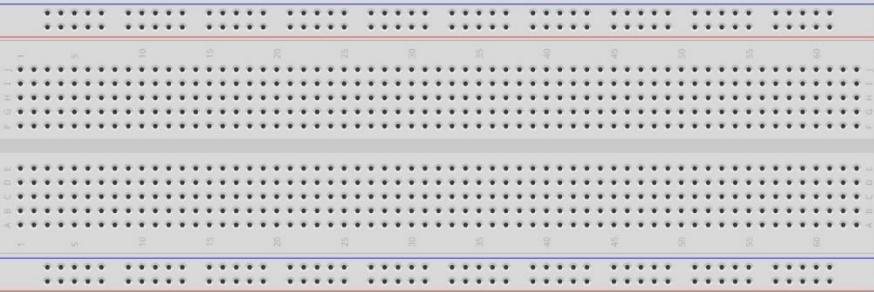
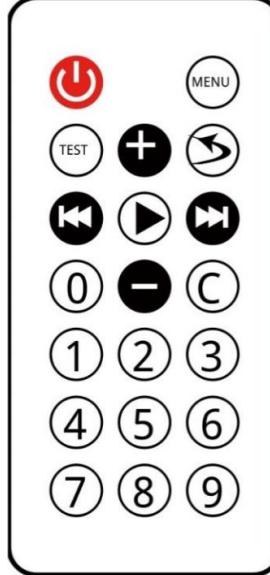
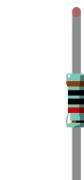
irGetCMD(): Object of infrared encoder, which is associated with GP15 by default.

ir_read(): The function that reads the key value of infrared remote. When the value is read, it will be returned; when no value is obtained, character **None** will be returned.

Project 24.2 Control LED through Infrared Remote

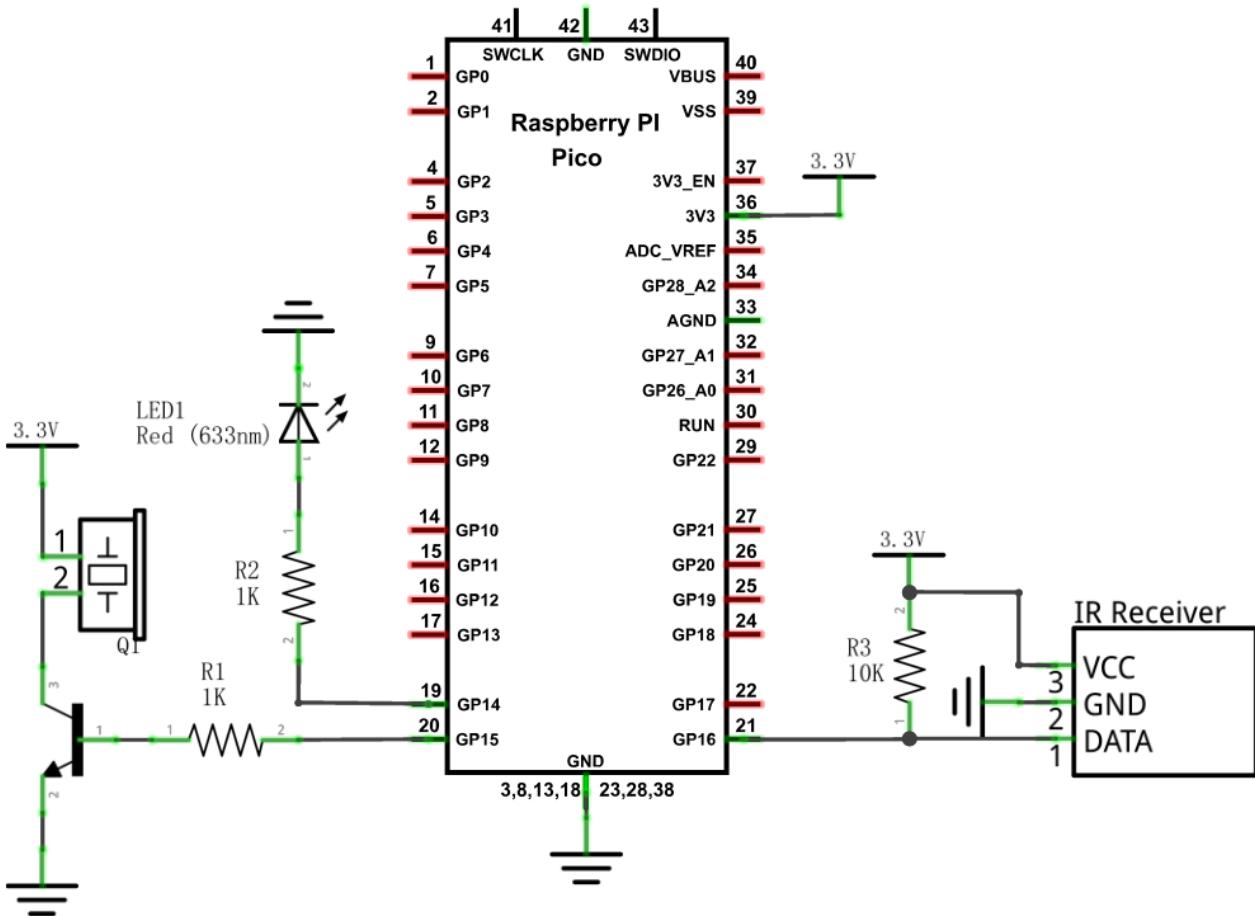
In this project, we will control the brightness of LED lights through an infrared remote control.

Component List

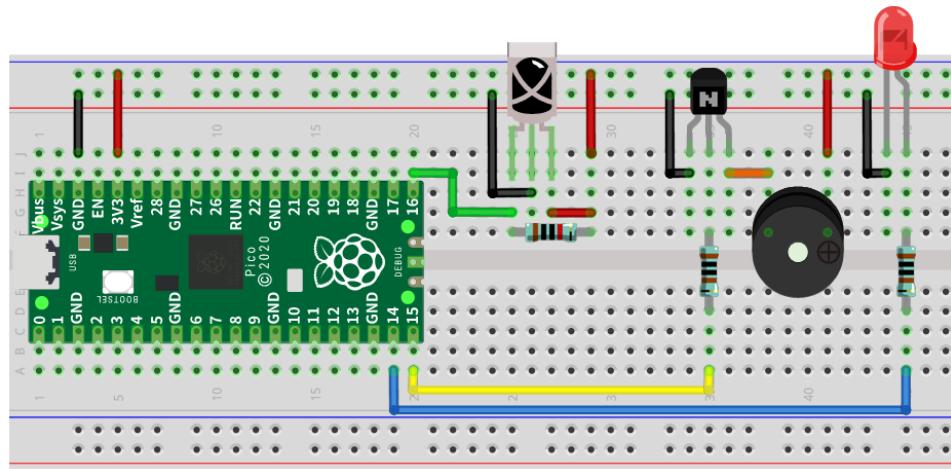
Raspberry Pi Pico x1			USB cable x1
			
Breadboard x1			
Jumper			
LED x1	Active buzzer x1	Resistor 1kΩ x2	Infrared Remote x1 (May need CR2025 battery x1, please check the battery holder)
			
Infrared receiver x1	NPN transistorx1 (S8050)	Resistor 10kΩ x1	
			

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

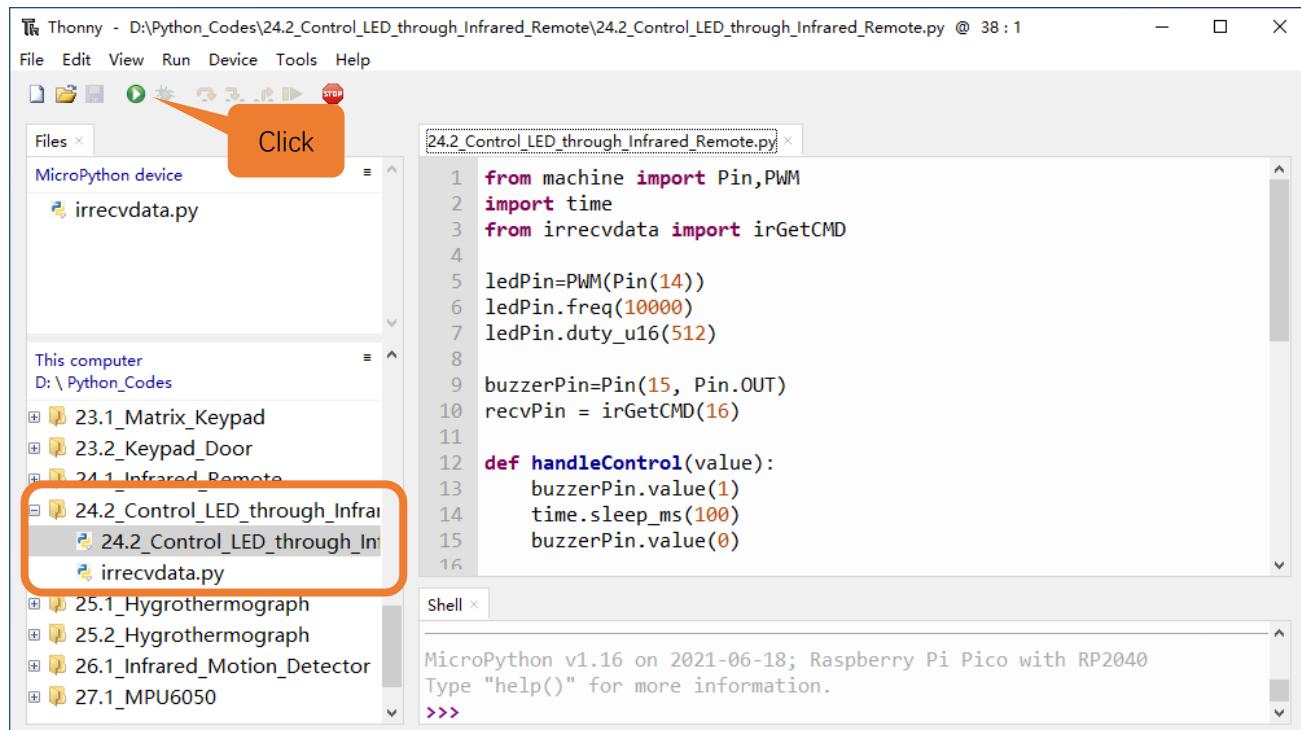


Code

The Code controls the brightness of the LED by determining the key value of the infrared received.

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "24.2_Control_LED_through_Infrared_Remote". Select "irrecvdata.py", right click your mouse to select "Upload to /", wait for "irrecvdata.py" to be uploaded to Raspberry Pi Pico and then double click "24.2_Control_LED_through_Infrared_Remote.py".

24.2_Control_LED_through_Infrared_Remote



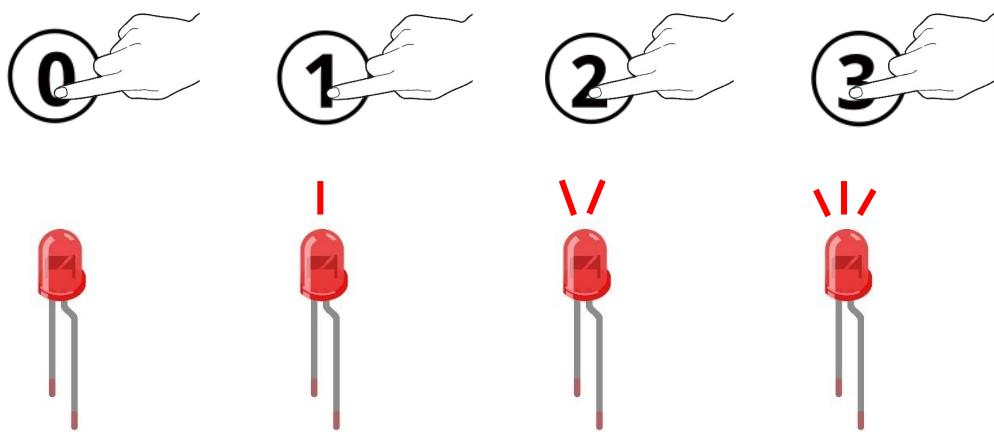
```
from machine import Pin,PWM
import time
from irrecvdata import irGetCMD

ledPin=PWM(Pin(14))
ledPin.freq(10000)
ledPin.duty_u16(512)

buzzerPin=Pin(15, Pin.OUT)
recvPin = irGetCMD(16)

def handleControl(value):
    buzzerPin.value(1)
    time.sleep_ms(100)
    buzzerPin.value(0)
```

Click “Run current script”. When pressing “0”, “1”, “2”, “3” of the infrared remote control, the buzzer will sound once, and the brightness of the LED light will change correspondingly. Press Ctrl+C or click “Stop/Restart backend” to exit the program.



The following is the program code:

```

1  from machine import Pin, PWM
2  import time
3  from irrecvdata import irGetCMD
4
5  ledPin=PWM(Pin(14))
6  ledPin.freq(10000)
7  ledPin.duty_u16(512)
8  buzzerPin=Pin(15, Pin.OUT)
9  recvPin = irGetCMD(16)
10
11 def handleControl(value):
12     buzzerPin.value(1)
13     time.sleep_ms(100)
14     buzzerPin.value(0)
15     if value == '0xff6897': #0
16         ledPin.duty_u16(1)
17     elif value == '0xff30cf': #1
18         ledPin.duty_u16(1023)
19     elif value == '0xff18e7': #2
20         ledPin.duty_u16(4096)
21     elif value == '0xff7a85': #3
22         ledPin.duty_u16(10000)
23     else:
24         return
25
26 try:
27     while True:

```

```
28     irValue = recvPin.ir_read()
29     if irValue:
30         print(irValue)
31         handleControl(irValue)
32     except:
33         ledPin.deinit()
```

The handleControl() function is used to execute events corresponding to infrared code values. Every time when the function is called, the buzzer sounds once and determines the brightness of the LED based on the infrared key value. If the key value is not "0", "1", "2", "3", the buzzer sounds once, but the brightness of LED will not change.

```
11 def handleControl(value):
12     buzzerPin.value(1)
13     time.sleep_ms(100)
14     buzzerPin.value(0)
15     if value == '0xff6897': #0
16         ledPin.duty_u16(1)
17     elif value == '0xff30cf': #1
18         ledPin.duty_u16(1023)
19     elif value == '0xff18e7': #2
20         ledPin.duty_u16(4096)
21     elif value == '0xff7a85': #3
22         ledPin.duty_u16(10000)
23     else:
24         return
```

Each time the key value of IR remote is received, function handleControl() will be called to process it.

```
27 while True:
28     irValue = recvPin.ir_read()
29     if irValue:
30         print(irValue)
31         handleControl(irValue)
```

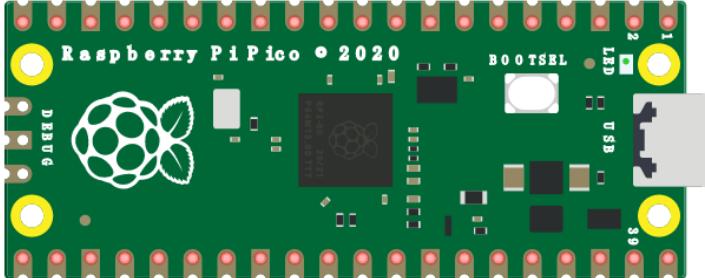
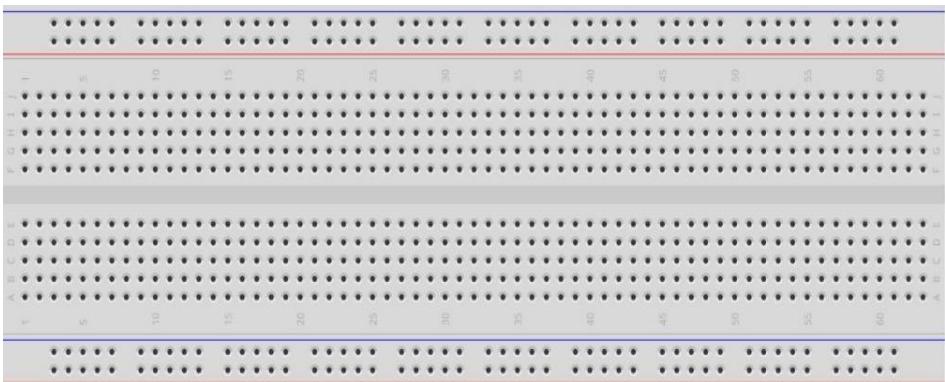
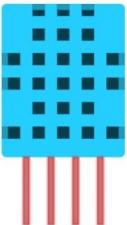
Chapter 25 Hygrothermograph DHT11

In this chapter, we will learn about a commonly used sensor called a Hygrothermograph DHT11.

Project 25.1 Hygrothermograph

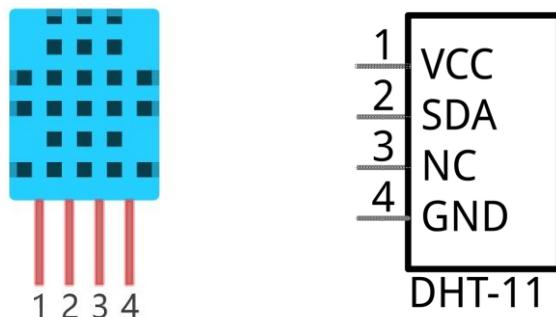
Hygrothermograph is an important tool in our lives to give us data on the temperature and humidity in our environment. In this project, we will use the Raspberry Pi Pico to read Temperature and Humidity data of the DHT11 Module.

Component List

Raspberry Pi Pico x1	USB cable x1	
		
Breadboard x1		
		
Jumper	DHT11 x1	Resistor 10kΩ x1
		

Component knowledge

The Temperature & Humidity Sensor DHT11 is a compound temperature & humidity sensor, and the output digital signal has been calibrated by its manufacturer.



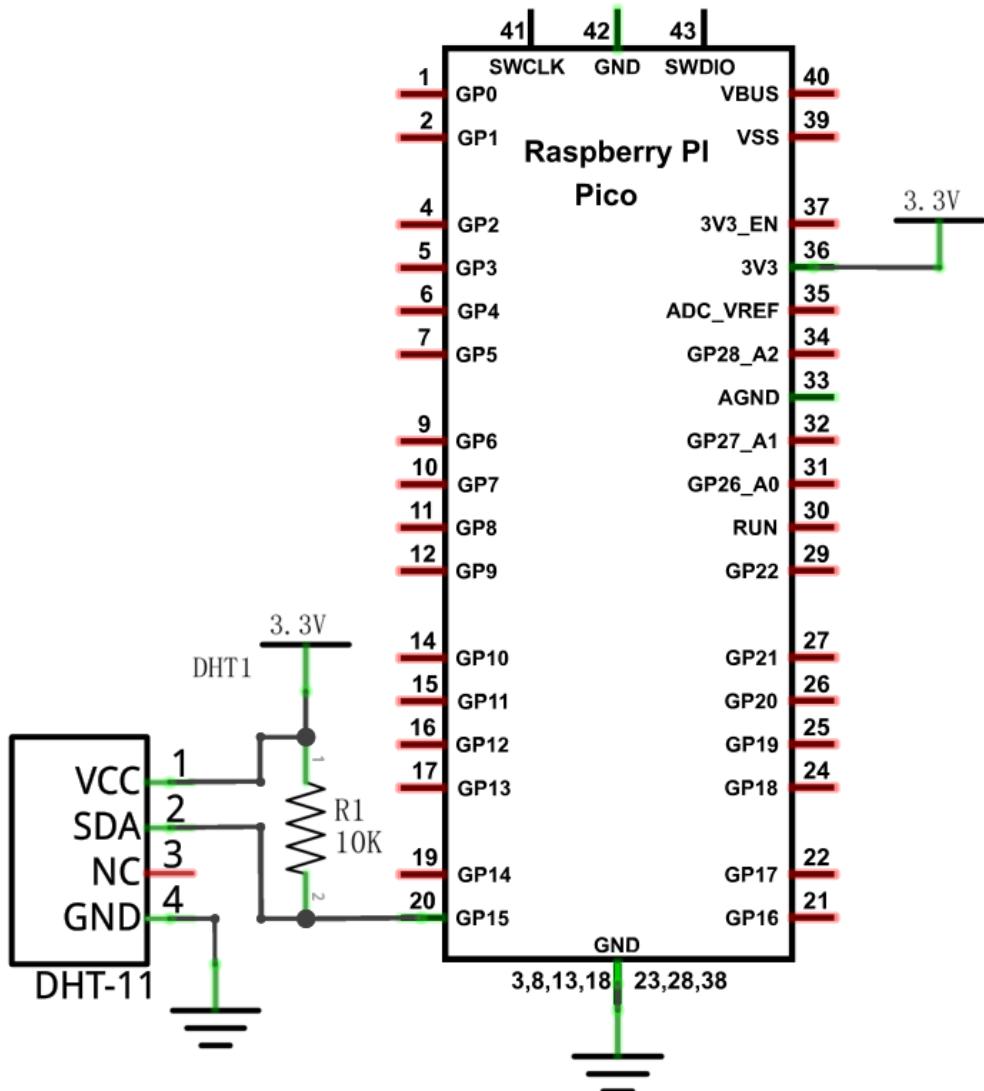
DHT11 uses customized single-line communication protocol, so we can use the library to read data more conveniently.

After being powered up, it will initialize in 1S's time. Its operating voltage is within the range of 3.3V-5.5V. The SDA pin is a data pin, which is used to communicate with other devices.

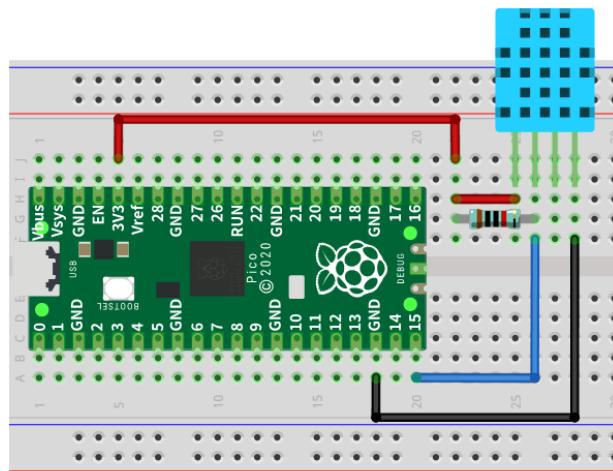
The NC pin (Not Connected Pin) is a type of pin found on various integrated circuit packages. Those pins have no functional purpose to the outside circuit (but may have an unknown functionality during manufacture and test). Those pins should not be connected to any of the circuit connections.

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

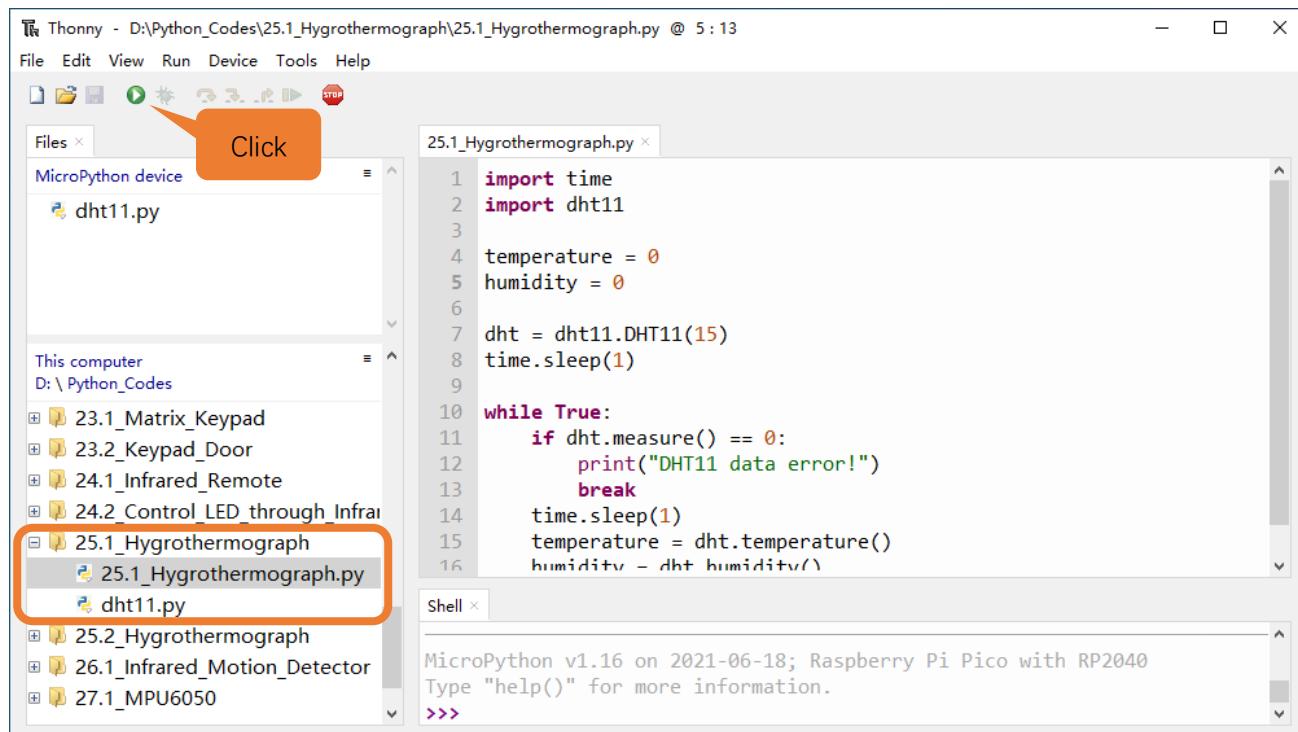


Any concerns? ✉ support@freenove.com

Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “25.1_Hygrothermograph”. Select “dht11.py”, right click your mouse to select “Upload to /”, wait for “dht11.py” to be uploaded to Raspberry Pi Pico and then double click “25.1_Hygrothermograph.py”.

25.1_Hygrothermograph



Make sure your circuit is correctly connected and you will see the following messages printed in “Shell”. Press Ctrl+C or click “Stop/Restart backend” to exit the program.

temperature: 26.02C	humidity: 46.00%
temperature: 26.02C	humidity: 46.00%
temperature: 26.02C	humidity: 46.00%
temperature: 26.05C	humidity: 46.00%
temperature: 26.06C	humidity: 46.00%
temperature: 26.06C	humidity: 46.00%
temperature: 26.07C	humidity: 46.00%



The following is the program code:

```

1 import time
2 import dht11
3
4 temperature = 0
5 humidity = 0
6
7 dht = dht11.DHT11(15)
8 time.sleep(1)
9
10 while True:
11     if dht.measure() == 0:
12         print("DHT11 data error!")
13         break
14     time.sleep(1)
15     temperature = dht.temperature()
16     humidity = dht.humidity()
17     print("temperature: %0.2fC  humidity: %0.2f%%(temperature, humidity) + %")

```

Import time and dht11 modules.

```

1 import time
2 import dht11

```

Create a DHT11 object and bind GP15 as the data transfer pin.

```
7 dht = dht11.DHT11(15)
```

Call the measure() function to start the DHT11 module to measure the temperature and humidity, and use the if statement to determine the returned value of the measure() function. If the returned value is 0, the data fails to be read.

```

11 if dht.measure() == 0:
12     print("DHT11 data error!")
13     break

```

Call the temperature() function and humidity() function to read the temperature and humidity data of DHT module and print the data to "shell".

```

15     temperature = dht.temperature()
16     humidity = dht.humidity()
17     print("temperature: %0.2fC  humidity: %0.2f%%(temperature, humidity) + %")

```

Obtain temperature and humidity data once per second and print them out.

```

10 while True:
11     if dht.measure() == 0:
12         print("DHT11 data error! ")
13         break
14     time.sleep(1)
15     temperature = dht.temperature()
16     humidity = dht.humidity()
17     print("temperature: %0.2fC  humidity: %0.2f%%(temperature, humidity) + %")

```

Reference

Class dht11

Before each use of object **dht11**, please add the statement “**import dht11**” to the top of python file.

DHT11(pin_name): Object of DHT11, pin_name parameter is to set the data output pin connecting to DHT11.

DHT11.measure(): Starts DHT11 to measure temperature and humidity data once.

DHT11.temperature(): Returns the temperature data measured by DHT11.

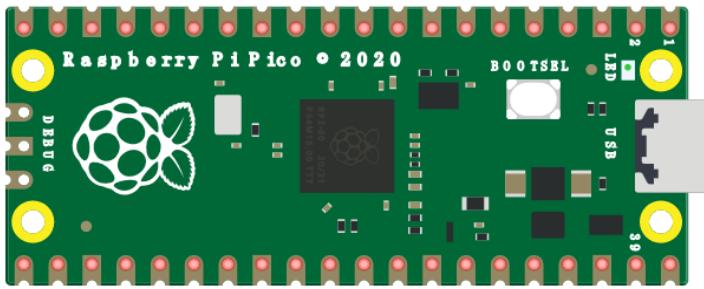
DHT11.humidity(): Returns the humidity data measured by DHT11.

Project 25.2 Hygrothermograph

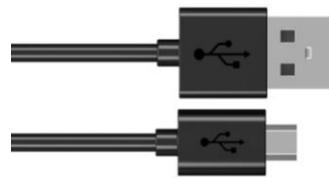
In this project, we use I2C-LCD1602 to display data collected by DHT11.

Component List

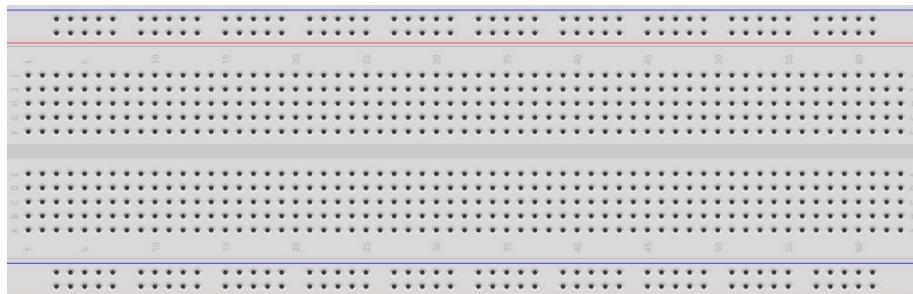
Raspberry Pi Pico x1



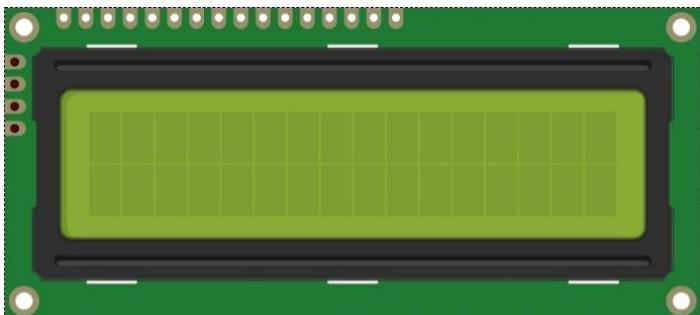
USB cable x1



Breadboard x1



LCD1602 Module x1



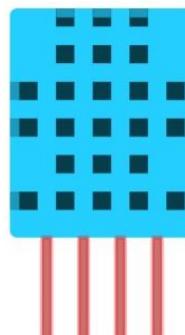
Resistor 10kΩ x1



Jumper

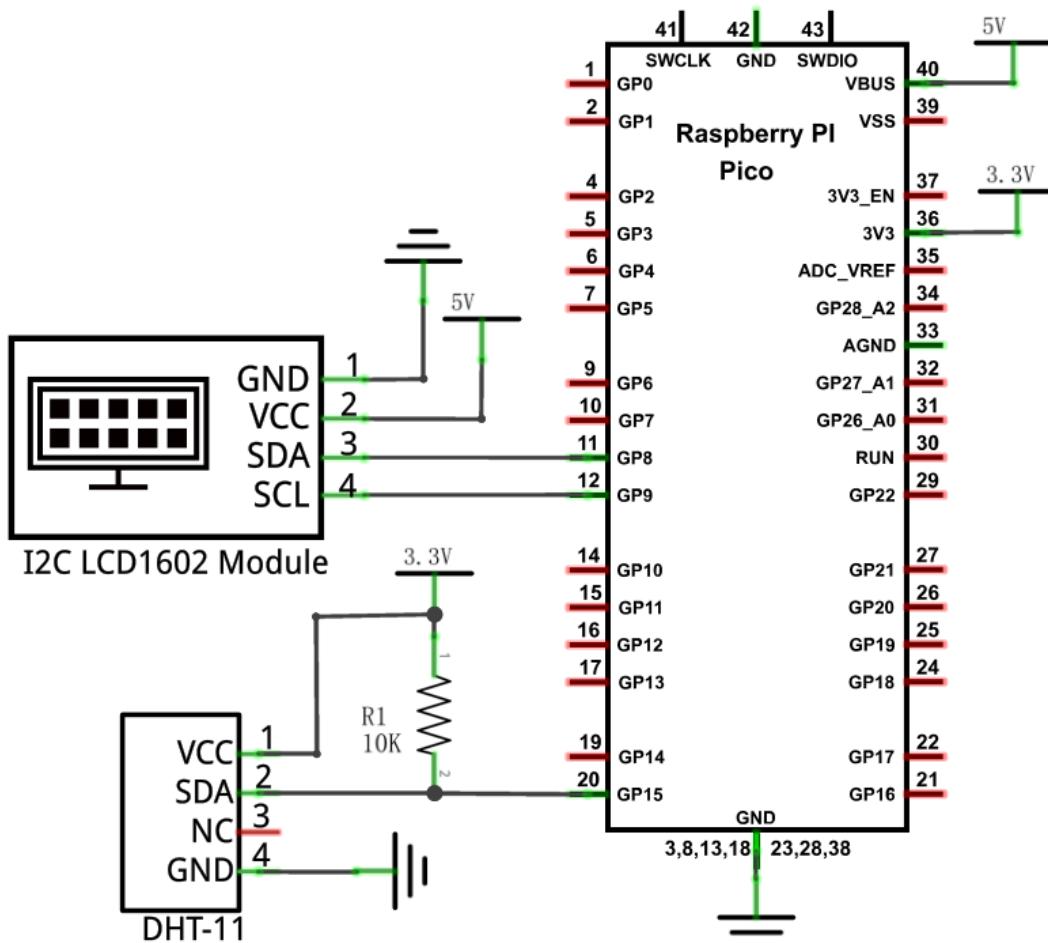


DHT11 x1

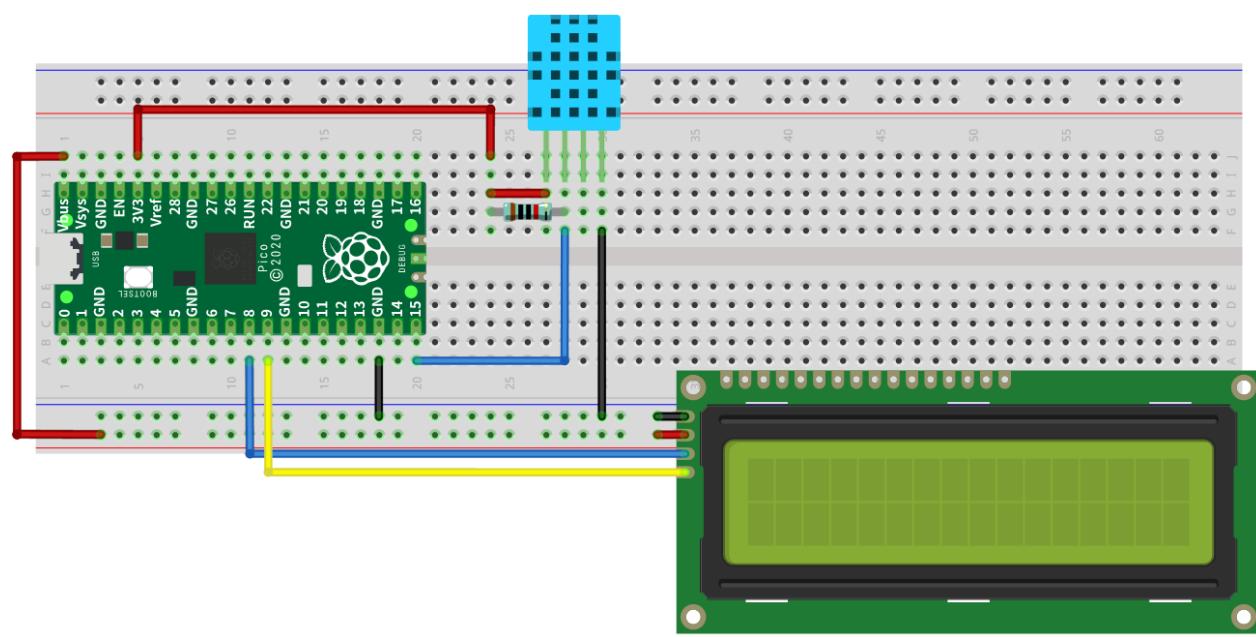


Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Any concerns? ✉ support@freenove.com

Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → 25.2_Hygrothermograph”. Select “dht11.py”、“I2C_LCD.py” and “LCD_API.py”, right click your mouse to select “Upload to /”, wait for “dht11.py”、“I2C_LCD.py” and “LCD_API.py” to be uploaded to Raspberry Pi Pico and then double click “25.2_Hygrothermograph.py”.

25.2_Hygrothermograph

```

import time
from machine import I2C, Pin
from I2C_LCD import I2CLcd
import dht11

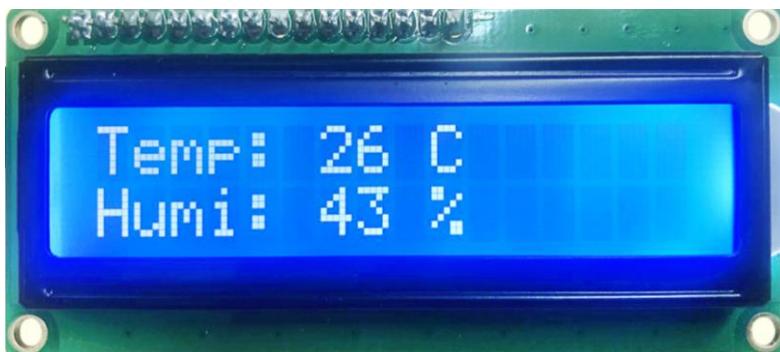
DHT = dht11.DHT11(15)

DEFAULT_I2C_ADDR = 0x27
i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=400000)
lcd = I2CLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)
time.sleep(1)

try:
    while True:
        if DHT.measure() == 0:
            print("DHT11 data error")

```

Click “Run current script”. The first row of LCD1602 is temperature value and the second row is humidity. Try to “pinch” the DHT11 (without touching the leads) with your index finger and thumb for a brief time, you should see that the displayed value on LCD1602 changes. Press Ctrl+C or click “Stop/Restart backend” to exit the program.



The following is the program code:

```

1 import time
2 from machine import I2C, Pin
3 from I2C_LCD import I2CLcd
4 import dht11
5
6 DHT = dht11.DHT11(15)
7
8 DEFAULT_I2C_ADDR = 0x27
9 i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=400000)
10 lcd = I2CLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)
11 time.sleep(1)
12
13 try:
14     while True:
15         if DHT.measure() == 0:
16             print("DHT11 data error")
17             break
18         temp = int(DHT.temperature())
19         humi = int(DHT.humidity())
20         lcd.move_to(0, 0)
21         lcd.putstr("Temp: ")
22         lcd.putstr(str(temp))
23         lcd.putstr(" C")
24         lcd.move_to(0, 1)
25         lcd.putstr("Humi: ")
26         lcd.putstr(str(humi))
27         lcd.putstr(" %")
28         time.sleep(1)
29 except:
30     pass

```

Import dht11 and I2C LCD1602 modules.

```

1 Import time
2 from machine import I2C, Pin
3 from I2C_LCD import I2CLcd
4 import dht11

```

Assign GP15 to DHT11, GP8 and GP9 to LCD1602.

```

6 DHT = dht11.myDHT11(15)
7
8 DEFAULT_I2C_ADDR = 0x27
9 i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=400000)
10 lcd = I2CLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)

```



Obtain data of Hygrothermograph every second and display them on LCD1602. The first line displays temperature and the second line displays humidity.

```
14     while True:  
15         if DHT.measure() == 0:  
16             print("DHT11 data error")  
17             break  
18         temp = int(DHT.temperature())  
19         humi = int(DHT.humidity())  
20         lcd.move_to(0, 0)  
21         lcd.putstr("Temp: ")  
22         lcd.putstr(str(temp))  
23         lcd.putstr(" C")  
24         lcd.move_to(0, 1)  
25         lcd.putstr("Humi: ")  
26         lcd.putstr(str(humi))  
27         lcd.putstr(" %")  
28         time.sleep(1)
```

Chapter 26 Infrared Motion Sensor

In this chapter, we will learn a widely used sensor, Infrared Motion Sensor.

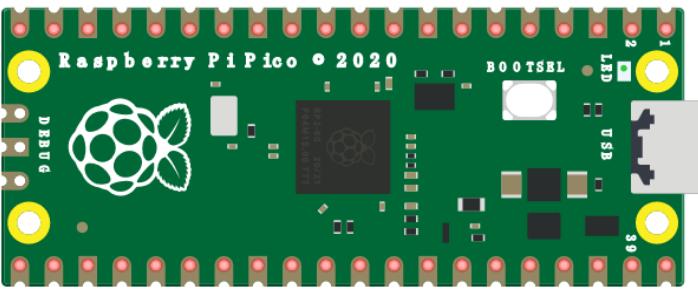
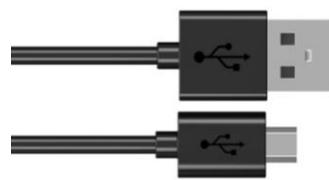
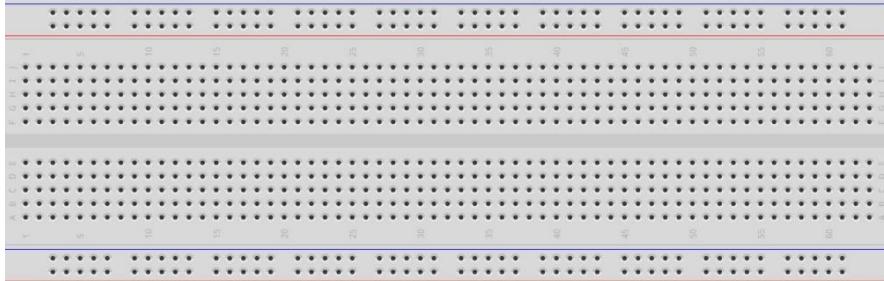
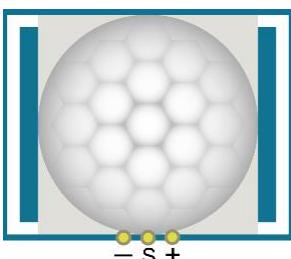
Project 26.1 Infrared Motion Detector with LED Indicator

In this project, we will make a Motion Detector, with the human body infrared pyroelectric sensors.

When someone is in close proximity to the Motion Detector, it will automatically light up and when there is no one close by, it will be out.

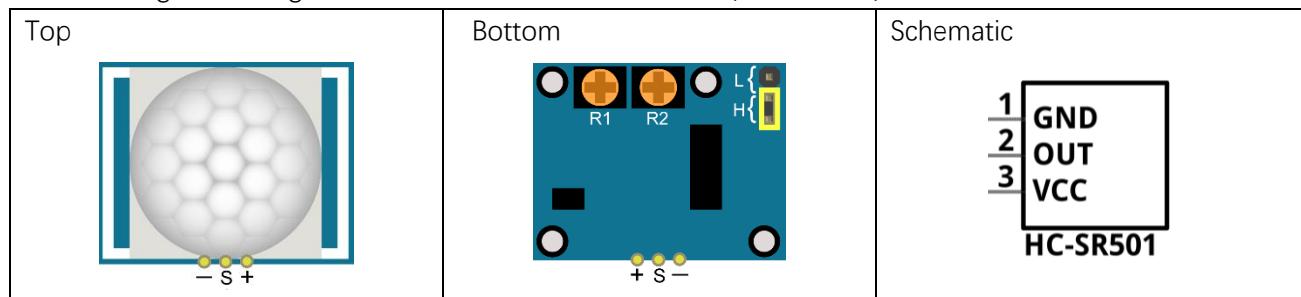
This Infrared Motion Sensor can detect the infrared spectrum (heat signatures) emitted by living humans and animals.

Component List

Raspberry Pi Pico x1		USB cable x1					
Breadboard x1							
HC SR501 x1		LED x1		Resistor 220Ω x1		Jumper	

Component knowledge

The following is the diagram of the infrared Motion sensor (HC SR-501) :



Description:

Working voltage: 5v-20v(DC) Static current: 65uA.

Automatic Trigger: When a living body enters into the active area of sensor, the module will output high level (3.3V). When the body leaves the sensor's active detection area, it will output high level lasting for time period T, then output low level(0V). Delay time T can be adjusted by the potentiometer R1.

According to the position of jumper cap, you can choose non-repeatable trigger mode or repeatable mode.

L: non-repeatable trigger mode. The module output high level after sensing a body, then when the delay time is over, the module will output low level. During high level time, the sensor no longer actively senses bodies.

H: repeatable trigger mode. The distinction from the L mode is that it can sense a body until that body leaves during the period of high level output. After this, it starts to time and output low level after delaying T time.

Induction block time: the induction will stay in block condition and does not induce external signal at lesser time intervals (less than delay time) after outputting high level or low level.

Initialization time: the module needs about 1 minute to initialize after being powered ON. During this period, it will alternately output high or low level.

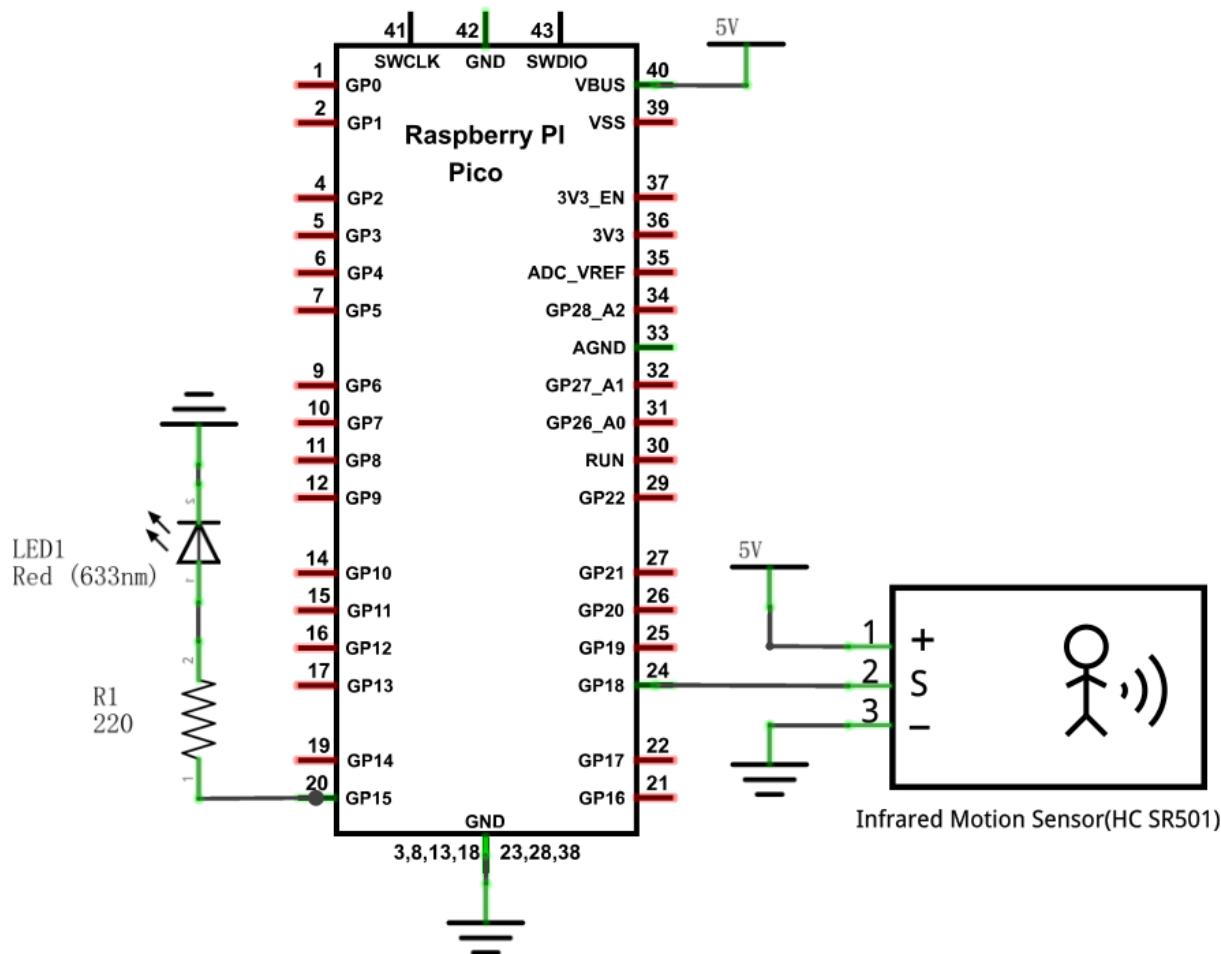
One characteristic of this sensor is when a body moves close to or moves away from the sensor's dome edge, the sensor will work at high sensitivity. When a body moves close to or moves away from the sensor's dome in a vertical direction, the sensor cannot detect well (please take note of this deficiency).

Note: The Sensing Range (distance before a body is detected) is adjusted by the potentiometer.

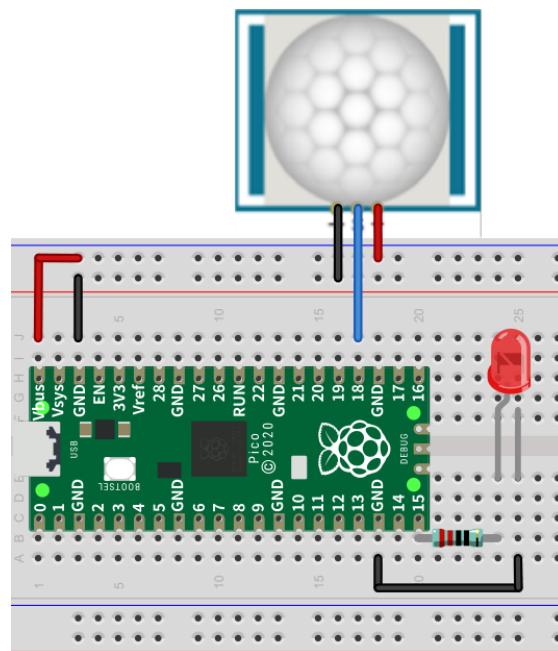
We can regard this sensor as a simple inductive switch when in use.

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



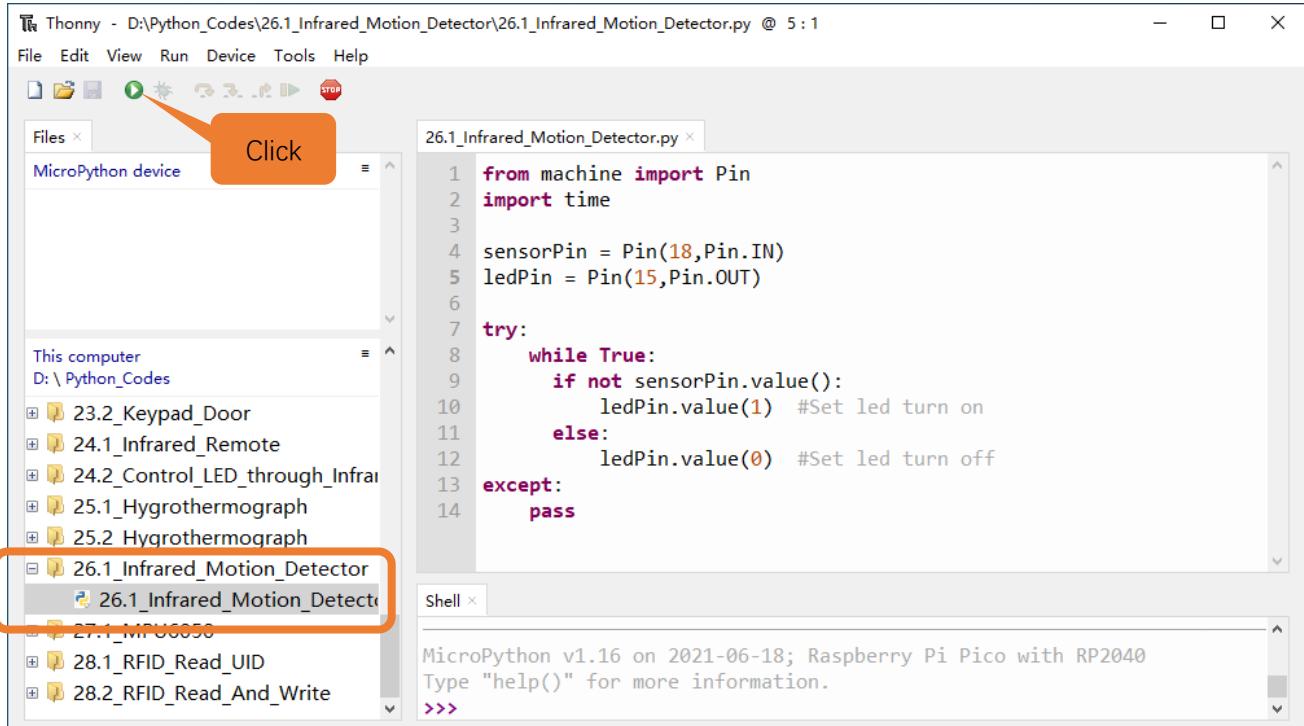
Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

Code

In this project, we will use an infrared motion sensor to trigger an LED, essentially using the infrared motion sensor as a motion switch. So the code of this project is similar to that of project "[Button & Led](#)". The difference is when infrared motion sensor detects changes, it will output high level; when it detects nothing, it will output low level. When the sensor outputs high level, LED turns ON; Otherwise, LED turns OFF.

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "26.1_Infrared_Motion_Detector" and then double click "26.1_Infrared_Motion_Detector.py".

26.1_Infrared_Motion_Detector



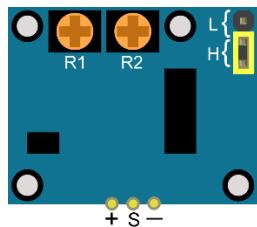
```
from machine import Pin
import time

sensorPin = Pin(18,Pin.IN)
ledPin = Pin(15,Pin.OUT)

try:
    while True:
        if not sensorPin.value():
            ledPin.value(1) #Set led turn on
        else:
            ledPin.value(0) #Set led turn off
except:
    pass
```

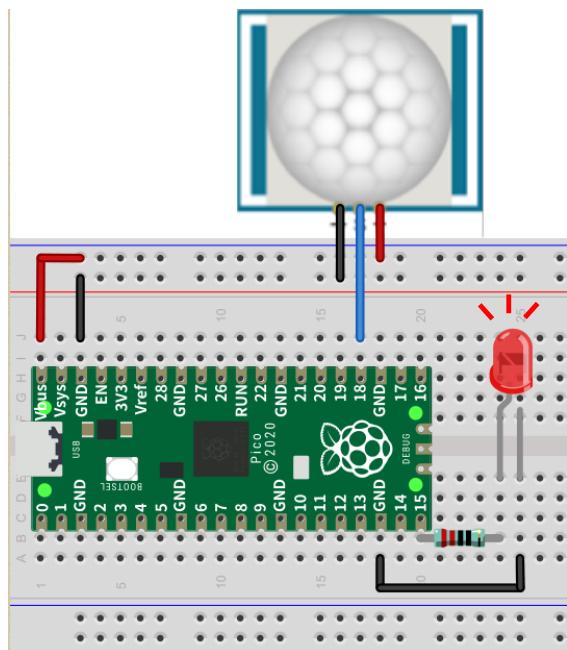
Click "Run current script". Put the sensor on a stationary table and wait for about a minute. Then try to move away from or move closer to the Infrared Motion Sensor and observe whether the LED turns ON or OFF automatically. Press Ctrl+C or click "Stop/Restart backend" to exit the program.

You can rotate the potentiometer on the sensor to adjust the detection effect, or use different modes by changing the jumper.

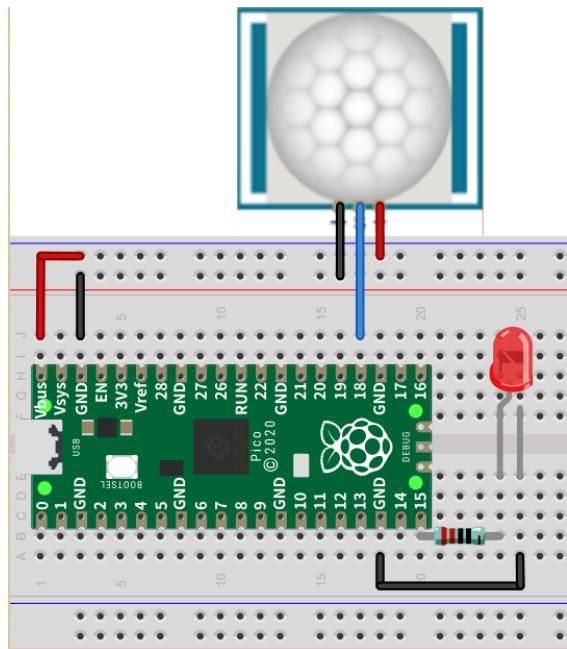


Apart from that, you can also use this sensor to control some other modules to implement different functions by reediting the code, such as the induction lamp, induction door.

Move to the Infrared Motion Sensor



Move away from the Infrared Motion Sensor



The following is the program code:

```

1  from machine import Pin
2  import time
3
4  sensorPin = Pin(18, Pin.IN)
5  ledPin = Pin(15, Pin.OUT)
6
7  try:

```

```
8     while True:  
9         if not sensorPin.value():  
10            ledPin.value(1) #Set led turn on  
11        else:  
12            ledPin.value(0) #Set led turn off  
13    except:  
14        pass
```

Configure the associated pin of the infrared Motion sensor as GP18 and set it to the input state; configure the associated pin of the LED light.

```
4     sensorPin = Pin(18, Pin.IN)  
5     ledPin = Pin(15, Pin.OUT)
```

Call value() to monitor the infrared Motion sensor output for high or low level.

```
9     if not sensorPin.value():
```

When the infrared Motion sensor detects an infrared signal, the LED light is on; when there is no infrared signal, the LED light turns off.

```
9     if not sensorPin.value():  
10       ledPin.value(1) #Set led turn on  
11     else:  
12       ledPin.value(0) #Set led turn off
```

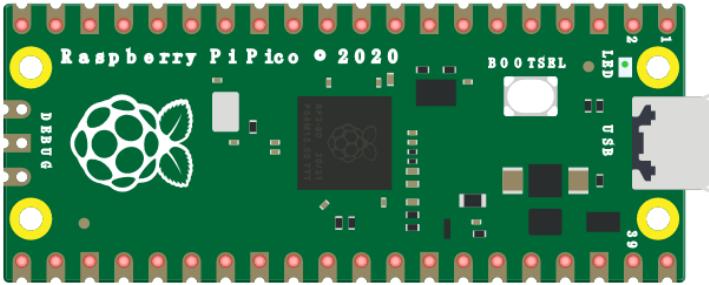
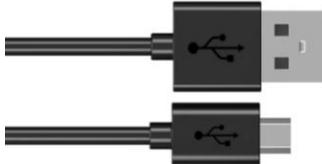
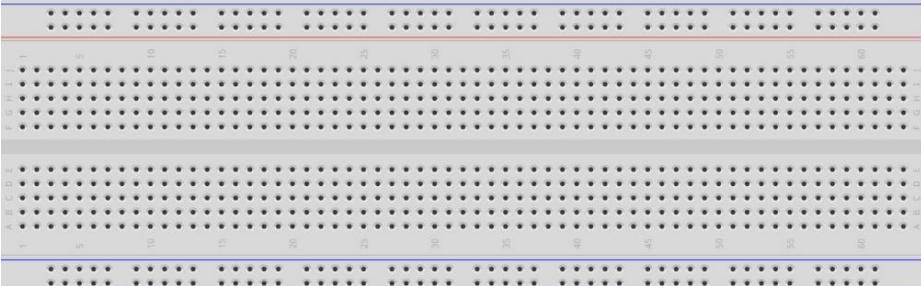
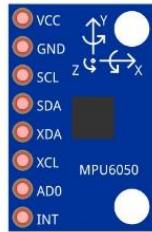
Chapter 27 Attitude Sensor MPU6050

In this chapter, we will learn about an MPU6050 Attitude Sensor which integrates an Accelerometer and Gyroscope.

Project 27.1 Read a MPU6050 Sensor Module

In this project, we will read Acceleration and Gyroscope Data of the MPU6050 Sensor.

Component List

Raspberry Pi Pico x1	USB cable x1
	
Breadboard x1	
	
Jumper	MPU6050 x1
	

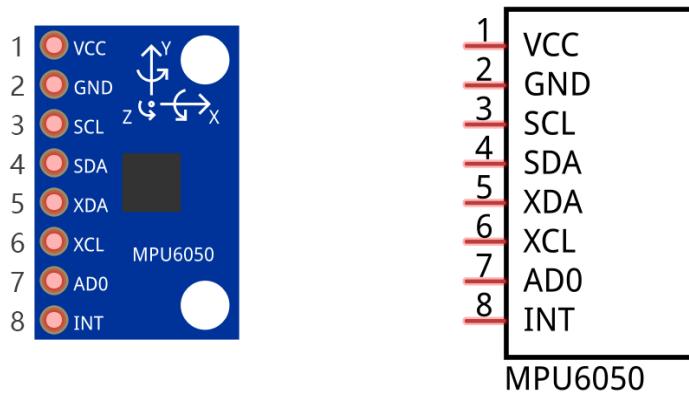


Component knowledge

MPU6050

MPU6050 Sensor Module is a complete 6-axis Motion Tracking Device. It combines a 3-axis Gyroscope, a 3-axis Accelerometer and a DMP (Digital Motion Processor) all in a small package. The settings of the Accelerometer and Gyroscope of MPU6050 can be changed. A precision wide range digital temperature sensor is also integrated to compensate data readings for changes in temperature, and temperature values can also be read. The MPU6050 Module follows the I2C communication protocol and the default address is 0x68.

MPU6050 is widely used to assist with balancing vehicles, robots and aircraft, mobile phones and other products which require stability to control stability and attitude or which need to sense same.



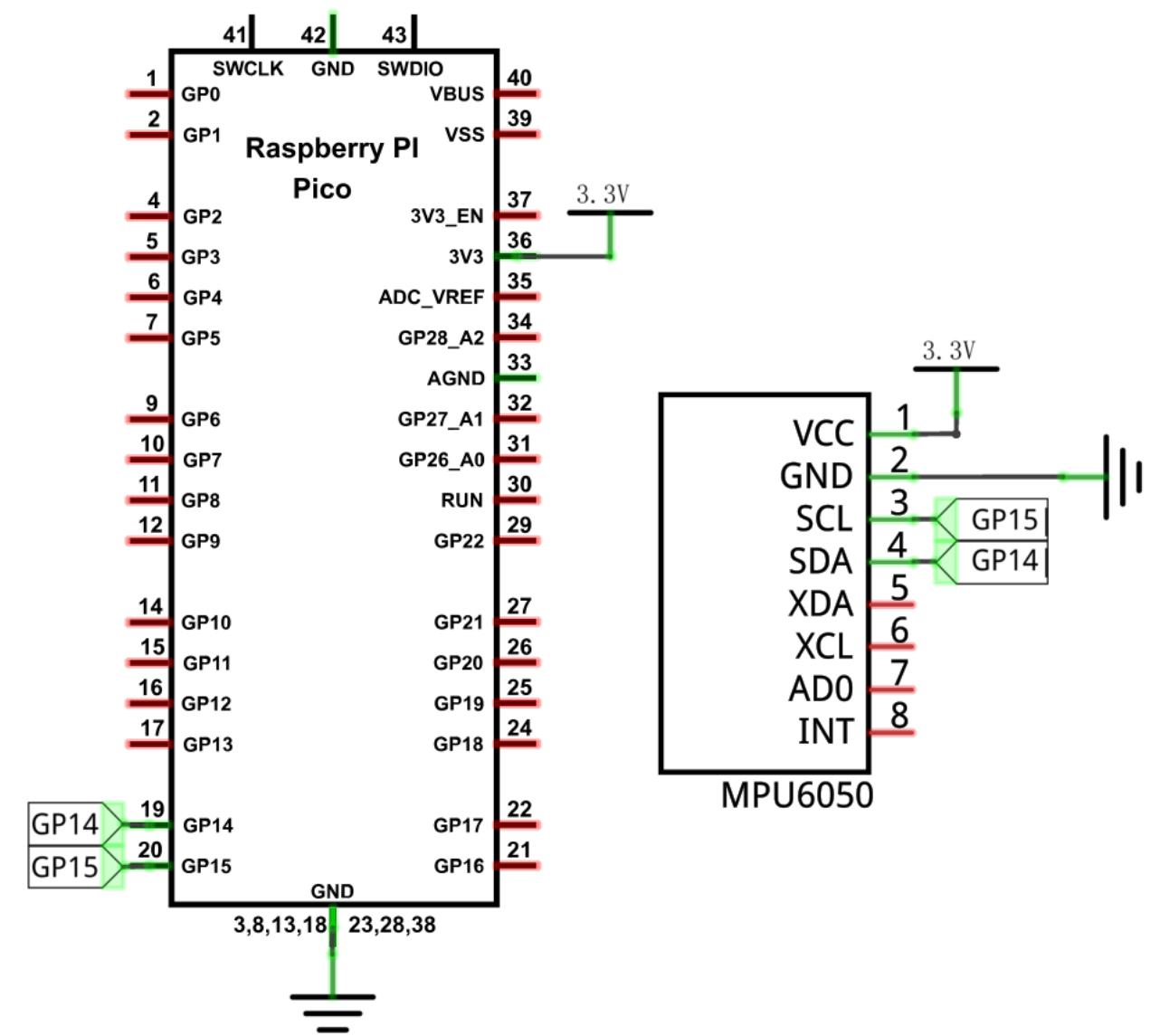
The port description of the MPU6050 module is as follows:

Pin name	Pin number	Description
VCC	1	Positive pole of power supply with voltage 5V
GND	2	Negative pole of power supply
SCL	3	I2C communication clock pin
SDA	4	I2C communication clock pin
XDA	5	I2C host data pin which can be connected to other devices.
XCL	6	I2C host clock pin which can be connected to other devices.
AD0	7	I2C address bit control pin. Low level: the device address is 0x68 High level: the device address is 0x69
INT	8	Output interrupt pin

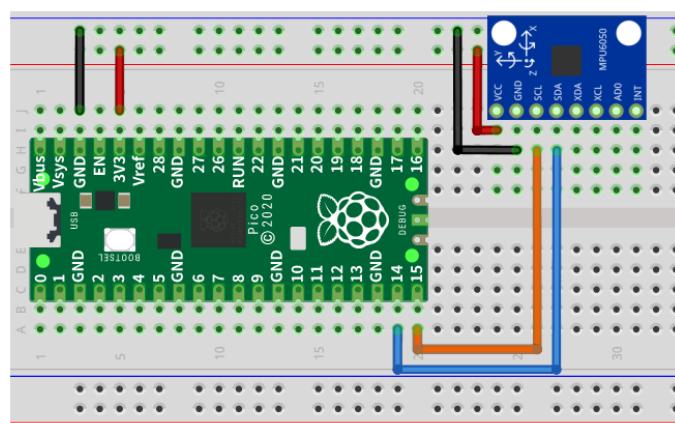
For more details, please refer to datasheet.

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

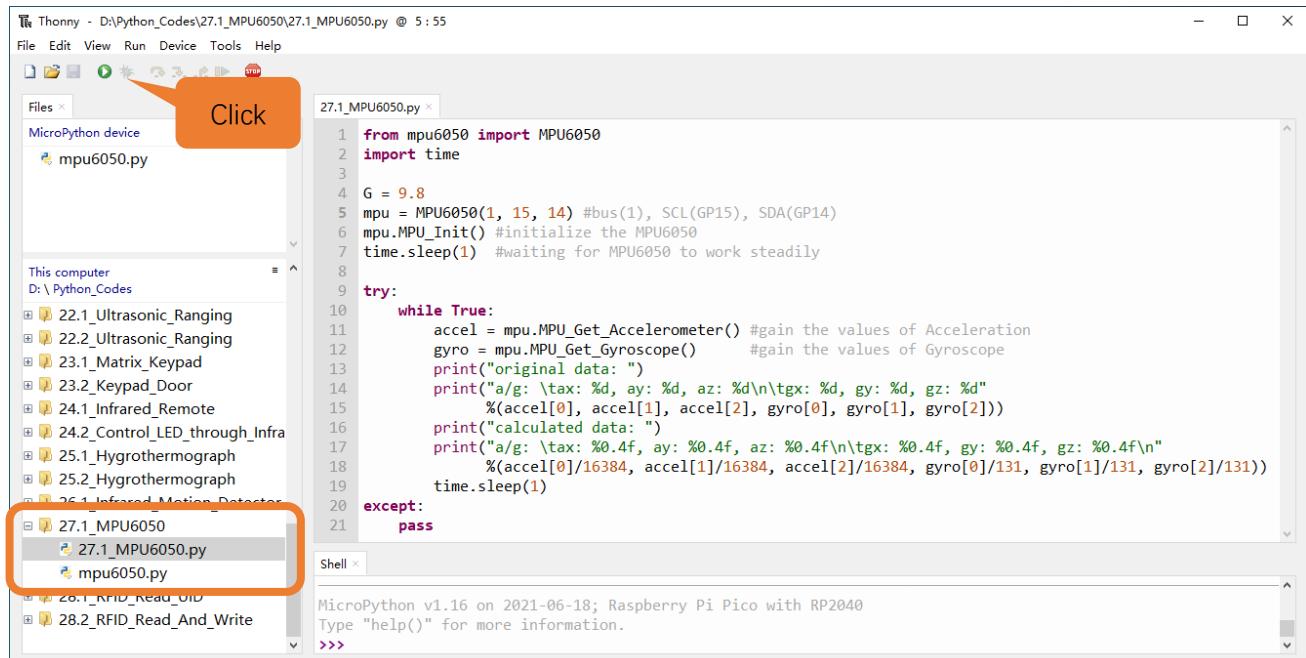


Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “27.1_MPU6050”. Select “mpu6050.py”, right click your mouse to select “Upload to /”, wait for “mpu6050.py” to be uploaded to Raspberry Pi Pico and then double click “27.1_MPU6050.py”.

27.1_MPU6050



Click “Run current script”. When Raspberry Pi Pico obtains acceleration data and gyroscope data of MPU605, it will print them in “Shell”. Press Ctrl+C or click “Stop/Restart backend” to exit the program.

The screenshot shows the MicroPython shell window. It displays the output of the program, which includes two sets of data. The first set is labeled "original data" and the second is labeled "calculated data". The data consists of acceleration (ax, ay, az) and gyroscope (gx, gy, gz) values. The values are printed in both raw form and scaled form (divided by 16384).

```

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

original data:
a/g:      ax: 792, ay: -44, az: 15658
          gx: 454, gy: -55, gz: -22
calculated data:
a/g:      ax: 0.0483, ay: -0.0027, az: 0.9557
          gx: 3.4656, gy: -0.4198, gz: -0.1679

original data:
a/g:      ax: 770, ay: -28, az: 15664
          gx: 454, gy: -56, gz: -22
calculated data:
a/g:      ax: 0.0470, ay: -0.0017, az: 0.9561

```

Note: The data transmission of MPU6050 is very sensitive. Therefore, when using it, please make sure the jumper wire is in good contact, otherwise the data may fail to be obtained.

The following is the program code:

```

1  from mpu6050 import MPU6050
2  import time
3
4  G = 9.8
5  mpu = MPU6050(1, 15, 14) #bus(1), SCL(GP15), SDA(GP14)
6  mpu.MPU_Init() #initialize the MPU6050
7  time.sleep(1) #waiting for MPU6050 to work steadily
8
9  try:
10     while True:
11         accel = mpu.MPU_Get_Accelerometer() #gain the values of Acceleration
12         gyro = mpu.MPU_Get_Gyroscope()       #gain the values of Gyroscope
13         print("original data: ")
14         print("a/g: \tax: %d, ay: %d, az: %d\n\tgx: %d, gy: %d, gz: %d"
15               %(accel[0], accel[1], accel[2], gyro[0], gyro[1], gyro[2]))
16         print("calculated data: ")
17         print("a/g: \tax: %.4f, ay: %.4f, az: %.4f\n\tgx: %.4f, gy: %.4f, gz: %.4f\n"
18               %(accel[0]/16384, accel[1]/16384, accel[2]/16384, gyro[0]/131, gyro[1]/131,
19                 gyro[2]/131))
20         time.sleep(1)
21     except:
22         pass

```

Import MPU6050 and time modules.

```

1  from mpu6050 import MPU6050
2  import time

```

Set I2C pins and associate them with MPU6050 module, and then initialize MPU6050 and wait for the initialization to complete.

```

4  mpu = MPU6050(1, 15, 14) #bus(1), SCL(GP15), SDA(GP14)
5  mpu.MPU_Init() #initialize the MPU6050

```

Obtain the acceleration data of MPU6050 and store it in accel. Obtain the gyroscope data and store it in gyro.

```

11    accel = mpu.MPU_Get_Accelerometer() #gain the values of Acceleration
12    gyro = mpu.MPU_Get_Gyroscope()       #gain the values of Gyroscope

```



Update and collect the original data of the gyroscope every second and print the original data and processed acceleration and angular velocity data in "Shell".

```
10     while True:  
11         accel = mpu.MPU_Get_Accelerometer() #gain the values of Acceleration  
12         gyro = mpu.MPU_Get_Gyroscope()       #gain the values of Gyroscope  
13         print("original data: ")  
14         print("a/g: \tax: %d, ay: %d, az: %d\n\tgx: %d, gy: %d, gz: %d"  
15             %(accel[0], accel[1], accel[2], gyro[0], gyro[1], gyro[2]))  
16         print("calculated data: ")  
17         print("a/g: \tax: %0.4f, ay: %0.4f, az: %0.4f\n\tgx: %0.4f, gy: %0.4f, gz: %0.4f\n"  
18             %(accel[0]/16384, accel[1]/16384, accel[2]/16384, gyro[0]/131, gyro[1]/131,  
19                 gyro[2]/131))  
20         time.sleep(1)
```

Reference

Class mpu6050

Before each use of **mpu6050**, please add the statement "**from mpu6050 import MPU6050**" to the top of the python file.

MPU6050(bus, sclpin, sdapin): Create MPU6050 object and set the bus and pins for I2C.

MPU_Init(): Initialize MPU6050 module.

MPU_Get_Accelerometer(): Obtain original data of MPU6050's acceleration.

MPU_Get_Gyroscope(): Obtain original data of MPU6050's Gyroscope.

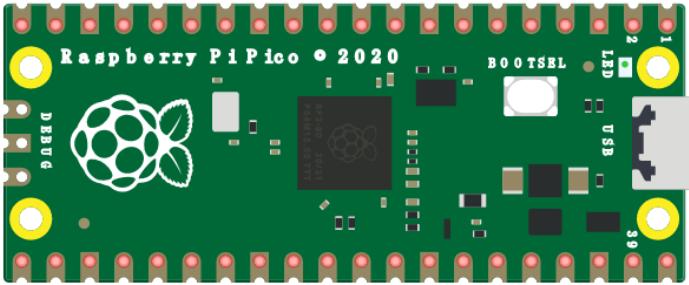
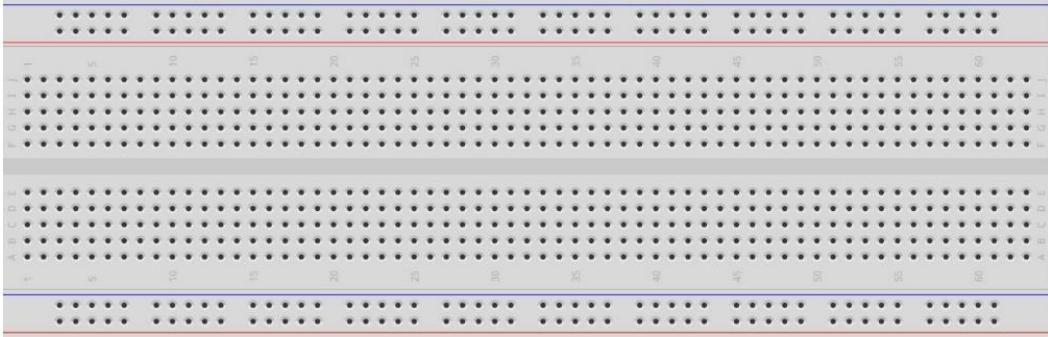
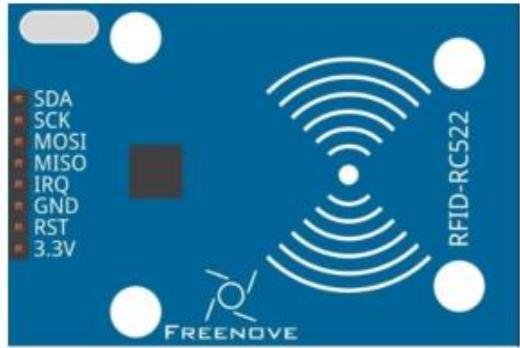
Chapter 28 RFID

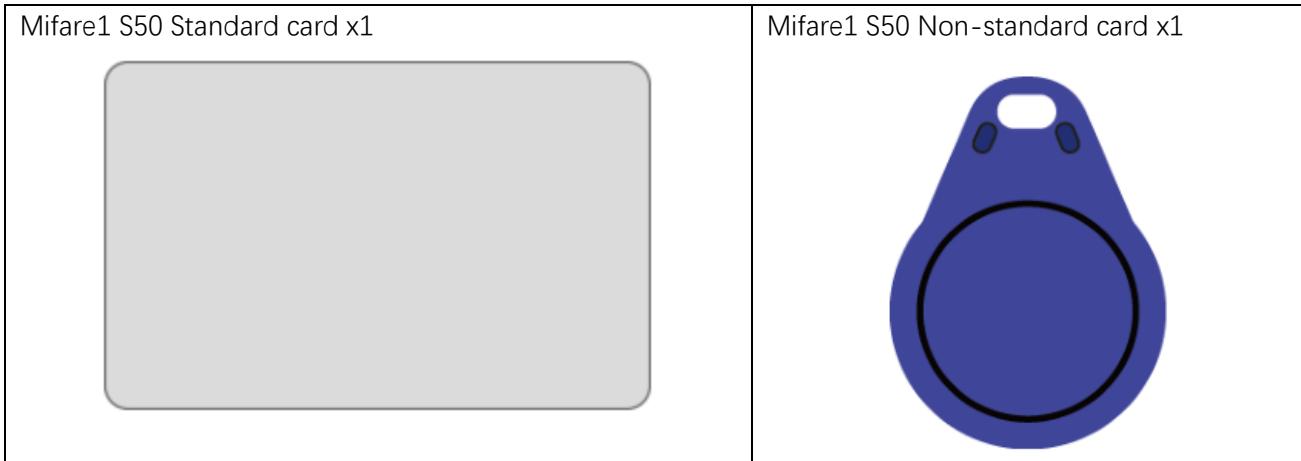
Now, we will learn to use the RFID (Radio Frequency Identification) wireless communication technology.

Project 28.1 RFID read UID

In this project, we will read the unique ID number (UID) of the RFID card, recognize the type of the RFID card and display the information through serial port.

Component List

Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
Jumper		RFID Module(RC522) x1	



Component knowledge

RFID

RFID (Radio Frequency Identification) is a wireless communication technology. A complete RFID system is generally composed of the responder and reader. Generally, we use tags as responders, and each tag has a unique code, which is attached to the object to identify the target object. The reader is a device for reading (or writing) tag information.

Products derived from RFID technology can be divided into three categories: passive RFID products, active RFID products and semi active RFID products. And Passive RFID products are the earliest, the most mature and most widely used products in the market among others. It can be seen everywhere in our daily life such as, the bus card, dining card, bank card, hotel access cards, etc., and all of these belong to close-range contact recognition. The main operating frequency of Passive RFID products are: 125KHZ (low frequency), 13.56MHZ (high frequency), 433MHZ (ultrahigh frequency), 915MHZ (ultrahigh frequency). Active and semi active RFID products work at higher frequencies.

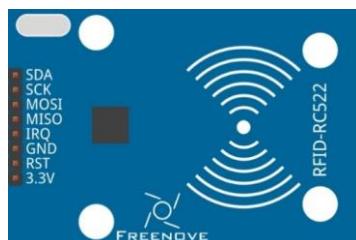
The RFID module we use is a passive RFID product with the operating frequency of 13.56MHz.

MFRC522 RFID Module

The MFRC522 is a highly integrated reader/writer IC for contactless communication at 13.56MHz.

The MFRC522's internal transmitter is able to drive a reader/writer antenna designed to communicate with ISO/IEC 14443 A/MIFARE cards and transponders without additional active circuitry. The receiver module provides a robust and efficient implementation for demodulating and decoding signals from ISO/IEC 14443 A/MIFARE compatible cards and transponders. The digital module manages the complete ISO/IEC 14443A framing and error detection (parity and CRC) functionality.

This RFID Module uses MFRC522 as the control chip, and SPI (Peripheral Interface Serial) as the reserved interface.



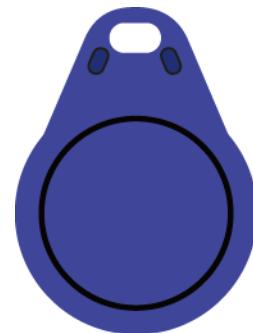
Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

Mifare1 S50 Card

Mifare1 S50 is often called Mifare Standard with the capacity of 1K bytes. And each card has a 4-bytes global unique identifier number (USN/UID), which can be rewritten 100 thousand times and read infinite times. Its storage period can last for 10 years. The ordinary Mifare1 S50 Card and non-standard Mifare1 S50 Card equipped for this kit are shown below.



Mifare1 S50 Standard card



Mifare1 S50 Non-standard card

The Mifare1 S50 capacity (1K byte) is divided into 16 sectors (Sector0-Sector15). Each sector contains 4 data block (Block0-Block3). 64 blocks of 16 sectors will be numbered according to absolute address, from 0 to 63. And each block contains 16 bytes (Byte0-Byte15), $64 \times 16 = 1024$. As is shown in the following table:

Sector No.	Block No.	Storage area	Block type	Absolute block No.
sector 0	block 0	vendor code	vendor block	0
	block 1		data block	1
	block 2		data block	2
	block 3	Password A-access control-password B	control block	3
sector 1	block 0		data block	4
	block 1		data block	5
	block 2		data block	6
	block 3	Password A-access control-password B	control block	7
.....
sector 15	block 0		data block	60
	block 1		data block	61
	block 2		data block	62
	block 3	Password A-access control-password B	control block	63

Each sector has a set of independent password and access control which are put in the last block of each sector, and the block is also known as sector trailer, that is Block 3 in each sector. Sector 0, block 0 (namely absolute address 0) of S50 is used to store the vendor code, which has been solidified and can't be changed, and the card serial number is stored here. In addition to the manufacturer and the control block, the rest of the cards are data blocks, which can be used to store data. Data block can be used for two kinds of applications: (1) used as general data storage and can be operated for reading and writing. (2) used as data value, and can be operated for initializing the value, adding value, subtracting and reading the value.



The sector trailer block in each sector is the control block, including a 6-byte password A, 4-byte access control and 6-byte password B. For example, the control block of a brand new card is as follows:

A0 A1 A2 A3 A4 A5	FF 07 80 69	B0 B1 B2 B3 B4 B5
password A	access control	password B

The default password of a brand new card is generally A0A1A2A3A4A5 for password A, B0B1B2B3B4B5 for password B, or both the password A and password B are 6 FF. Access control is used to set the access conditions for each block (including the control block itself) in a sector.

Blocks of S50 are divided into data blocks and control blocks. There are four operations, "read", "write", "add value", "subtract value (including transmission and storage)" for data blocks, and there are two operations, "read" and "write" for control blocks.

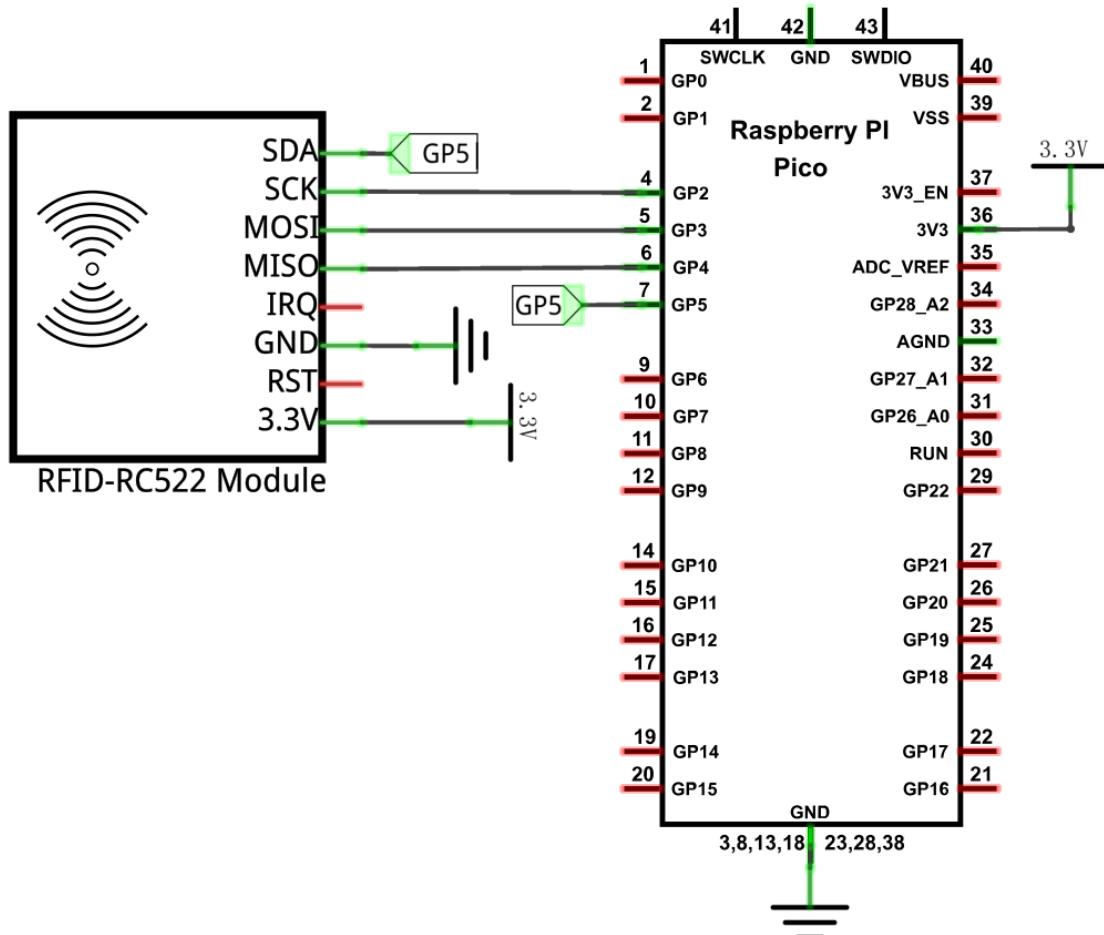
For more details about how to set data blocks and control blocks, please refer to Datasheet.

By default, after verifying password A or password B, we can do reading or writing operation to data blocks. And after verifying password A, we can do reading or writing operation to control blocks. But password A can never be read. If you choose to verify password A and then you forget the password A, the block will never be able to read again. **It is highly recommended that beginners should not try to change the contents of control blocks.**

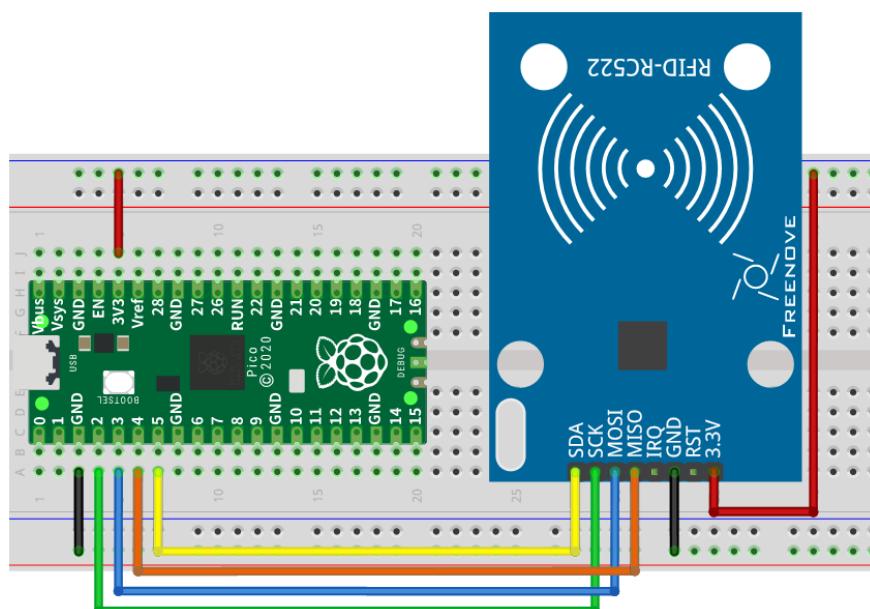
Circuit

The connection of control board and RFID module is shown below.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

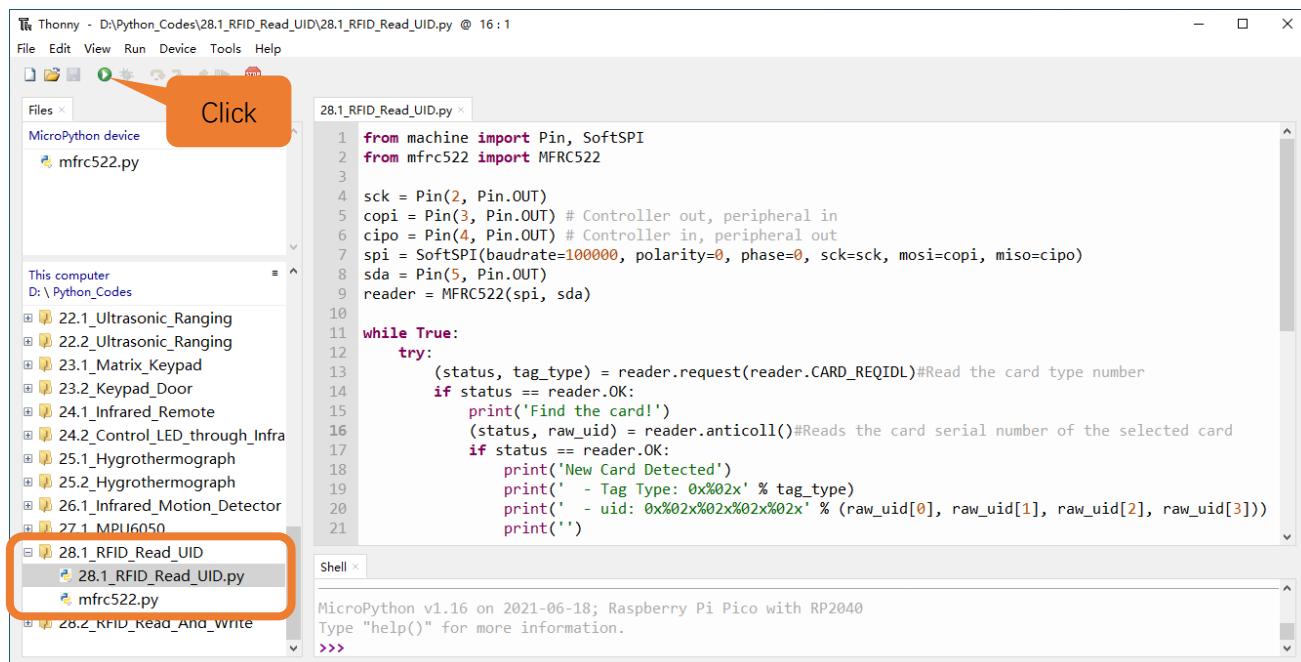


Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

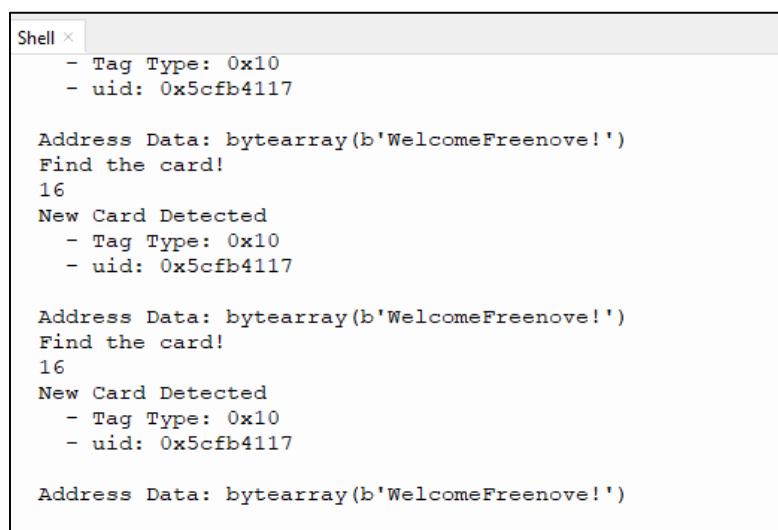
Code

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "28.1_RFID_Read_UID". Select "mfrc522.py", right click to select "Upload to /", wait for "mfrc522.py" to be uploaded to Raspberry Pi Pico and then double click "28.1_RFID_Read_UID.py".

28.1_RFID_Read_UID



Click "Run current script", put the white or blue magnetic card near the RFID module, and Pico will print out the read information such as card type, card UID and card address data to "Shell". Press Ctrl+C or click "Stop/Restart backend" to exit the program.



The following is the program code:

```

1  from machine import Pin, SoftSPI
2  from mfrc522 import MFRC522
3
4  sck = Pin(2, Pin.OUT)
5  copi = Pin(3, Pin.OUT) # Controller out, peripheral in
6  cipo = Pin(4, Pin.OUT) # Controller in, peripheral out
7  spi = SoftSPI(baudrate=100000, polarity=0, phase=0, sck=sck, mosi=copi, miso=cipo)
8  sda = Pin(5, Pin.OUT)
9  reader = MFRC522(spi, sda)
10
11 while True:
12     try:
13         (status, tag_type) = reader.request(reader.CARD_REQIDL) #Read the card type number
14         if status == reader.OK:
15             print('Find the card!')
16             print(tag_type)
17             (status, raw_uid) = reader.anticoll() #Reads the card serial number of the selected
card
18             if status == reader.OK:
19                 print('New Card Detected')
20                 print(' - Tag Type: 0x%02x' % tag_type)
21                 print(' - uid: 0x%02x%02x%02x%02x' % (raw_uid[0], raw_uid[1], raw_uid[2],
raw_uid[3]))
22                 print('')
23                 if reader.select_tag(raw_uid) == reader.OK: #Read card memory capacity
24                     key = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF]
25                     if reader.auth(reader.AUTH, 8, key, raw_uid) == reader.OK: #Verification card
password
26                         print("Address Data: %s" % bytarray(reader.read(8)))
27                         reader.stop_crypto1()
28                     else:
29                         print("AUTH ERROR")
30                 else:
31                     print("FAILED TO SELECT TAG")
32             except KeyboardInterrupt:
33                 break

```

Import Pin, SoftSPI and MFRC522 modules.

```

1  from machine import Pin, SoftSPI
2  from mfrc522 import MFRC522

```

Set SPI Pins and associate them with RFID-RC522 module, and then set MFRC522 module.

```

4  sck = Pin(2, Pin.OUT)
5  copi = Pin(3, Pin.OUT) # Controller out, peripheral in
6  cipo = Pin(4, Pin.OUT) # Controller in, peripheral out

```

```

7   spi = SoftSPI(baudrate=100000, polarity=0, phase=0, sck=sck, mosi=copi, miso=cipo)
8   sda = Pin(5, Pin.OUT)
9   reader = MFRC522(spi, sda)

```

Call the request() function to determine whether the RFID module detects the card. If so, read the model information of the card.

```

13  (status, tag_type) = reader.request(reader.CARD_REQIDL) #Read the card type number

```

Call anticol1() function to read serial number of the selected card and save the data to the raw_uid variable.

```

17  (status, raw_uid) = reader.anticoll() #Reads the card serial number of the selected card

```

Call select_tag() function to determine whether the memory information of the card can be read.

```

23  reader.select_tag(raw_uid)

```

Call the auth() function to verify whether the card's password information is correct.

```

25  reader.auth(reader.AUTH, 8, key, raw_uid)

```

Read the data of the card.

```

26  reader.read(8)

```

Stop reading the card.

```

27  reader.stop_crypto1()

```

The process of reading card information:

First determine whether the RFID module detects the card. If it does, read the model information and UID of the card, and then determine whether the memory information of the card is readable. If it is readable, first verify the password of the card. If the password is correct, read the address data of the card, and print out all the read information to the "Shell".

```

12 try:
13     (status, tag_type) = reader.request(reader.CARD_REQIDL) #Read the card type number
14     if status == reader.OK:
15         print('Find the card!')
16         print(tag_type)
17     (status, raw_uid) = reader.anticoll() #Reads the card serial number of the selected card
18     if status == reader.OK:
19         print('New Card Detected')
20         print(' - Tag Type: 0x%02x' % tag_type)
21         print(' - uid: 0x%02x%02x%02x%02x' % (raw_uid[0], raw_uid[1], raw_uid[2],
22           raw_uid[3]))
23         print('')
24         if reader.select_tag(raw_uid) == reader.OK:#Read card memory capacity
25             key = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF]
26             if reader.auth(reader.AUTH, 8, key, raw_uid) == reader.OK:#Verification card
27               password
28                 print("Address Data: %s" % bytarray(reader.read(8)))
29                 reader.stop_crypto1()
30             else:
31                 print("AUTH ERROR")
32         else:
33             print("FAILED TO SELECT TAG")

```

Reference

Class MFRC522()

Before each use of mfrc522 module, please add the statement “**from mfrc522 import MFRC522**” to the top of the python file.

MFRC522(spi, cs): Creat an object MFRC522.

spi: instance of the SoftSPI class.

cs: Chip select.

Default variables in the MFRC522 class:

 MFRC522.OK = 0

 MFRC522.NO_TAG_ERR = 1

 MFRC522.ERR = 2

 MFRC522.CARD_REQIDL = 0x26

 MFRC522.AUTH = 0x60

MFRC522.init(): Initialize MFRC522 module.

MFRC522.reset(): Reset MFRC522 module.

MFRC522.request(mode): Request to read card data. The function returns a status code and an int. The status code is used to check whether a card is being read. The data of the int type is the model data of the card.

mode: CARD_REQIDL = 0x26.

MFRC522.anticol1(): Read the serial number of the card. The function returns a list and a status code. The data in the list is the serial number of the card.

MFRC522.select_tag(): Check whether the memory information of the card can be read.

MFRC522.auth(mode, addr, sect, serial_number): The function is used to verify the card password, and returns a boolean value. If it is correct, it returns True, if it is wrong, it returns False.

mode: AUTH = 0x60.

addr: data address.

sect: password of the card.

serial_number: serial number of the card.

MFRC522.read(addr): read data in addr address.

MFRC522.write(addr, data): Function to write card data. addr is the address of the data to be written and data is the data to be written. Returns a status code.

MFRC522.Read_Data(sect, serial_number): Read all the data written to the card and print it out.

sect: password of the card.

serial_number: serial number of the card.

MFRC522.Write_Data(sect, serial_number, datas): Write datas data into the card, the original data in the card will be overwritten.

MFRC522.Clear_Data(sect, serial_number): Clear the data written to the card.

MFRC522.Write_Row_Data(sect, serial_number, addr, datas): Write the datas data to the addr address.

MFRC522.Read_Row_Data(sect, serial_number, addr): Read the data at the addr address and print it out.

MFRC522.Clear_Row_Data(sect, serial_number, addr): Clear the data of the addr address.

Class SoftSPI

Before each use of SoftSPI module, please add the statement “**from machine import SoftSPI**” to the top of the python file.

Machine.SoftSPI(id, baudrate, polarity, phase, bits, sck, mosi, miso): construct a new software SPI object.
Additional parameters must be give, usually at least id, baudrate, sck, mosi, miso.

SoftSPI.init(baudrate, polarity, phase, bits, sck, mosi, miso):

Baudrate: the SCK clock rate.

Polarity: It is set as 0 or 1, which represents the level of the clock line when it is idle.

Phase: It is set as 0 or 1, to sample data on the first or second clock edge, respectively.

sck, mosi, miso: Corresponding pins of SPI bus in Raspberry Pi Pico.

SoftSPI.deinit(): Close the SoftSPI bus.

SoftSPI.read(nbytes, write=0): Read a number of bytes specified by nbytes while continuously writing the single byte given by write. Returns a bytes object with the data that was read.

SoftSPI.readinto(buf, write=0): Read into the buffer specified by buf while continuously writing the single byte given by write. Returns None.

SoftSPI.write(buf): Write the bytes contained in buf. Returns None.

SoftSPI.write_readinto(write_buf, read_buf): Write the bytes from write_buf while reading into read_buf. The buffers can be the same or different, but both buffers must have the same length. Returns None.

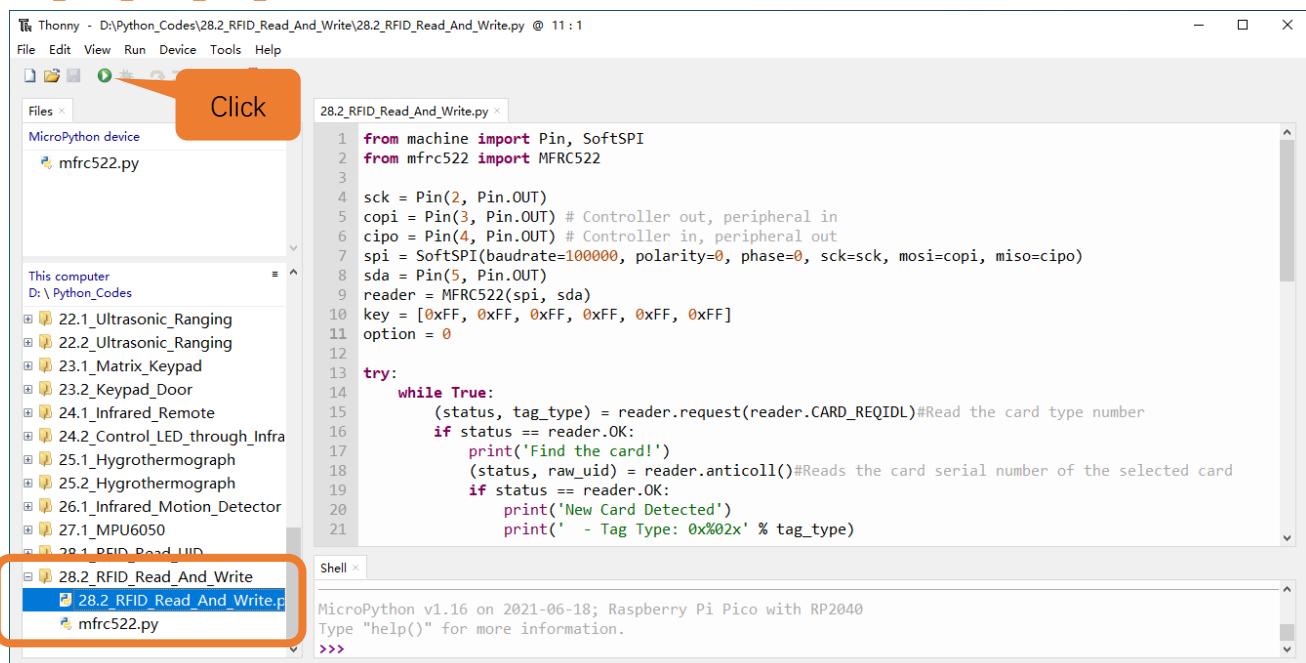
Project 28.2 RFID Read and Write

In this project, we will do reading and writing operations to the card.

Code

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "28.2_RFID_Read_And_Write". Select "mfrc522.py", right click to select "Upload to /", wait for "mfrc522.py" to be uploaded to Raspberry Pi Pico and then double click "28.2_RFID_Read_And_Write.py".

282_RFID_Read_And_Write



Click "Run current script", put the white magnetic card or blue magnetic card close to the induction area of the RFID module, the "Shell" window prints the read information such as card UID, card type and other data. It also provides the option to perform regular read and write operations on the card. Press Ctrl+C or click "Stop/Restart backend" to exit the program.

```

>>> %Run -c $EDITOR_CONTENT
Find the card!
New Card Detected
- Tag Type: 0x10
- uid: 0xfc804849

**Please enter the options**
1: Read the card data
2: Write data to the card
3: Clear data
4: Read a row of data
5: Write a line of data
6: Clear a line of data
7: Exit or Ctrl + C

```

For example: Option 1 is to read all the data of the card.

```
>>> %Run -c $EDITOR_CONTENT
Find the card!
New Card Detected
- Tag Type: 0x10
- uid: 0xfc804849

**Please enter the options**
1: Read the card data
2: Write data to the card
3: Clear data
4: Read a row of data
5: Write a line of data
6: Clear a line of data
7: Exit or Ctrl + C
Input option: 1

1: 123566666666
reading end
```

Option2 is to write data to the card.

```
2: Write data to the card
3: Clear data
4: Read a row of data
5: Write a line of data
6: Clear a line of data
7: Exit or Ctrl + C
Input option: 2

please input datas: Hello World! Welcome Freenove!
Write card OK!

**Please enter the options**
1: Read the card data
2: Write data to the card
3: Clear data
4: Read a row of data
5: Write a line of data
6: Clear a line of data
7: Exit or Ctrl + C
Input option: 1

1: Hello World! Wel
2: come Freenove!
reading end
```

Option3 is to clear data.

```
3: Clear data
4: Read a row of data
5: Write a line of data
6: Clear a line of data
7: Exit or Ctrl + C
Input option: 3

Clear Data!

**Please enter the options**
1: Read the card data
2: Write data to the card
3: Clear data
4: Read a row of data
5: Write a line of data
6: Clear a line of data
7: Exit or Ctrl + C
Input option: 1

reading end
No Data!
```

The following is the program code:

```
1  from machine import Pin, SoftSPI
2  from mfrc522 import MFRC522
3
4  sck = Pin(2, Pin.OUT)
5  copi = Pin(3, Pin.OUT) # Controller out, peripheral in
6  cipo = Pin(4, Pin.OUT) # Controller in, peripheral out
7  spi = SoftSPI(baudrate=100000, polarity=0, phase=0, sck=sck, mosi=copi, miso=cipo)
8  sda = Pin(5, Pin.OUT)
9  reader = MFRC522(spi, sda)
10 key = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF]
11 option = 0
12
13 try:
14     while True:
15         (status, tag_type) = reader.request(reader.CARD_REQIDL)#Read the card type number
16         if status == reader.OK:
17             print('Find the card!')
18             (status, raw_uid) = reader.anticoll()#Reads the card serial number of the selected
19             card
20             if status == reader.OK:
21                 print('New Card Detected')
22                 print(' - Tag Type: 0x%02x' % tag_type)
23                 print(' - uid: 0x%02x%02x%02x%02x' % (raw_uid[0], raw_uid[1], raw_uid[2],
24                 raw_uid[3]))
25                 print('')
26                 if reader.select_tag(raw_uid) == reader.OK:#Read card memory capacity
27                     while option != "7":
28                         print("**Please enter the options**")
29                         print("1: Read the card data")
30                         print("2: Write data to the card")
31                         print("3: Clear data")
32                         print("4: Read a row of data")
33                         print("5: Write a line of data")
34                         print("6: Clear a line of data")
35                         print("7: Exit or Ctrl + C")
36                         option = input("Input option: ")
37                         print("")
38                         if option == "1":
39                             reader.Read_Data(key, raw_uid)
40                         elif option == "2":
41                             datas = input("plase input datas: ")
42                             reader.Write_Data(key, raw_uid, datas)
43                         elif option == "3":
```

```

42         reader.Clear_Data(key, raw_uid)
43     elif option == "4":
44         row_index = int(input("Please enter the row index: "))
45         reader.Read_Row_Data(key, raw_uid, row_index)
46     elif option == "5":
47         row_index = int(input("Please enter the row index: "))
48         datas = input("please input datas: ")
49         reader.Write_Row_Data(key, raw_uid, row_index, datas)
50     elif option == "6":
51         row_index = int(input("Please enter the row index: "))
52         reader.Clear_Row_Data(key, raw_uid, row_index)
53     elif option == "7":
54         reader.stop_cryptol()
55         print("Exit the success")
56     else:
57         print("Input error")
58         print("")
59         break
60 except KeyboardInterrupt:
61     pass

```

Print the function of each option, use the option variable to receive the options input by the user.

```

26     print("**Please enter the options**")
27     print("1: Read the card data")
28     print("2: Write data to the card")
29     print("3: Clear data")
30     print("4: Read a row of data")
31     print("5: Write a line of data")
32     print("6: Clear a line of data")
33     print("7: Exit or Ctrl + C")
34     option = input("Input option: ")

```

Read_Data() function reads data in the card (excluding the password).

```
37     reader.Read_Data(key, raw_uid)
```

Call Write_Data() to continuously write data to the card. Call the function again will overwrite data previously input.

```
40     reader.Write_Data(key, raw_uid, datas)
```

Clear_Data() function is used to clear all input data in the card.

```
42     reader.Clear_Data(key, raw_uid)
```

Call Read_Row_Data() function to read data in designated address.

```
45     reader.Read_Row_Data(key, raw_uid, row_index)
```

Write_Row_Data() function writes data to designated address.

```
49     reader.Write_Row_Data(key, raw_uid, row_index, datas)
```

Clear_Row_Data() clears data of designated address.

```
52     reader.Clear_Row_Data(key, raw_uid, row_index)
```

Based on the prompt, users can input option in "Shell" to read, write, or erase data on the card, and output the result in the "Shell". You can exit the program by inputting 7 or pressing Ctrl + C.

```
25     while option != "7":  
26         print("Please enter the options**")  
27         print("1: Read the card data")  
28         print("2: Write data to the card")  
29         print("3: Clear data")  
30         print("4: Read a row of data")  
31         print("5: Write a line of data")  
32         print("6: Clear a line of data")  
33         print("7: Exit or Ctrl + C")  
34         option = input("Input option: ")  
35         print("")  
36         if option == "1":  
37             reader.Read_Data(key, raw_uid)  
38         elif option == "2":  
39             datas = input("plase input datas: ")  
40             reader.Write_Data(key, raw_uid, datas)  
41         elif option == "3":  
42             reader.Clear_Data(key, raw_uid)  
43         elif option == "4":  
44             row_index = int(input("Please enter the row index: "))  
45             reader.Read_Row_Data(key, raw_uid, row_index)  
46         elif option == "5":  
47             row_index = int(input("Please enter the row index: "))  
48             datas = input("plase input datas: ")  
49             reader.Write_Row_Data(key, raw_uid, row_index, datas)  
50         elif option == "6":  
51             row_index = int(input("Please enter the row index: "))  
52             reader.Clear_Row_Data(key, raw_uid, row_index)  
53         elif option == "7":  
54             reader.stop_crypto1()  
55             print("Exit the success")  
56         else:  
57             print("Input error")  
58         print("")  
59     break
```



What's Next?

THANK YOU for participating in this learning experience!

We have reached the end of this Tutorial. If you find errors, omissions or you have suggestions and/or questions about the Tutorial or component contents of this Kit, please feel free to contact us:
support@freenove.com

We will make every effort to make changes and correct errors as soon as feasibly possible and publish a revised version.

If you want to learn more about Arduino, Raspberry Pi, Smart Cars, Robotics and other interesting products in science and technology, please continue to visit our website. We will continue to launch fun, cost-effective, innovative and exciting products.

<https://www.freenove.com/>

Thank you again for choosing Freenove products.

Any concerns? support@freenove.com