

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.....	3
Raspberry Pi Pico.....	4
Raspberry Pi Pico W	8
Raspberry Pi Pico 2.....	11
Raspberry Pi Pico 2W.....	14
Chapter 0 Getting Ready (Important).....	17
0.1 Installing Thonny (Important)	17
0.2 Basic Configuration of Thonny.....	22
0.3 Burning Micropython Firmware (Important).....	24
0.4 Thonny Connected to Raspberry Pi Pico	28
0.5 Testing codes (Important).....	32
0.6 Thonny Common Operation.....	43
0.7 Paste the Sticker on the Breadboard	48
Chapter 1 LED (Important).....	49
Project 1.1 Blink	49
Project 1.2 Blink	59
Chapter 2 Button & LED	67
Project 2.1 Button & LED	68
Project 2.2 MINI table lamp.....	72
Chapter 3 LED Bar	75
Project 3.1 Flowing Light	75
Chapter 4 Analog & PWM	81
Project 4.1 Breathing LED.....	81
Project 4.2 Meteor Flowing Light	87
How to import a custom python module.....	91
Chapter 5 RGBLED.....	93
Project 5.1 Random Color Light.....	93
Project 5.2 Gradient Color Light.....	99

Chapter 6 Buzzer	102
Project 6.1 Doorbell.....	102
Project 6.2 Alertor	108
Chapter 7 Serial Communication.....	112
Project 7.1 Serial Print.....	112
Project 7.2 Serial Read and Write	117
Chapter 8 AD Converter.....	121
Project 8.1 Read the Voltage of Potentiometer.....	121
Chapter 9 Potentiometer & LED	127
Project 9.1 Soft Light.....	127
Chapter 10 Photoresistor & LED	131
Project 10.1 Control LED through Photoresistor.....	131
Chapter 11 Thermistor	136
Project 11.1 Thermometer	136
Chapter 12 WiFi Working Modes (Only for Pico W/Pico 2W)	141
Project 12.1 Station mode.....	141
Project 12.2 AP mode.....	146
Project 12.3 AP+Station mode	150
Chapter 13 TCP/IP (Only for Pico W/Pico 2W).....	154
Project 13.1 as Client	154
Project 13.2 as Server	167
Chapter 14 Control LED with Web (Only for Pico W/Pico 2W)	173
Project 14.1 Control_LED_through_Web.....	173
Chapter 15 Bluetooth (Only for Pico W/Pico 2W).....	180
Project 15.1 Bluetooth Low Energy Data Passthrough.....	180
Project 15.2 Bluetooth Low Energy Control LED	192
What's Next?	200

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 have not downloaded the related material for Raspberry Pi Pico tutorial, you can download it from this link:

https://github.com/Freenove/Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico

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

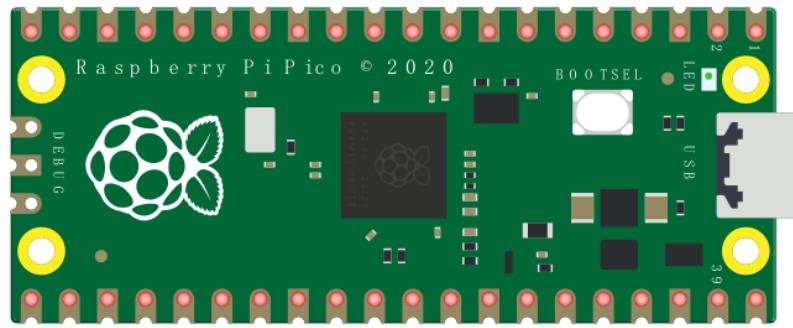
- 1, Component list: helps users to learn and find what components needed in each project.
- 2, Component knowledge: allows you to learn the features and usage of the components.
- 3, Circuit: assists to build circuit for each project.
- 4, Code and annotation: makes it easier for users to learn to use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

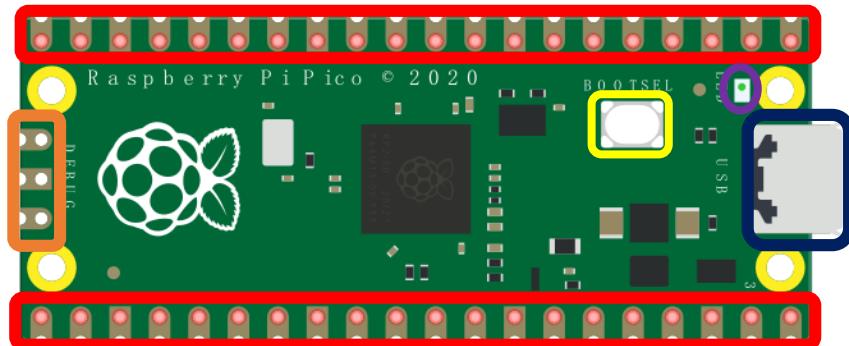
If you have any problems or difficulties using this product, please contact us for quick and free technical support: support@freenove.com

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.

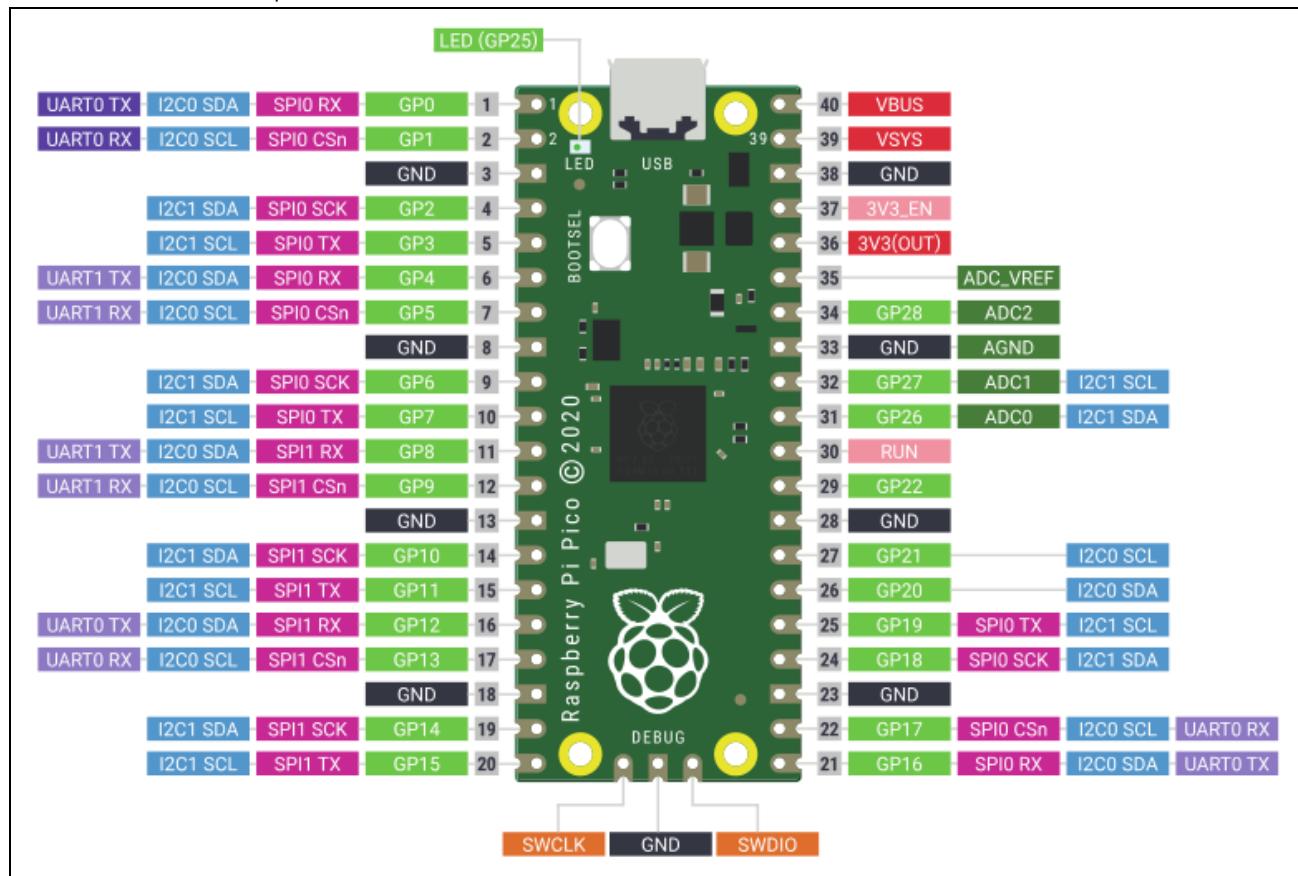


The hardware interfaces are distributed as follows:



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
Purple	UART(default)	Lavender	UART
Magenta	SPI	Blue	I2C
Pink	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

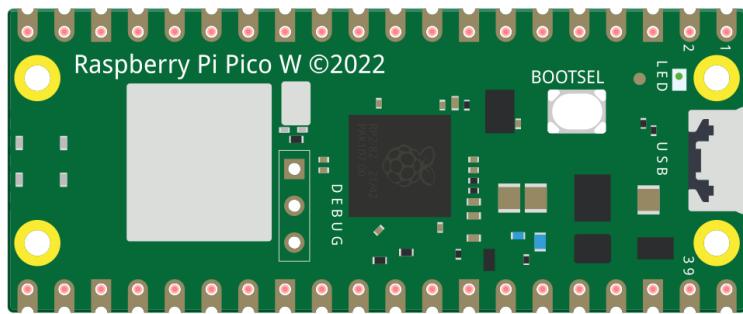
For more detailed information, please refer to:

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

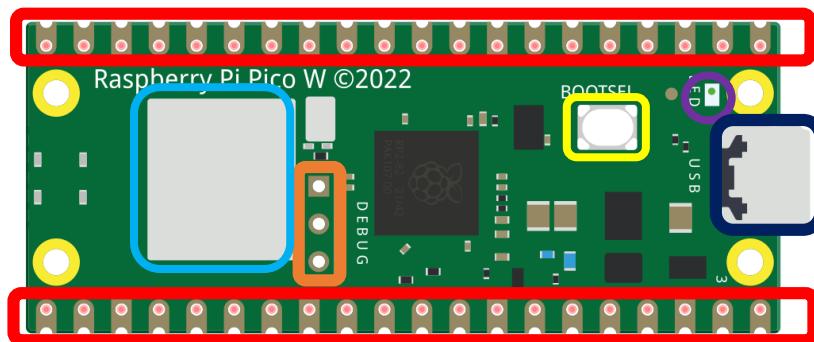
Raspberry Pi Pico W

Raspberry Pi Pico W applies to all chapters in this tutorial.

Raspberry Pi Pico W adds CYW43439 as the WiFi function based on Raspberry Pi Pico. It is connected to RP2040 chip through SPI interface.

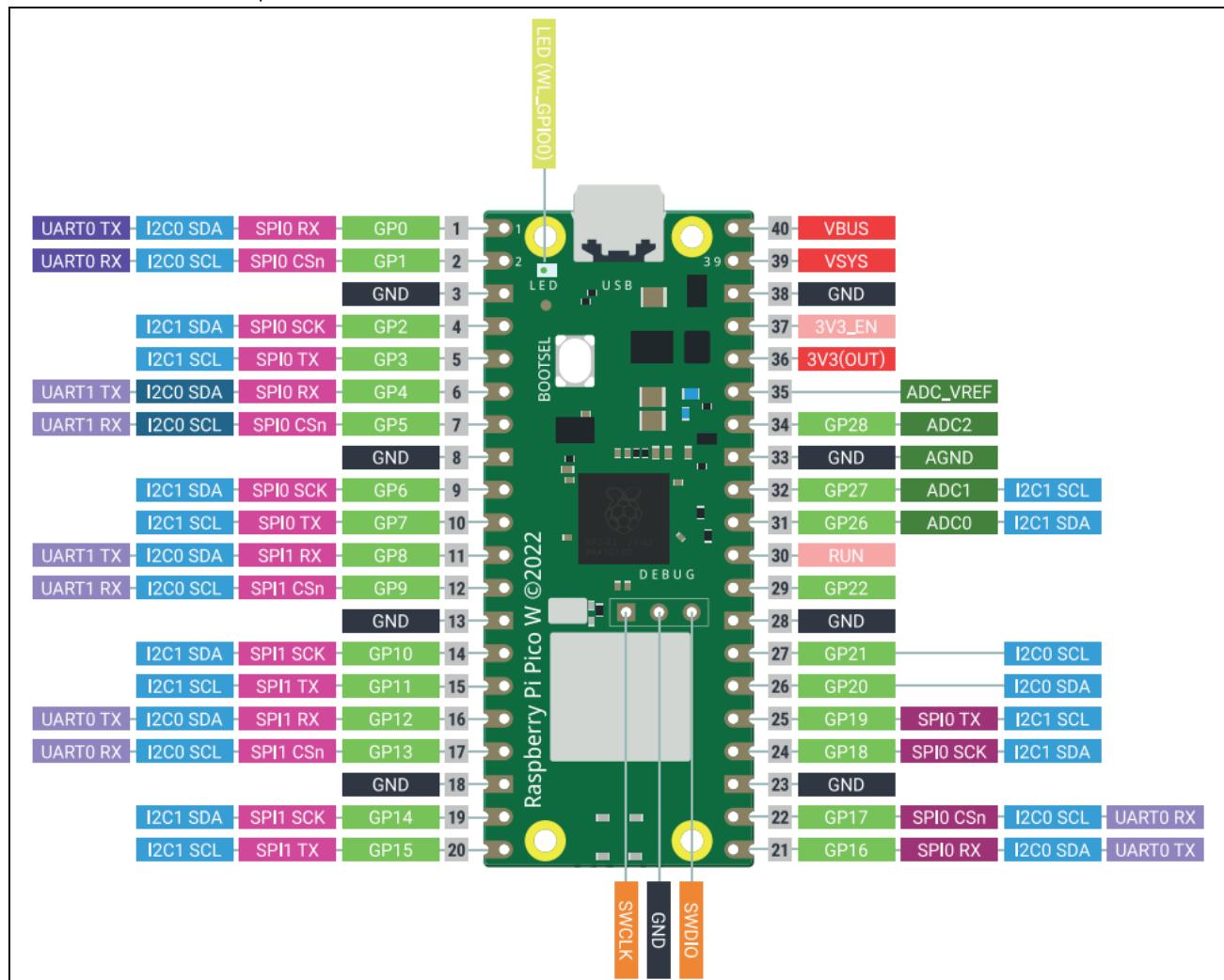


The hardware interfaces are distributed as follows:



Frame color	Description
Red	Pins
Yellow	BOOTSEL button
Blue	USB port
Purple	LED
Orange	Debugging
Light Blue	Wireless

Function definition of pins:



Color	Pins	Color	Pins
Black	GND	Red	Power
Green	GPIO	Dark Green	ADC
Purple	UART(default)	Lavender	UART
Magenta	SPI	Cyan	I2C
Light Red	System Control	Orange	Debugging

For details: <https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf>



UART, I2C, SPI, Wireless Default Pin

The default pins of serial port are Pin0 and Pin1.

Note: Serial port is virtualized by RP2040. Therefore, when using the serial port, please enable the verification function of DTR. It can work under any baud rate.

UART

Function	Default
UART_BAUDRATE	X
UART_BITS	8
UART_STOP	1
UART_TX	Pin 0
UART_RX	Pin 1

I2C

Function	Default
I2C Frequency	400000
I2C_SDA	Pin 4
I2C_SCL	Pin 5

SPI

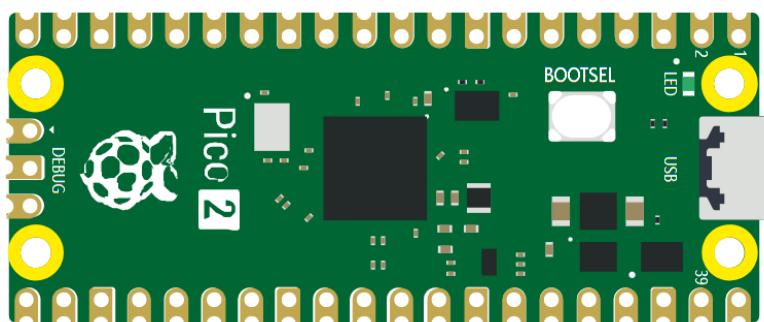
Function	Default
SPI_BAUDRATE	1000000
SPI_POLARITY	0
SPI_PHASE	0
SPI_BITS	8
SPI_FIRSTBIT	MSB
SPI_SCK	Pin 18
SPI_MOSI	Pin 19
SPI_MISO	Pin 16
SPI_SS	Pin 17

Wireless

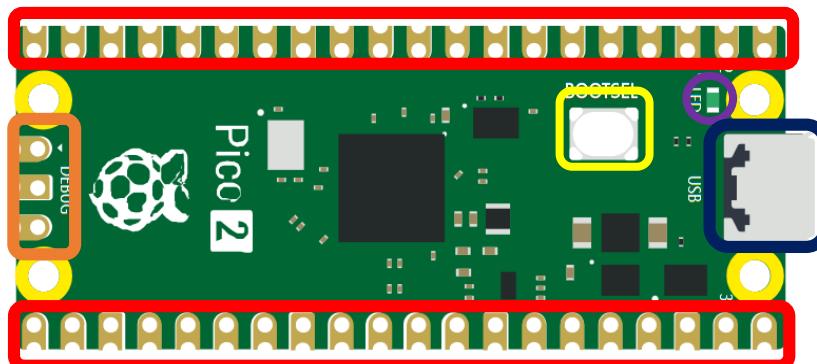
Function	Default
WL_ON	GPIO23
WL_D	GPIO24
WL_CLK	GPIO29_ADC
WL_CS	GPIO25

Raspberry Pi Pico 2

Raspberry Pi Pico 2 is applicable to all chapters in this tutorial except RFID and those involving WiFi. Raspberry Pi Pico 2 uses RP2350 chip as the main controller, which equipped with dual Cortex-M33 or Hazard3 processors, capable of running up to 150 MHz, providing a significant boost in processing power, compared with the original Pico. It also doubles the memory with 520KB of SRAM and 4MB of onboard flash memory, with the ADC sampling frequency increasing to up to 500ksps. In addition, it adds 8 more PWM channels, and features additional interfaces like 2× Timer with 4 alarms, 1× AON Timer and 4 x PIO.

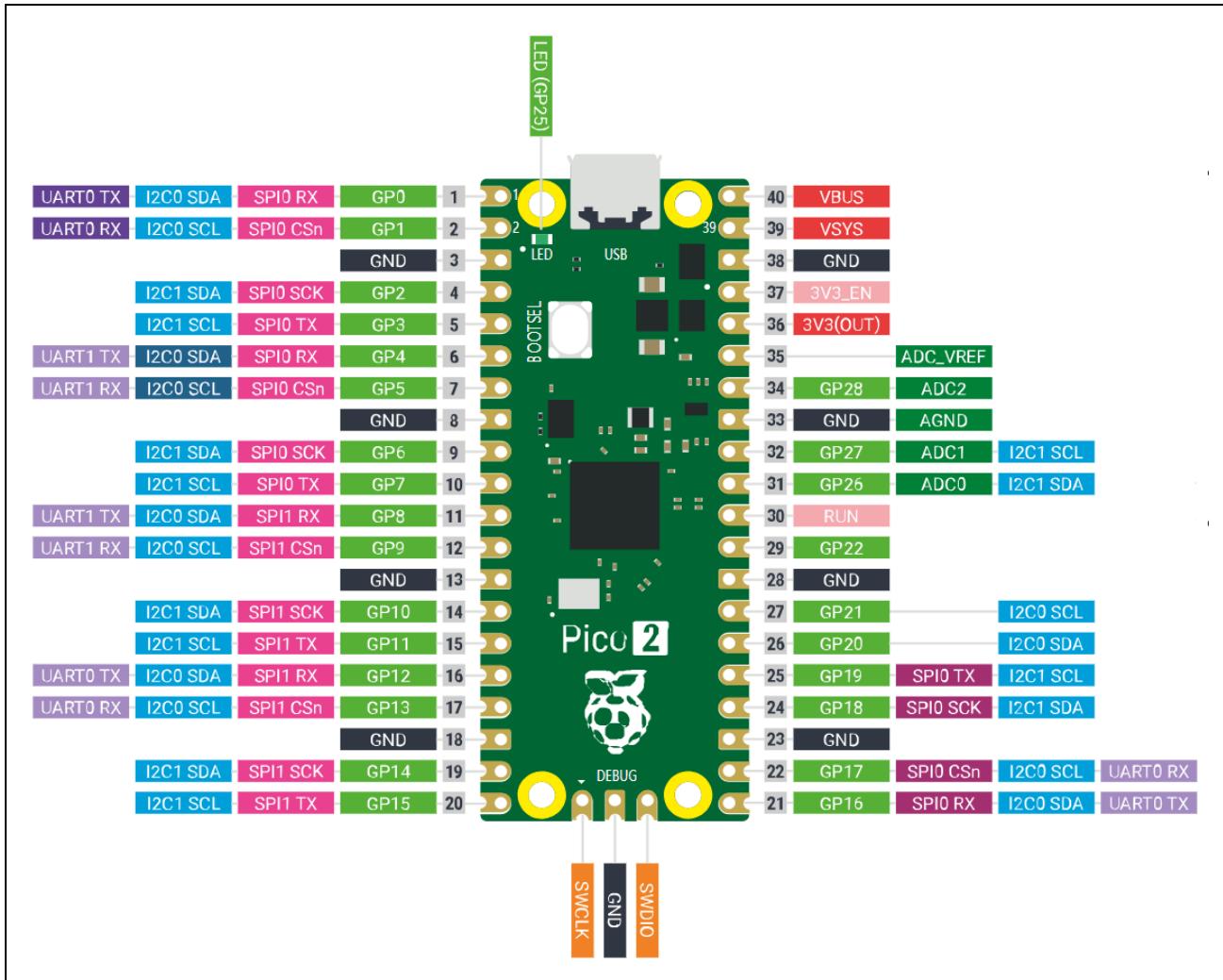


The hardware interfaces are distributed as follows:



Frame color	Description
	Pins
	BOOTSEL button
	USB port
	LED
	Debugging

Function definition of pins:



Color	Pins	Color	Pins
	GND		Power
	GPIO		ADC
	UART(default)		UART
	SPI		I2C
	System Control		Debugging

For details: <https://datasheets.raspberrypi.com/pico/pico-2-datasheet.pdf>

Any concerns? ✉ support@freenove.com

UART, I2C, SPI Default Pin

The default pins of serial port are Pin0 and Pin1.

Note: Serial port is virtualized by RP2350. Therefore, when using the serial port, please enable the verification function of DTR. It can work under any baud rate.

UART

Function	Default
UART_BAUDRATE	X
UART_BITS	8
UART_STOP	1
UART_TX	Pin 0
UART_RX	Pin 1

I2C

Function	Default
I2C Frequency	400000
I2C_SDA	Pin 4
I2C_SCL	Pin 5

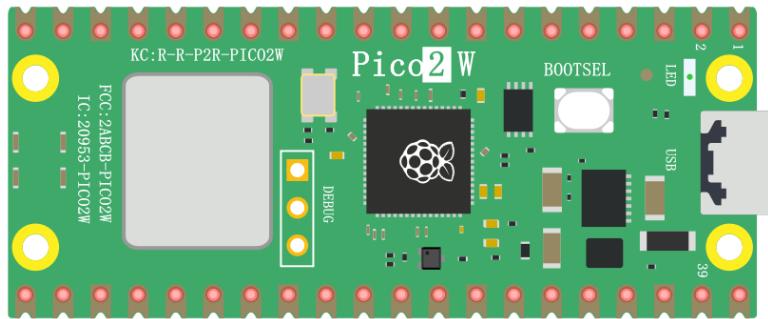
SPI

Function	Default
SPI_BAUDRATE	1000000
SPI_POLARITY	0
SPI_PHASE	0
SPI_BITS	8
SPI_FIRSTBIT	MSB
SPI_SCK	Pin 18
SPI_MOSI	Pin 19
SPI_MISO	Pin 16
SPI_SS	Pin 17

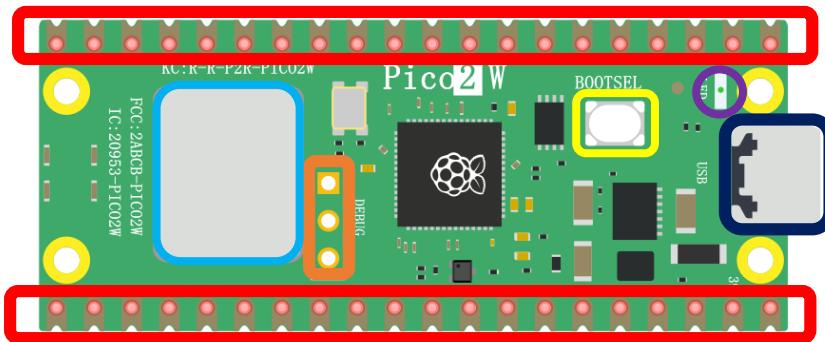
Raspberry Pi Pico 2W

Raspberry Pi Pico 2W applies to all chapters in this tutorial.

The Raspberry Pi Pico 2W adds WiFi functionality based on the Raspberry Pi Pico 2 by incorporating the CYW43439 module, which connects to the RP2350 chip via an SPI interface.

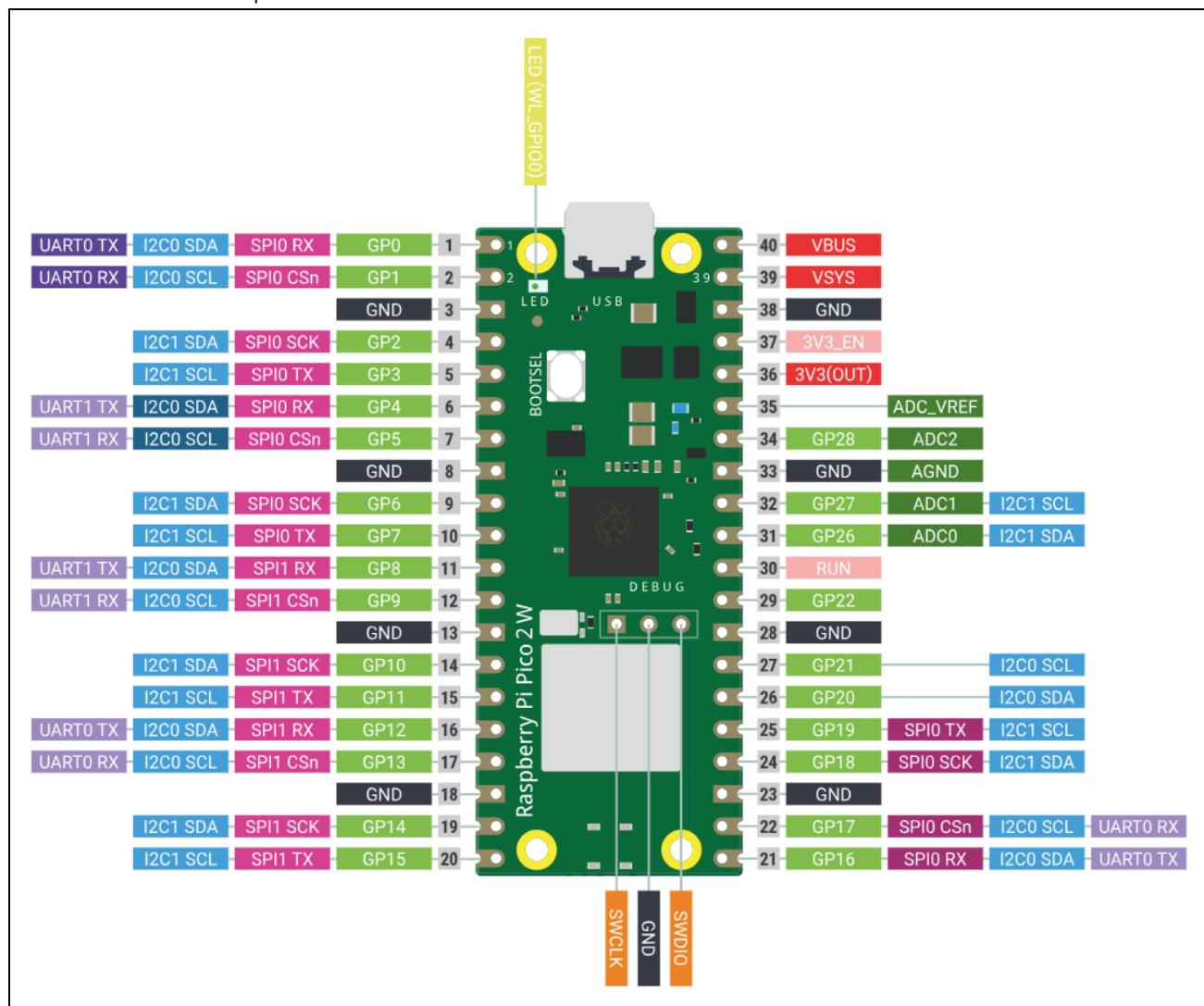


The hardware interfaces are distributed as follows:



Frame color	Description
Red	Pins
Yellow	BOOTSEL button
Blue	USB port
Purple	LED
Orange	Debugging
Cyan	Wireless

Function definition of pins:



Color	Pins	Color	Pins
Black	GND	Red	Power
Green	GPIO	Dark Green	ADC
Purple	UART(default)	Lavender	UART
Magenta	SPI	Cyan	I2C
Pink	System Control	Orange	Debugging

For details: <https://datasheets.raspberrypi.com/picow/pico-2-w-datasheet.pdf>



UART, I2C, SPI, Wireless Default Pin

The default pins of serial port are Pin0 and Pin1.

Note: Serial port is virtualized by RP2350. Therefore, when using the serial port, please enable the verification function of DTR. It can work under any baud rate.

UART

Function	Default
UART_BAUDRATE	X
UART_BITS	8
UART_STOP	1
UART_TX	Pin 0
UART_RX	Pin 1

I2C

Function	Default
I2C Frequency	400000
I2C_SDA	Pin 4
I2C_SCL	Pin 5

SPI

Function	Default
SPI_BAUDRATE	1000000
SPI_POLARITY	0
SPI_PHASE	0
SPI_BITS	8
SPI_FIRSTBIT	MSB
SPI_SCK	Pin 18
SPI_MOSI	Pin 19
SPI_MISO	Pin 16
SPI_SS	Pin 17

Wireless

Function	Default
WL_ON	GPIO23
WL_D	GPIO24
WL_CLK	GPIO29_ADC
WL_CS	GPIO25

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

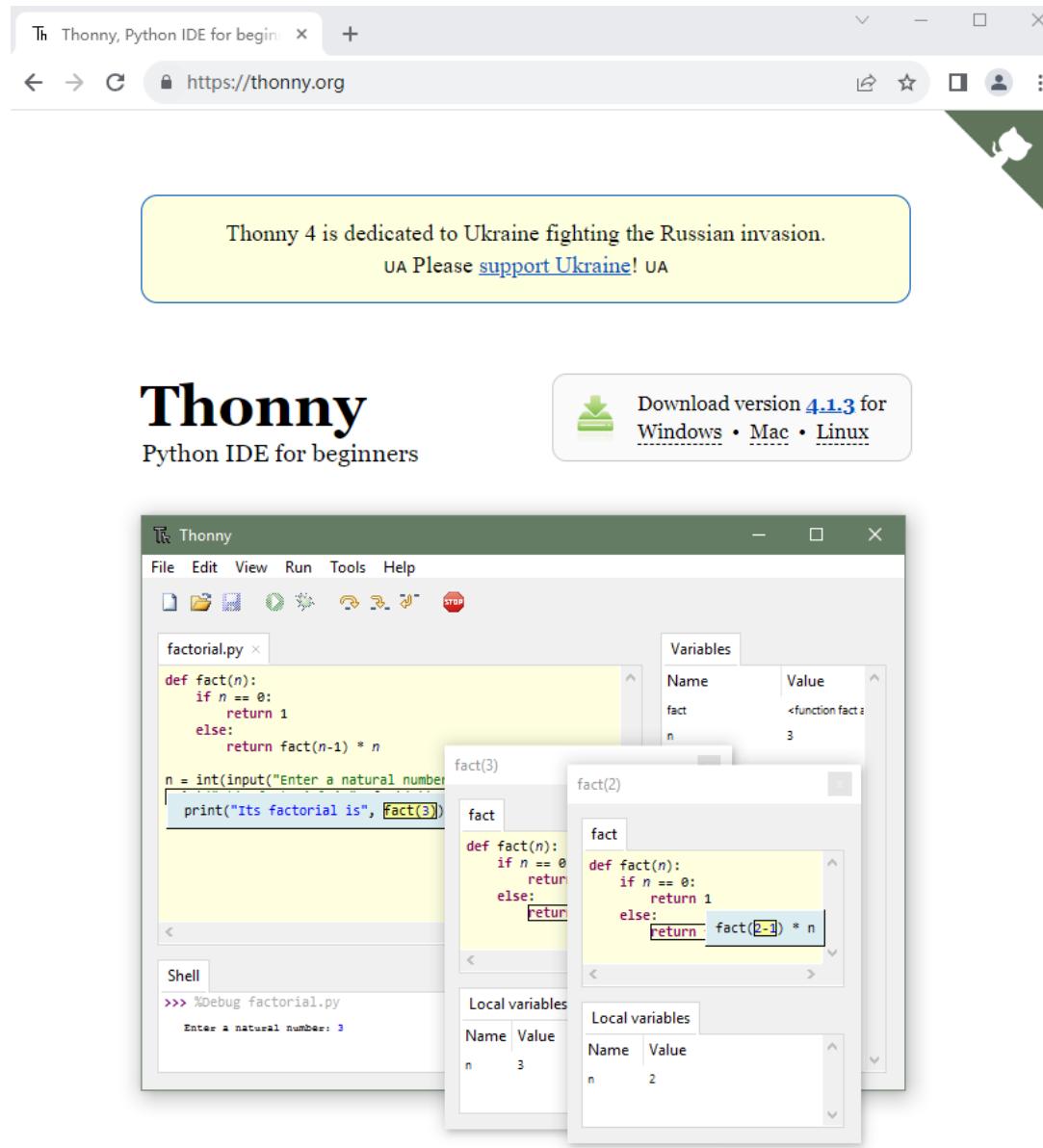
Official website of Thonny: <https://thonny.org>

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.)

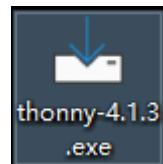
Operating System	Download links/methods
Windows	https://github.com/thonny/thonny/releases/download/v4.1.3/thonny-4.1.3.exe
Mac OS	https://github.com/thonny/thonny/releases/download/v4.1.3/thonny-4.1.3.pkg
Linux	The latest version: Binary bundle for PC (Thonny+Python): bash <(wget -O - https://thonny.org/installer-for-linux) With pip: pip3 install thonny Distro packages (may not be the latest version): Debian, Raspbian, Ubuntu, Mint and others: sudo apt install thonny Fedoras: sudo dnf install thonny

You can also open “[Freenove_Basic_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-4.1.3.exe".



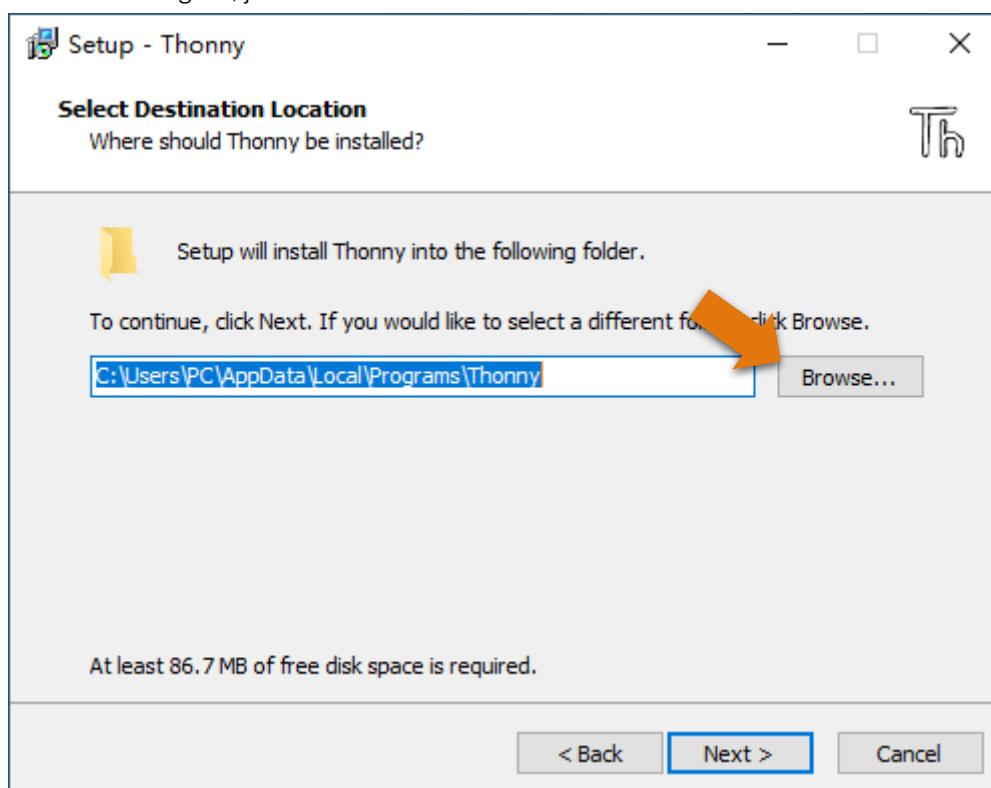
Any concerns? ✉ support@freenove.com

If you are not familiar with computer software installation, you can simply keep clicking “Next” until the installation completes.



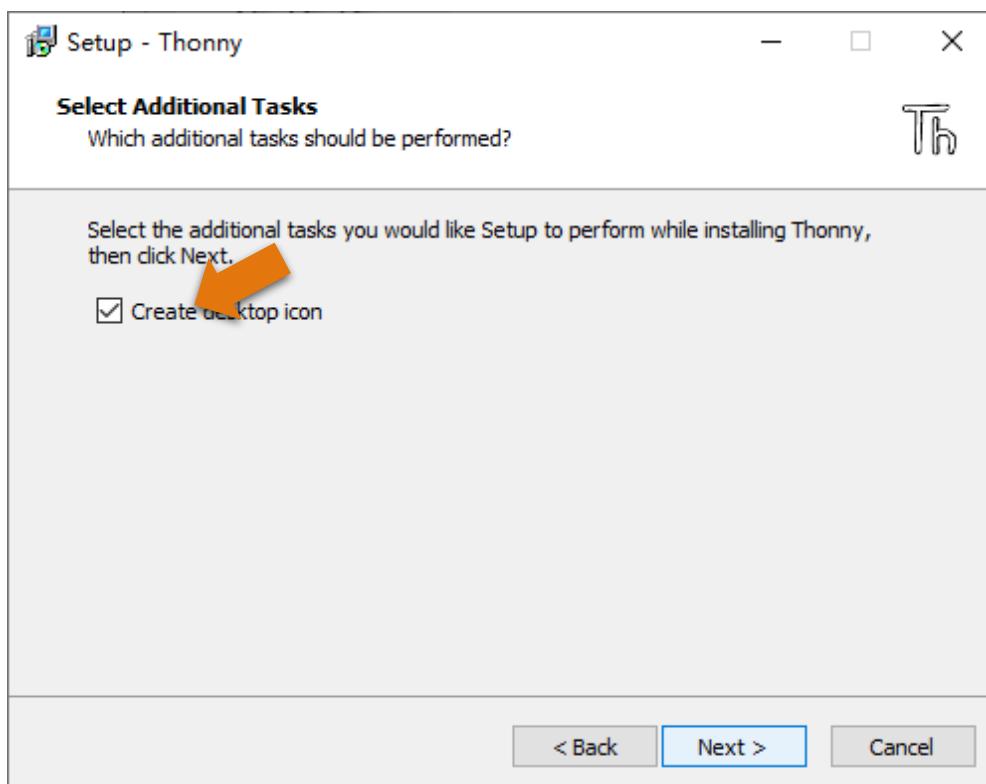
If you want to change Thonny's installation path, you can click “Browse” to modify it. After selecting installation path, click “OK”.

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

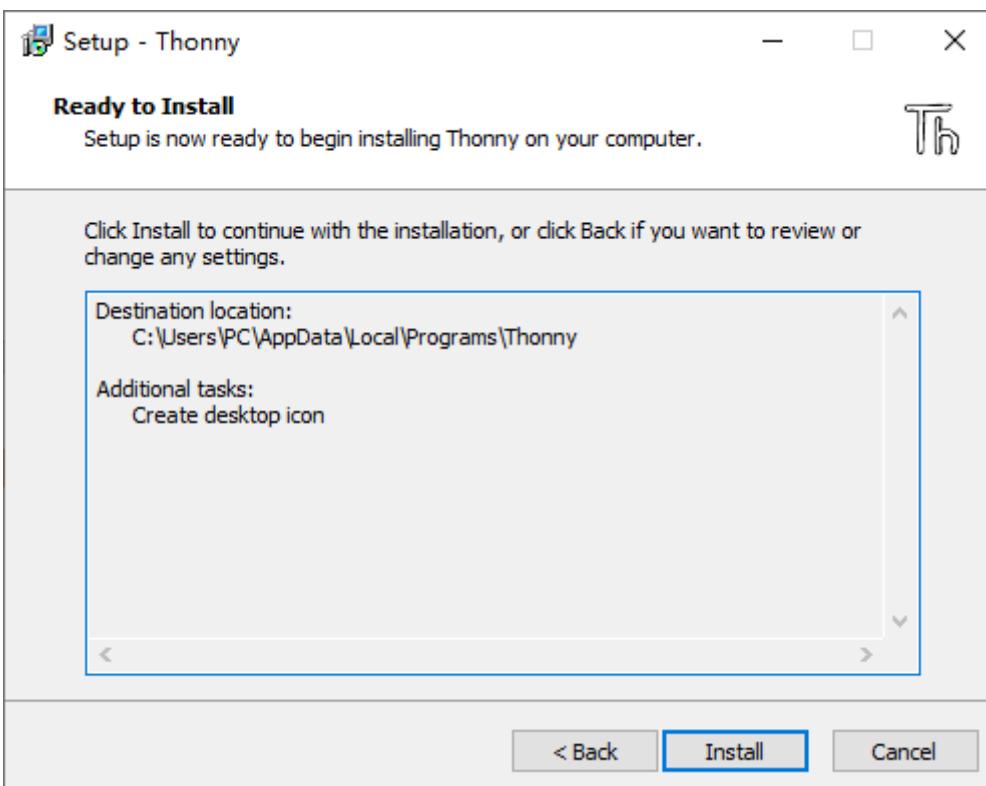




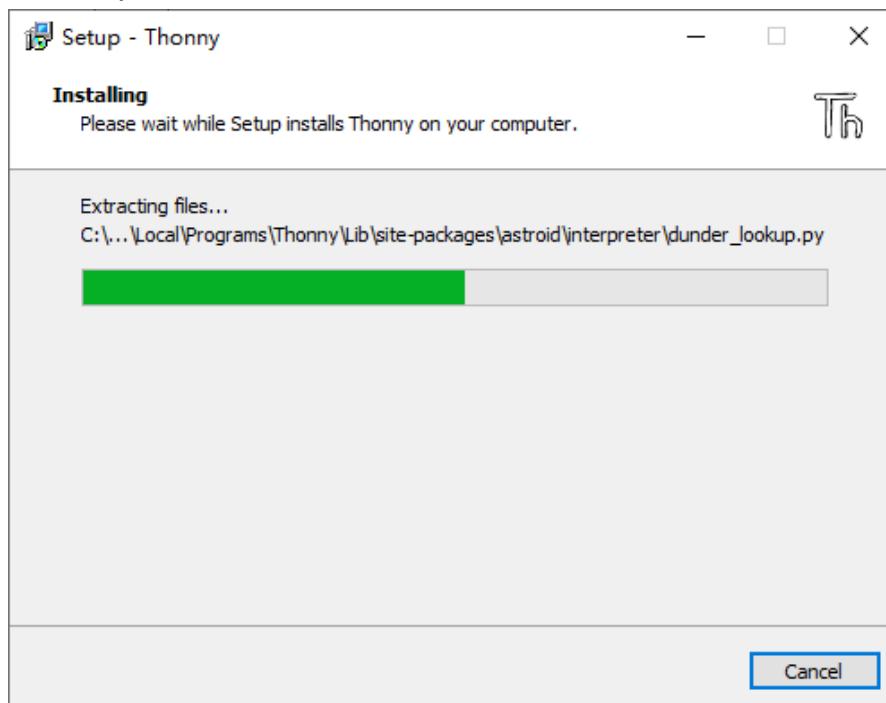
Check “Create desktop icon” and then it will generate a shortcut on your desktop to facilitate you to open Thonny later.



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.



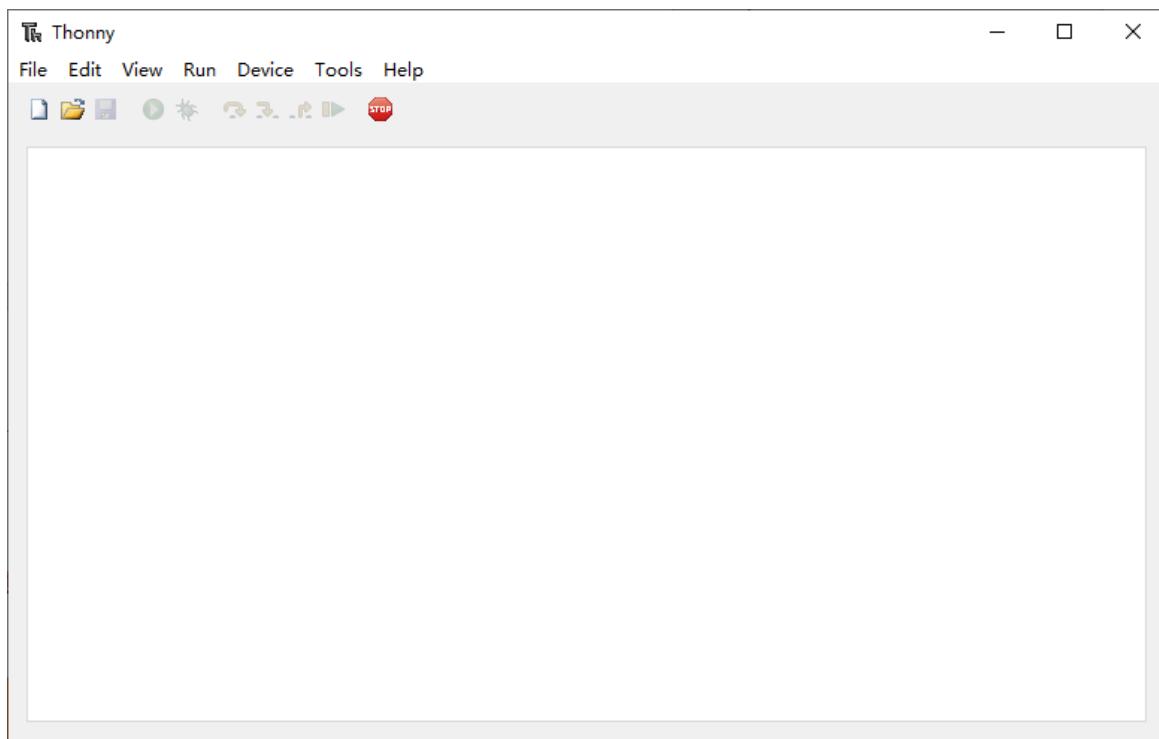
If you've checked "Create desktop icon" during the installation process, you can see the below icon on your desktop.



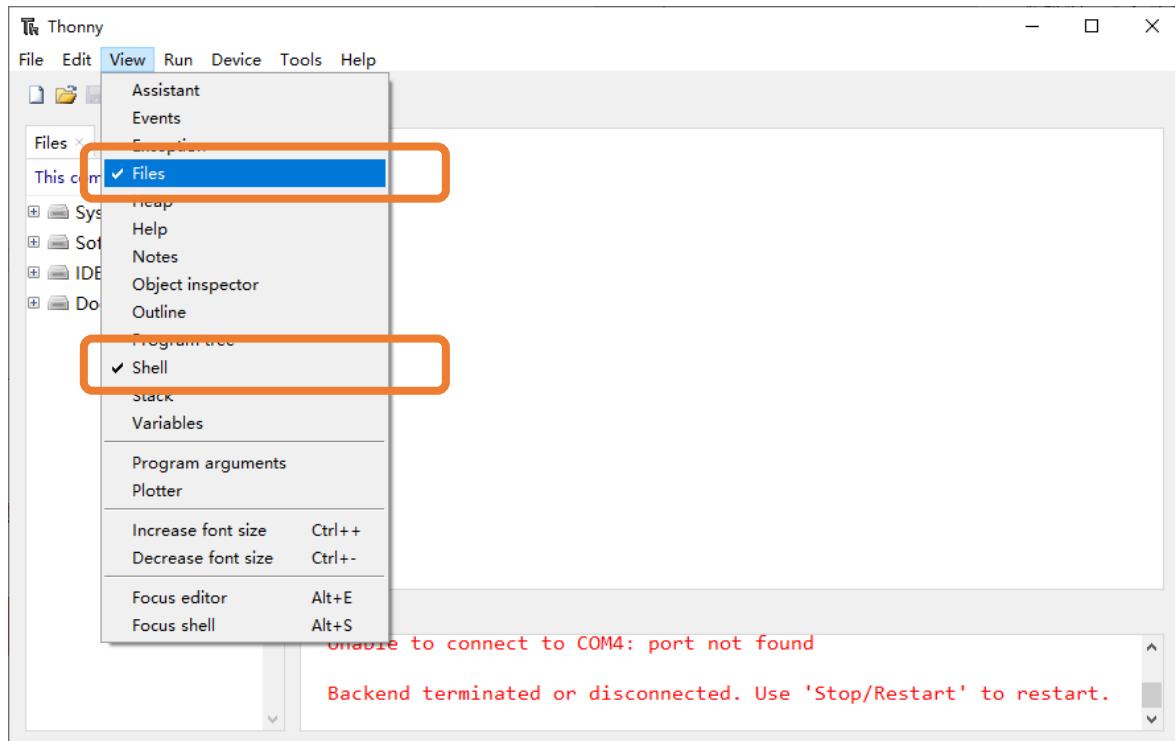


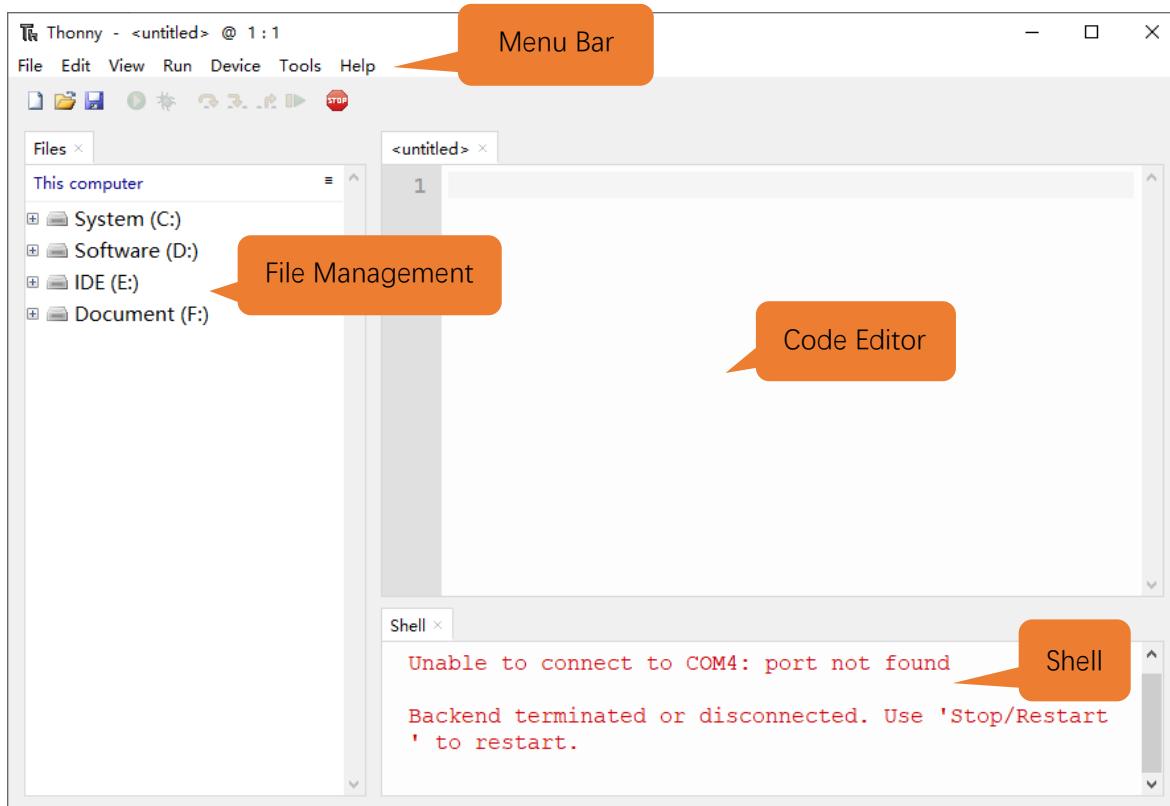
0.2 Basic Configuration of Thonny

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.com/documentation/microcontrollers/>

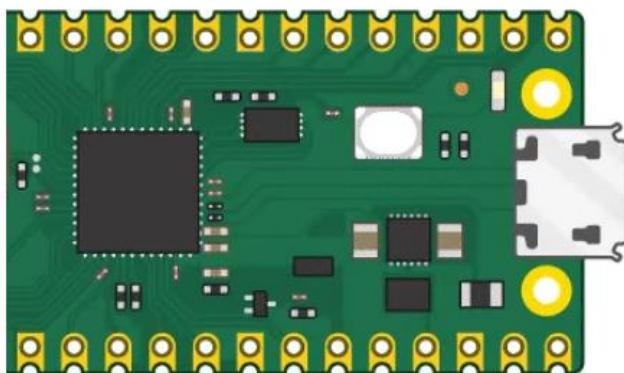
1. Click Micropython.

The screenshot shows a section of the Raspberry Pi Microcontrollers page. It features a grid of six items under the 'Microcontrollers' tab:

- RP2040**: Our first microcontroller device. It includes a small icon of a chip and a brief description.
- Raspberry Pi Pico and Pico W**: Support for Raspberry Pi Pico, Pico H, and Pico W. It includes a small icon of a small board and a brief description.
- MicroPython**: Getting started with MicroPython. This item is highlighted with a red rounded rectangle.
- The C/C++ SDK**: Getting started with the C/C++ SDK. It includes a small icon of a hexagon with a 'C' and a brief description.
- PIP**: The Product Information Portal (PIP) for Raspberry Pi compliance documents. It includes a small icon of three documents and a brief description.
- Datasheets**: The Datasheets site for PDF-based documentation. It includes a small icon of two books and a brief description.

2. In the new interface, find the following content and download the corresponding UF2 according to your Pico version.

2



Download the correct MicroPython UF2 file for your board:

- Pico
- Pico W
- Pico 2
- Pico 2 W

For more information about using Wi-Fi and Bluetooth on Raspberry Pi Pico W-series devices with C/C++ or MicroPython, see [Connecting to the Internet with Raspberry Pi Pico W-series](#).

For more information about [supported Bluetooth protocols and profiles](#), see the Blue Kitchen [BTStack](#) Github repository.

If your board is a standard Pico board, please click [Raspberry Pi Pico](#) to download it.

If it is a Pico W board, click [Raspberry Pi Pico W](#).

If it is a Pico 2 board, click [Raspberry Pi Pico 2](#).

If it is a Pico 2 W board, click [Raspberry Pi Pico 2 W](#).

If it is Pico 2, click the MicroPython download page, or click the following link:

https://micropython.org/download/RPI_PICO2/

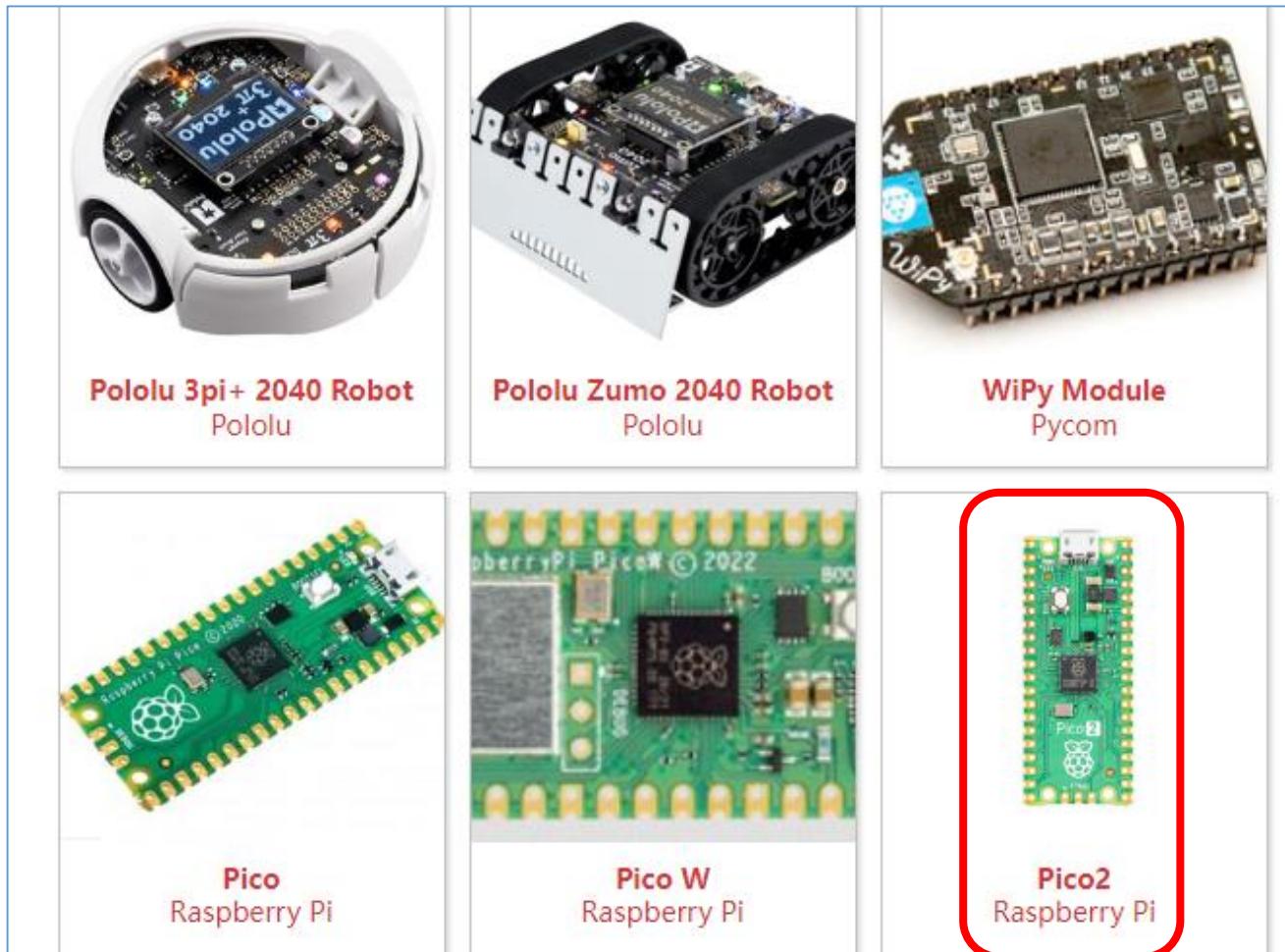
Download the correct MicroPython UF2 file for your board:

- [Raspberry Pi Pico](#)
- [Raspberry Pi Pico W with Wi-Fi and Bluetooth LE support](#)

To work with Wi-Fi and Bluetooth on Raspberry Pi Pico W with C/C++ or MicroPython, see the [Connecting to the Internet with Raspberry Pi Pico W](#) book. For details about [supported Bluetooth protocols and profiles](#), see the Blue Kitchen [BTStack](#) Github repository.

NOTE

MicroPython distributions for other RP2040-based boards are available on the [MicroPython download page](#).



Firmware

Preview builds

v1.24.0-preview.201.g269a0e0e1 (2024-08-09).uf2

(These are automatic builds of the development branch for the next release)

Firmware (RISC-V CPU mode)

No releases are available for this board.

Preview builds

v1.24.0-preview.201.g269a0e0e1 (2024-08-09).uf2

(These are automatic builds of the development branch for the next release)

As Raspberry Pi Foundation has specially added two Cortex-M33 and two RISC-V cores in RP2350, both of the above two uf2 files can work.

Option 2

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:

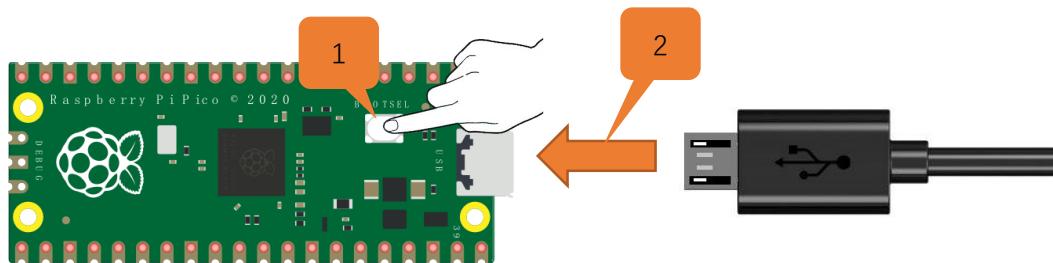
[Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico/Python_Firmware](#)

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

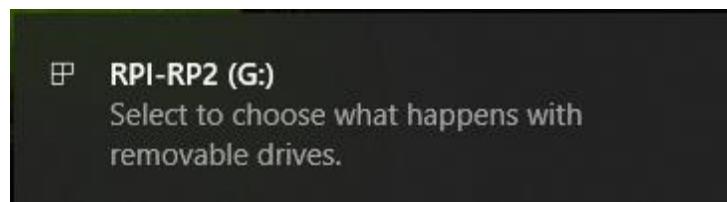
Burning a Micropython Firmware

Note: Pico, Pico W, Pico 2 and Pico 2W burn firmware in the same way. Pico's map is used here as an introduction.

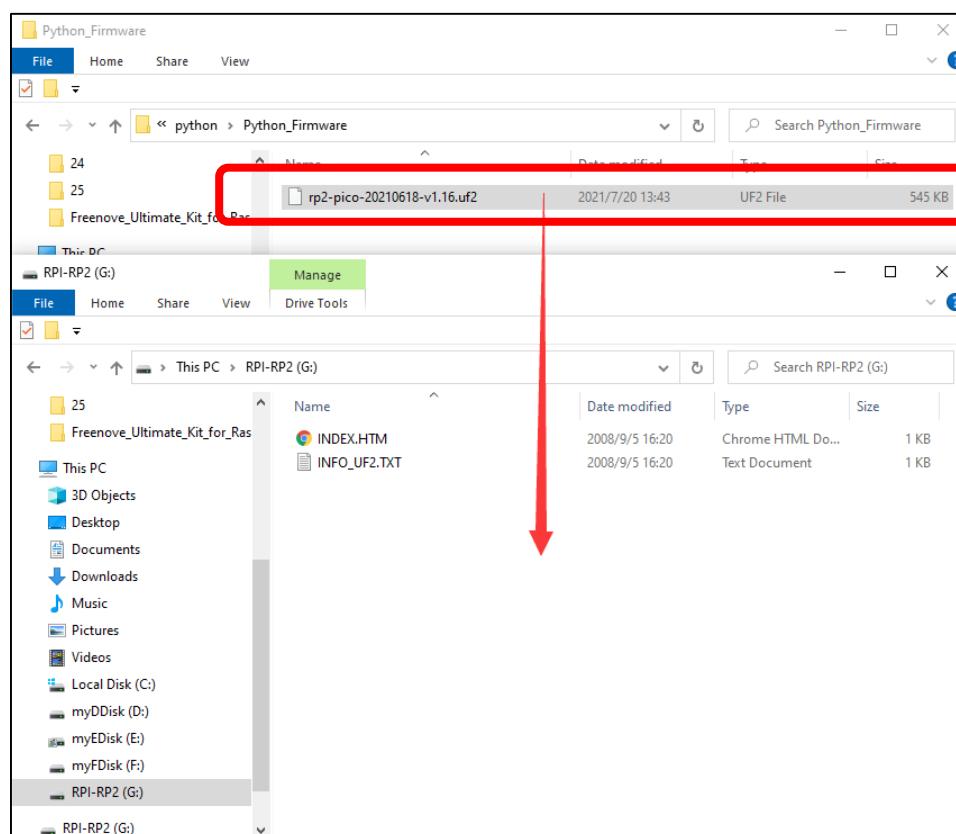
1. Connect a USB cable to your computer.
2. Long press BOOTSEL button on Raspberry Pi Pico and connect it to your computer with the USB cable.



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



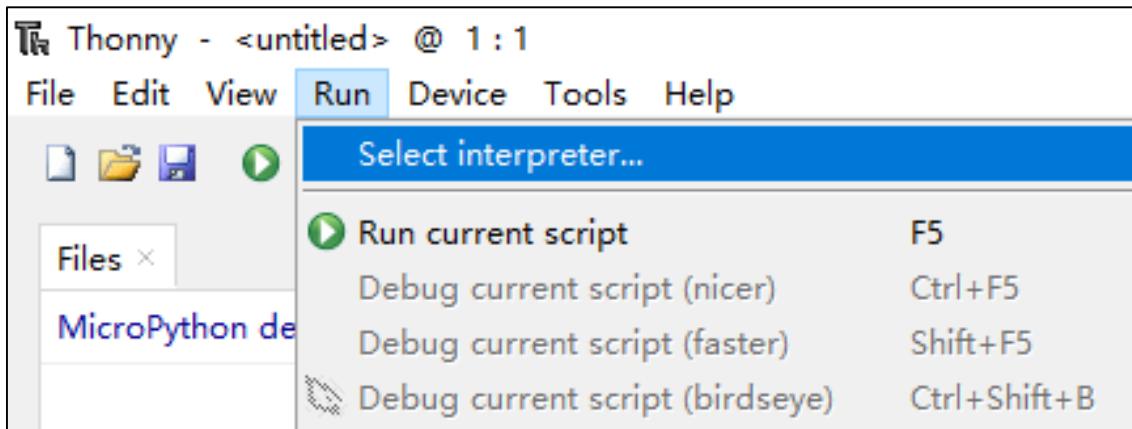
4. Copy the file(**rp2-pico-20210618-v1.16.uf2**) to RPI-RP2 and wait for it to finish, just like copy file to a U disk.



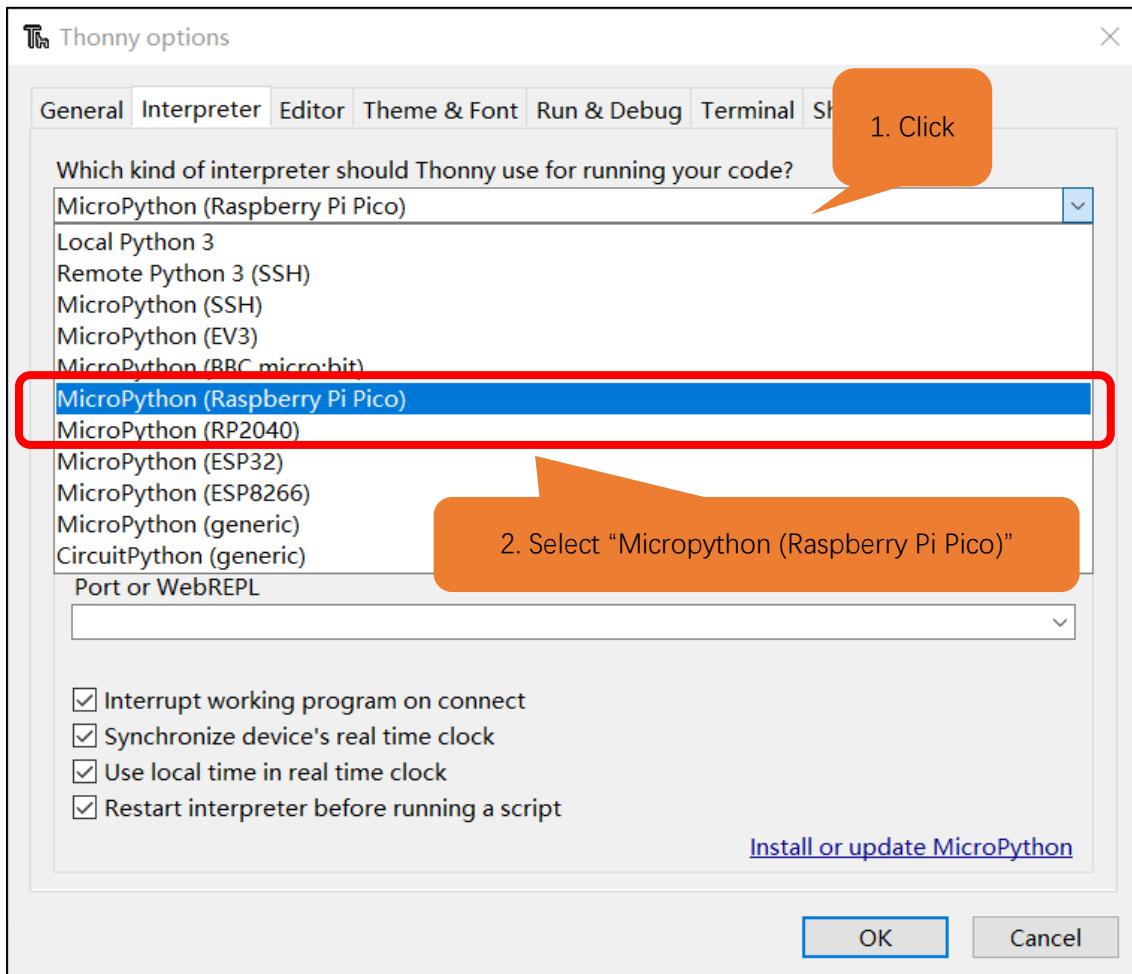
5. When the firmware finishes programming, Raspberry Pi Pico will reboot automatically.
After that, you can run Micropython.

0.4 Thonny Connected to Raspberry Pi Pico

1. Open Thonny, click “run” and select “Select interpreter...”



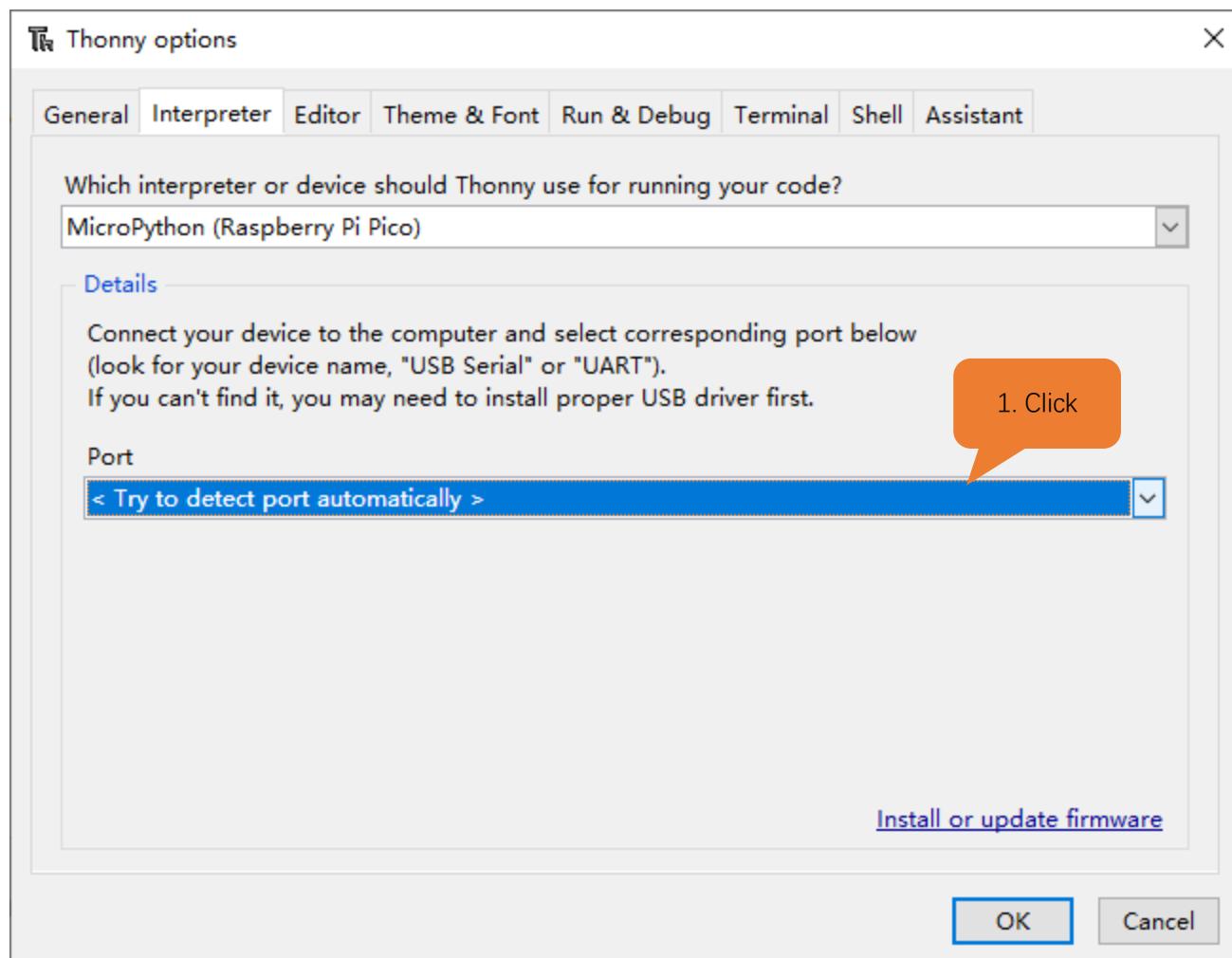
2. Select “Micropython (Raspberry Pi Pico)”. If you are using Raspberry Pi Pico (W), select “Micropython (RP2040)”;If Raspberry Pi Pico 2 (W) is used, select “Micropython (RP2350)”.



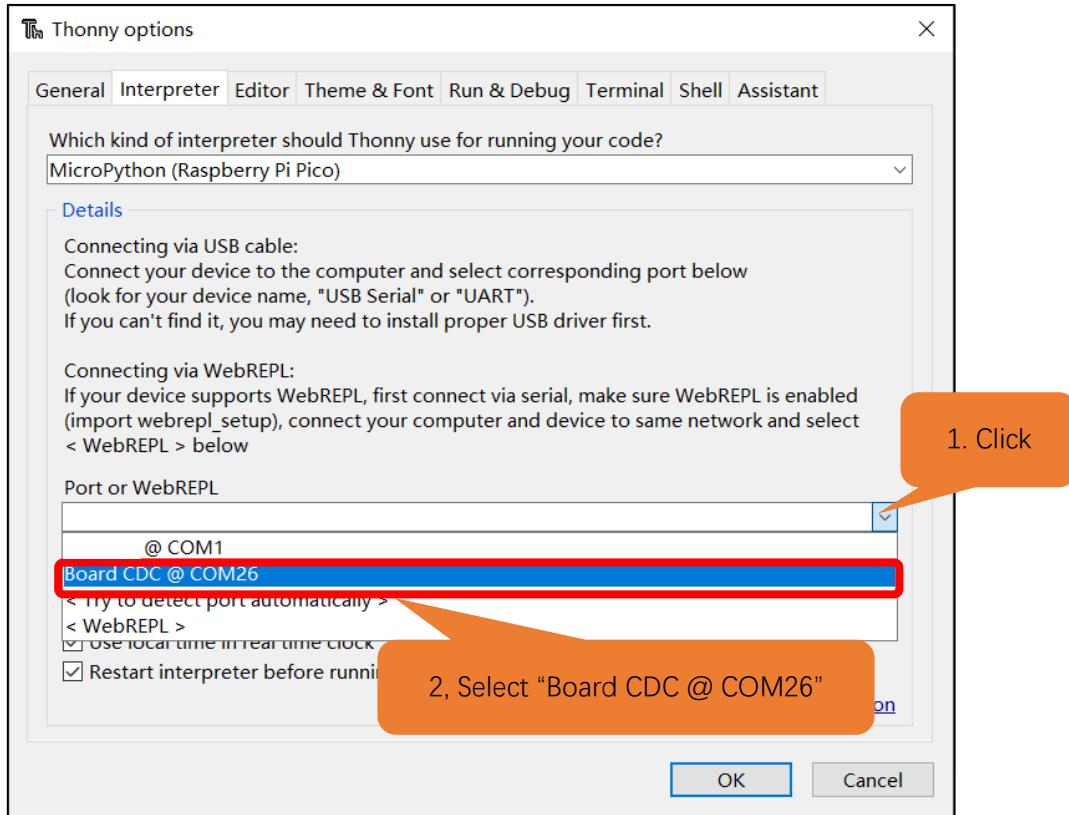
3. Select “Board CDC @ COMx”, The number x of COMx may vary among different computers. You only need to make sure selecting Board CDC @ COMx

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

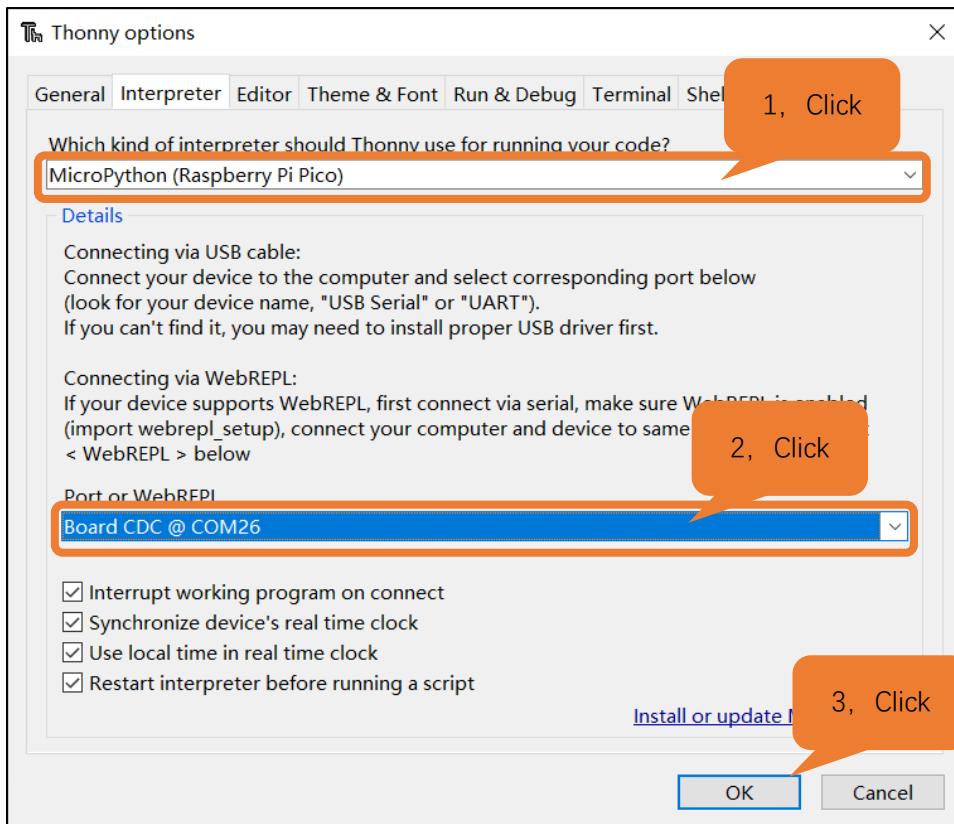
Step 1: When Pico **doesn't** connect to computer, open Thonny, click “Run”, select “Select interpreter” and then a dialog box will pop up, click “Port” and you can check the ports currently connected to your computer, as shown below:



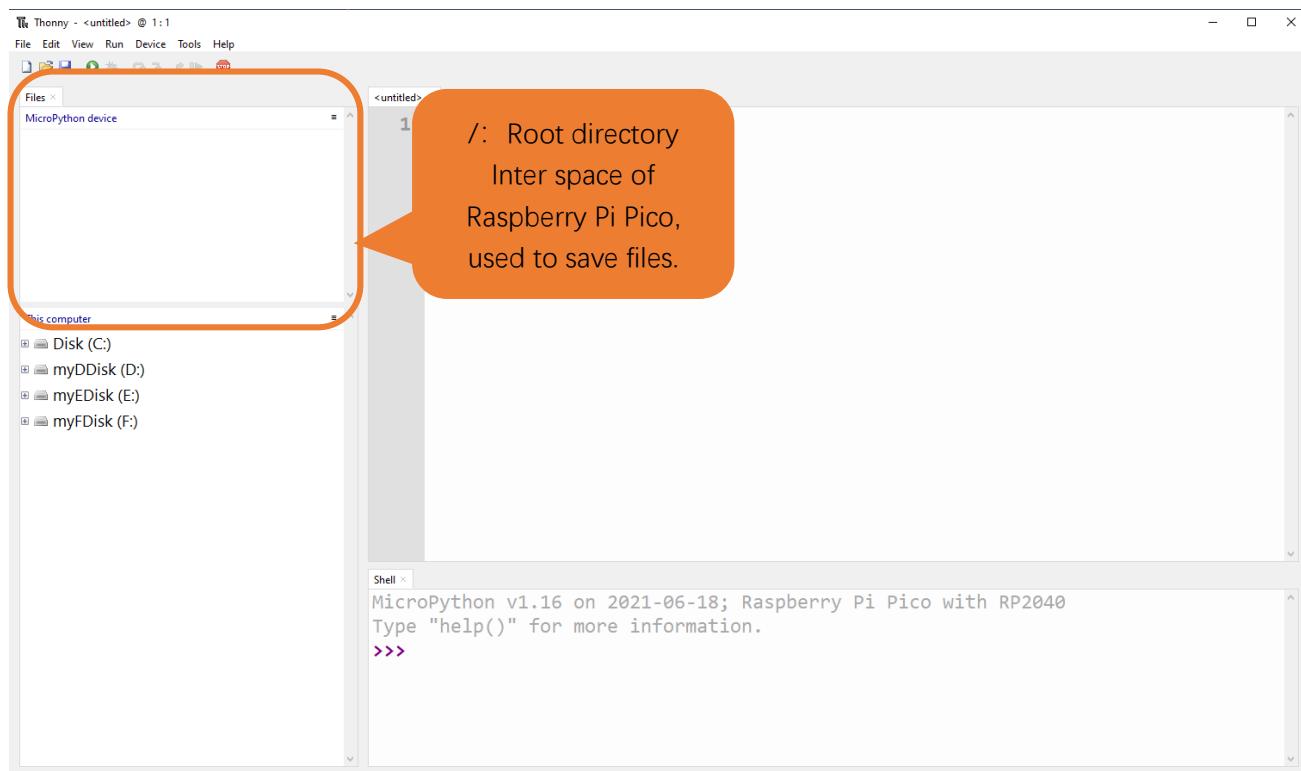
Step 2: Close the dialog box. Connect Pico to your computer, click “Run” again and select “Select interpreter”. Click “Port” on the pop-up window and check the current ports. Now there is a newly added port, with which Pico communicates with the computer



- After selecting “Micropython (Raspberry Pi Pico)” 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. The 'Files' pane on the left displays a file tree under 'MicroPython device' and 'This computer'. The '00.0_HelloWorld' file is selected. The 'Shell' pane on the right shows the Python interpreter's response to the command `>>> print("hello world!")`, which outputs `hello world!`. An orange callout bubble points to the output text with the text "Successfully display".

```
MicroPython v1.16 on 2021-06-18; RP2040 with RP2040
Type "help()" for more information
>>> print("hello world!")
hello world!
>>>
```

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:

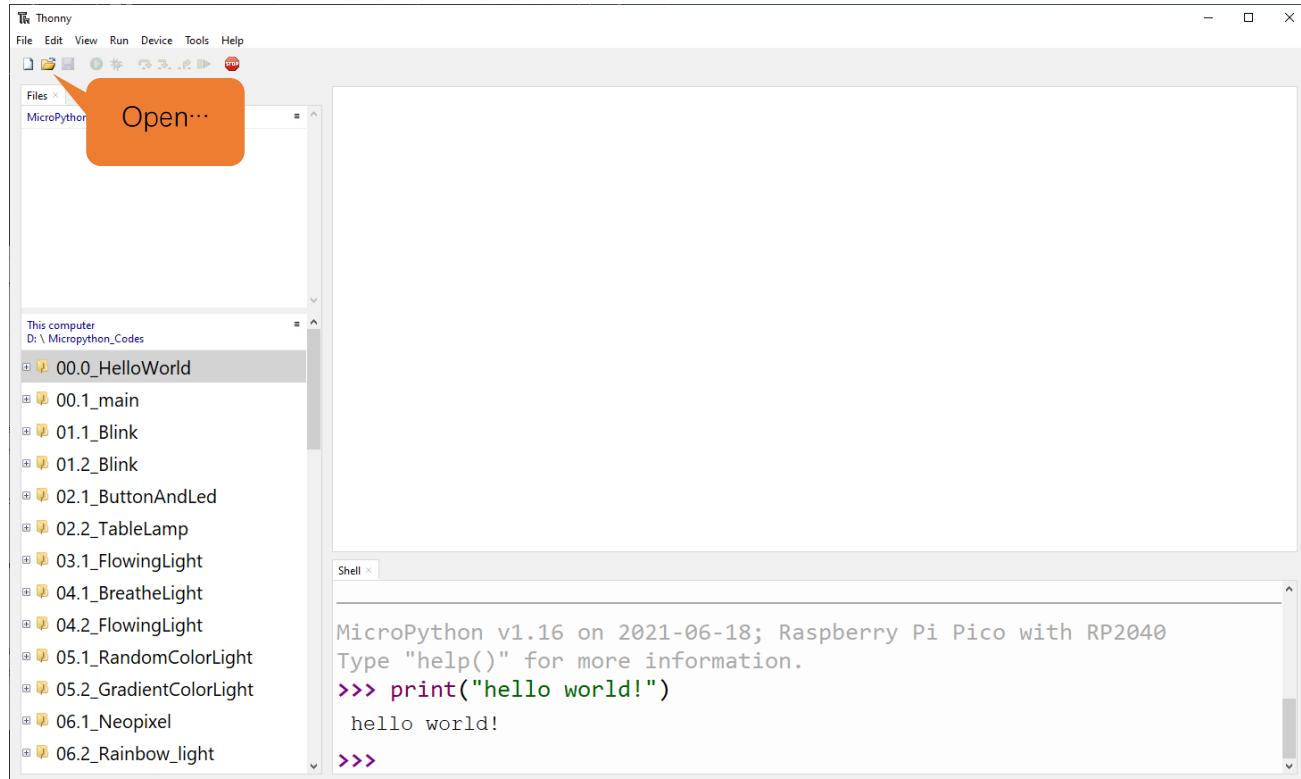
1. Users can use Thonny to compile or debug programs.
2. Through the "Shell" window, users can read the error information and output results generated during the running of the program and query related function information online to help improve the program.

Disadvantages:

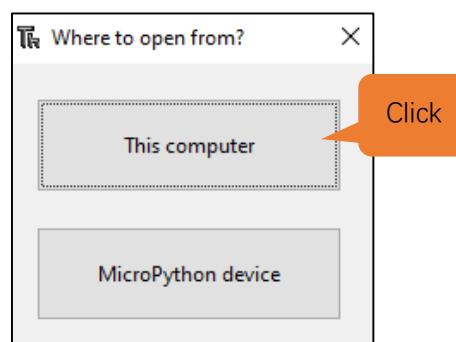
1. To run Raspberry Pi Pico online, you have to be connected to a computer and run with Thonny.
2. If Raspberry Pi Pico disconnects from computer, the program will not run again when they reconnect to each other.

Operation

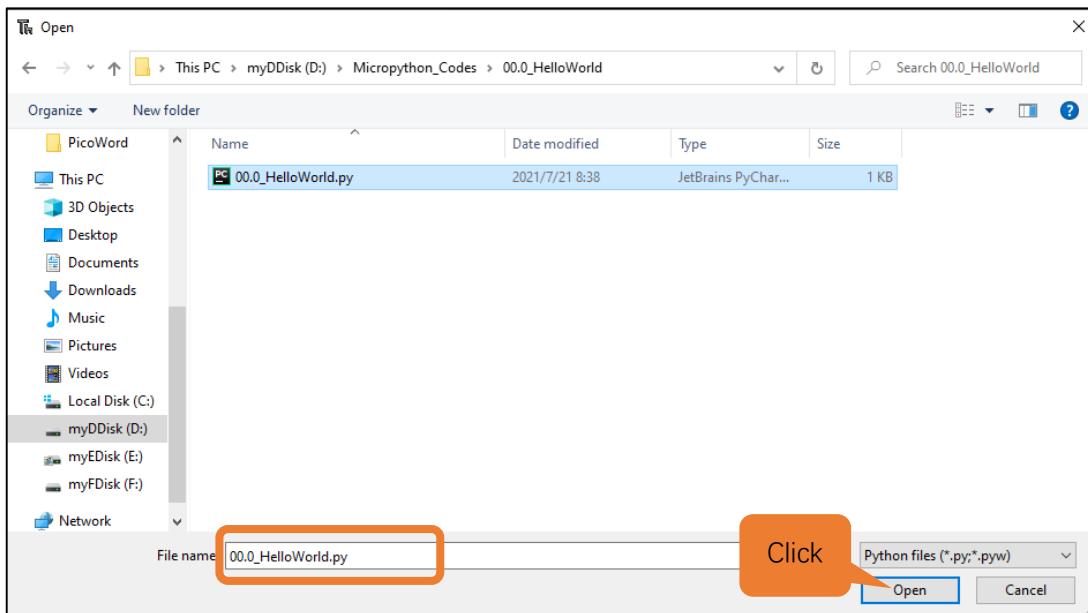
1. Open Thonny and click "Open…".



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



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

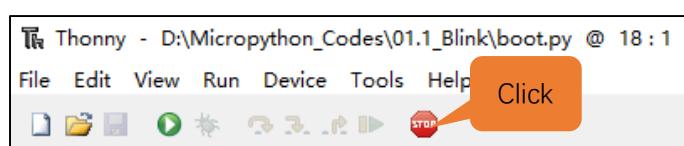


Click “Run current script” to execute the program and “Hello World!”, “Welcome Freenove” will be printed in “Shell”.



Exiting Running Online

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



Running Offline

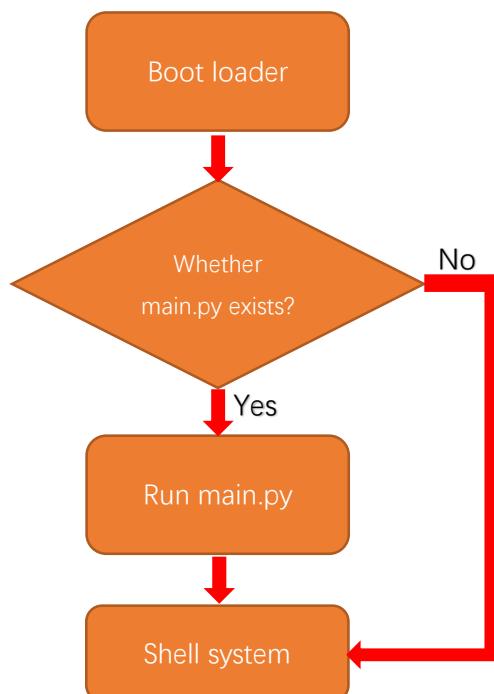
When running offline, Raspberry Pi Pico does not need to connect to computer and Thonny. It can run the programs stored in main.py on the device once powered up.

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

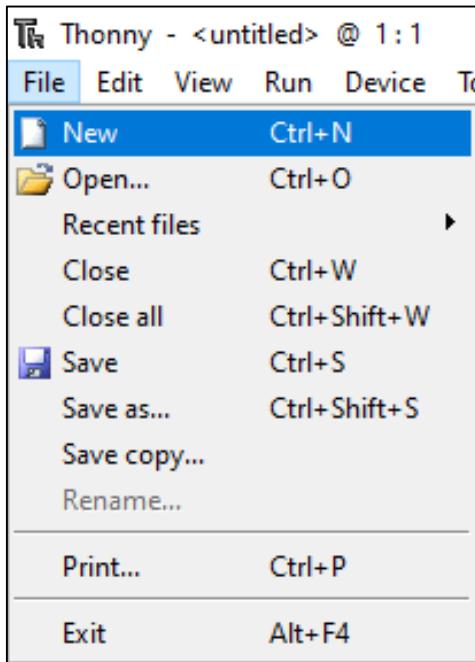
Disadvantage: The program will stop automatically when error occurs or Raspberry Pi Pico is out of power.
Code cannot be changed easily.

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 does not exist, it will enter shell command system directly.



- 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.

The screenshot shows the Thonny IDE interface with a MicroPython script named "01.1_Blink" open in the code editor. The script contains the following 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

```

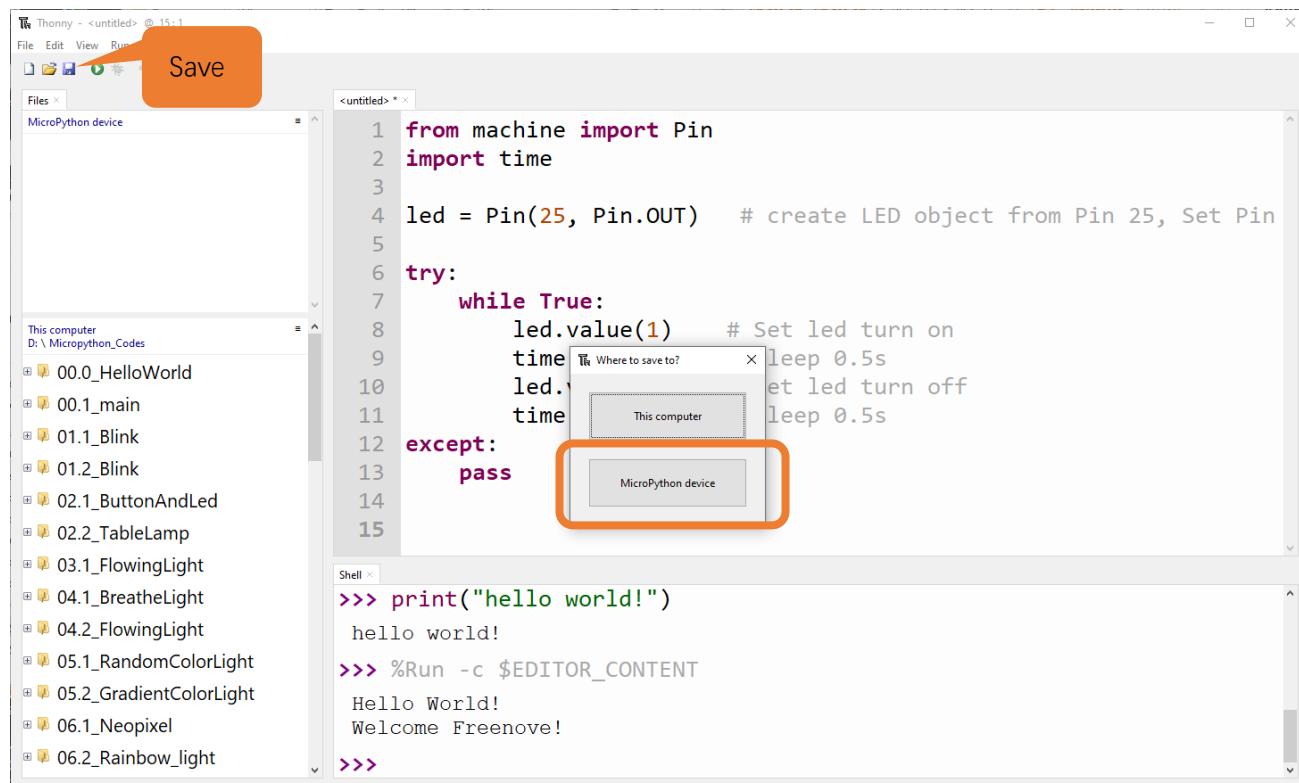
Below the code editor is a "Shell" window showing the output of running the script:

```

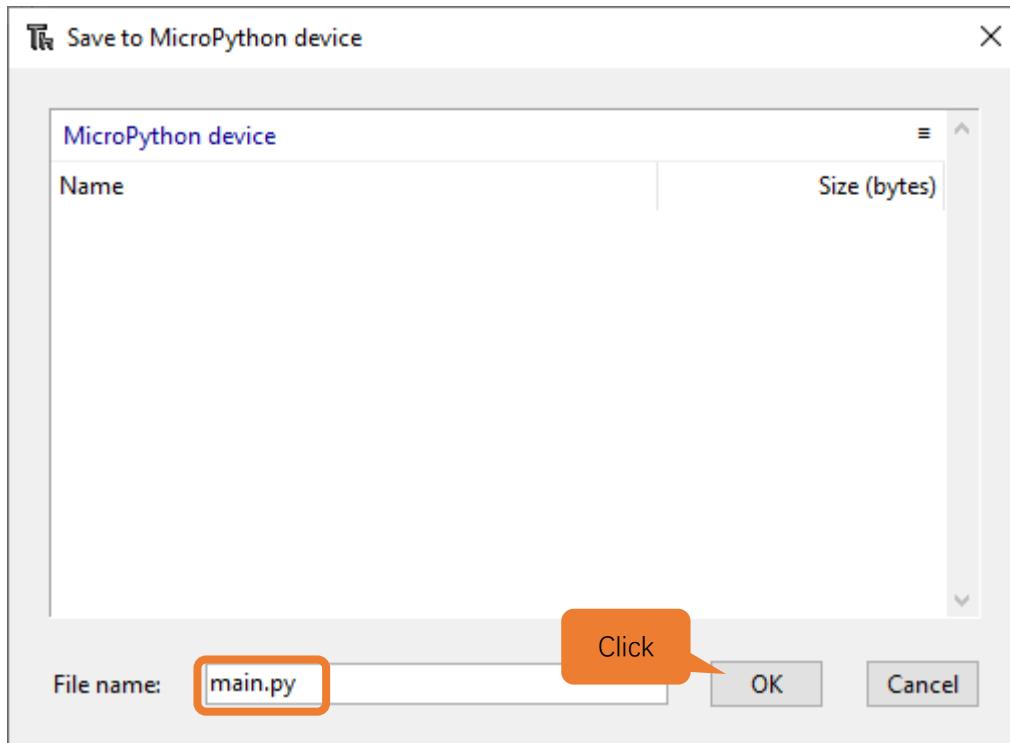
>>> print("hello world!")
hello world!
>>> %Run -c $EDITOR_CONTENT
Hello World!
Welcome Freenove!
>>>

```

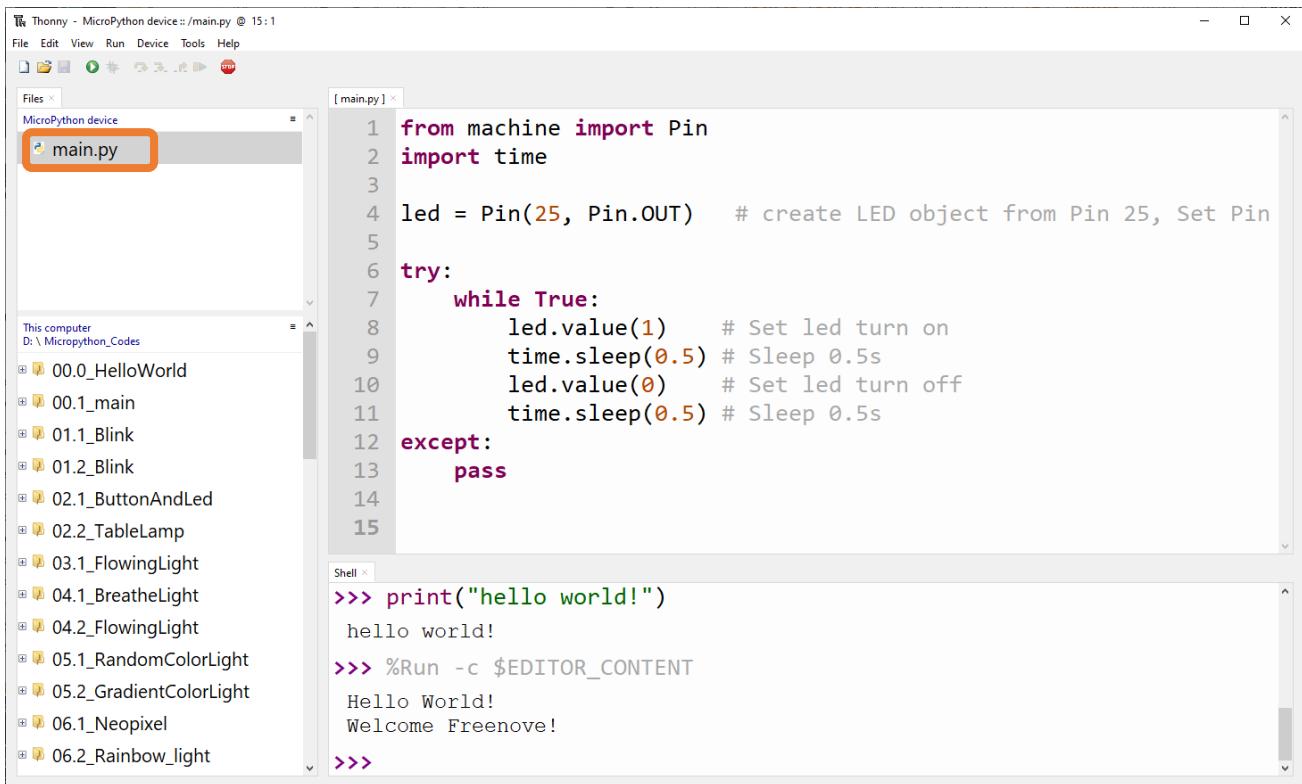
3. Click “Save” on the menu bar. You can save the codes either to your computer or to Raspberry Pi Pico.



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 displays a file tree under 'MicroPython device' with several example projects like '00.0_HelloWorld' and '06.2_Rainbow_light'. The main window has two tabs: 'main.py' and 'Shell'. The 'main.py' tab 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

```

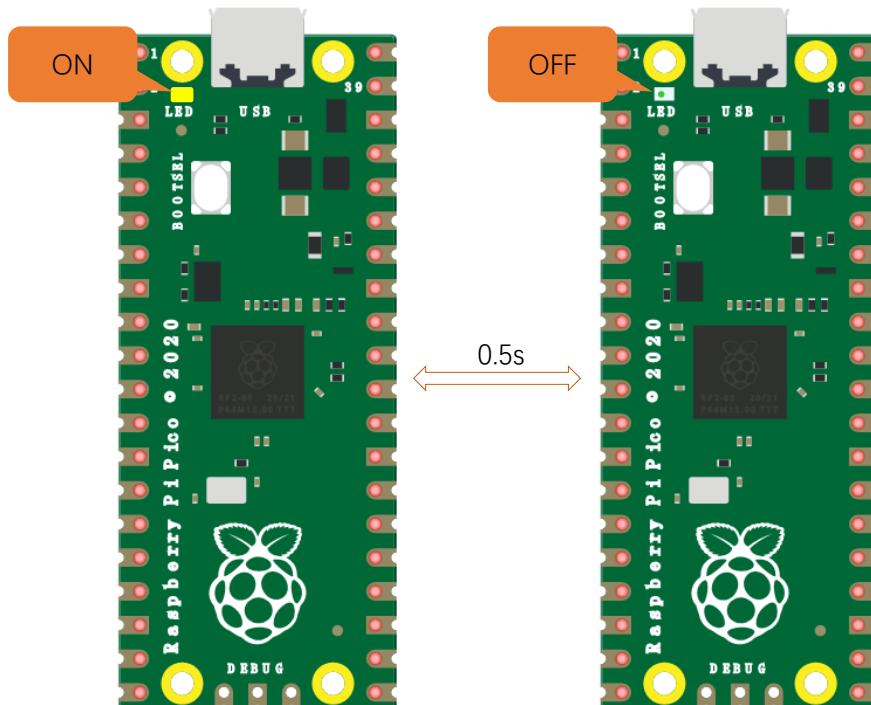
The 'Shell' tab shows the output of running the code:

```

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

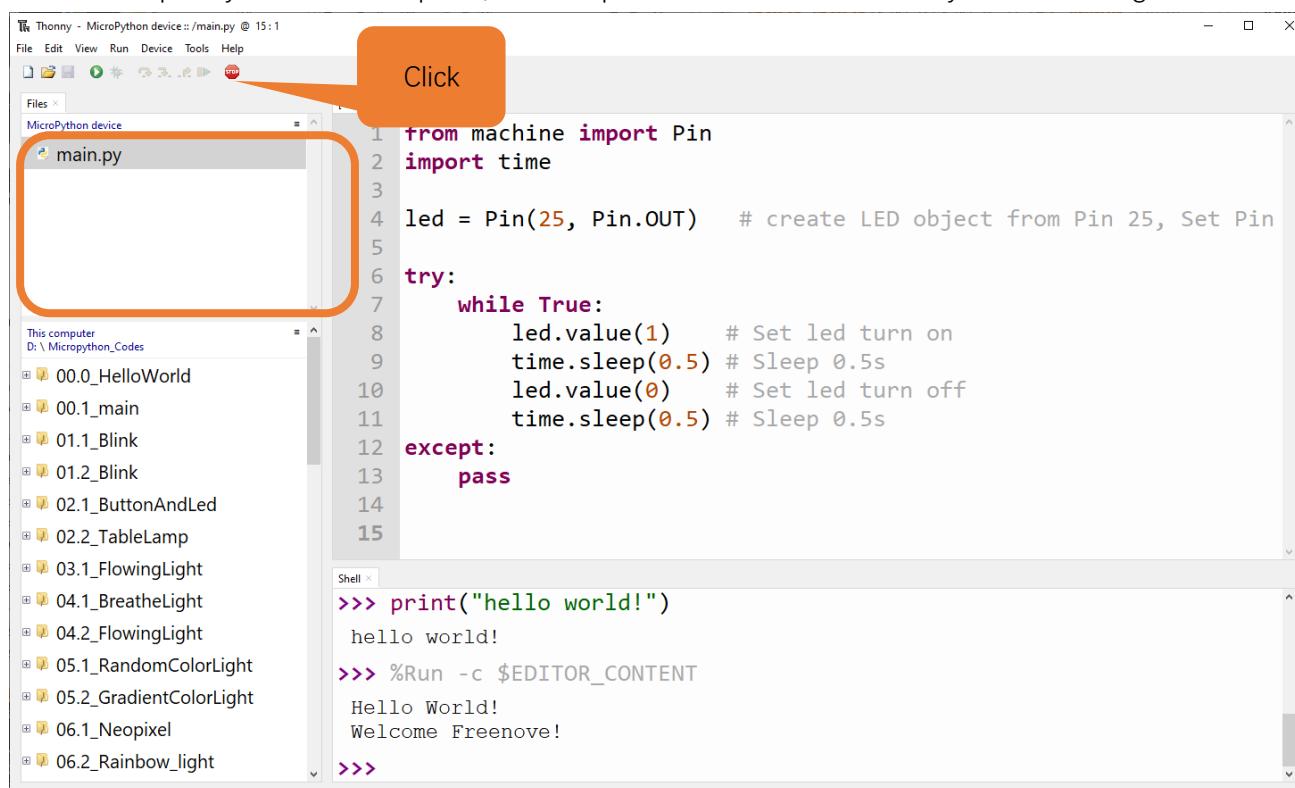
```

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.



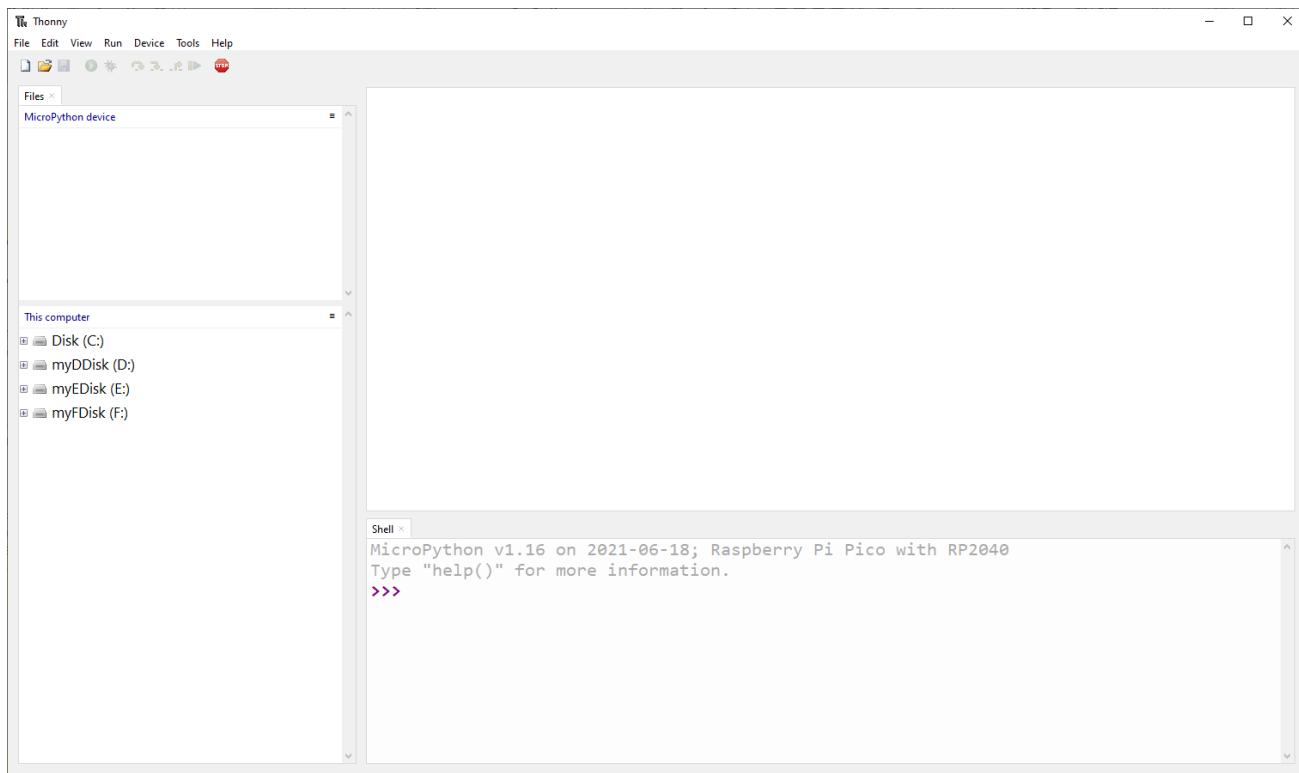
If it does not work, please click on “stop/restart backend” for more times or reconnect Pico.



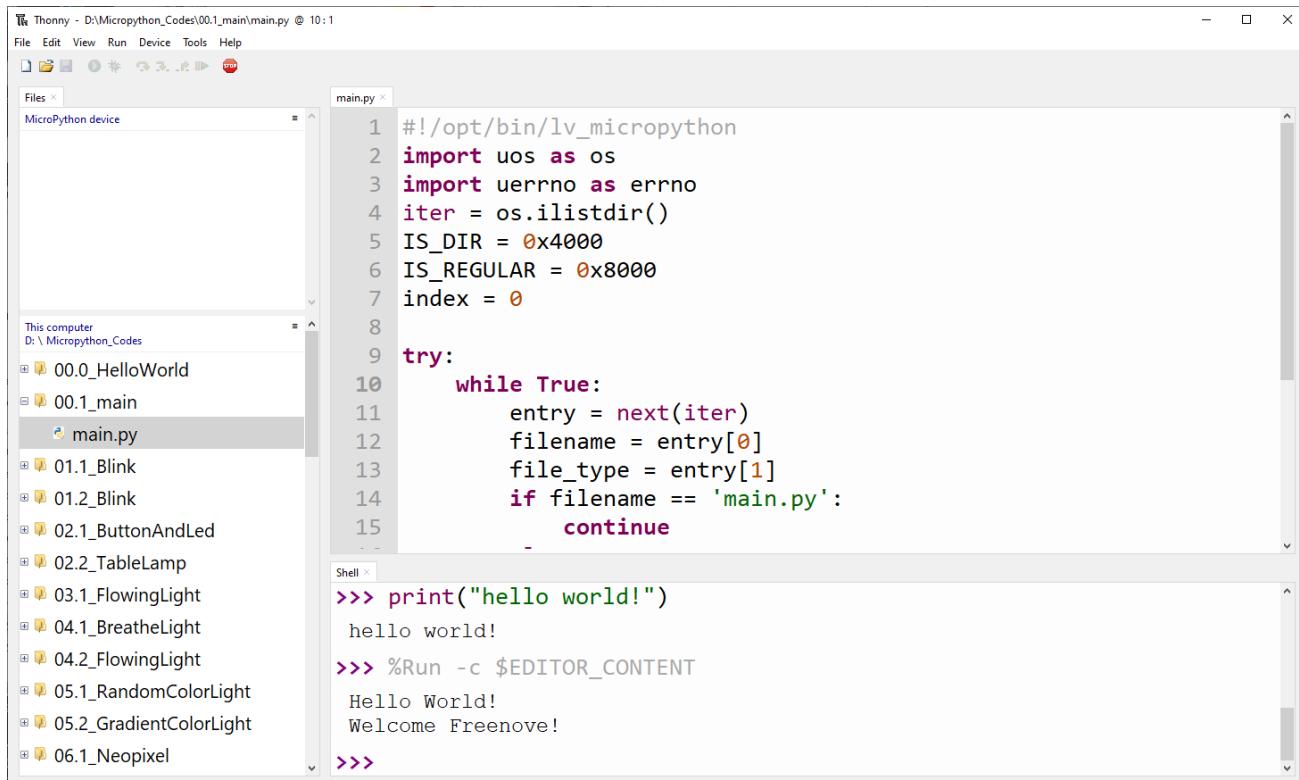


We provide a main.py file for running offline. The code added to main.py is a bootstrap that executes the user's code file. All you need to do is upload the offline project's code file (.py) to the Raspberry Pi Pico device.

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

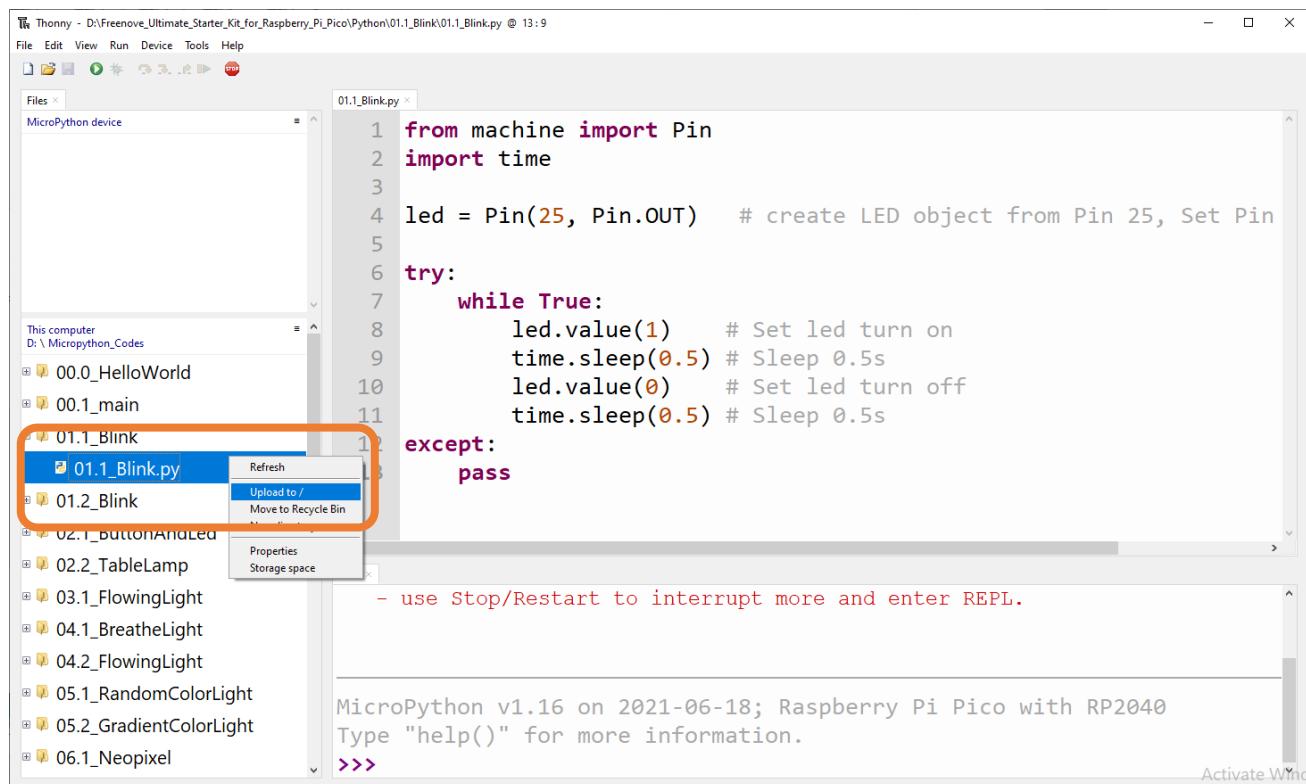


2. Expand “00.1_main” in the “Micropython_Codes” in the directory of disk(D), and double-click main.py, which is provided by us to enable programs in “MicroPython device” to run offline.

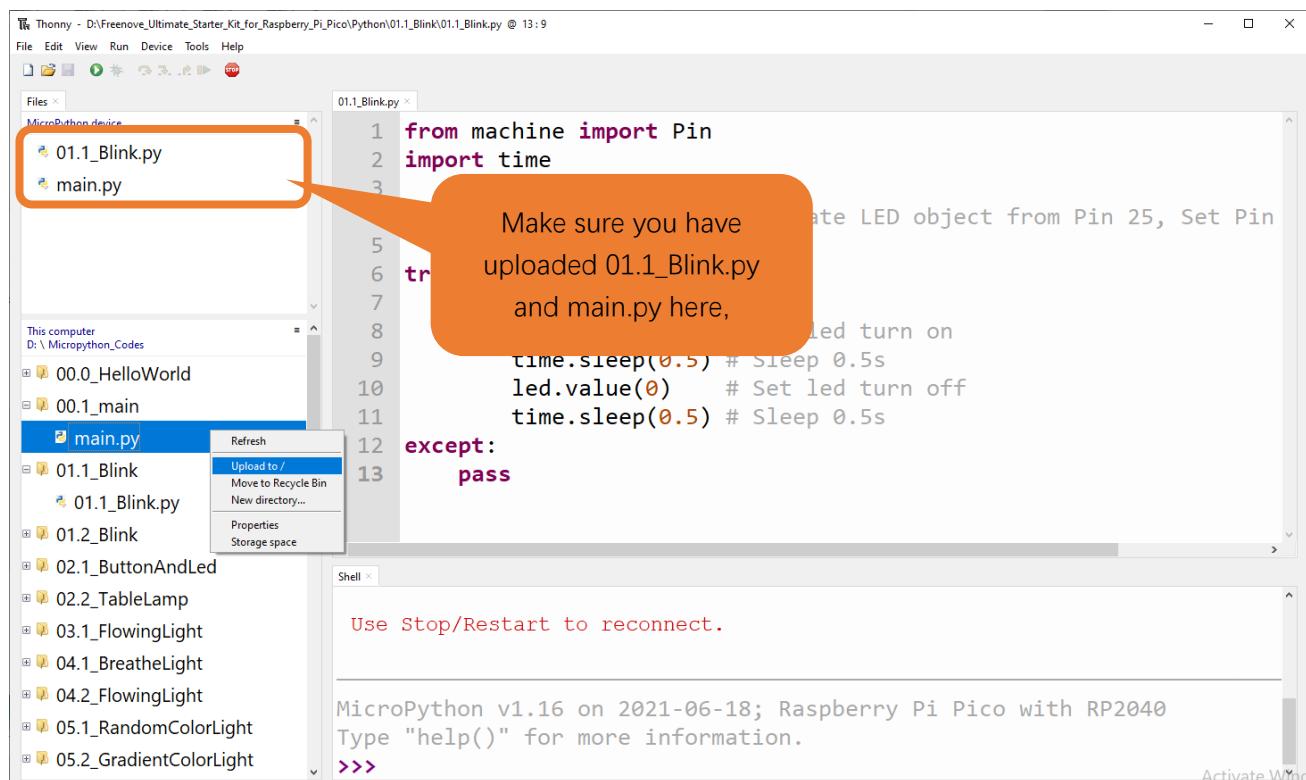


Here we use 00.1 and 01.1 cases as demonstration. The LED on Raspberry Pi Pico is used to show the result, which uses GP25 pin. If you have modified 01.1_Blink.py file, you need to change it accordingly.

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.

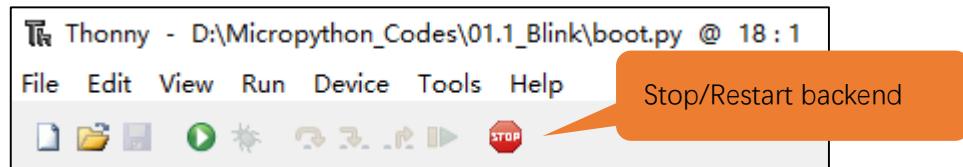


Disconnect Raspberry Pi Pico USB cable and reconnect it, the LED on pico will blink repeatedly.

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

Note:

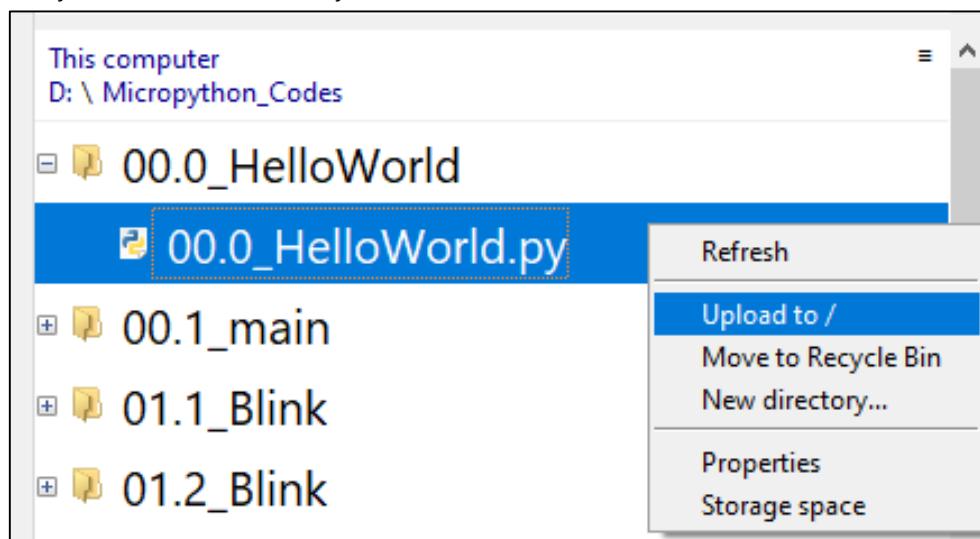
Codes here are run offline. If you want to stop running offline and enter Shell, just click "Stop" in Thonny.



0.6 Thonny Common Operation

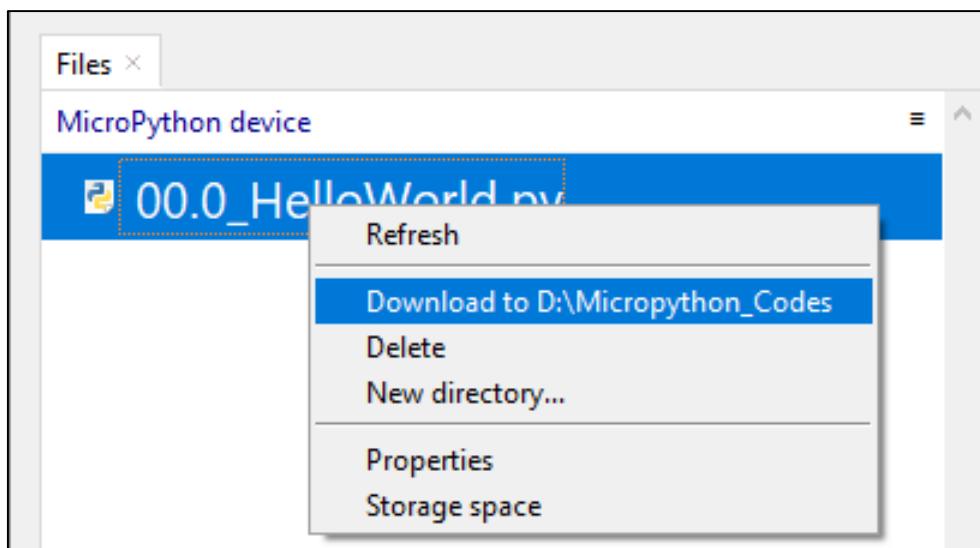
Uploading Code to Raspberry Pi Pico

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

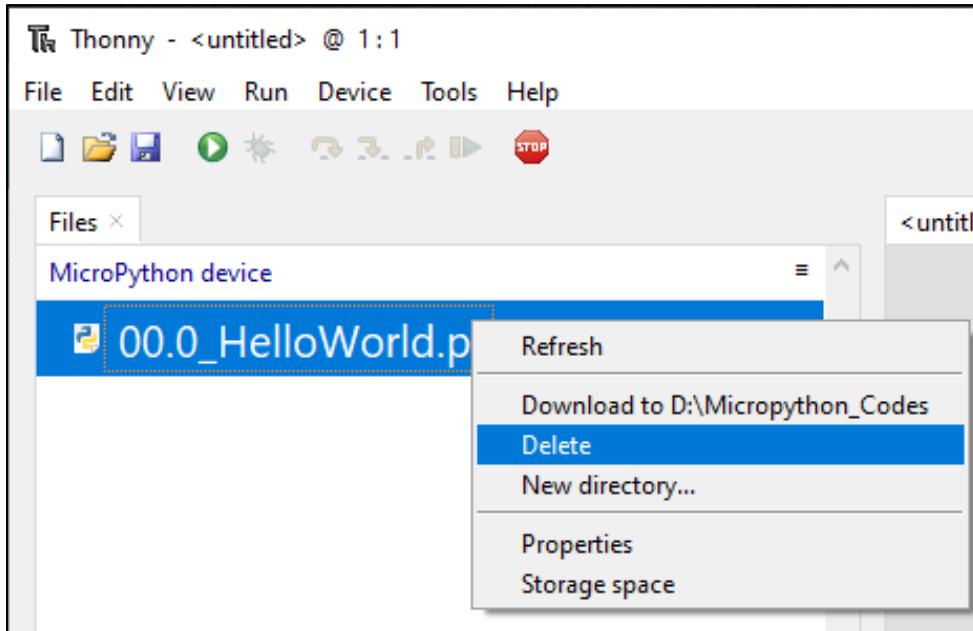
Select “00.0_HelloWorld.py” in “MicroPython device”, right-click to select “Download to ...” to download the code to your computer.





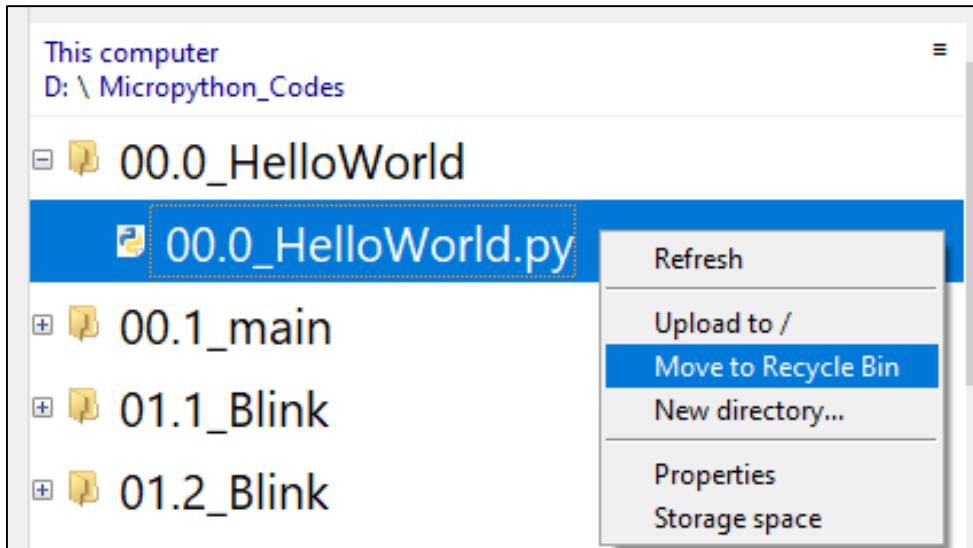
Deleting Files from Raspberry Pi Pico's Root Directory

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.



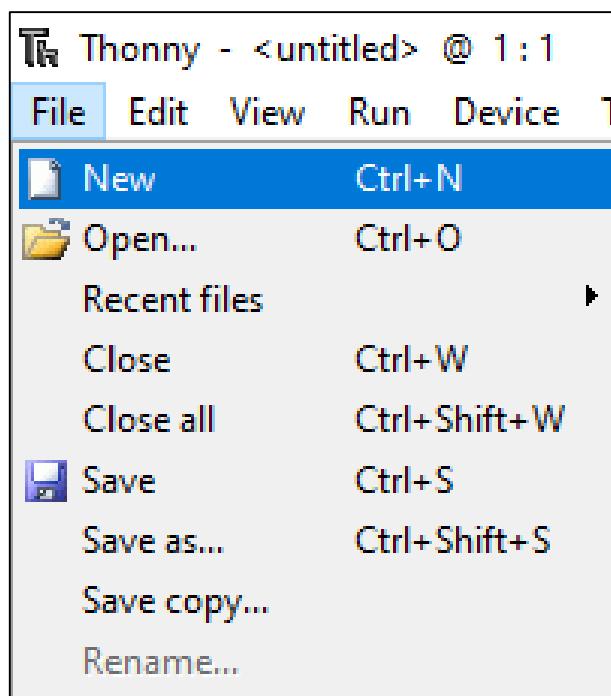
Deleting Files from your Computer Directory

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.

A screenshot of the Thonny IDE interface. The top bar shows "Thonny - <untitled> @ 13:9" and the "File" menu is open. The main area has three tabs: "Files", "MicroPython device", and "Shell". The "Files" tab shows a file tree with several projects like "00.0_HelloWorld", "01.1_main", "01.1_Blink", etc. The "MicroPython device" tab shows a code editor with the following Python script:

```
from machine import Pin
import time

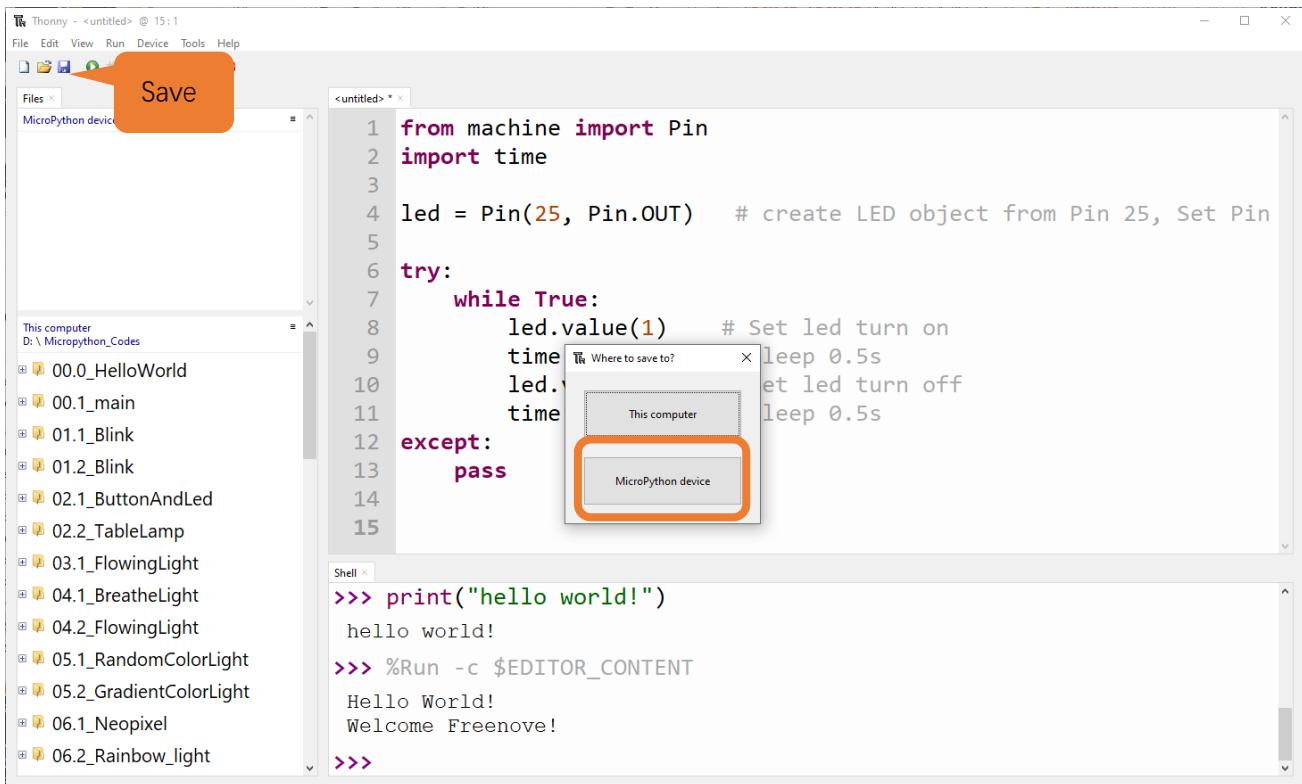
led = Pin(25, Pin.OUT) # create LED object from Pin 25, Set Pin

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 shows the output of running the script:

```
>>> 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.



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

The screenshot shows the Thonny IDE interface. The left sidebar displays a file tree under 'MicroPython device' with various project folders like 00.0_HelloWorld, 00.1_main, etc. A red box highlights the 'main.py' file in the list. The main window has two tabs: 'Files' (selected) and 'Shell'. The 'Files' tab shows the code for 'main.py':

```

1 from machine import Pin
2 import time
3
4 led = Pin(25, Pin.OUT)      # create LED object from Pin 25, Set Pin
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
14
15

```

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

```

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

```

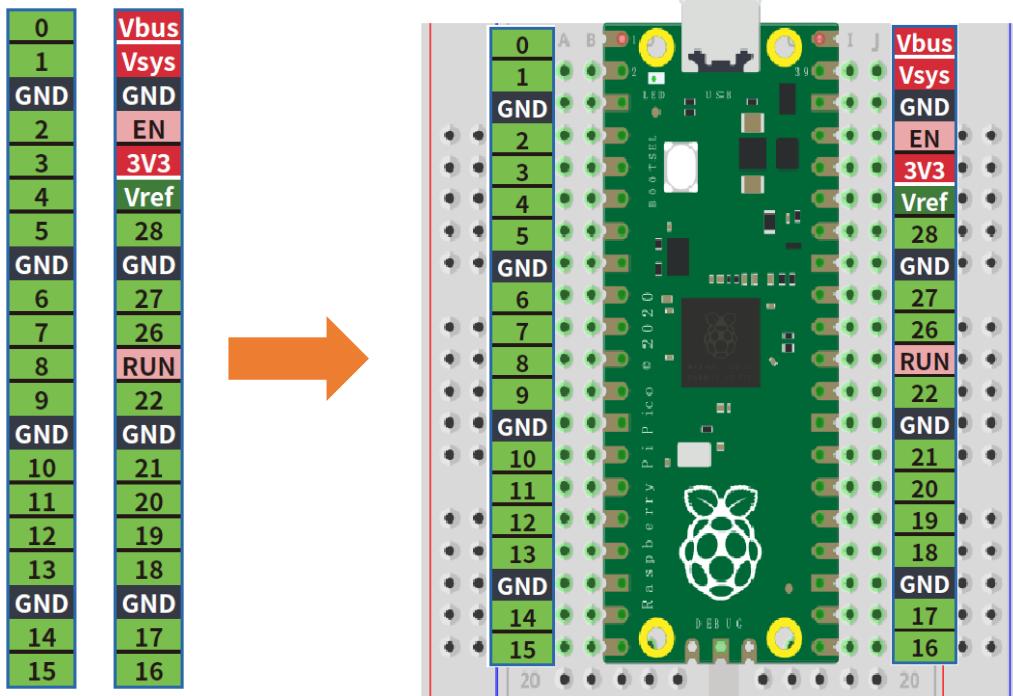
Click "Run" and the LED on Raspberry Pi Pico will blink periodically.

The screenshot shows the Thonny IDE interface again. A red box highlights the 'Run' button in the toolbar. The left sidebar and main editor window are identical to the previous screenshot, displaying the 'main.py' code and its execution results in the shell.



0.7 Paste the Sticker on the Breadboard

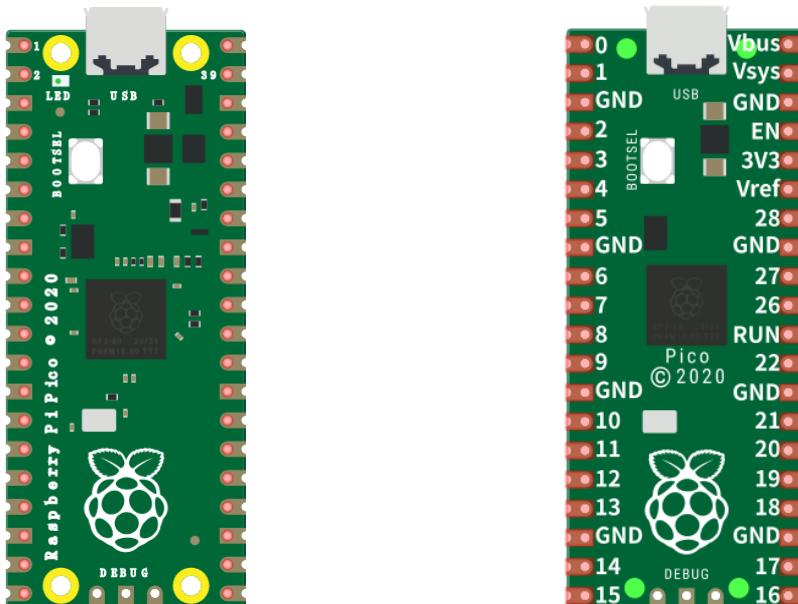
It is not difficult to use the Pico. However, officially, the pin functions are printed on the back of the board, which makes it inconvenient to use. To help users finish each project in the tutorial faster and easier, we provide stickers of the pin functions as follows:



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

Note: The functional pin sequence of Pico, Pico W, Pico 2 and Pico 2W is the same. Therefore, even if your kit is Pico W, Pico 2 and Pico 2W, it is also applicable to the sticker above.

To make the tutorial more intuitive, we have 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)

Note: Raspberry Pi Pico, Raspberry Pi Pico W, Raspberry Pi Pico 2 and Raspberry Pi Pico 2W only differ by one wireless function, and are almost identical in other aspects. In this tutorial, except for the wireless function, other parts use Raspberry Pi Pico's map for tutorial demonstration.

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.

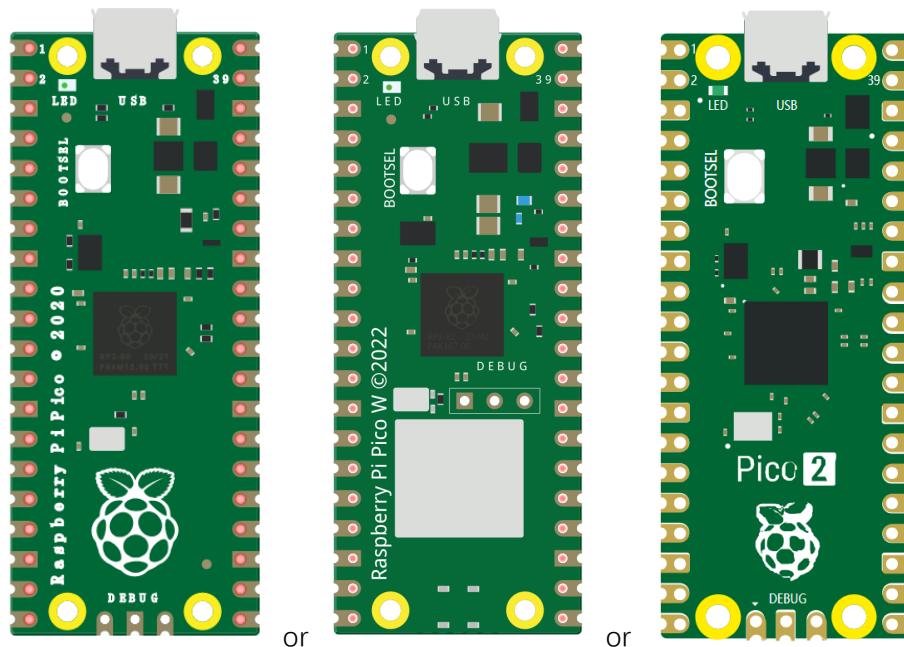
If you have not yet installed Thonny, click [here](#).

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 (or Pico W or Pico 2) x1



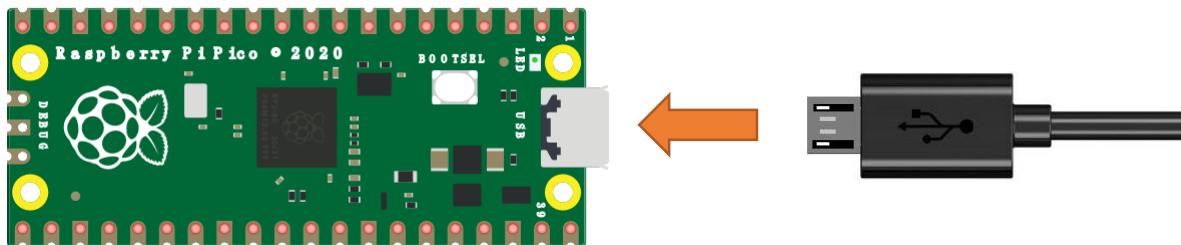
USB cable x1





Power

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



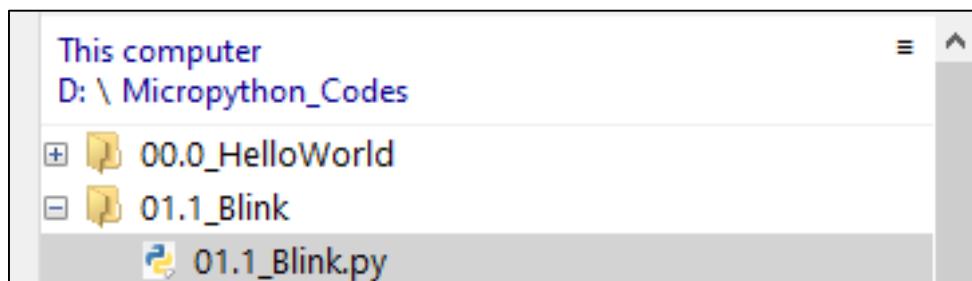
Code

Codes used in this tutorial are saved in

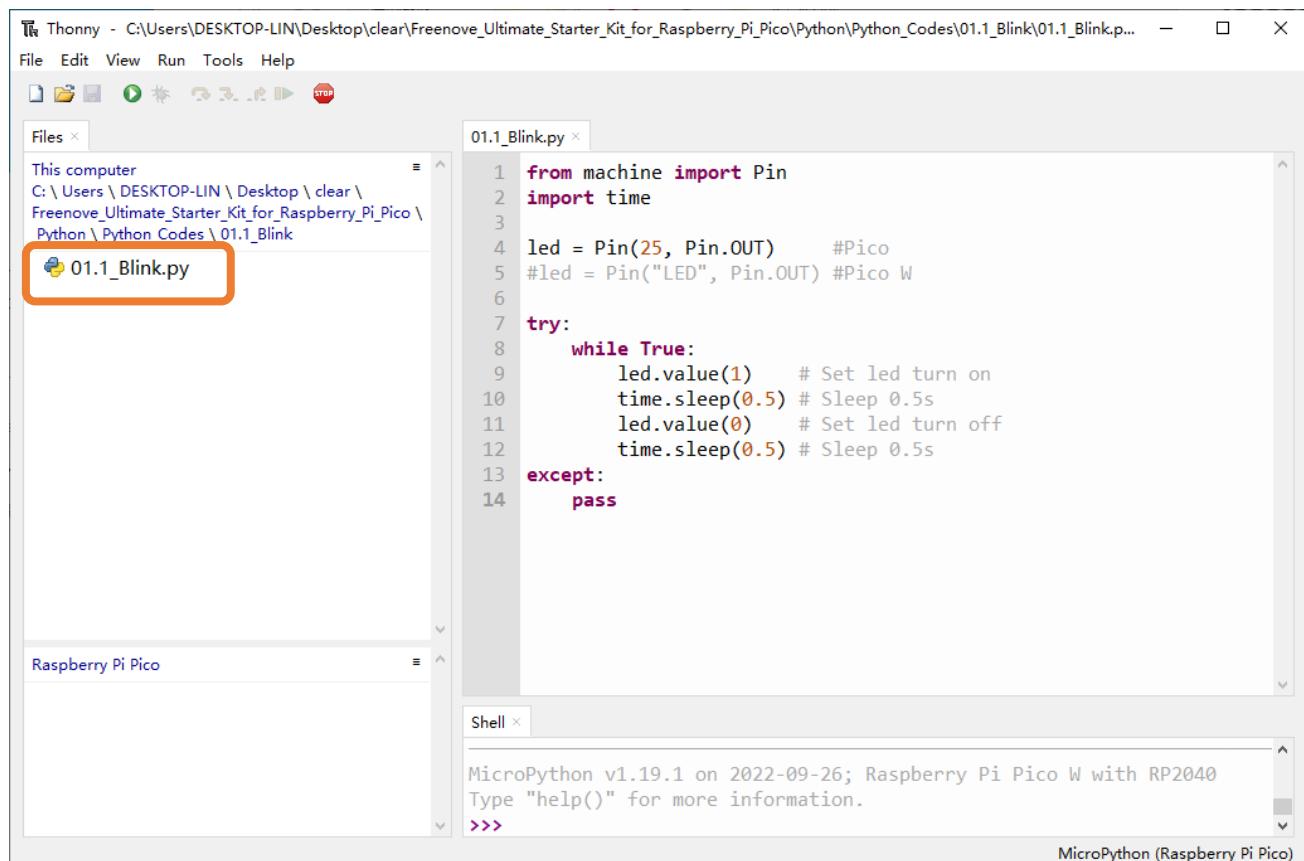
"Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico/Python_Codes". You can move the codes to any location. For example, we save the codes in Disk(D) with the path of "**D:/Micropython_Codes**".

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

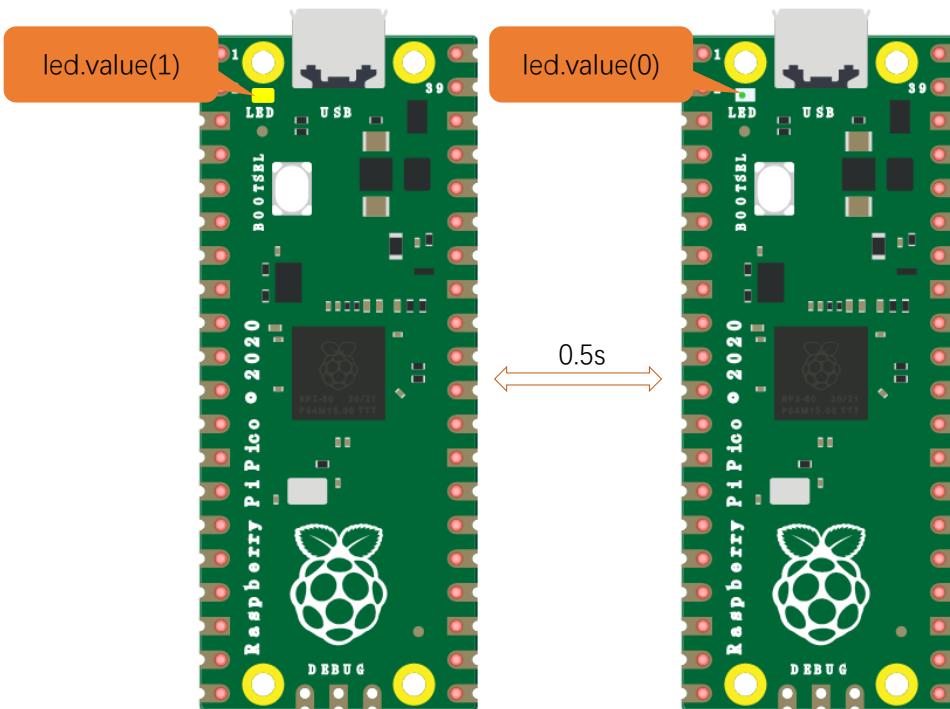
try:
    while True:
        led.value(1)      # Set led turn on
        time.sleep(0.5)   # Sleep 0.5s
        led.value(0)      # Set led turn off
        time.sleep(0.5)   # Sleep 0.5s
except:
    pass

```

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

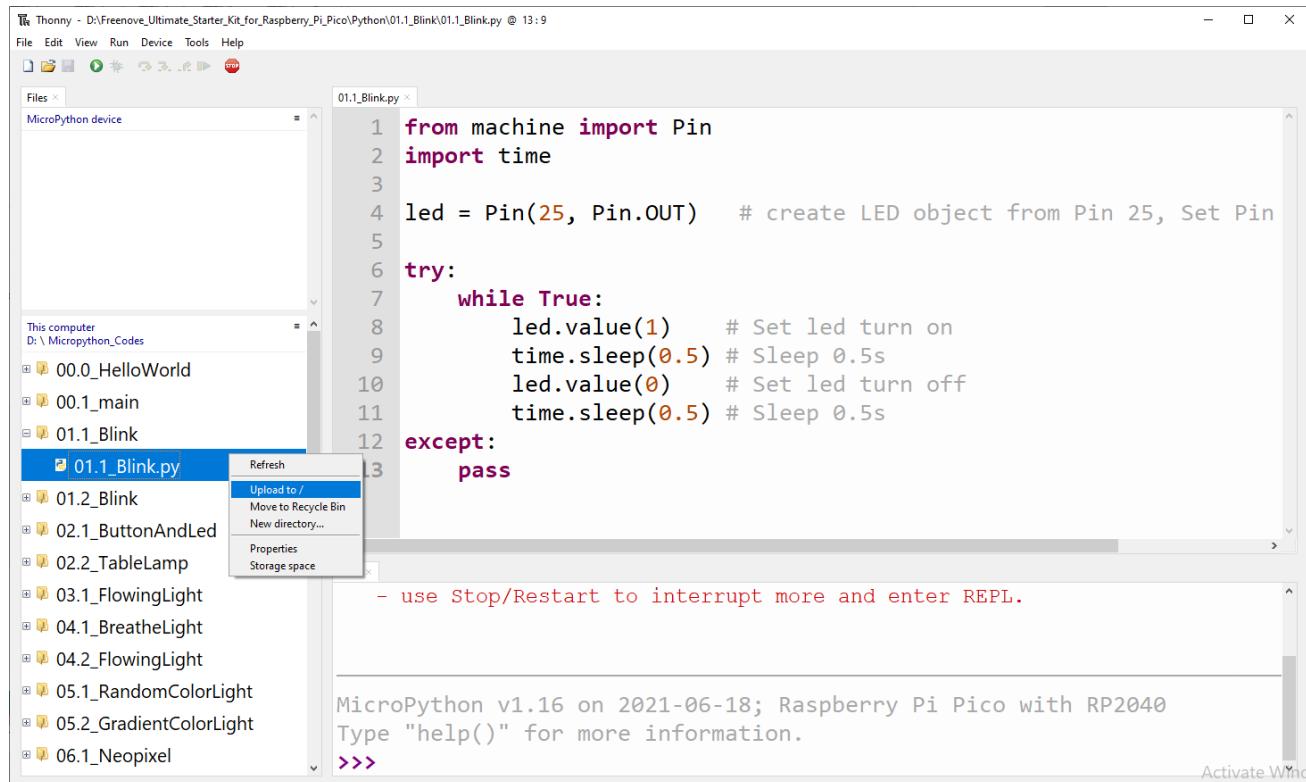
led = Pin(25, Pin.OUT) # create LED object from Pin 25, Set Pin

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
```

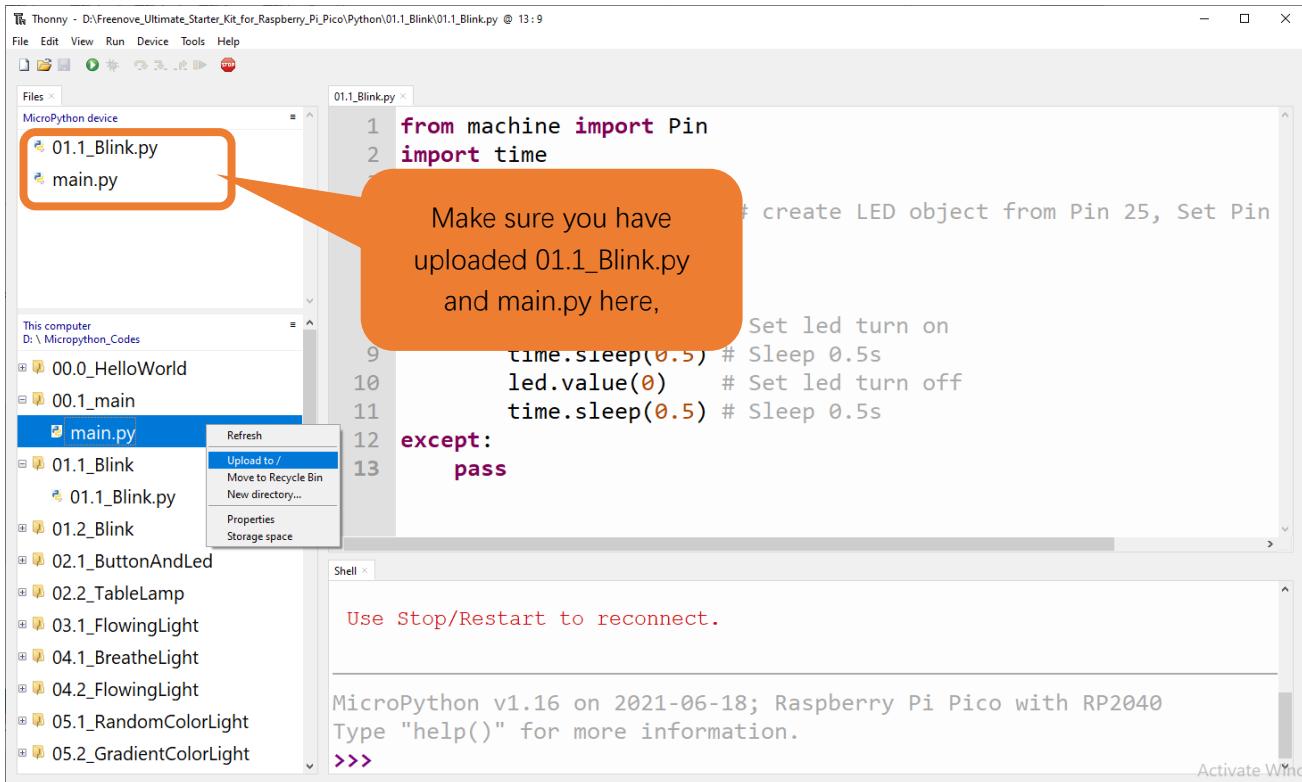
MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
->
Connection lost (EOF)
Use Stop/Restart to reconnect.

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.



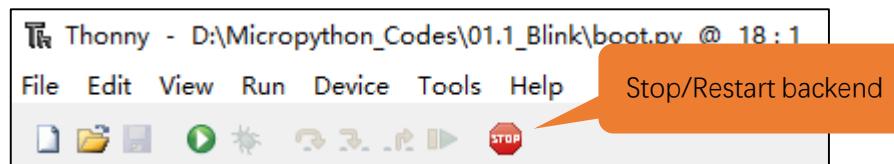
Upload main.py in the same way.



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

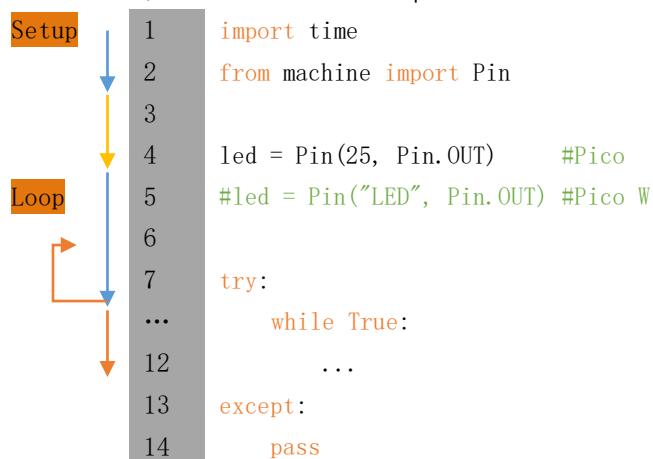
The following is the program code:

```

1 import time
2 from machine import Pin
3
4 led = Pin(25, Pin.OUT)      #Pico
5 #led = Pin("LED", Pin.OUT) #Pico W
6
7 try:
8     while True:
9         led.value(1)      #Set led turn on
10        time.sleep(0.5) #Sleep 0.5s
11        led.value(0)      #Set led turn off
12        time.sleep(0.5) #Sleep 0.5s
13 except:
14     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".

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

```

4 led = Pin(25, Pin.OUT)      #Pico
5 #led = Pin("LED", Pin.OUT) #Pico W

```

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 led pin 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 except:  
13     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

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.

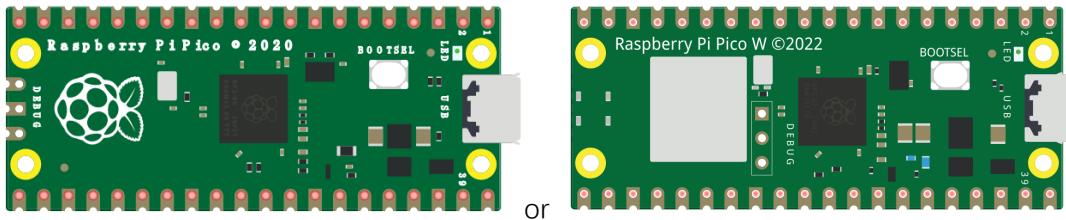
Handler: callback function.

Project 1.2 Blink

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

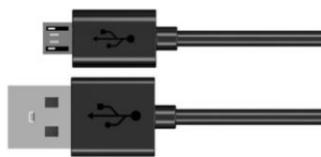
Component List

Raspberry Pi Pico (or Pico W) x1

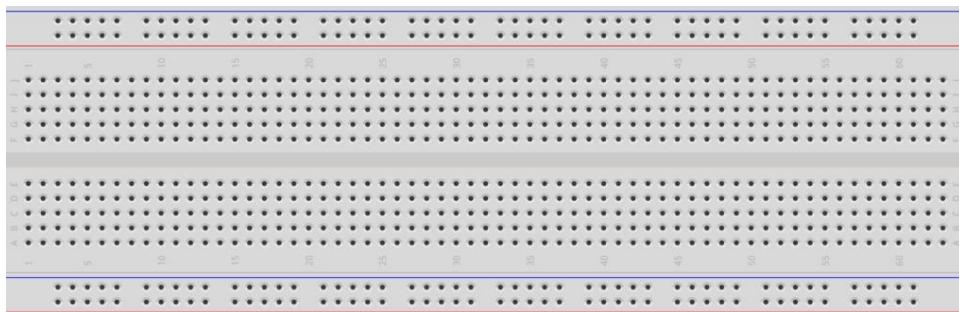


or

USB cable x1



Breadboard x1



LED x1



Resistor 220Ω x1



Jumper

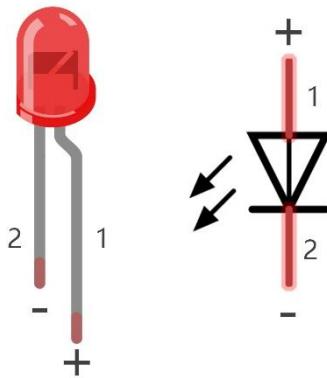


Component knowledge

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 two-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.



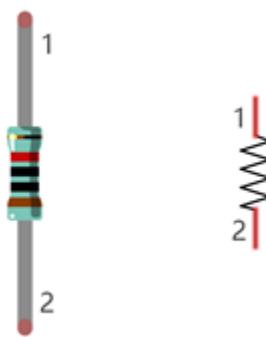
LED	Voltage	Maximum current	Recommended current
Red	1.9 - 2.2V	20mA	10mA
Green	2.9 - 3.4V	10mA	5mA
Blue	2.9 - 3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

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). $1M\Omega=1000k\Omega$, $1k\Omega=1000\Omega$.

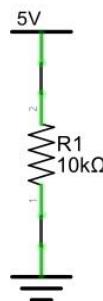
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



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: $I=U/R=5V/10k\Omega=0.0005A=0.5mA$.



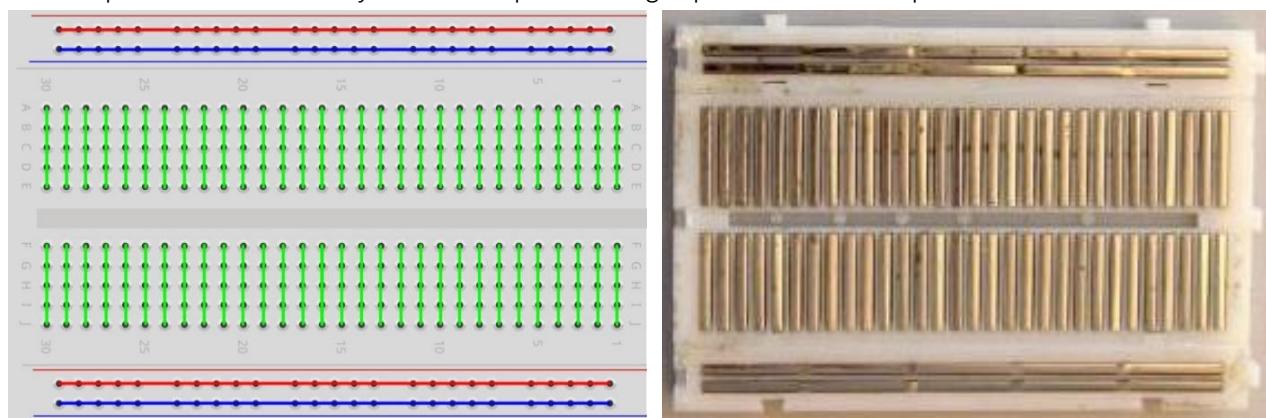
WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

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

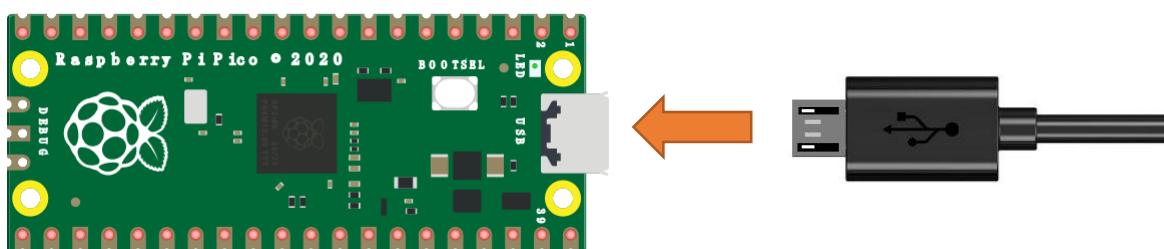
Here we have a small breadboard as an example of how the rows of holes (sockets) are electrically attached.

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.

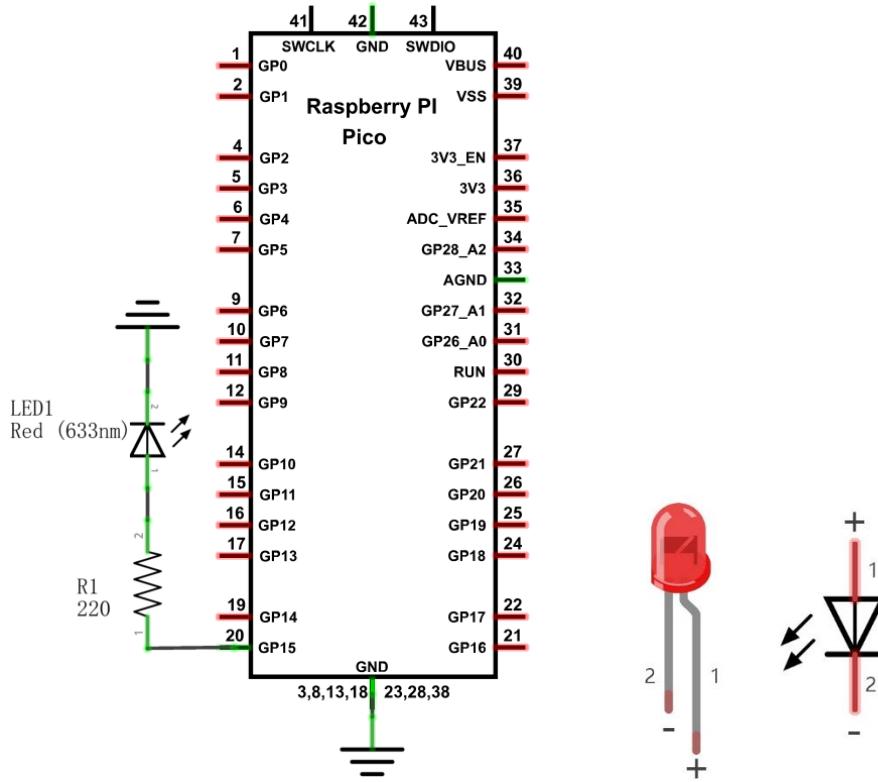


Circuit

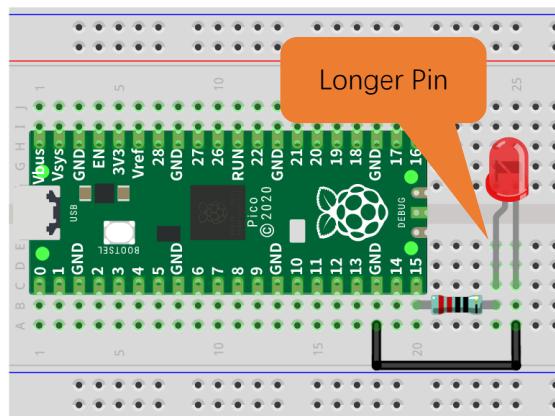
First, disconnect all power from the Raspberry Pi Pico. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the PC to Raspberry Pi Pico.

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

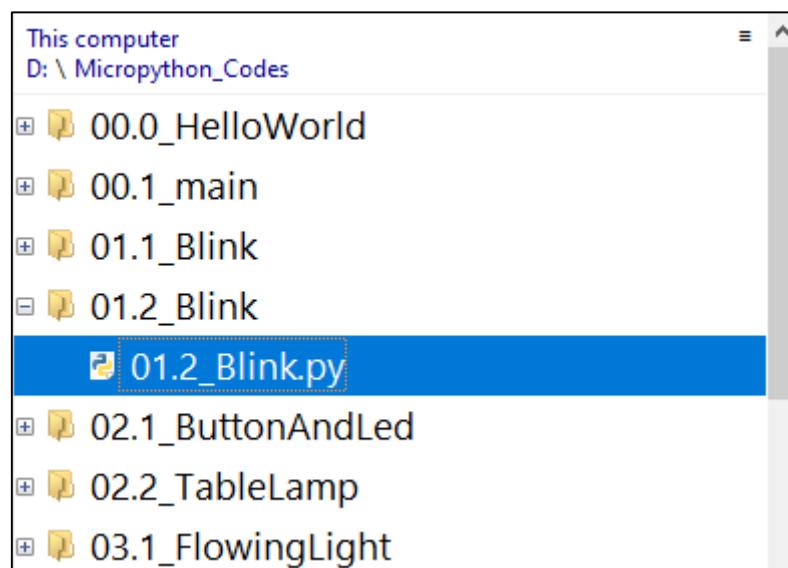
Code

Codes used in this tutorial are saved in

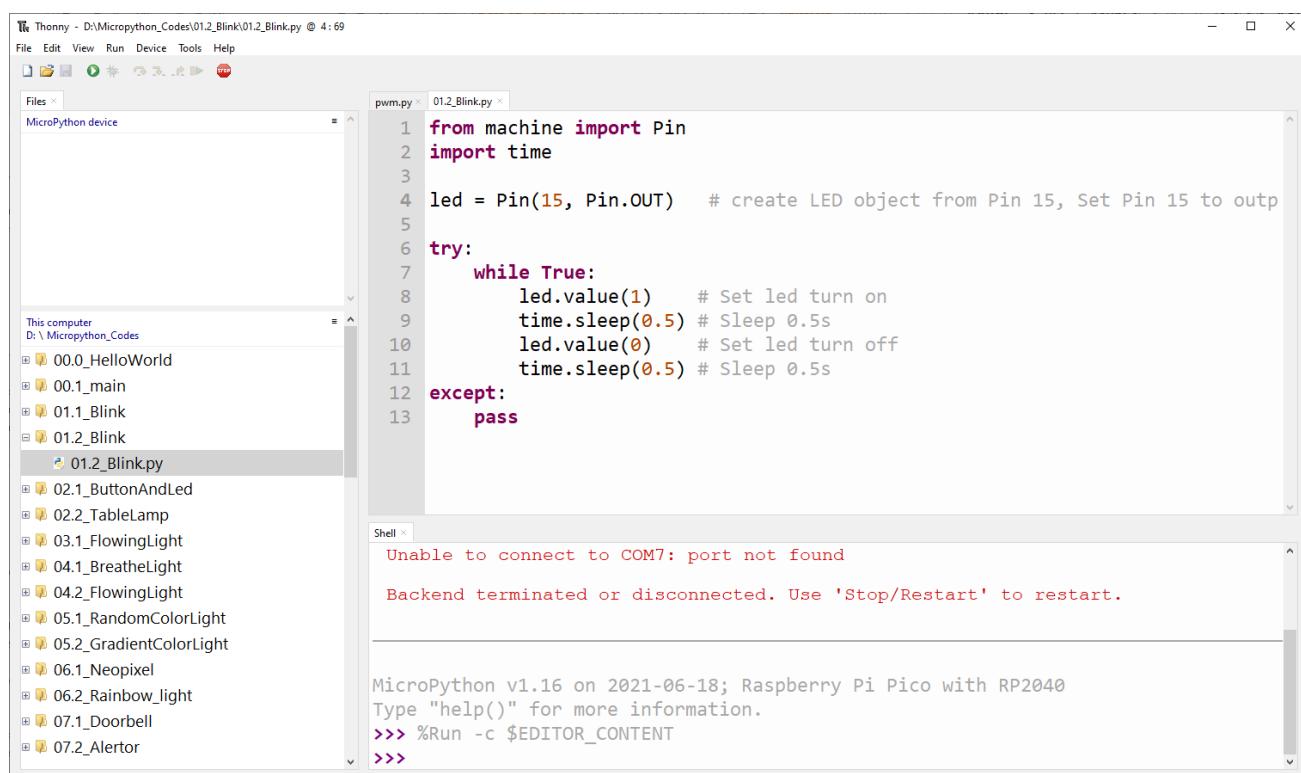
"Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico/Python_Codes". You can move the codes to any location. For example, we save the codes in Disk(D) with the path of "**D:/Micropython_Codes**".

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.

The screenshot shows the Thonny IDE interface. A file named '01.2_Blink.py' is open in the editor. The code is as follows:

```

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

```

The 'Run' button in the toolbar is highlighted with a red circle. A callout bubble labeled '1, Stop/Restart backend' points to the 'Stop/Restart' button in the toolbar. Another callout bubble labeled '2, Run current script' points to the 'Run' button. The 'Files' sidebar shows other projects like '00.0_HelloWorld', '00.1_main', etc. The 'Shell' tab shows the following text:

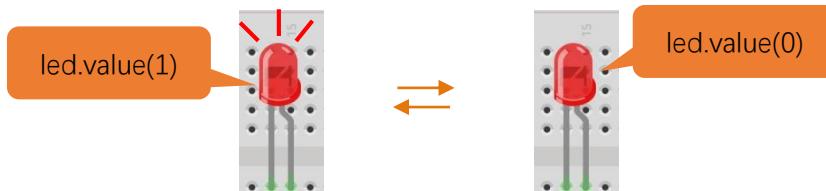
```

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

```

A callout bubble on the right says: "This indicates that the connection is successful."

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:

```

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.

```

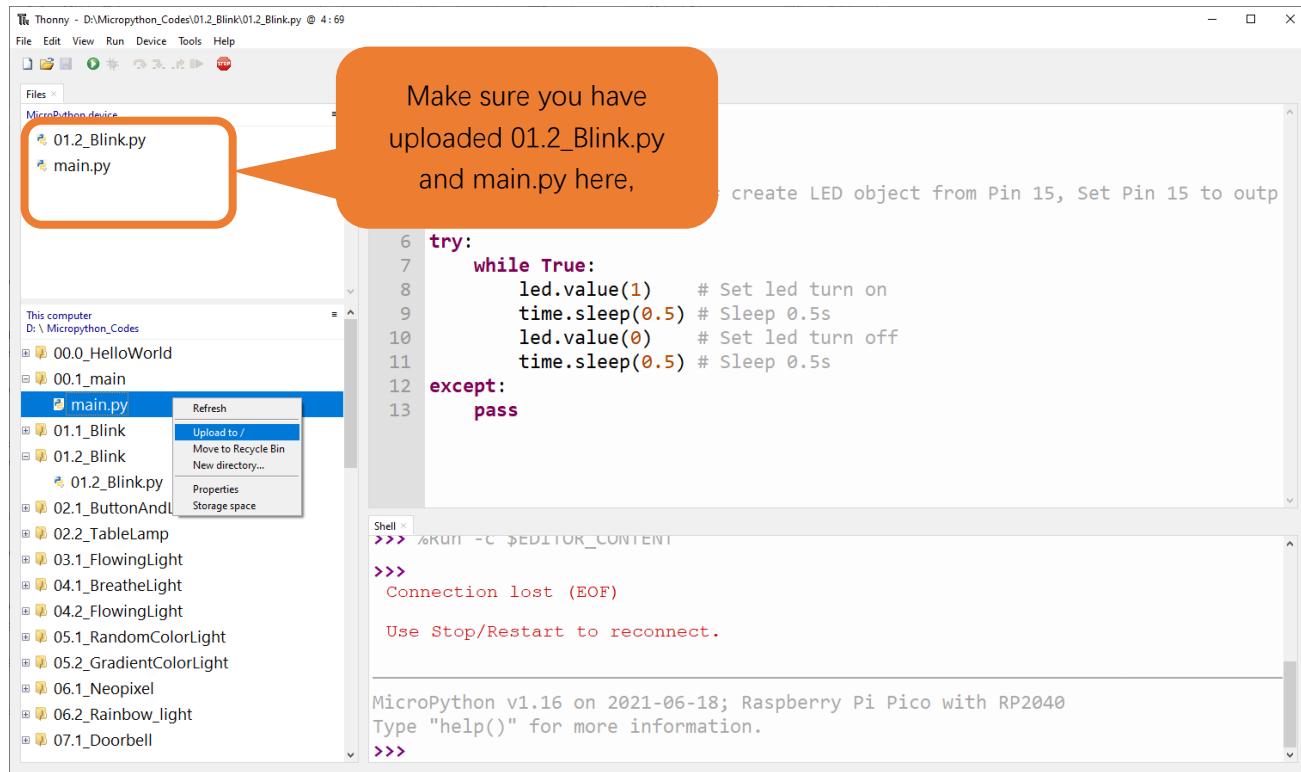
Uploading code to Raspberry Pi Pico

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 remains the same as the previous screenshot.

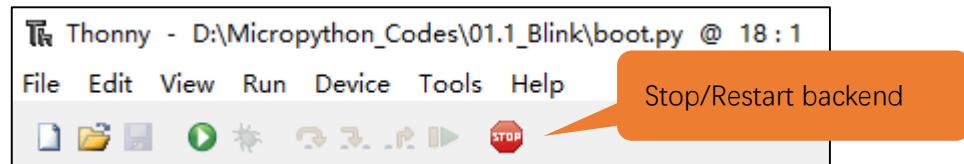
Upload main.py in the same way.



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.



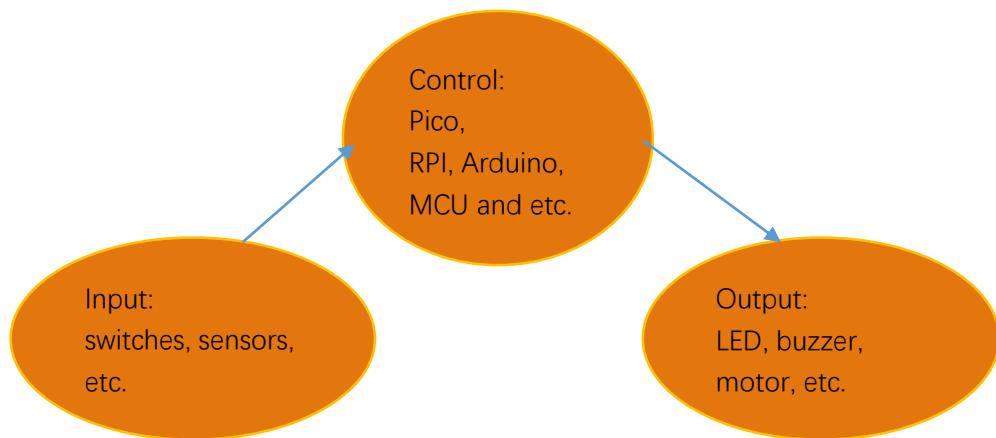
If you have any concerns, please contact us via: support@freenove.com

Any concerns? ✉ support@freenove.com

Chapter 2 Button & LED

Note: Raspberry Pi Pico, Raspberry Pi Pico W, Raspberry Pi Pico 2 and Raspberry Pi Pico 2W only differ by one wireless function, and are almost identical in other aspects. In this tutorial, except for the wireless function, other parts use Raspberry Pi Pico's map for tutorial demonstration.

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

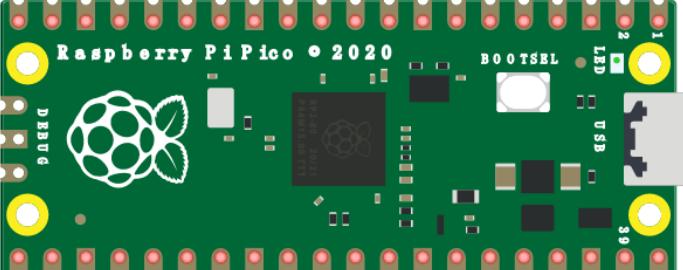
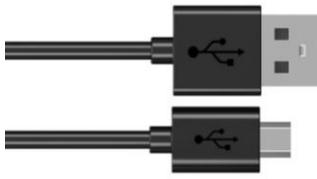
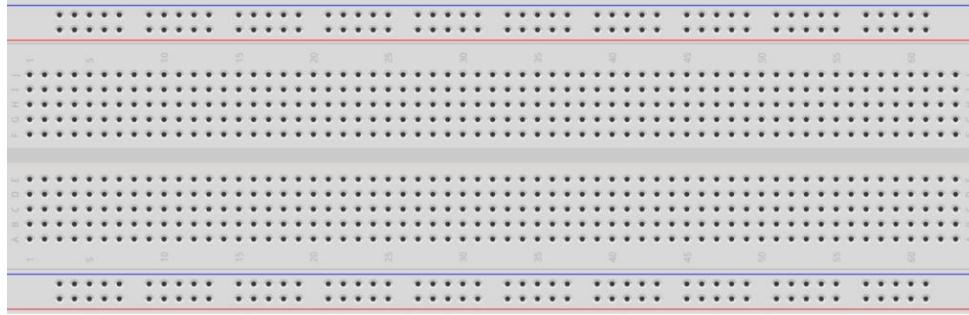
Control: Raspberry Pi Pico

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.

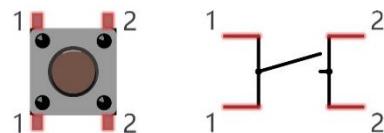
Component List

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

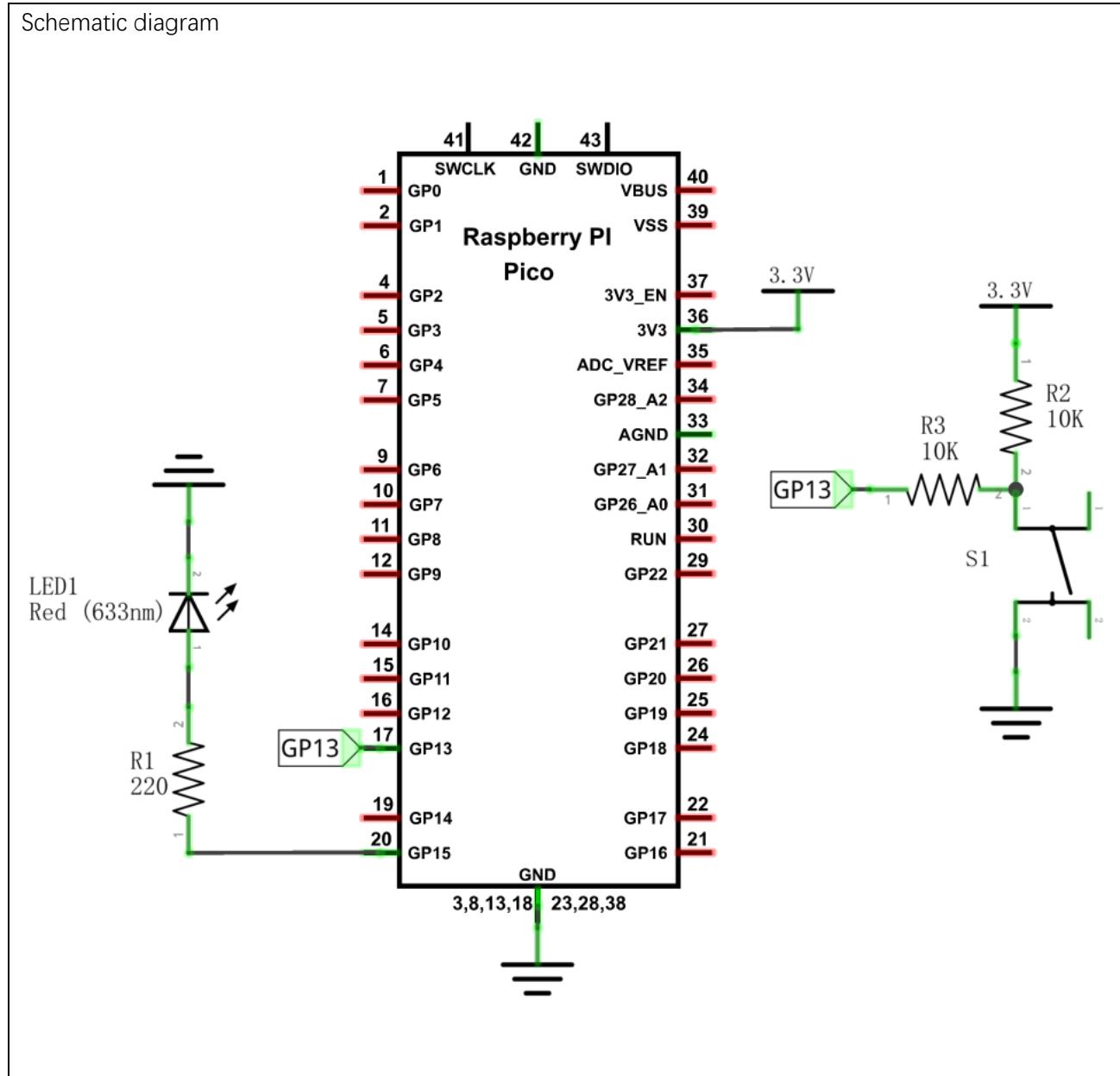
This type of Push Button Switch has four pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



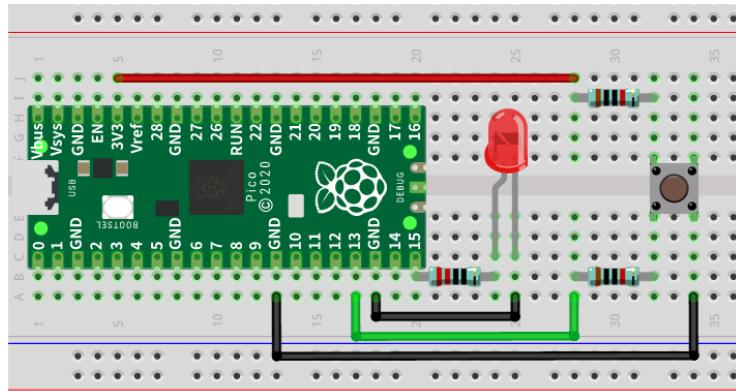
When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

Circuit

Schematic diagram



Hardware connection. If you need any support, please 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 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_Basic_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

```

from machine import Pin
import time

led = Pin(15, Pin.OUT)
button = Pin(13, Pin.IN)      #Create button object from Pin13 , Set GP13 to input

try:
    while True:
        if not button.value():
            led.value(1)          #Set led turn on
        else:
            led.value(0)          #Set led turn off
except:
    pass

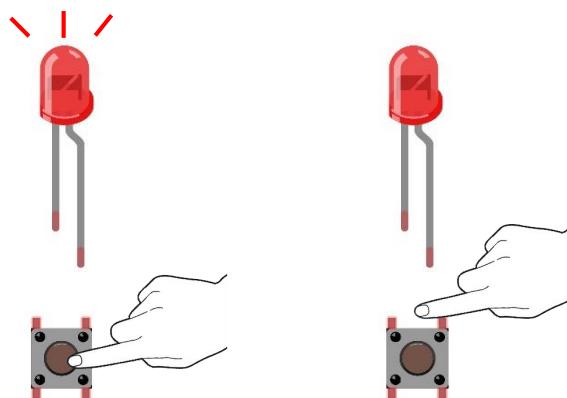
```

Shell >>> %Run -c \$EDITOR_CONTENT

MPY: soft reboot

MicroPython (Raspberry Pi Pico) • Board CDC @ COM17

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)      #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)      #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

```

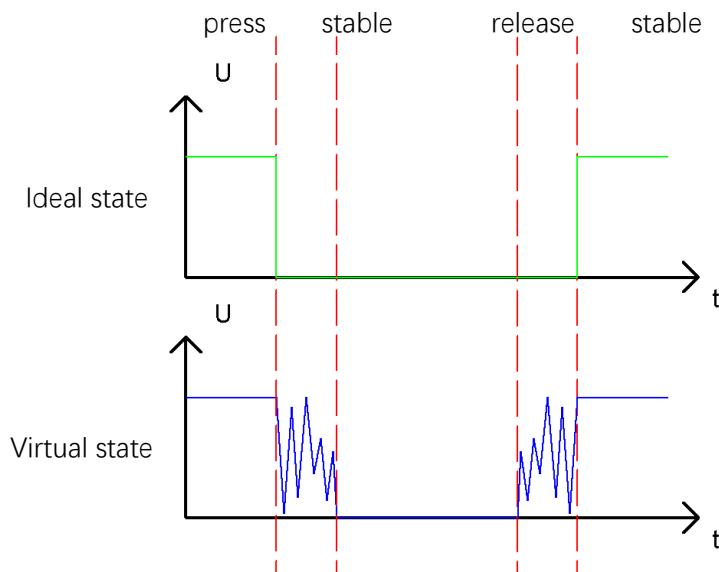
Project 2.2 MINI table lamp

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_Basic_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)      #Create button object from Pin13 , Set GP13 +ve to GND

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():
            reverseGPIO()

```

Shell x

```

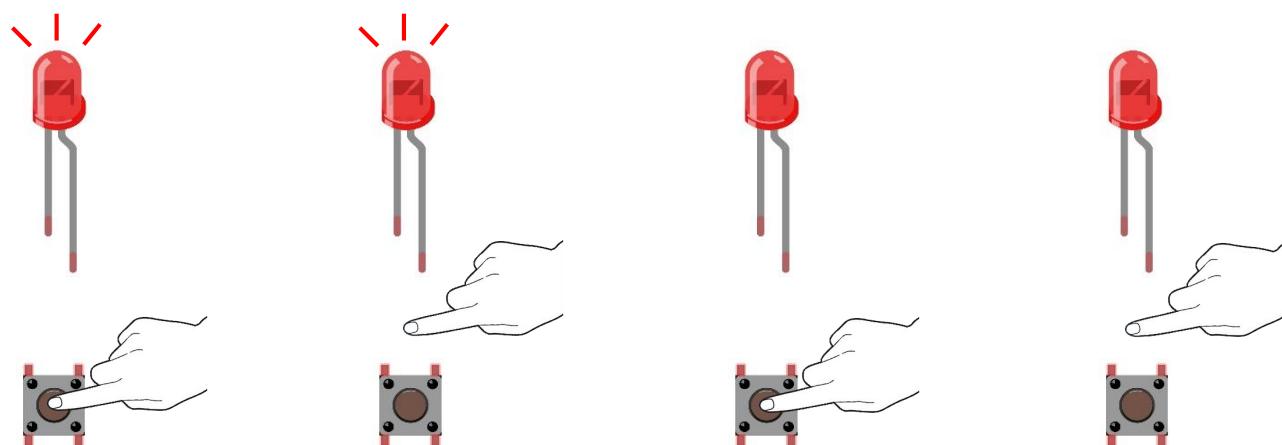
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot

```

MicroPython (Raspberry Pi Pico) • Board CDC @ COM17

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)      #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

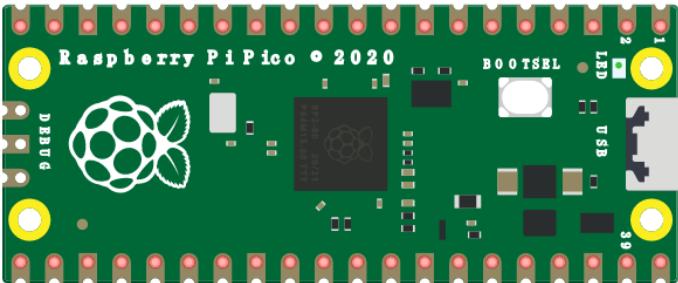
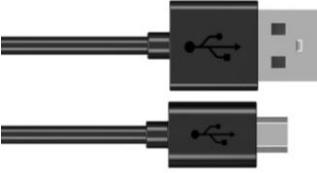
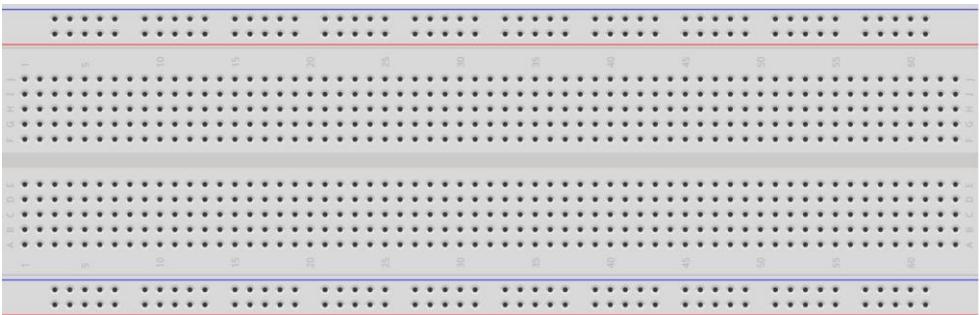
Note: Raspberry Pi Pico, Raspberry Pi Pico W, Raspberry Pi Pico 2 and Raspberry Pi Pico 2W only differ by one wireless function, and are almost identical in other aspects. In this tutorial, except for the wireless function, other parts use Raspberry Pi Pico's map for tutorial demonstration.

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	USB cable x1	
 A green printed circuit board (PCB) for the Raspberry Pi Pico. It features a central Broadcom SoC chip, a USB Type-C port, a microSD card slot, and several pins along the edges. The board is labeled "Raspberry Pi Pico • 2020".	 Two standard black USB cables, each with a black USB-A male connector at one end and a grey USB-B male connector at the other.	
Breadboard x1	 A breadboard with four horizontal rows of 40 numbered pins (1 through 40). The columns are labeled A, B, C, D, E, F, G, H, I along the left and right edges. The breadboard has a central vertical connection point and a red power rail across the top.	
Jumper	LED bar graph x1	Resistor 220Ω x10

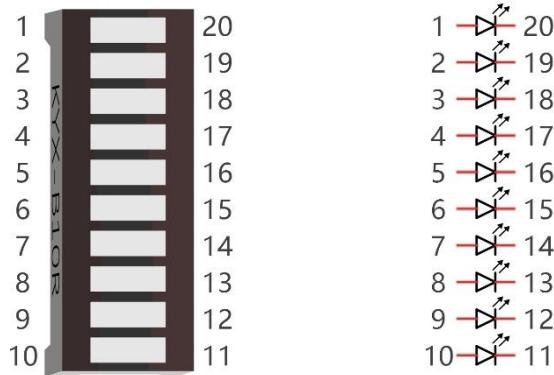


Component knowledge

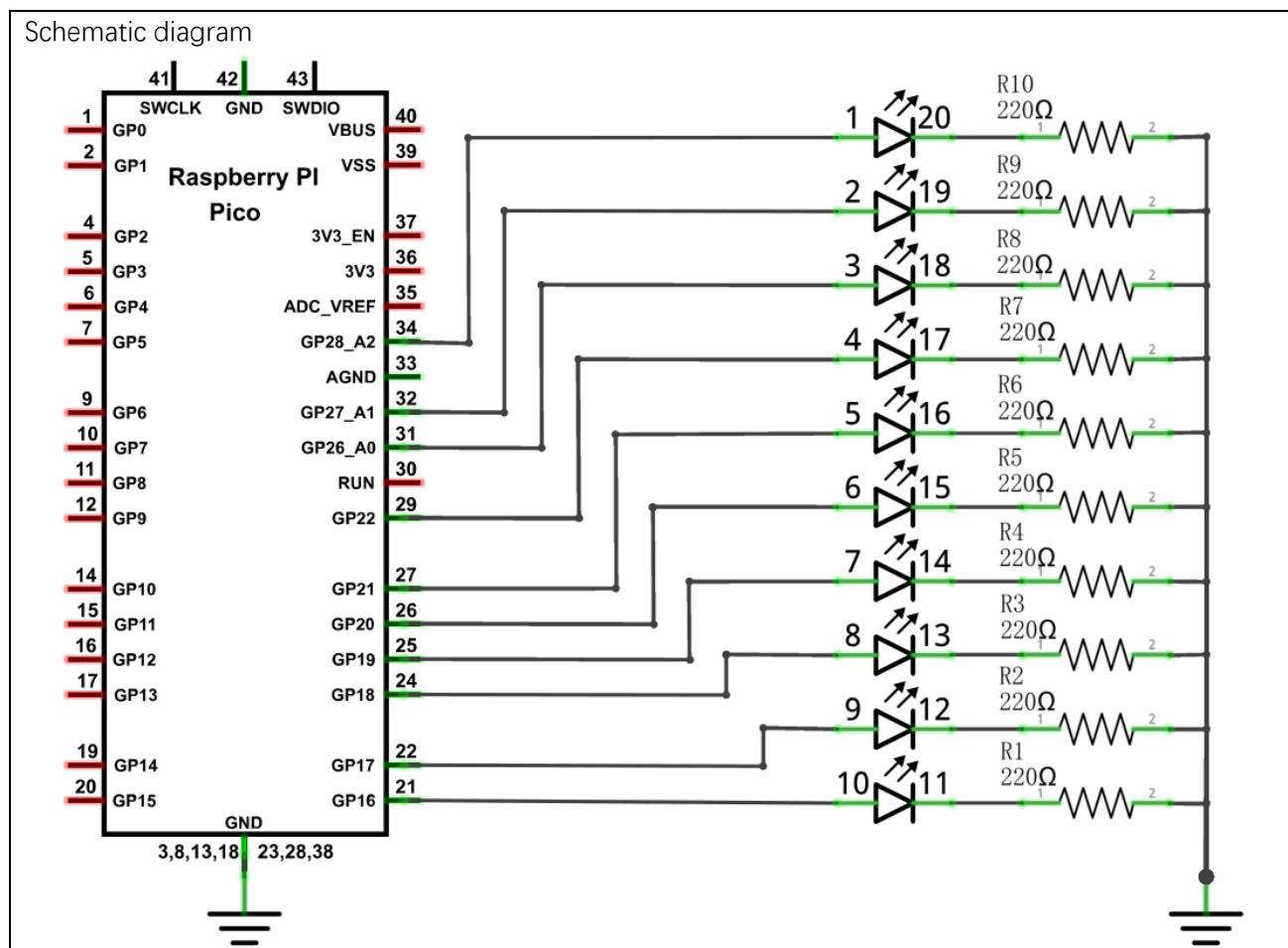
Let us learn about the basic features of these components to use and understand them better.

LED bar

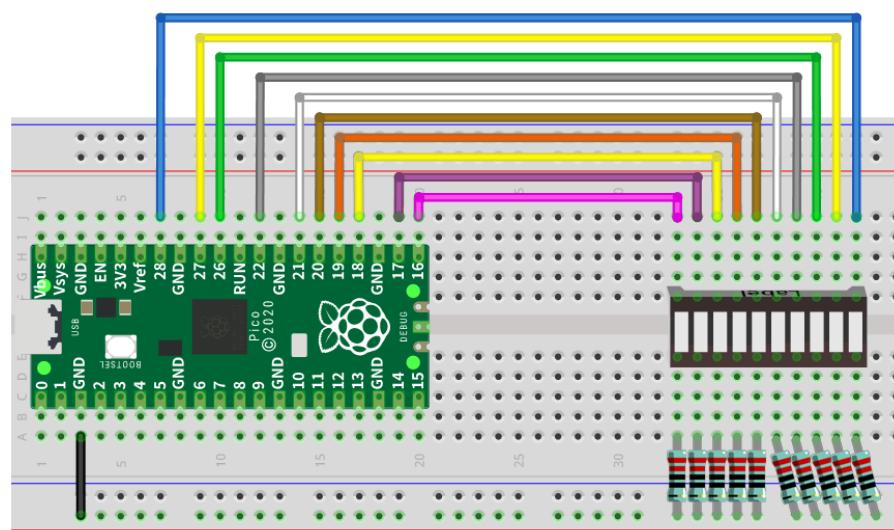
A Bar Graph LED has 10 LEDs integrated into one compact component. The two rows of pins at its bottom are paired to identify each LED like the single LED used earlier.



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 LED bar does not work, try to rotate LED bar for 180°. The label is random.

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

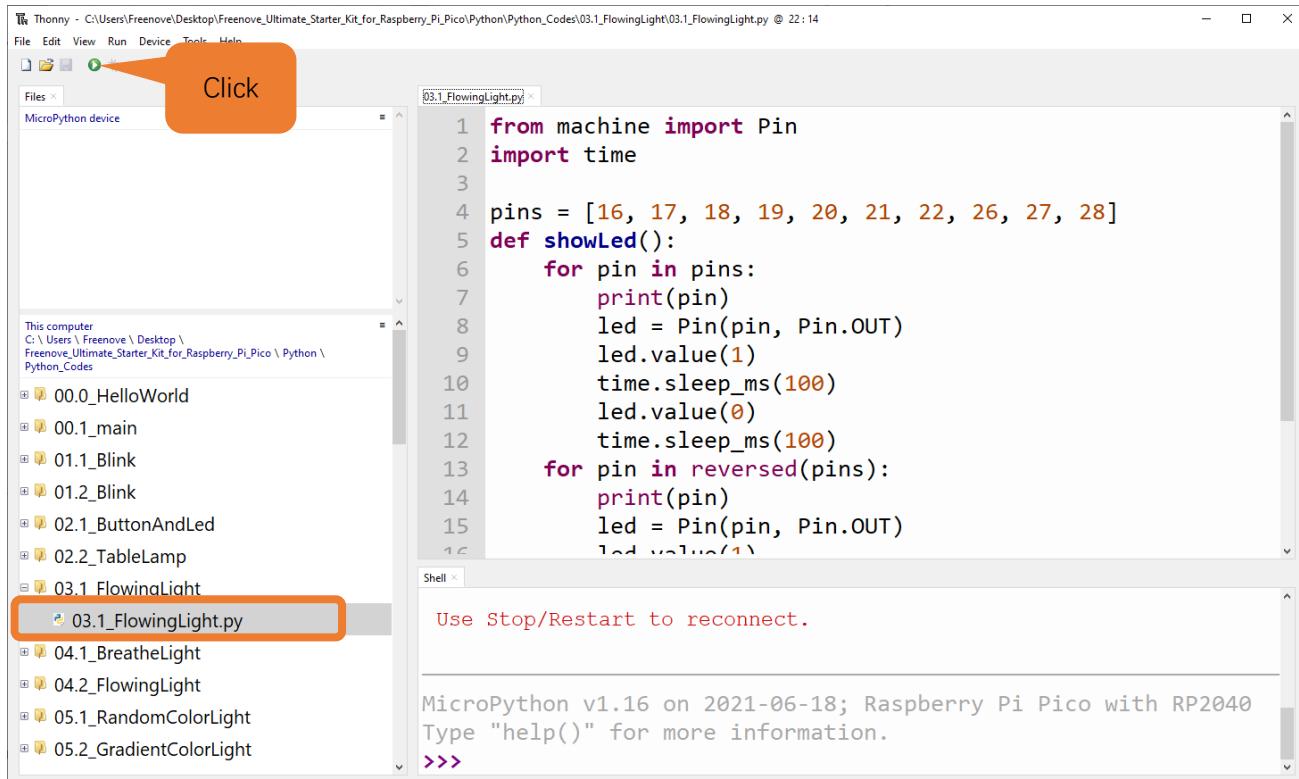
Code

This project is designed to make a flowing water lamp. Which are these actions: First turn LED #1 ON, and 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_Basic_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

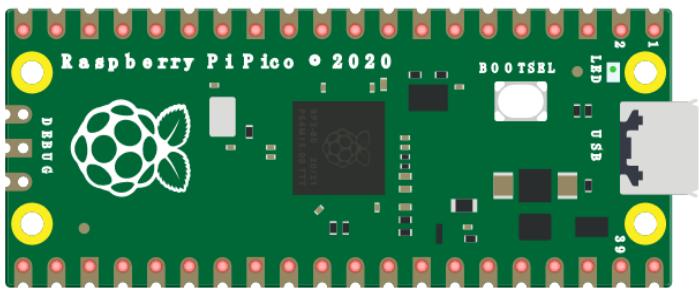
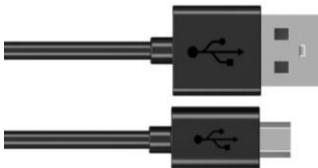
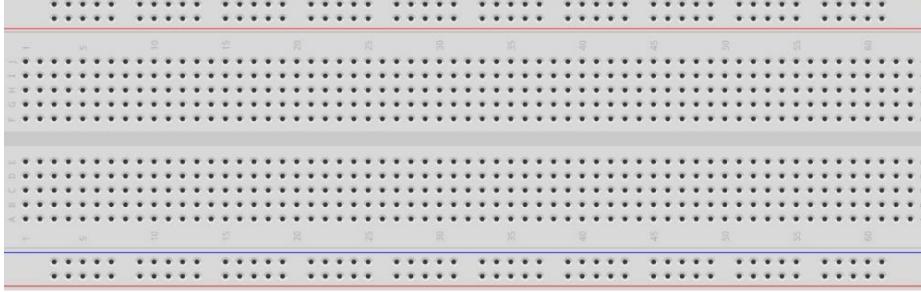
In previous study, we have known that one button has two states: pressed and released, and LED has light-on/off state, then how to enter a middle state? How to output an intermediate state to let LED "semi bright"? That is what we are going to learn.

First, let us 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.

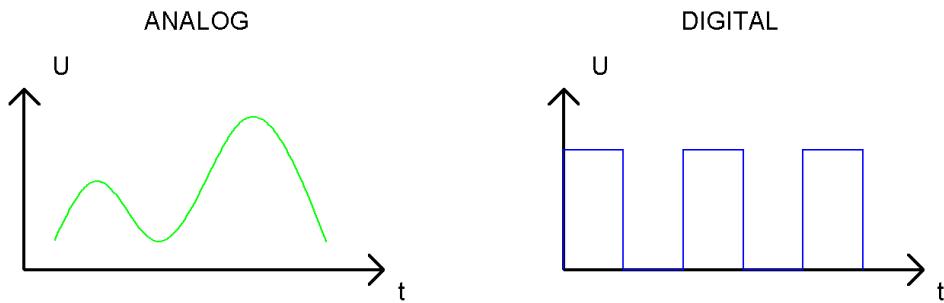
Component List

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.



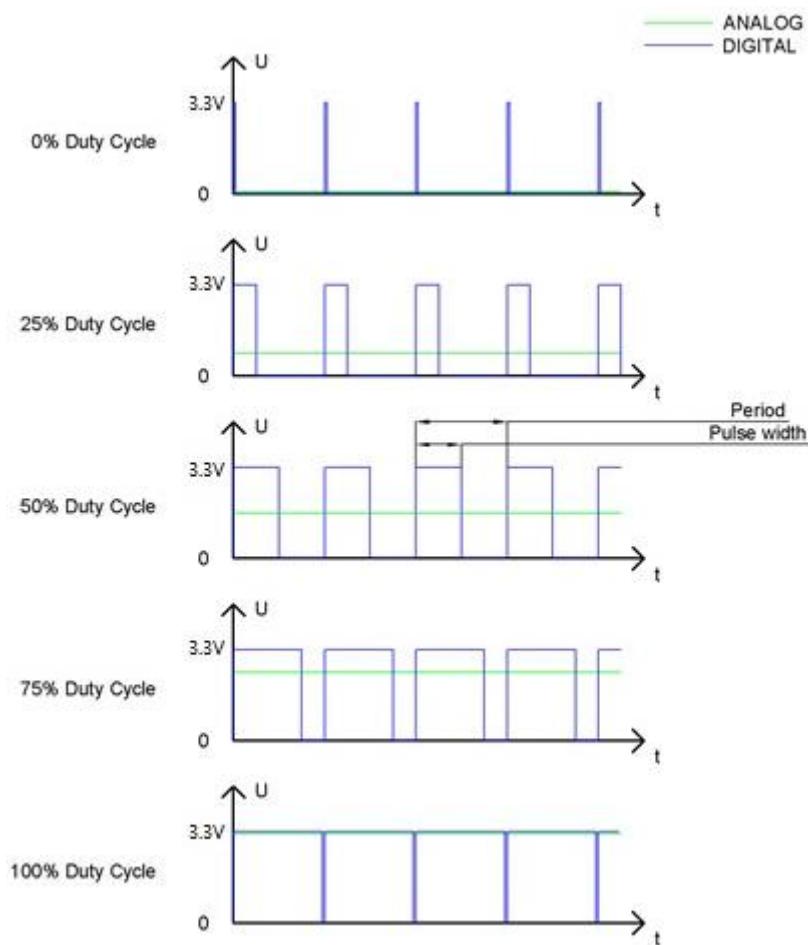
In practical application, we often use binary as the digital signal, which is a series of 0's and 1's. Since a binary signal only has two values (0 or 1), it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

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. Therefore, we can control the output power of the LED and other output modules to achieve different effects.

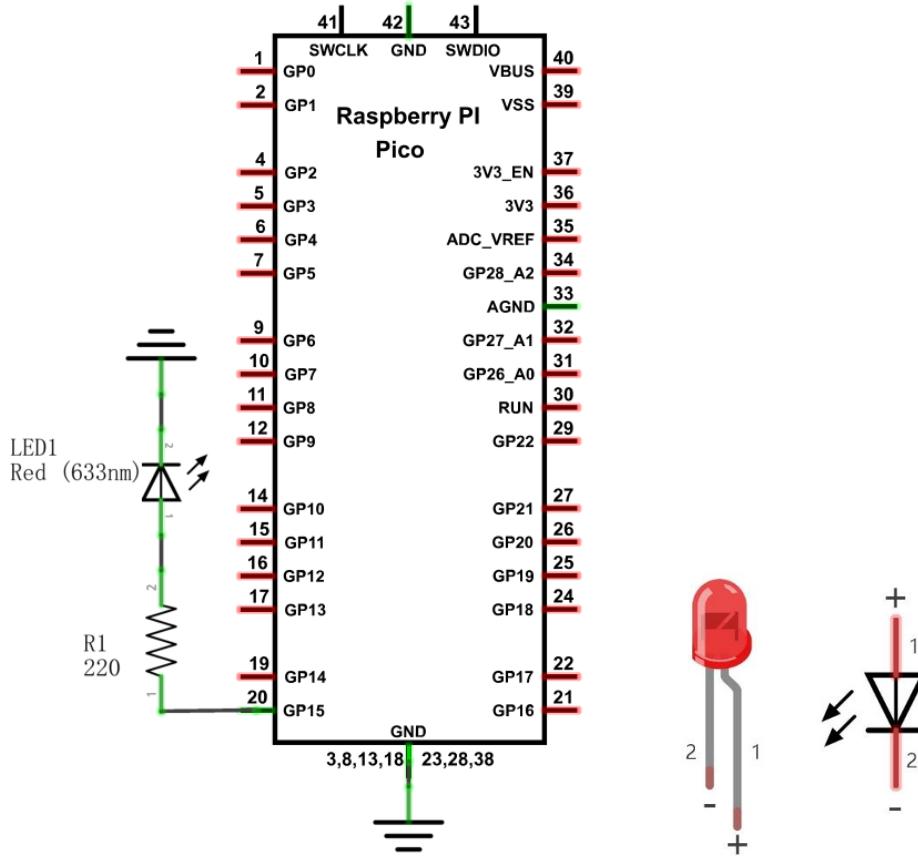
Raspberry Pi Pico and PWM

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.

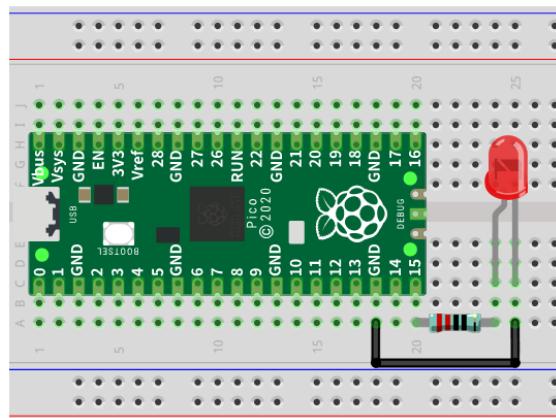
Circuit

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



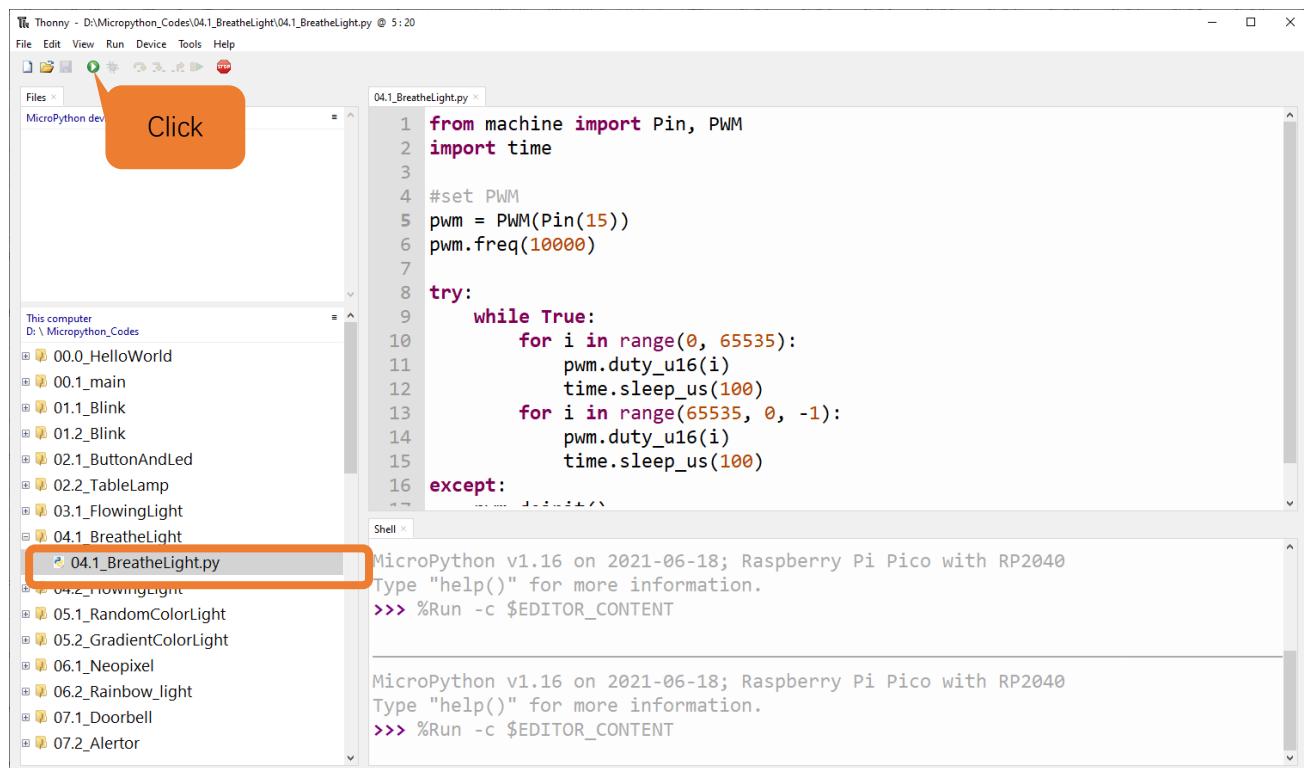
Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

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)

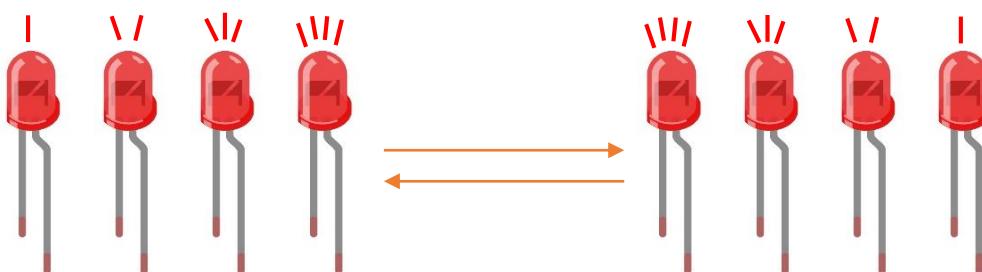
try:
    while True:
        for i in range(0, 65535):
            pwm.duty_u16(i)
            time.sleep_us(100)
        for i in range(65535, 0, -1):
            pwm.duty_u16(i)
            time.sleep_us(100)
except:
    pass

```

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 will 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.

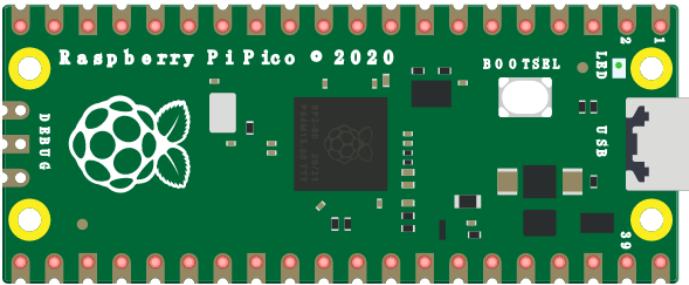
PWM.duty_u16(duty_val): the function is used to set PWM duty cycle, among which, duty_val ranges from 0 to 65535. If there is no parameter, the function returns to currently set duty cycle. If duty cycle has not yet been set, it returns 0.

PWM.deinit(): Turn OFF PWM.

Project 4.2 Meteor Flowing Light

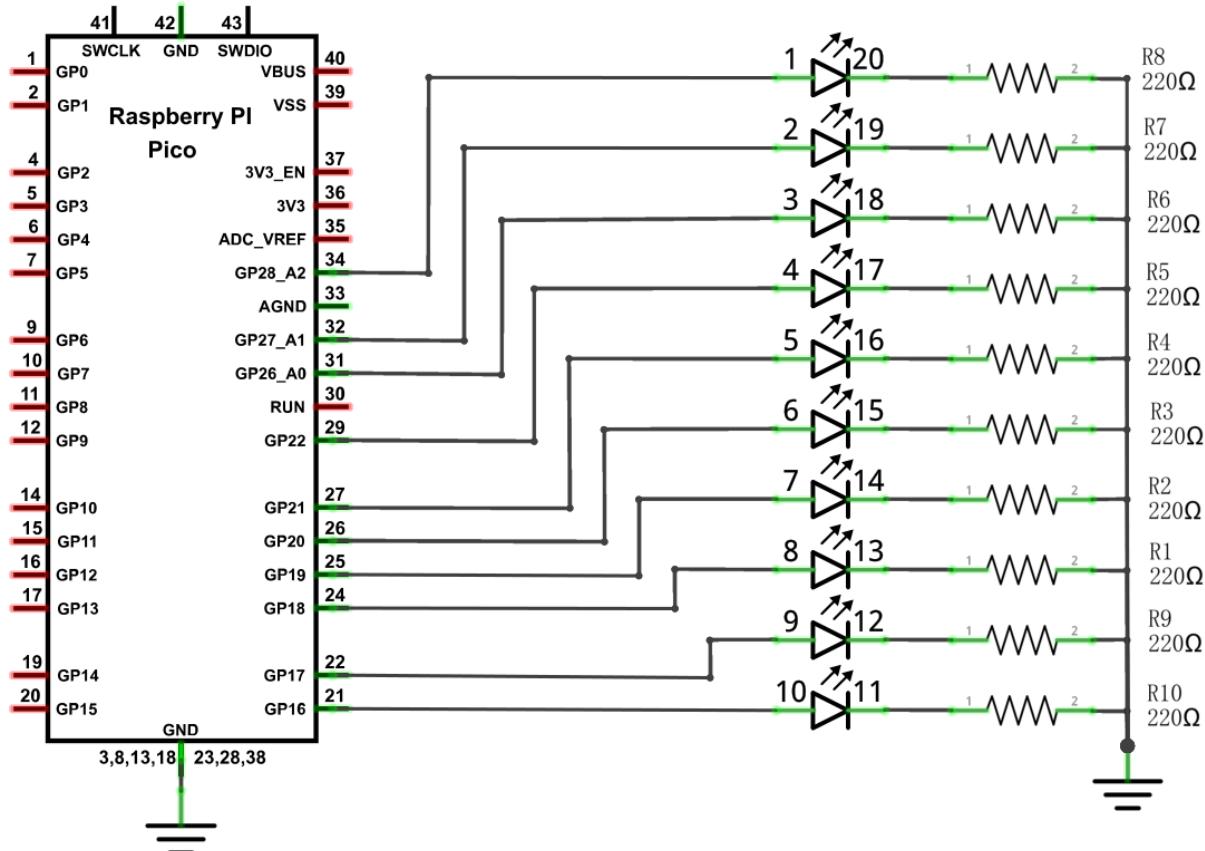
Having learned about PWM, we can use it to control LED Bar Graph and realize a cooler Flowing Light.

Component List

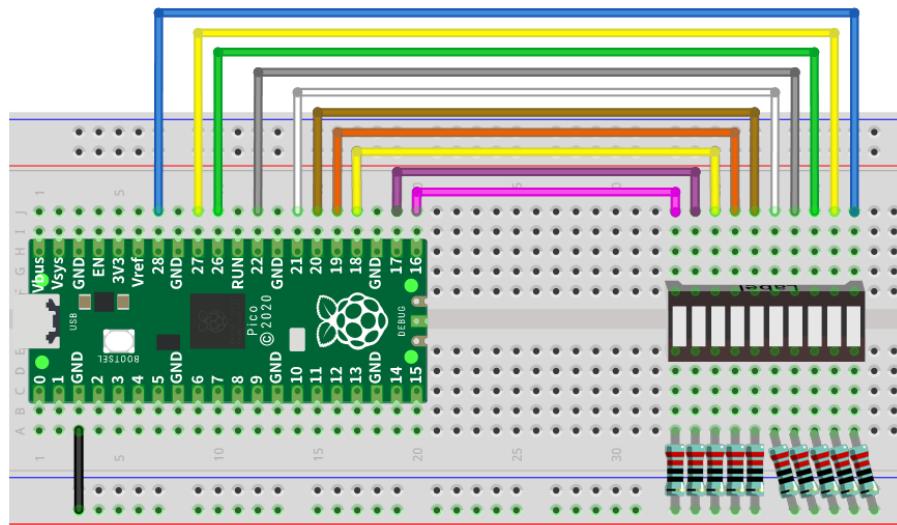
Raspberry Pi Pico x1	USB cable x1	
 A photograph of a Raspberry Pi Pico development board. It is a green printed circuit board with a central Broadcom SoC, labeled "Raspberry Pi Pico • 2020". It features several pins along the top edge and various components like resistors and capacitors. A small yellow circular component is visible on the left side.		
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 LED bar does not work, try rotating it for 180°. The label is random.

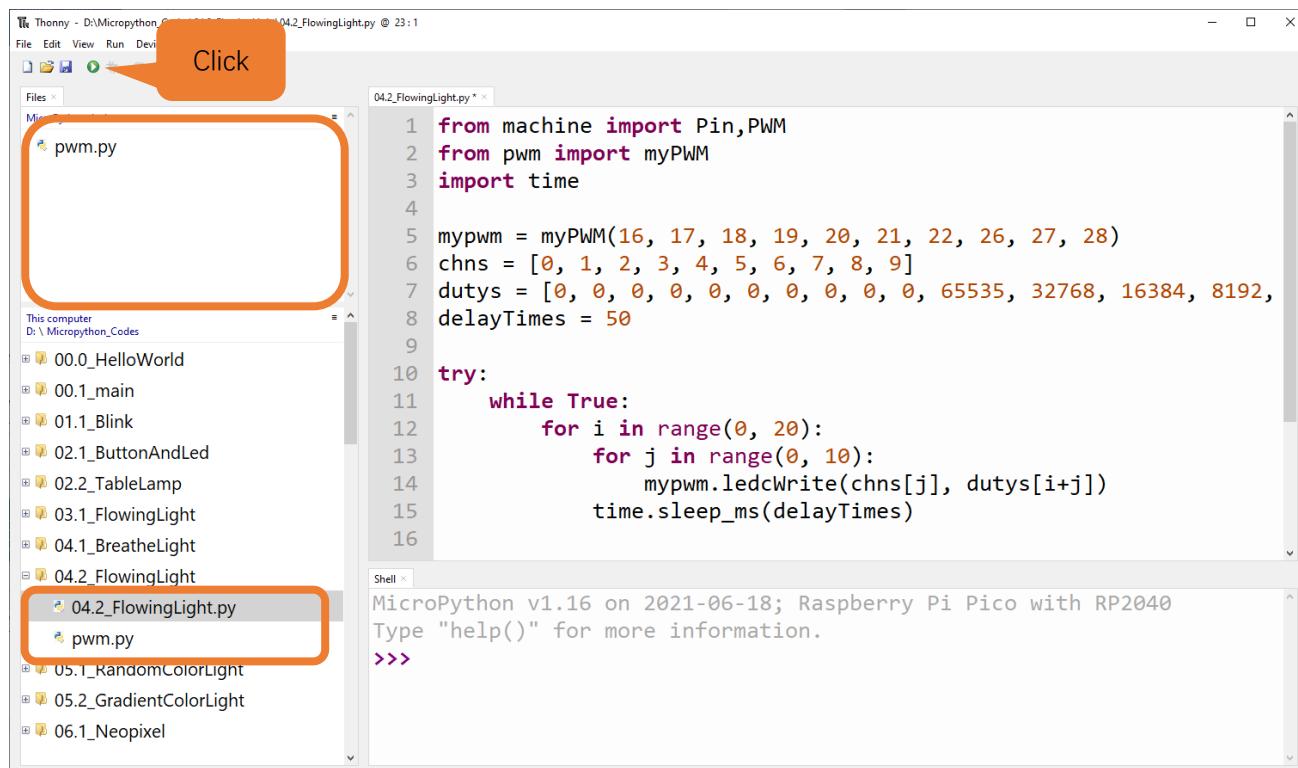
Any concerns? ✉ support@freenove.com

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/Rerun backend” to exit program.

The following is the program code:

```
1 from machine import Pin, PWM
2 from pwm import myPWM
3 import time
4
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, 0, 65535, 32768, 16384, 8192, 4096, 2048, 1024, 512, 256,
8 128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
9 delayTimes = 50
10
11 try:
12     while True:
13         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)
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.

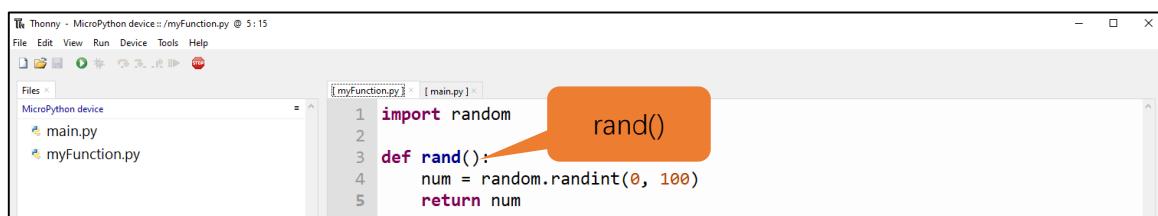
0	1	2	3	...	7	8	9	1	1	1	1	1	1	1	1	1	1	1	2	2	...	2	2	3
d	0	0	0	...	0	0	0	0	6	3	1	8	4	2	1	5	2	1	0	0	0	0	0	0
i									5	2	6	1	0	0	0	1	5	2						
									5	7	3	9	9	4	2	2	6	8						
									3	6	8	2	6	8	4									
									5	8	4													
0																								
1																								
...								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

How to import a custom python module

Each Python file, as long as it is 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.

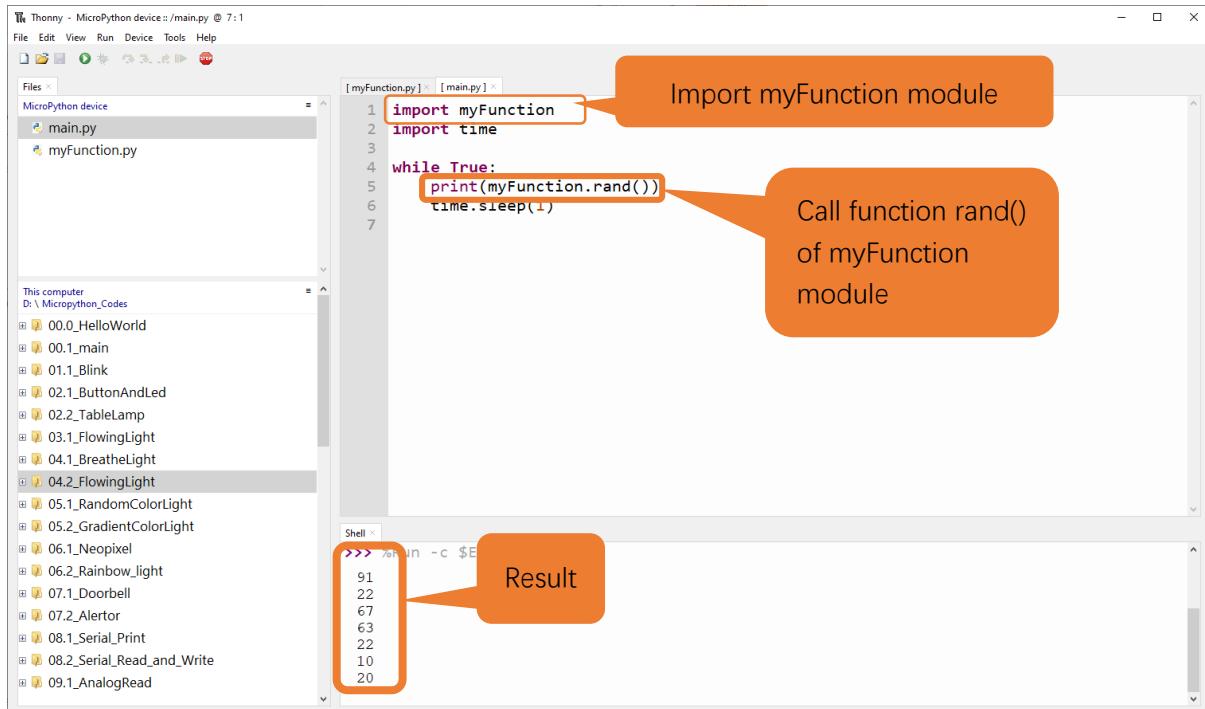
Code

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())
```

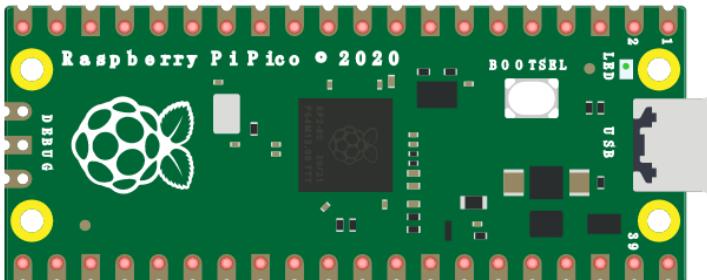
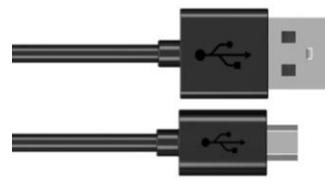
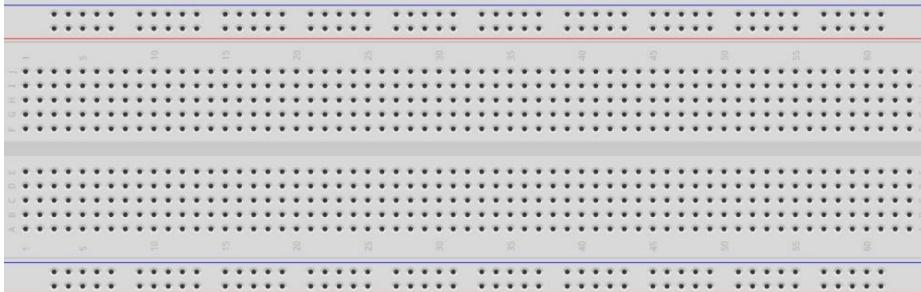
Chapter 5 RGBLED

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

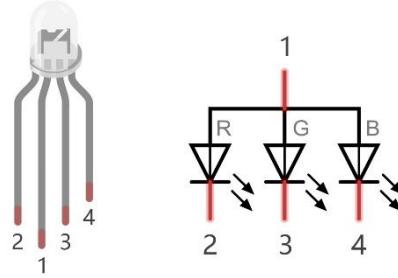
In this project, we will make a multicolored LED and control it to switch different colors automatically.

Component List

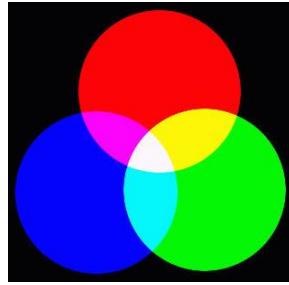
Raspberry Pi Pico x1	USB cable x1	
		
Breadboard x1		
		
RGBLED x1	Resistor 220Ω x3	Jumper
		

Related knowledge

RGB LED has integrated three LEDs that can respectively emit red, green and blue light, and it has four 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 three 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.

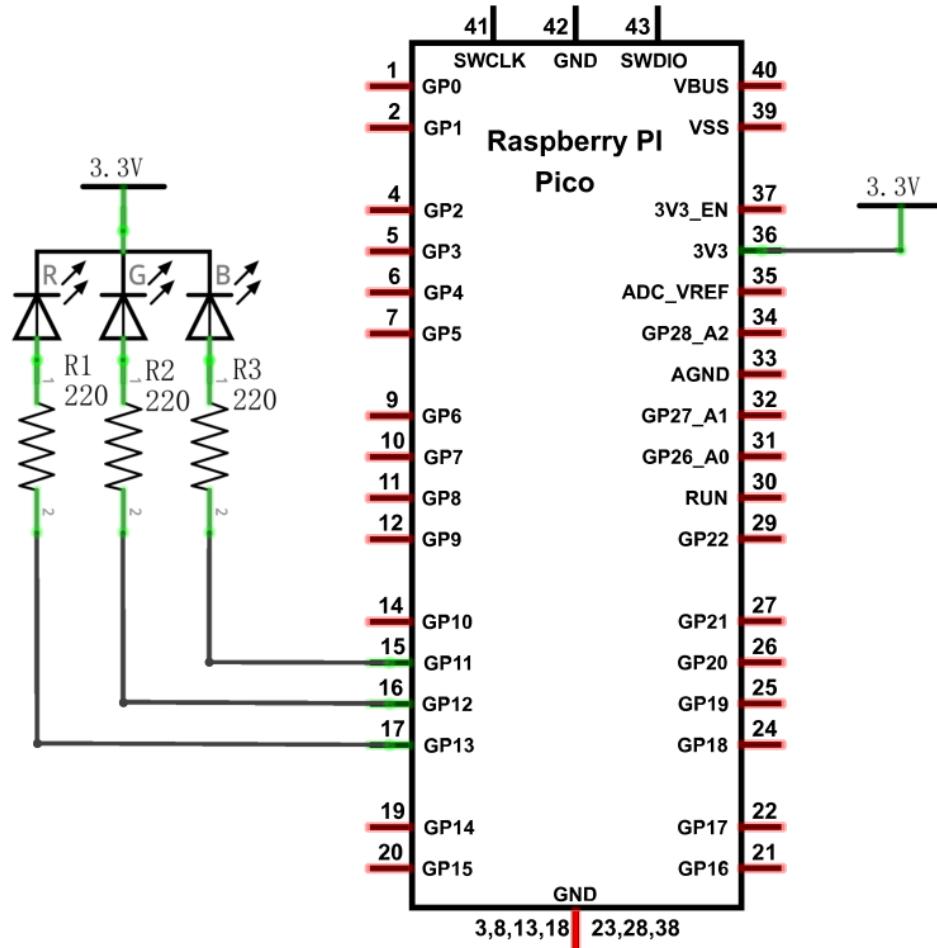


RGB

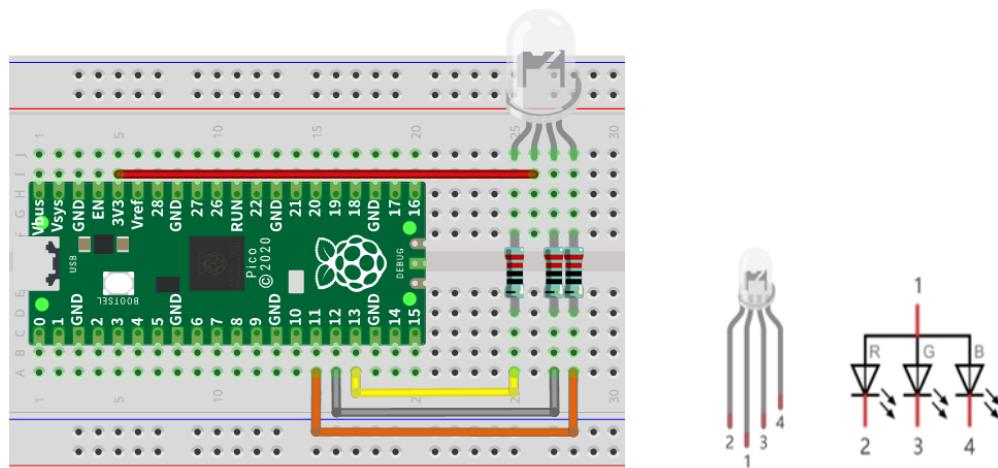
If we use three 10-bit PWM to control the RGBLED, in theory, we can create $2^{10} * 2^{10} * 2^{10} = 1,073,741,824$ (1 billion) colors through different combinations.

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.

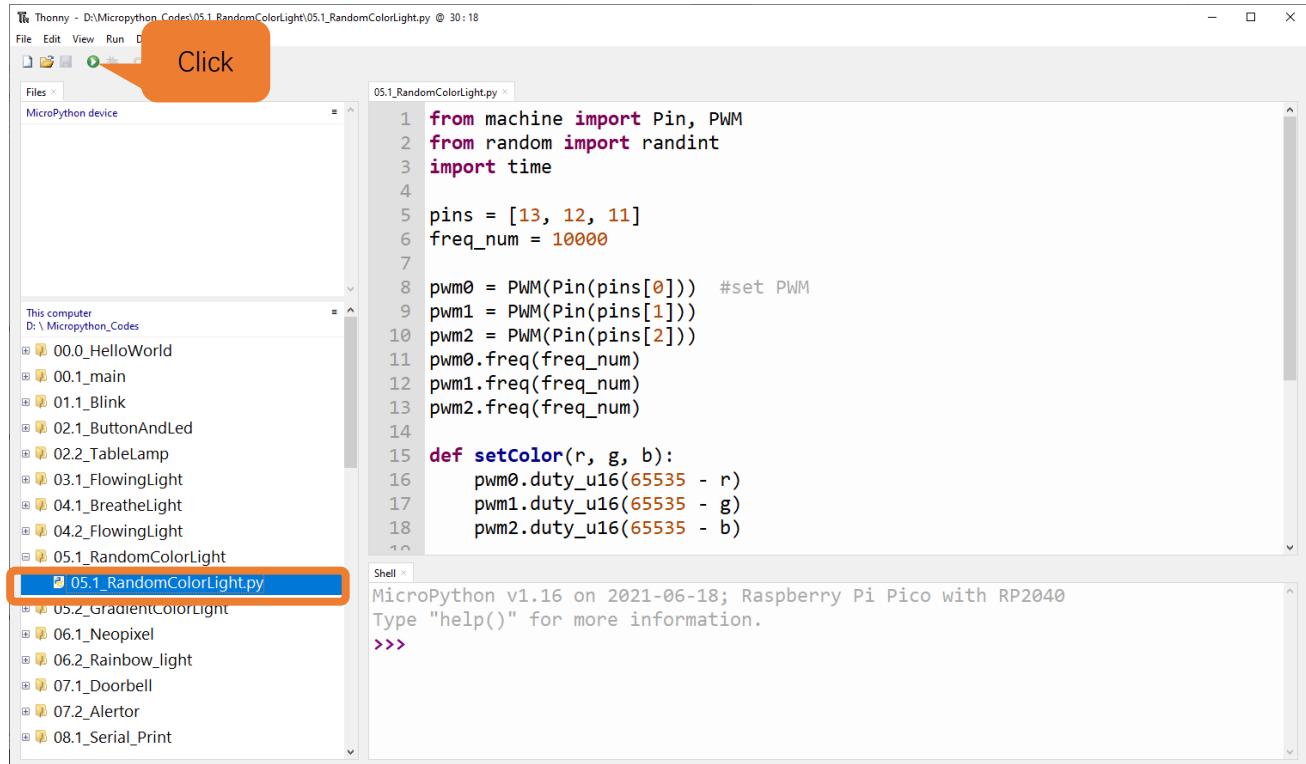
Any concerns? support@freenove.com

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 0b1110.

size = 8, it generates an integer in the range of 0 to 0b11111110.

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.

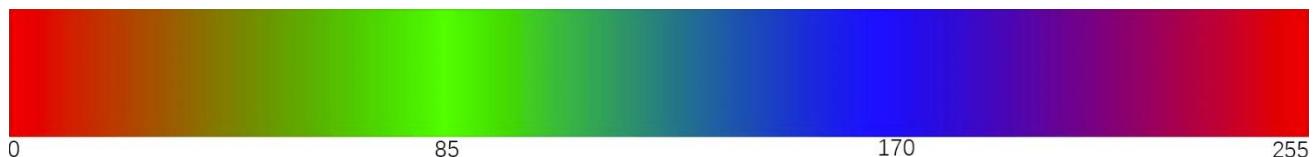
obj: list of elements.

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 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))

```

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

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 two 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 three 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))
```



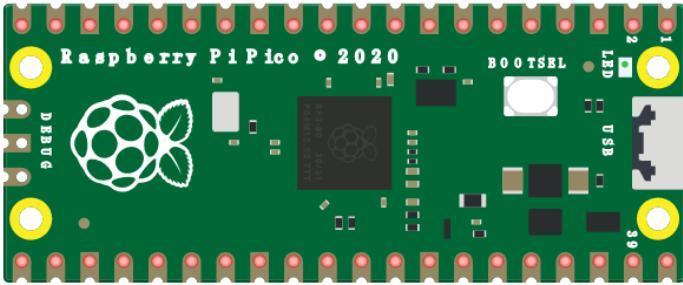
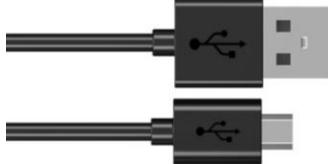
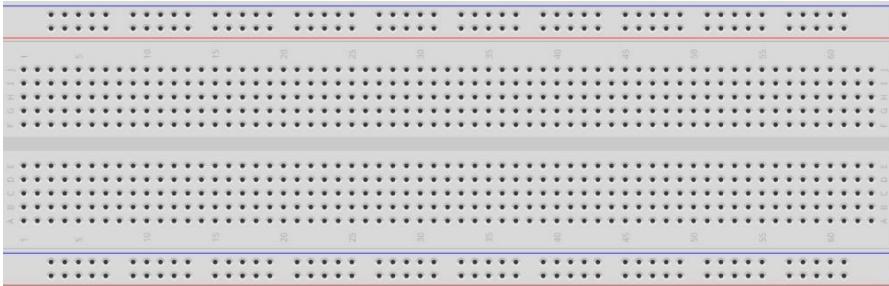
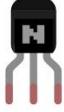
Chapter 6 Buzzer

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

Project 6.1 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

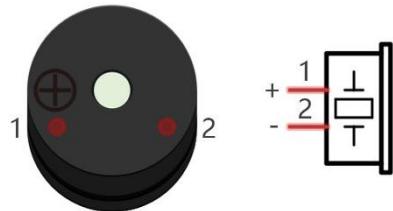
Raspberry Pi Pico x1		USB cable x1							
Breadboard x1									
Jumper									
NPN transistor x1 (S8050)		Active buzzer x1		Push button x1		Resistor 1kΩ x1		Resistor 10kΩ x2	

Component knowledge

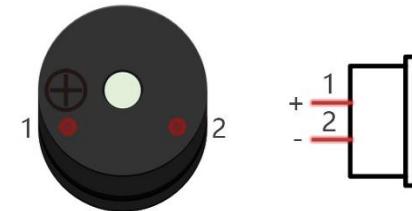
Buzzer

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 2 kHz, which means the passive buzzer is loudest when its resonant frequency is 2 kHz.

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

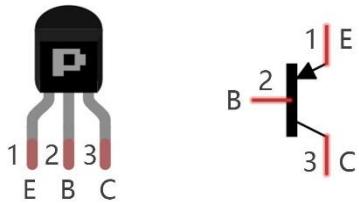
Because the buzzer requires such large current that GP of Raspberry Pi Pico output capability cannot meet the requirement, a transistor of NPN type is needed here to amplify the current.

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

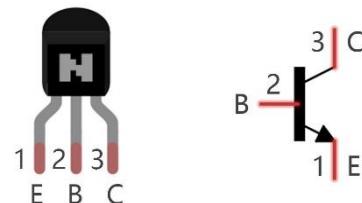


can be used to amplify weak signal, or works as a switch. It has three electrodes(PINs): base (b), collector (c) and emitter (e). When there is current passing between "be", "ce" will allow several-fold current (transistor magnification) pass, at this point, transistor works in the amplifying area. When 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

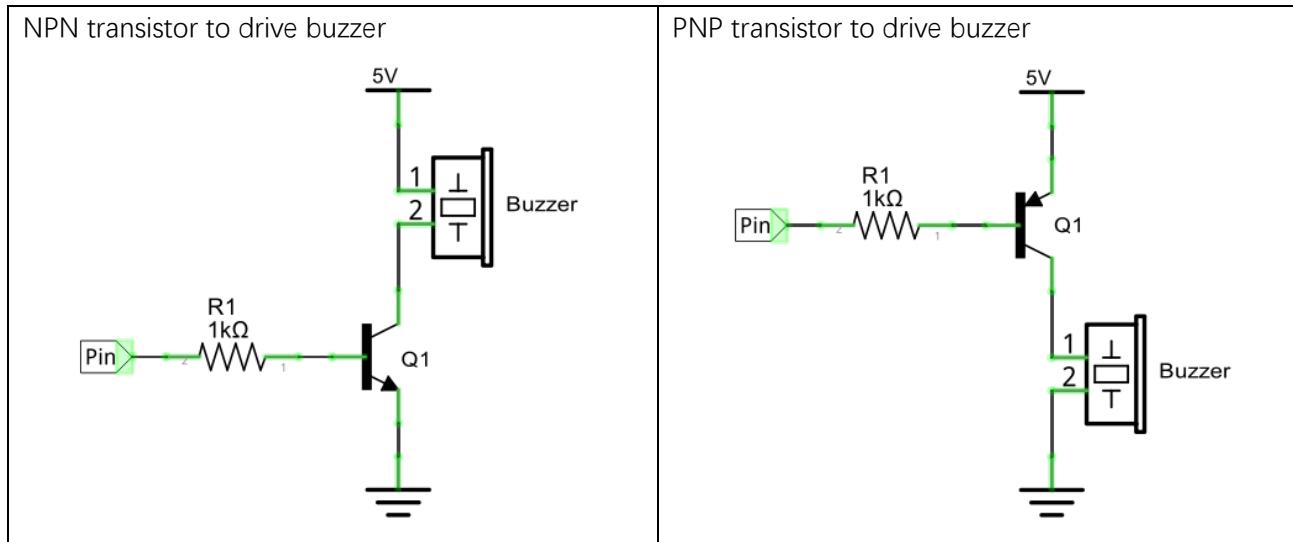


In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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.

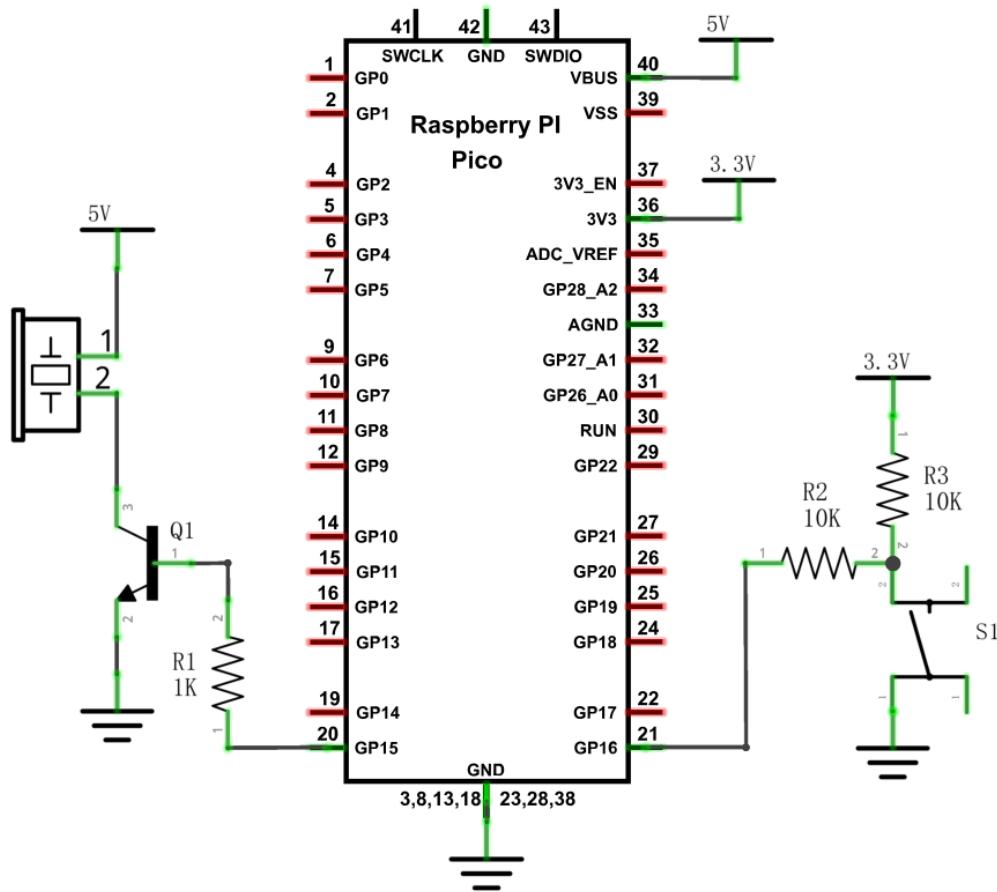
When using NPN transistor to drive buzzer, we often adopt the following method. If GP outputs high level, current will flow through R1, the transistor will be conducted, and the buzzer will sound. If GP outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer will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

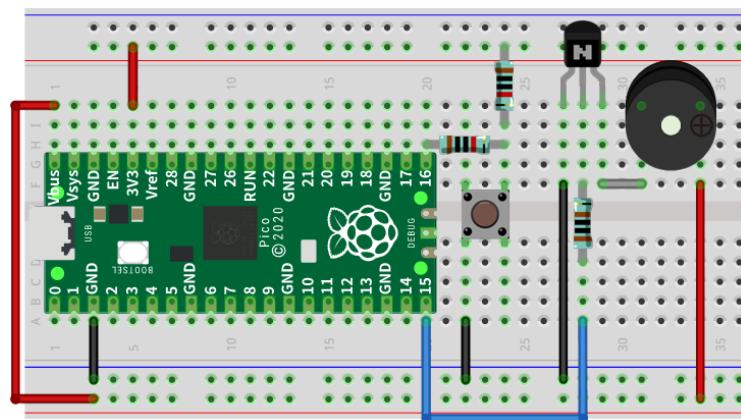


Circuit

Schematic diagram



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



Note:

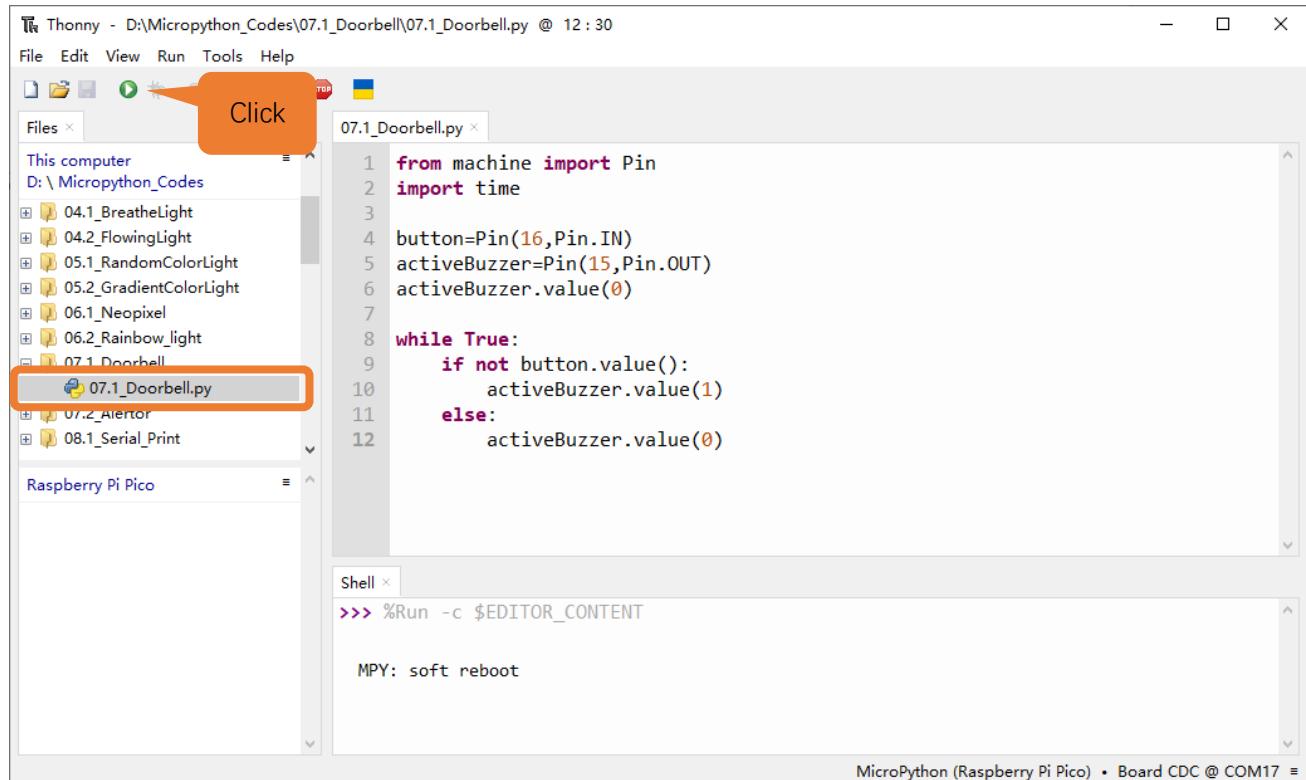
1. In this circuit, the power supply for buzzer is 5V, and pull-up resistor of the button connected to the power 3.3V. The buzzer can work when connected to power 3.3V, but it will reduce the loudness.
2. VBUS should be connect to the positive end of USB cable. If it connects to GND, it may burn the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

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” → “06.1_Doorbell” and double click “06.1_Doorbell.py”.

06.1_Doorbell



The screenshot shows the Thonny IDE interface. The title bar says "Thonny - D:\Micropython_Codes\07.1_Doorbell\07.1_Doorbell.py @ 12 : 30". The menu bar includes File, Edit, View, Run, Tools, and Help. The toolbar has icons for file operations and a play button. The left sidebar shows a tree view of files under "This computer" and "Raspberry Pi Pico". A file named "07.1_Doorbell.py" is selected and highlighted with an orange box. An orange arrow points from the text "Click" to this highlighted file. The main editor window displays the following Python code:

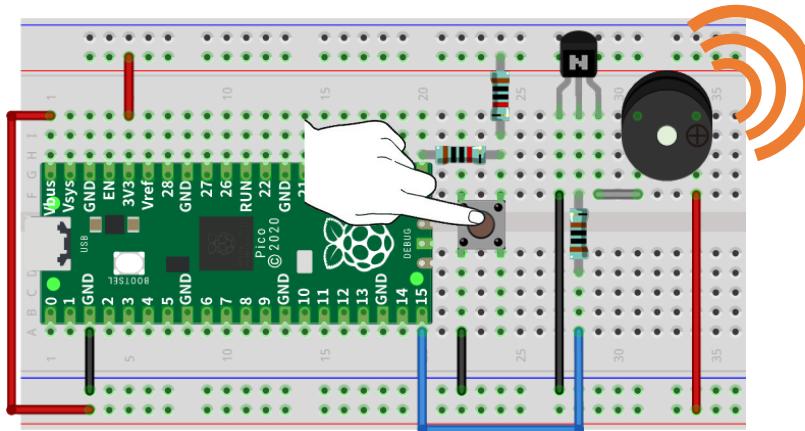
```
from machine import Pin
import time

button=Pin(16,Pin.IN)
activeBuzzer=Pin(15,Pin.OUT)
activeBuzzer.value(0)

while True:
    if not button.value():
        activeBuzzer.value(1)
    else:
        activeBuzzer.value(0)
```

The bottom shell window shows the command ">>> %Run -c \$EDITOR_CONTENT" and the response "MPY: soft reboot". The status bar at the bottom right indicates "MicroPython (Raspberry Pi Pico) • Board CDC @ COM17".

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/Restart 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)
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 6.2 Alertor

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

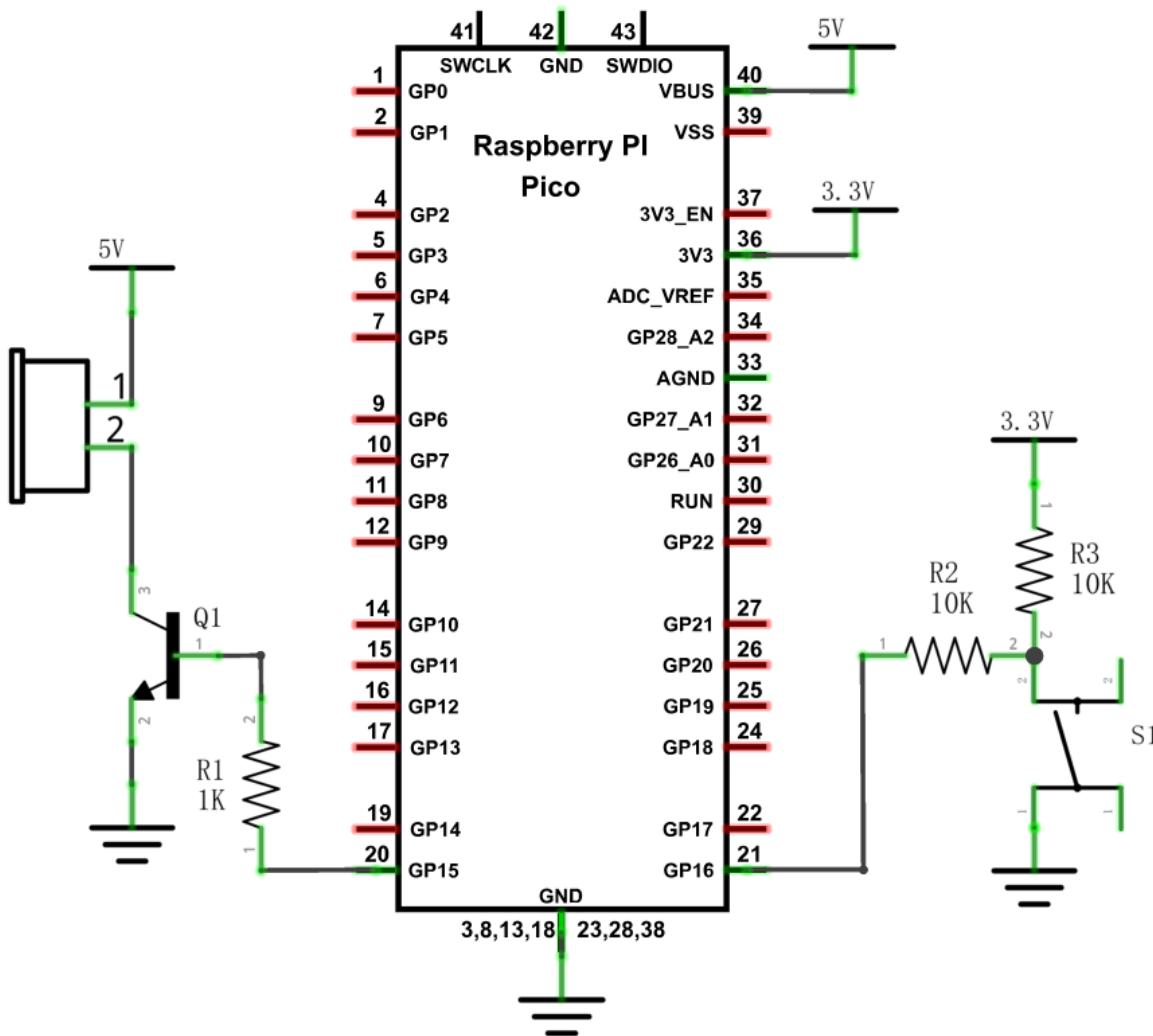
Component list and the circuit part is similar to last section. In the Doorbell circuit, only the **active buzzer** needs to be **replaced** with a **passive buzzer**.

Component List

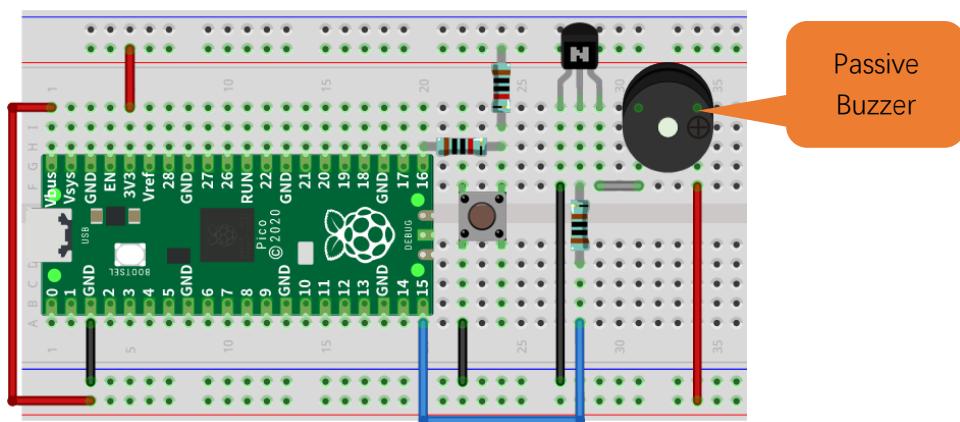
Raspberry Pi Pico x1	USB cable x1			
A green printed circuit board with a central Broadcom SoC, labeled "Raspberry Pi Pico • 2020". It has a USB Type-C port, a microSD slot, and various pins along the edges.	Two standard black USB cables, each with a black USB-A connector at one end and a grey USB-B connector at the other.			
Breadboard x1	A grey breadboard with two rows of 40 numbered holes (1 through 40) for each row, labeled A through H on both sides.			
Jumper	A long, thin black wire with small red caps at each end, used for connecting components.			
NPN transistor x1 (S8050)	Passive buzzer x1	Push button x1	Resistor 1kΩ x1	Resistor 10kΩ x2
A small black plastic package with three pins, representing an NPN transistor.	A circular black component with a central green button, used for sound generation.	A small rectangular component with two red legs, used for manual control.	A cylindrical resistor with a brown band, labeled 1kΩ.	A cylindrical resistor with a brown band, labeled 10kΩ.

Circuit

Schematic diagram



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

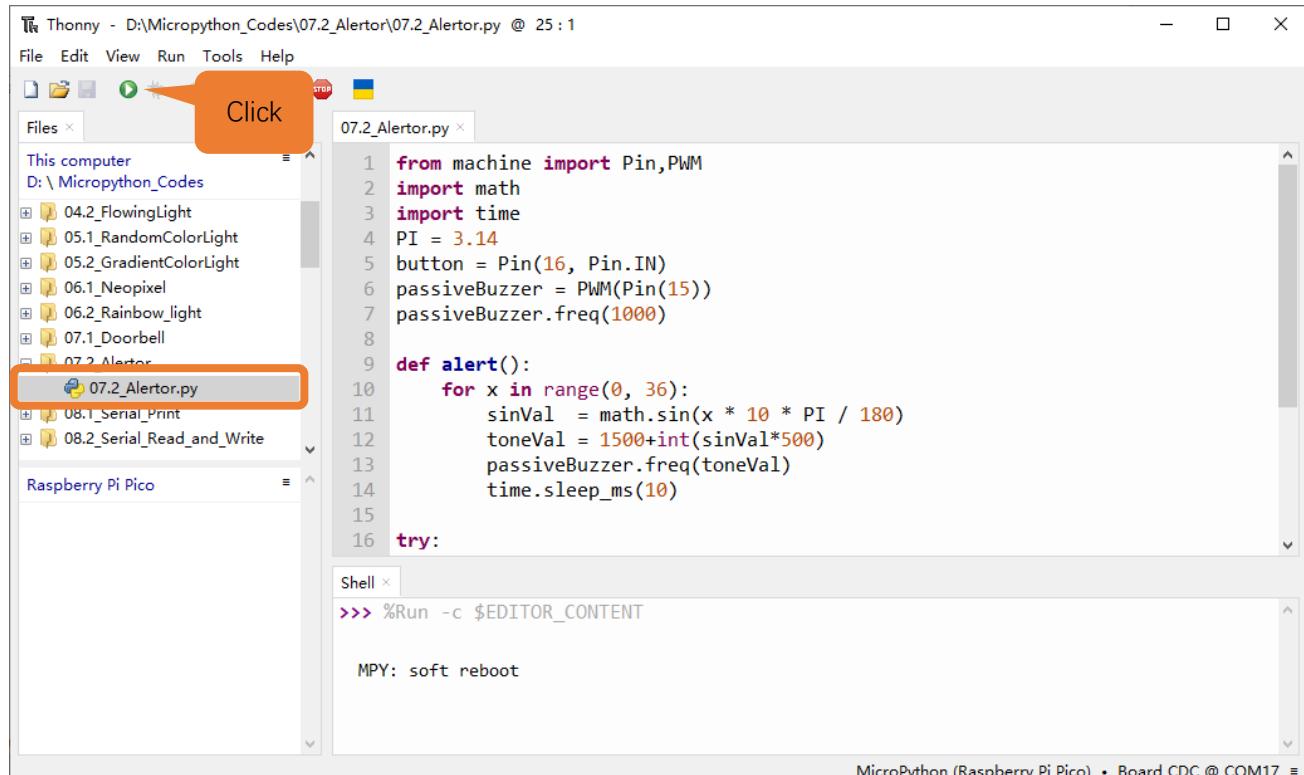


Code

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” → “06.2_Alertor”, and double click “06.2_Alertor.py”.

06.2_Alertor



```

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

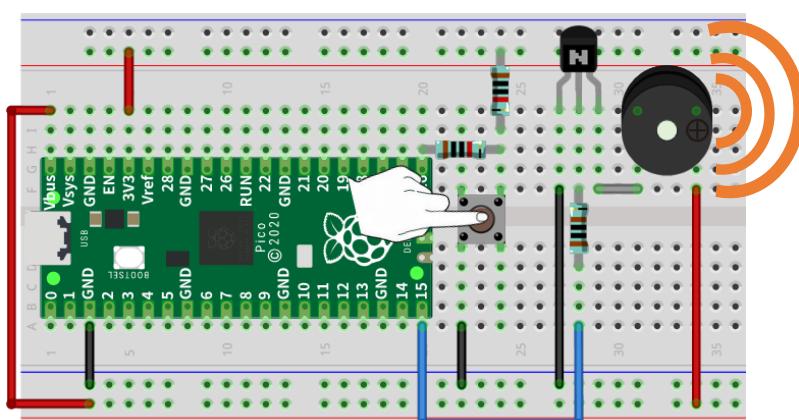
def alert():
    for x in range(0, 36):
        sinVal = math.sin(x * 10 * PI / 180)
        toneVal = 1500+int(sinVal*500)
        passiveBuzzer.freq(toneVal)
        time.sleep_ms(10)

try:
    while True:
        if button.value() == 0:
            alert()
        else:
            break
except KeyboardInterrupt:
    print("MPY: soft reboot")

```

Click “Run current script”, press the button, and the alarm sounds; 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)
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()
```



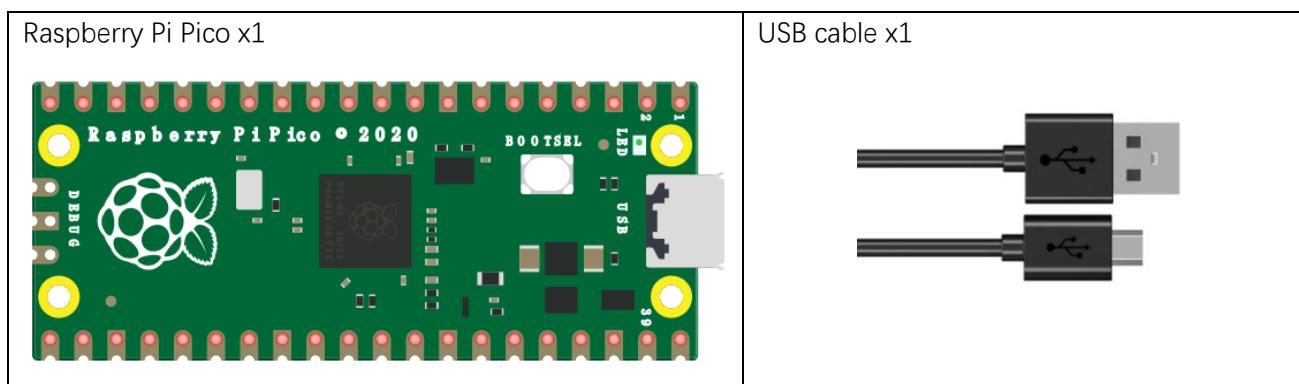
Chapter 7 Serial Communication

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

Project 7.1 Serial Print

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

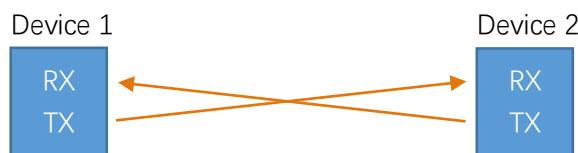
Component List



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.

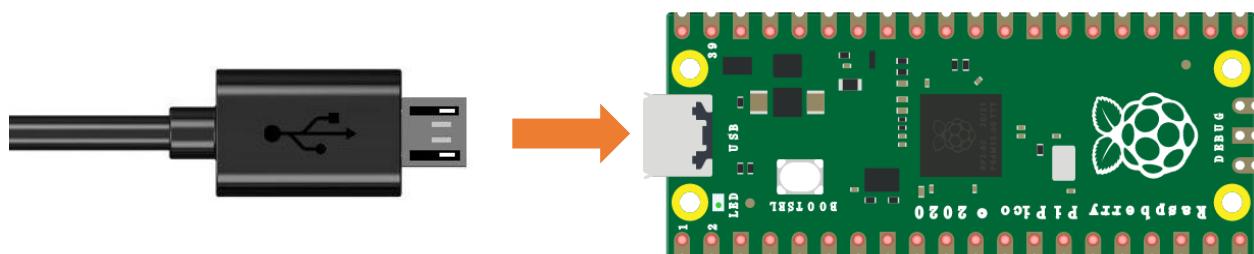
Serial port on Raspberry Pi Pico

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



Circuit

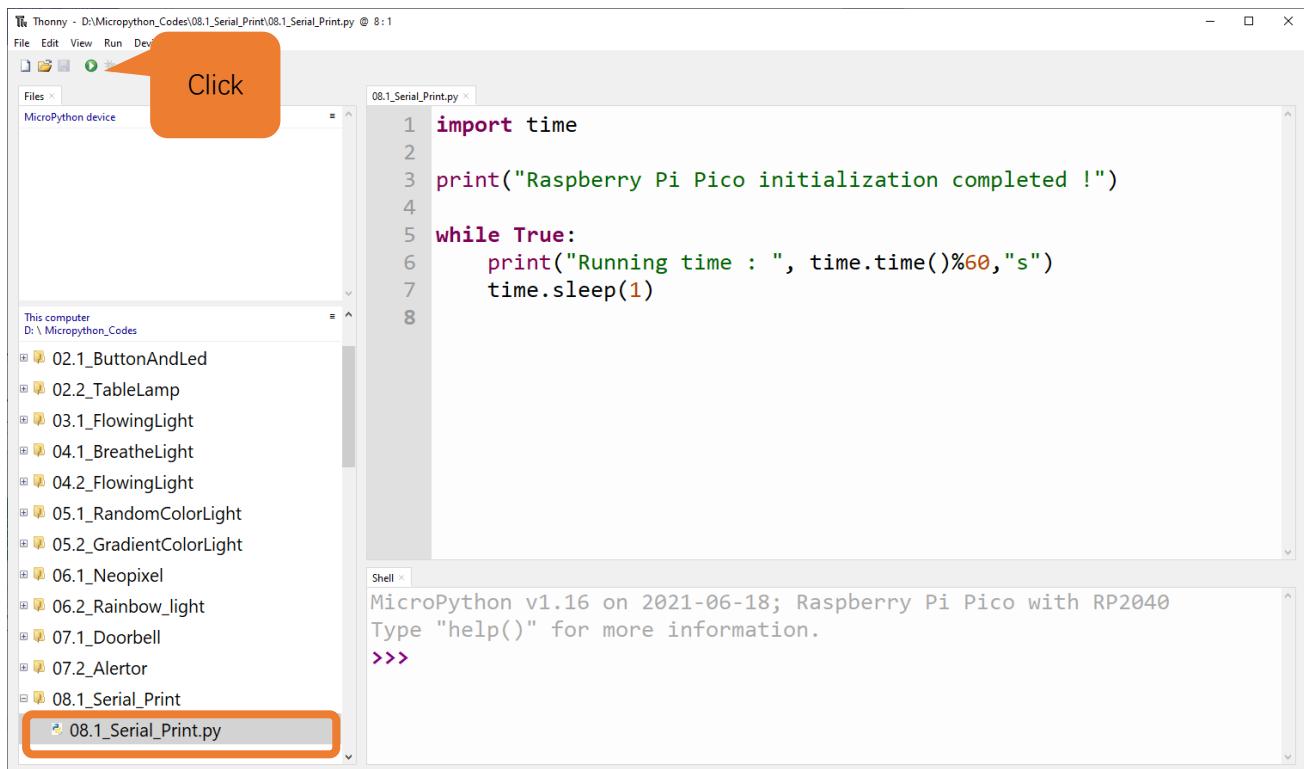
Connect Raspberry Pi Pico to the computer with USB cable.



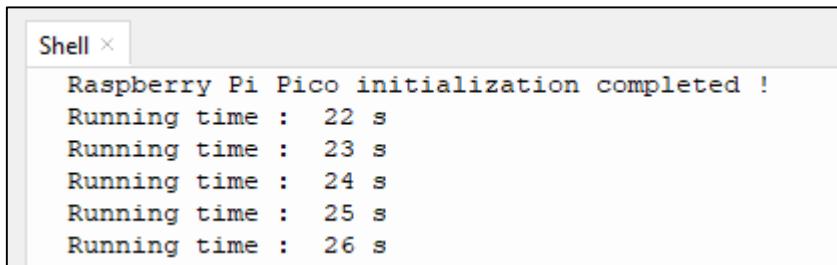
Code

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

07.1_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.

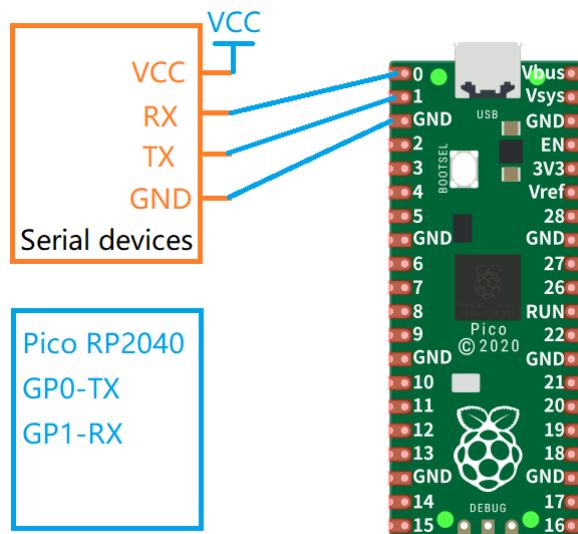


The following is the program code:

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.

You can use them to communicate with serial devices.



Default pin for UART0

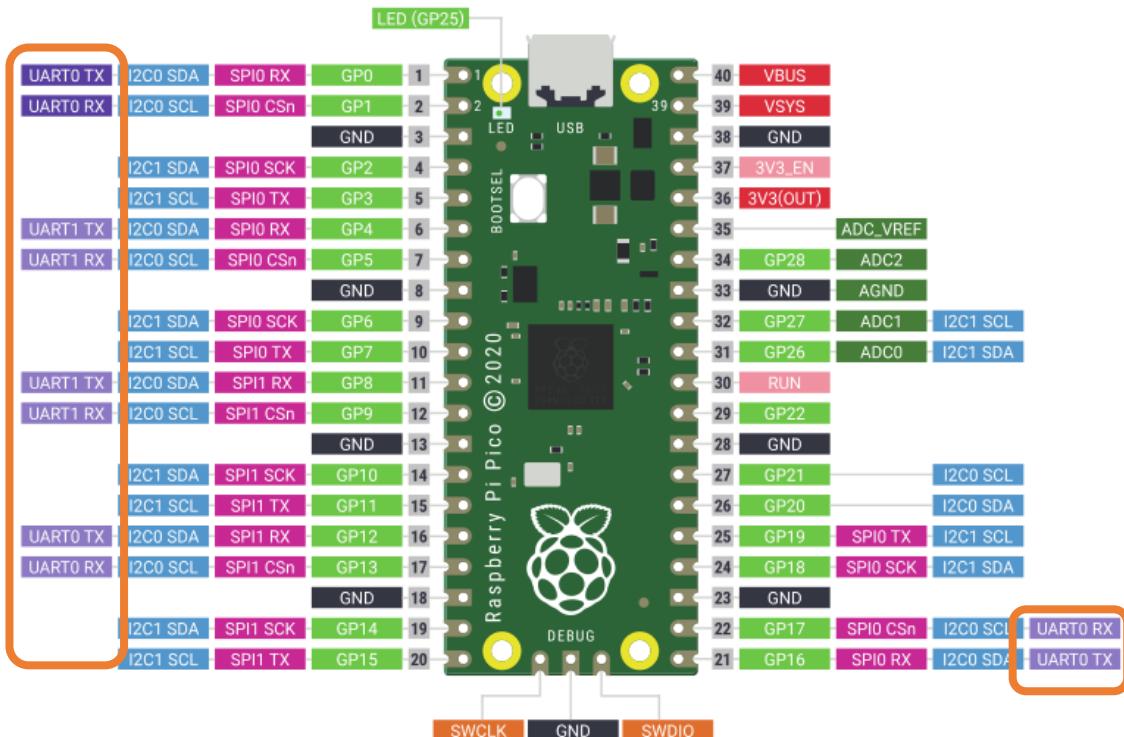
UART0_TX	Pin 0
UART0_RX	Pin 1

Default pin for UART1

UART1_TX	Pin 4
UART1_RX	Pin 5

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

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.

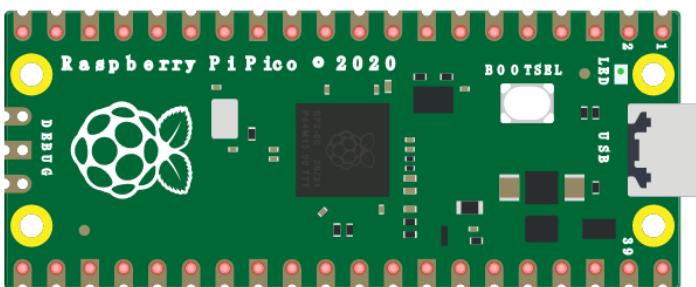
Note: When usART0 or USART1 is used, it must be used with the serial port adapter board or serial port device. If not, you may not observe any symptoms.

Project 7.2 Serial Read and Write

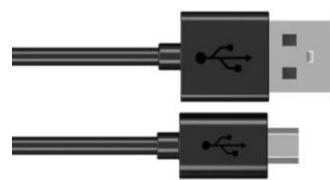
In the following example, we use Raspberry Pi Pico's UART1 to send data to UART0, and read the data received by UART0 and print it out through the "Shell".

Component List

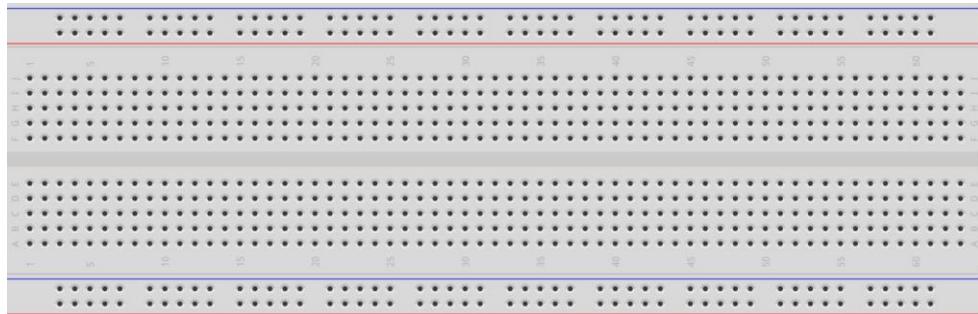
Raspberry Pi Pico x1



USB cable x1



Breadboard x1

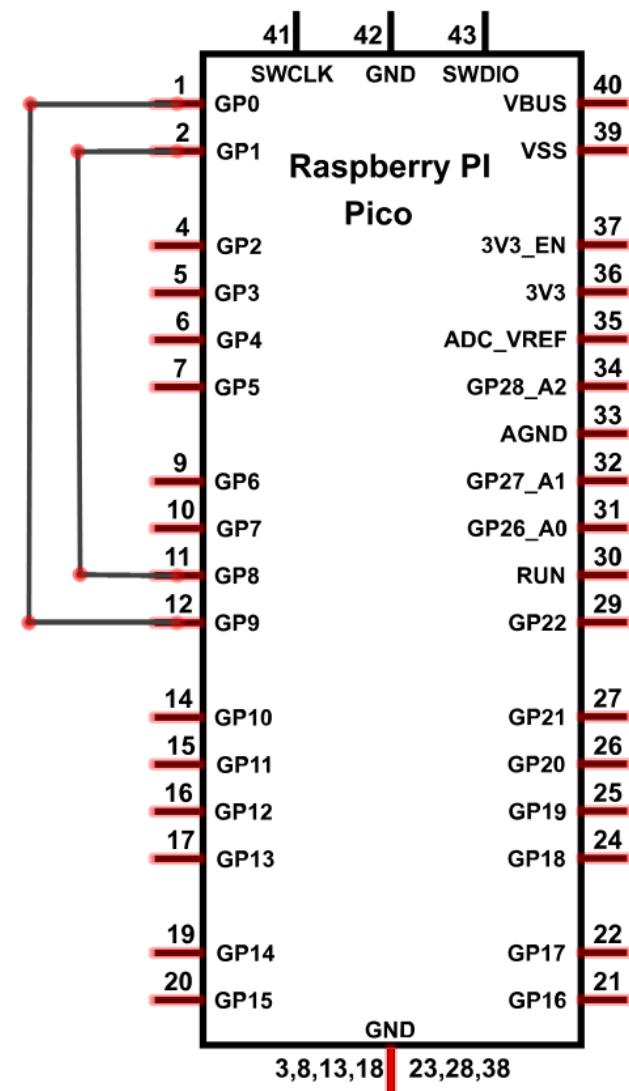


Jumper

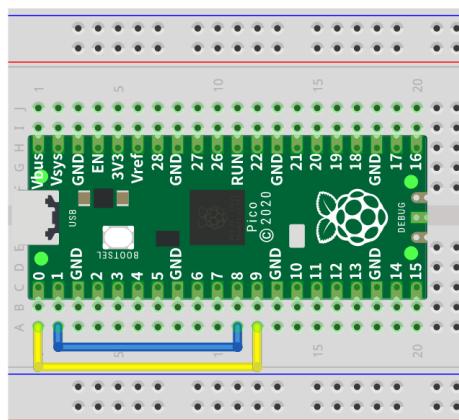


Circuit

Schematic diagram



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



Any concerns? support@freenove.com

Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “07.2_Serial_Read_and_Write” and double click “07.2_Serial_Read_and_Write_UART1_to_UART0.py”.

07.2_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". Two files are highlighted with orange boxes: "08.2_Serial_Read_and_Write.py" and "08.2_Serial_Read_and_Write_Uart0_to_Uart1.py". An orange callout bubble labeled "Click" points to the "Run" button in the toolbar at the top of the editor window. The main editor area contains the Python code for serial communication between UART1 and UART0. The code uses the `UART` class from the `machine` module to initialize both ports at 9600 baud rate, 8 bits, and no parity. It then enters a loop where it reads data from UART0 and prints it to the shell, while also writing user input to UART1.

```

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'))

```

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. It displays the MicroPython version and a prompt. The user has entered several lines of data into the shell, which is highlighted by an orange callout bubble labeled "Data that users input.". The shell then outputs the same data, preceded by "myUsart1:", which is highlighted by another orange callout bubble labeled "Data read by UART0".

```

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

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:

```

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'))
```

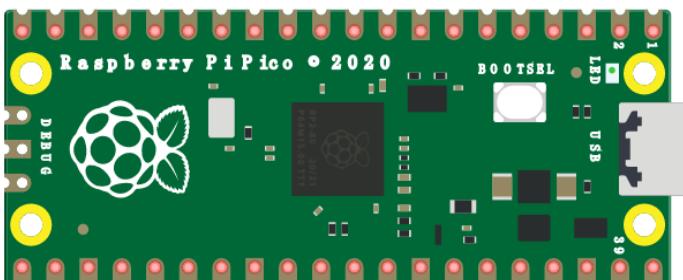
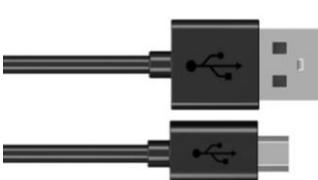
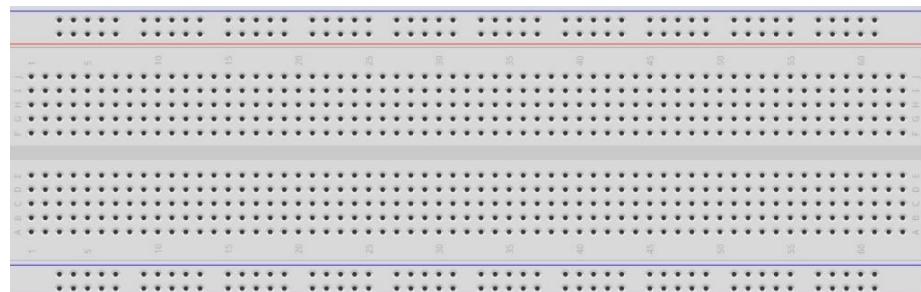
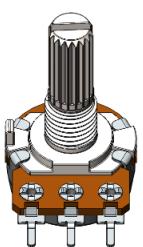
Chapter 8 AD Converter

In this chapter we learn to use the ADC function of Raspberry Pi Pico.

Project 8.1 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 photograph of a green Raspberry Pi Pico development board. It features a central Broadcom SoC, several pins along the edges, and various component markings like 'Raspberry Pi Pico • 2020' and 'BOOTSEL'.	USB cable x1	 A diagram showing two standard USB-A male connectors connected by a cable.
Breadboard x1			 A photograph of a grey breadboard with multiple horizontal and vertical rows of pins for connecting components.
Rotary potentiometer x1	 A photograph of a brown three-terminal rotary potentiometer with a silver cylindrical knob.	Jumper	 A photograph of a long black jumper wire with two small black caps at the ends.

Related knowledge

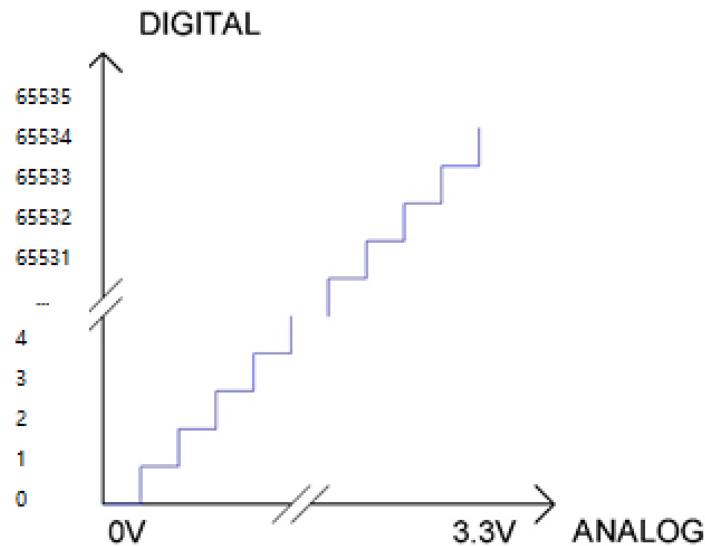
ADC

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 changed to 0-65535, to make it the same as the ADC of other Micropython microcontrollers. The way that ADC converts does not change but only the resolution changes. Therefore, if the measured analog voltage ranges from 0-3.3V, ADC can divide 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

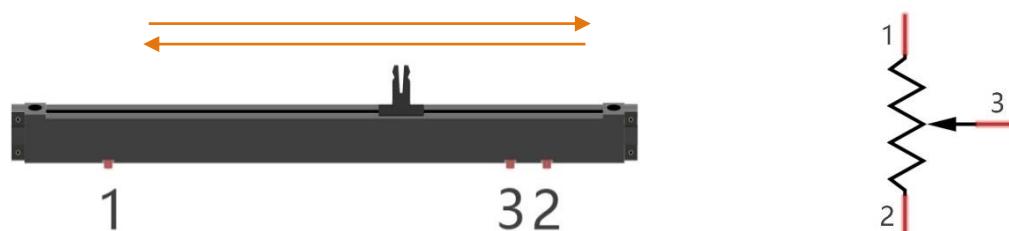
Raspberry Pi Pico has five ADC channels, which are ADC0(GP26), ADC1(GP27), ADC2(GP28), ADC3(GP29): used to measure VSYS on Pico board, and ADC4, which directly connects to the built-in temperature sensor of RP2040 chip. Therefore, there are only three generic ADC channels that can be directly used, namely,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

Any concerns? ✉ support@freenove.com

Component knowledge

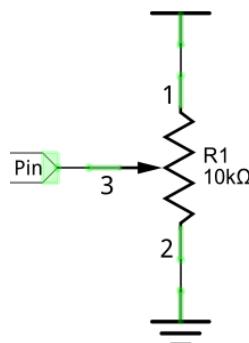
Potentiometer

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.



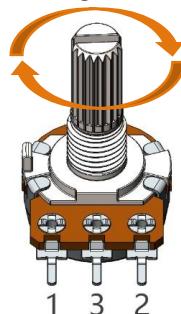
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly (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.



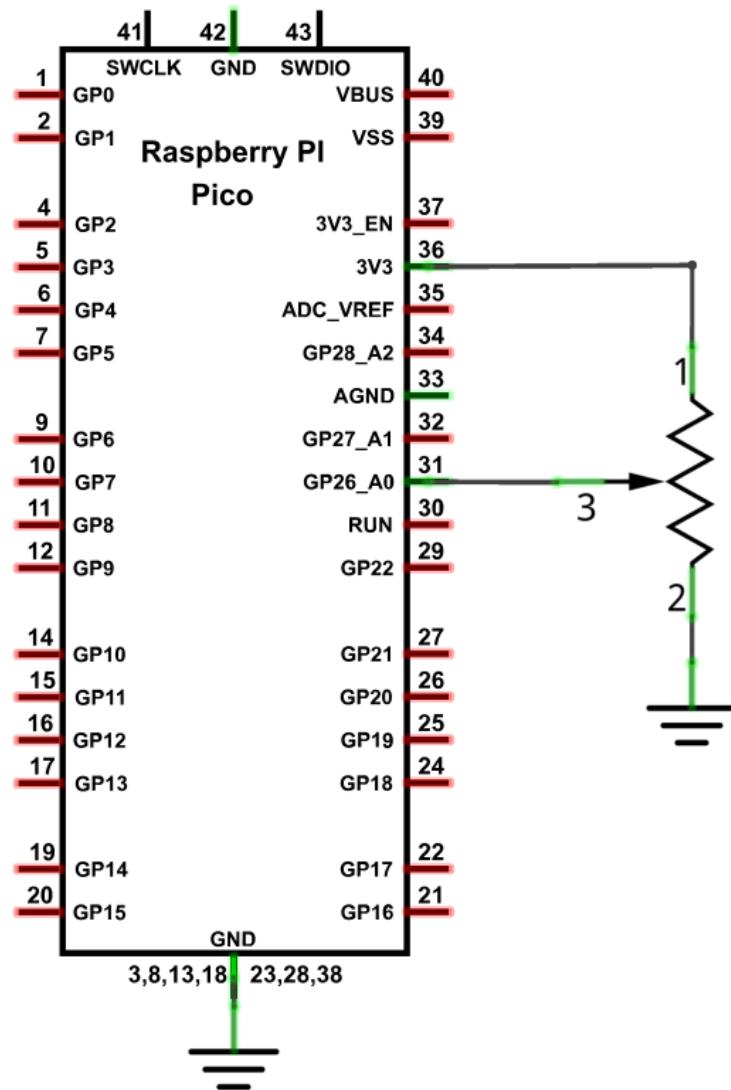
Rotary potentiometer

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

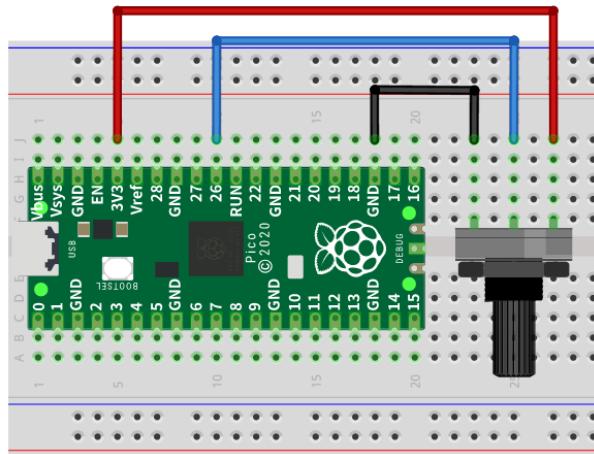


Circuit

Schematic diagram



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

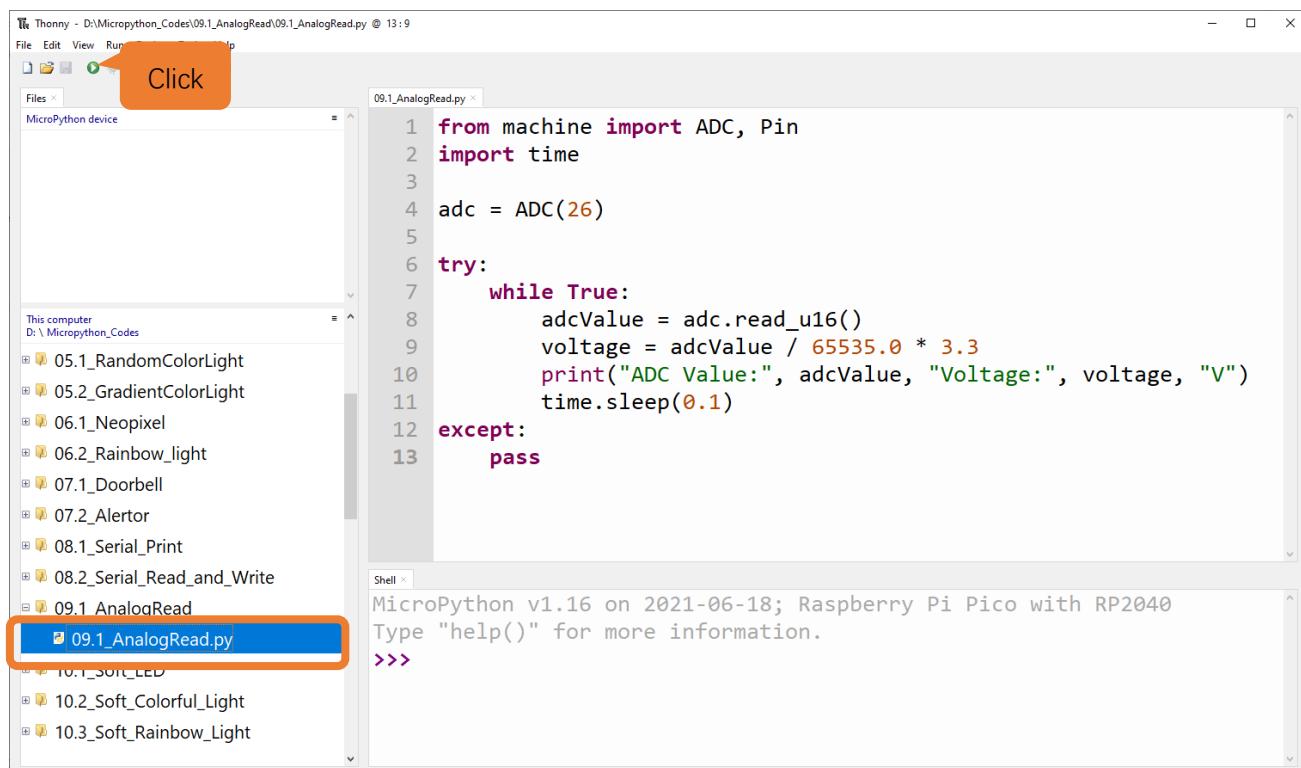


Any concerns? ✉ support@freenove.com

Code

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

08.1_AnalogRead



```

from machine import ADC, Pin
import time

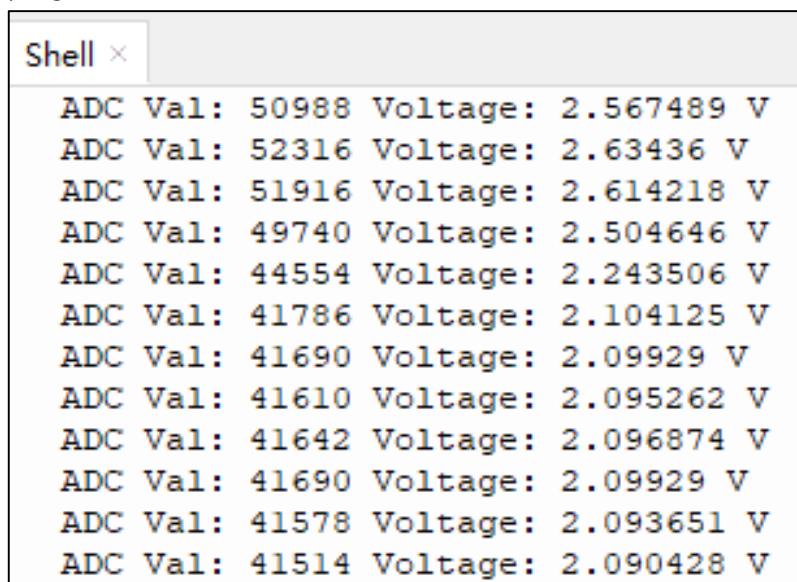
adc = ADC(26)

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
52316	2.63436
51916	2.614218
49740	2.504646
44554	2.243506
41786	2.104125
41690	2.09929
41610	2.095262
41642	2.096874
41690	2.09929
41578	2.093651
41514	2.090428



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,GP12.

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.

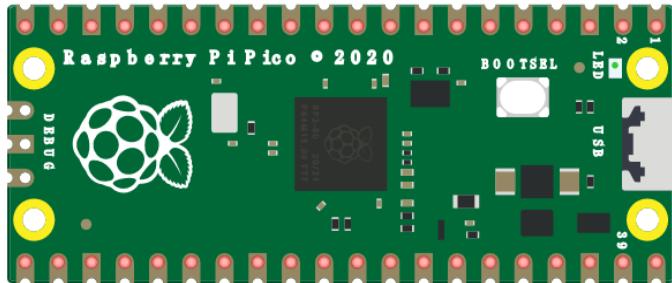
Chapter 9 Potentiometer & LED

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 9.1 Soft Light

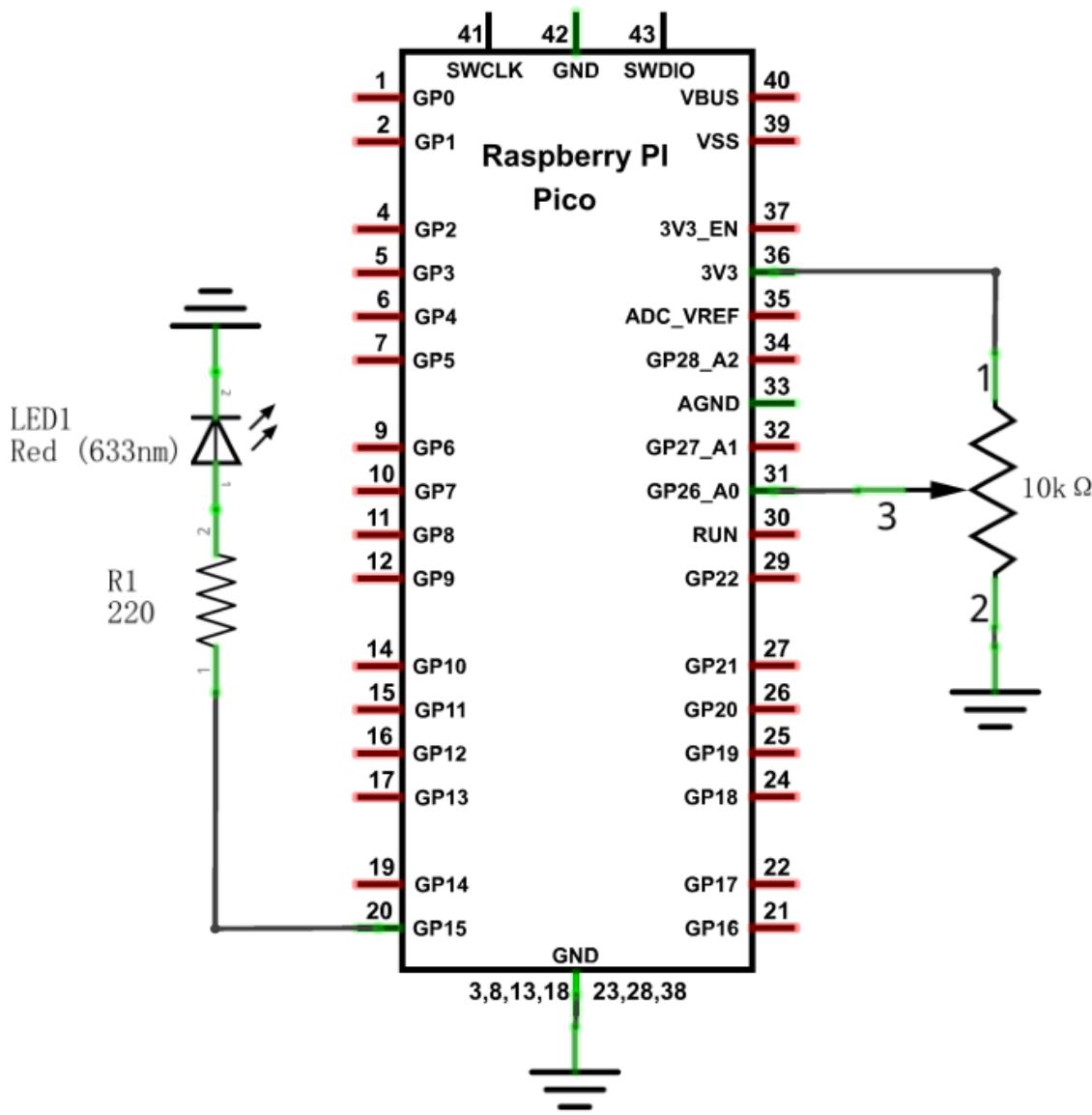
In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle of the PWM used to control the brightness of an LED. Then you can change the brightness of an LED by adjusting the potentiometer.

Component List

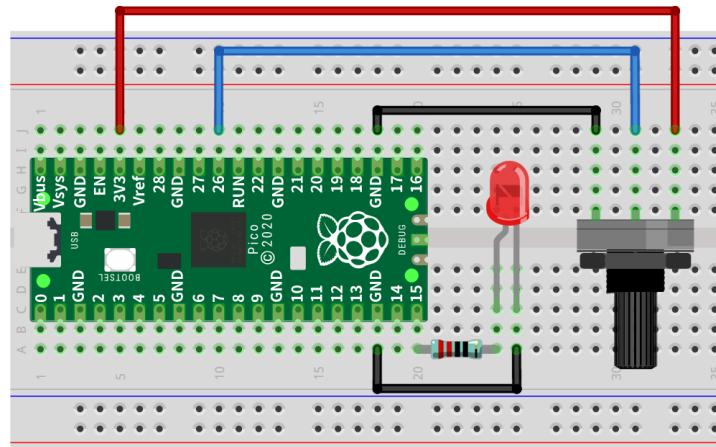
Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
Rotary potentiometer x1	Resistor 220Ω x1	LED x1	Jumper

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

Code

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

09.1_Soft_LED

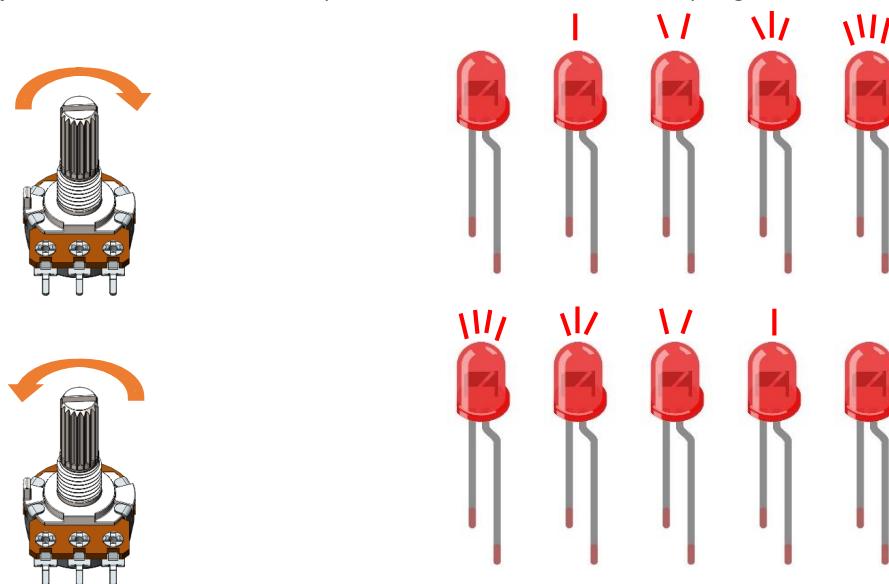


```

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()

```

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.

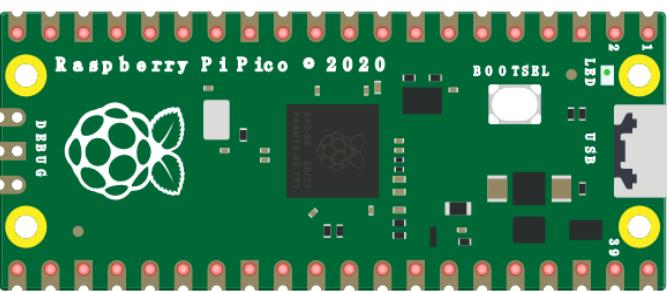
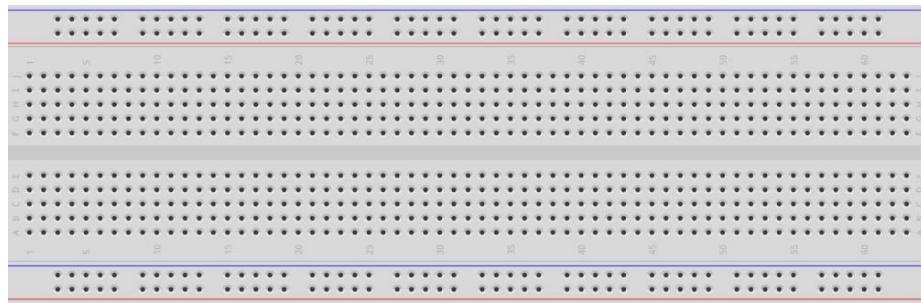
Chapter 10 Photoresistor & LED

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

Project 10.1 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

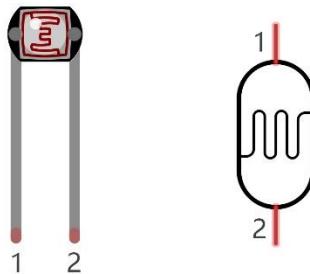
Raspberry Pi Pico x1		USB cable x1
Breadboard x1		
Photoresistor x1	Resistor 220Ω x1 10KΩ x1	LED x1 Jumper



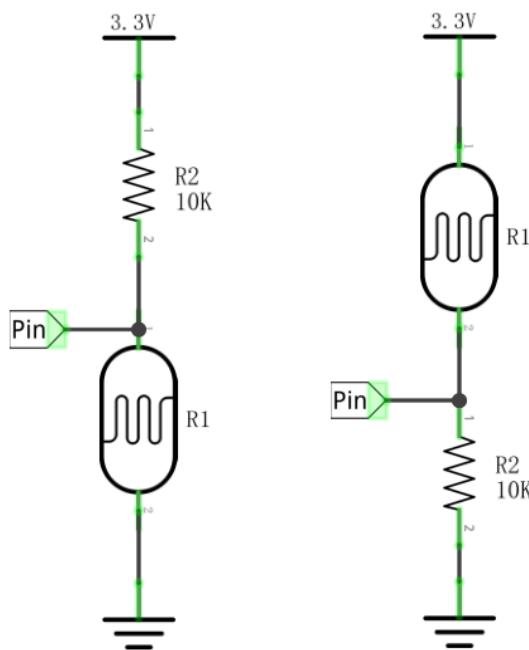
Component knowledge

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:

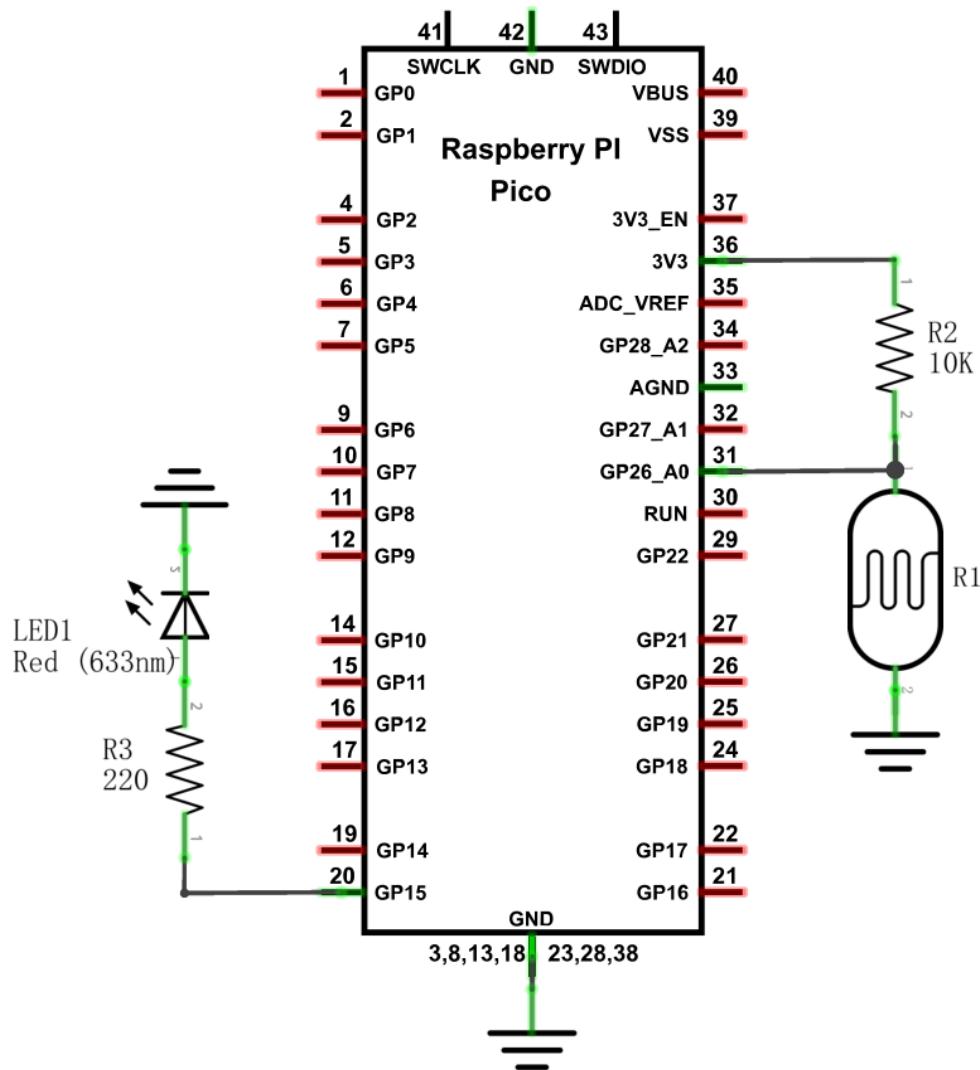


In the above circuit, when a Photoresistor's resistance value changes due to a change in light intensity, the voltage between the Photoresistor and Resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

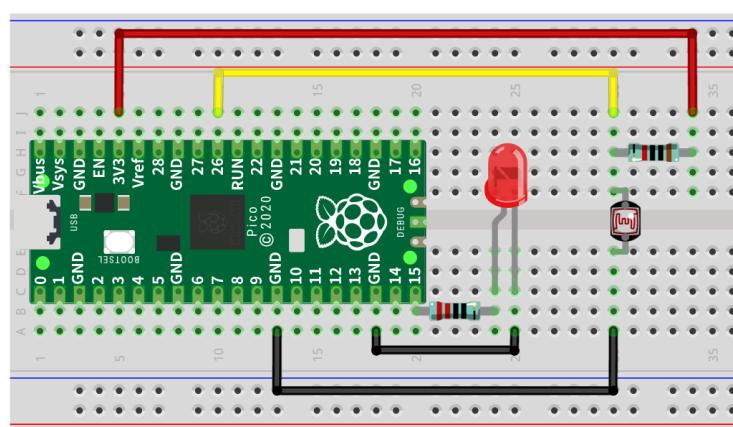
Circuit

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](mailto: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” → “10.1_Photoresistor” and double click “10.1_Photoresistor.py”.

10.1_Photoresistor

```

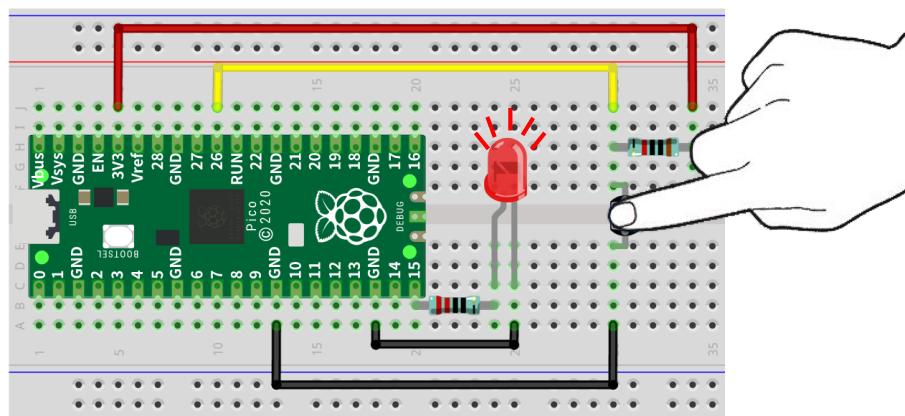
1 from machine import Pin, ADC, PWM
2 import time
3
4 adc = ADC(26)
5 pwm = PWM(Pin(15))
6 pwm.freq(10000)
7 try:
8     while True:
9         pwm.duty_u16(adc.read_u16())
10        time.sleep(0.1)
11 except:
12     pwm.deinit()

```

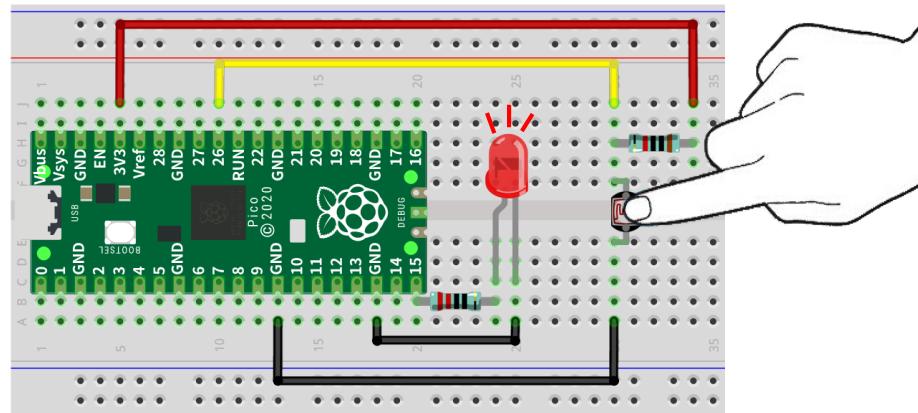
Shell x
MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>

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

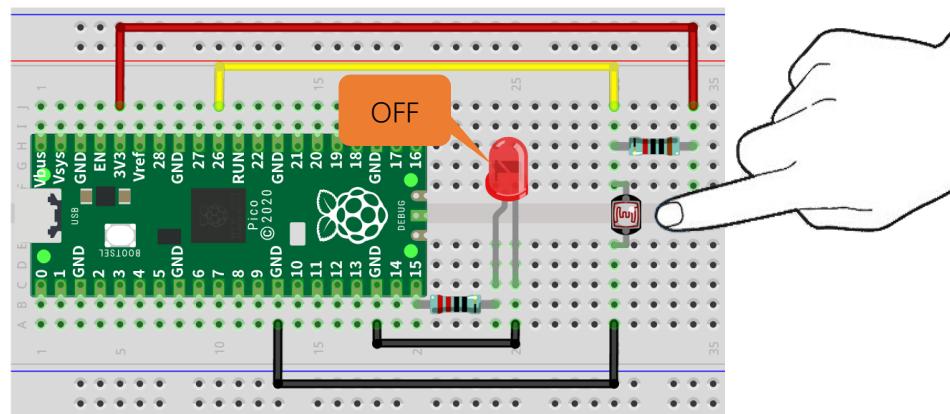
Fully cover the photoresistor:



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()

```



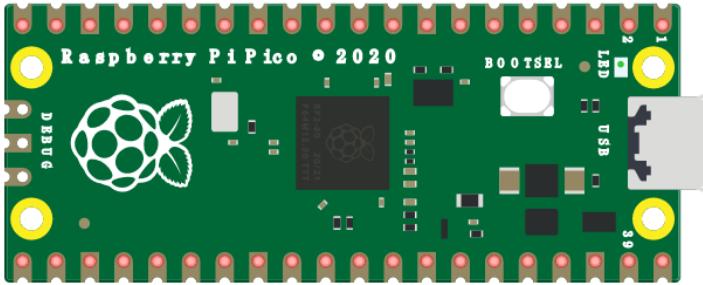
Chapter 11 Thermistor

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

Project 11.1 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.

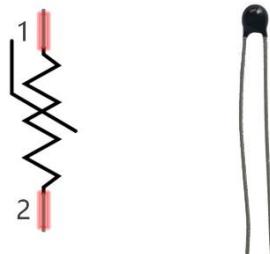
Component List

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:

R_t is the thermistor resistance under T_2 temperature;

R is the nominal resistance of thermistor under T_1 temperature;

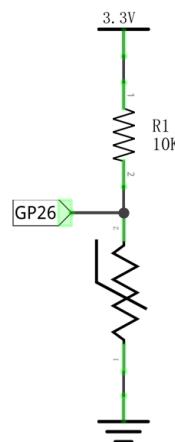
EXP[n] is nth power of e;

B is for thermal index;

T_1, T_2 is Kelvin temperature (absolute temperature). Kelvin temperature=273.15 + Celsius temperature.

For the parameters of the Thermistor, we use: $B=3950$, $R=10\text{k}\Omega$, $T_1=25^\circ\text{C}$.

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.

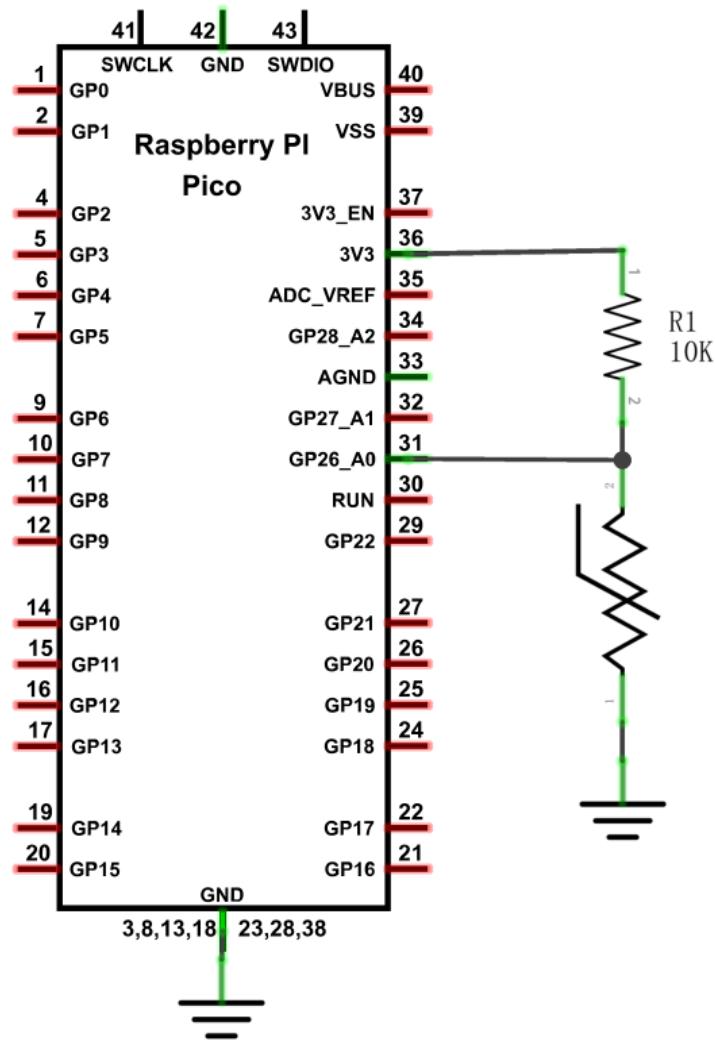
Therefore, the temperature formula can be derived as:

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

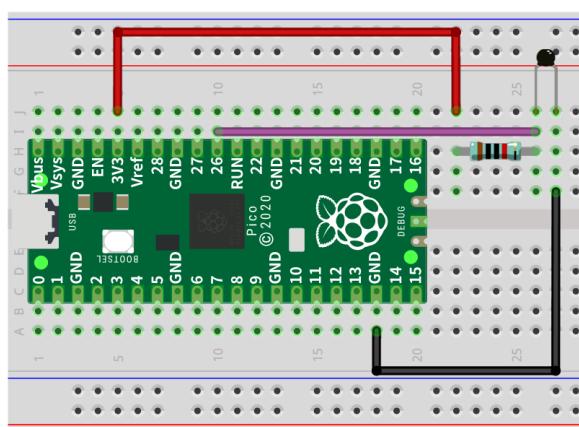
Circuit

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

Code

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

11.1_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)

```

Shell x

ADC value	voltage	Temperature
32055	1.61	26C
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

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)
```

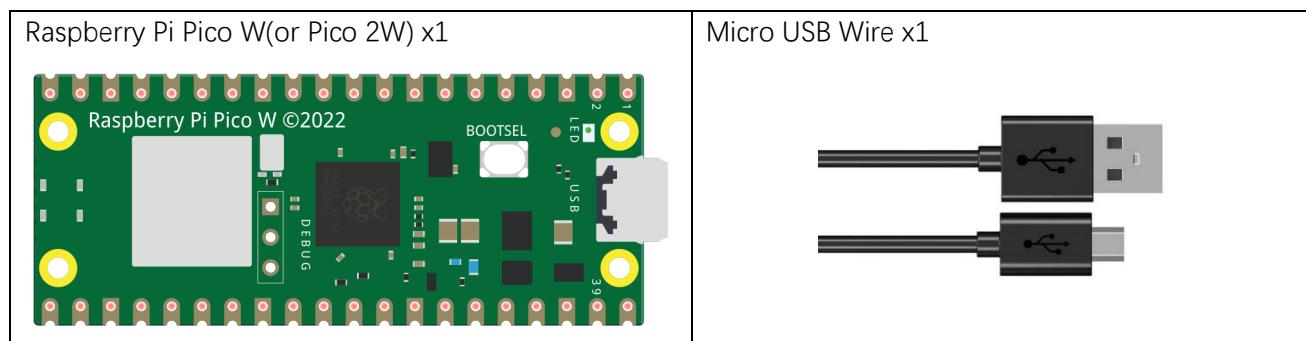
Chapter 12 WiFi Working Modes (Only for Pico W/Pico 2W)

The biggest difference between the raspberry pi pico and the raspberry pi pico W is that the raspberry pi Pico W is equipped with a WiFi function module. At the beginning of this chapter, we will learn about the WiFi function of Pico W of Raspberry Pi.

If you have Pico in your hand, please change it to Pico W before continuing to learn.

Project 12.1 Station mode

Component List



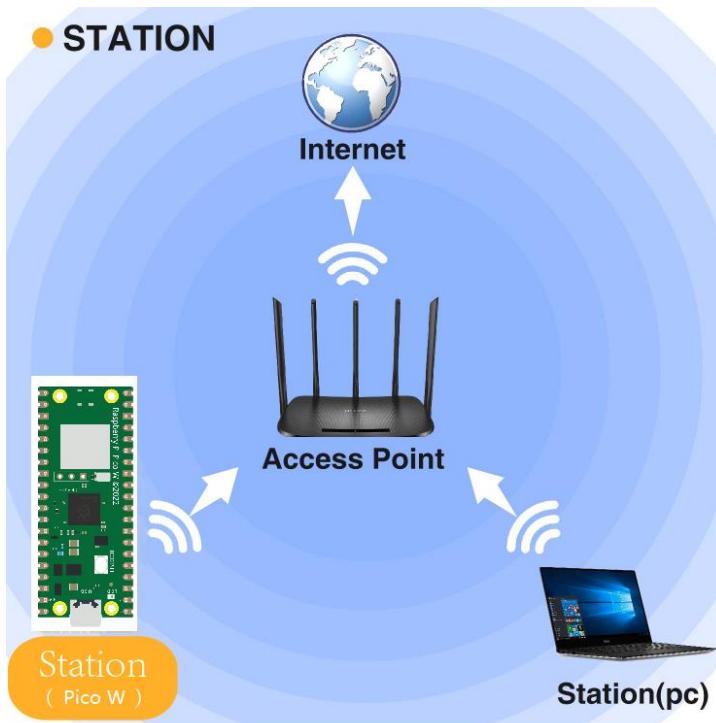
Component knowledge

Wireless

Pico W has an on-board 2.4GHz wireless interface using an Infineon CYW43439. The antenna is an onboard antenna licensed from ABRACON (formerly ProAnt). The wireless interface is connected via SPI to the RP2040.

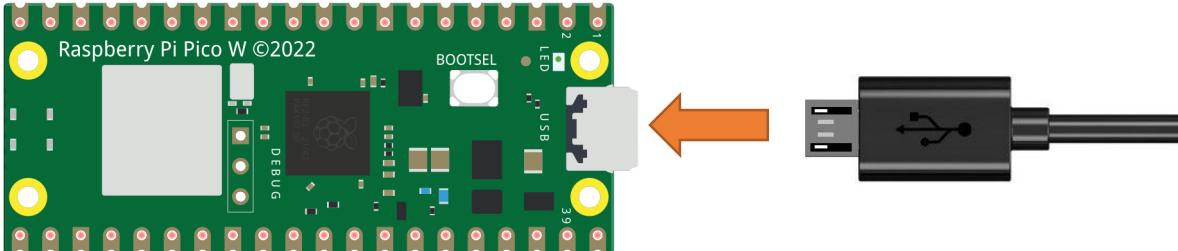
Station mode

When PICO W selects Station mode, it acts as a WiFi client. It can connect to the router network and communicate with other devices on the router via WiFi connection. As shown below, the PC is connected to the router, and if PICO W wants to communicate with the PC, it needs to be connected to the router.



Circuit

Connect Pico W to the computer using the USB cable.

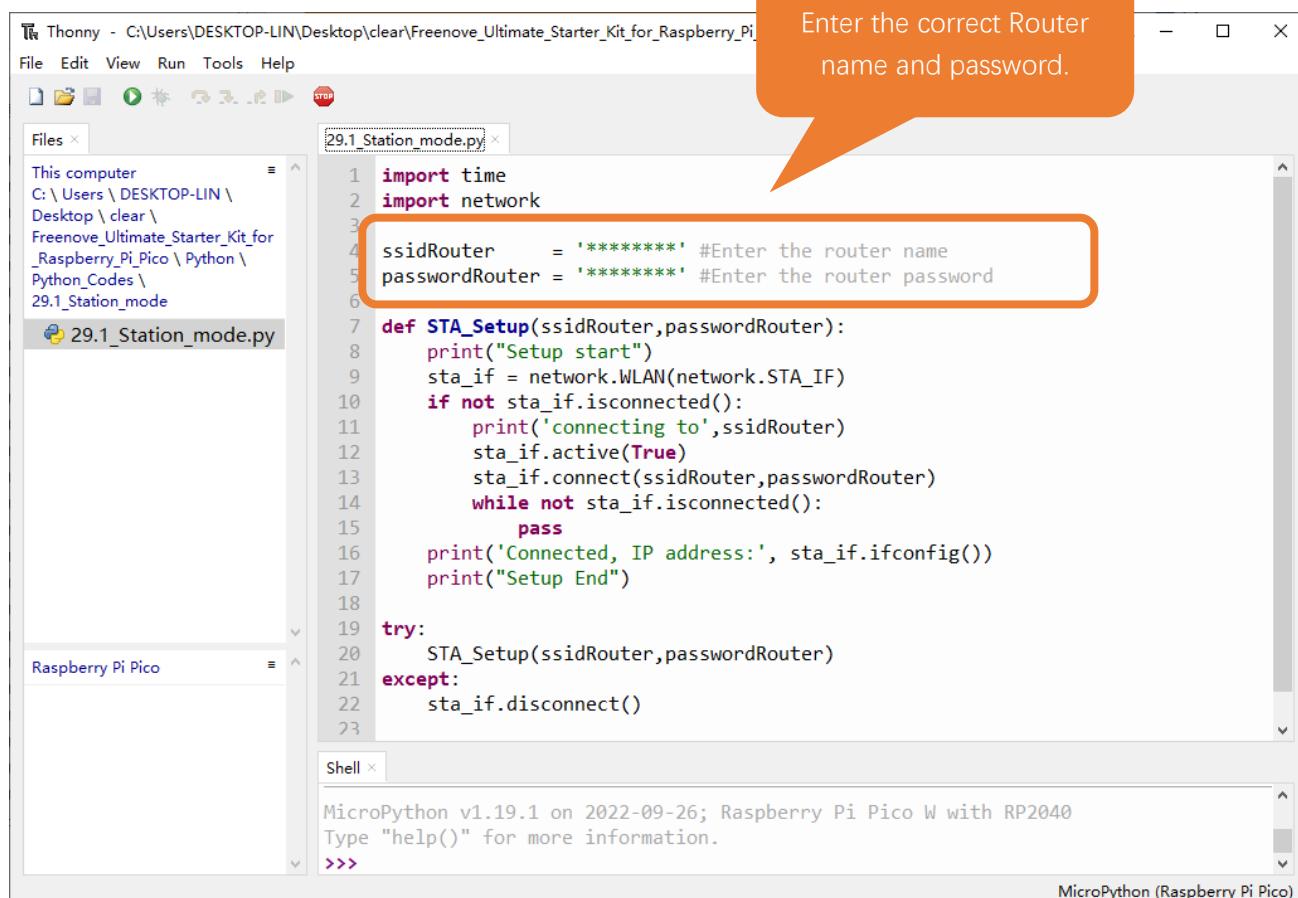


Code

Move the program folder "**Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "12.1_Station_mode" and double click "12.1_Station_mode.py".

12.1_Station_mode



The screenshot shows the Thonny IDE interface. In the top menu bar, it says "Thonny - C:\Users\DESKTOP-LIN\Desktop\clear\Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi\Python_Codes\29.1_Station_mode". The left sidebar shows a file tree with "This computer" and "Raspberry Pi Pico" sections, and a file "29.1_Station_mode.py" is selected. The main code editor window contains the following Python code:

```

1 import time
2 import network
3
4 ssidRouter      = '*****' #Enter the router name
5 passwordRouter = '*****' #Enter the router password
6
7 def STA_Setup(ssidRouter,passwordRouter):
8     print("Setup start")
9     sta_if = network.WLAN(network.STA_IF)
10    if not sta_if.isconnected():
11        print('connecting to',ssidRouter)
12        sta_if.active(True)
13        sta_if.connect(ssidRouter,passwordRouter)
14        while not sta_if.isconnected():
15            pass
16    print('Connected, IP address:', sta_if.ifconfig())
17    print("Setup End")
18
19 try:
20     STA_Setup(ssidRouter,passwordRouter)
21 except:
22     sta_if.disconnect()
23

```

The lines `ssidRouter` and `passwordRouter` are highlighted with a red rectangle, and an orange callout bubble points to them with the text "Enter the correct Router name and password". Below the code editor is a "Shell" window showing the MicroPython prompt:

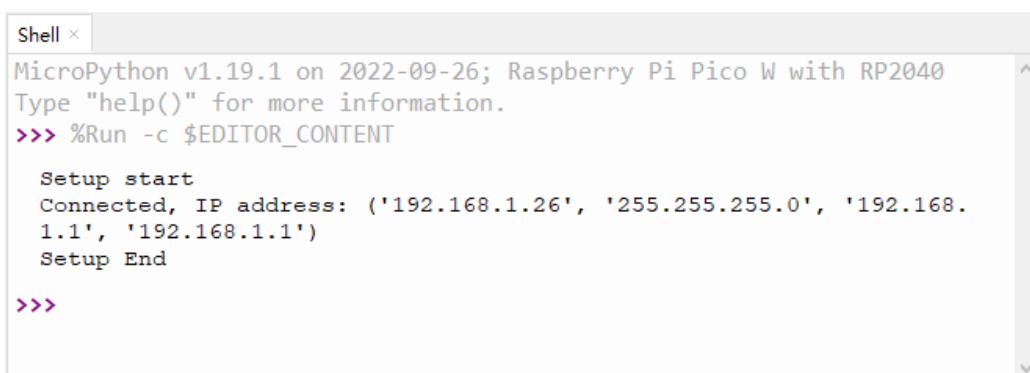
```

MicroPython v1.19.1 on 2022-09-26; Raspberry Pi Pico W with RP2040
Type "help()" for more information.
>>>

```

Because the names and passwords of routers in various places are different, before the Code runs, users need to enter the correct router's name and password in the box as shown in the illustration above.

After making sure the router name and password are entered correctly, compile and upload codes to PICO W, wait for PICO W to connect to your router and print the IP address assigned by the router to PICO W in "Shell".



The screenshot shows the MicroPython Shell window. It displays the following text:

```

MicroPython v1.19.1 on 2022-09-26; Raspberry Pi Pico W with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

Setup start
Connected, IP address: ('192.168.1.26', '255.255.255.0', '192.168.
1.1', '192.168.1.1')
Setup End

>>>

```

The following is the program code:

1	import time
2	import network
3	
4	ssidRouter = '*****' #Enter the router name
5	passwordRouter = '*****' #Enter the router password
6	

```

7 def STA_Setup(ssidRouter, passwordRouter):
8     print("Setup start")
9     sta_if = network.WLAN(network.STA_IF)
10    if not sta_if.isconnected():
11        print(' connecting to',ssidRouter)
12        sta_if.active(True)
13        sta_if.connect(ssidRouter, passwordRouter)
14        while not sta_if.isconnected():
15            pass
16        print(' Connected, IP address:', sta_if.ifconfig())
17        print("Setup End")
18
19 try:
20     STA_Setup(ssidRouter, passwordRouter)
21 except:
22     sta_if.disconnect()

```

Import network module.

```
2 import network
```

Enter correct router name and password.

```

4 const char *ssid_Router      = "*****"; //Enter the router name
5 const char *password_Router = "*****"; //Enter the router password

```

Set PICO W in Station mode.

```
9 sta_if = network.WLAN(network.STA_IF)
```

Activate Pico W's Station mode, initiate a connection request to the router and enter the password to connect.

```

12     sta_if.active(True)
13     sta_if.connect(ssidRouter, passwordRouter)

```

Wait for PICO W to connect to router until they connect to each other successfully.

```

14     while not sta_if.isconnected():
15         pass

```

Print the IP address assigned to PICO W in "Shell".

```
16 Print('Connected, IP address:', sta_if.ifconfig())
```

Reference

Class network

Before each use of **network**, please add the statement "**import network**" to the top of the python file.

WLAN(interface_id): Set to WiFi mode.

network.STA_IF: Client, connecting to other WiFi access points.

network.AP_IF: Access points, allowing other WiFi clients to connect.

active(is_active): With parameters, it is to check whether to activate the network interface; Without parameters, it is to query the current state of the network interface.

scan(ssid, bssid, channel, RSSI, authmode, hidden): Scan for wireless networks available nearby (only scan on STA interface), return a tuple list of information about the WiFi access point.

bssid: The hardware address of the access point, returned in binary form as a byte object. You can use

Any concerns? ✉ support@freenove.com

ubinascii.hexlify() to convert it to ASCII format.

authmode: Access type

AUTH_OPEN = 0
AUTH_WEP = 1
AUTH_WPA_PSK = 2
AUTH_WPA2_PSK = 3
AUTH_WPA_WPA2_PSK = 4
AUTH_MAX = 6

Hidden: Whether to scan for hidden access points

False: Only scanning for visible access points

True: Scanning for all access points including the hidden ones.

isconnected(): Check whether PICO W is connected to AP in Station mode. In STA mode, it returns True if it is connected to a WiFi access point and has a valid IP address; Otherwise, it returns False.

connect(ssid, password): Connecting to wireless network.

ssid: WiFi name

password: WiFi password

disconnect(): Disconnect from the currently connected wireless network.



Project 12.2 AP mode

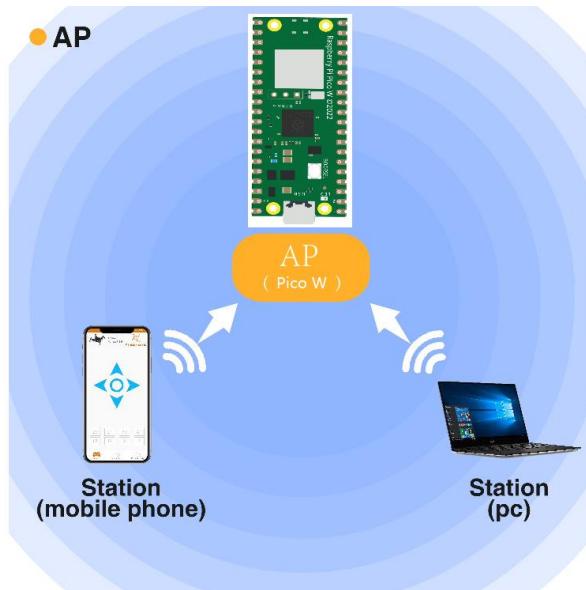
Component List & Circuit

Component List & Circuit are the same as in Section 12.1.

Component knowledge

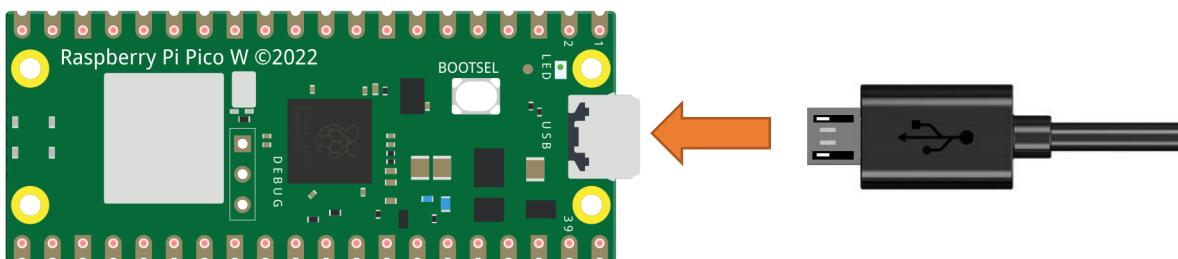
AP mode

When PICO W selects AP mode, it creates a hotspot network that is separated from the Internet and waits for other WiFi devices to connect. As shown in the figure below, PICO W is used as a hotspot. If a mobile phone or PC wants to communicate with PICO W, it must be connected to the hotspot of PICO W. Only after a connection is established with PICO W can they communicate.



Circuit

Connect Pico W to the computer using the USB cable.

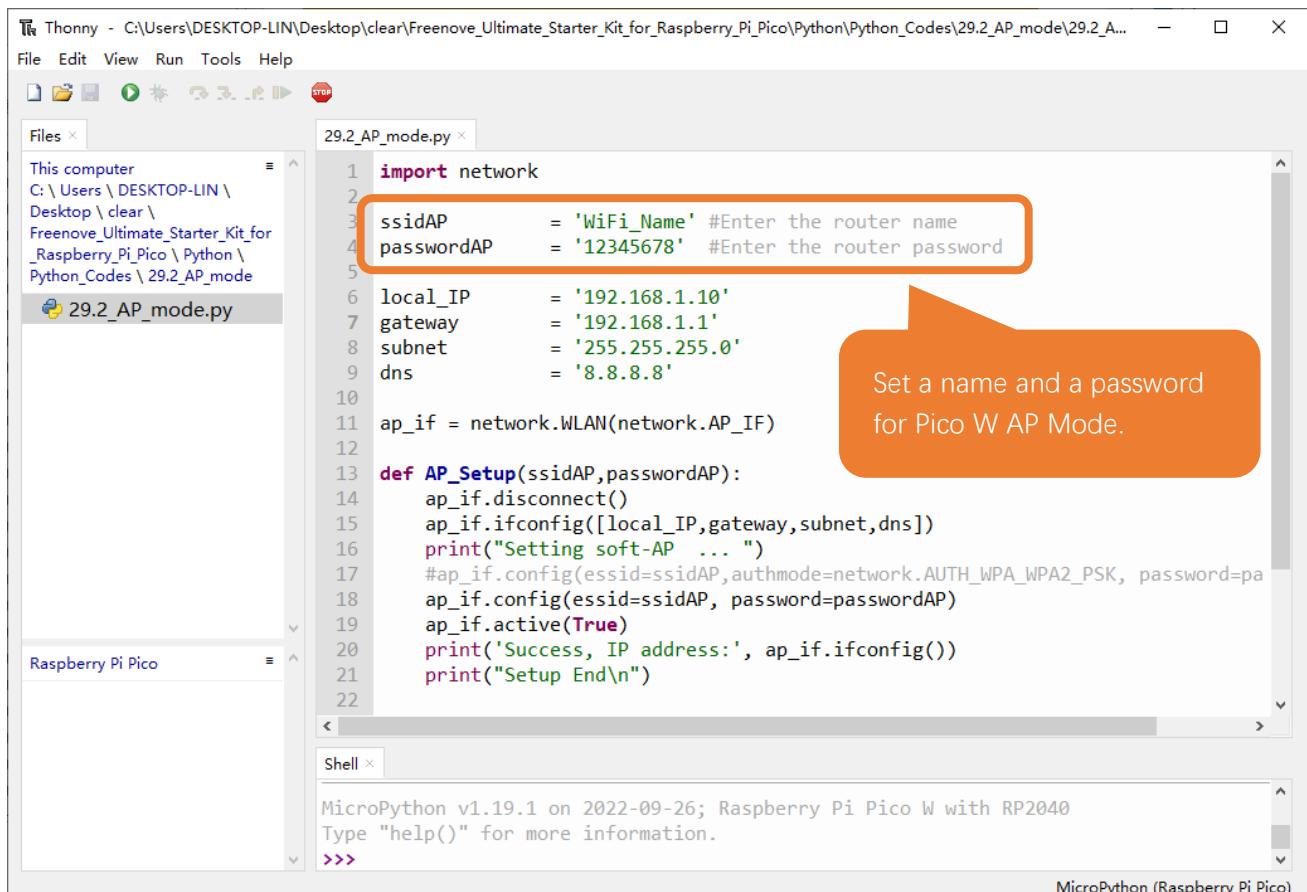


Code

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

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “12.2_AP_mode”. and double click “12.2_AP_mode.py”.

12.2_AP_mode



```

import network
ssidAP      = 'WiFi_Name' #Enter the router name
passwordAP  = '12345678' #Enter the router password
local_IP    = '192.168.1.10'
gateway     = '192.168.1.1'
subnet      = '255.255.255.0'
dns         = '8.8.8.8'

ap_if = network.WLAN(network.AP_IF)

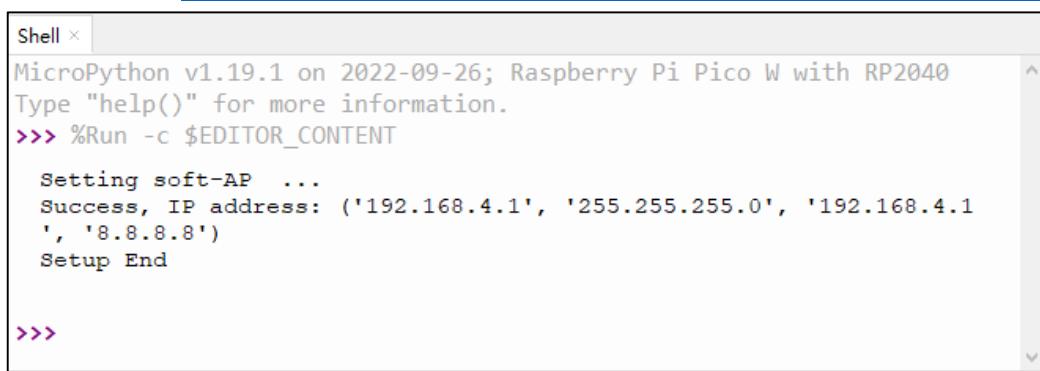
def AP_Setup(ssidAP,passwordAP):
    ap_if.disconnect()
    ap_if.ifconfig([local_IP,gateway,subnet,dns])
    print("Setting soft-AP ... ")
    #ap_if.config(essid=ssidAP,authmode=network.AUTH_WPA_WPA2_PSK, password=passwordAP)
    ap_if.config(essid=ssidAP, password=passwordAP)
    ap_if.active(True)
    print('Success, IP address:', ap_if.ifconfig())
    print("Setup End\n")

```

Set a name and a password for Pico W AP Mode.

Before the Code runs, you can make any changes to the AP name and password for PICO W in the box as shown in the illustration above. Of course, you can leave it alone by default.

Click “Run current script”, open the AP function of PICO W and print the access point information.



```

MicroPython v1.19.1 on 2022-09-26; Raspberry Pi Pico W with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

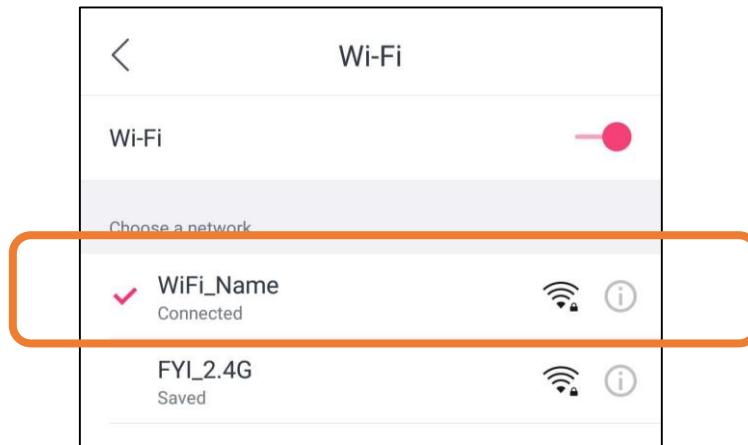
Setting soft-AP ...
Success, IP address: ('192.168.4.1', '255.255.255.0', '192.168.4.1',
', '8.8.8.8')
Setup End

>>>

```



Turn on the WiFi scanning function of your phone, and you can see the ssid_AP on PICO W, which is called "WiFi_Name" in this Code. You can enter the password "12345678" to connect it or change its AP name and password by modifying Code.



The following is the program code:

```

1 import network
2
3 ssidAP      = 'WiFi_Name' #Enter the router name
4 passwordAP  = '12345678' #Enter the router password
5
6 local_IP    = '192.168.1.10'
7 gateway     = '192.168.1.1'
8 subnet      = '255.255.255.0'
9 dns         = '8.8.8.8'
10
11 ap_if = network.WLAN(network.AP_IF)
12
13 def AP_Setup(ssidAP, passwordAP):
14     ap_if.ifconfig([local_IP, gateway, subnet, dns])
15     print("Setting soft-AP ... ")
16     ap_if.config(essid=ssidAP, password=passwordAP)
17     ap_if.active(True)
18     print(' Success, IP address:', ap_if.ifconfig())
19     print("Setup End\n")
20
21 try:
22     AP_Setup(ssidAP, passwordAP)
23 except:
24     ap_if.disconnect()

```

Import network module.

1	import network
---	----------------

Enter correct AP name and password.

```
3    ssidAP      = 'WiFi_Name' #Enter the router name
4    passwordAP  = '12345678' #Enter the router password
```

Set PICO W in AP mode.

```
11   ap_if = network.WLAN(network.AP_IF)
```

Configure IP address, gateway and subnet mask for PICO W.

```
14   ap_if.ifconfig([local_IP, gateway, subnet, dns])
```

Turn on an AP in PICO W, whose name is set by ssid_AP and password is set by password_AP.

```
16   ap_if.config(essid=ssidAP, password=passwordAP)
17   ap_if.active(True)
```

If the program is running abnormally, the AP disconnection function will be called.

```
14   ap_if.disconnect()
```

Reference

Class network

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

WLAN(interface_id): Set to WiFi mode.

network.STA_IF: Client, connecting to other WiFi access points

network.AP_IF: Access points, allowing other WiFi clients to connect

active(is_active): With parameters, it is to check whether to activate the network interface; Without parameters, it is to query the current state of the network interface

isconnected(): In AP mode, it returns True if it is connected to the station; otherwise it returns False.

connect(ssid, password): Connecting to wireless network

ssid: WiFi name

password: WiFi password

config(essid, channel): To obtain the MAC address of the access point or to set the WiFi channel and the name of the WiFi access point.

ssid: WiFi account name

channel: WiFi channel

ifconfig([(ip, subnet, gateway, dns)]): Without parameters, it returns a 4-tuple (ip, subnet_mask, gateway, DNS_server); With parameters, it configures static IP.

ip: IP address

subnet_mask: subnet mask

gateway: gateway

DNS_server: DNS server

disconnect(): Disconnect from the currently connected wireless network

status(): Return the current status of the wireless connection



Project 12.3 AP+Station mode

Component List & Circuit

Component List & Circuit are the same as in Section 12.1.

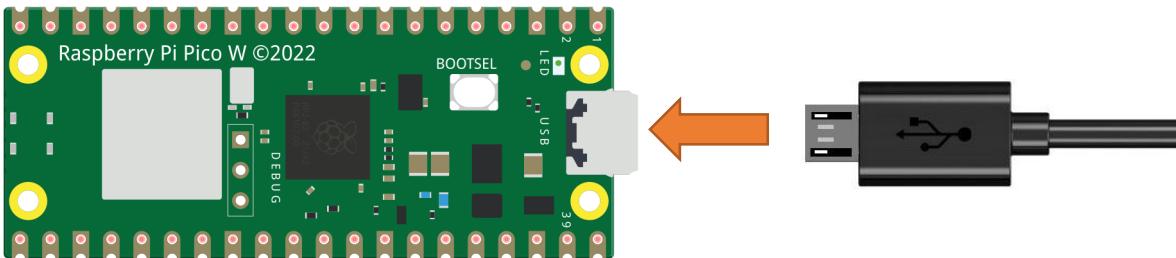
Component knowledge

AP+Station mode

PICO W currently does not support simultaneous use of AP mode and Station mode, so this section can be skipped. In the actual mode configuration, the last configured mode shall prevail.

Circuit

Connect Pico W to the computer using the USB cable.

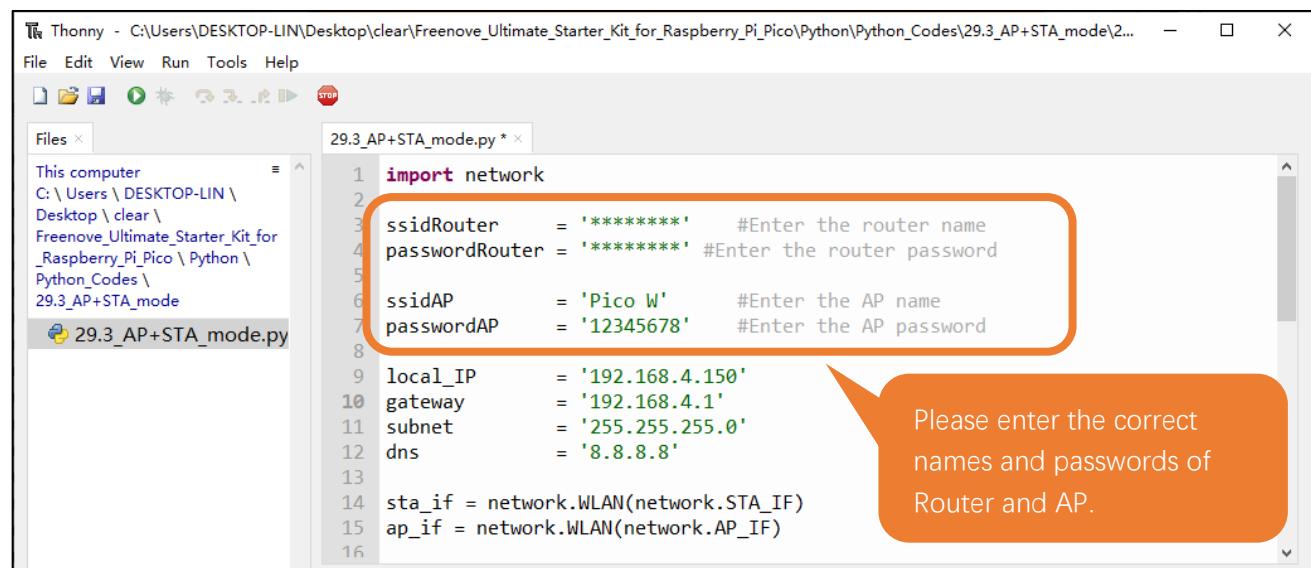


Code

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

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “12.3_AP+STA_mode” and double click “12.3_AP+STA_mode.py”.

12.3 AP+STA_mode



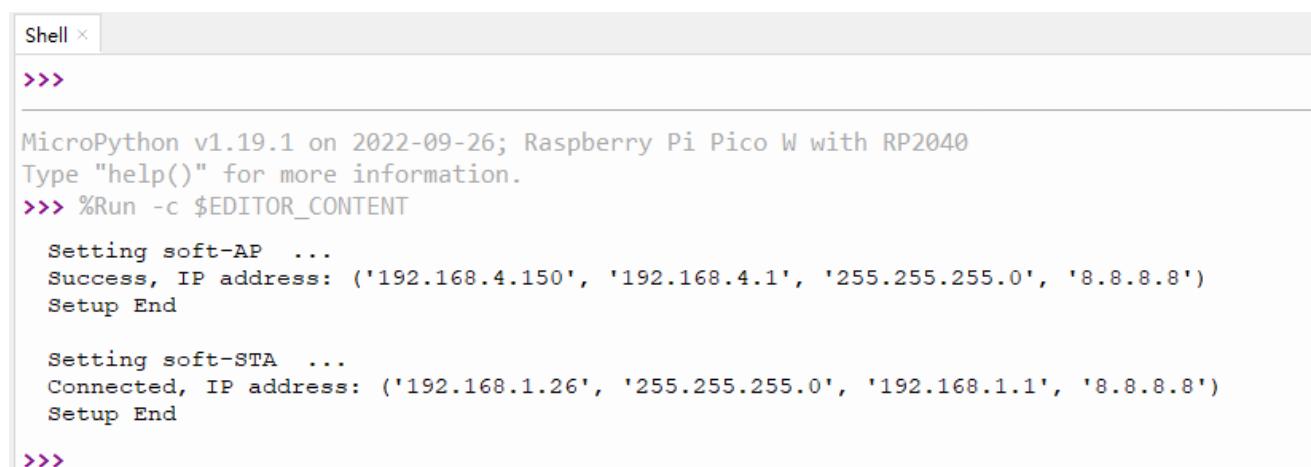
```

Thonny - C:\Users\DESKTOP-LIN\Desktop\clear\Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi_Pico\Python\Python_Codes\29.3_AP+STA_mode\2...
File Edit View Run Tools Help
Files 29.3_AP+STA_mode.py *
1 import network
2
3 ssidRouter = '*****' #Enter the router name
4 passwordRouter = '*****' #Enter the router password
5
6 ssidAP = 'Pico W' #Enter the AP name
7 passwordAP = '12345678' #Enter the AP password
8
9 local_IP = '192.168.4.150'
10 gateway = '192.168.4.1'
11 subnet = '255.255.255.0'
12 dns = '8.8.8.8'
13
14 sta_if = network.WLAN(network.STA_IF)
15 ap_if = network.WLAN(network.AP_IF)
16

```

Please enter the correct names and passwords of Router and AP.

It is analogous to Project 12.1 and Project 12.2. Before running the Code, you need to modify ssidRouter, passwordRouter, ssidAP and passwordAP shown in the box of the illustration above.
After making sure that the code is modified correctly, click “Run current script” and the “Shell” will display as follows:



```

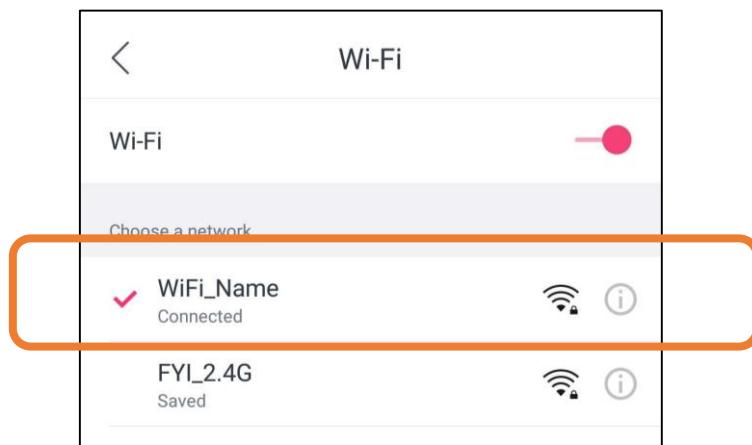
Shell >>>
>>>
MicroPython v1.19.1 on 2022-09-26; Raspberry Pi Pico W with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
Setting soft-AP ...
Success, IP address: ('192.168.4.150', '192.168.4.1', '255.255.255.0', '8.8.8.8')
Setup End

Setting soft-STA ...
Connected, IP address: ('192.168.1.26', '255.255.255.0', '192.168.1.1', '8.8.8.8')
Setup End
>>>

```



Turn on the WiFi scanning function of your phone, and you can see the ssidAP on PICO W.



The following is the program code:

```

1 import network
2
3 ssidRouter      = '*****' #Enter the router name
4 passwordRouter = '*****' #Enter the router password
5
6 ssidAP          = 'WiFi_Name' #Enter the AP name
7 passwordAP      = '12345678' #Enter the AP password
8
9 local_IP        = '192.168.4.150'
10 gateway         = '192.168.4.1'
11 subnet          = '255.255.255.0'
12 dns             = '8.8.8.8'
13
14 sta_if = network.WLAN(network.STA_IF)
15 ap_if = network.WLAN(network.AP_IF)
16
17 def STA_Setup(ssidRouter, passwordRouter):
18     print("Setting soft-STA ... ")
19     if not sta_if.isconnected():
20         print(' connecting to',ssidRouter)
21         sta_if.active(True)
22         sta_if.connect(ssidRouter, passwordRouter)
23         while not sta_if.isconnected():
24             pass
25     print(' Connected, IP address:', sta_if.ifconfig())
26     print("Setup End")
27
28 def AP_Setup(ssidAP, passwordAP):
29     ap_if.ifconfig([local_IP, gateway, subnet, dns])
30     print("Setting soft-AP ... ")

```

Any concerns? ✉ support@freenove.com

```
31     ap_if.config(essid=ssidAP , password=passwordAP)
32     ap_if.active(True)
33     print(' Success, IP address:', ap_if.ifconfig())
34     print("Setup End\n")
35
36 try:
37     AP_Setup(ssidAP,passwordAP)
38     STA_Setup(ssidRouter,passwordRouter)
39 except:
40     sta_if.disconnect()
41     ap_if.disconnect()
```



Chapter 13 TCP/IP (Only for Pico W/Pico 2W)

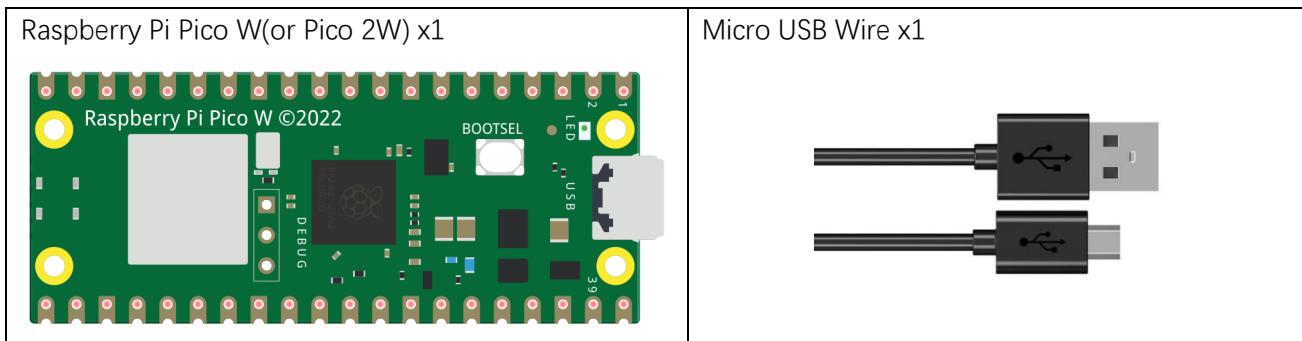
If you have Pico in your hand, please change it to Pico W before continuing to learn.

In this chapter, we will introduce how PICO W implements network communications based on TCP/IP protocol. There are two roles in TCP/IP communication, namely Server and Client, which will be implemented respectively with two projects in this chapter.

Project 13.1 as Client

In this section, PICO W is used as Client to connect Server on the same LAN and communicate with it.

Component List



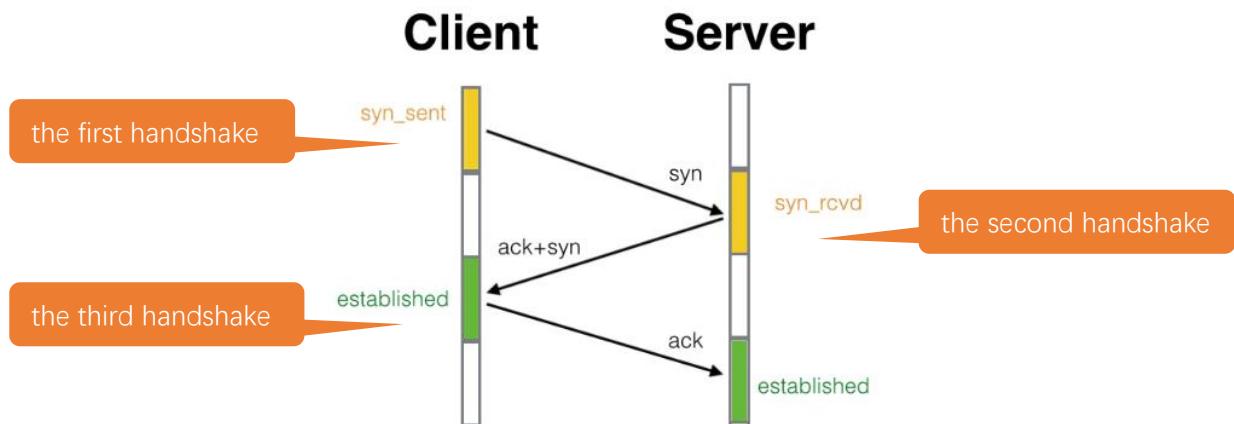
Component knowledge

TCP connection

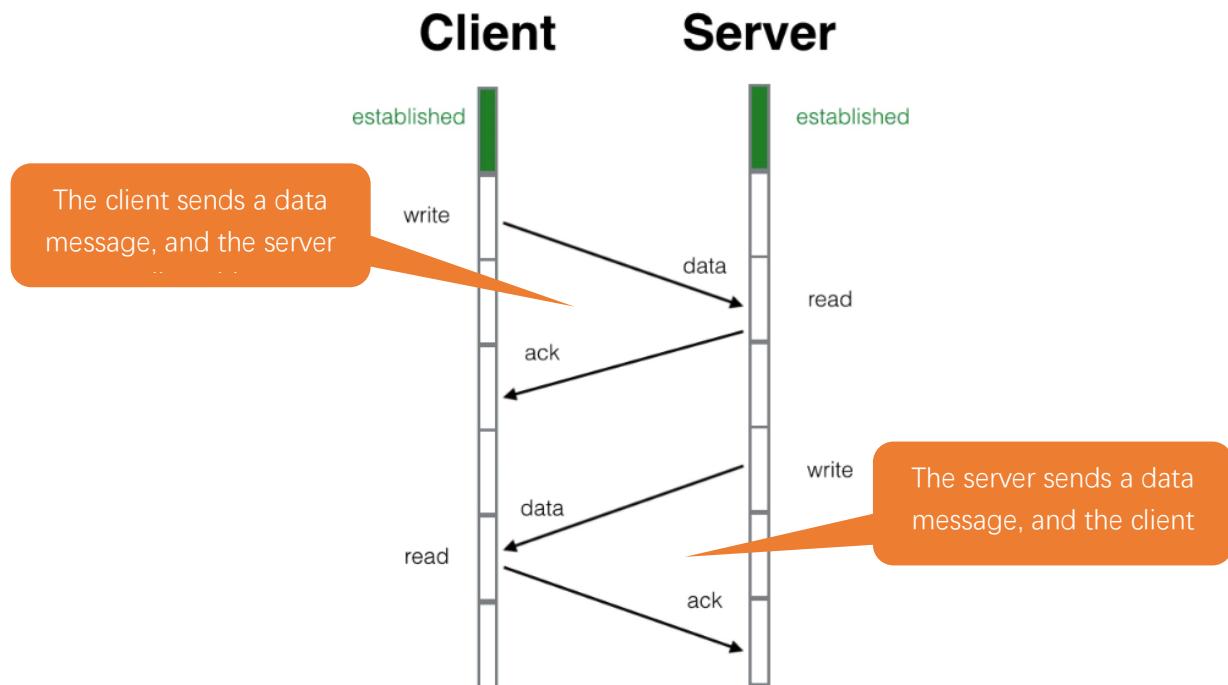
Before transmitting data, TCP needs to establish a logical connection between the sending end and the receiving end. It provides reliable and error-free data transmission between the two computers. In the TCP connection, the client and the server must be clarified. The client sends a connection request to the server, and each time such a request is proposed, a "three-time handshake" is required.

Three-time handshake: In the TCP protocol, during the preparation phase of sending data, the client and the server interact three times to ensure the reliability of the connection, which is called "three-time handshake". The first handshake, the client sends a connection request to the server and waits for the server to confirm. The second handshake, the server sends a response back to the client informing that it has received the connection request.

The third handshake, the client sends a confirmation message to the server again to confirm the connection.



TCP is a connection-oriented, low-level transmission control protocol. After TCP establishes a connection, the client and server can send and receive messages to each other, and the connection will always exist as long as the client or server does not initiate disconnection. Each time one party sends a message, the other party will reply with an ack signal.





Install Processing

In this tutorial, we use Processing to build a simple TCP/IP communication platform.

If you have not installed Processing, you can download it by clicking <https://processing.org/download/>. You can choose an appropriate version to download according to your PC system.

The screenshot shows the official Processing website. At the top, there's a navigation bar with links for "Processing", "p5.js", "Processing.py", "Processing for Android", "Processing for Pi", and "Processing Foundation". Below the navigation bar is a large banner featuring the word "Processing" in a bold, sans-serif font, overlaid on a dark background with a geometric, wireframe-like pattern. To the right of the banner is a search bar with a magnifying glass icon. On the left side of the main content area, there's a sidebar with links: "Cover", "Download", "Donate", "Exhibition", "Reference", "Libraries", "Tools", "Environment", "Tutorials", "Examples", "Books", "Overview", and "People". In the center, under the heading "Download Processing", it says "Processing is available for Linux, Mac OS X, and Windows. Select your choice to download the software below." Below this text is a large circular logo with a stylized "P" inside. To the right of the logo, the version "3.5.4 (17 January 2020)" is shown, along with download links for "Windows 64-bit", "Windows 32-bit", "Linux 64-bit", and "Mac OS X".

Unzip the downloaded file to your computer. Click "processing.exe" as the figure below to run this software.

core	2020/1/17 12:16
java	2020/1/17 12:17
lib	2020/1/17 12:16
modes	2020/1/17 12:16
tools	2020/1/17 12:16
processing.exe	2020/1/17 12:16
processing-java.exe	2020/1/17 12:16
revisions.txt	2020/1/17 12:16

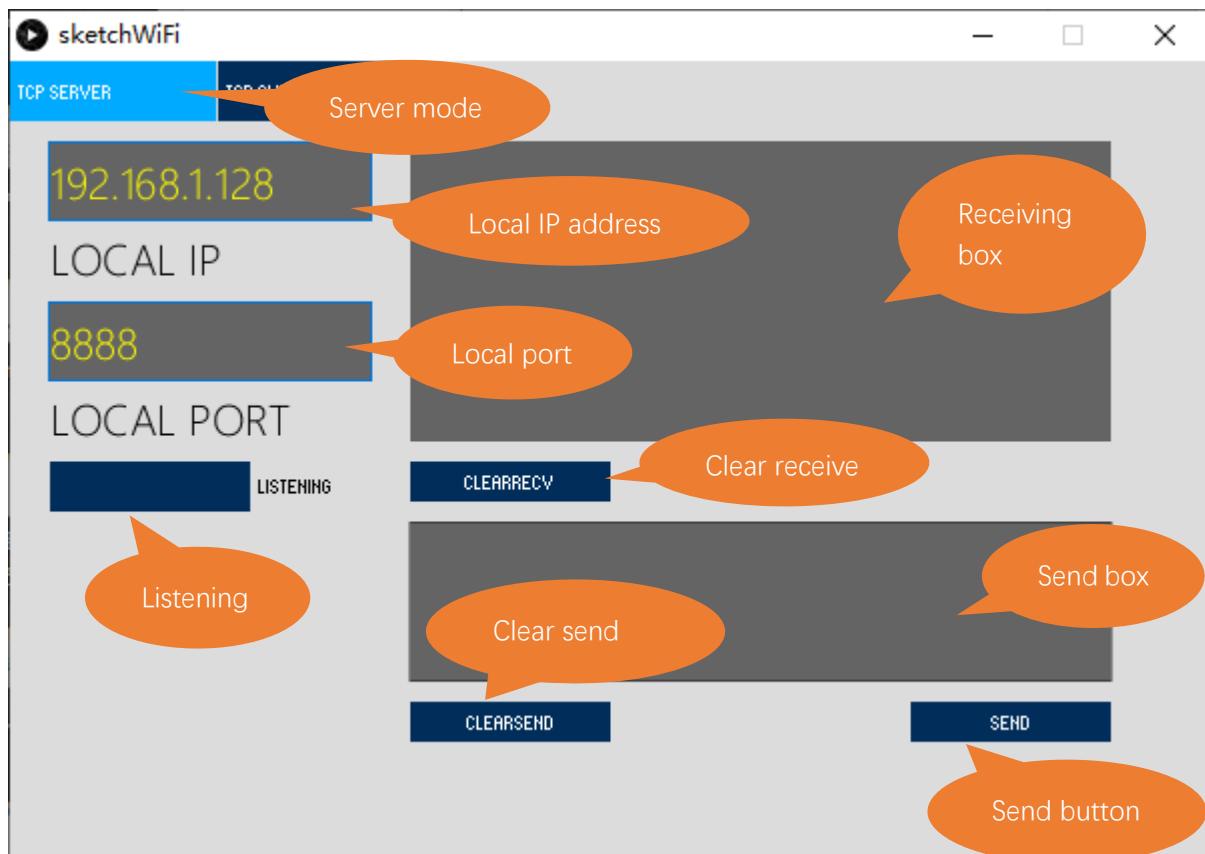
Use Server mode for communication

Open the

"Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico/Codes/Micropython_Codes/13.1_TCP_as_Client/sketchWiFi/sketchWiFi.pde". Click "Run".

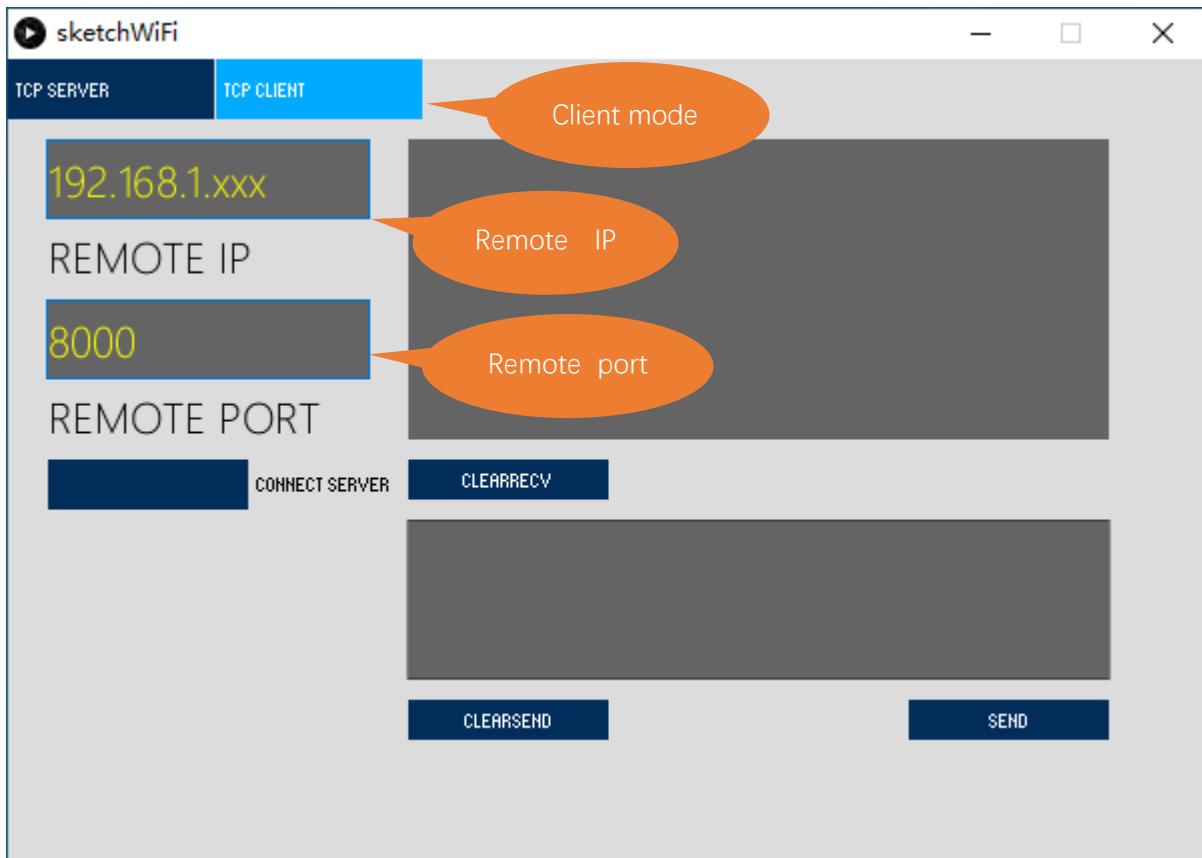


The new pop-up interface is as follows. If PICO W is used as Client, select TCP SERVER mode for sketchWiFi.



When sketchWiFi selects TCP SERVER mode, PICO W Code needs to be changed according to sketchWiFi's displaying of LOCAL IP or LOCAL PORT.

If PICO W serves as Server, select TCP CLIENT mode for sketchWiFi.



When sketchWiFi selects TCP CLIENT mode, the LOCAL IP and LOCAL PORT of sketchWiFi need to be changed according to the IP address and port number printed by the serial monitor.

Mode selection: select **Server mode/Client mode**.

IP address: In Server mode, this option does not need to be filled in, and the computer will automatically obtain the IP address.

In Client mode, fill in the remote IP address to be connected.

Port number: In Server mode, fill in a port number for client devices to make an access connection.

In client mode, fill in port number given by the Server devices to make an access connection.

Start button: In server mode, push the button, and then the computer will serve as Server and open a port number for Client to make access connection. During this period, the computer will keep monitoring.

In client mode, before pushing the button, please make sure the server is on, remote IP address and remote port number is correct; push the button, and the computer will make access connection to the remote port number of the remote IP as a Client.

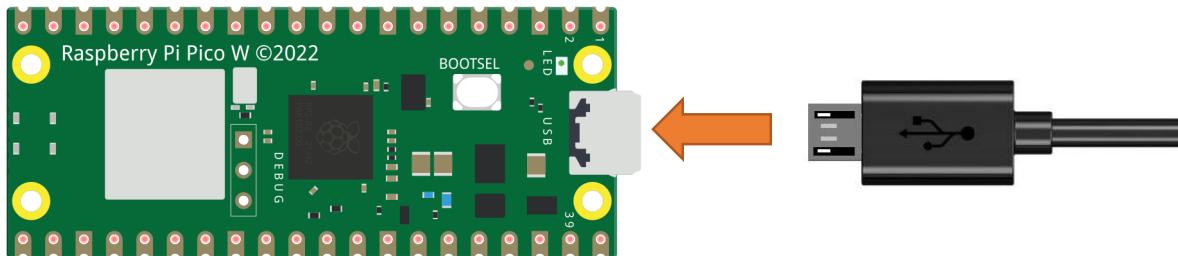
clear receive: clear out the content in the receiving text box

clear send: clear out the content in the sending text box

Sending button: push the sending button, the computer will send the content in the text box to others.

Circuit

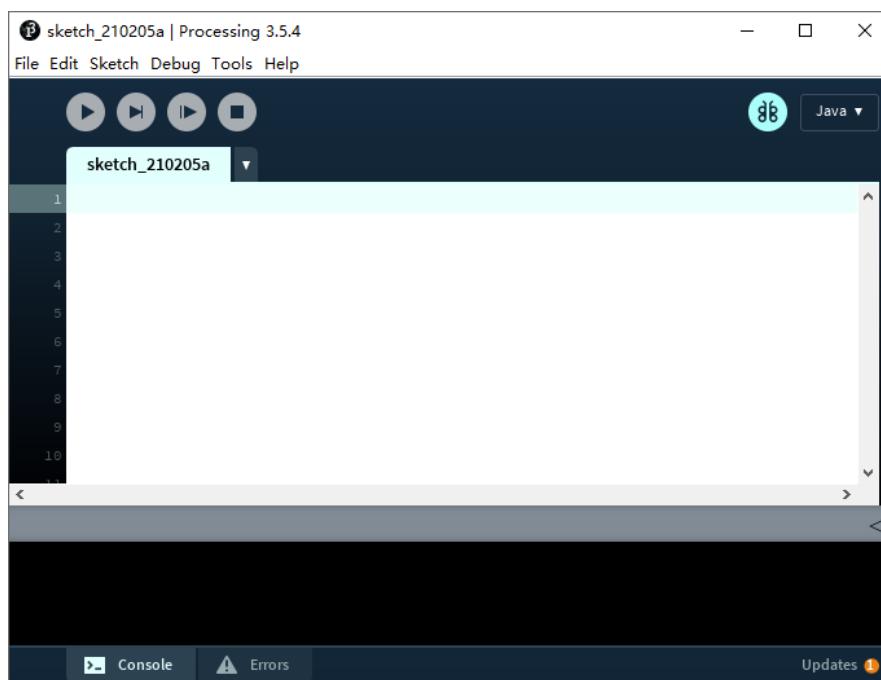
Connect Pico W to the computer using the USB cable.



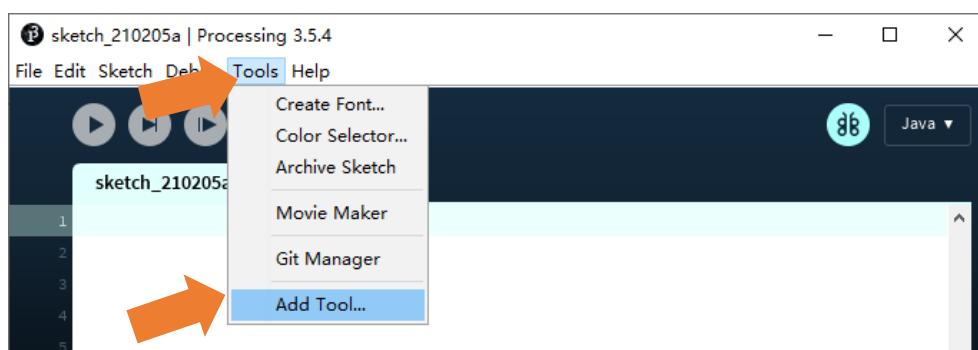
Code

If you have not installed “ControlP5”, please follow the following steps to continue the installation, if you have installed, please skip this section.

Open Processing.

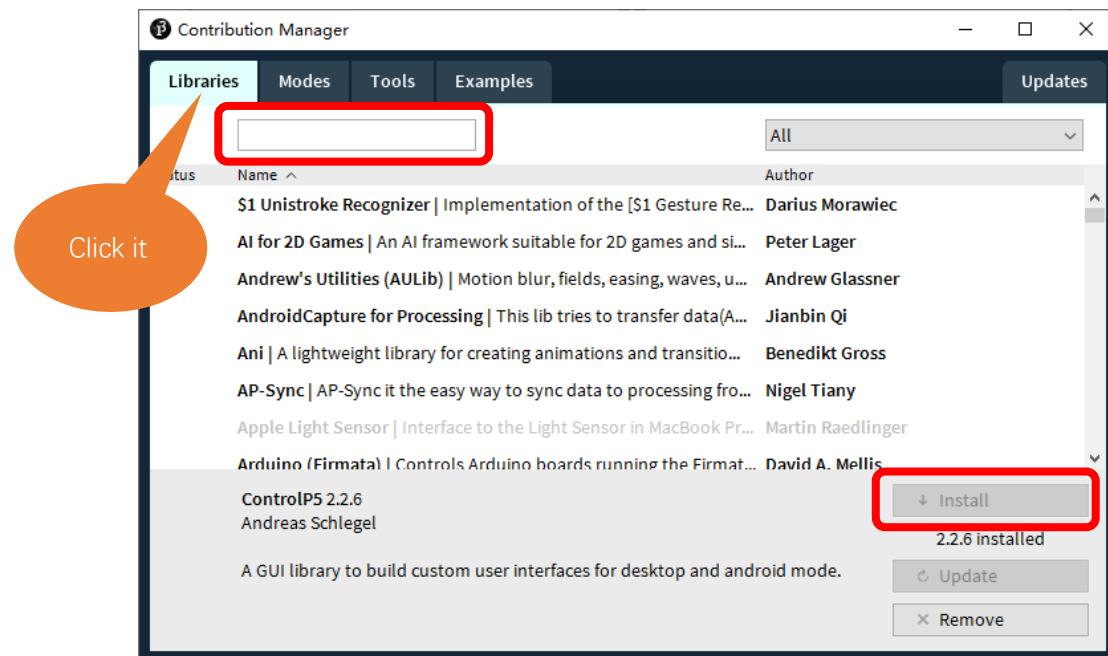


Click Add Tool under Tools.



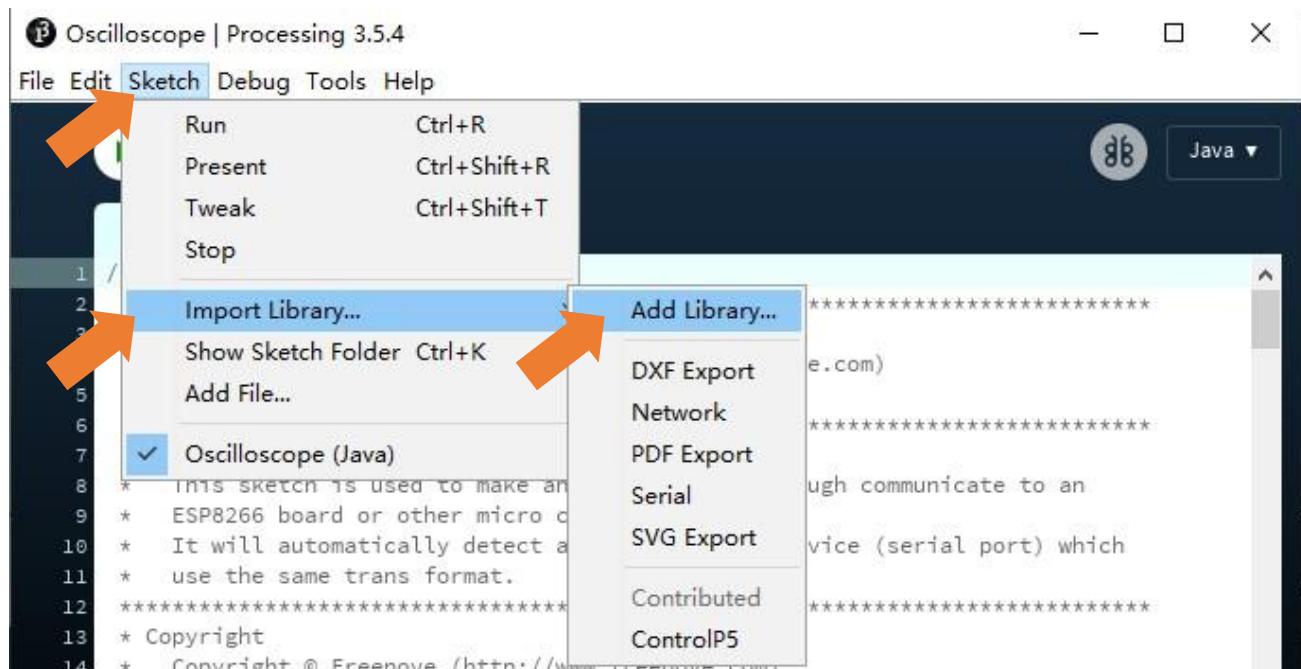
Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

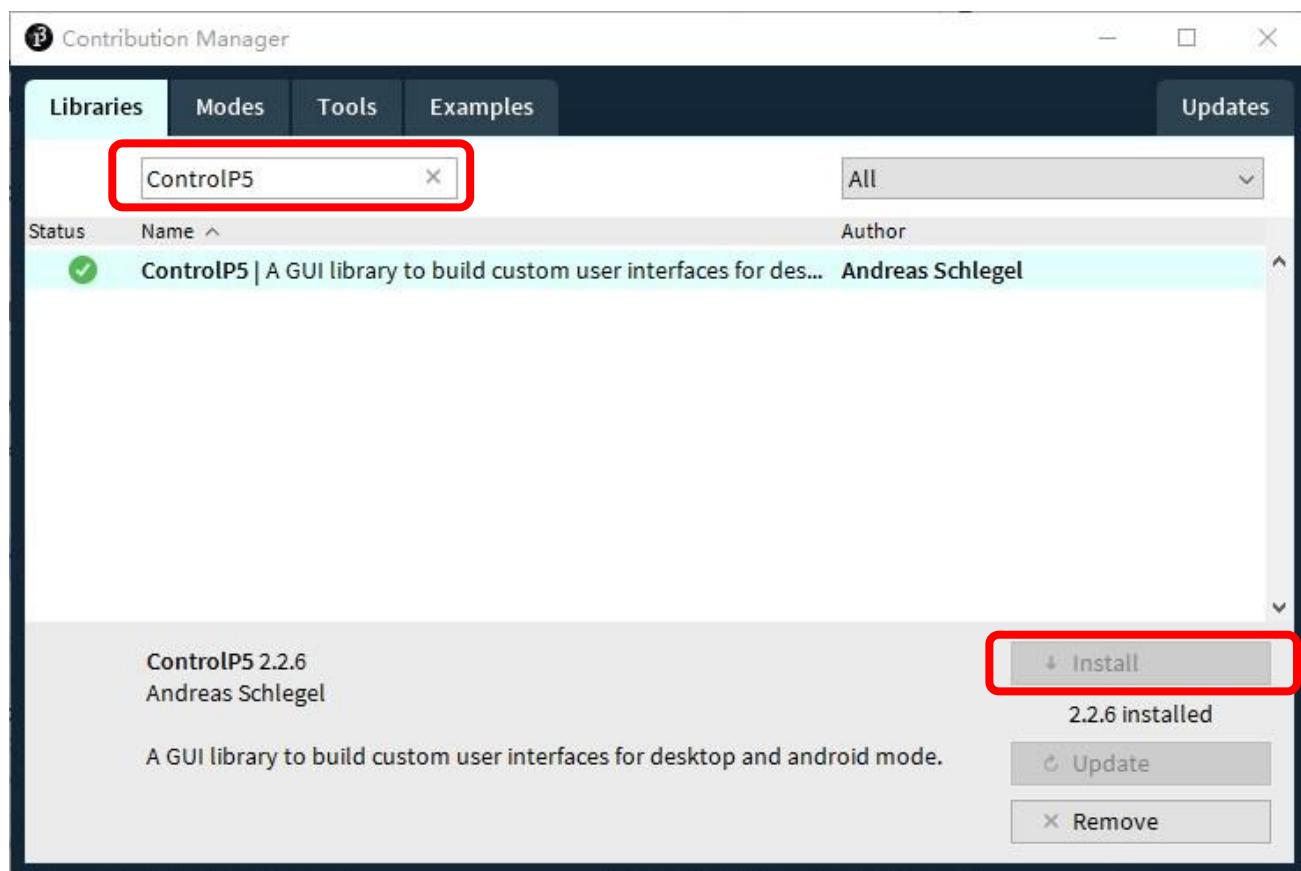
Select Libraries in the pop-up window.



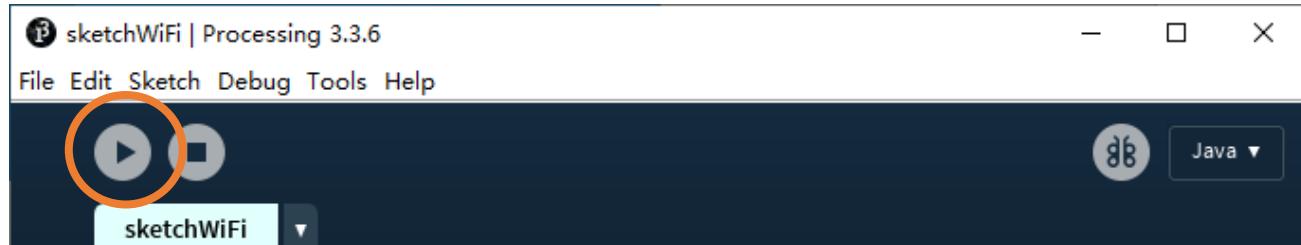
Input "ControlP5" in the searching box, and then select the option as below. Click "Install" and wait for the installation to finish.

You can also click Add Library under 'Import Library' under 'Sketch'.

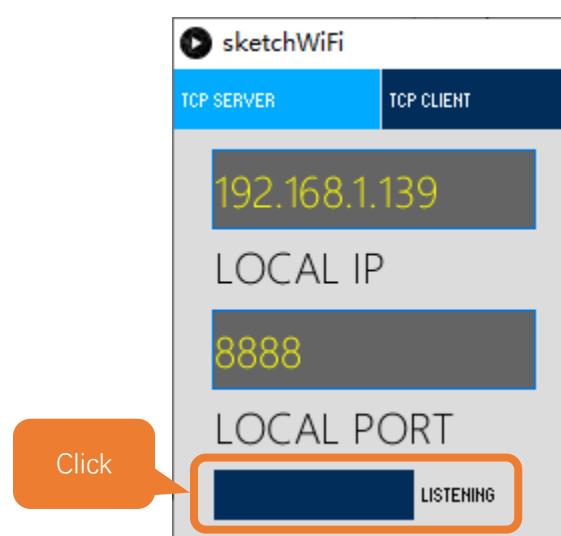




Before running the Code, please open "sketchWiFi.pde." first, and click "Run".



The newly pop up window will use the computer's IP address by default and open a data monitor port. Click "Listening".





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

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “13.1_TCP_as_Client” and double click “13.1_TCP_as_Client.py”.

Before clicking “Run current script”, please modify the name and password of your router and fill in the “host” and “port” according to the **IP information in processing app** shown in the box below:

13.1_TCP_as_Client

```

1 import network
2 import socket
3 import time
4
5 ssidRouter      = "*****"          #Enter the router name
6 passwordRouter = "*****"          #Enter the router password
7 host            = "192.168.1.139"    #input the remote server
8 port             = 8888            #input the remote port
9
10 wlan=None
11 s=None
12
13 def connectWifi(ssid,passwd):
14     global wlan
15     wlan=network.WLAN(network.STA_IF)
16     wlan.active(True)
17     wlan.disconnect()
18     wlan.connect(ssid,passwd)
19     while(wlan.ifconfig()[0]=='0.0.0.0'):
20         time.sleep(1)
21     return True
22 try:
23     connectWifi(ssidRouter,passwordRouter)

```

Click “Run current script” and in “Shell”, you can see PICO W automatically connects to sketchWiFi.

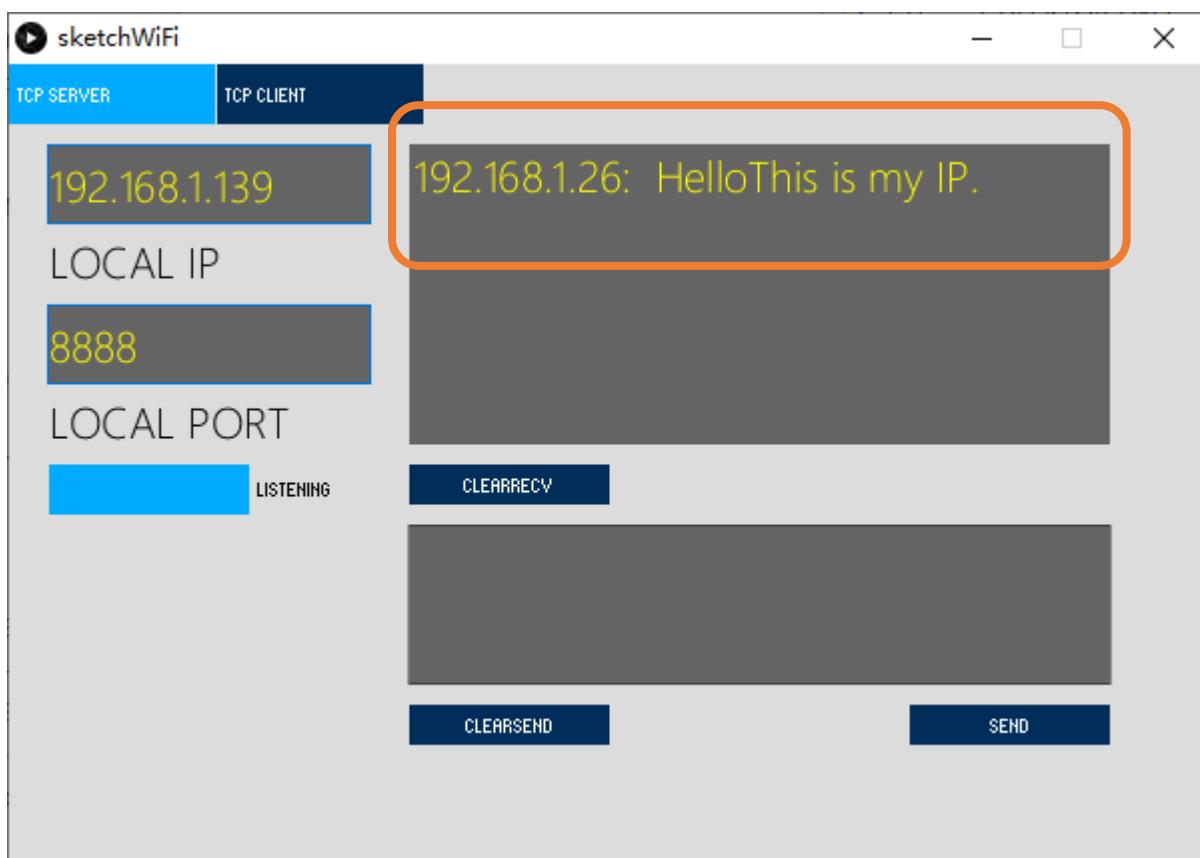


If you do not click "Listening" for sketchWiFi, PICO W will fail to connect and will print information as follows:

```
Shell < />
>>>

MicroPython v1.19.1 on 2022-09-26; Raspberry Pi Pico W with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
TCP close, please reset!
>>>
```

PICO W connects with TCP SERVER, and TCP SERVER receives messages from PICO W, as shown in the figure below.



At this point, you can send data to Pico W through sketchWiFi. Pico W will send the received data back to sketchWiFi after receiving it.

The following is the program code:

```
1 import network
2 import socket
3 import time
4
5 ssidRouter      = "*****"      #Enter the router name
6 passwordRouter = "*****"      #Enter the router password
7 host           = "*****"      #input the remote server
8 port           = 8888          #input the remote port
9
```

```

10 wlan=None
11 s=None
12
13 def connectWifi(ssid,passwd):
14     global wlan
15     wlan= network.WLAN(network.STA_IF)
16     wlan.active(True)
17     wlan.disconnect()
18     wlan.connect(ssid,passwd)
19     while(wlan.ifconfig()[0]=='0.0.0.0'):
20         time.sleep(1)
21     return True
22 try:
23     connectWifi(ssidRouter,passwordRouter)
24     s = socket.socket()
25     s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
26     s.connect((host, port))
27     print("TCP Connected to:", host, ":", port)
28     s.send('Hello')
29     s.send('This is my IP.')
30     while True:
31         data = s.recv(1024)
32         if(len(data) == 0):
33             print("Close socket")
34             s.close()
35             break
36         print(data)
37         ret=s.send(data)
38 except:
39     print("TCP close, please reset!")
40     if (s):
41         s.close()
42     wlan.disconnect()
43     wlan.active(False)

```

Import network、socket、time modules.

```

1 import network
2 import socket
3 import time

```

Enter the actual router name, password, remote server IP address, and port number.

```

5 ssidRouter      = "*****"      #Enter the router name
6 passwordRouter = "*****"      #Enter the router password
7 host           = "*****"      #input the remote server
8 port           = 8888          #input the remote port

```

Connect specified Router until it is successful.

```

13 def connectWifi(ssid,passwd):
14     global wlan
15     wlan= network.WLAN(network.STA_IF)
16     wlan.active(True)
17     wlan.disconnect()
18     wlan.connect(ssid,passwd)
19     while(wlan.ifconfig()[0]=='0.0.0.0'):
20         time.sleep(1)
21     return True

```

Connect router and then connect it to remote server.

```

23 connectWifi(ssidRouter,passwordRouter)
24 s = socket.socket()
25 s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
26 s.connect((host,port))
27 print("TCP Connected to:", host, ":", port)

```

Send messages to the remote server, receive the messages from it and print them out, and then send the messages back to the server.

```

28 s.send('Hello')
29 s.send('This is my IP.')
30 while True:
31     data = s.recv(1024)
32     if(len(data) == 0):
33         print("Close socket")
34         s.close()
35         break
36     print(data)
37     ret=s.send(data)

```

If an exception occurs in the program, for example, the remote server is shut down, execute the following program, turn off the socket function, and disconnect the WiFi.

```

39 print("TCP close, please reset!")
40 if (s):
41     s.close()
42     wlan.disconnect()
43     wlan.active(False)

```

Reference

Class socket

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

socket([af, type, proto]): Create a socket.

af: address

socket.AF_INET: IPv4

socket.AF_INET6: IPv6

type: type

socket.SOCK_STREAM : TCP stream

socket.SOCK_DGRAM : UDP datagram

socket.SOCK_RAW : Original socket

socket.SO_REUSEADDR : socket reusable

proto: protocol number

socket.IPPROTO_TCP: TCPmode

socket.IPPROTO_UDP: UDPmode

socket.setsockopt(level, optname, value): Set the socket according to the options.

Level: Level of socket option

socket.SOL_SOCKET: Level of socket option. By default, it is 4095.

optname: Options of socket

socket.SO_REUSEADDR: Allowing a socket interface to be tied to an address that is already in use.

value: The value can be an integer or a bytes-like object representing a buffer.

socket.connect(address): To connect to server.

Address: Tuple or list of the server's address and port number

send(bytes): Send data and return the bytes sent.

recv(bufsize): Receive data and return a bytes object representing the data received.

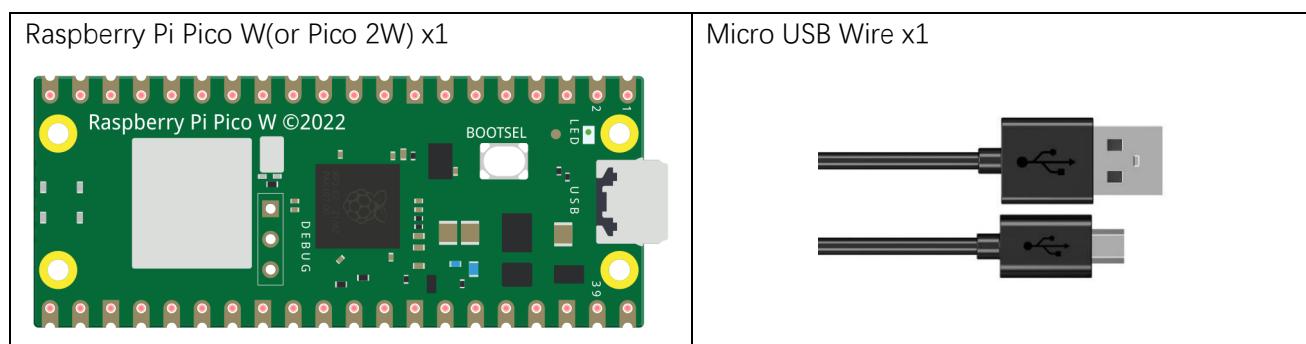
close(): Close socket.

To learn more please visit: <http://docs.micropython.org/en/latest/>

Project 13.2 as Server

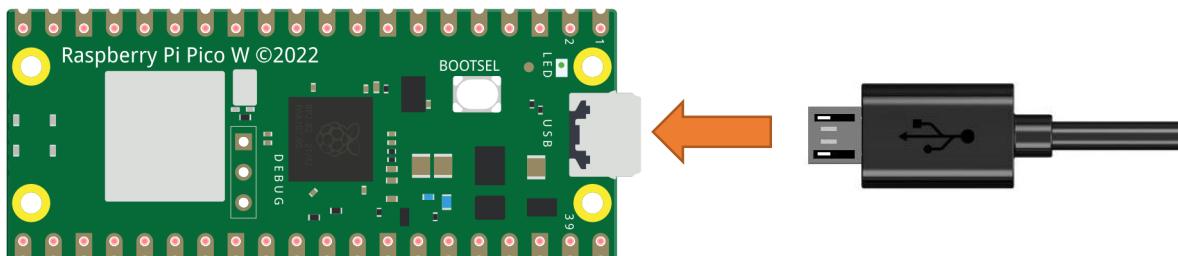
In this section, PICO W is used as a Server to wait for the connection and communication with Client on the same LAN.

Component List



Circuit

Connect Pico W to the computer using the USB cable.



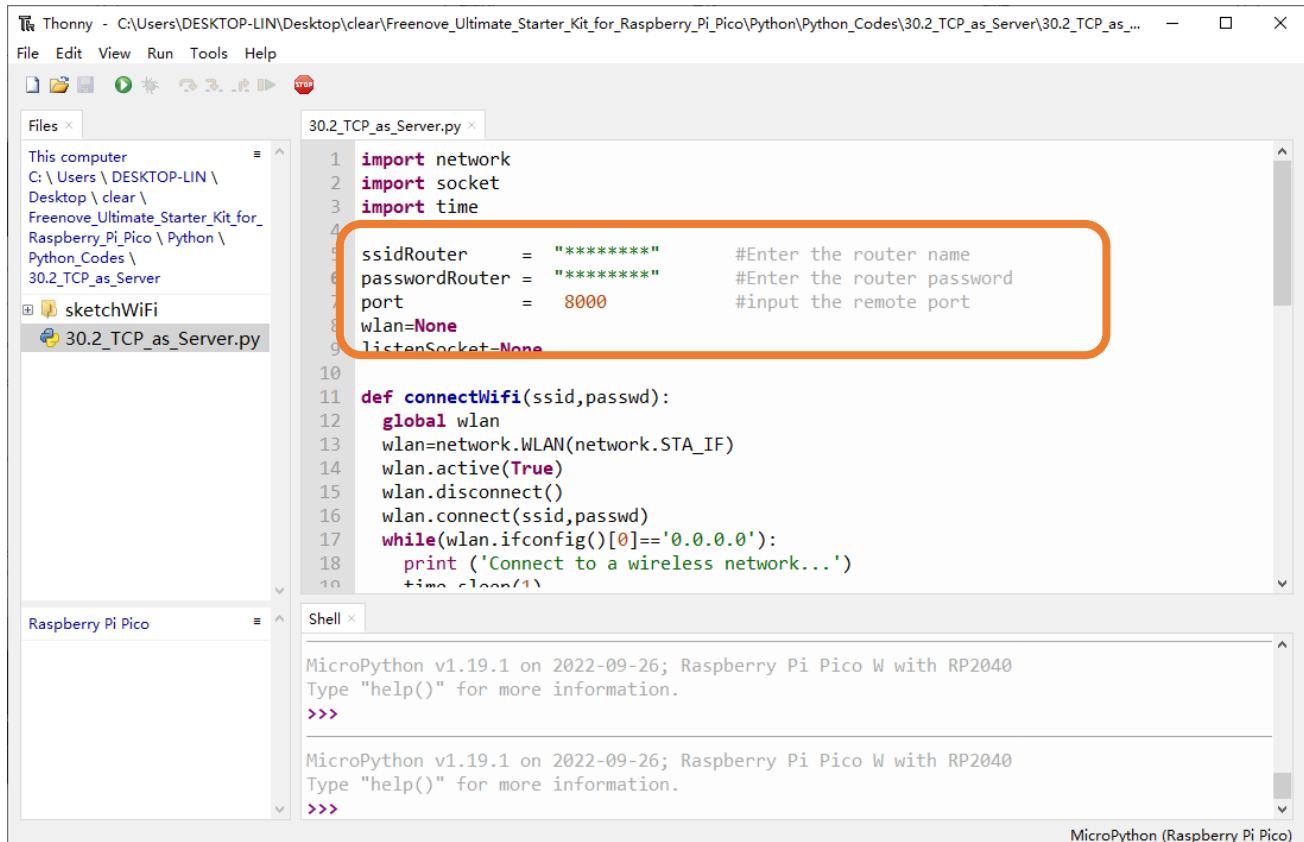
Code

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

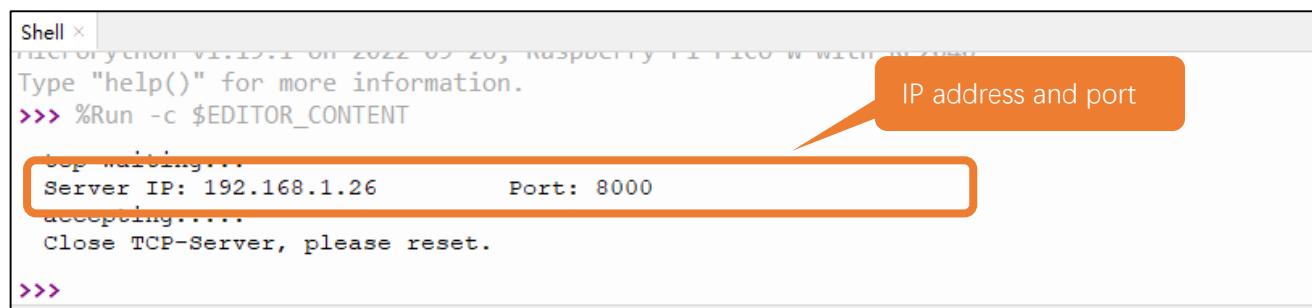
Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “13.2_TCP_as_Server” and double click “13.2_TCP_as_Server.py”.

Before clicking “Run current script”, please modify the name and password of your router shown in the box below.

13.2_TCP_as_Server



After making sure that the router's name and password are correct, click “Run current script” and in “Shell”, you can see a server opened by the PICO W waiting to connecting to other network devices.

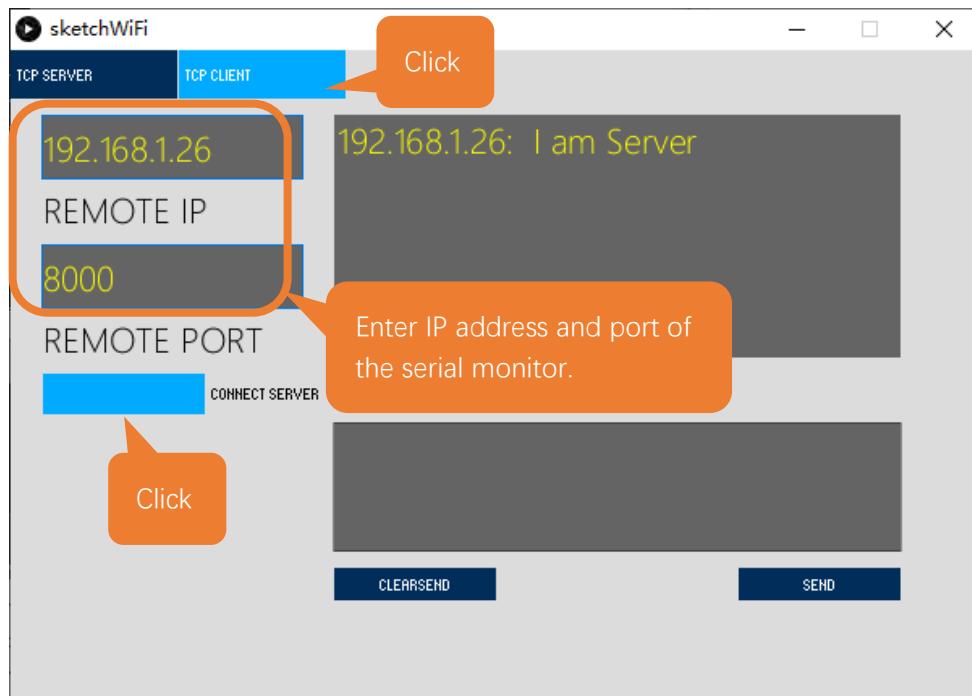


Processing:

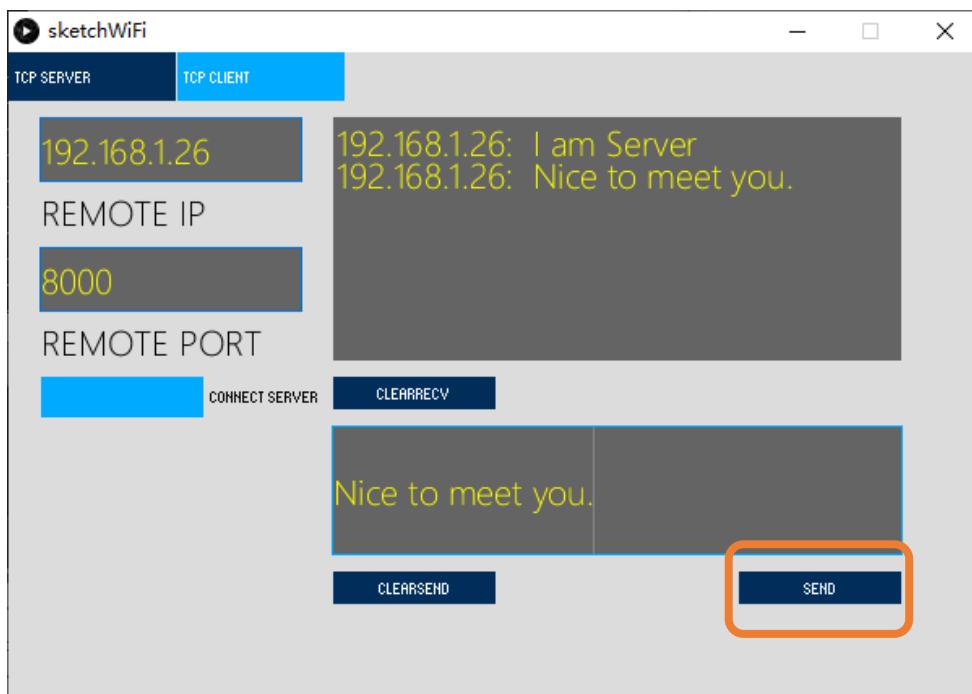
Open the

"Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico/Codes/MicroPython_Codes/13.2_TCP_as_Server/sketchWiFi/sketchWiFi.pde".

Based on the message printed in "Shell", enter the correct IP address and port when processing, and click to establish a connection with PICO W to communicate.



You can enter any information in the "Send Box" of sketchWiFi. Click "Send" and PICO W will print the received messages to "Shell" and send them back to sketchWiFi.



```

Shell x
Close TCP-Server, please reset.

>>> %Run -c $EDITOR_CONTENT

tcp waiting...
Server IP: 192.168.1.26          Port: 8000
accepting.....
('192.168.1.139', 59648) connected
b'Nice to meet you.'

```

The following is the program code:

```

1 import network
2 import socket
3 import time
4
5 ssidRouter      = "*****"           #Enter the router name
6 passwordRouter = "*****"           #Enter the router password
7 port            = 8000              #input the remote port
8 wlan            = None
9 listenSocket    = None
10
11 def connectWifi(ssid,passwd):
12     global wlan
13     wlan=network.WLAN(network.STA_IF)
14     wlan.active(True)
15     wlan.disconnect()
16     wlan.connect(ssid,passwd)
17     while(wlan.ifconfig()[0]=='0.0.0.0'):
18         time.sleep(1)
19     return True
20
21 try:
22     connectWifi(ssidRouter,passwordRouter)
23     ip=wlan.ifconfig()[0]
24     listenSocket = socket.socket()
25     listenSocket.bind((ip,port))
26     listenSocket.listen(1)
27     listenSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
28     print(' tcp waiting... ')
29     while True:
30         print("Server IP:",ip,"\tPort:",port)
31         print("accepting..... ")
32         conn,addr = listenSocket.accept()
33         print(addr,"connected")
34         break
35         conn.send(' I am Server' )

```

```

36     while True:
37         data = conn.recv(1024)
38         if(len(data) == 0):
39             print("close socket")
40             listenSocket.close()
41             wlan.disconnect()
42             wlan.active(False)
43             break
44         else:
45             print(data)
46             ret = conn.send(data)
47     except:
48         print("Close TCP-Server, please reset.")
49         if(listenSocket):
50             listenSocket.close()
51             wlan.disconnect()
52             wlan.active(False)

```

Call function `connectWifi()` to connect to router and obtain the dynamic IP that it assigns to PICO W.

22	<code>connectWifi(ssidRouter, passwordRouter)</code>
23	<code>ip=wlan.ifconfig()[0]</code>

Open the socket server, bind the server to the dynamic IP, and open a data monitoring port.

24	<code>listenSocket = socket.socket()</code>
25	<code>listenSocket.bind((ip, port))</code>
26	<code>listenSocket.listen(1)</code>
27	<code>listenSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)</code>

Print the server's IP address and port, monitor the port and wait for the connection of other network devices.

29	<code>while True:</code>
30	<code> print("Server IP:", ip, "\tPort:", port)</code>
31	<code> print("accepting.....")</code>
32	<code> conn, addr = listenSocket.accept()</code>
33	<code> print(addr, "connected")</code>
34	<code> break</code>

Each time receiving data, print them in "Shell" and send them back to the client.

36	<code>while True:</code>
37	<code> data = conn.recv(1024)</code>
38	<code> if(len(data) == 0):</code>
39	<code> print("close socket")</code>
40	<code> listenSocket.close()</code>
41	<code> wlan.disconnect()</code>
42	<code> wlan.active(False)</code>
43	<code> break</code>
44	<code> else:</code>
45	<code> print(data)</code>
46	<code> ret = conn.send(data)</code>



If the client is disconnected, close the server and disconnect WiFi.

```
47 except:  
48     print("Close TCP-Server, please reset.")  
49     if(listenSocket):  
50         listenSocket.close()  
51     wlan.disconnect()  
52     wlan.active(False)
```

Chapter 14 Control LED with Web (Only for Pico W/Pico 2W)

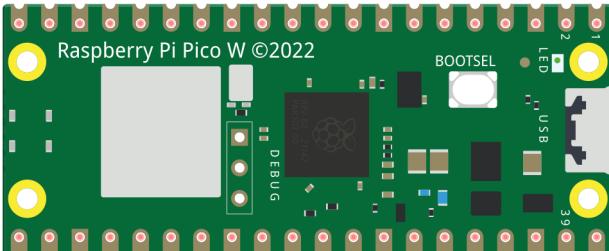
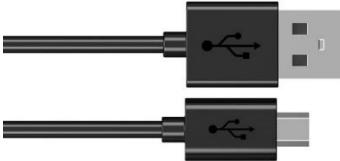
If you have Pico in your hand, please change it to Pico W before continuing to learn.

In this chapter, we will use PICO W to make a simple smart home. We will learn how to control LED lights through web pages.

Project 14.1 Control_LED_through_Web

In this project, we need to build a Web Service and then use PICO W to control the LED through the Web browser of the phone or PC. Through this example, you can remotely control the appliances in your home to achieve smart home.

Component List

Raspberry Pi Pico W(or Pico 2W) x1	Micro USB Wire x1
 A photograph of a Raspberry Pi Pico W development board. It is a green printed circuit board with a central Broadcom chip. Key components visible include the chip, a USB-C port, a microUSB port, a BOOTSEL button, and several yellow circular pads. The text "Raspberry Pi Pico W ©2022" is printed on the board.	 A diagram showing two ends of a standard microUSB cable. Each end has a black plastic housing with a metal connector. The cable is shown in a straight configuration.



Component knowledge

HTML

Hypertext Markup Language (HTML) is a standard Markup Language for creating web pages. It includes a set of tags that unify documents on the network and connect disparate Internet resources into a logical whole. HTML text is descriptive text composed of HTML commands that describe text, graphics, animations, sounds, tables, links, etc. The extension of the HTML file is HTM or HTML. Hypertext is a way to organize information. It uses hyperlinks to associate words and charts in Text with other information media. These related information media may be in the same Text, other files, or files located on a remote computer. This way of organizing information connects the information resources distributed in different places, which is convenient for people to search and retrieve information.

The nature of the Web is hypertext Markup Language (HTML), which can be combined with other Web technologies (e.g., scripting languages, common gateway interfaces, components, etc.) to create powerful Web pages. Thus, Hypertext Markup Language (HTML) is the foundation of World Wide Web (Web) programming, that is, the World Wide Web is based on hypertext. Hypertext Markup Language is called hypertext Markup language because the text contains so-called "hyperlink" points.

You can build your own WEB site using HTML, which runs on the browser and is parsed by the browser.

Example analysis is shown in the figure below:



<!DOCTYPE html>: Declare it as an HTML5 document

<html>: Is the root element of an HTML page

<head>: Contains meta data for the document, such as < meta charset="utf-8" > Define the web page encoding format to UTF-8.

<title>: Notes the title of the document

<body>: Contains visible page content

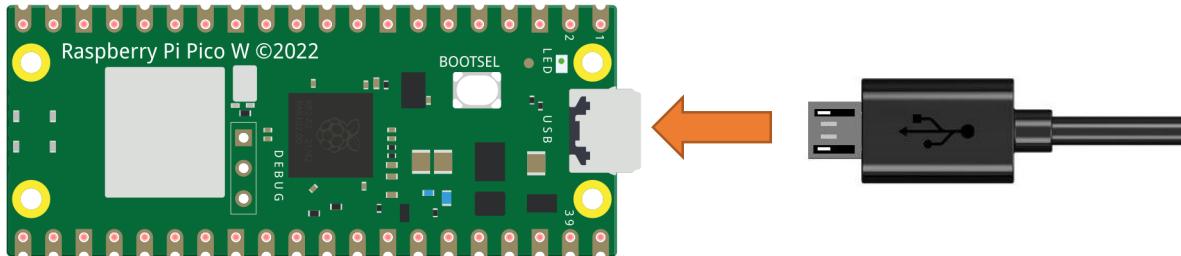
<h1>: Define a big heading

<p>: Define a paragraph

For more information, please visit: <https://developer.mozilla.org/en-US/docs/Web/HTML>

Circuit

Connect Pico W to the computer using the USB cable.



Code

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

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “14.1_Control_LED_through_Web”, and double click “14.1_Control_LED_through_Web.py”.

14.1_Control_LED_through_Web

```

import machine
import time
import socket
import network

# set led pin
led = machine.Pin("LED", machine.Pin.OUT)

ssid = '*****'          #Enter the router name
password = '*****'       #Enter the router password

wifi_status = network.WLAN(network.STA_IF)
wifi_status.disconnect()
wifi_status.active(True)
wifi_status.connect(ssid, password)

def WebPage():
    if led.value() == 1:
        gpio_state = 'OFF'
    else:
        gpio_state = 'ON'

    # html code ...
    html = """
<html>
    <head>
        <title>Pico W Web Server</title>
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link rel="icon" href="data:,>">
        <style>
            html{font-family: Helvetica; display:inline-block; margin: 0px auto;
            h1{color: #0F3376; padding: 2vh; font-size: 1.5em;}}


MicroPython v1.19.1 on 2022-09-26; Raspberry Pi Pico W with RP2040  
Type "help()" for more information.  
=>


```

Because the names and passwords of routers in various places are different, before the Code runs, users need to enter the correct router's name and password in the box as shown in the illustration above.

After making sure the router name and password are entered correctly, compile and upload codes to PICO W, wait for PICO W to connect to your router and print the IP address assigned by the router to PICO W in "Shell".

```

MicroPython v1.19.1 on 2022-09-26; Raspberry Pi Pico W with RP2040
Type "help()" for more information.
=> %Run -c $EDITOR_CONTENT

Connect to a wireless network...
Connect to a wireless network...
Connect to a wireless network...
Wifi connect successful
('192.168.1.26', '255.255.255.0', '192.168.1.1', '192.168.1.1')

```

When PICO W successfully connects to "ssid", "Shell" displays the IP address assigned to PICO W by the router. Access <http://192.168.1.26> in a computer browser on the LAN. As shown in the following figure:



You can click the corresponding button to control the LED on and off.

The following is the program code:

```
1 import machine
2 import time
3 import socket
4 import network
5
6 # set led pin
7 led = machine.Pin("LED", machine.Pin.OUT)
8
9 ssid = '*****'          #Enter the router name
10 password = '*****'      #Enter the router password
11
12 wifi_status = network.WLAN(network.STA_IF)
13 wifi_status.disconnect()
14 wifi_status.active(True)
15 wifi_status.connect(ssid, password)
16
17 def WebPage():
18     if led.value() == 1:
19         gpio_state = 'On'
20     else:
21         gpio_state = 'OFF'
22
23     # html code ...
24     html = """
25         <html>
26             <head>
27                 <title>Pico W Web Server</title>
```

```
28         <meta name="viewport" content="width=device-width, initial-scale=1">
29         <link rel="icon" href="data:, ">
30         <style>
31             html{font-family: Helvetica; display:inline-block; margin: 0px auto; text-
32             align: center;}
33             h1{color: #0F3376; padding: 2vh;}
34             p{font-size: 1.5rem;}
35             button{display: inline-block; background-color: #4286f4; border: none; border-
36             radius: 4px; color: white; padding: 16px 40px; text-decoration: none; font-size: 30px; margin:
37             2px; cursor: pointer;}
38             button2{background-color: #4286f4;}
39         </style>
40     </head>
41     <body> <h1>Pico W Web Server</h1>
42         <p>GPIO state: <strong>"" + gpio_state + ""</strong></p>
43         <p><a href="/?led=on"><button class="button">ON</button></a></p>
44         <p><a href="/?led=off"><button class="button button2">OFF</button></a></p>
45     </body>
46 </html>
47 """
48
49     return html
50
51 # check wifi connected
52 while wifi_status.isconnected() == False:
53     time.sleep(1)
54     print('Connect to a wireless network... ')
55
56 # if connected
57 print('Wifi connect successful')
58 print(wifi_status.ifconfig())
59
60
61 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
62 s.bind(('', 80))
63 s.listen(5)
64
65 try:
66     while True:
67         conn, addr = s.accept()
68         #print('Connection: %s' % str(addr))
69         req = conn.recv(1024)
70         req = str(req)
71         print('Connect = %s' % req)
72         led_on = req.find('/?led=on')
73         led_off = req.find('/?led=off')
74         if led_on == 6:
75             gpio_state = 1
76         else:
77             gpio_state = 0
78         html = <html>
79             <head>
80                 <meta name="viewport" content="width=device-width, initial-scale=1">
81                 <link rel="icon" href="data:, ">
82                 <style>
83                     html{font-family: Helvetica; display:inline-block; margin: 0px auto; text-
84                     align: center;}
85                     h1{color: #0F3376; padding: 2vh;}
86                     p{font-size: 1.5rem;}
87                     button{display: inline-block; background-color: #4286f4; border: none; border-
88                     radius: 4px; color: white; padding: 16px 40px; text-decoration: none; font-size: 30px; margin:
89                     2px; cursor: pointer;}
90                     button2{background-color: #4286f4;}
91                 </style>
92             </head>
93             <body> <h1>Pico W Web Server</h1>
94                 <p>GPIO state: <strong>"" + gpio_state + ""</strong></p>
95                 <p><a href="/?led=on"><button class="button">ON</button></a></p>
96                 <p><a href="/?led=off"><button class="button button2">OFF</button></a></p>
97             </body>
98 </html>
99 """
100
101     return html
```

```

69         print('LED ON')
70         led.value(1)
71     else:
72         print('LED OFF')
73         led.value(0)
74     if led.value() == 1:
75         gpio_state = 'On'
76     else:
77         gpio_state = 'OFF'
78     response = WebPage()
79     conn.send('HTTP/1.1 200 OK\n')
80     conn.send('Content-Type: text/html\n')
81     conn.send('Connection: close\n\n')
82     conn.sendall(response)
83     conn.close()
84 except:
85     conn.close()
86     s.close()
87     pass

```

Import socket module and Import network module.

```

3 import socket
4 import network

```

Enter correct AP name and password.

```

3 ssid = '*****'          #Enter the router name
4 password = '*****'      #Enter the router password

```

Set PICO W in Station mode and connect it to your router.

```

12 wifi_status = network.WLAN(network.STA_IF)
13 wifi_status.disconnect()
14 wifi_status.active(True)
15 wifi_status.connect(ssid, password)

```

"Shell" displays the IP address assigned to PICO W.

```

53 print(wifi_status.ifconfig())

```

Click the button on the web page to control the LED light on and off.

```

68     if led_on == 6:
69         print('LED ON')
70         led.value(0)
71     else:
72         print('LED OFF')
73         led.value(1)
74     if led.value() == 1:
75         gpio_state = 'OFF'
76     else:
77         gpio_state = 'ON'

```



Chapter 15 Bluetooth (Only for Pico W/Pico 2W)

Note: As the official MicroPtyhon firmware supports only Bluetooth Low Energy (BLE) at the time of this writing, code for classic Bluetooth is not included in this tutorial.

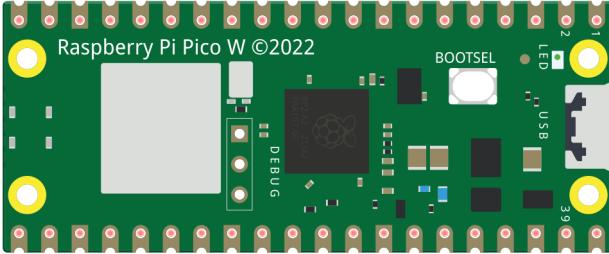
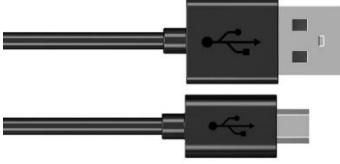
In June 2023, Raspberry Pi Official has updated to add Bluetooth support to the Pico W.

Pico W's Bluetooth 5.2 supports Bluetooth Classic and Bluetooth Low Energy (BLE) functionality. At the beginning of thie chapter, we will learn the Pico W's Bluetooth function.

If you have Pico in your hand, please change it to Pico W before continuing to learn.

Project 15.1 Bluetooth Low Energy Data Passthrough

Component List

Raspberry Pi Pico W(or Pico 2W) x1	Micro USB Wire x1
	

Component knowledge

Pico W's wireless functionality is provided by the Infineon CYW43439 device, which contains a 2.4 GHz radio providing both 802.11n Wi-Fi and Bluetooth 5.2, supporting Bluetooth Classic and Bluetooth Low Energy (BLE) functionality.

For simple data transfer, there are two modes:

Master mode

A device works in master mode can connect to one or more slave devices.

We can search and select the slave devices nearby to connect.

When a device initiates a connection request in master mode, it requires information about other Bluetooth devices, including their addresses and pairing keys.

Once the devices are paired, a direct connection can be established.

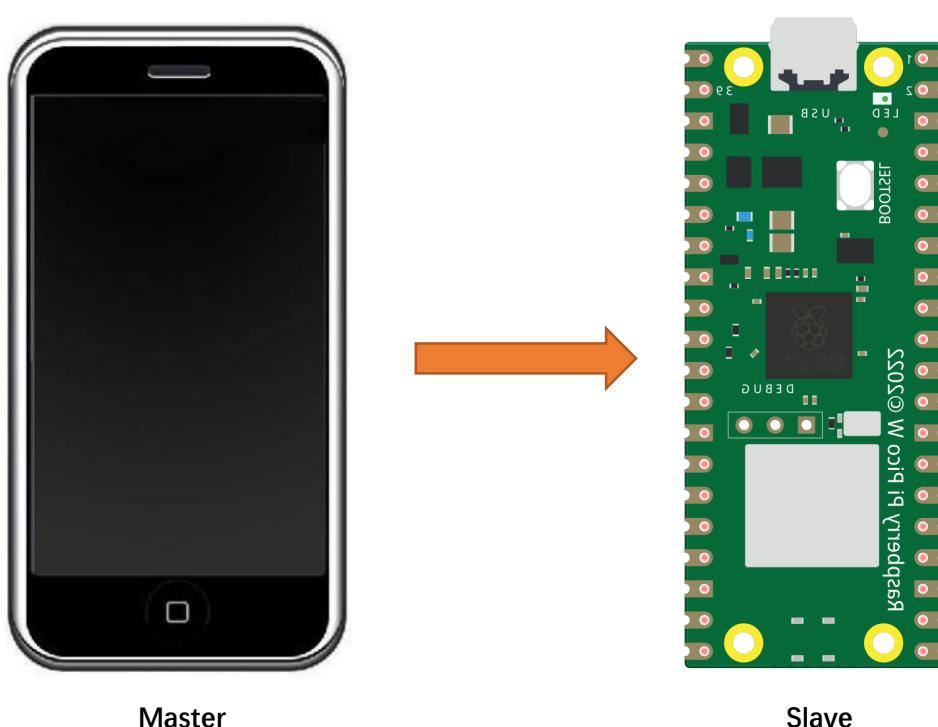
Slave mode

A Bluetooth module operating in slave mode can only receive connection requests from a master device and cannot actively initiate connections.

After establishing a connection with a master device, it can either send or receive data.

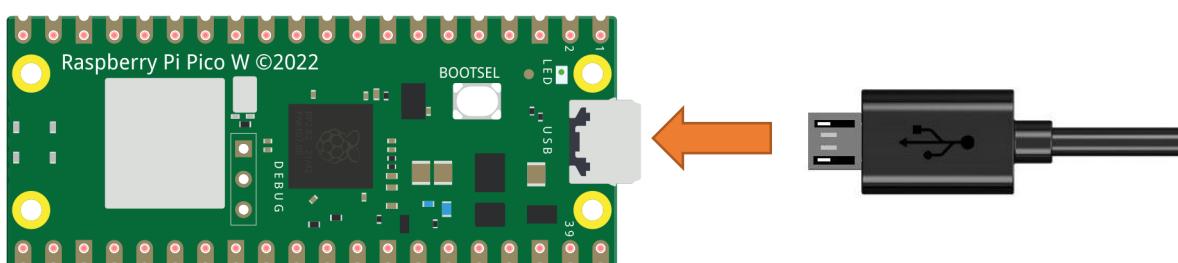
Any concerns? ✉ support@freenove.com

Bluetooth devices can interact with each other, with one in master mode and the other in slave mode. During data communication, the master device searches for and selects nearby devices to connect with. Once a connection is made, data can be exchanged between the devices. In the case of data exchange between a smartphone and a Raspberry Pi Pico W, the smartphone typically operates in master mode, while the Raspberry Pi Pico W functions in slave mode.



Circuit

Connect Pico W to the computer using the USB cable.





LightBlue

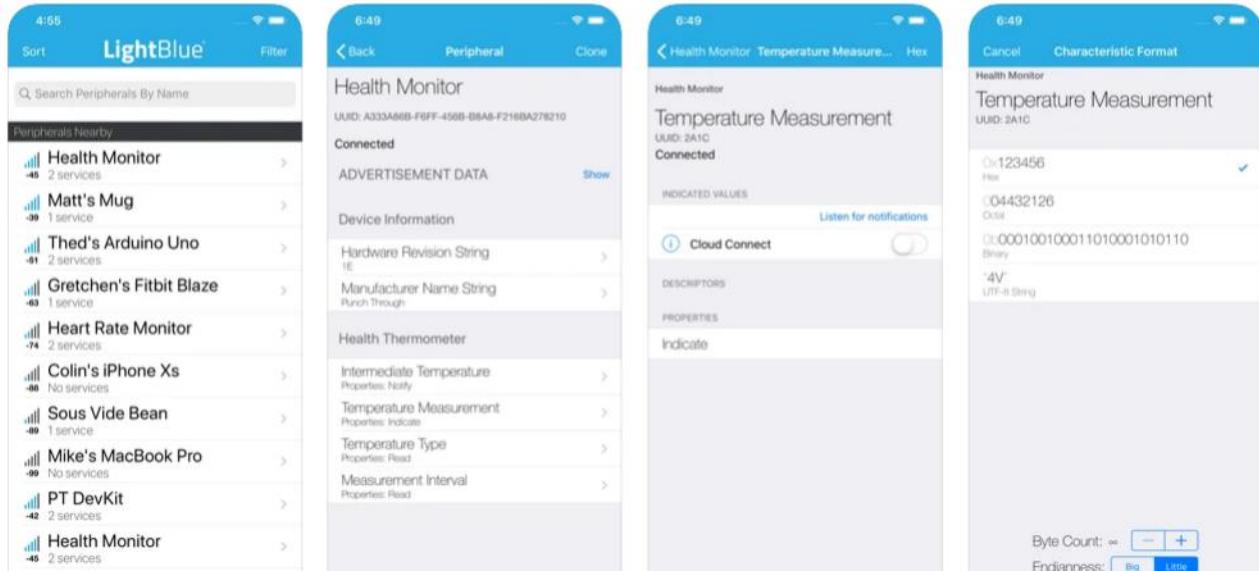
If Serial Bluetooth cannot be installed on your phone, try LightBlue. If you do not have this software installed on your phone, you can refer to this link:

<https://apps.apple.com/us/app/lightblue/id557428110?platform=iphone>



LightBlue® [4+]
The go-to BLE development tool
Punch Through
★★★★★ 4.0 • 4 Ratings
Free

Screenshots Mac iPhone iPad

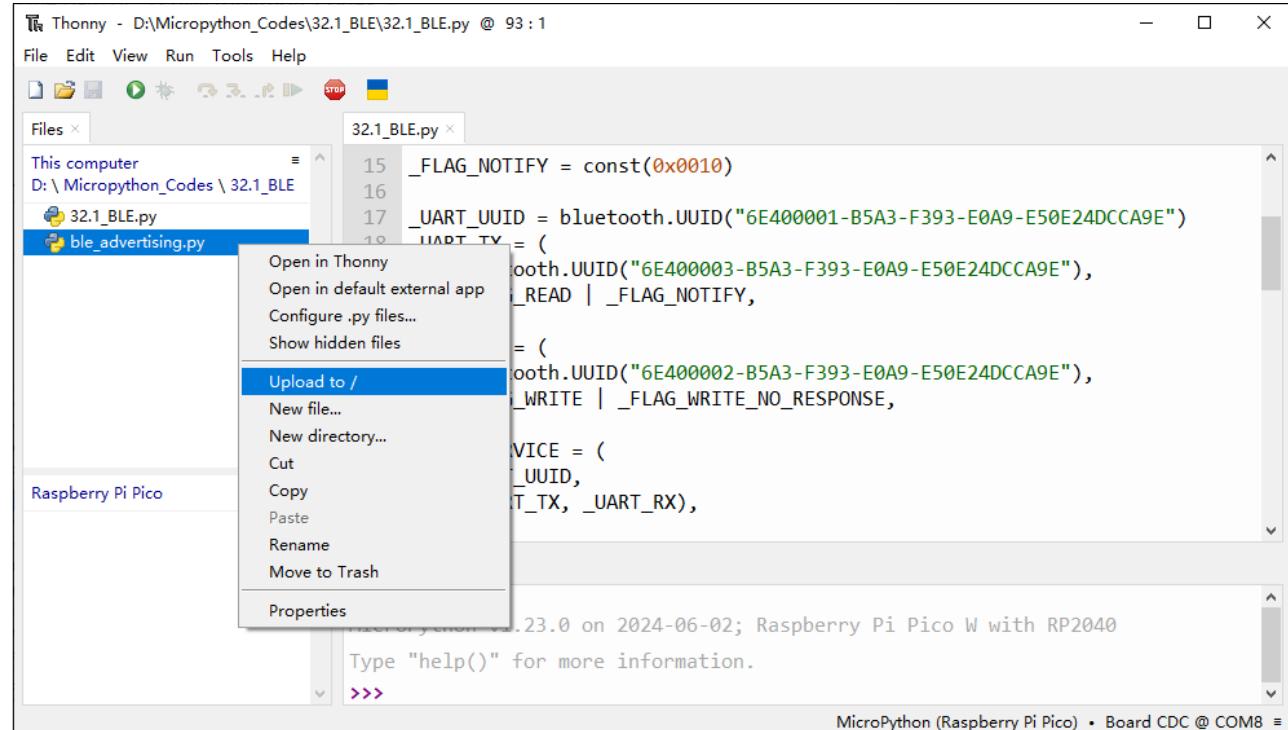


Code

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

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “15.1_BLE”. **Right click** “ble_advertising.py” and select “Upload to /” to upload it to Pico W. Then, double click “ble.py” to open the file.

15.1_BLE



Click the Run button to run BLE.py.

```

Thonny - D:\Micropython_Codes\32.1_BLE\32.1_BLE.py @ 93 : 1
File Edit View Run Tools Help
Files x
This computer D:\ Micropython_Codes \ 32.1_BLE
32.1_BLE.py ble_advertising.py
Raspberry Pi Pico
ble_advertising.py
32.1_BLE.py x
32.1_BLE.py x
19     bluetooth.UUID("6E400003-B5A3-F393-E0A9-E50E24DCCA9E"),
20     _FLAG_READ | _FLAG_NOTIFY,
21 )
22 _UART_RX = (
23     bluetooth.UUID("6E400003-B5A3-F393-E0A9-E50E24DCCA9E"),
24     _FLAG_WRITE | _FLAG_READ | _FLAG_NOTIFY,
25 )
26 _UART_SERVICE = (
27     _UART_UUID,
28     (_UART_RX, _UART_RX),
29 )
30
31 class BLESimplePeripheral:
32     def __init__(self, ble, name="Picow"):
33         self._ble = ble
34         self._ble.active(True)
35         self._ble.irq(self._irq)

Shell x
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Starting advertising
Please use LightBlue to connect to Picow.

MicroPython (Raspberry Pi Pico) • Board CDC @ COM17 =

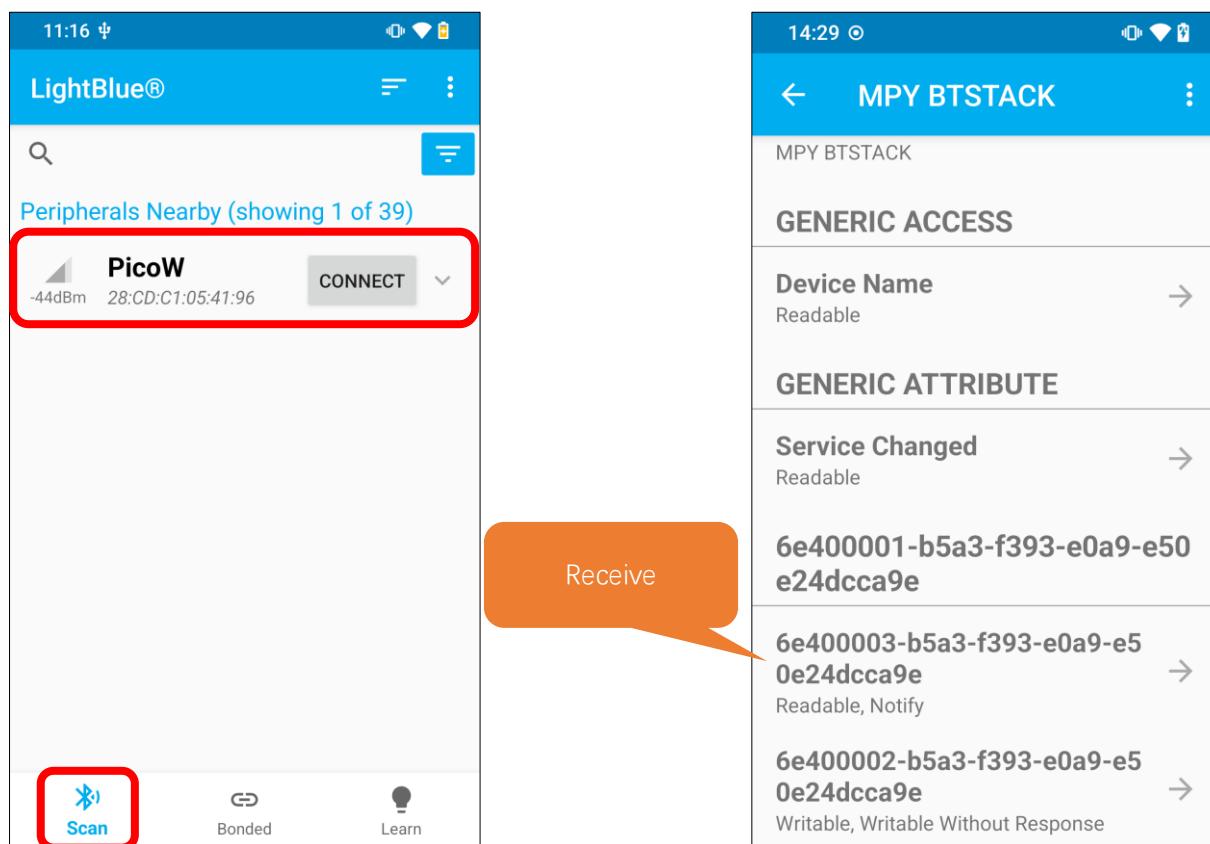
```

Note: If the Bluetooth name cannot be found, you can try modifying the current Bluetooth name.

Turn ON Bluetooth on your phone and open LightBlue APP.



At the device scan page, scroll down to refresh the devices nearby. Select PicoW to connect.

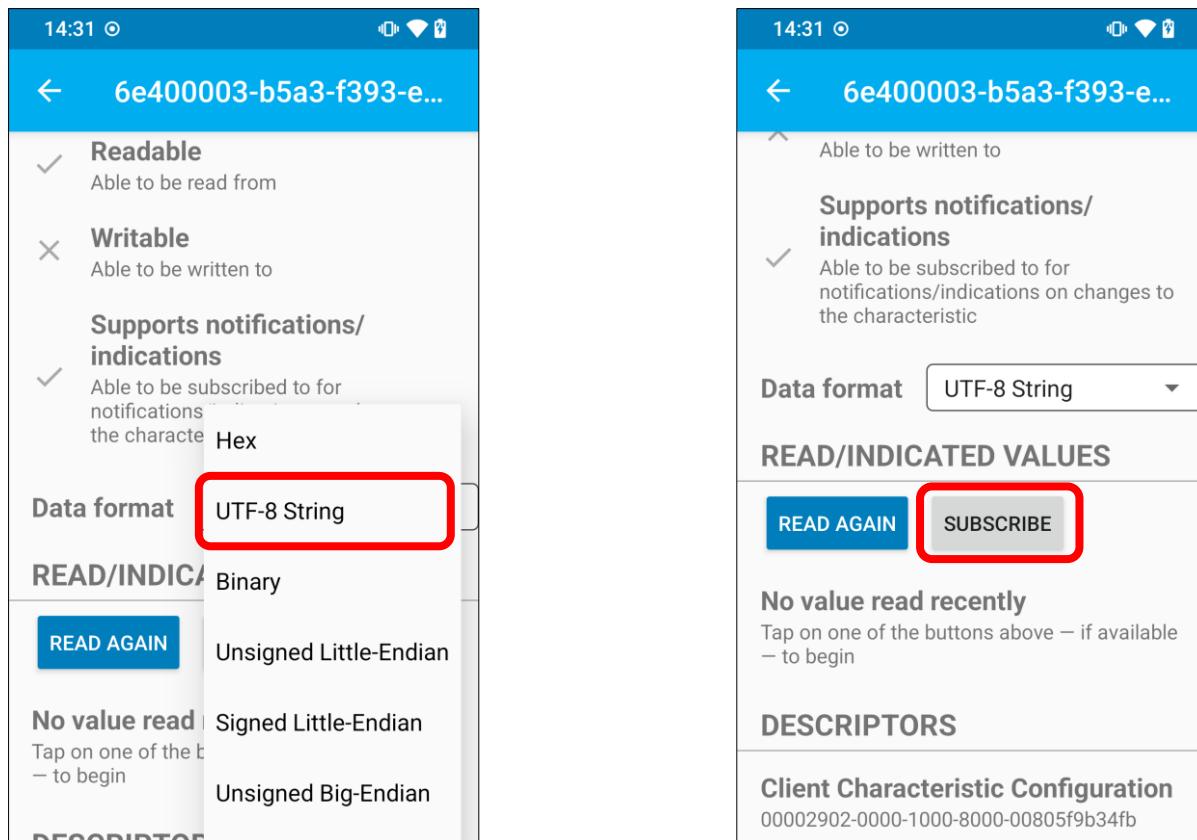


Once the Bluetooth connects successfully, it will print the messages as shown below.

```
>>> %Run -c $EDITOR_CONTENT
```

```
MPY: soft reboot
Starting advertising
Please use LightBlue to connect to PicoW.
New connection 64
Enter anything:
```

Click "Receive". Select the corresponding data format in the box on the right side of the data format, such as HEX hexadecimal, utf-string string, Binary, etc. Then click SUBSCRIBE.

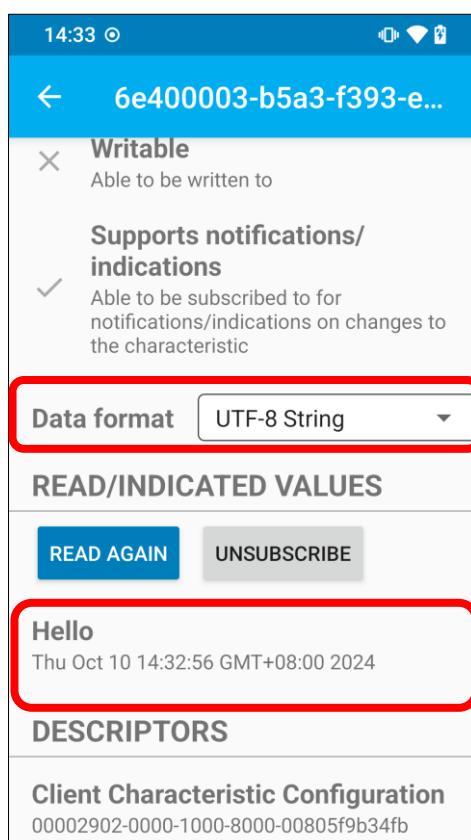


You can type “Hello” in Shell and press “Enter” to send.

```
>>> %Run -c $EDITOR_CONTENT
```

```
MPY: soft reboot
Starting advertising
Please use LightBlue to connect to PicoW.
New connection 64
Enter anything: Hello
Send: Hello
Enter anything:
```

Then you can see the mobile Bluetooth has received the message.



Similarly, you can select “Send” on your phone. Set Data format, and then enter anything in the sending box and click Write to send.

MPY BTSTACK

Device Name →
Readable

GENERIC ATTRIBUTE

Service Changed →
Readable

**6e400001-b5a3-f393-e0a9-e50
e24dcca9e**

**6e400003-b5a3-f393-e0a9-e5
0e24dcca9e** →
Readable, Notify

**6e400002-b5a3-f393-e0a9-e5
0e24dcca9e**

Writable, Writable Without Response

Send

6e400002-b5a3-f393-e0a9-e...

Data format UTF-8 String

READ/INDICATED VALUES

READ AGAIN

No value read recently
Tap on one of the buttons above – if available – to begin

WRITTEN VALUES

\b, \f, \n, \r, \t, \x00, \u0000 and \000 escape sequences are supported.

hello **WRITE**

No value written recently
Input some data and tap on the "Write" button to begin

DESCRIPTORS

Any concerns? ✉ support@freenove.com



You can check the message from Bluetooth in “Shell”.

```
MPY: soft reboot
Starting advertising
Please use LightBlue to connect to PicoW.
New connection 64
Enter anything: Hello
Send: Hello
Enter anything:
Receive: hello
```

The following is the program code:

```
1 import bluetooth
2 import random
3 import struct
4 import time
5 from ble_advertising import advertising_payload
6 from micropython import const
7
8 _IRQ_CENTRAL_CONNECT = const(1)
9 _IRQ_CENTRAL_DISCONNECT = const(2)
10 _IRQ_GATTS_WRITE = const(3)
11
12 _FLAG_READ = const(0x0002)
13 _FLAG_WRITE_NO_RESPONSE = const(0x0004)
14 _FLAG_WRITE = const(0x0008)
15 _FLAG_NOTIFY = const(0x0010)
16
17 _UART_UUID = bluetooth.UUID("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")
18 _UART_TX = (
19     bluetooth.UUID("6E400003-B5A3-F393-E0A9-E50E24DCCA9E"),
20     _FLAG_READ | _FLAG_NOTIFY,
21 )
22 _UART_RX = (
23     bluetooth.UUID("6E400002-B5A3-F393-E0A9-E50E24DCCA9E"),
24     _FLAG_WRITE | _FLAG_WRITE_NO_RESPONSE,
25 )
26 _UART_SERVICE = (
27     _UART_UUID,
28     (_UART_TX, _UART_RX),
29 )
30
31 class BLESimplePeripheral:
32     def __init__(self, ble, name="PicoW"):
33         self._ble = ble
```

Any concerns? ✉ support@freenove.com

```
34         self._ble.active(True)
35         self._ble.irq(self._irq)
36         ((self._handle_tx, self._handle_rx),) =
37     self._ble.gatts_register_services((_UART_SERVICE,))
38         self._connections = set()
39         self._write_callback = None
40         self._payload = advertising_payload(name=name, services=[_UART_UUID])
41         self._advertise()
42
43     def _irq(self, event, data):
44         # Track connections so we can send notifications.
45         if event == _IRQ_CENTRAL_CONNECT:
46             conn_handle, _, _ = data
47             print("New connection", conn_handle)
48             self._connections.add(conn_handle)
49         elif event == _IRQ_CENTRAL_DISCONNECT:
50             conn_handle, _, _ = data
51             print("Disconnected", conn_handle)
52             self._connections.remove(conn_handle)
53             # Start advertising again to allow a new connection.
54             self._advertise()
55         elif event == _IRQ_GATTS_WRITE:
56             conn_handle, value_handle = data
57             value = self._ble.gatts_read(value_handle)
58             if value_handle == self._handle_rx and self._write_callback:
59                 self._write_callback(value)
60
61     def send(self, data):
62         for conn_handle in self._connections:
63             self._ble.gatts_notify(conn_handle, self._handle_tx, data)
64
65     def is_connected(self):
66         return len(self._connections) > 0
67
68     def _advertise(self, interval_us=500000):
69         print("Starting advertising")
70         self._ble.gap_advertise(interval_us, adv_data=self._payload)
71
72     def on_write(self, callback):
73         self._write_callback = callback
74
75
76     def demo():
77         ble = bluetooth.BLE()
```

```

78     p = BLESimplePeripheral(ble)
79
80     def on_rx(rx_data):
81         rx_data = rx_data.decode('utf-8').rstrip('\r\n')
82         print("\nReceive:", rx_data)
83     p.on_write(on_rx)
84     print("Please use LightBlue to connect to PicoW.")
85
86     while True:
87         if p.is_connected():
88             # Short burst of queued notifications.
89             tx_data = input("Enter anything: ")
90             print("Send: ", tx_data)
91             p.send(tx_data)
92     if __name__ == "__main__":
93         demo()
94

```

Define the specified UUID number for BLE vendor.

```

17     _UART_UUID = bluetooth.UUID("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")
18     _UART_TX = (
19         bluetooth.UUID("6E400003-B5A3-F393-E0A9-E50E24DCCA9E"),
20         _FLAG_READ | _FLAG_NOTIFY,
21     )
22     _UART_RX = (
23         bluetooth.UUID("6E400002-B5A3-F393-E0A9-E50E24DCCA9E"),
24         _FLAG_WRITE | _FLAG_WRITE_NO_RESPONSE,
25     )

```

Write a '_irq' function to manage BLE interrupt events

```

43     def _irq(self, event, data):
44         # Track connections so we can send notifications.
45         if event == _IRQ_CENTRAL_CONNECT:
46             conn_handle, _, _ = data
47             print("New connection", conn_handle)
48             self._connections.add(conn_handle)
49         elif event == _IRQ_CENTRAL_DISCONNECT:
50             conn_handle, _, _ = data
51             print("Disconnected", conn_handle)
52             self._connections.remove(conn_handle)
53             # Start advertising again to allow a new connection.
54             self._advertise()
55         elif event == _IRQ_GATTS_WRITE:
56             conn_handle, value_handle = data
57             value = self._ble.gatts_read(value_handle)
58             if value_handle == self._handle_rx and self._write_callback:

```

Any concerns? ✉ support@freenove.com

```
59             self._write_callback(value)
```

Initialize the BLE function and name it.

```
32     def __init__(self, ble, name="PicoW"):
```

When the smartphone sends data to the Pico W via BLE Bluetooth, convert the data to utf-8 format and remove the trailing '\r\n', then print it out via the serial port. When the Pico W receives data from the serial port, it sends the data back to the smartphone via BLE Bluetooth.

```
76     def demo():
77         ble = bluetooth.BLE()
78         p = BLESimplePeripheral(ble)
79
80         def on_rx(rx_data):
81             rx_data = rx_data.decode('utf-8').rstrip('\r\n')
82             print("\nReceive:", rx_data)
83             p.on_write(on_rx)
84             print("Please use LightBlue to connect to PicoW.")
85
86         while True:
87             if p.is_connected():
88                 # Short burst of queued notifications.
89                 tx_data = input("Enter anything: ")
90                 print("Send: ", tx_data)
91                 p.send(tx_data)
```

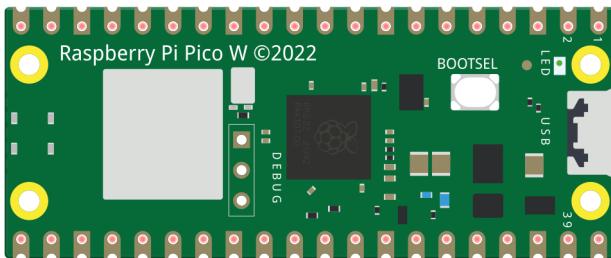


Project 15.2 Bluetooth Low Energy Control LED

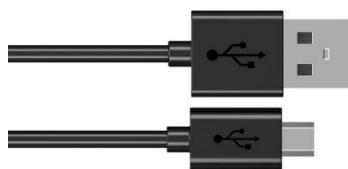
In this project, we will control an LED via Pico W's Bluetooth function.

Component List

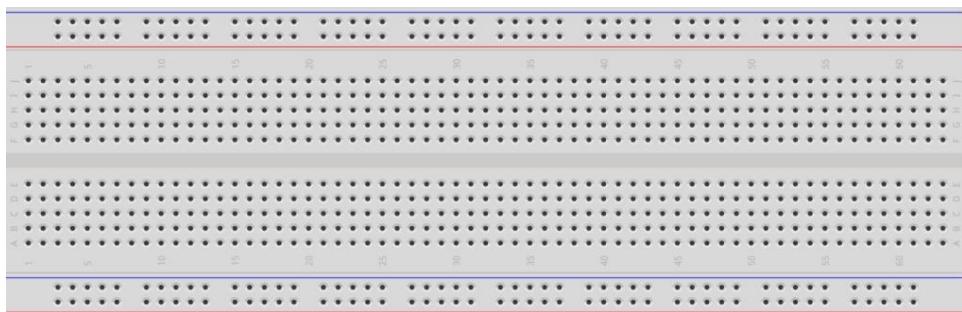
Raspberry Pi Pico W(or Pico 2W) x1



Micro USB Wire x1



Breadboard x1



LED x1



Resistor 220Ω x1



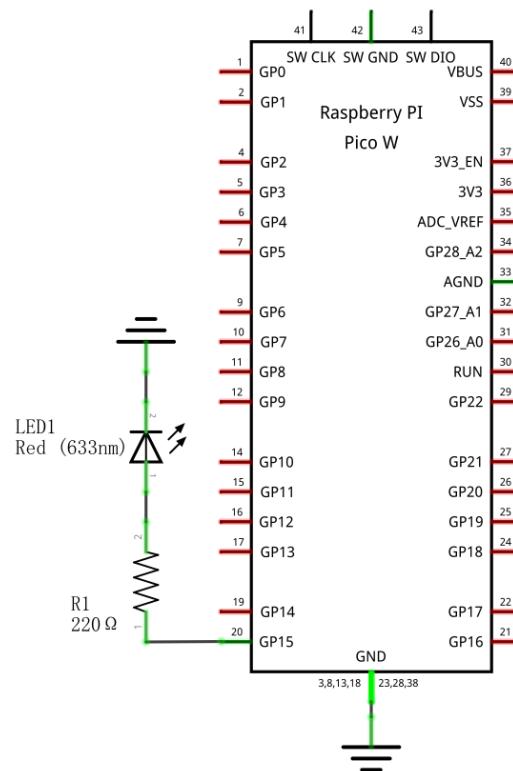
Jumper



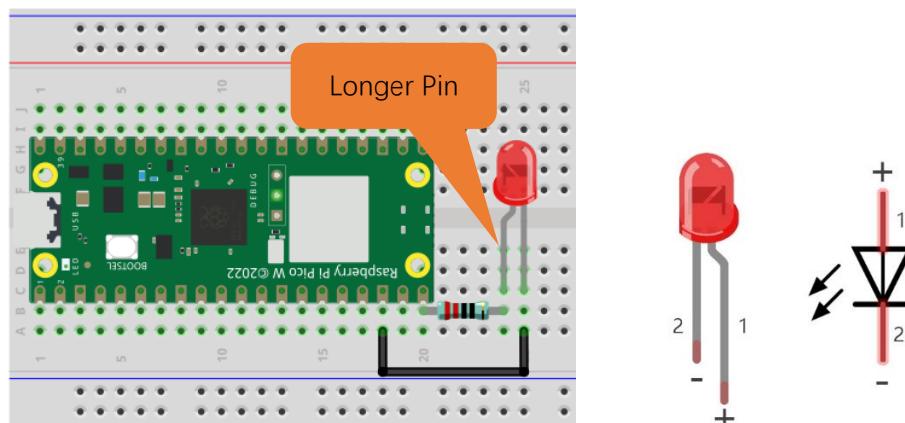
Circuit

Connect Pico W to your computer with the USB cable.

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.

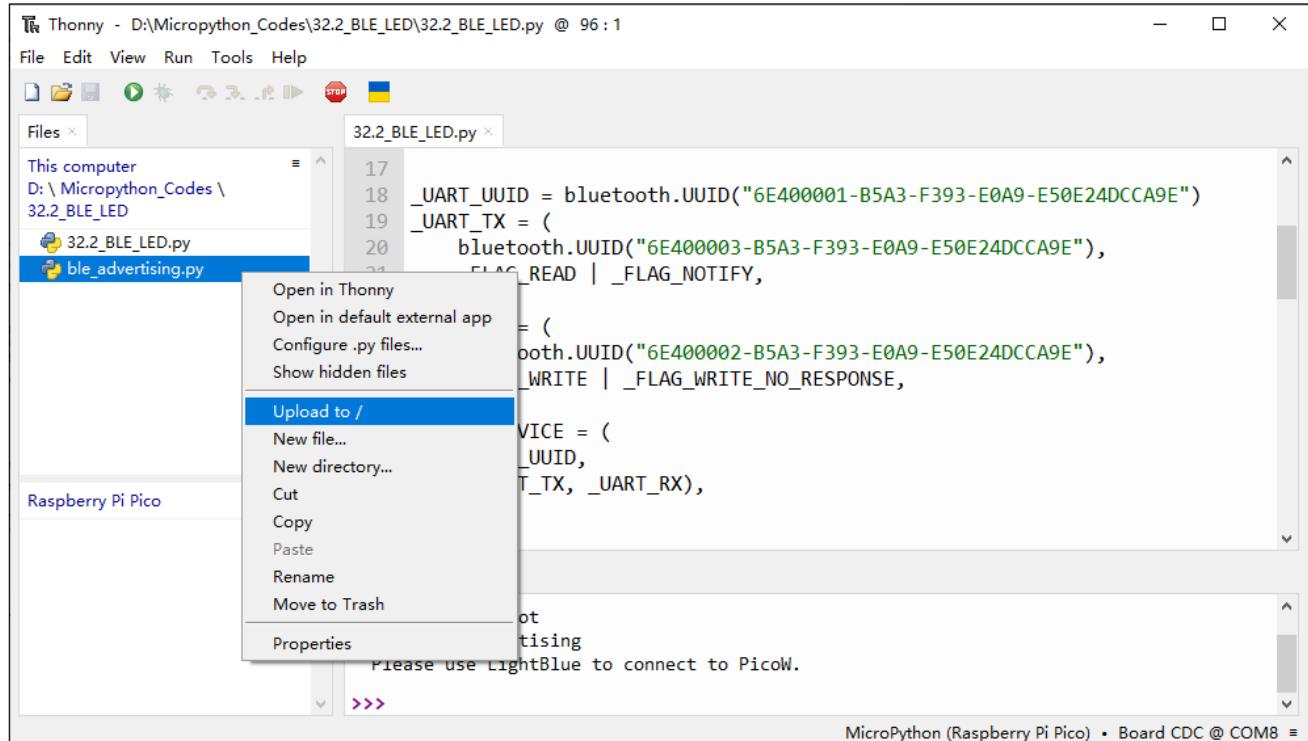


Code

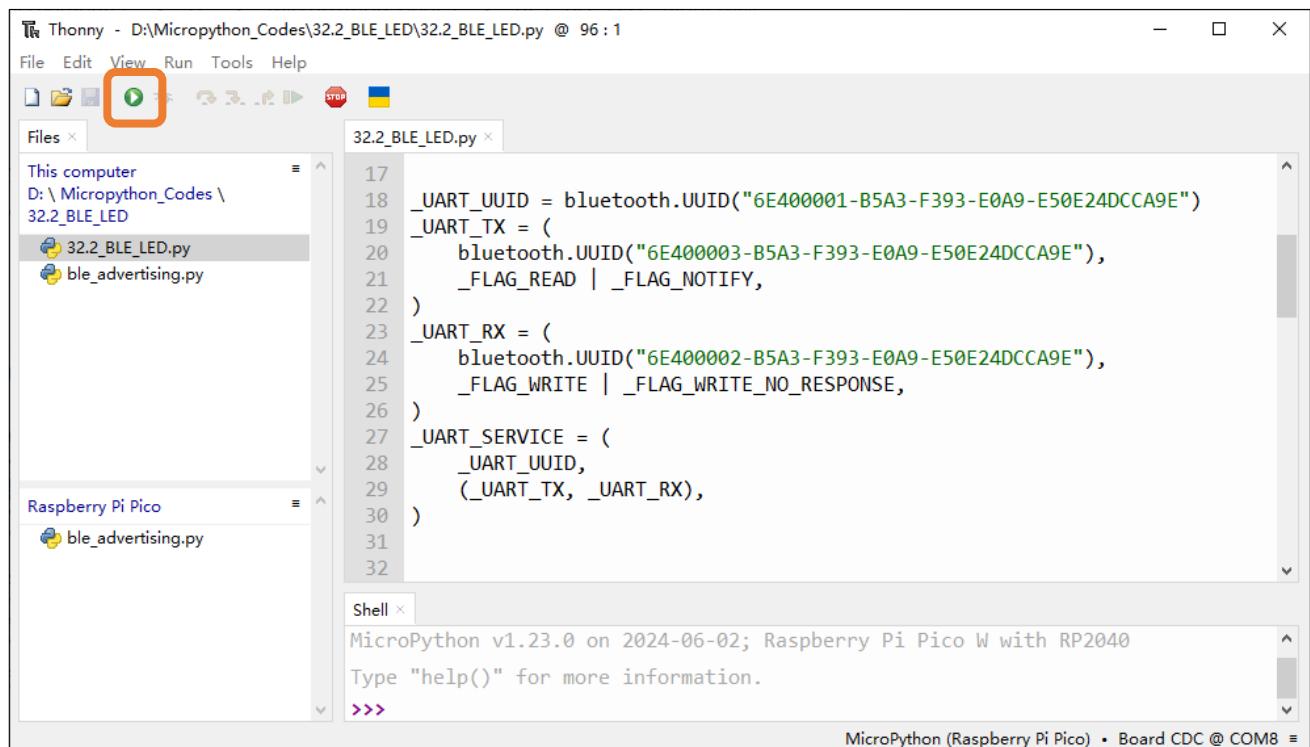
Move the program folder “**Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico/Python/Python_Codes**” to disk(D) in advance with the path of “**D:/Micropython_Codes**”.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “15.2_BLE_LED”. **Right click** “ble_advertising.py” and select “Upload to /” to upload it to Pico W. Then, double click “ble_led.py” to open the file.

15.1_BLE_LED

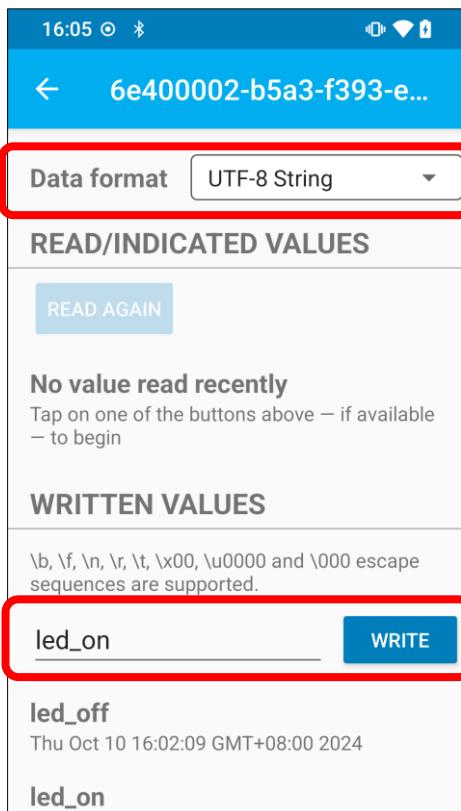


Click the Run button to run BLE_LED.py.





The operation on the phone app is similar to that in project 33.1. You just need to change the sending messages to “led_on”, “led_off” and “led_toggle” to control the status of the LED.



You can check the message sent by Bluetooth in “Shell”.

```
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Starting advertising
Please use LightBlue to connect to PicoW.

>>> New connection 64
Receive: led_on
Receive: led_off
Receive: led_on
Receive: led_off
```

LED Status.



Note: If the messages you send are not "led_on", "led_off", or "led toggle", the status of the LED will not change.

For example, when the LED is already ON, it will remain ON unless the message "led_off", or "led toggle" is received.

The following is the program code:

```

1 import bluetooth
2 import random
3 import struct
4 import time
5 from ble_advertising import advertising_payload
6 from machine import Pin
7 from micropython import const
8
9 _IRQ_CENTRAL_CONNECT = const(1)
10 _IRQ_CENTRAL_DISCONNECT = const(2)
11 _IRQ_GATTS_WRITE = const(3)
12
13 _FLAG_READ = const(0x0002)
14 _FLAG_WRITE_NO_RESPONSE = const(0x0004)
15 _FLAG_WRITE = const(0x0008)
16 _FLAG_NOTIFY = const(0x0010)
17
18 _UART_UUID = bluetooth.UUID("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")
19 _UART_TX = (
20     bluetooth.UUID("6E400003-B5A3-F393-E0A9-E50E24DCCA9E"),
21     _FLAG_READ | _FLAG_NOTIFY,
22 )
23 _UART_RX = (
24     bluetooth.UUID("6E400002-B5A3-F393-E0A9-E50E24DCCA9E"),
25     _FLAG_WRITE | _FLAG_WRITE_NO_RESPONSE,
26 )
27 _UART_SERVICE = (
28     _UART_UUID,
29     (_UART_TX, _UART_RX),

```



```

30 )
31
32 class BLESimplePeripheral:
33     def __init__(self, ble, name="PicoW"):
34         self._ble = ble
35         self._ble.active(True)
36         self._ble.irq(self._irq)
37         ((self._handle_tx, self._handle_rx),) =
38         self._ble.gatts_register_services((_UART_SERVICE,))
39         self._connections = set()
40         self._write_callback = None
41         self._payload = advertising_payload(name=name, services=[_UART_UUID])
42         self._advertise()
43
44     def _irq(self, event, data):
45         # Track connections so we can send notifications.
46         if event == _IRQ_CENTRAL_CONNECT:
47             conn_handle, _, _ = data
48             print("New connection", conn_handle)
49             self._connections.add(conn_handle)
50         elif event == _IRQ_CENTRAL_DISCONNECT:
51             conn_handle, _, _ = data
52             print("Disconnected", conn_handle)
53             self._connections.remove(conn_handle)
54         # Start advertising again to allow a new connection.
55         self._advertise()
56     elif event == _IRQ_GATTS_WRITE:
57         conn_handle, value_handle = data
58         value = self._ble.gatts_read(value_handle)
59         if value_handle == self._handle_rx and self._write_callback:
60             self._write_callback(value)
61
62     def send(self, data):
63         for conn_handle in self._connections:
64             self._ble.gatts_notify(conn_handle, self._handle_tx, data)
65
66     def is_connected(self):
67         return len(self._connections) > 0
68
69     def _advertise(self, interval_us=500000):
70         print("Starting advertising")
71         self._ble.gap_advertise(interval_us, adv_data=self._payload)
72
73     def on_write(self, callback):

```

```
74     self._write_callback = callback
75
76
77 def demo():
78     ble = bluetooth.BLE()
79     p = BLESimplePeripheral(ble)
80     led = Pin(15, Pin.OUT)
81
82     def on_rx(rx_data):
83         rx_data = rx_data.decode('utf-8').rstrip('\r\n')
84         print("Receive:", rx_data)
85         if rx_data == 'led_on':
86             led.value(1)
87         elif rx_data == 'led_off':
88             led.value(0)
89         else:
90             pass
91
92     p.on_write(on_rx)
93     print("Please use LightBlue to connect to PicoW.")
94 if __name__ == "__main__":
95     demo()
```

Compare received message with "led_on" and "led_off" and take action accordingly.

```
85     if rx_data == 'led_on':
86         led.value(1)
87     elif rx_data == 'led_off':
88         led.value(0)
89     else:
90         pass
```

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