

# 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](mailto: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 resource 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](mailto:support@freenove.com)

# Contents

Welcome.....	1
Contents .....	1
Preface.....	2
Raspberry Pi Pico.....	3
Chapter 0 Getting Ready (Important).....	7
0.1 Installing Thonny (Important) .....	7
0.2 Basic Configuration of Thonny .....	12
0.3 Burning Micropython Firmware (Important).....	14
0.4 Thonny Connected to Raspberry Pi Pico .....	17
0.5 Testing codes (Important).....	21
0.6 Thonny Common Operation.....	32
Chapter 1 LCD1602 .....	37
Project 1.1 LCD1602.....	37
Chapter 2 LCD2004 .....	49
Project 1.1 LCD2004.....	49
What's Next? .....	56

## 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.

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

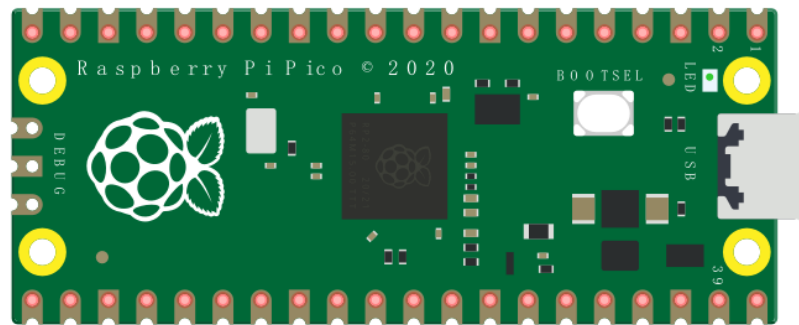
- 1, Component list: helps users to learn and find what components needed in each project.
- 2, Component knowledge: allows you to learn the features and usage of the components.
- 3, Circuit: assists to build circuit for each project.
- 4, Code and annotation: makes it easier for users to learn to use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

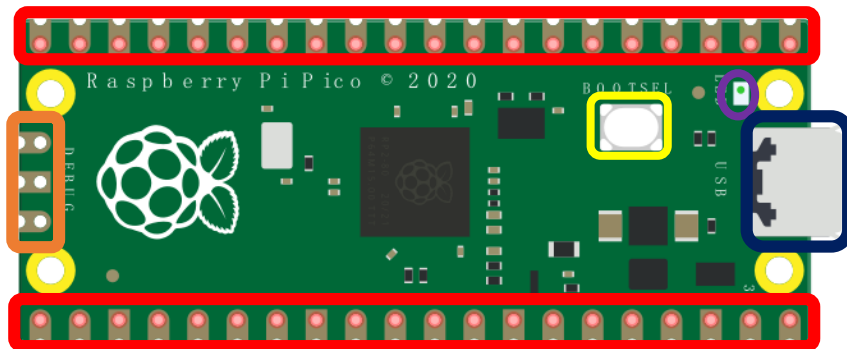
If you have any problems or difficulties using this product, please contact us for quick and free technical support: [support@freenove.com](mailto:support@freenove.com)






## 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.

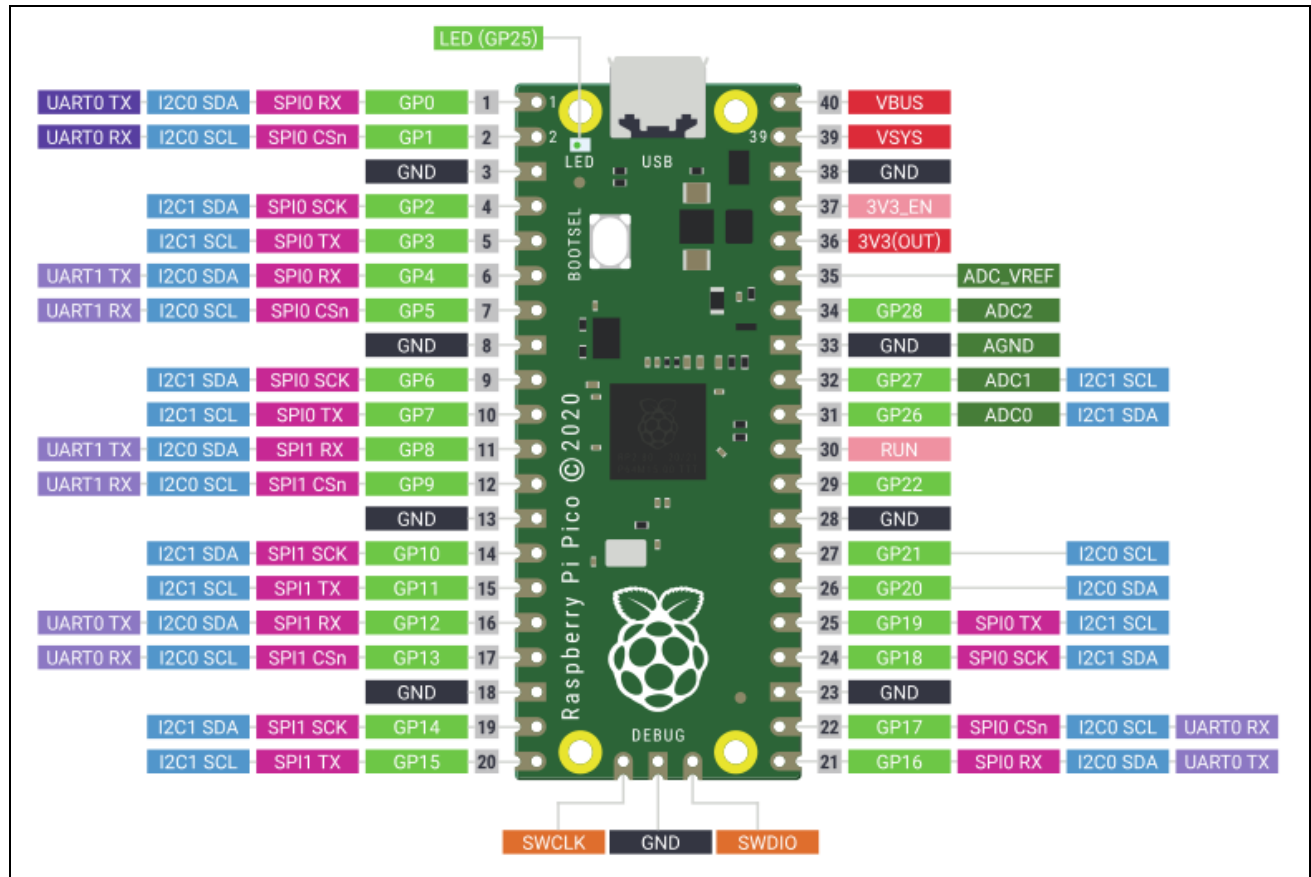










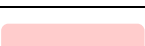

The hardware interfaces are distributed as follows:



Frame color	Description
	Pins
	BOOTSE 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.org/pico/pico-datasheet.pdf>

GND	Ground Pin
Power	VBUS(microUSB Voltage)、VSY(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 VSY on Pico board; ADC4 is directly connected to the RP2040's built-in temperature sensor. ADC_VREF can connect to external accurate voltmeter as ADC reference. ADC_GND pin is used as the reference point for grounding.
PWM	There are 16 PWM channels on Raspberry Pi Pico, each of which can control frequency and duty cycle independently. The GPIO pins are switched to PWM function.
UART	There are 2 UART: UART0, UART1.
SPI	There are 2 SPI: SPI0, SPI1.
I2C	2 I2C: I2C0, I2C1.
Debugging	It is used when debugging code.

## UART, I2C, SPI Default Pin

### UART

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

### I2C

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

## SPI

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

For more detailed information, please refer to:

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



# Chapter 0 Getting Ready (Important)

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

## 0.1 Installing Thonny (Important)

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

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

## Downloading Thonny

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

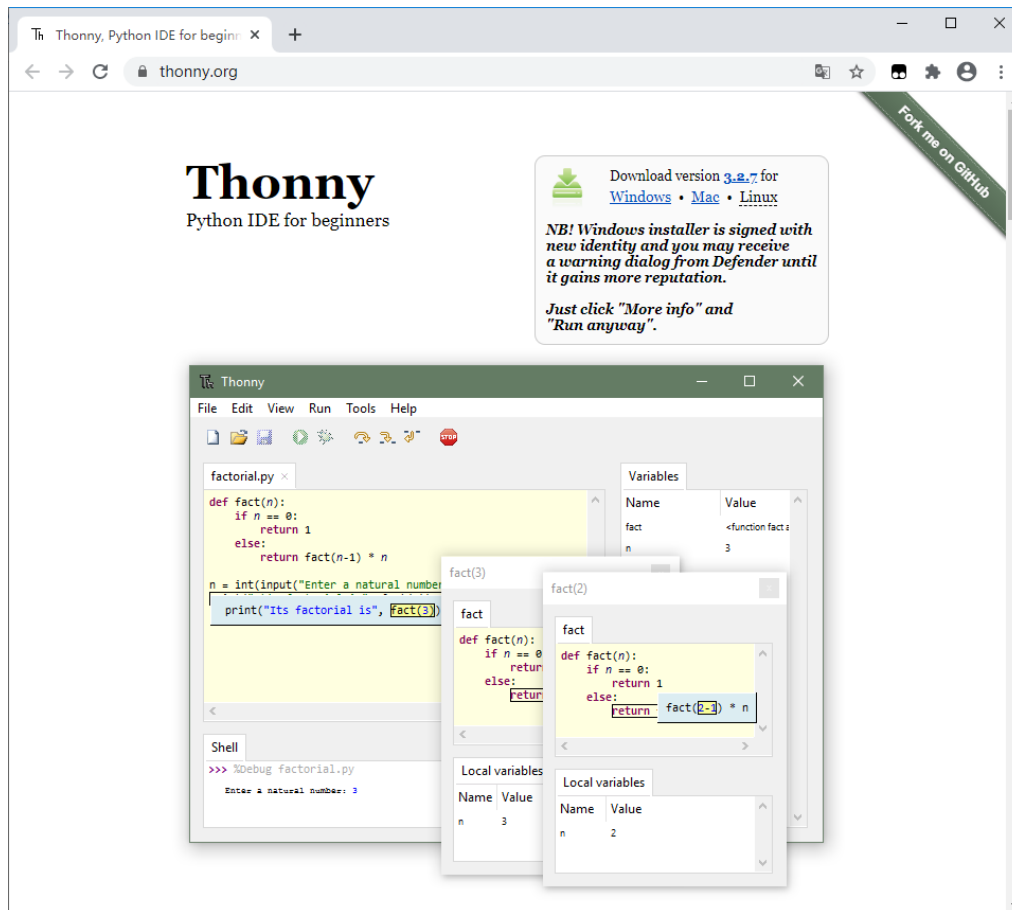
Open-source code repositories of Thonny: <https://github.com/thonny/thonny>

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

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

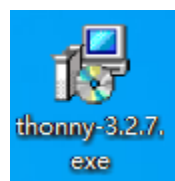
You can also open

"[Freenove\\_LCD\\_Module/Freenove\\_LCD\\_Module\\_for\\_Raspberry\\_Pi\\_Pico/Python\\_Software](#)", we have prepared it in advance.



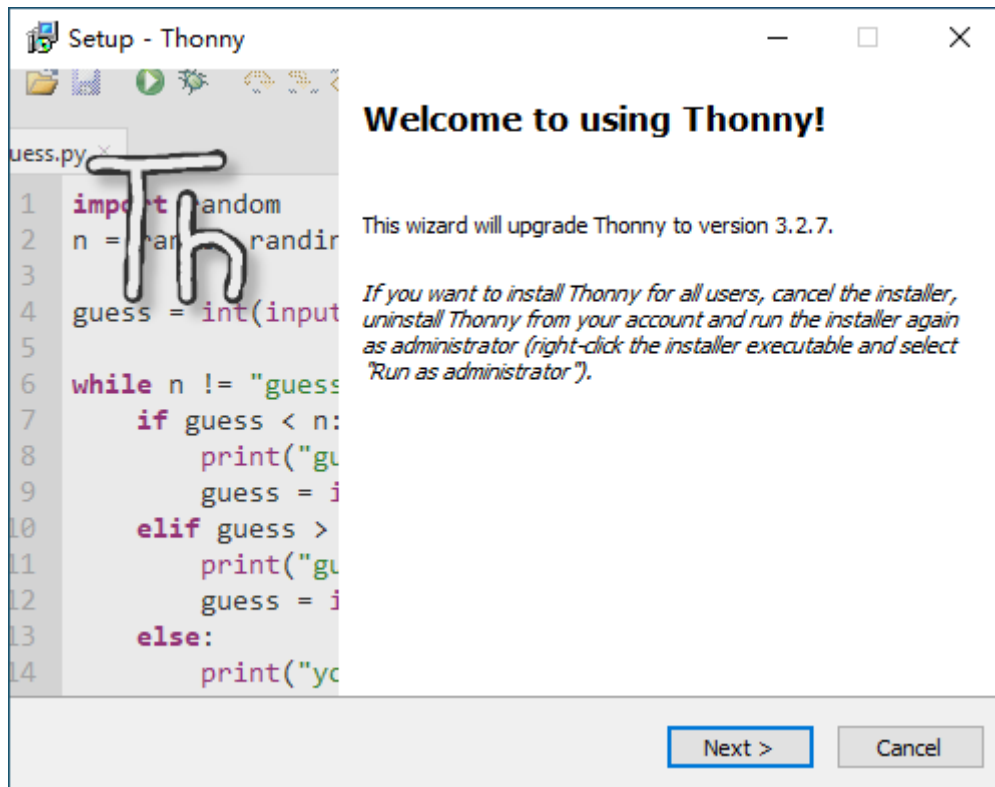
## Installing on Windows

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



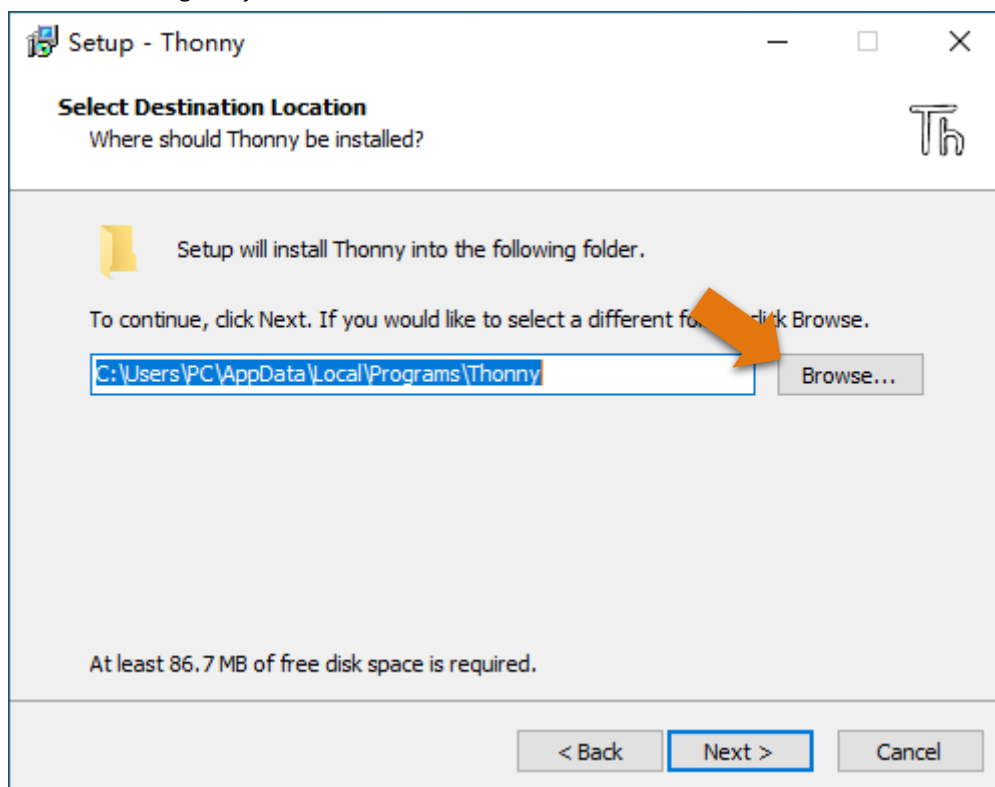
Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

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

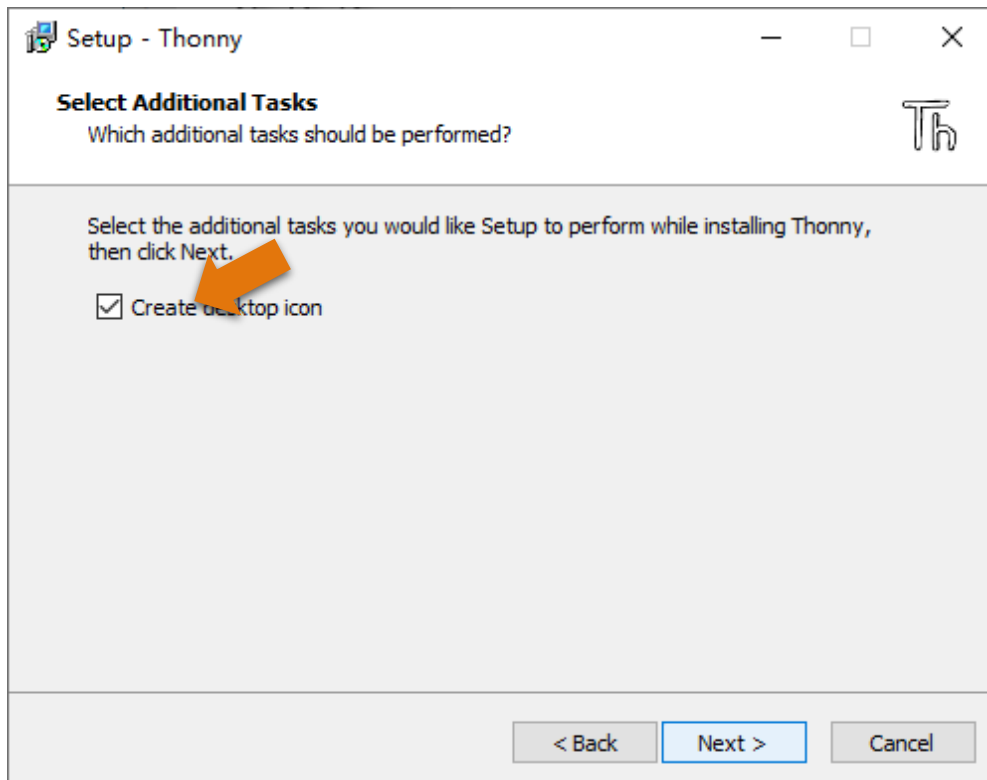


If you want to change Thonny's installation path, you can click "Browse" to modify it. After selecting installation path, click "OK".

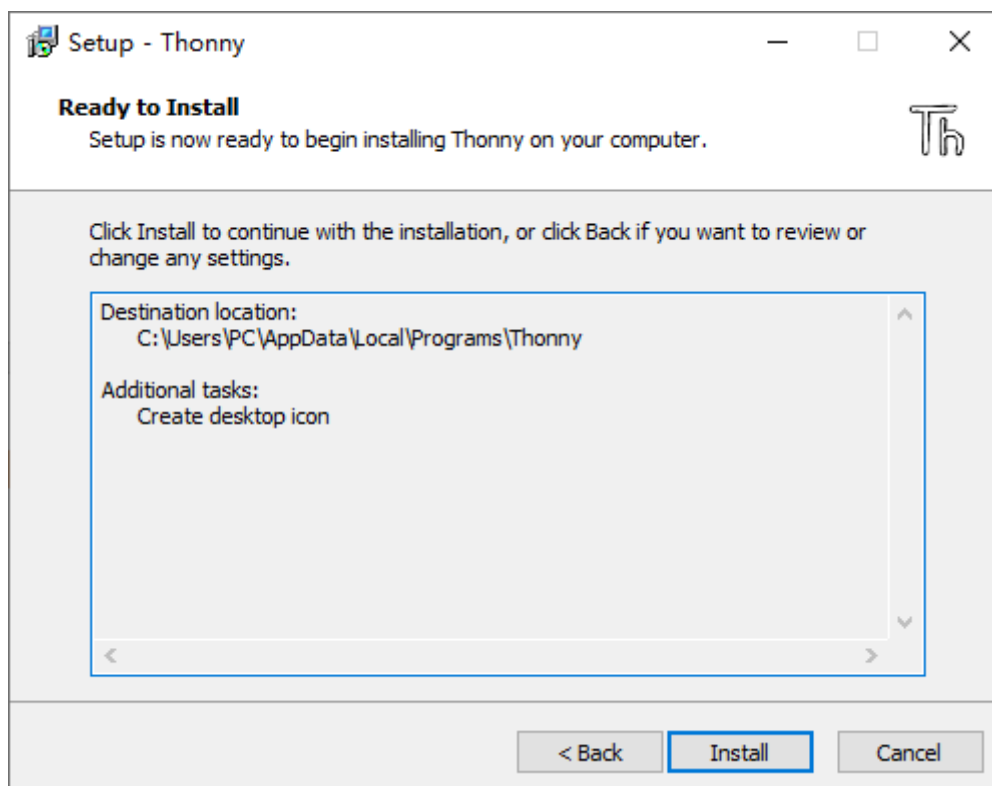
If you do not want to change it, just click "Next".



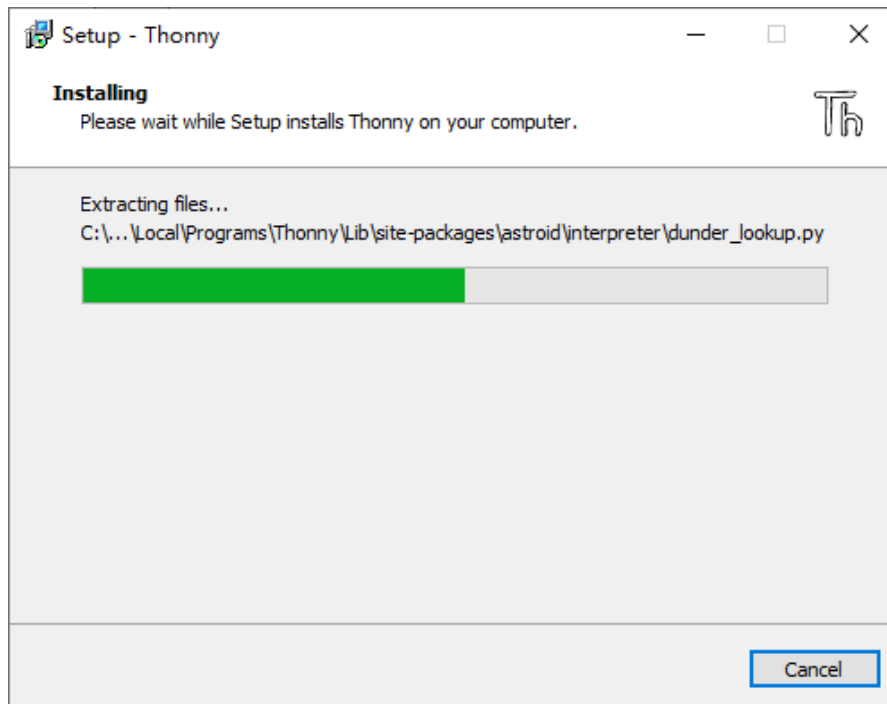
Check “Create desktop icon” and then it will generate a shortcut on your desktop to facilitate you to open Thonny later.



Click “install” to install the software.



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



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

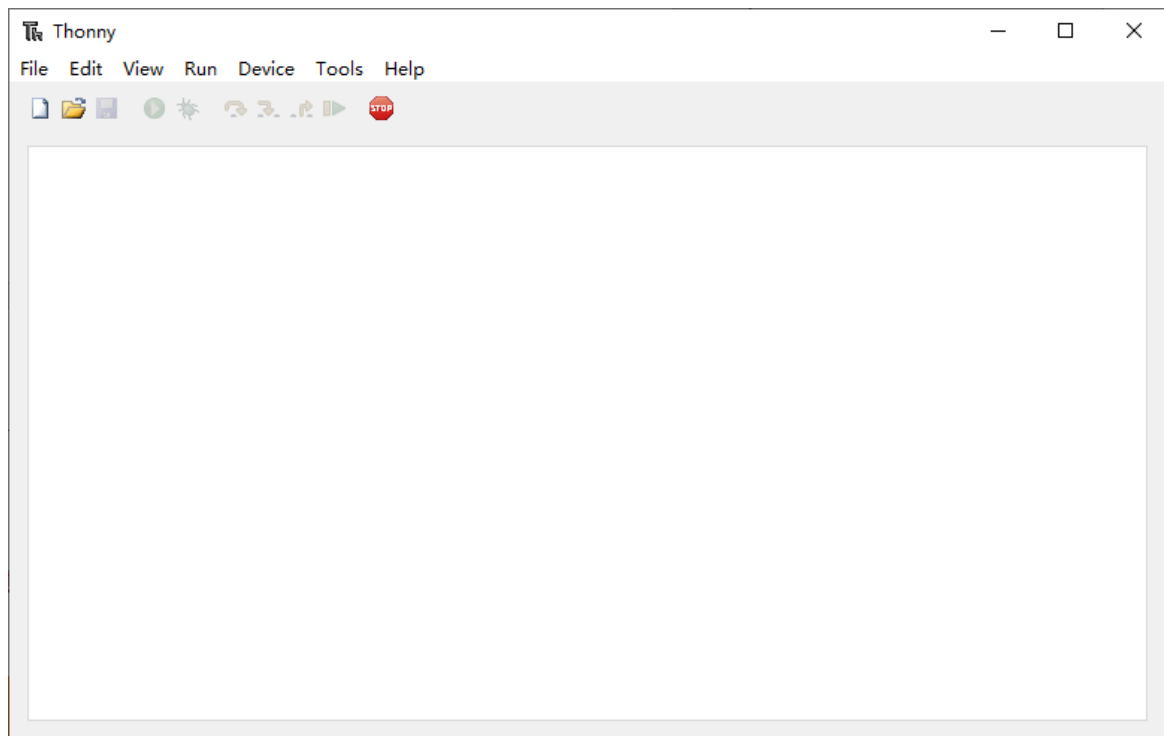


If you've checked "Create desktop icon" during the installation process, you can see the below icon on your desktop.

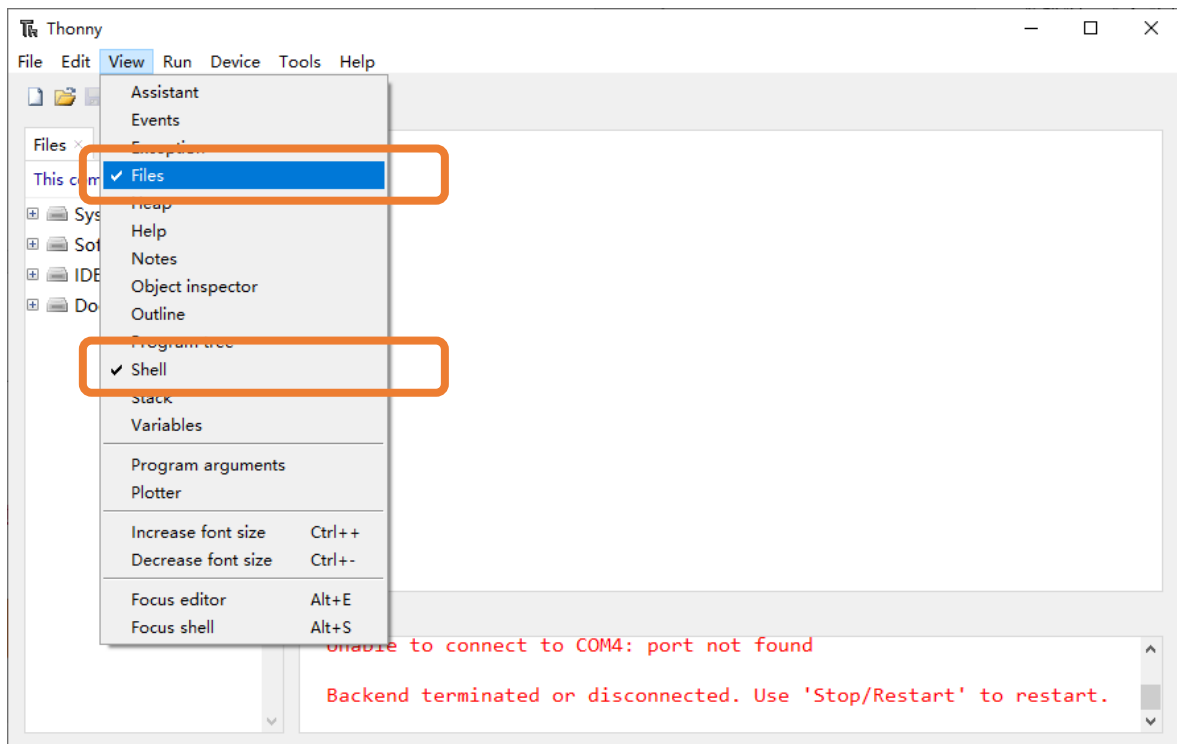


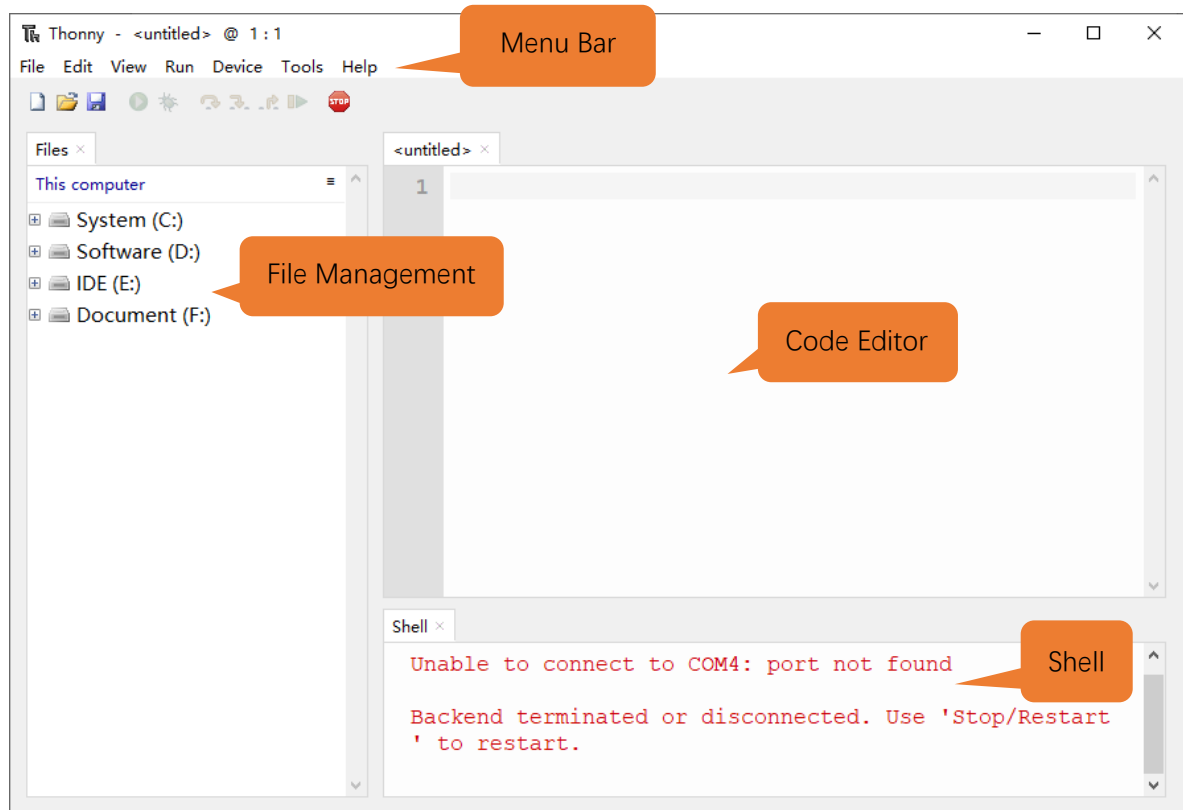
## 0.2 Basic Configuration of Thonny

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



Select “View”→ “Files” and “Shell”.





## 0.3 Burning Micropython Firmware (Important)

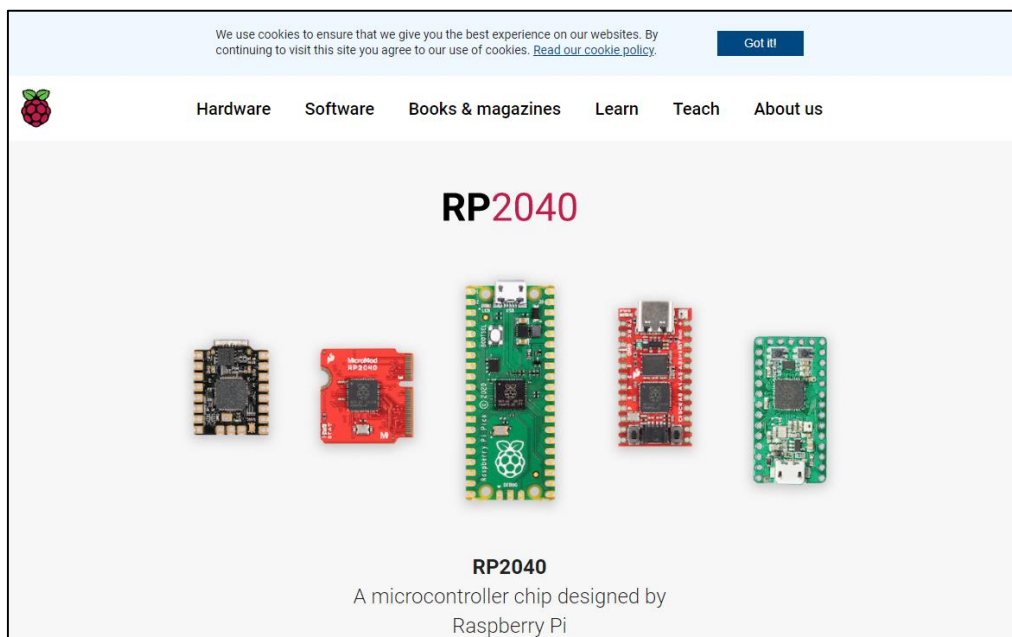
To run Python programs on Raspberry Pi Pico, we need to burn a firmware to Raspberry Pi Pico first.

### Downloading Micropython Firmware

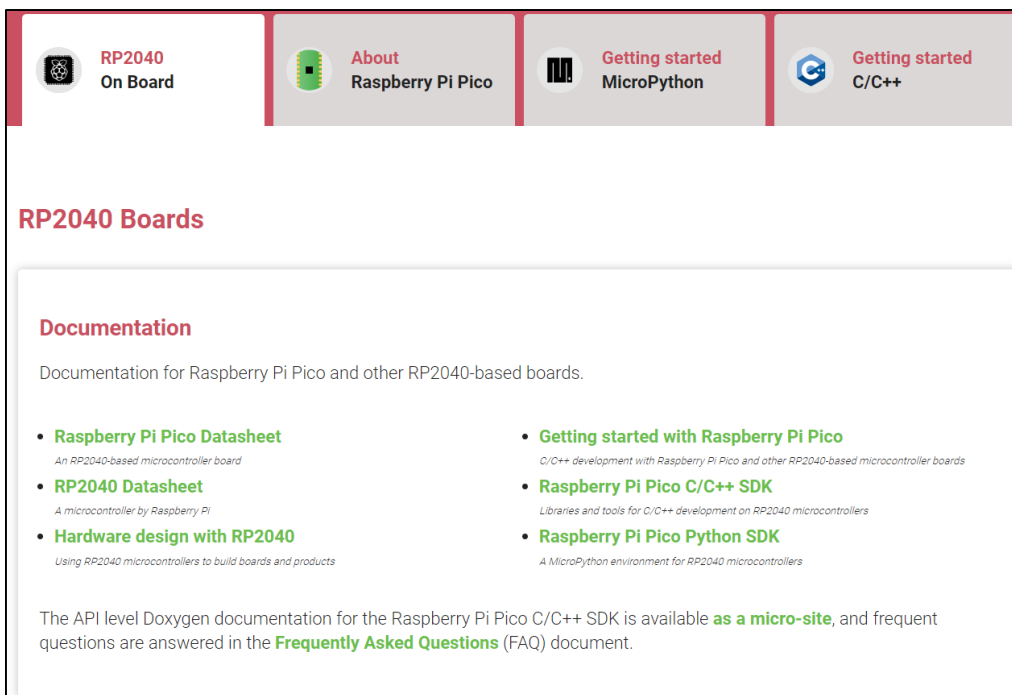
Option 1:

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

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



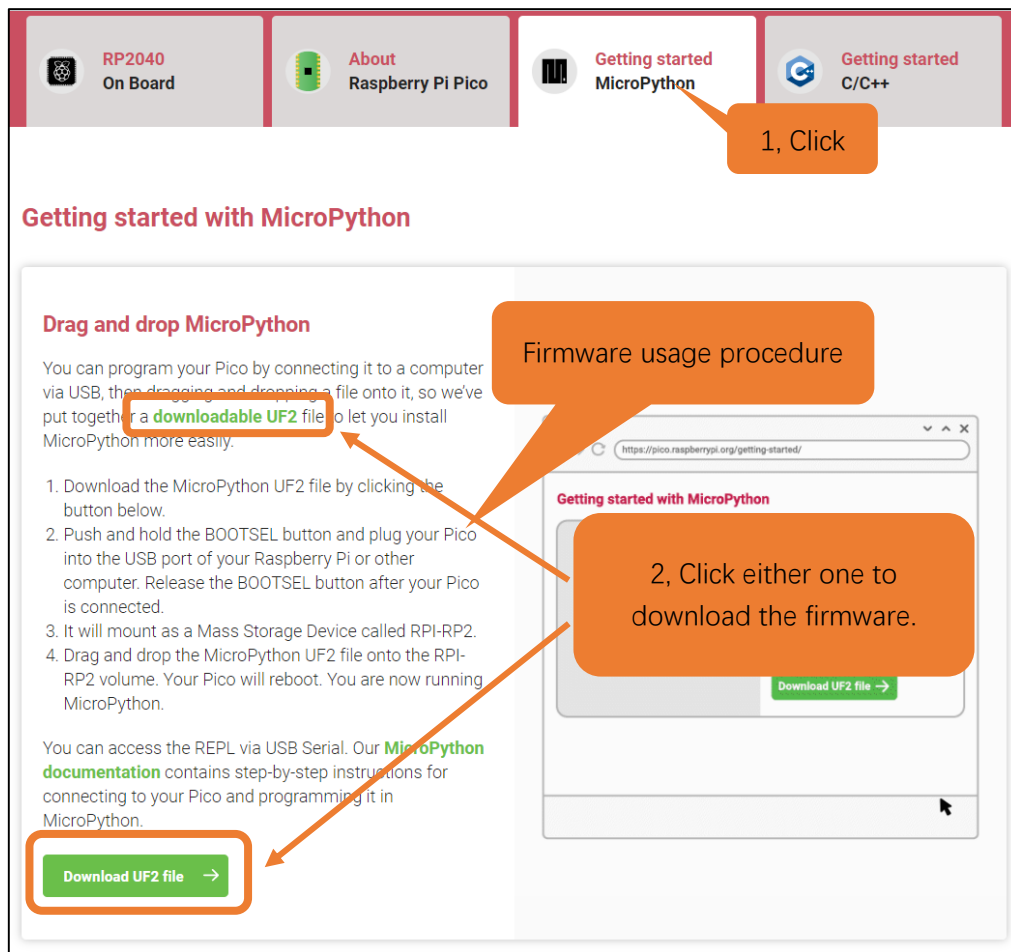
2, Roll the mouse and you can see the links for RP2040 documents.



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



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



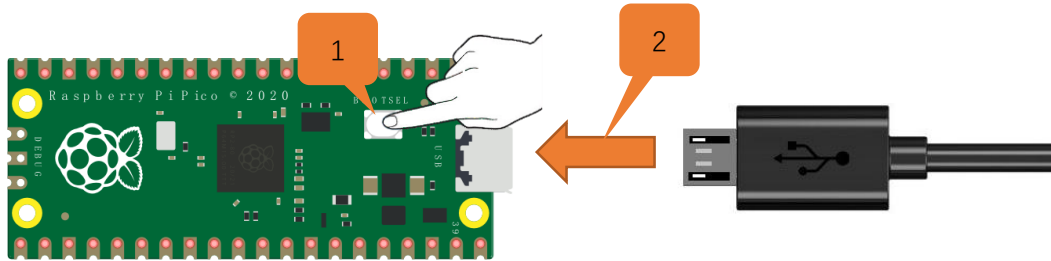
Option 2: Directly download via [rp2-pico-20220618-v1.19.1.uf2](https://github.com/raspberrypi/pico/blob/master/firmware/binary/uf2/rp2-pico-20220618-v1.19.1.uf2)

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

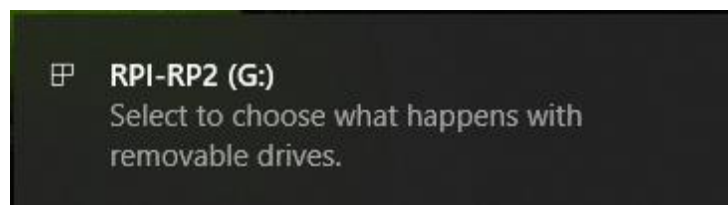
**Freenove\_LCD\_Module/Freenove\_LCD\_Module\_for\_Raspberry\_Pi\_Pico/Python\_Firmware**

## Burning a Micropython Firmware

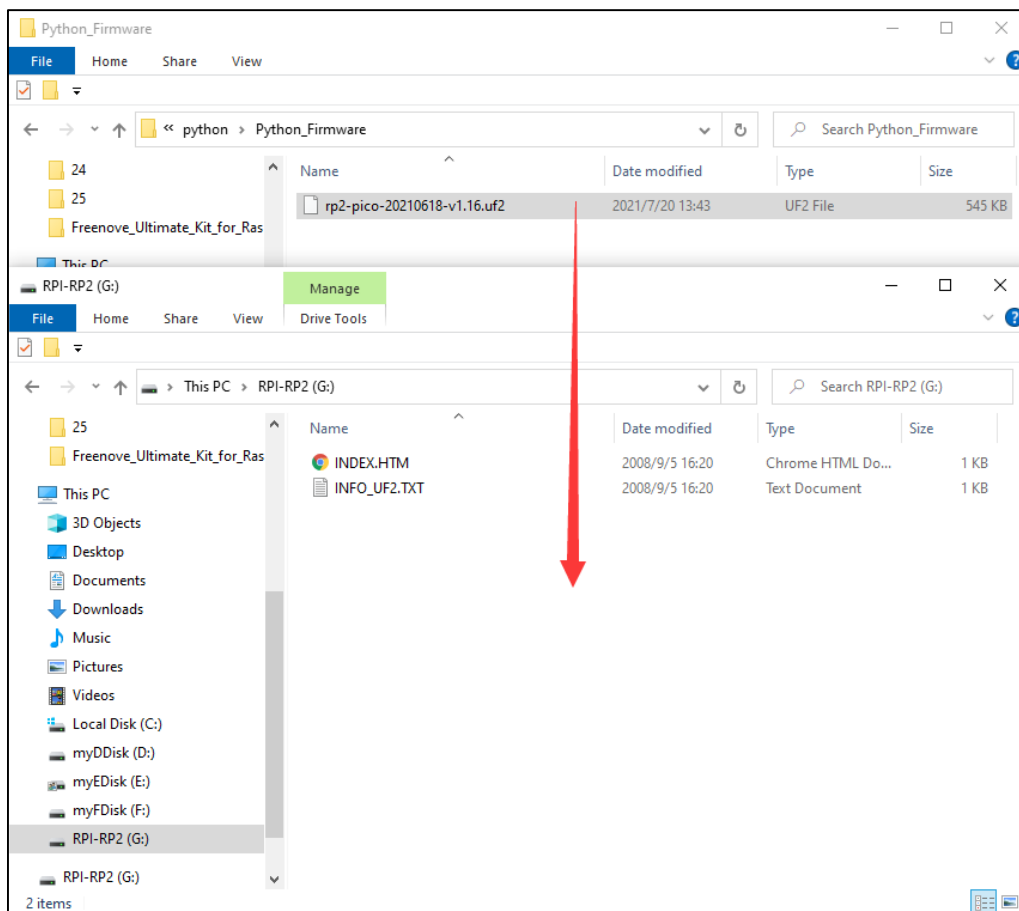
1. Connect a USB cable to your computer.
2. Long press BOOTSEL button on Raspberry Pi Pico and connect it to your computer with the USB cable.



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



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

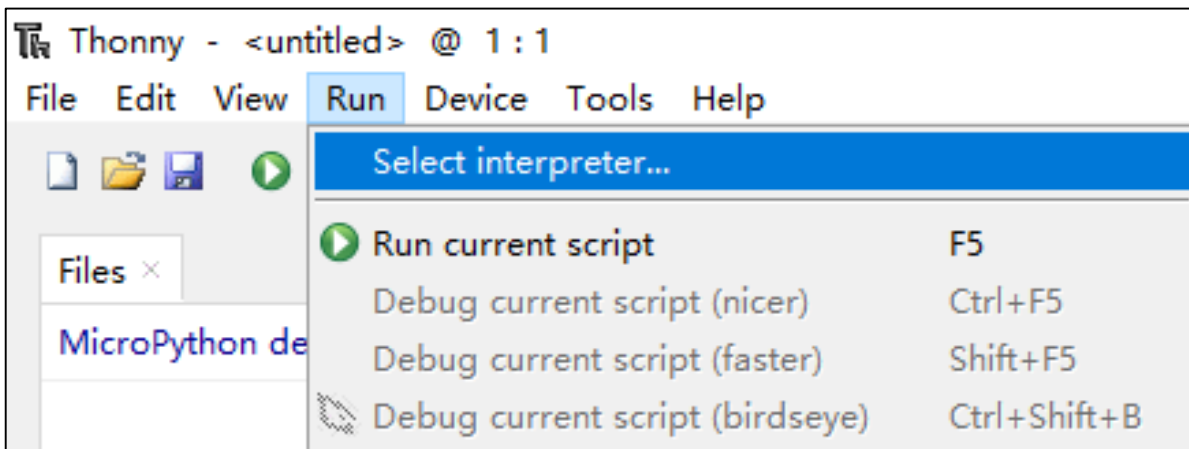


5. When the firmware finishes programming, Raspberry Pi Pico will reboot automatically. After that, you can run Micropython.

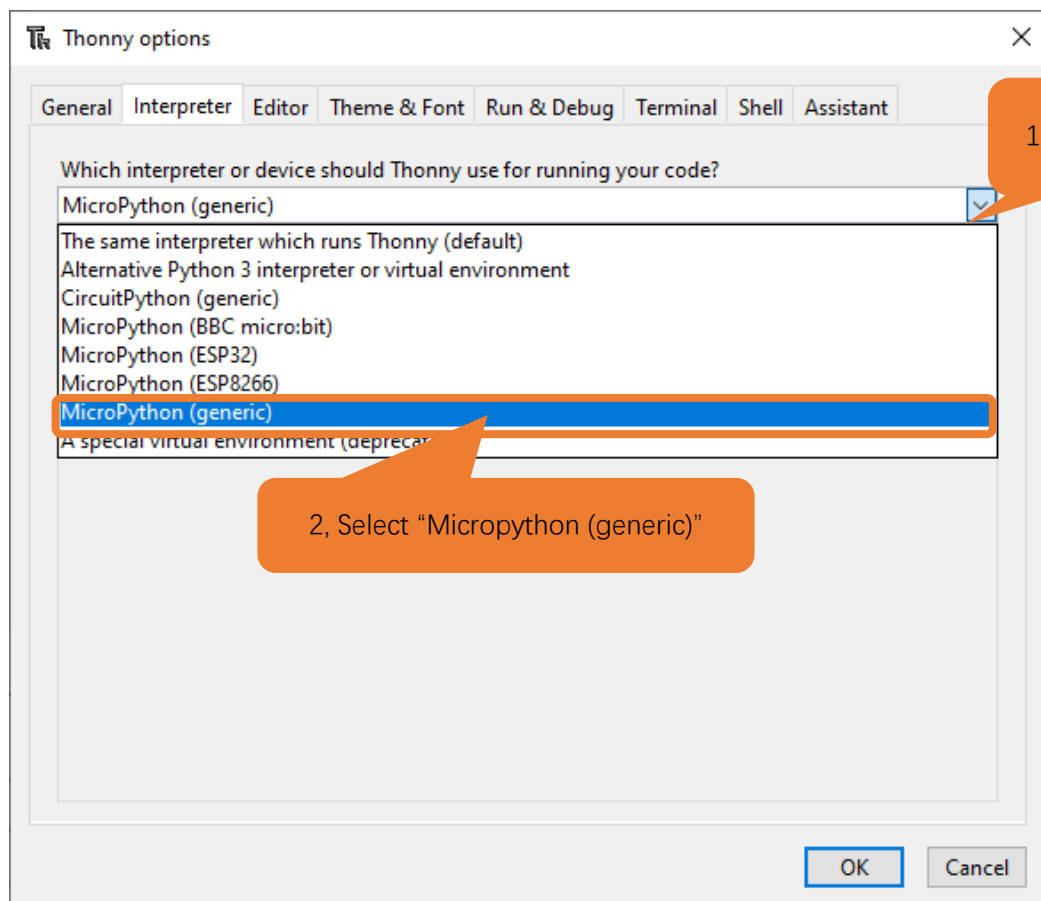
Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

## 0.4 Thonny Connected to Raspberry Pi Pico

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



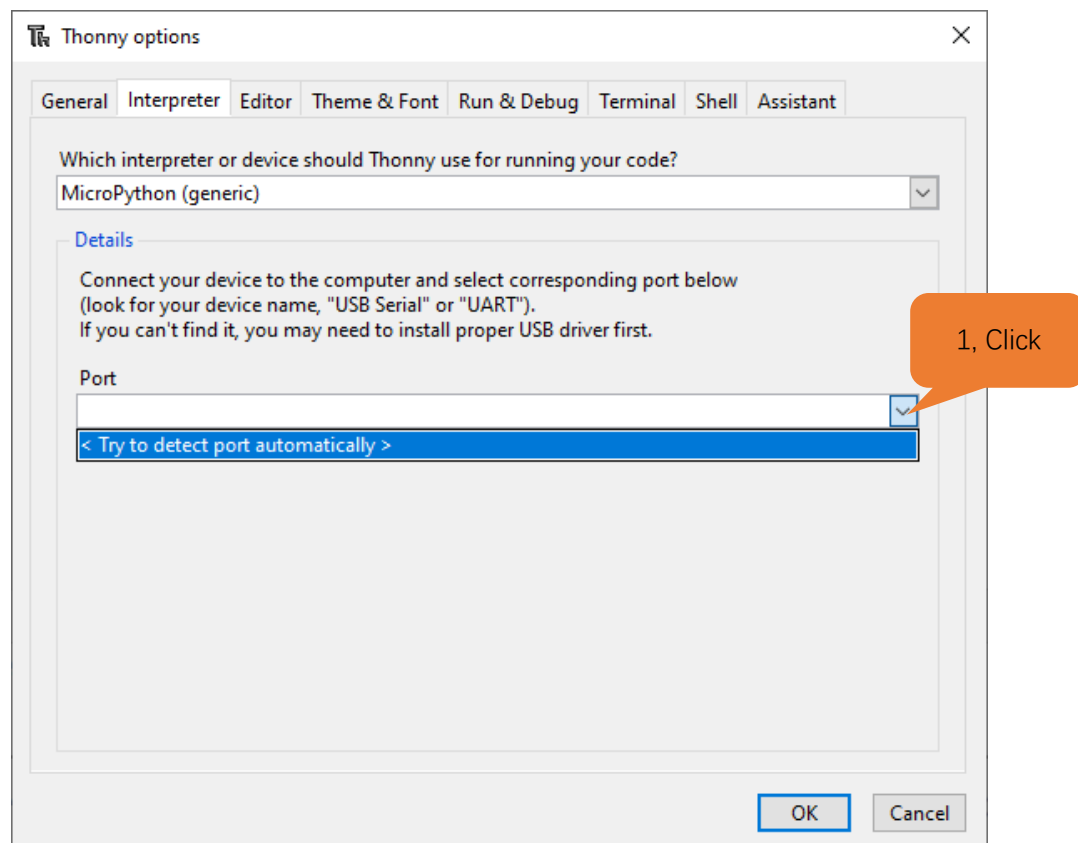
2. Select "Micropython (generic)" or "Micropython (Raspberry Pi Pico)".  
How to select "Micropython (generic)"? As shown below:



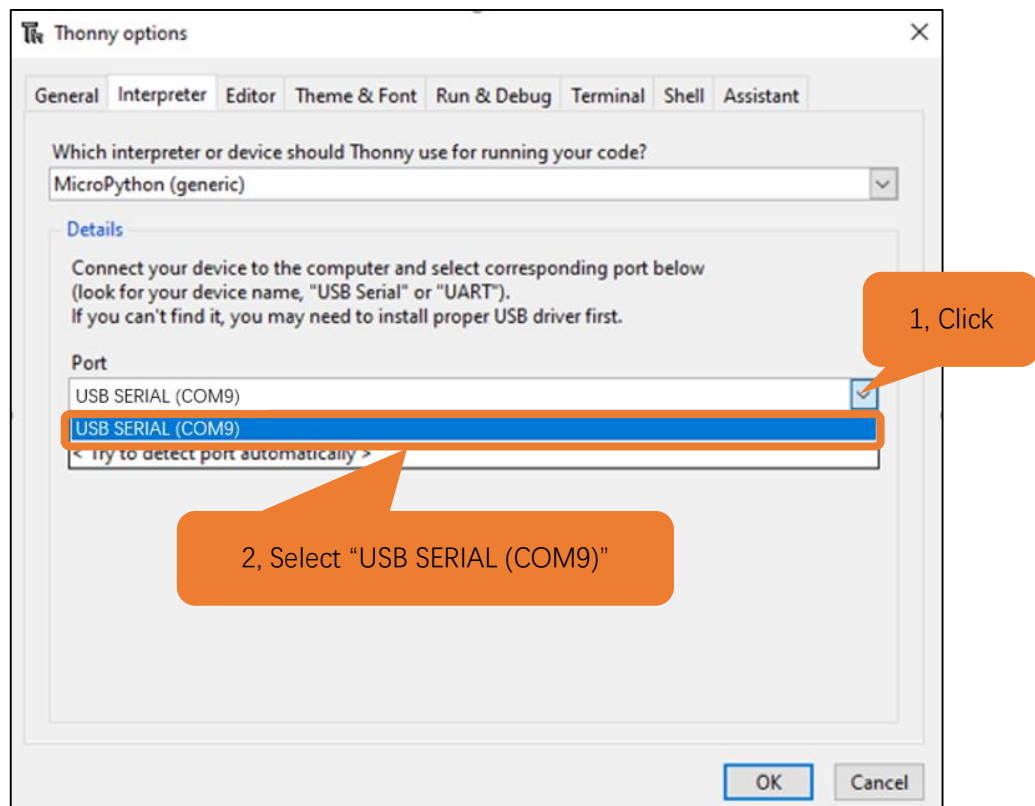
3. Select "USB-SERIAL (COMx)", The number x of COMx may varies among different computers. You only need to make sure selecting USB-SERIAL (COMx).

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

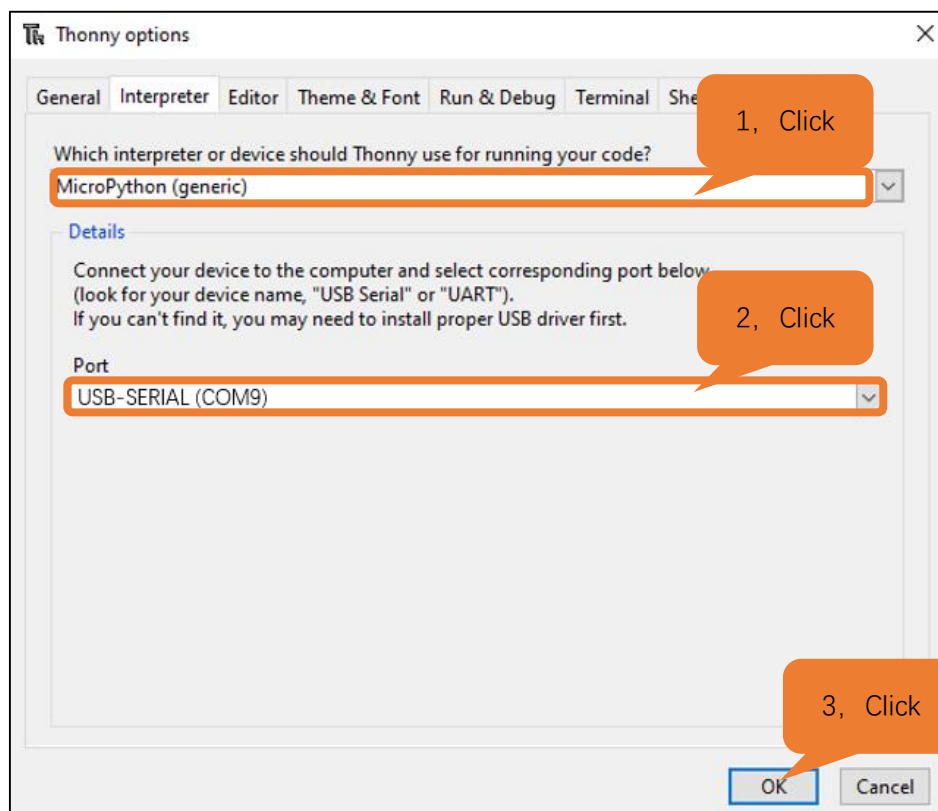
Step 1: When Pico **doesn't** connect to computer, open Thonny, click "Run", select "Select interpreter" and then a dialog box will pop up, click "Port" and you can check the ports currently connected to your computer, as shown below:



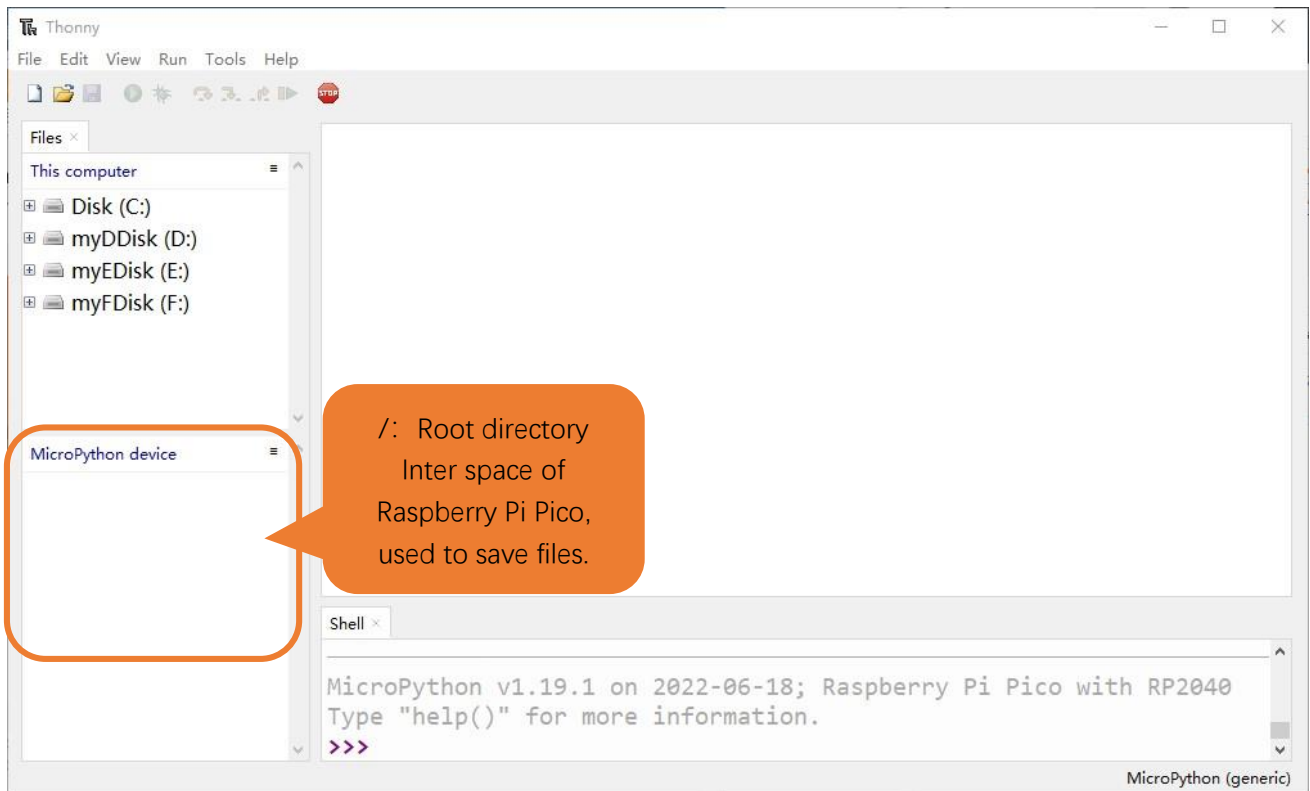
Step 2: Close the dialog box. Connect Pico to your computer, click "Run" again and select "Select interpreter". Click "Port" on the pop-up window and check the current ports. Now there is a newly added port, with which Pico communicates with the computer.



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



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



So far, all the preparations have been made.

## 0.5 Testing codes (Important)

### Testing Shell Command

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



## 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:

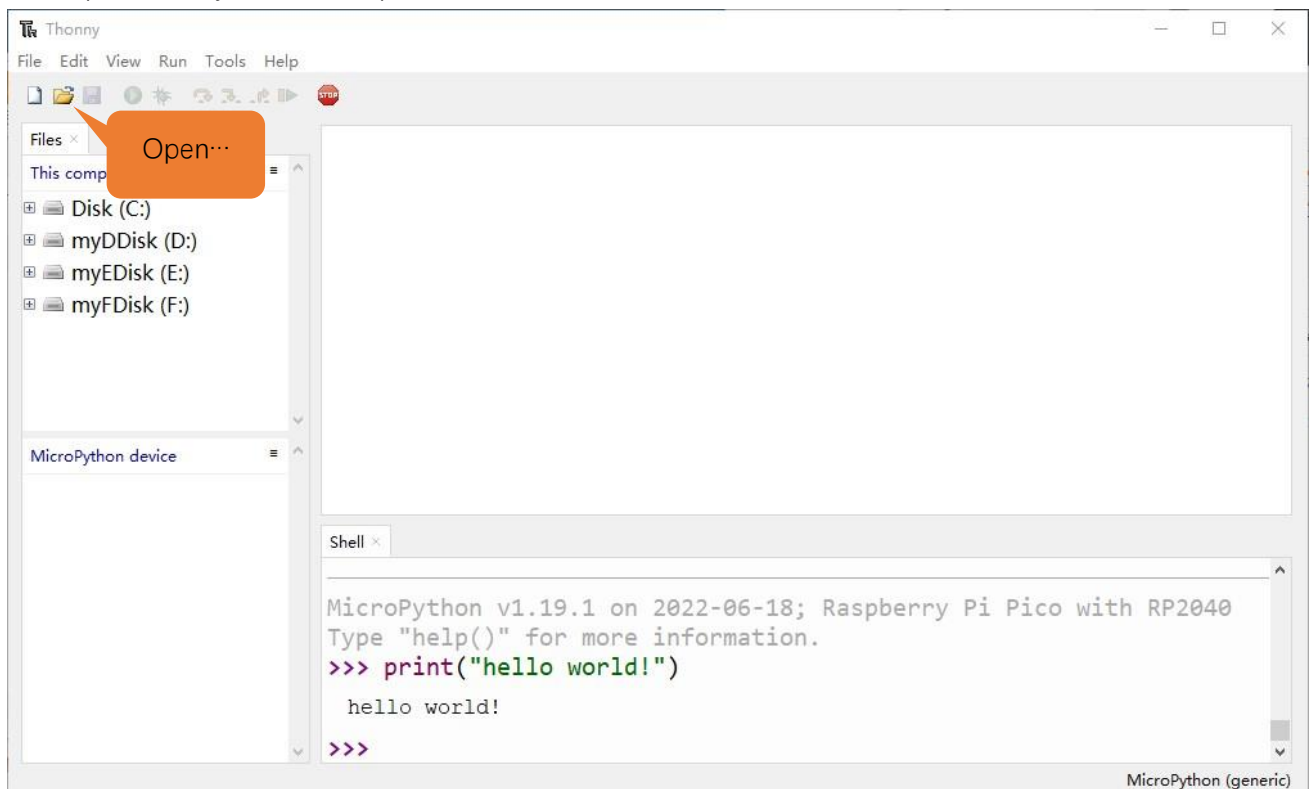
1. Users can use Thonny to compile or debug programs.
2. Through the "Shell" window, users can read the error information and output results generated during the running of the program and query related function information online to help improve the program.

Disadvantages:

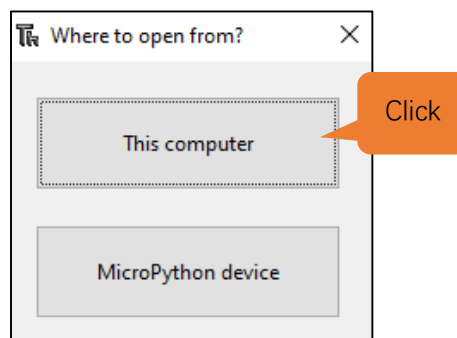
1. To run Raspberry Pi Pico online, you have to be connected to a computer and run with Thonny.
2. If Raspberry Pi Pico disconnects from computer, the program won't run again when they reconnect to each other.

### Opertation

1. Open Thonny and click "Open..."



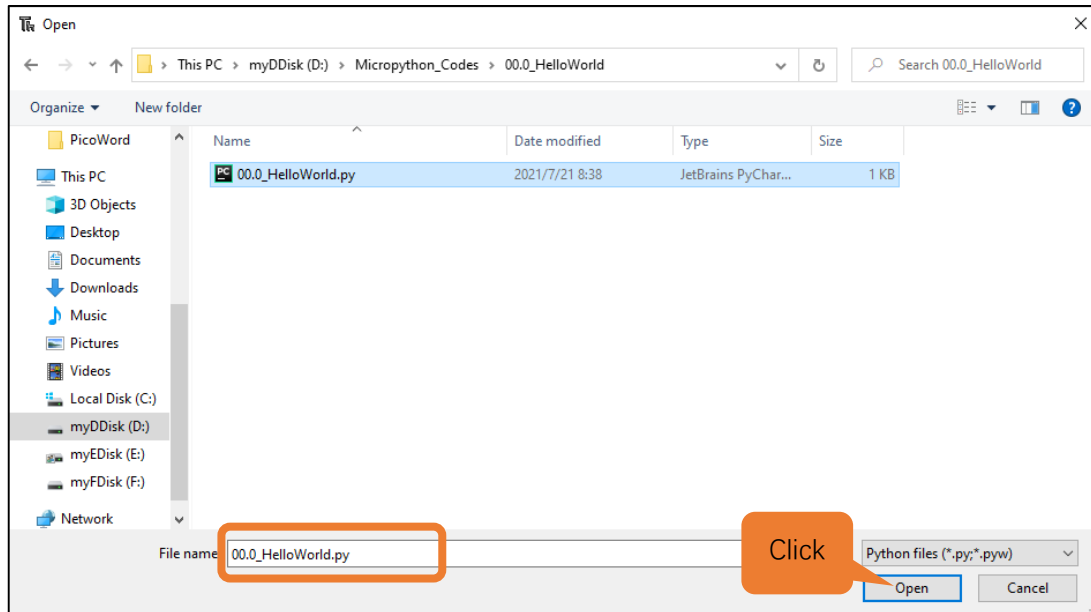
2. On the newly pop-up window, click "This computer".



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



In the new dialog box, select “00.0\_HelloWorld.py” in  
“Freenove\_LCD\_Module/Freenove\_LCD\_Module\_for\_Raspberry\_Pi\_Pico/Python/00.0\_HelloWorld”  
folder.

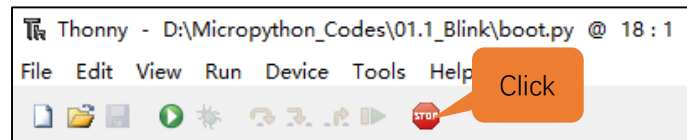


Click “Run current script” to execute the program and “Hello World!”, “Welcome Freenove” will be printed in “Shell”.



### Exiting Running Online

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



## Running Offline

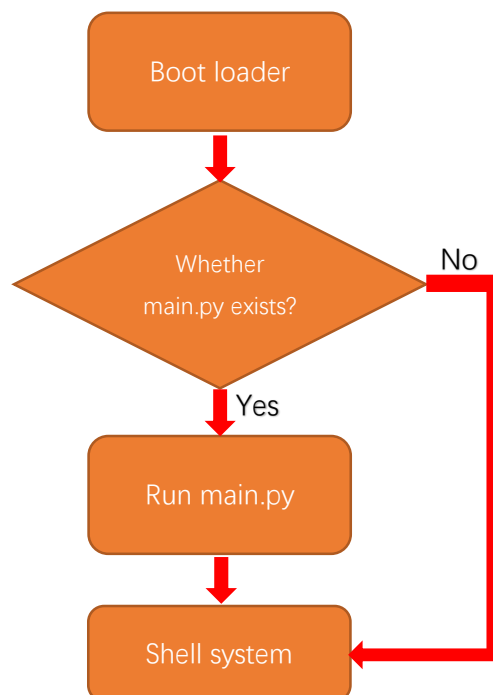
When running offline, Raspberry Pi Pico doesn't need to connect to computer and Thonny. It can run the programs stored in main.py on the device once powered up.

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

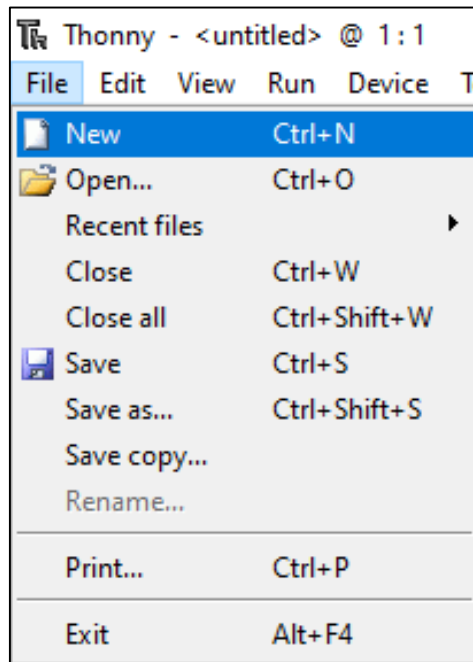
Disadvantage: The program will stop automatically when error occurs or Raspberry Pi Pico is out of power. Code cannot be changed easily.

### Operation

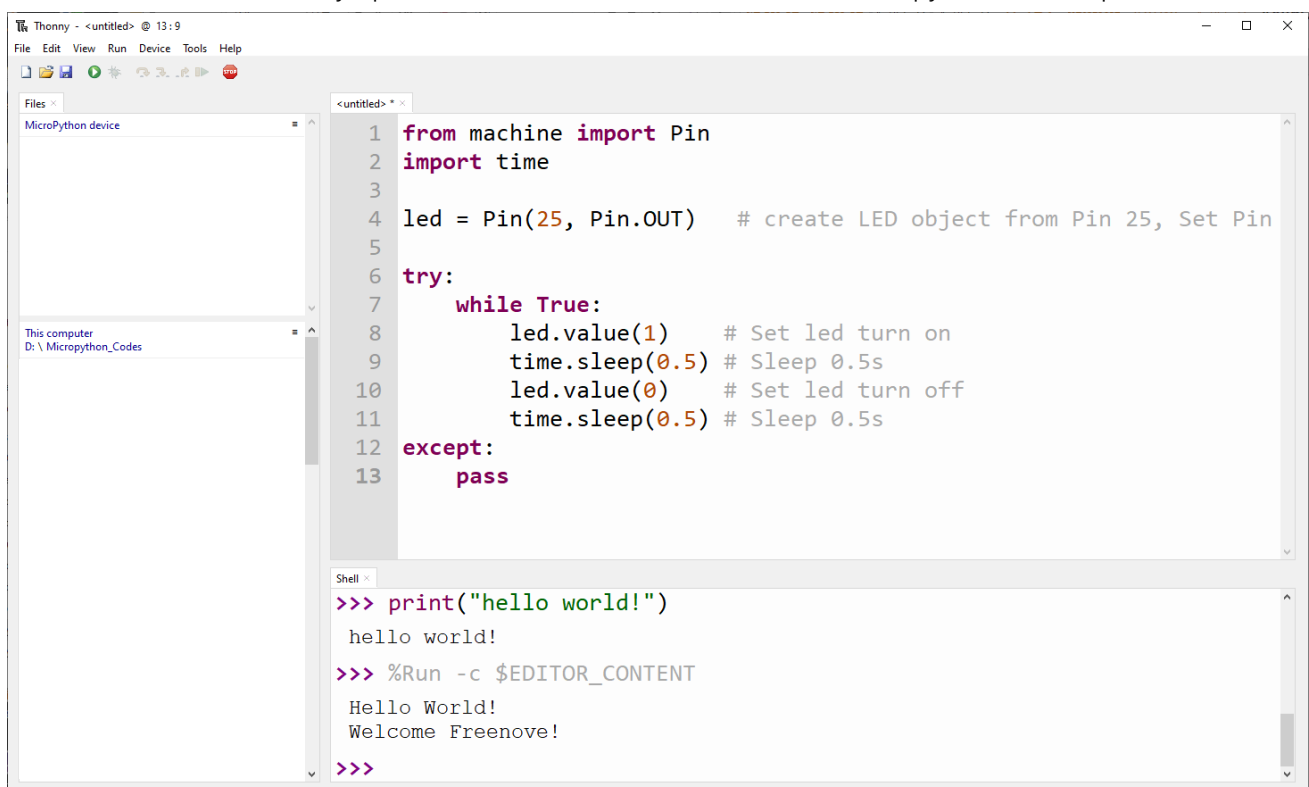
Once powered up, Raspberry Pi Pico will automatically check whether there is main.py existing on the device. If there is, it runs the programs in main.py and then enter shell command system. (If you want the code to run offline, you can save it as main.py); If main.py doesn't exist, it will enter shell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



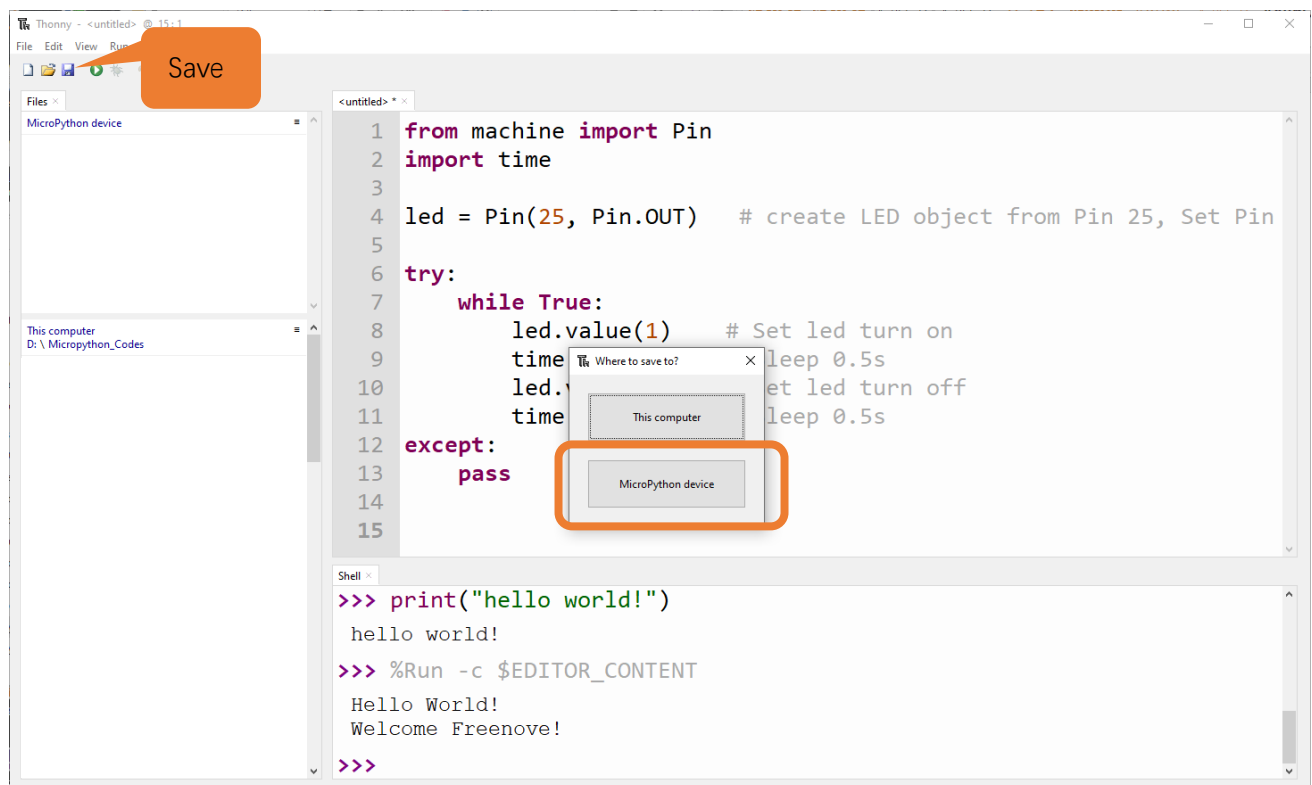
1. Click "File"→"New" to create and write codes.



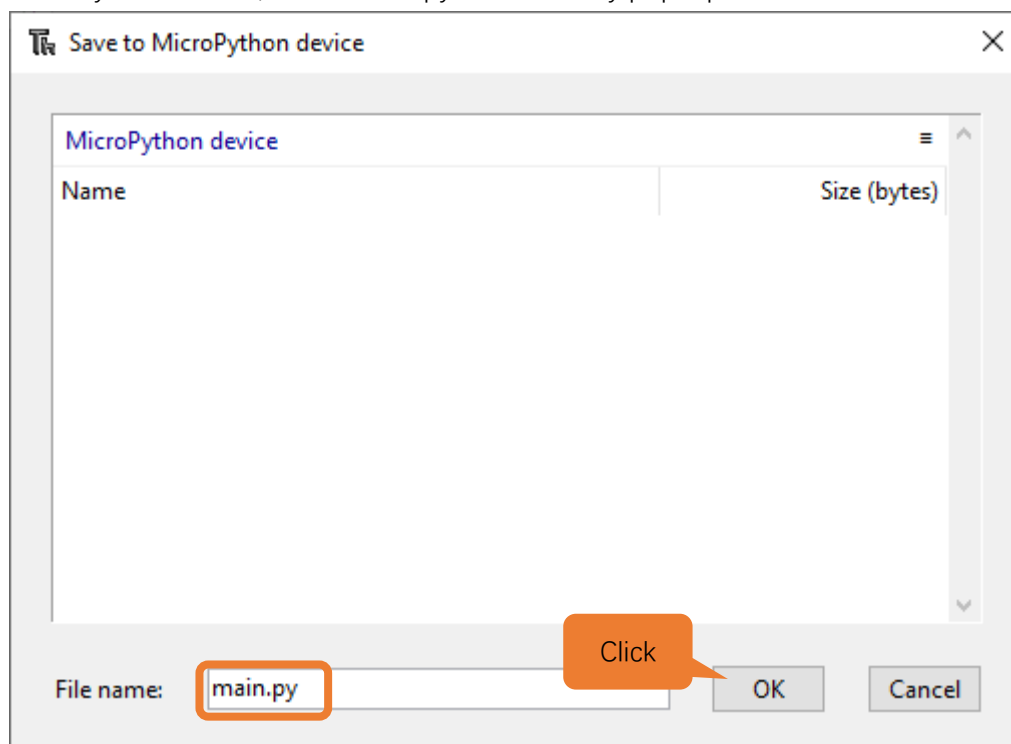
2. Enter codes in the newly opened file. Here we use codes of "00.2\_Blink.py" as an example.



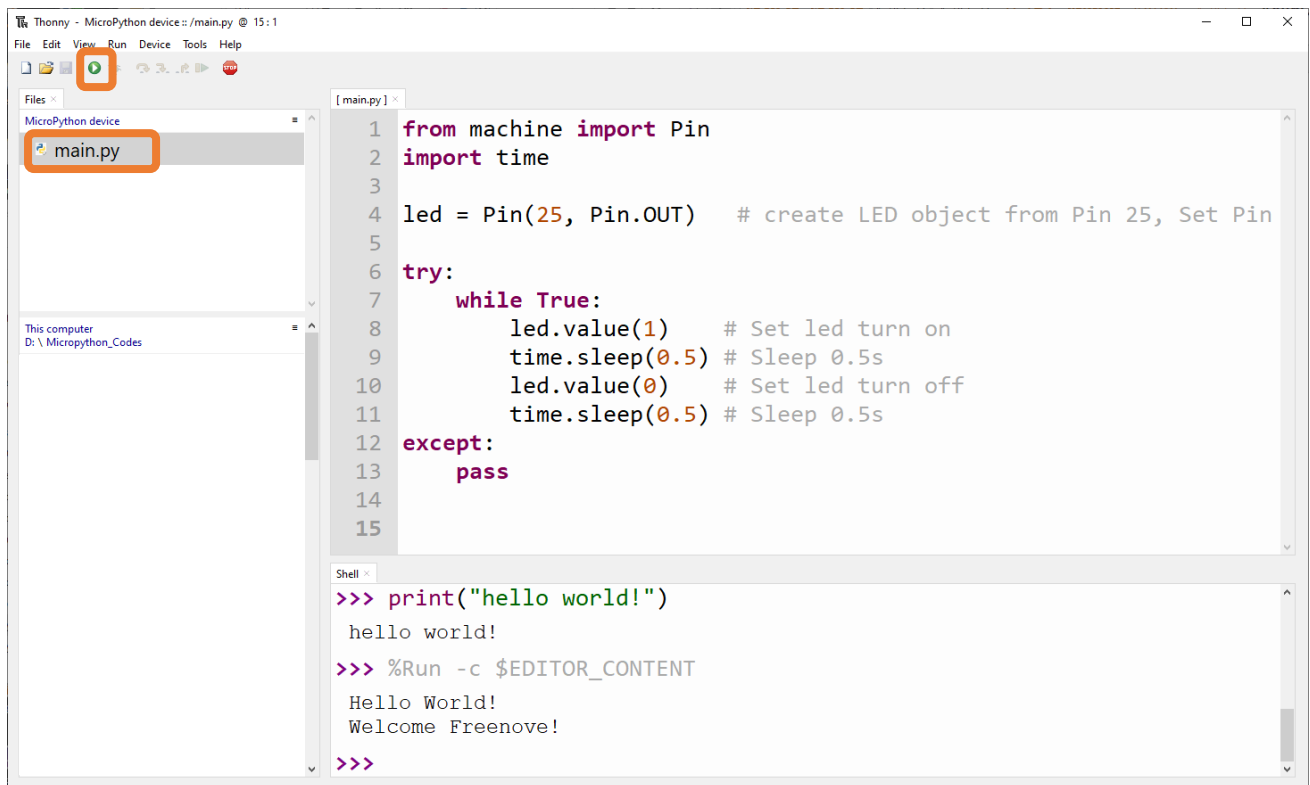
3. Click “Save” on the menu bar. You can save the codes either to your computer or to Raspberry Pi Pico.



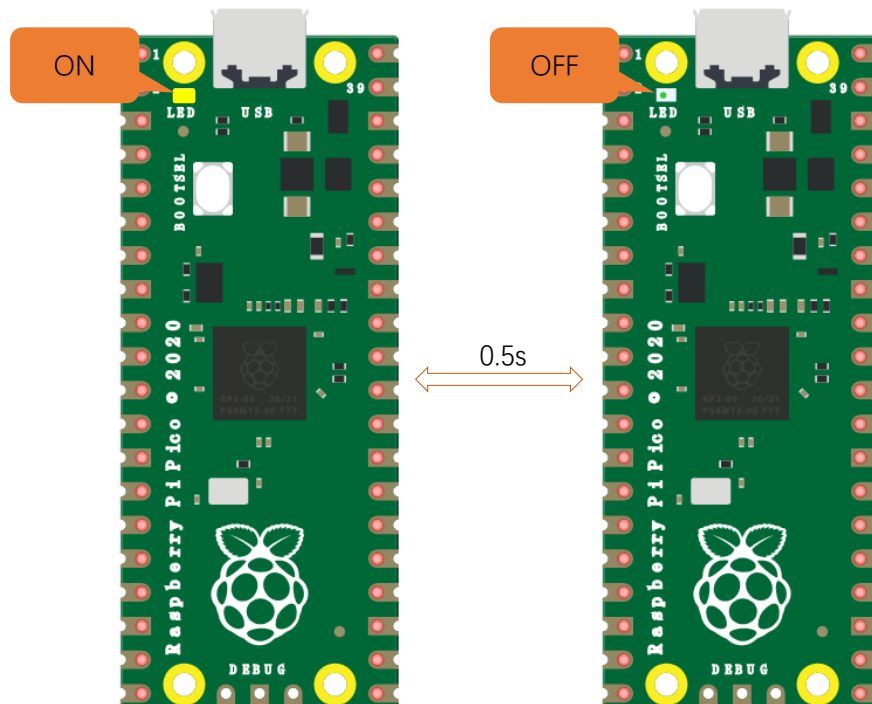
4. Select “MicroPython device”, enter “main.py” in the newly pop-up window and click “OK”.



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

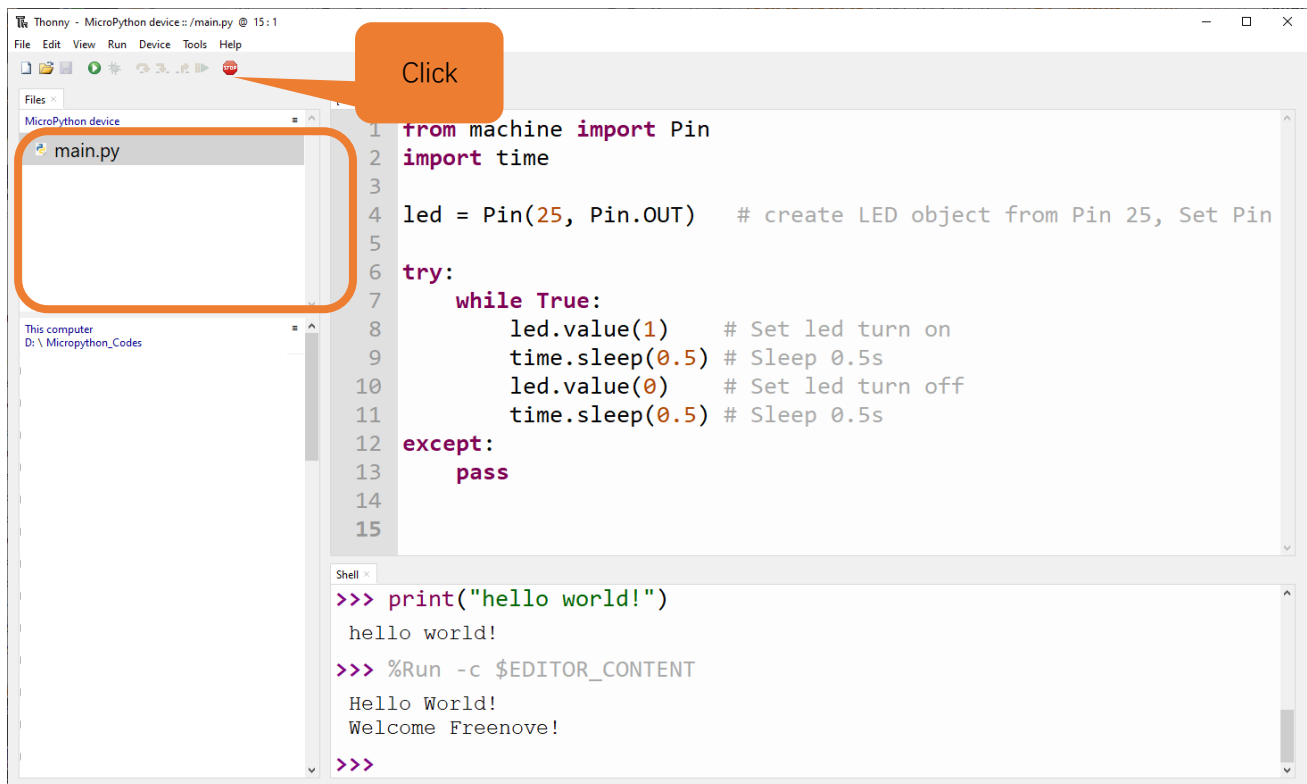


6. Disconnect Raspberry Pi Pico USB cable and then reconnect it, the LED on Raspberry Pi Pico will blink repeatedly.



## Exiting Offline Running

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

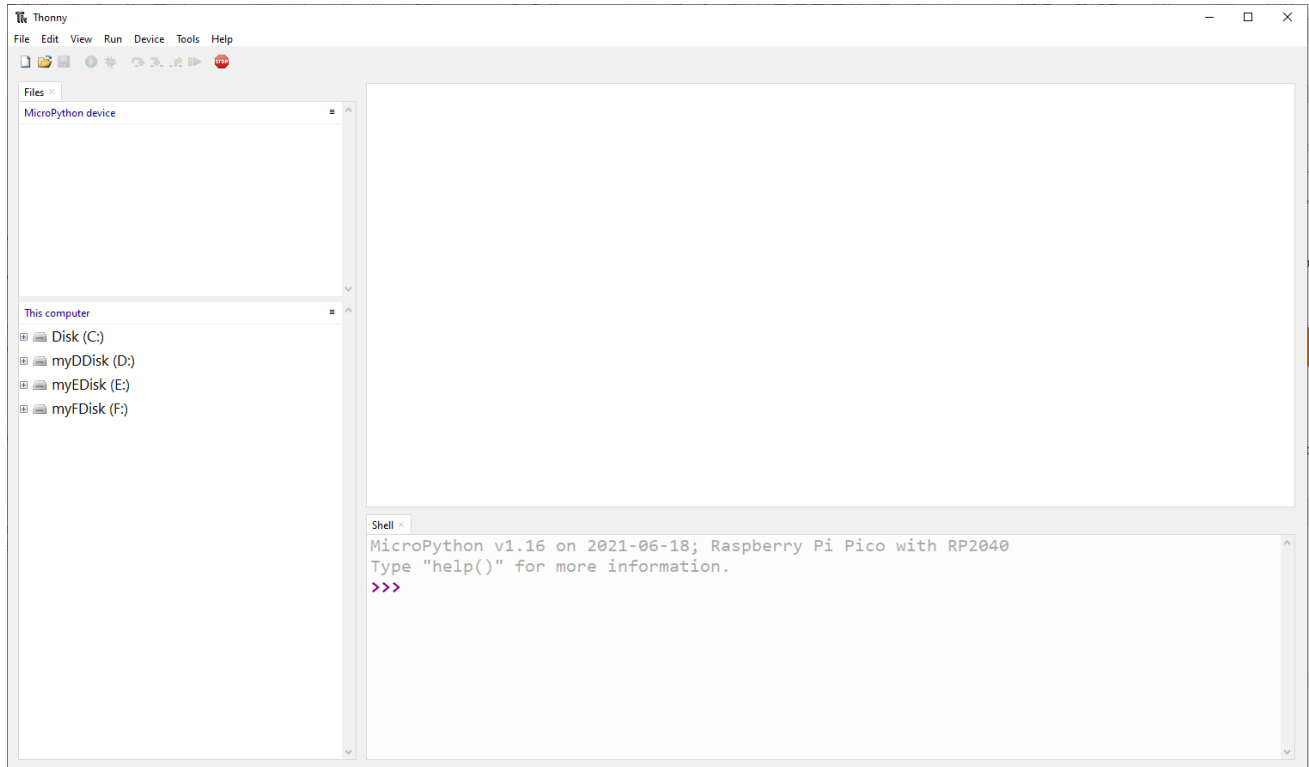


If it doesn't work, please click on “stop/restart backend” for more times or reconnect Pico.

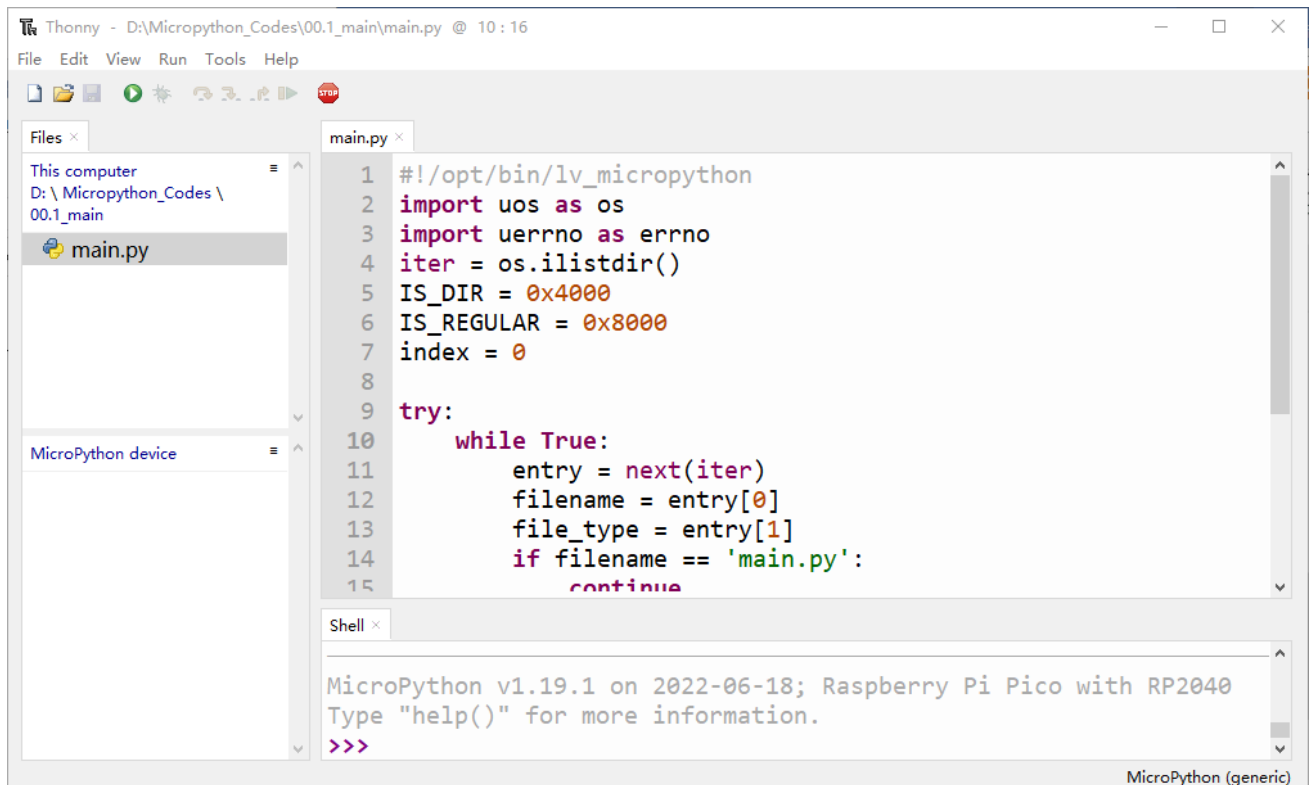


We provide a main.py file for running offline. The code added to main.py is a bootstrap that executes the user's code file. All you need to do is upload the offline project's code file (.py) to the Raspberry Pi Pico device.

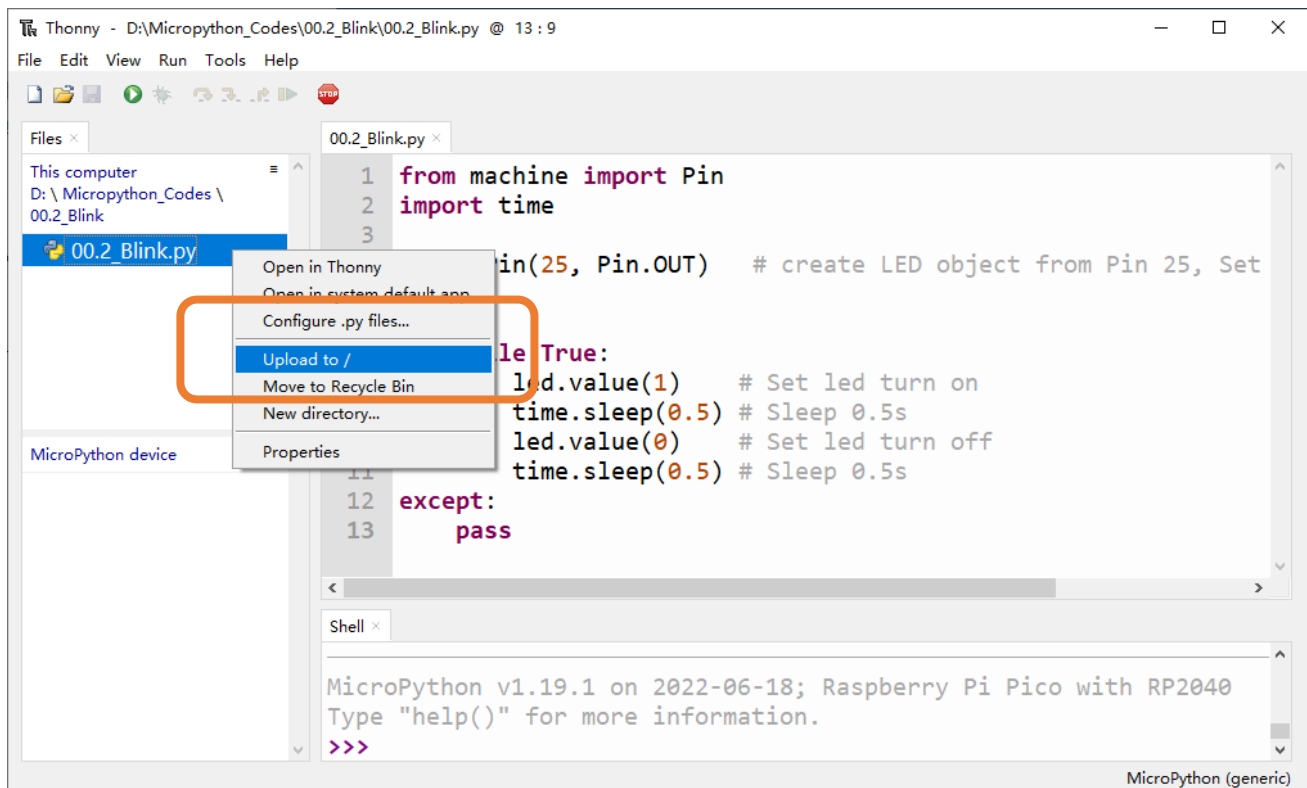
1. Move the program folder “**Freenove\_LCD\_Module/Freenove\_LCD\_Module\_for\_Raspberry\_Pi\_Pico/Python\_Codes**” to disk(D) in advance with the path of “**D:/Micropython\_Codes**”. Open “Thonny”.



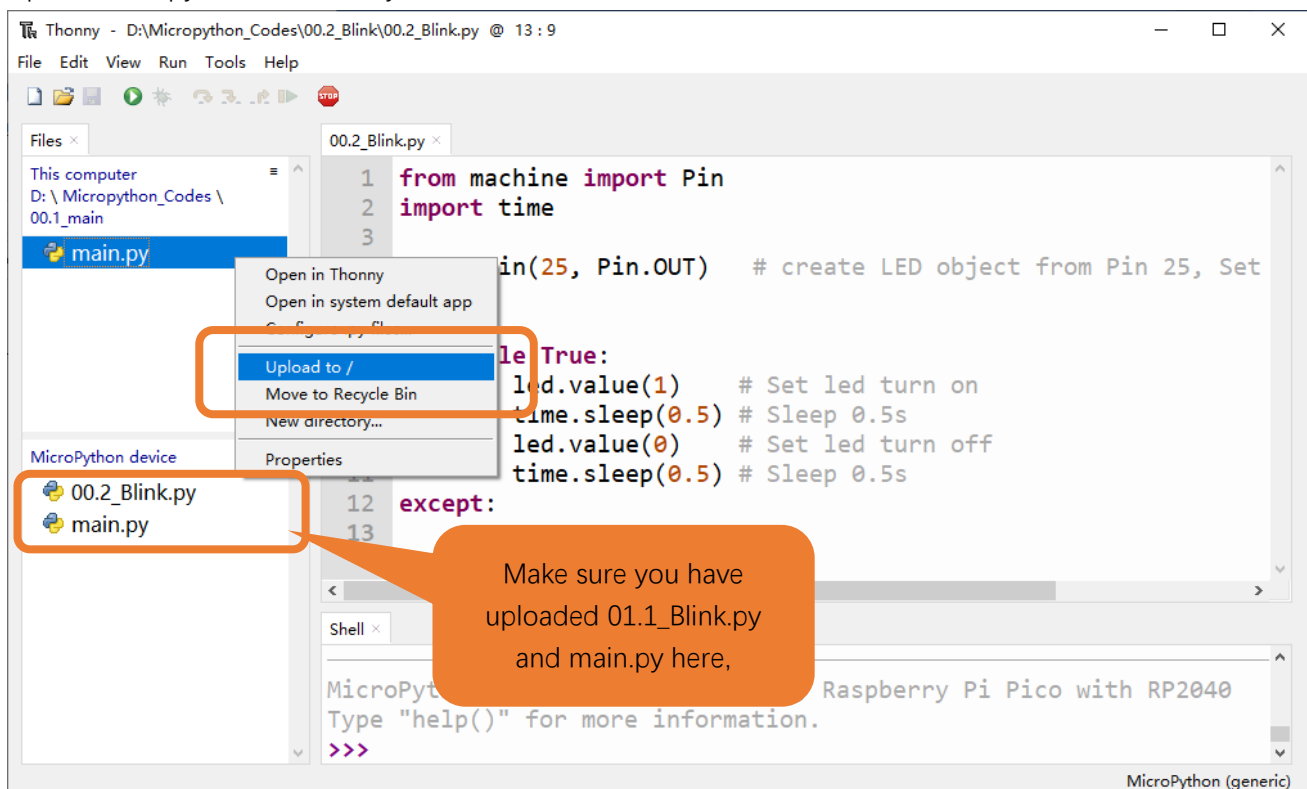
2. Expand “00.1\_main” in the “Micropython\_Codes” in the directory of disk(D), and double-click main.py, which is provided by us to enable programs in “MicroPython device” to run offline.



Here we use 00.1 and 00.2 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 00.2\_Blink.py and select "Upload to /" to upload code to Raspberry Pi Pico.



Upload main.py in the same way.



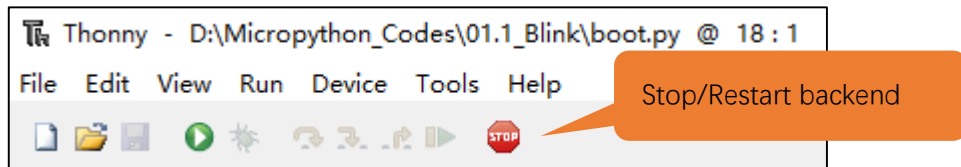
Disconnect Raspberry Pi Pico USB cable and reconnect it, the LED on pico will blink repeatedly.

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



Note:

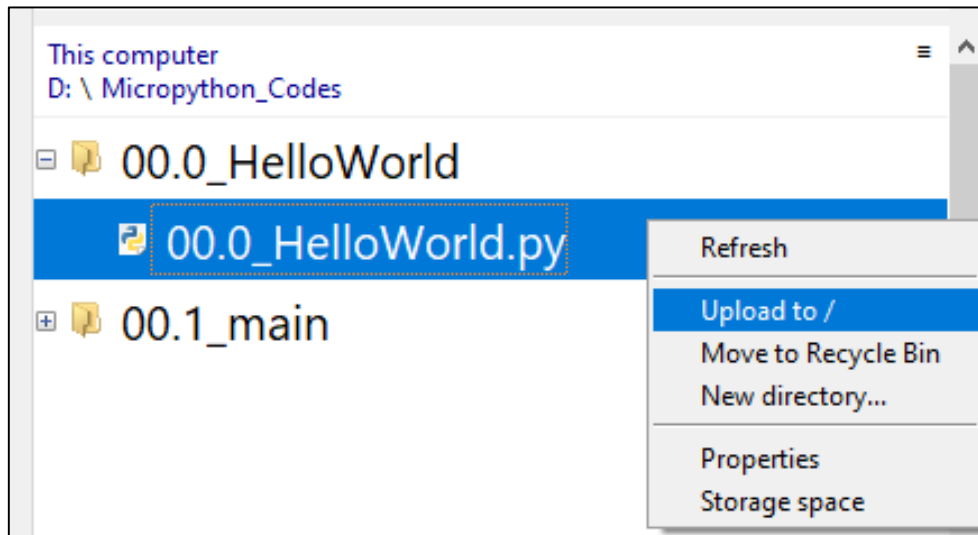
Codes here are run offline. If you want to stop running offline and enter Shell, just click "Stop" in Thonny.



## 0.6 Thonny Common Operation

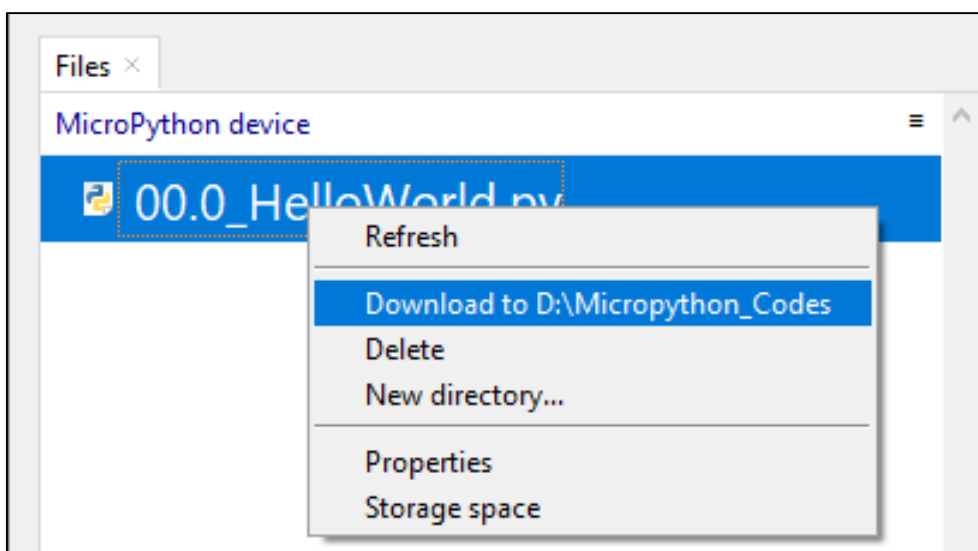
### Uploading Code to Raspberry Pi Pico

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.



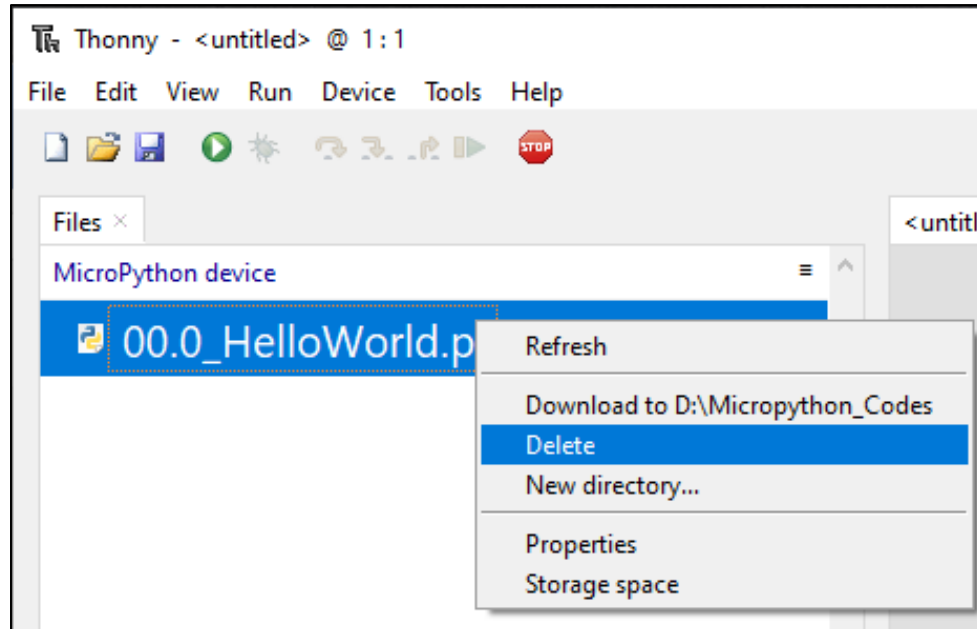
### Downloading Code to Computer

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



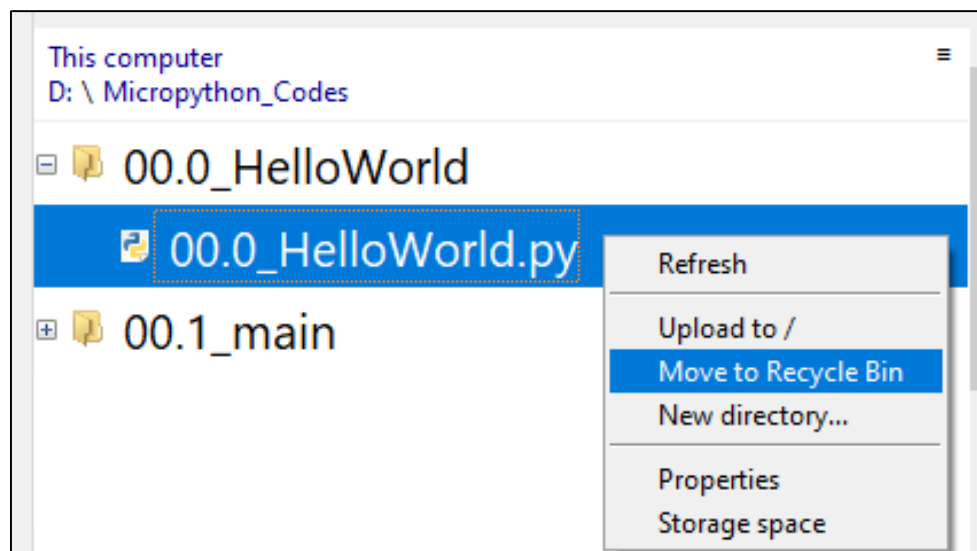
## Deleting Files from Raspberry Pi Pico's Root Directory

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.



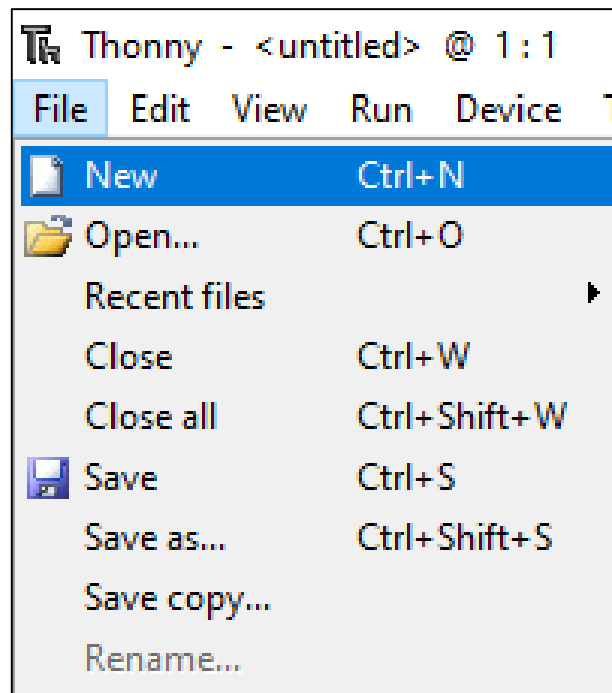
## Deleting Files from your Computer Directory

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".

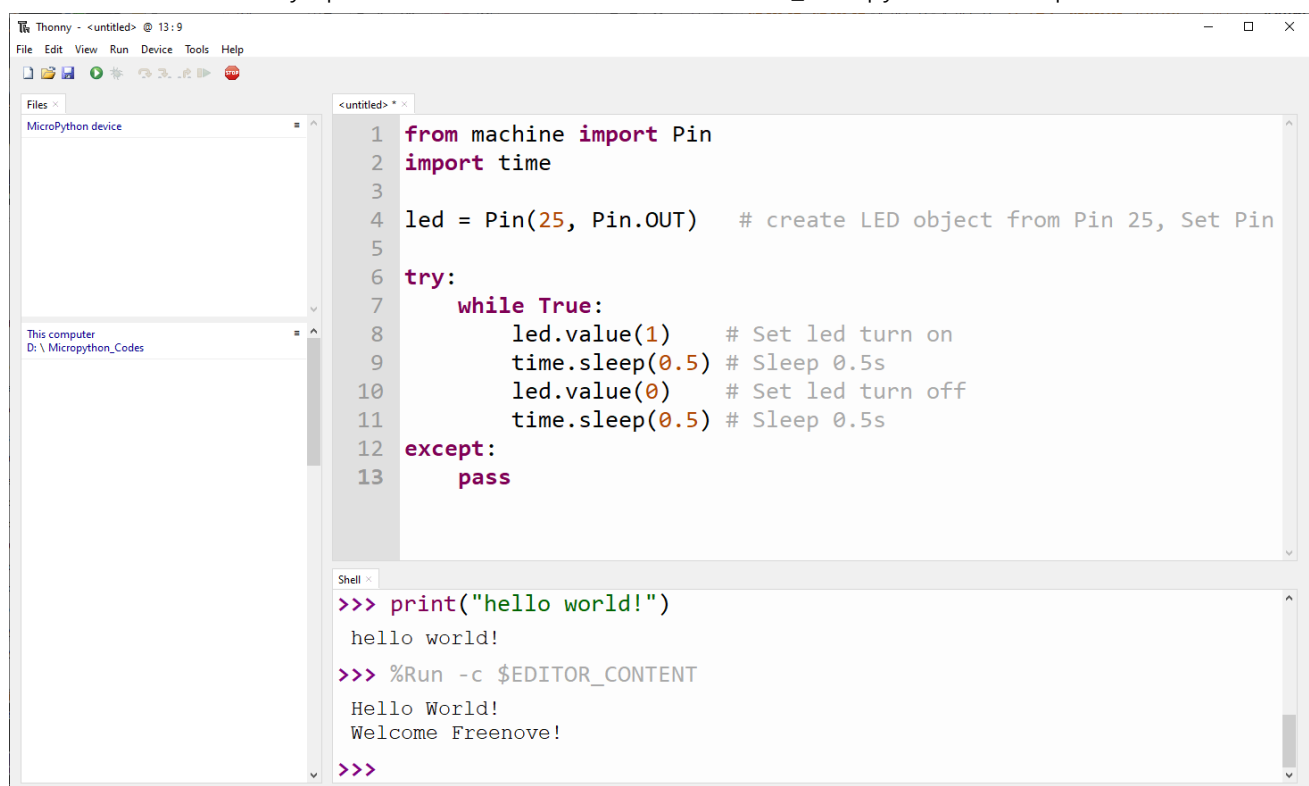


## Creating and Saving the code

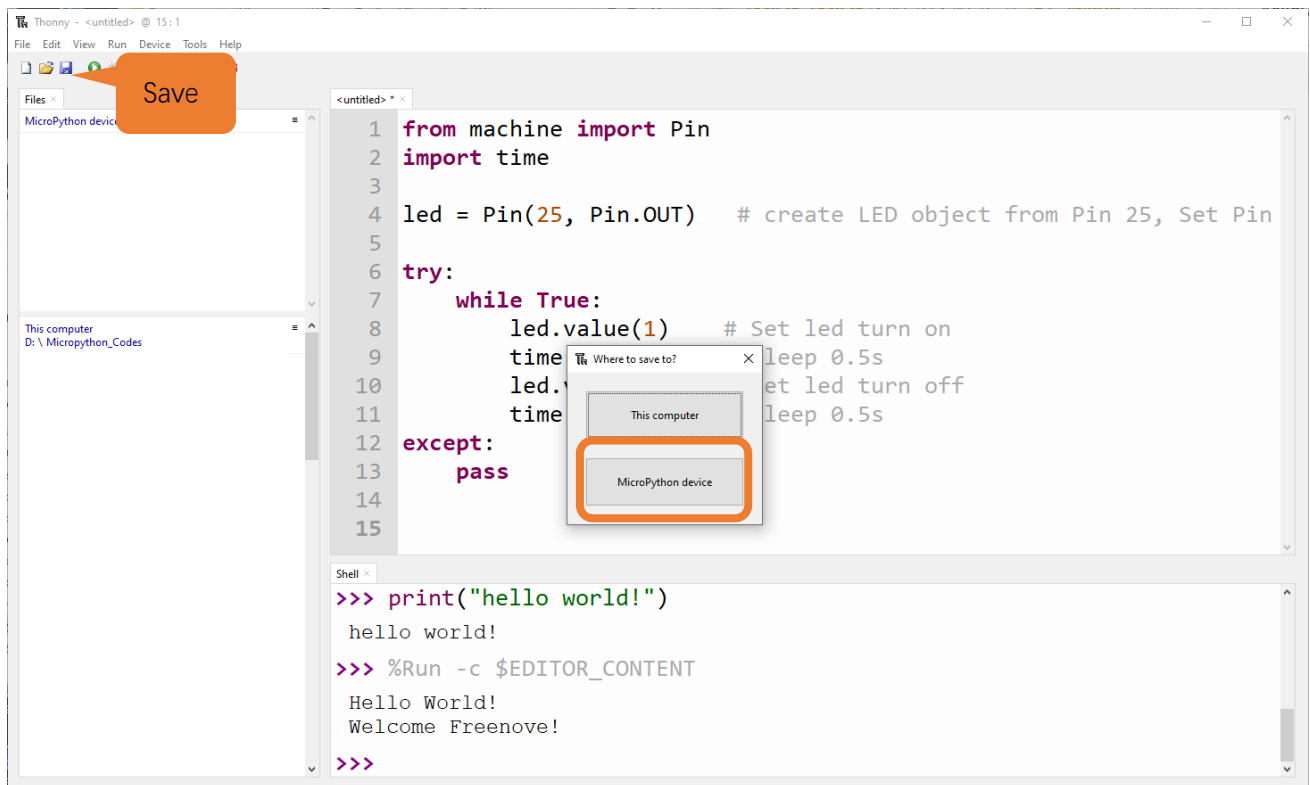
Click “File”→“New” to create and write codes.



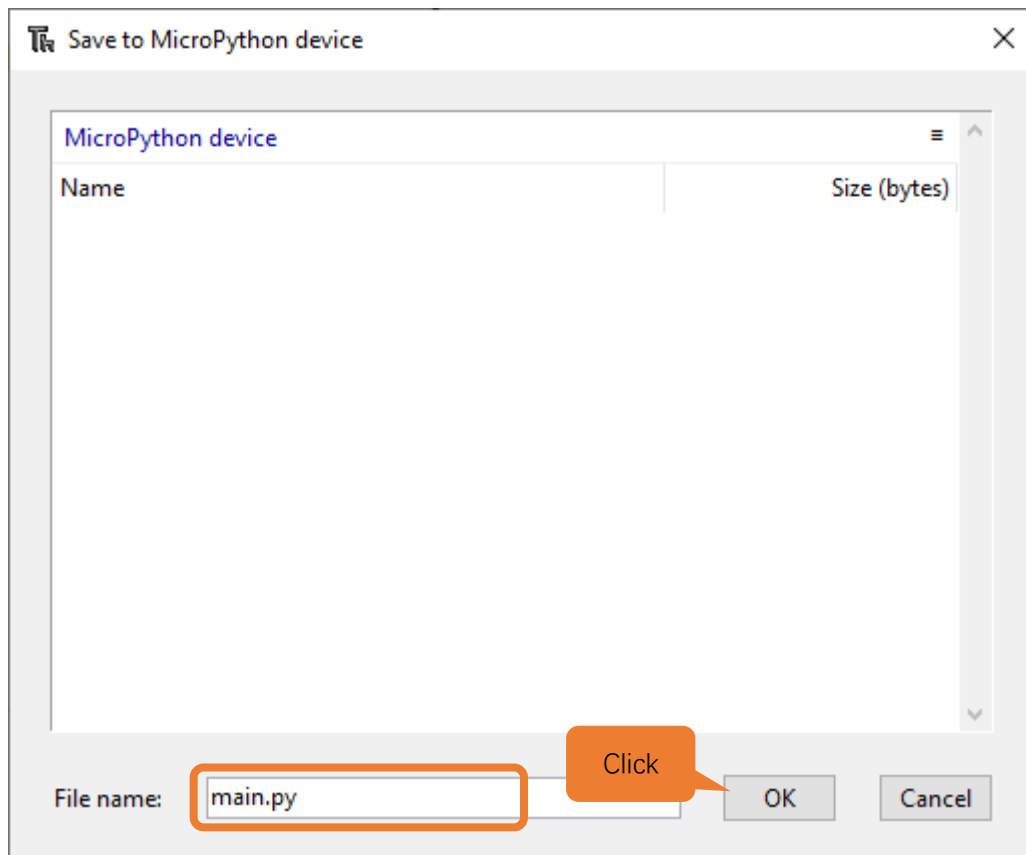
Enter codes in the newly opened file. Here we use codes of “00.2\_Blink.py” as an example.



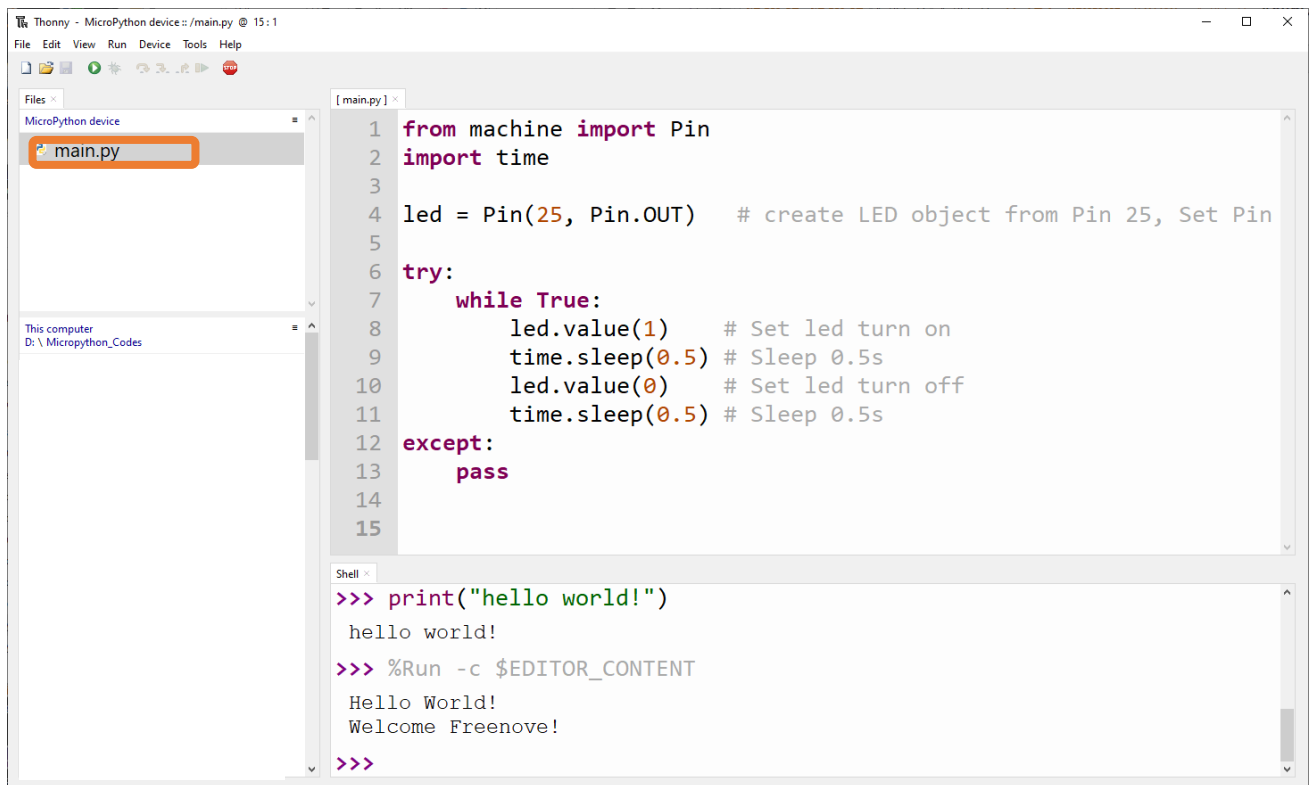
Click “Save” on the menu bar. You can save the codes either to your computer or to Raspberry Pi Pico.



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



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



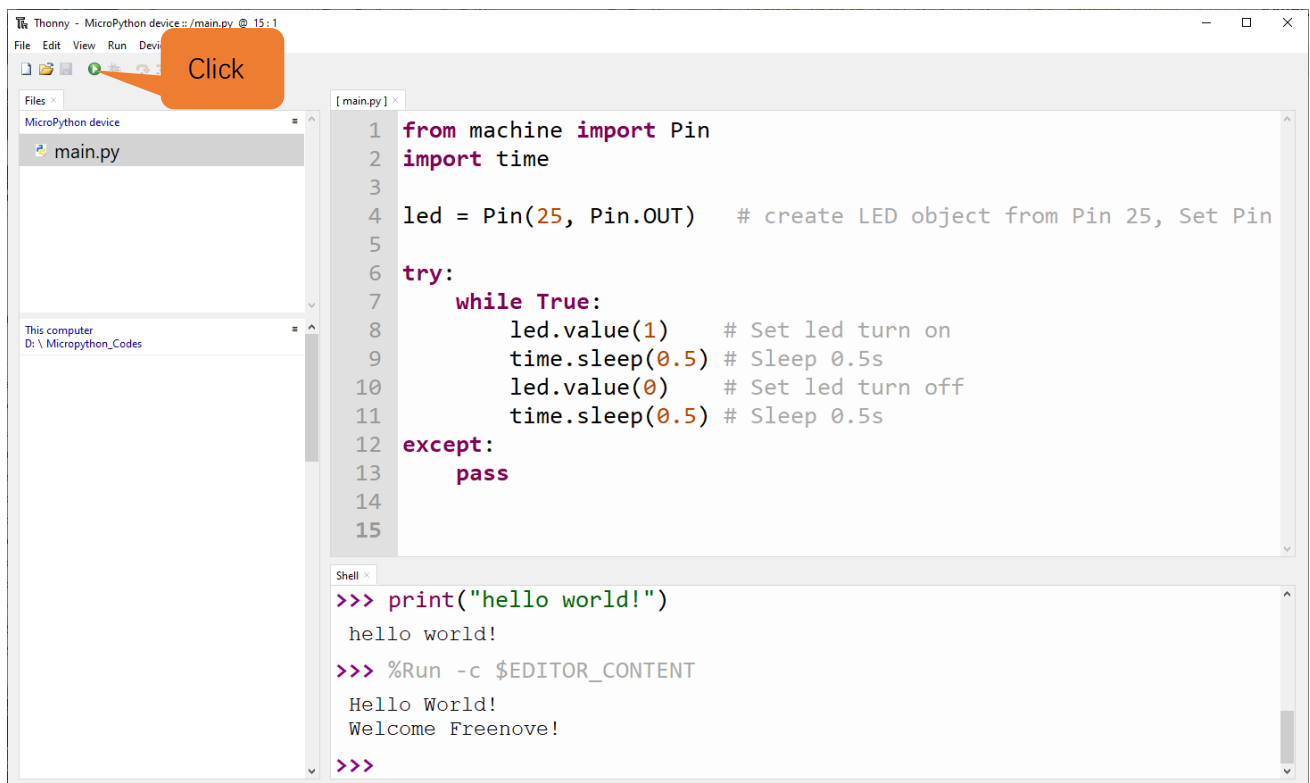
The screenshot shows the Thonny IDE interface. The top menu bar includes File, Edit, View, Run, Device, Tools, and Help. The left sidebar shows the 'Files' pane with 'main.py' selected under the 'MicroPython device' section. The main editor window displays the following Python code:

```
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 bottom pane shows the 'Shell' output:

```
>>> 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.



This screenshot is identical to the previous one, but with an orange callout bubble containing the word 'Click' pointing to the 'Run' button in the top toolbar of the Thonny IDE.

# Chapter 1 LCD1602

In this chapter, we will learn about the LCD1602 Display Screen.

## Project 1.1 LCD1602

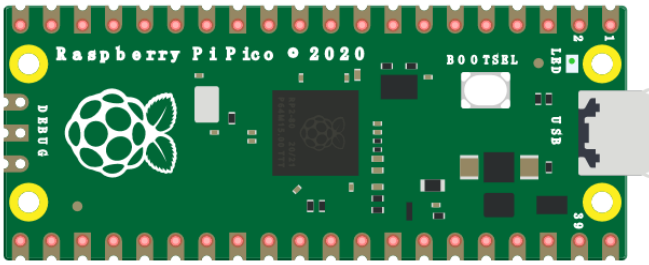
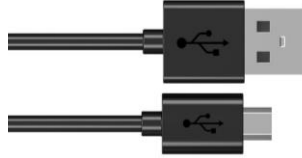
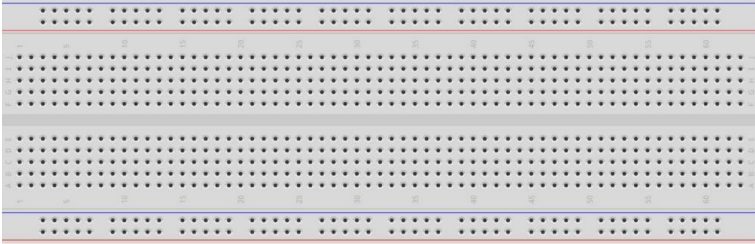
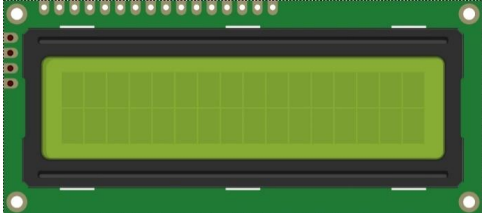

In this section we learn how to use LCD1602 to display something.

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

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

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

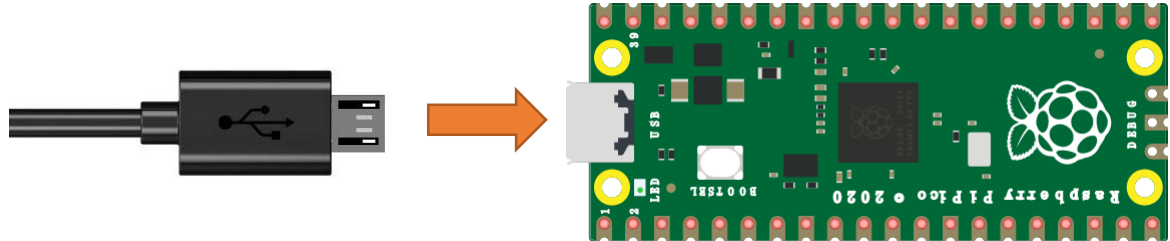
## Component List

Raspberry Pi Pico x1 	USB cable x1 
Breadboard x1 	
LCD1602 Module x1 	Jumper 

## Component knowledge

### Power

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

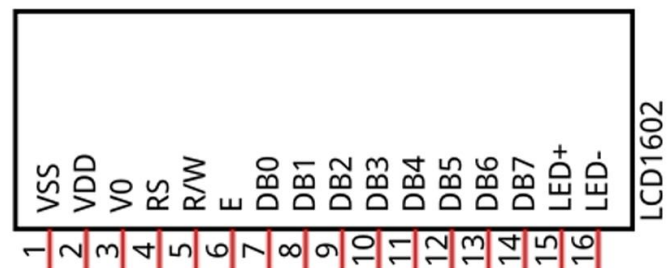
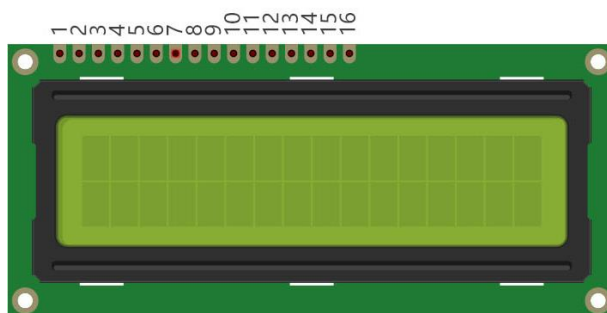


### I2C communication

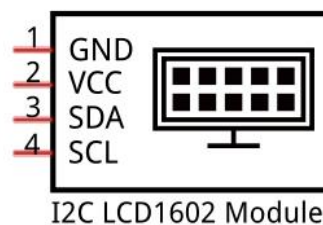
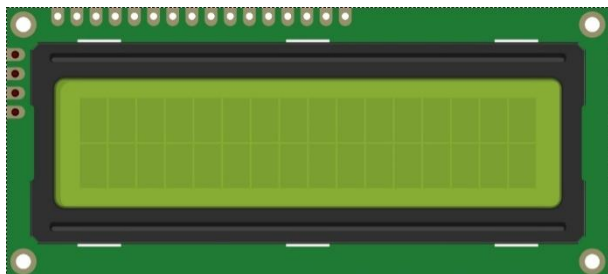
I2C (Inter-Integrated Circuit) is a two-wire serial communication mode, which can be used for the connection of micro controllers and their peripheral equipment. Devices using I2C communication must be connected to the serial data (SDA) line, and serial clock (SCL) line (called I2C bus). Each device has a unique address and can be used as a transmitter or receiver to communicate with devices connected to the bus.

### LCD1602 communication

The LCD1602 Display Screen can display 2 lines of characters in 16 columns. It is capable of displaying numbers, letters, symbols, ASCII code and so on. As shown below is a monochrome LCD1602 Display Screen along with its circuit pin diagram.



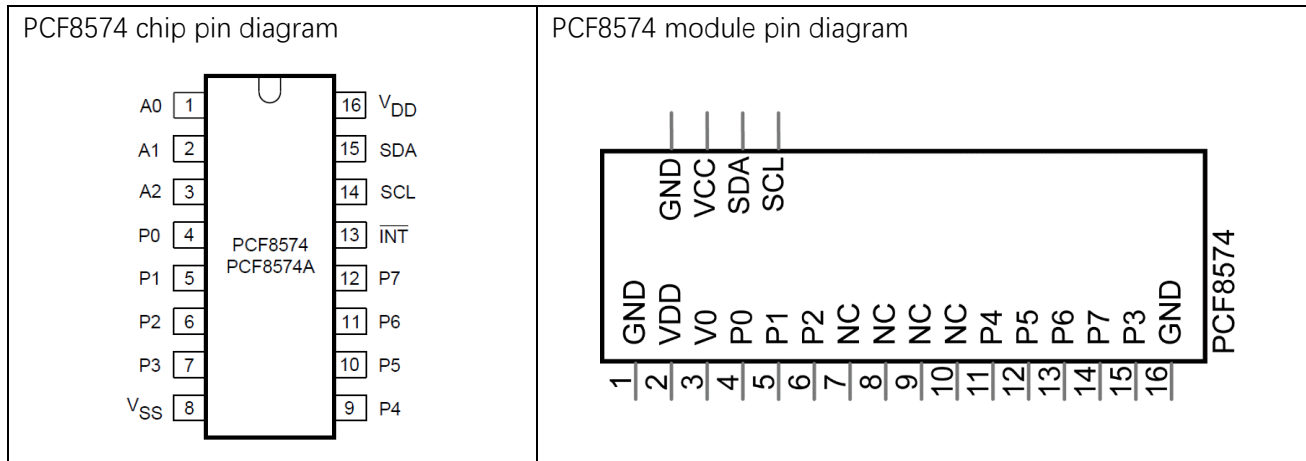
I2C LCD1602 Display Screen integrates an I2C interface, which connects the serial-input & parallel-output module to the LCD1602 Display Screen. This allows us to use only 4 lines to operate the LCD1602.



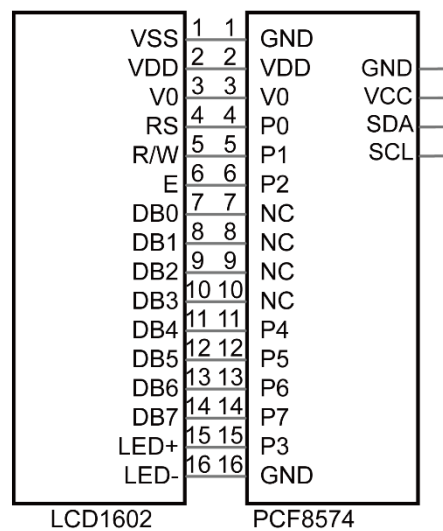
The serial-to-parallel IC chip used in this module is PCF8574T (PCF8574AT), and its default I2C address is 0x27(0x3F).



Below is the PCF8574 pin schematic diagram and the block pin diagram:



PCF8574 module pin and LCD1602 pin are corresponding to each other and connected with each other:



So we only need 4 pins to control the 16 pins of the LCD1602 Display Screen through the I2C interface. In this project, we will use the I2C LCD1602 to display some static characters and dynamic variables.



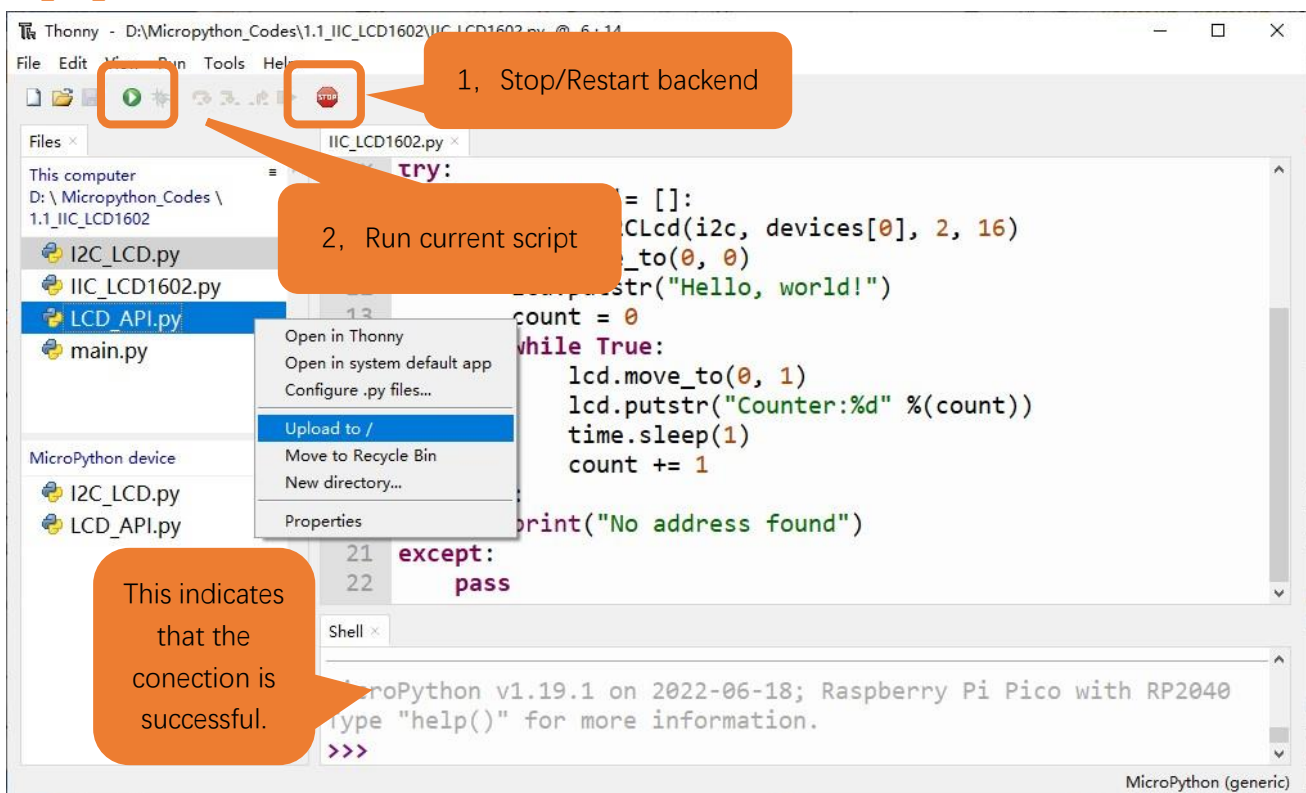
## Code

Codes used in this tutorial are saved in

“**Freenove\_LCD\_Module/Freenove\_LCD\_Module\_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**”.

Open “Thonny”, click “This computer” → “D:” → “Micropython\_Codes” → “1.1\_I2C\_LCD1602”. Select “I2C\_LCD.py” and “LCD\_API.py”, right click your mouse to select “Upload to /”, wait for “I2C\_LCD.py” and “LCD\_API.py” to be uploaded to Raspberry Pi Pico and then double click “I2C\_LCD1602.py”.

### 1.1\_I2C\_LCD1602



Click “Run current script” and LCD1602 displays some characters. Press Ctrl+C or click “Stop/Restart backend” to exit the program.



Note: If you cannot see anything on the display or the display is not clear, try rotating the white knob on back of LCD1602 slowly, which adjusts the contrast, until the screen can display clearly.



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.

```

Shell ×

MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

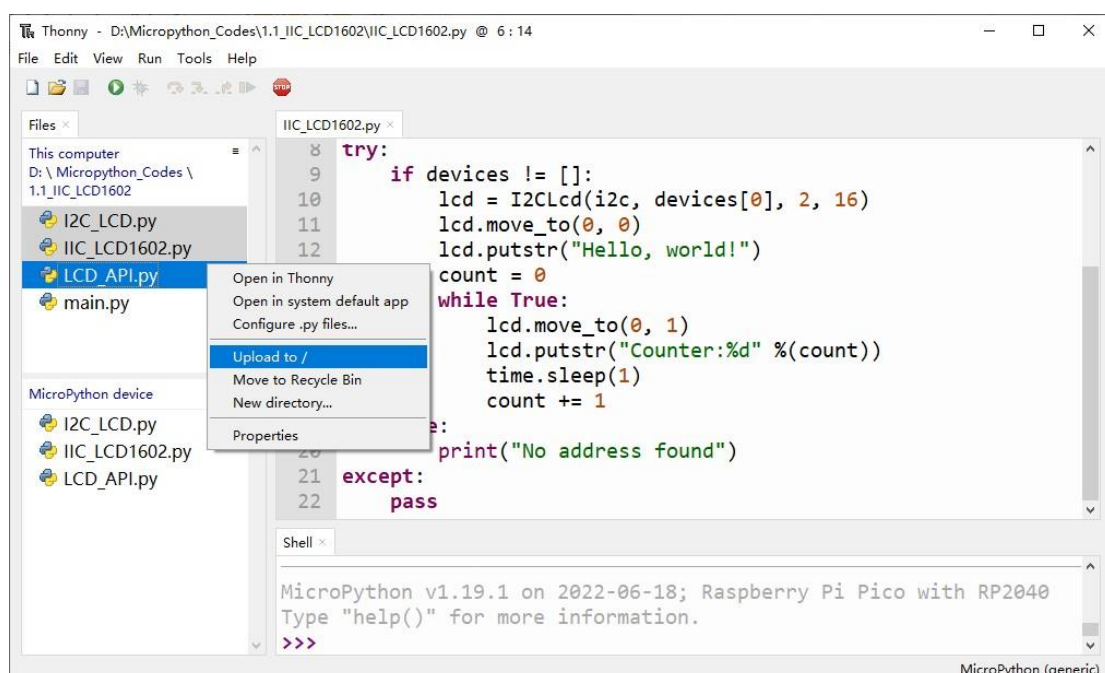
Connection lost (EOF)

Use Stop/Restart to reconnect.

MicroPython (generic)
  
```

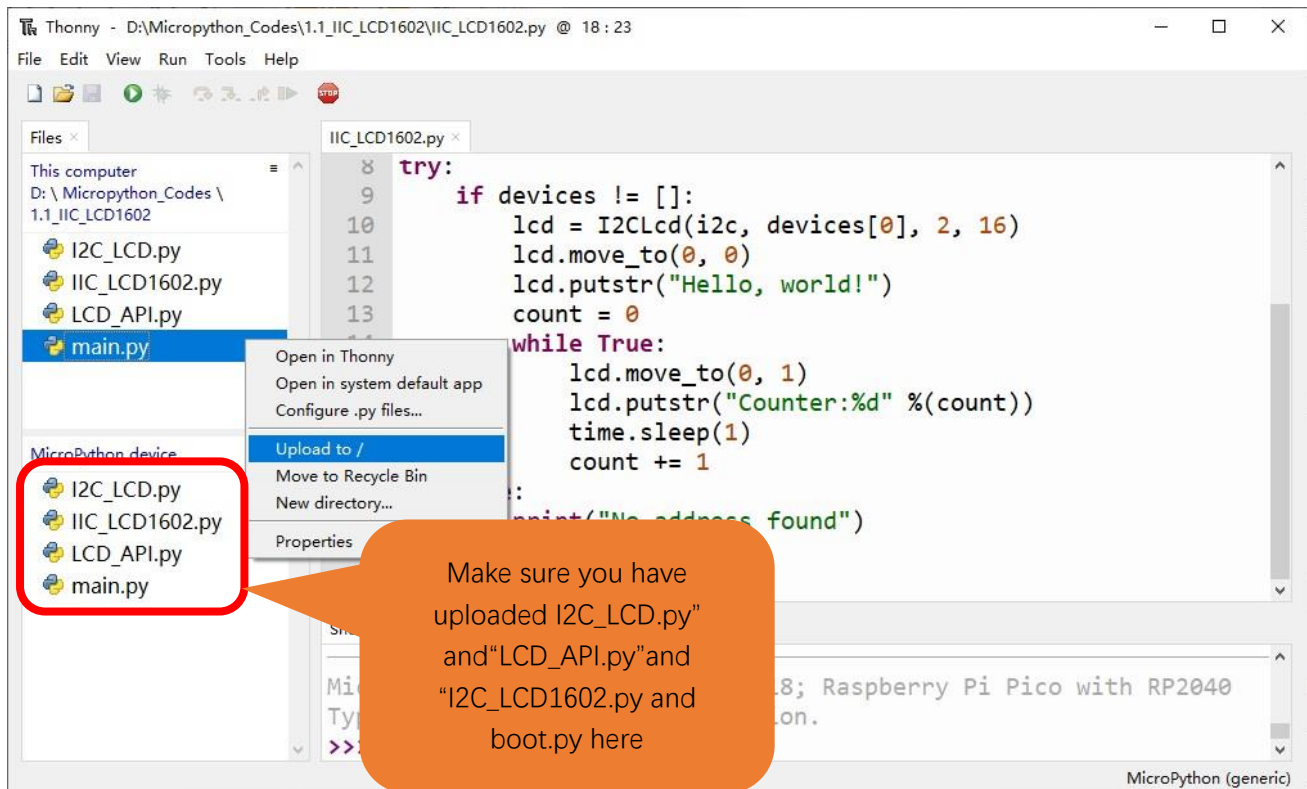
### Uploading code to Raspberry Pi Pico

As shown in the following illustration, Select "I2C\_LCD.py" and "LCD\_API.py" and "I2C\_LCD1602.py", right click your mouse to select "Upload to /" to upload code to Raspberry Pi Pico.



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

Upload main.py in the same way.

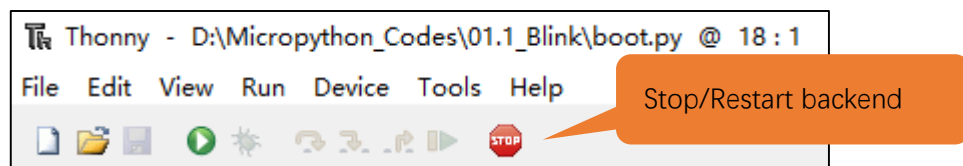


Disconnect Raspberry Pi Pico USB cable and reconnect it, you can see LCD1602 displays some characters.



Note:

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



The following is the program code:

```

1  import time
2  from machine import I2C, Pin
3  from I2C_LCD import I2CLcd
4
5  i2c = I2C(0, sda=Pin(4), scl=Pin(5), freq=400000)
6  devices = i2c.scan()
7
8  try:
9      if devices != []:
```

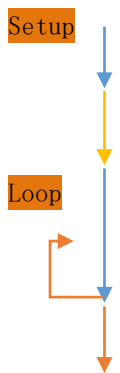


```

10     lcd = I2CLcd(i2c, devices[0], 2, 16)
11     lcd.move_to(0, 0)
12     lcd.putstr("Hello, world!")
13     count = 0
14     while True:
15         lcd.move_to(0, 1)
16         lcd.putstr("Counter:%d" %(count))
17         time.sleep(1)
18         count += 1
19     else:
20         print("No address found")
21 except:
22     pass

```

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.



```

1     import time
2     from machine import I2C, Pin
3     from I2C_LCD import I2CLcd
4
8     try:
14         while True:
...             ...
21     except:
22         pass

```

Import time, I2C and I2C\_LCD modules.

```

1     import time
2     from machine import I2C, Pin
3     from I2C_LCD import I2CLcd

```

Create an I2C object, initialize the I2C parameter configuration, and associate it with the LCD1602 pin. Call the scan() function to query the LCD1602 device address.

```

4     i2c = I2C(0, sda=Pin(4), scl=Pin(5), freq=400000)
5     devices = i2c.scan()

```

Use the if statement to determine whether the queried I2C device address is empty. If it is not empty, create an I2CLcd object and set the created I2C object, I2C device address, and the number of rows and columns of LCD1602; if it is empty, print out "No address" found" and the program exits.

```

9     if devices != []:
10         lcd = I2CLcd(i2c, devices[0], 2, 16)
...
19     else:
20         print("No address found")

```

Move the cursor of LCD1602 to the first row, first column, and print out "Hello, world!"

```

11     lcd.move_to(0, 0)

```

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

```
12    lcd.putstr("Hello, world!")
```

The second line of LCD1602 continuously prints the number of seconds after the Raspberry Pi Pico program runs.

```
14    while True:
15        lcd.move_to(0, 1)
16        lcd.putstr("Counter:%d" %(count))
17        time.sleep(1)
18        count += 1
```

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.

```
8   try:
...
21  except:
22      pass
```

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.

```
14      while True:
15          lcd.move_to(0, 1)
16          lcd.putstr("Counter:%d" %(count))
17          time.sleep(1)
18          count += 1
```

### How to import python files

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

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

```
import random

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

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

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

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

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



## Reference

**Class machine**

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

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

**freq\_val**: 125000000Hz(125MHz).

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

**machine.unique\_id()**: Obtains MAC address of the device.

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

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

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

**machine.time\_pulse\_us(pin, pulse\_level, timeout\_us=1000000)**:

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

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

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

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

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

**Class time**

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

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

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

**time.sleep\_ms(ms)**: Sleeps for the given number of milliseconds, ms should be an int.

**time.sleep\_us(us)**: Sleeps for the given number of microseconds, us should be an int.

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

**time.ticks\_ms()**: Returns the incrementing millisecond counter value, which recounts after some values.

**time.ticks\_us()**: Returns microsecond.

**time.ticks\_cpu()**: Similar to `ticks_ms()` and `ticks_us()`, but it is more accurate(return clock of CPU).

**time.ticks\_add(ticks, delta)**: Gets the timestamp after the offset.

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

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

**time.ticks\_diff(old\_t, new\_t)**: Calculates the interval between two timestamps, such as `ticks_ms()`, `ticks_us()` or `ticks_cpu()`.

**old\_t**: Starting time.

**new\_t**: Ending time.

### Class Pin(id, mode, pull, value)

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

**id**: Arbitrary pin number.

**mode**: Mode of pins.

**Pin.IN**: Input Mode.

**Pin.OUT**: Output Mode.

**Pin.OPEN\_DRAIN**: Open-drain Mode.

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

**None**: No pull up or pull down resistors.

**Pin.PULL\_UP**: Pull-up Mode, outputting high level by default.

**Pin.PULL\_DOWN**: Pull-down Mode, outputting low level by default.

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

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

**Pin.value([value])**: Obtain or set state of the pin level, return 0 or 1 according to the logic level of pins. Without parameter, it reads input level. With parameter given, it is to set output level.

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

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

**Pin.IRQ\_FALLING**: interrupt on falling edge.

**Pin.IRQ\_RISING**: interrupt on rising edge.

**Handler**: callback function.

### Class I2CLcd

Before each use of the object **I2CLcd**, please make sure that **I2C\_LCD.py** and **LCD\_API.py** have been uploaded to "/" of Raspberry Pi Pico, and then add the statement "**from I2C\_LCD import I2CLcd**" to the top of the python file.

**clear()**: Clear the LCD1602 screen display.

**show\_cursor()**: Show the cursor of LCD1602.

**hide\_cursor()**: Hide the cursor of LCD1602.

**blink\_cursor\_on()**: Turn on cursor blinking.

**blink\_cursor\_off()**: Turn off cursor blinking.

**display\_on()**: Turn on the display function of LCD1602.

**display\_off()**: Turn on the display function of LCD1602.

**backlight\_on()**: Turn on the backlight of LCD1602.

**backlight\_off()**: Turn on the backlight of LCD1602.

**move\_to(cursor\_x, cursor\_y)**: Move the cursor to a specified position.

**cursor\_x**: Column cursor\_x.

**cursor\_y**: Row cursor\_y.

**putchar(char)**: Print the character in the bracket on LCD1602.

**putstr(string)**: Print the string in the bracket on LCD1602.

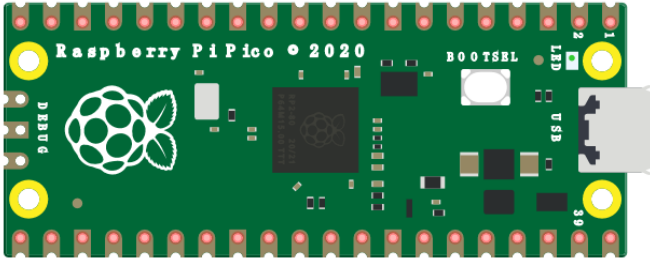
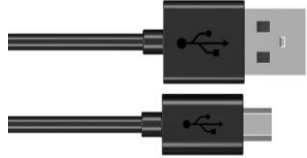
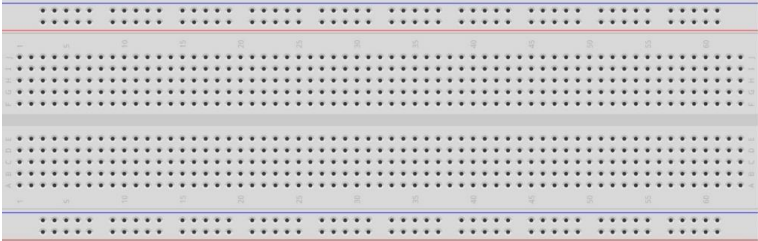
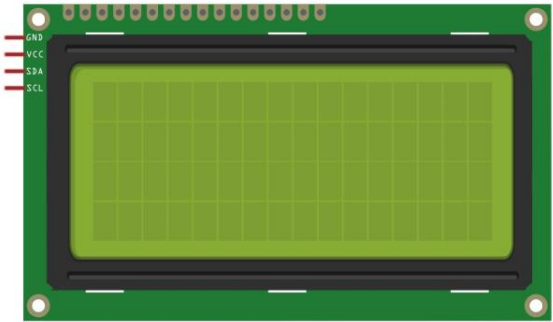

## Chapter 2 LCD2004

In the previous chapter, we studied the LCD1602 display. In order to display more content, In this chapter, we will learn about the LCD2004 Display Screen.

### Project 1.1 LCD2004

In this section we learn how to use lcd2004 to display something.

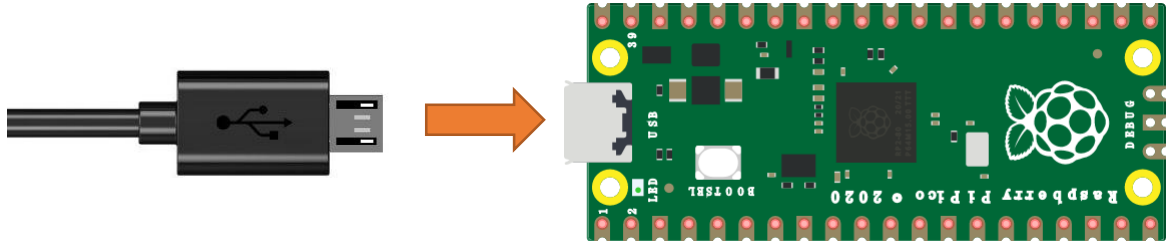
### Component List

Raspberry Pi Pico x1 	USB cable x1 
Breadboard x1 	
LCD2004 Module x1 	Jumper 

## Component knowledge

### Power

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

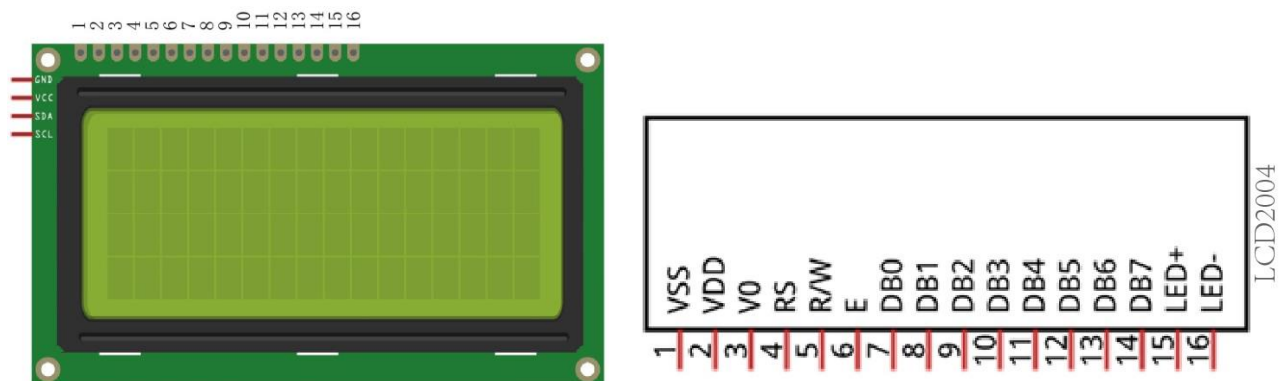


### I2C communication

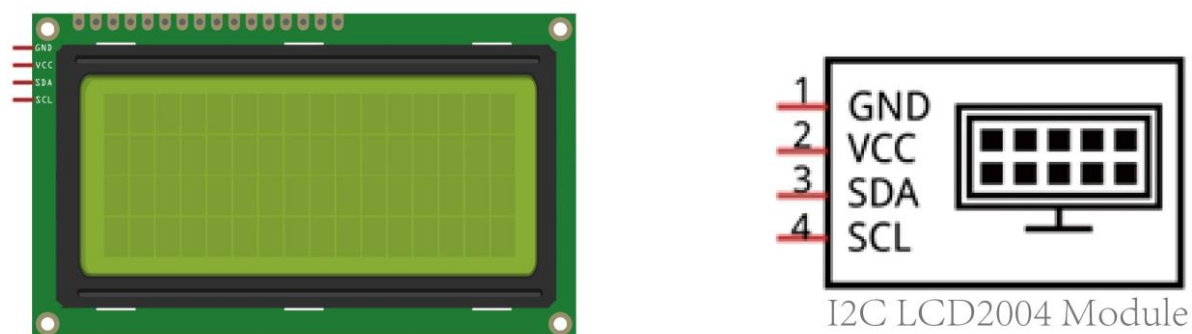
I2C (Inter-Integrated Circuit) is a two-wire serial communication mode, which can be used for the connection of micro controllers and their peripheral equipment. Devices using I2C communication must be connected to the serial data (SDA) line, and serial clock (SCL) line (called I2C bus). Each device has a unique address and can be used as a transmitter or receiver to communicate with devices connected to the bus.

### LCD2004 communication

The LCD2004 Display Screen can display 4 lines of characters in 20 columns. It is capable of displaying numbers, letters, symbols, ASCII code and so on. As shown below is a monochrome LCD2004 Display Screen along with its circuit pin diagram



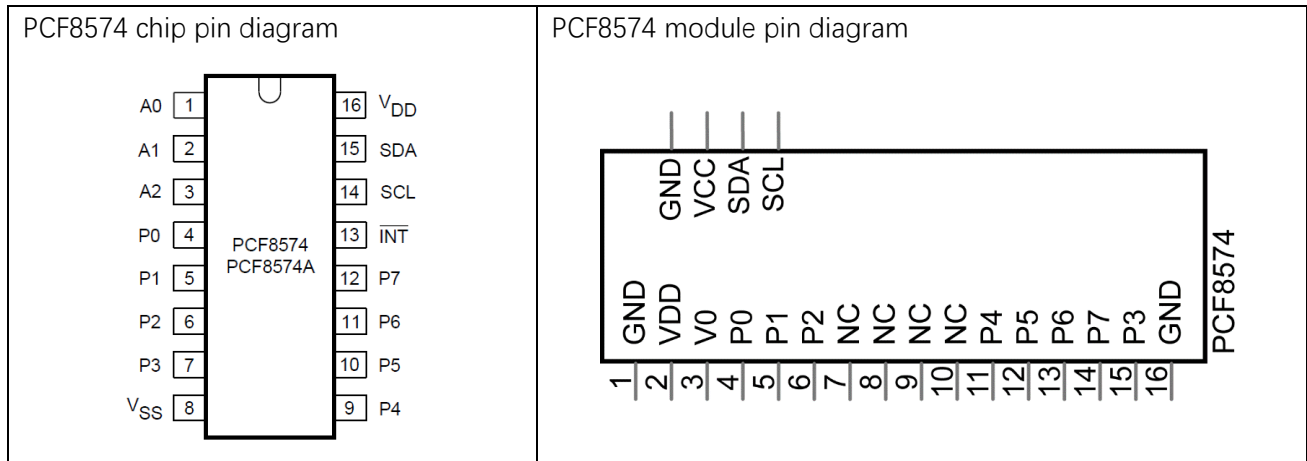
I2C LCD2004 Display Screen integrates an I2C interface, which connects the serial-input & parallel-output module to the LCD2004 Display Screen. This allows us to use only 4 lines to operate the LCD2004.



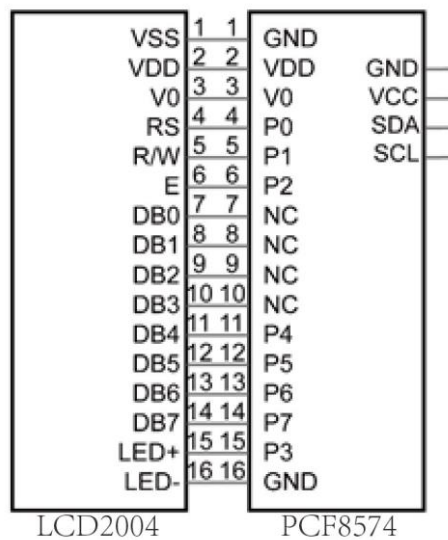
The serial-to-parallel IC chip used in this module is PCF8574T (PCF8574AT), and its default I2C address is 0x27(0x3F).

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

Below is the PCF8574 pin schematic diagram and the block pin diagram:



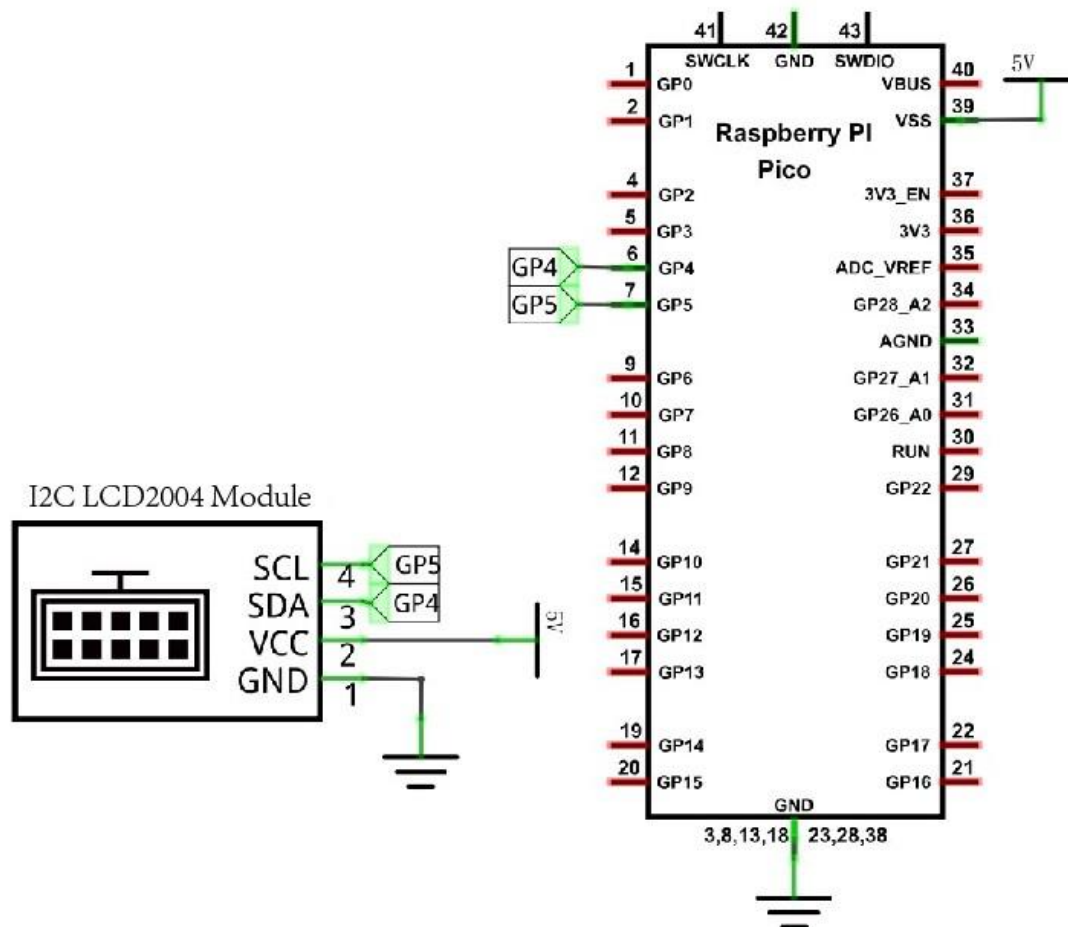
PCF8574 module pin and LCD2004 pin are corresponding to each other and connected with each other:



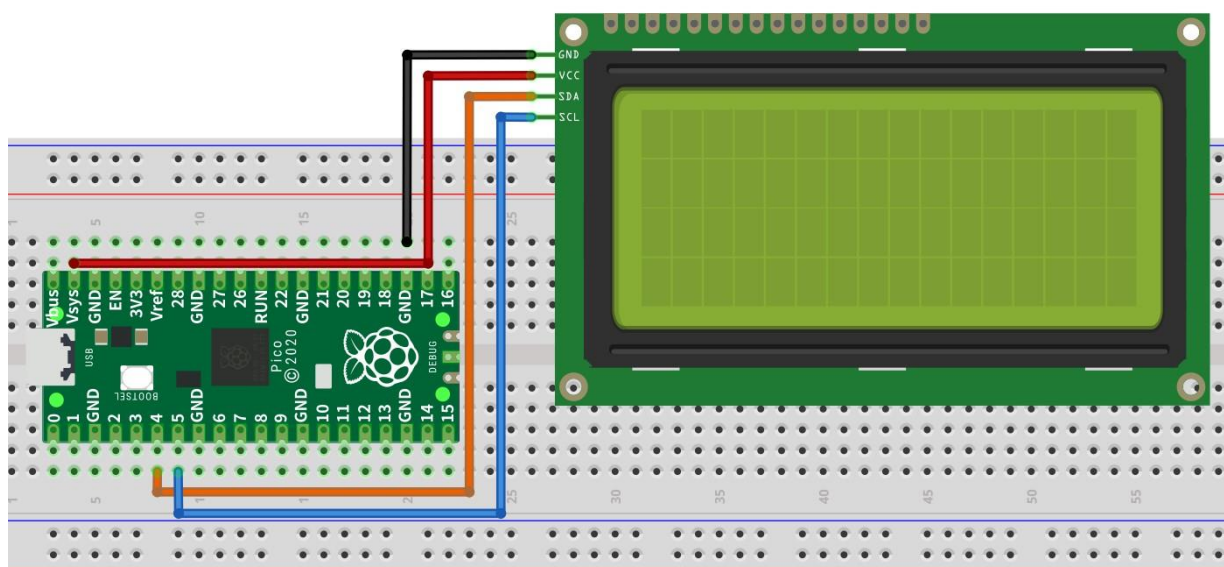
So we only need 4 pins to control the 16 pins of the LCD2004 display screen through the I2C interface. In this project, we will use the I2C LCD2004 to display some static characters and dynamic variables.

## Circuit

Schematic diagram



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





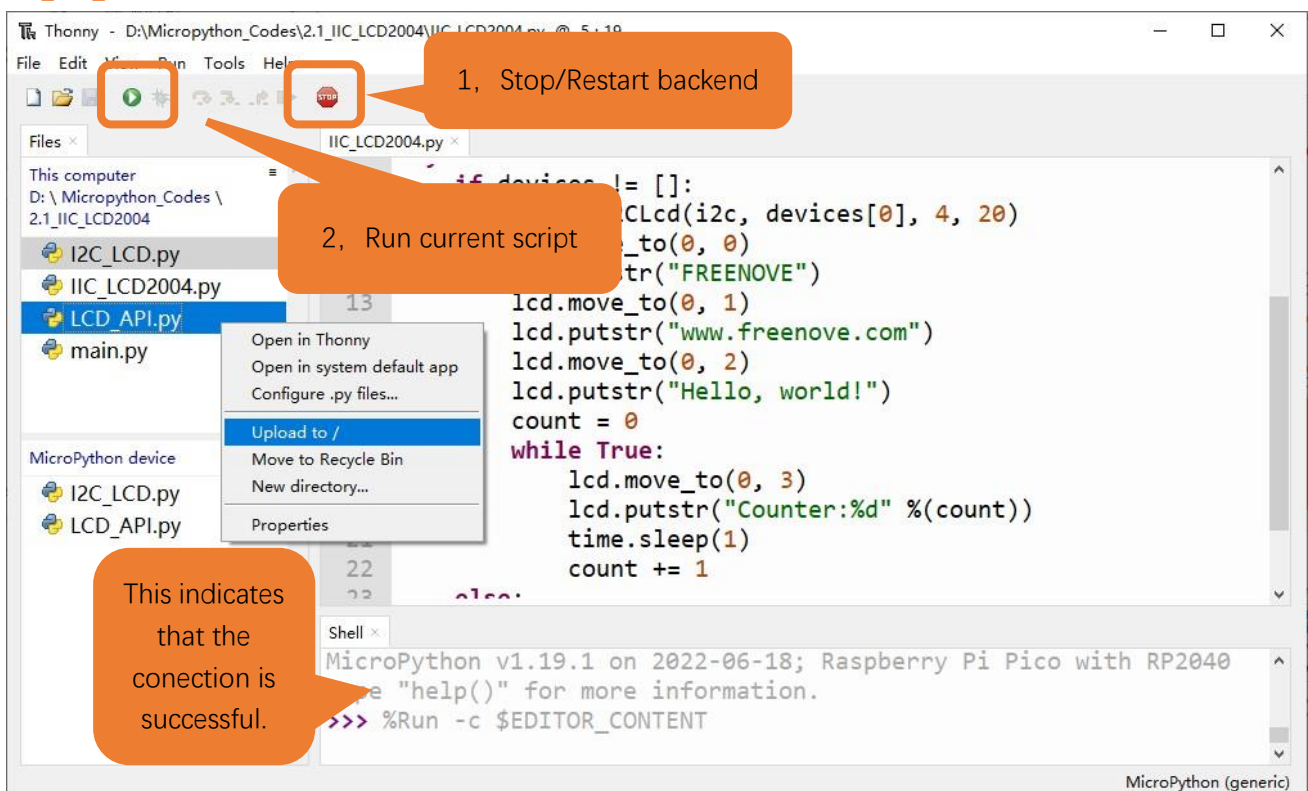
## Code

Codes used in this tutorial are saved in

**"Freenove\_LCD\_Module/Freenove\_LCD\_Module\_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"**.

Open "Thonny", click "This computer" → "D:" → "Micropython\_Codes" → "2.1\_I2C\_LCD2004". Select **"I2C\_LCD.py"** and **"LCD\_API.py"**, right click your mouse to select **"Upload to /"**, wait for **"I2C\_LCD.py"** and **"LCD\_API.py"** to be uploaded to Raspberry Pi Pico and then double click **"I2C\_LCD2004.py"**.

### 2.1\_I2C\_LCD2004



Click "Run current script" and LCD2004 displays some characters. Press Ctrl+C or click "Stop/Restart backend" to exit the program.



Note: If you cannot see anything on the display or the display is not clear, try rotating the white knob on back of LCD2004 slowly, which adjusts the contrast, until the screen can display clearly.



The following is the program code:

```

1  import time
2  from machine import I2C, Pin
3  from I2C_LCD import I2CLcd
4
5  i2c = I2C(0, sda=Pin(4), scl=Pin(5), freq=400000)
6  devices = i2c.scan()
7
8  try:
9      if devices != []:
10         lcd = I2CLcd(i2c, devices[0], 4, 20)
11         lcd.move_to(0, 0)
12         lcd.putstr("FREENOVE")
13         lcd.move_to(0, 1)
14         lcd.putstr("www.freenove.com")
15         lcd.move_to(0, 2)
16         lcd.putstr("Hello, world!")
17         count = 0
18         while True:
19             lcd.move_to(0, 3)
20             lcd.putstr("Counter:%d" %(count))
21             time.sleep(1)
22             count += 1
23         else:
24             print("No address found")
25 except:
26     pass

```

Import time, I2C and I2C\_LCD modules.

```

1  import time
2  from machine import I2C, Pin
3  from I2C_LCD import I2CLcd

```

Create an I2C object, initialize the I2C parameter configuration, and associate it with the LCD2004 pin. Call the scan() function to query the LCD2004 device address.

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



```

5  i2c = I2C(0, sda=Pin(4), scl=Pin(5), freq=400000)
6  devices = i2c.scan()

```

Use the if statement to determine whether the queried I2C device address is empty. If it is not empty, create an I2CLcd object and set the created I2C object, I2C device address, and the number of rows and columns of LCD2004; if it is empty, print out "No address" found" and the program exits.

```

9      if devices != []:
10         lcd = I2CLcd(i2c, devices[0], 4, 20)
11         ...
23     else:
24         print("No address found")

```

Move the cursor of LCD2004 to the third row, first column, and print out "Hello, world!"

```

15     lcd.move_to(0, 2)
16     lcd.putstr("Hello, world!")

```

The fourth line of LCD2004 continuously prints the number of seconds after the Raspberry Pi Pico program runs.

```

18     while True:
19         lcd.move_to(0, 3)
20         lcd.putstr("Counter:%d" %(count))
21         time.sleep(1)
22         count += 1

```

## Reference

### Class I2CLcd

Before each use of the object **I2CLcd**, please make sure that **I2C\_LCD.py** and **LCD\_API.py** have been uploaded to "/" of Raspberry Pi Pico, and then add the statement "**from I2C\_LCD import I2CLcd**" to the top of the python file.

**clear()**: Clear the LCD2004 screen display.

**show\_cursor()**: Show the cursor of LCD2004.

**hide\_cursor()**: Hide the cursor of LCD2004.

**blink\_cursor\_on()**: Turn on cursor blinking.

**blink\_cursor\_off()**: Turn off cursor blinking.

**display\_on()**: Turn on the display function of LCD2004.

**display\_off()**: Turn on the display function of LCD2004.

**backlight\_on()**: Turn on the backlight of LCD2004.

**backlight\_off()**: Turn on the backlight of LCD2004.

**move\_to(cursor\_x, cursor\_y)**: Move the cursor to a specified position.

**cursor\_x**: Column cursor\_x.

**cursor\_y**: Row cursor\_y.

**putchar(char)**: Print the character in the bracket on LCD2004.

**putstr(string)**: Print the string in the bracket on LCD2004.

## 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](mailto: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.

<http://www.freenove.com/>

Thank you again for choosing Freenove products.