Important Information

Thank you for choosing Freenove products!

Getting Started

First, please read the **Start Here.pdf** document in the unzipped folder you created.
If you have not yet downloaded the zip file, associated with this kit, please do so now and unzip it.

Get Support and Offer Input

Freenove provides free and responsive product and technical support, including but not limited to:

- Product quality issues
- Product use and build issues
- Questions regarding the technology employed in our products for learning and education
- Your input and opinions are always welcome
- We also encourage your ideas and suggestions for new products and product improvements

For any of the above, you may send us an email to:

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.

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

sale@freenove.com

Copyright

FREENOVE™

Contents

Preface

ESP32-S3 is a micro control unit with integrated Wi-Fi launched by Espressif, which features strong properties and integrates rich peripherals. It can be designed and studied as an ordinary Single Chip Micyoco(SCM) chip, or connected to the Internet and used as an Internet of Things device.

ESP32-S3 can be developed using the Arduino platform, which will definitely make it easier for people who have learned arduino to master. Moreover, the code of ESP32-S3 is completely open-source, so beginners can quickly learn how to develop and design IOT smart household products including smart curtains, fans, lamps and clocks.

Generally, ESP32-S3 projects consist of code and circuits. Don't worry even if you've never learned code and circuits, because we will gradually introduce the basic knowledge of C programming language and electronic circuits, from easy to difficult. Our products contain all the electronic components and modules needed to complete these projects. It's especially suitable for beginners.

We divide each project into four parts, namely Component List, Component Knowledge, Circuit and Code. Component List helps you to prepare material for the experiment more quickly. Component Knowledge allows you to quickly understand new electronic modules or components, while Circuit helps you understand the operating principle of the circuit. And Code allows you to easily master the use of SEP32 and accessory kit. After finishing all the projects in this tutorial, you can also use these components and modules to make products such as smart household, smart cars and robots to transform your creative ideas into prototypes and new and innovative products.

In addition, if you have any difficulties or questions with this tutorial or toolkit, feel free to ask for our quick and free technical support through **support@freenove.com**

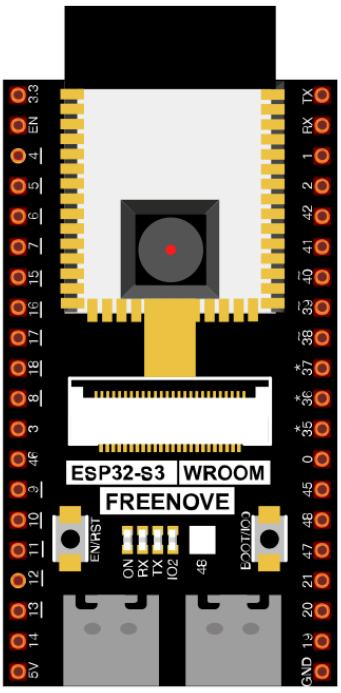
ESP32-S3 WROOM

ESP32-S3-WROOM-1 has launched a total of two antenna packages, PCB on-board antenna and IPEX antenna respectively. The PCB on-board antenna is an integrated antenna in the chip module itself, so it is convenient to carry and design. The IPEX antenna is a metal antenna derived from the integrated antenna of the chip module itself, which is used to enhance the signal of the module.

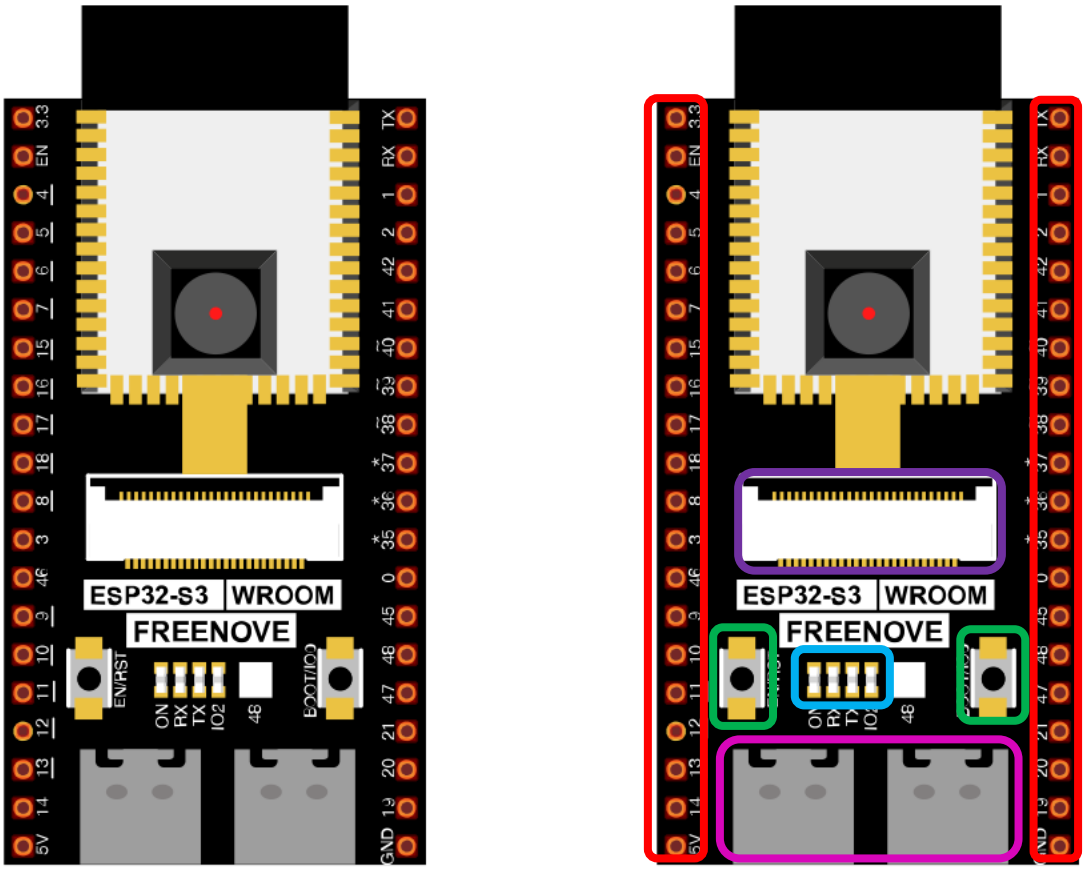
PCB on-board antenna	IPEX antenna
	

In this tutorial, the ESP32-S3 WROOM is designed based on the PCB on-board antenna-packaged ESP32-S3-WROOM-1 module.

ESP32-S3 WROOM



The hardware interfaces of ESP32-S3 WROOM are distributed as follows:



Compare the left and right images. We've boxed off the resources on the ESP32-S3 WROOM in different colors to facilitate your understanding of the ESP32-S3 WROOM.

Box color	Corresponding resources introduction
	GPIO pin
	LED indicator
	Camera interface
	Reset button, Boot mode selection button
	USB port

For more information, please visit: https://www.espressif.com.cn/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf.

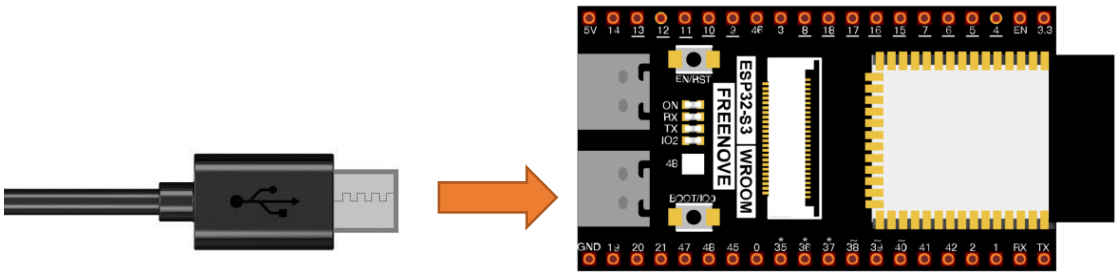
CH343 (Importance)

ESP32-S3 WROOM uses CH343 to download codes. So before using it, we need to install CH343 driver in our computers.

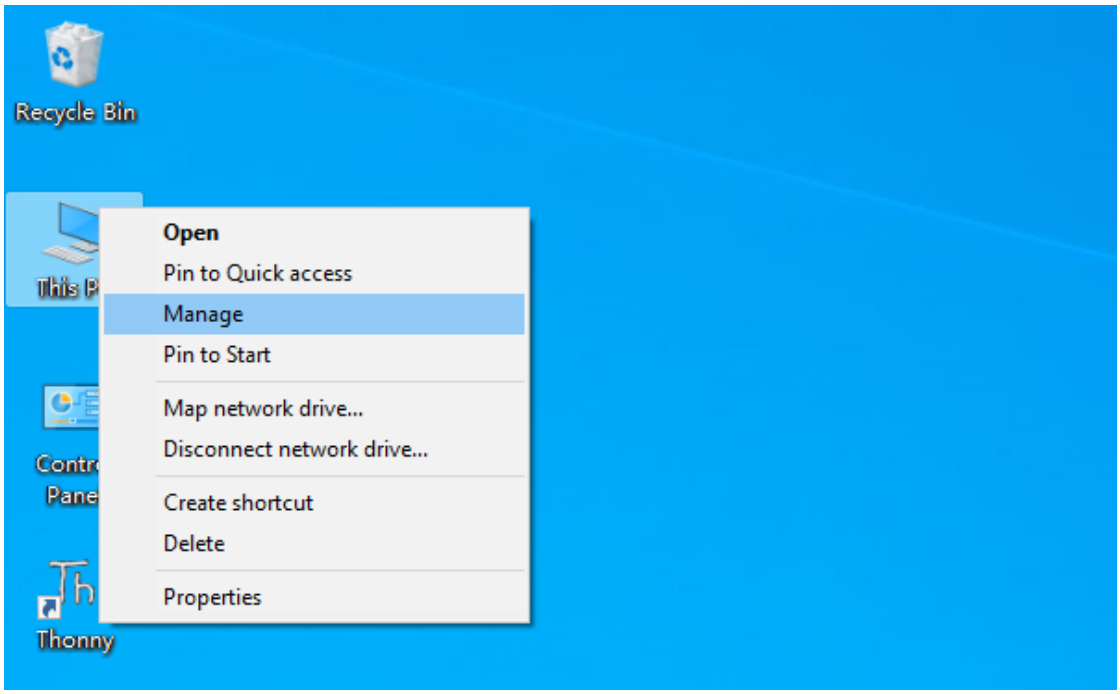
Windows

Check whether CH343 has been installed

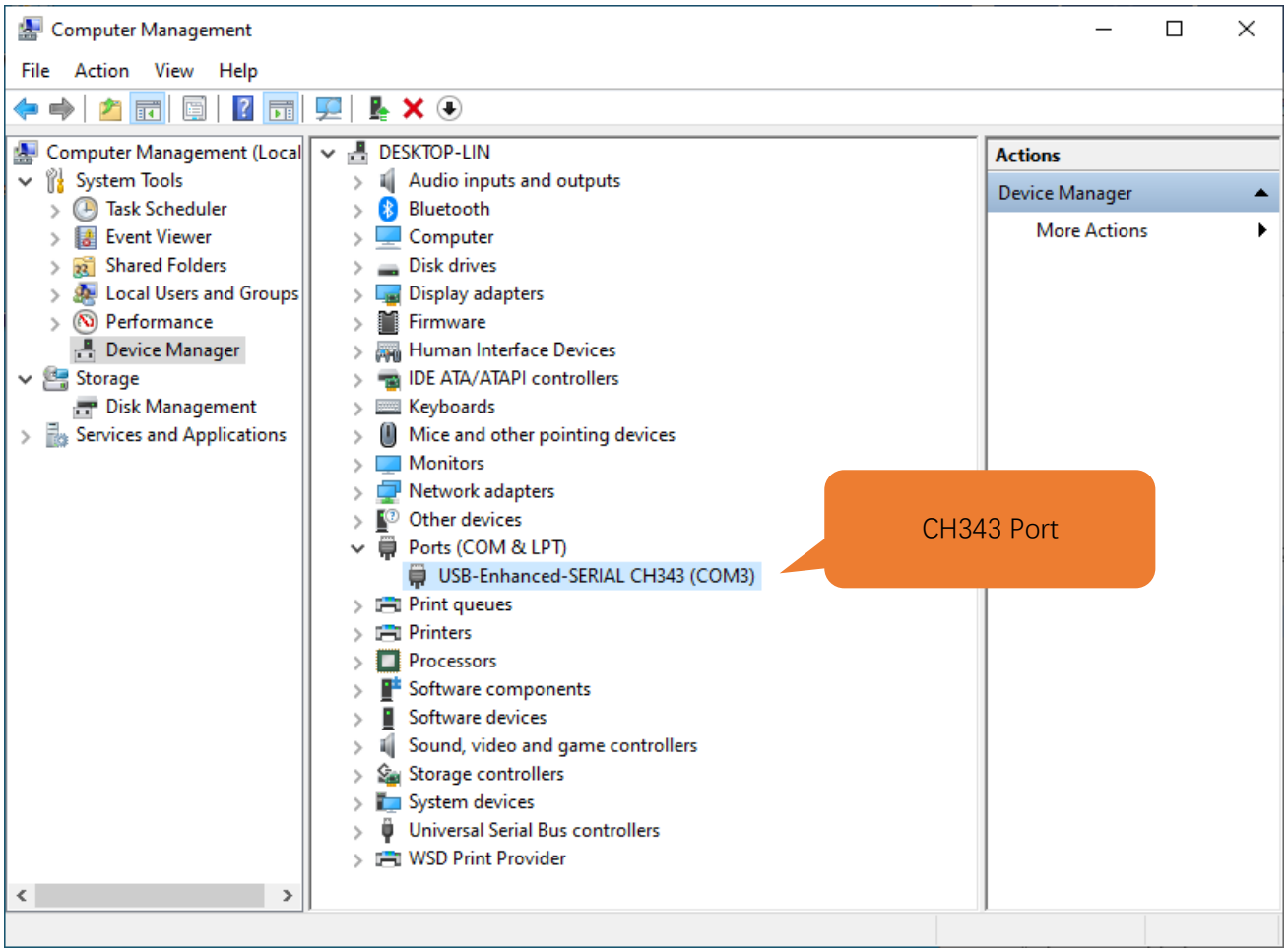
1. Connect your computer and ESP32-S3 WROOM with a USB cable.



2. Turn to the main interface of your computer, select “This PC” and right-click to select “Manage”.

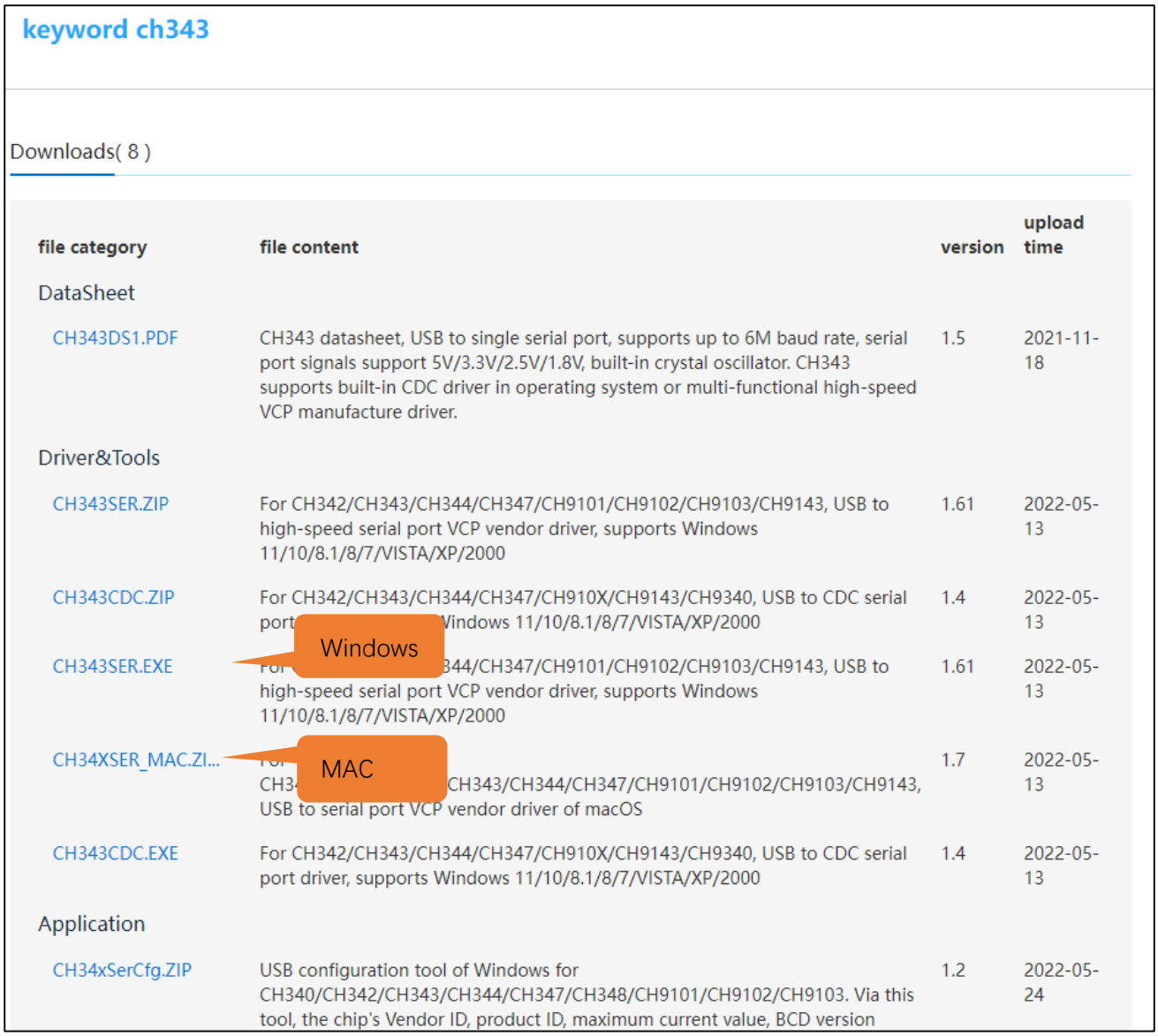


3. Click "Device Manager". If your computer has installed CH343, you can see"USB-Enhances-SERIAL CH343 (COMx)". And you can click here to move to the next step.

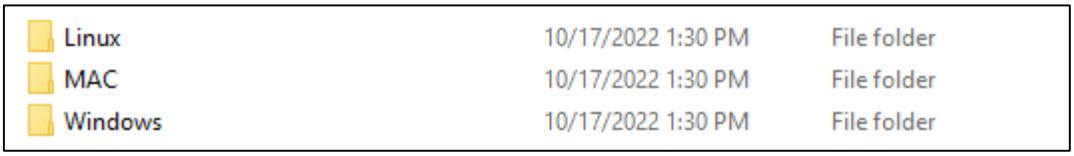


Installing CH343

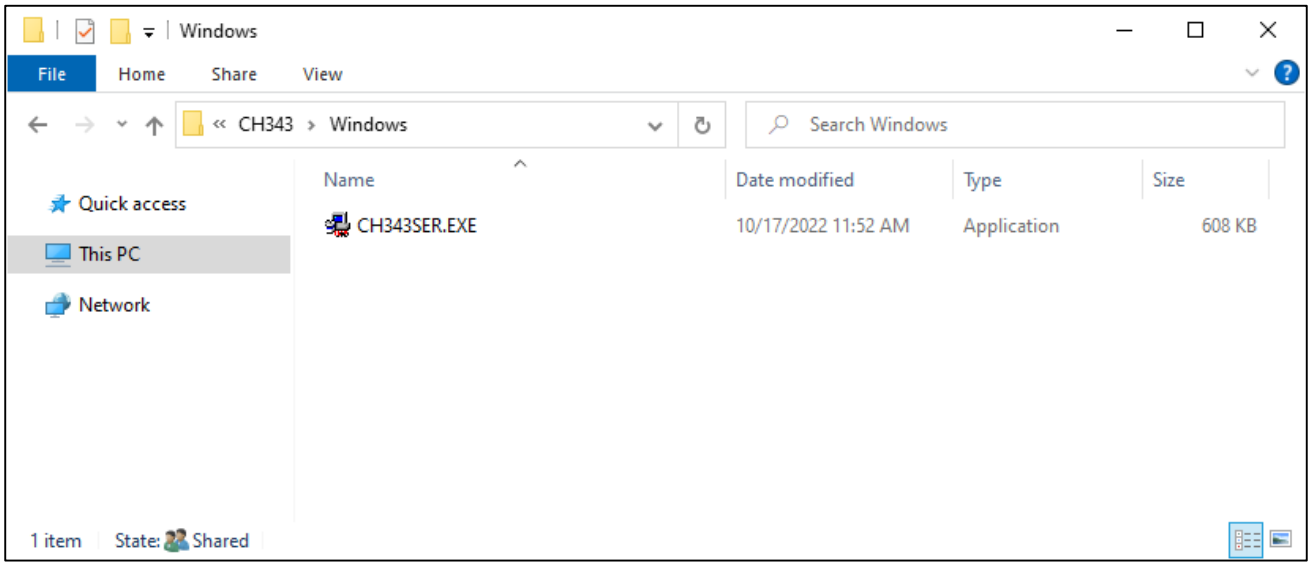
1. First, download CH343 driver, click http://www.wch-ic.com/search?t=all&q=ch343 to download the appropriate one based on your operating system.



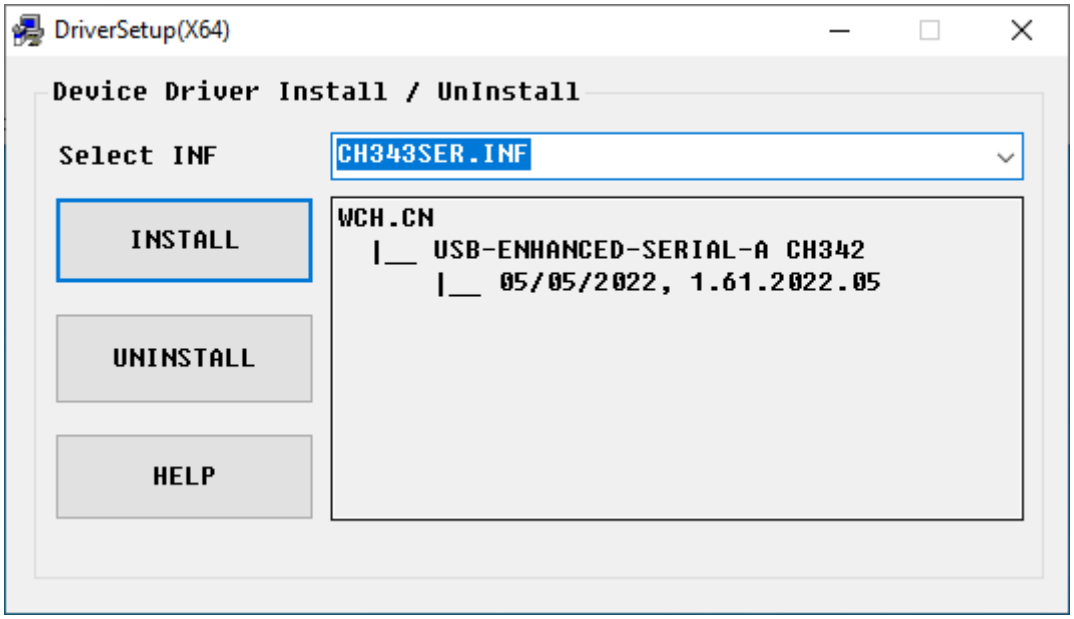
If you would not like to download the installation package, you can open **"Freenove_ESP32_S3_WROVER_Board/CH343"**, we have prepared the installation package.



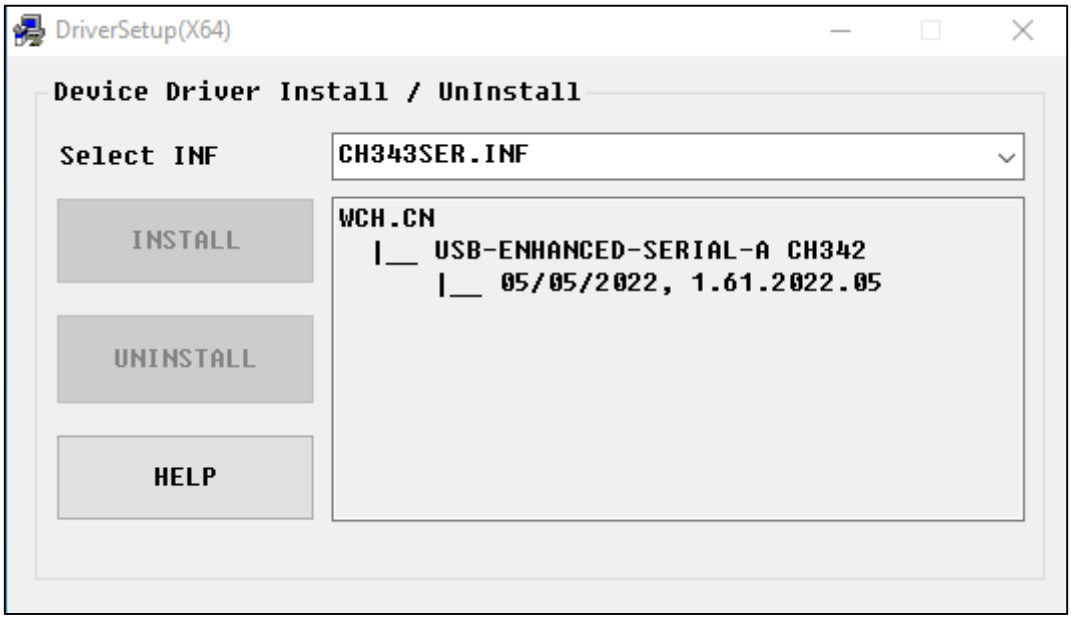
2. Open the folder "**Freenove_ESP32_S3_WROVER_Board/CH343/Windows/**"



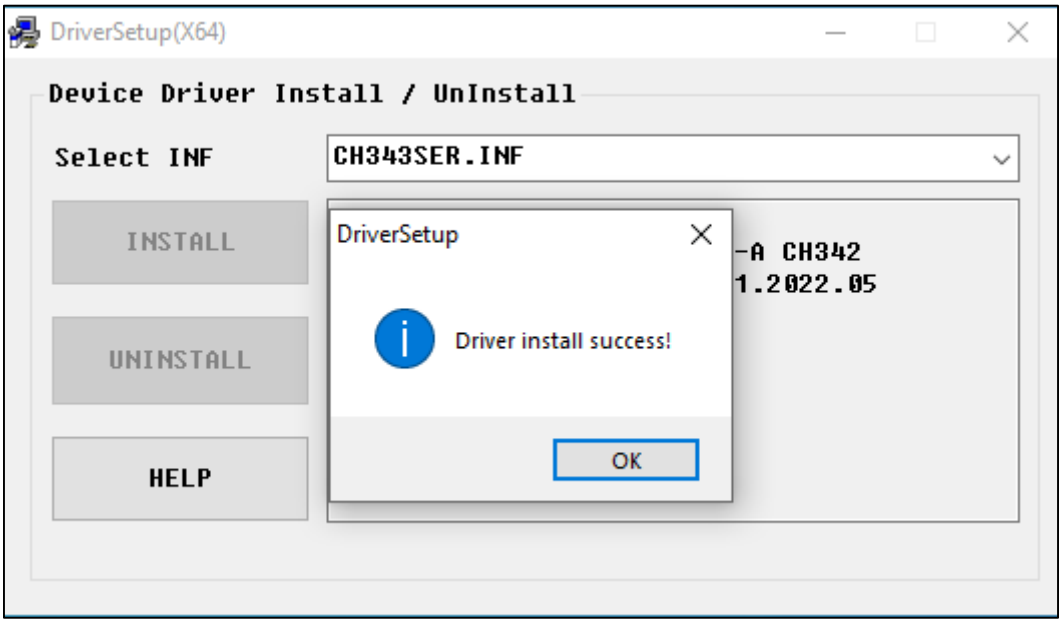
3. Double click "**CH343SER.EXE**".



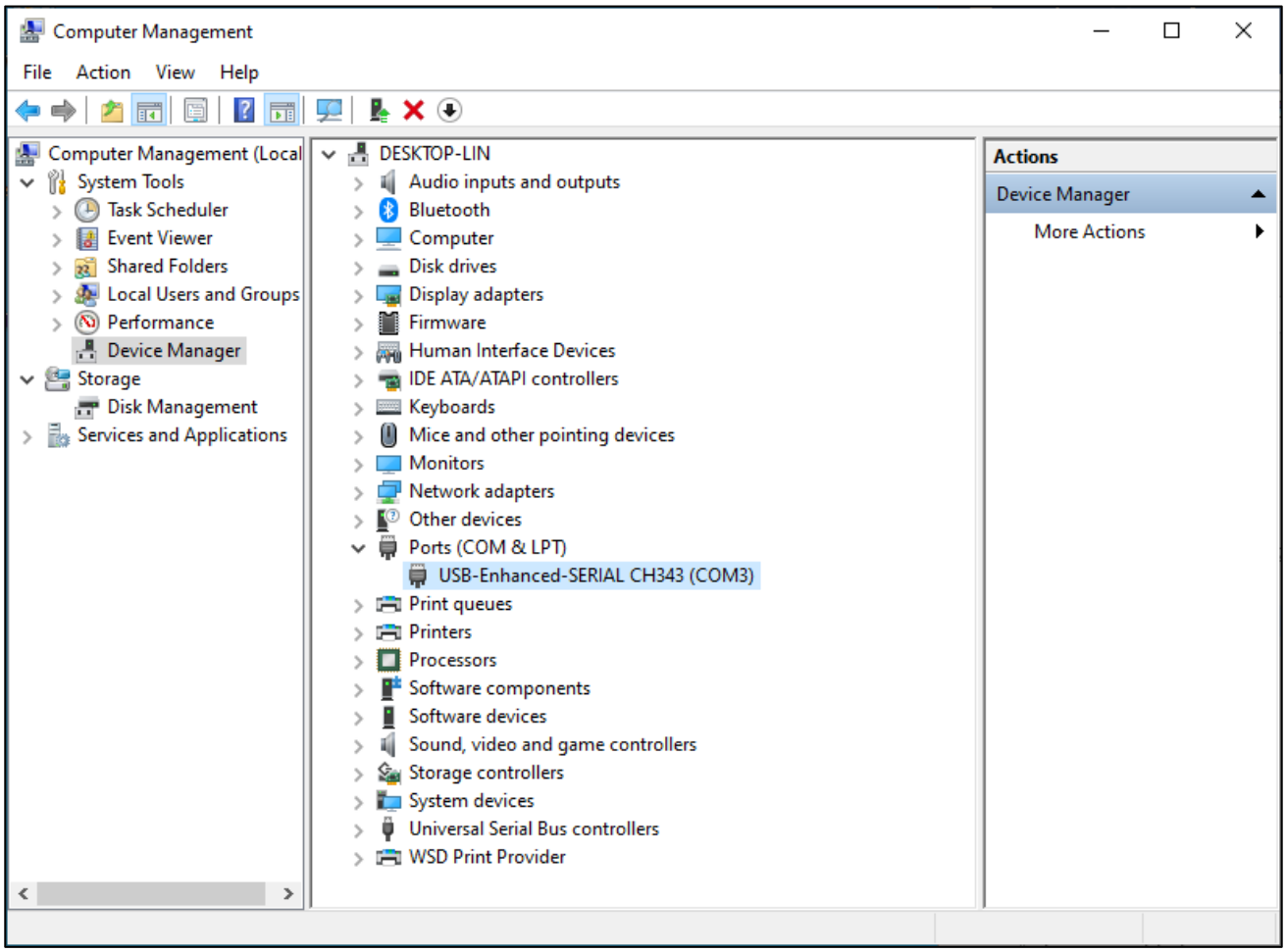
4. Click “INSTALL” and wait for the installation to complete.



5. Install successfully. Close all interfaces.



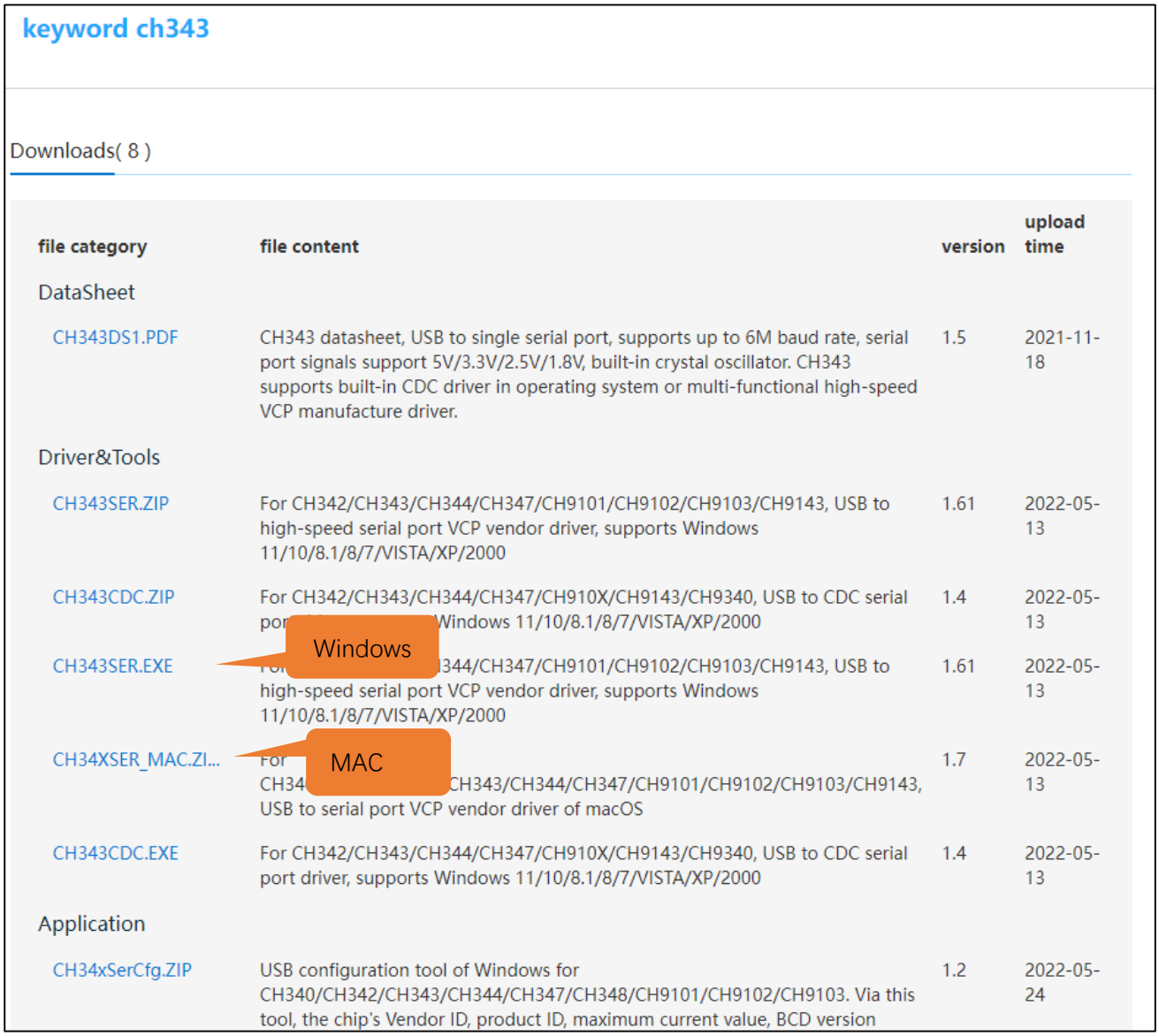
6. When ESP32-S3 WROOM is connected to computer, select “This PC”, right-click to select “Manage” and click “Device Manager” in the newly pop-up dialog box, and you can see the following interface.



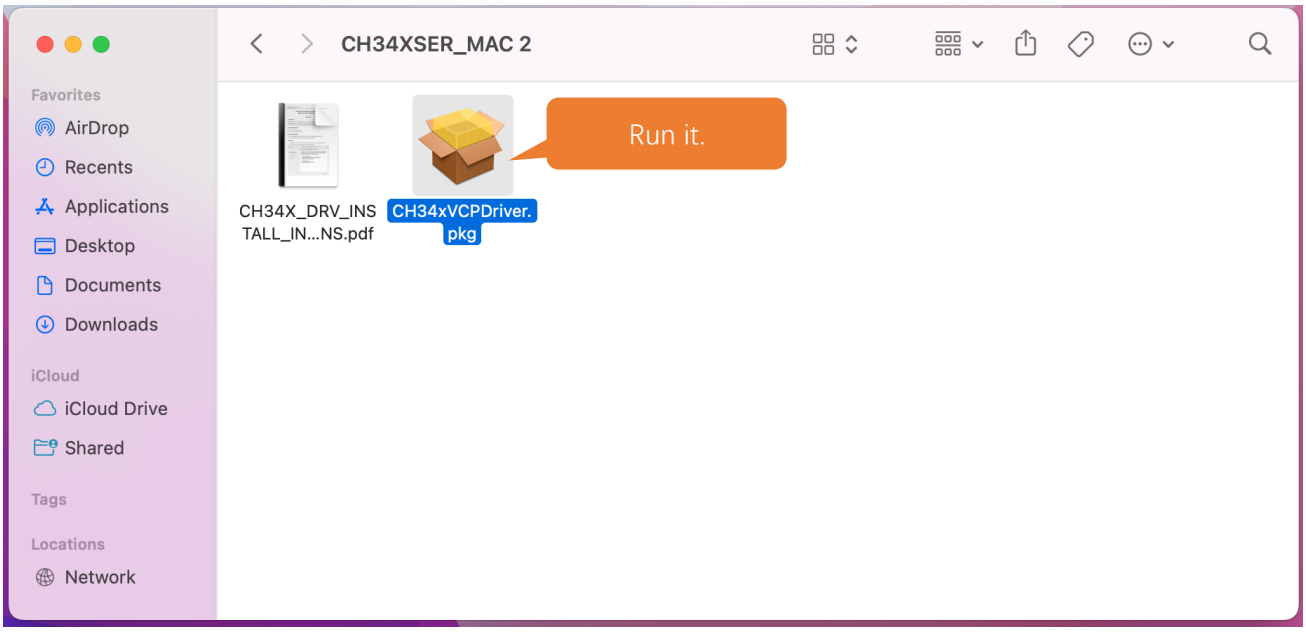
7. So far, CH343 has been installed successfully. Close all dialog boxes.

MAC

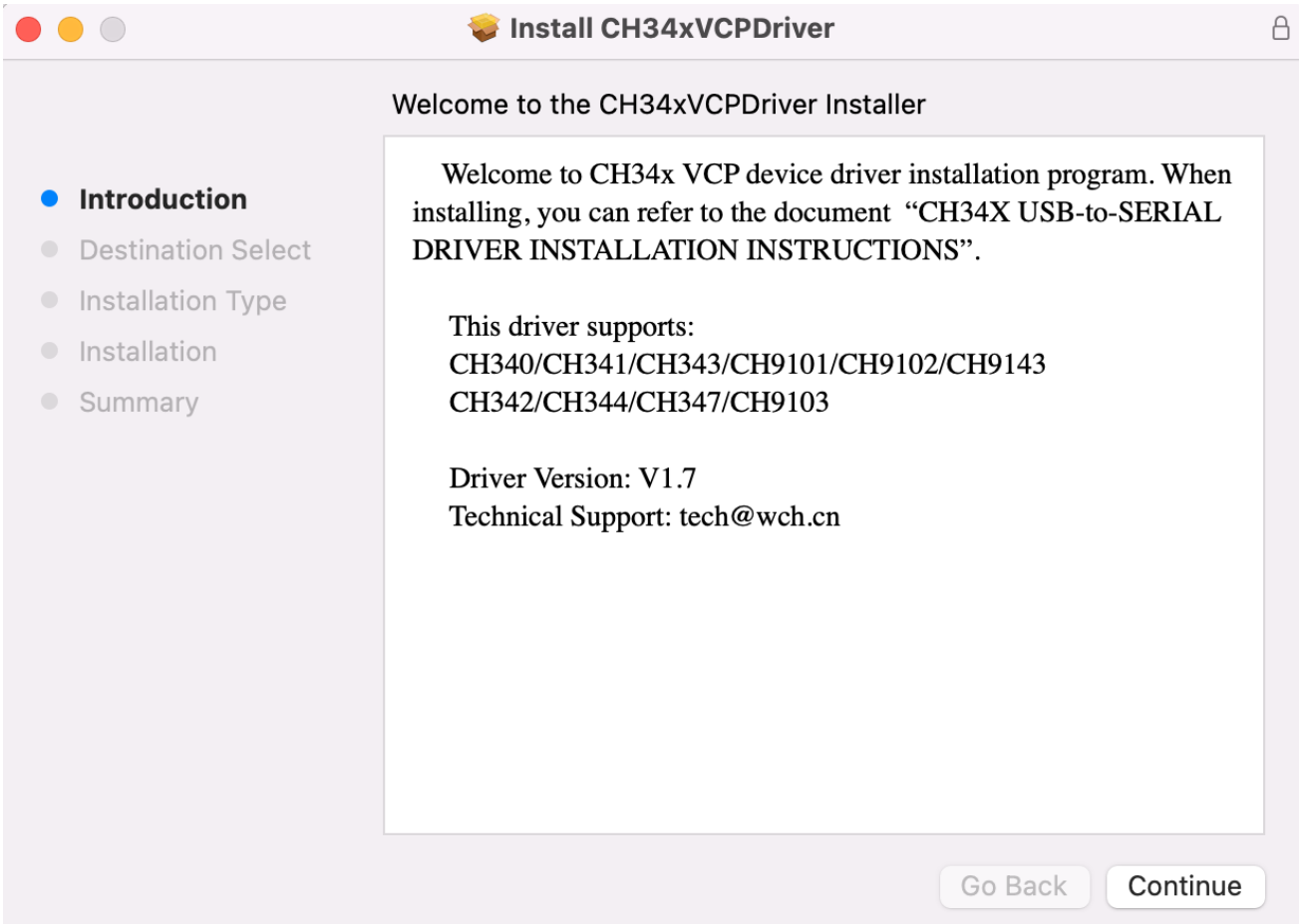
First, download CH343 driver, click http://www.wch-ic.com/search?t=all&q=ch343 to download the appropriate one based on your operating system.



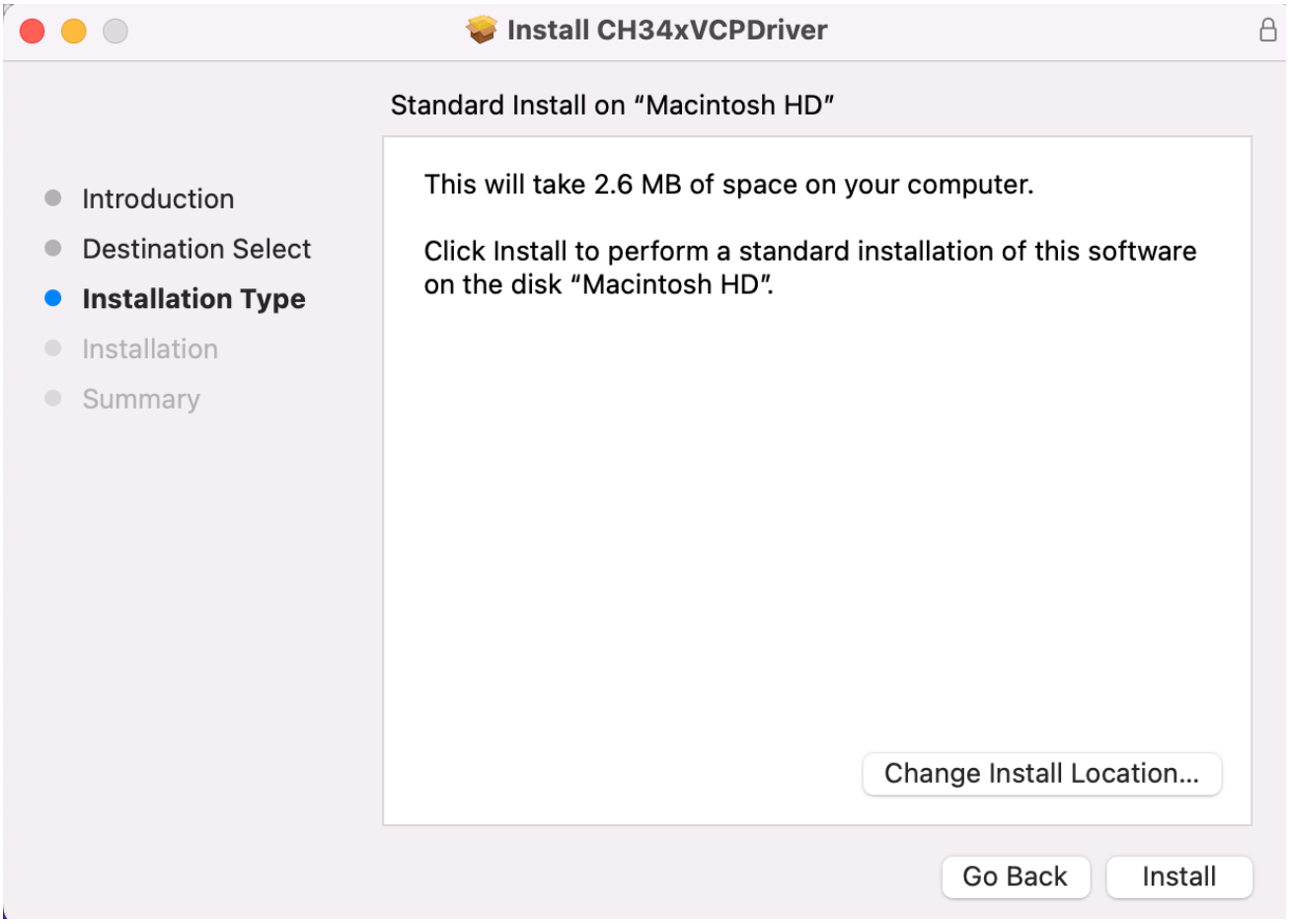
If you would not like to download the installation package, you can open **"Freenove_ESP32_S3_WROVER_Board/CH343"**, we have prepared the installation package.
Second, open the folder "**Freenove_ESP32_S3_WROVER_Board/CH343/MAC/**"



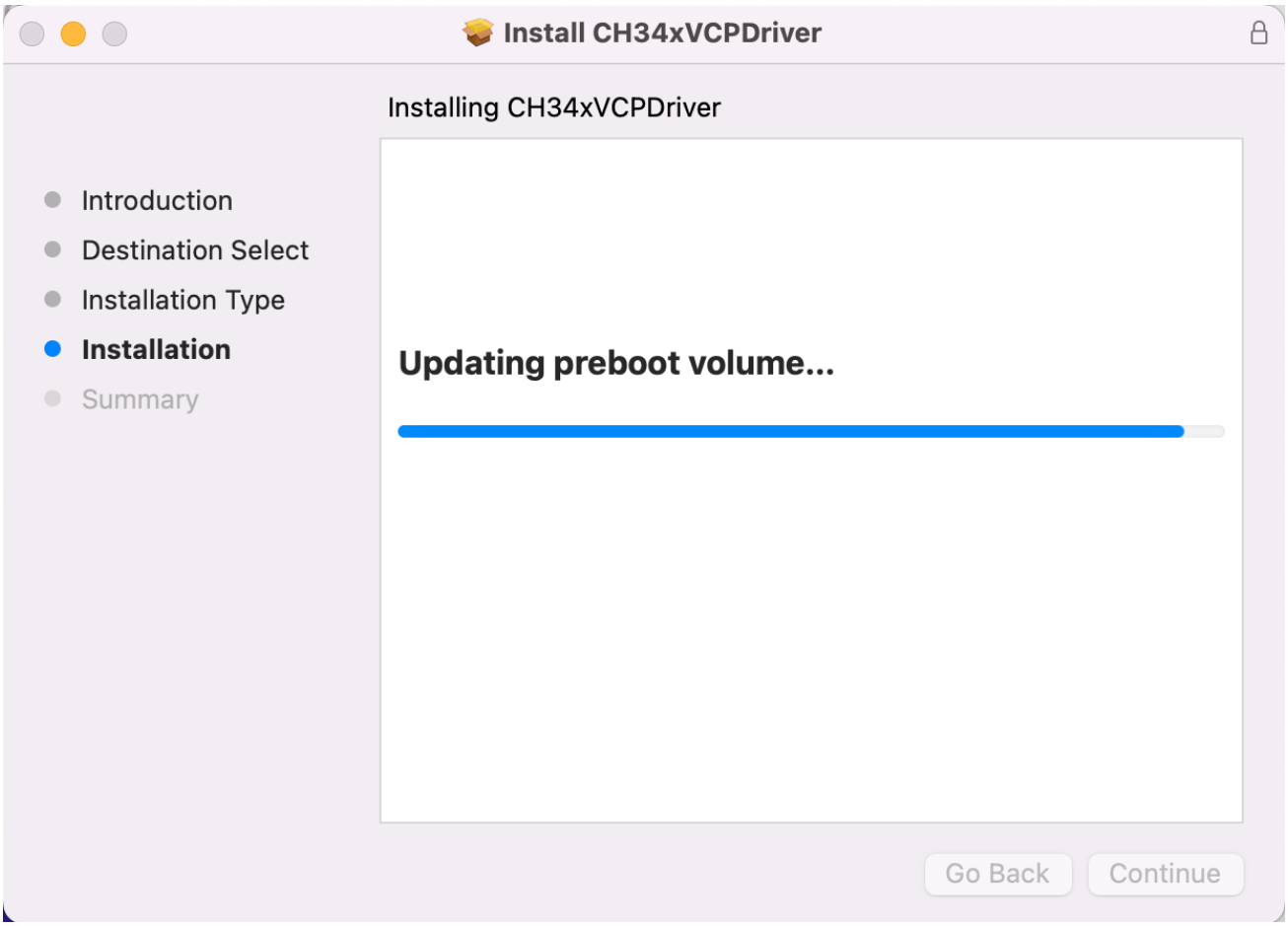
Third, click Continue.



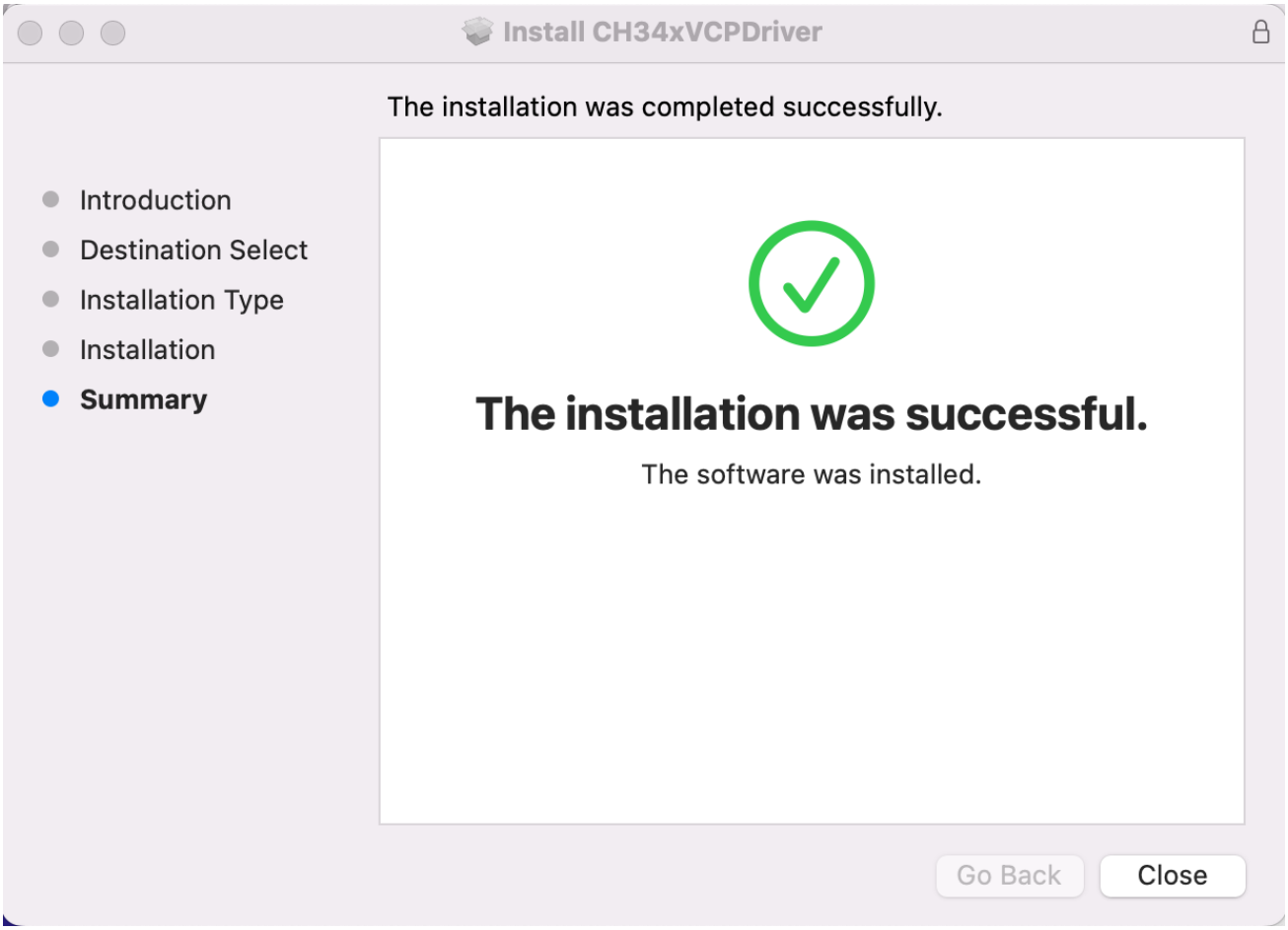
Fourth, click Install.


Install CH34xVCPDriver
Standard Install on "Macintosh HD"
Introduction
Destination Select
Installation Type
Installation
Summary
This will take 2.6 MB of space on your computer.
Click Install to perform a standard installation of this software on the disk "Macintosh HD".
Change Install Location...
Go Back
Install

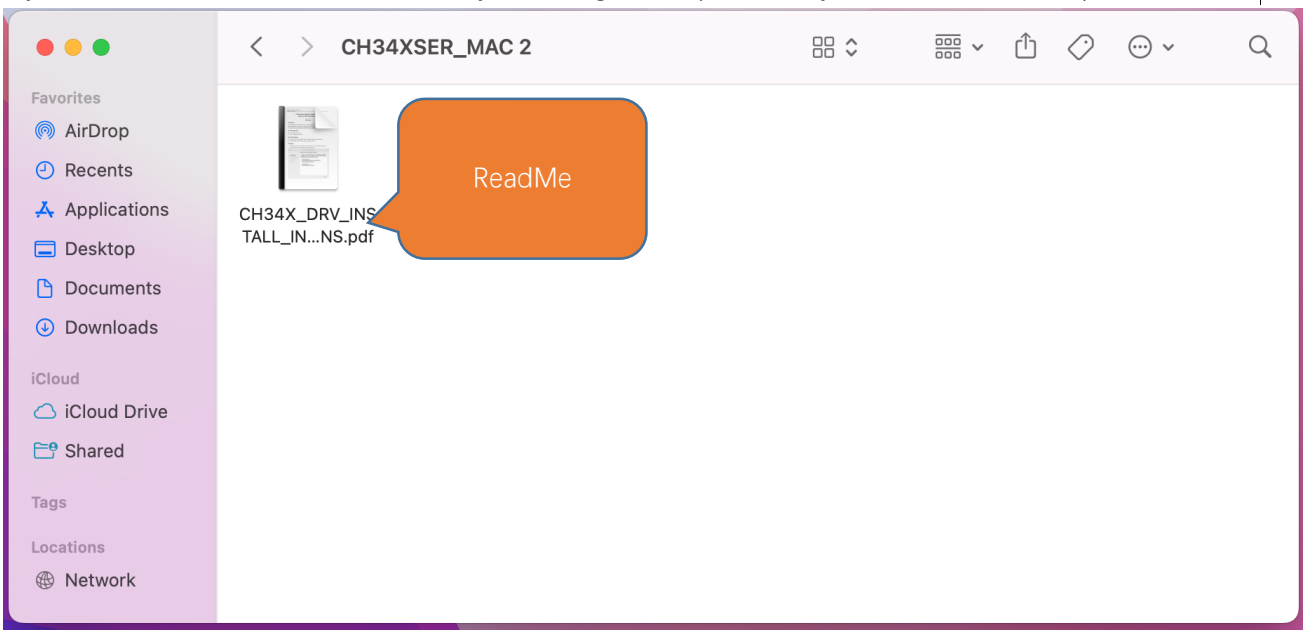
Then, waiting Finsh.



Finally, restart your PC.


Install CH34xVCPDriver
The installation was completed successfully.
Introduction
Destination Select
Installation Type
Installation
Summary
The installation was successful.
The software was installed.
Go Back
Close

If you still haven't installed the CH340 by following the steps above, you can view readme.pdf to install it.



Programming Software

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 ESP32-S3 during the whole process.
Thonny supports various operating system, including Windows、Mac OS、Linux.

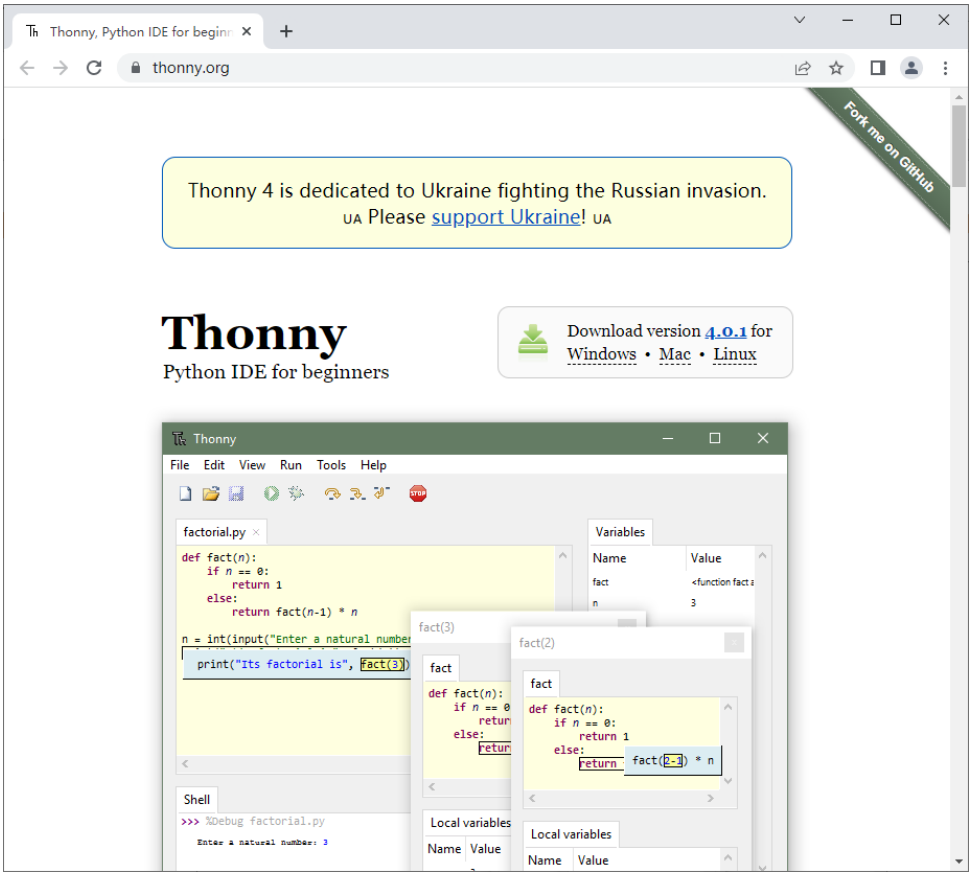
Downloading Thonny

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.)

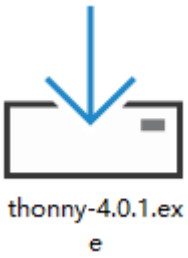
Operating System	Download links/methods
Windows	https://github.com/thonny/thonny/releases/download/v4.0.1/thonny-4.0.1.exe
Mac OS	https://github.com/thonny/thonny/releases/download/v4.0.1/thonny-4.0.1.pkg
Linux	**Official downloads for Linux** **Installer** (installs private Python 3.10 on x86_64, uses existing python3 elsewhere) bash <(wget -O - https://thonny.org/installer-for-linux) **Re-using an existing Python installation** (for advanced users) pip3 install thonny **3rd party distributions** (may have older version) **Flatpak** flatpak install org.thonny.Thonny **Debian, Raspbian, Ubuntu, Mint and others** sudo apt install thonny **Fedora** sudo dnf install thonny

You can also open "**Freenove_ESP32_S3_WROVER_Board/Python/Python_Software**", we have prepared it in advance.

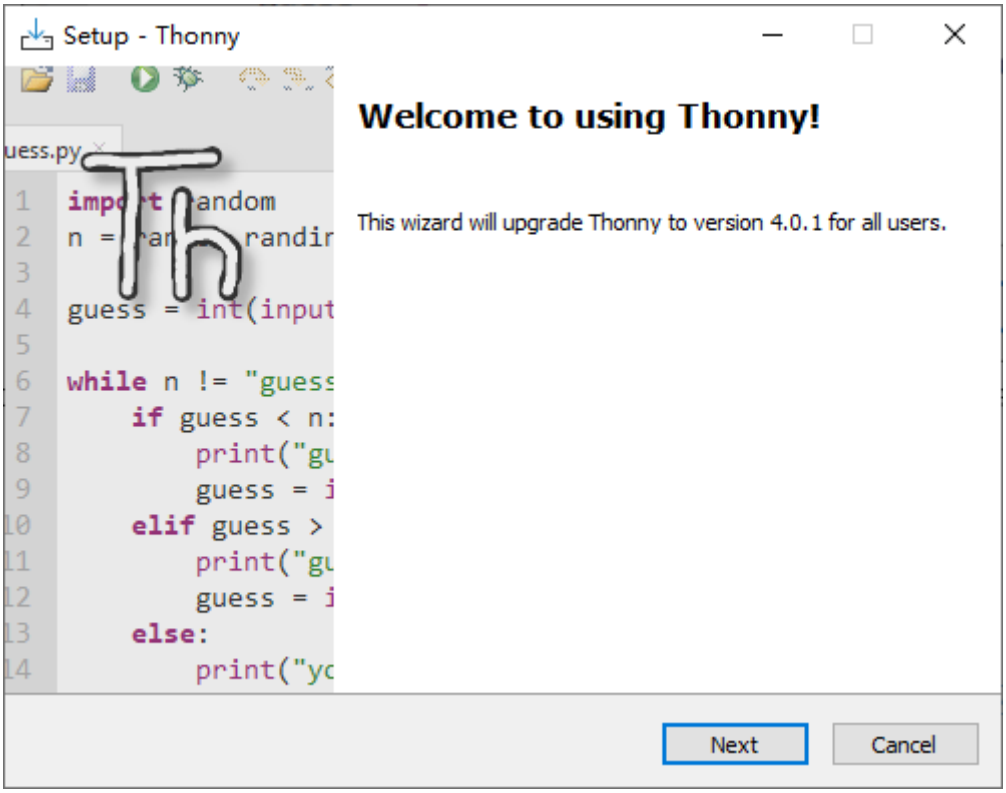


Installing on Windows

The icon of Thonny after downloading is as below. Double click “thonny-4.0.1.exe”.

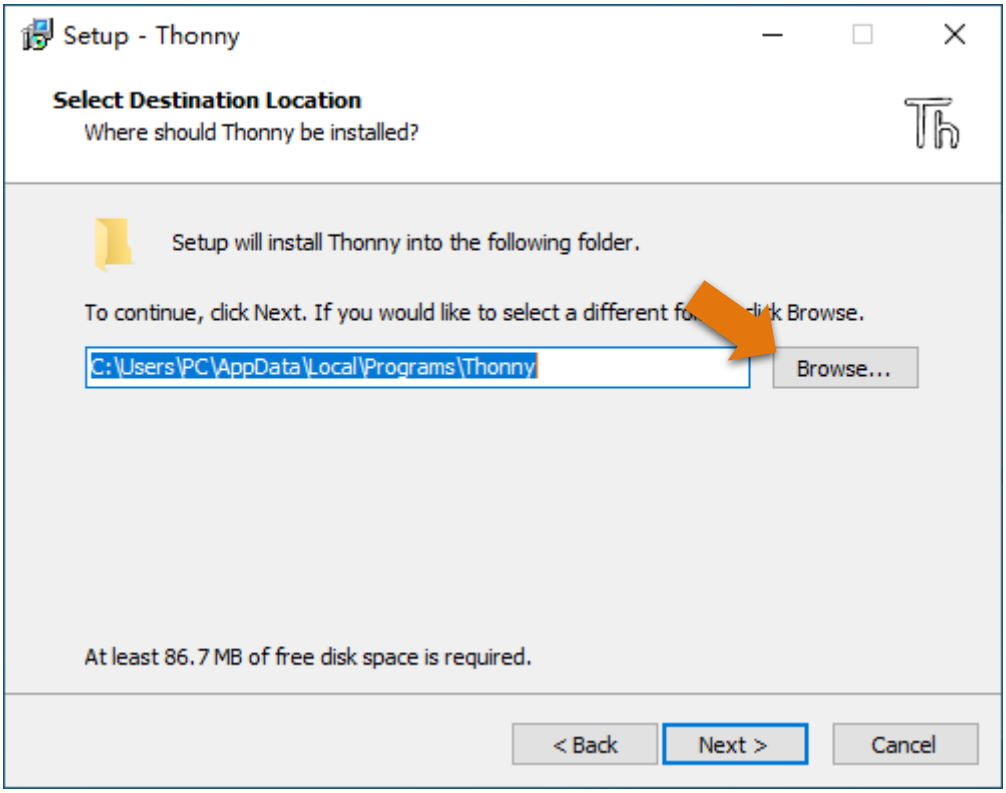


If you're not familiar with computer software installation, you can simply keep clicking "Next" until the installation completes.

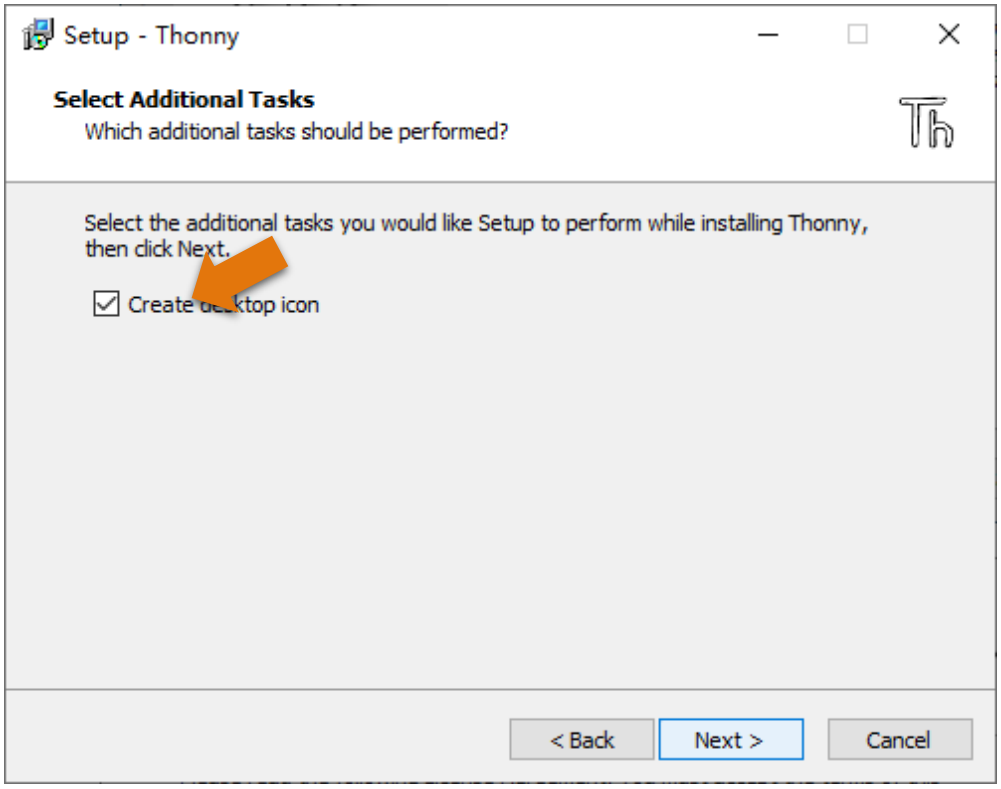


If you want to change Thonny's installation path, you can click "Browse" to modify it. After selecting installation path, click "OK".

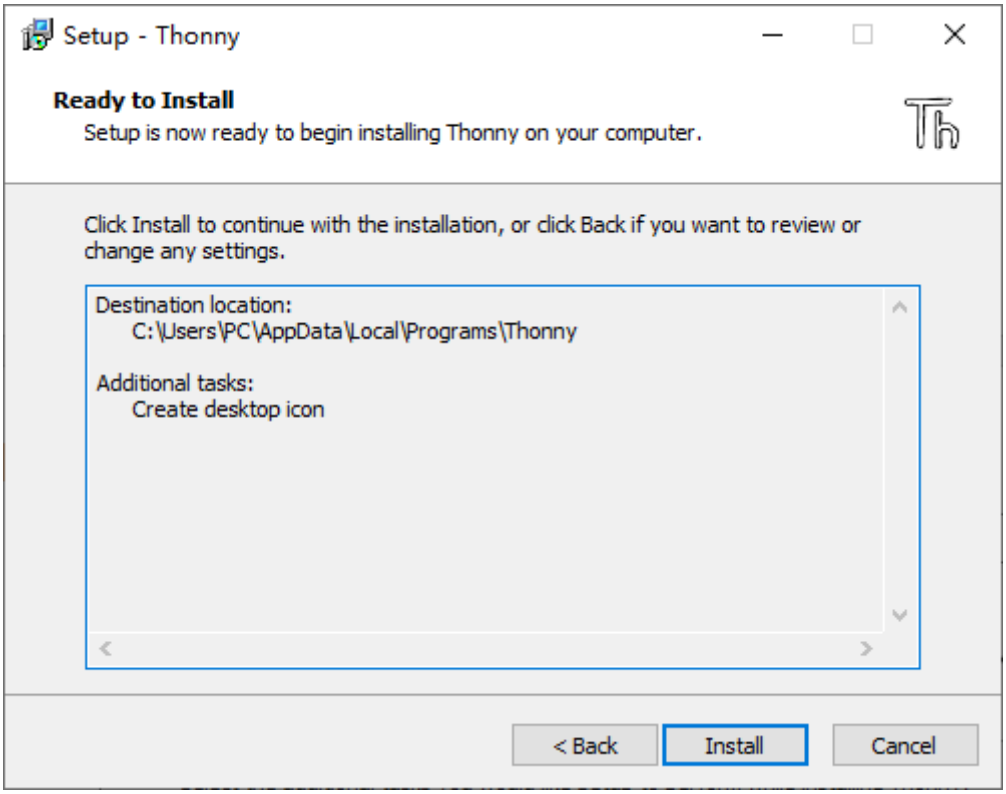
If you do not want to change it, just click "Next".



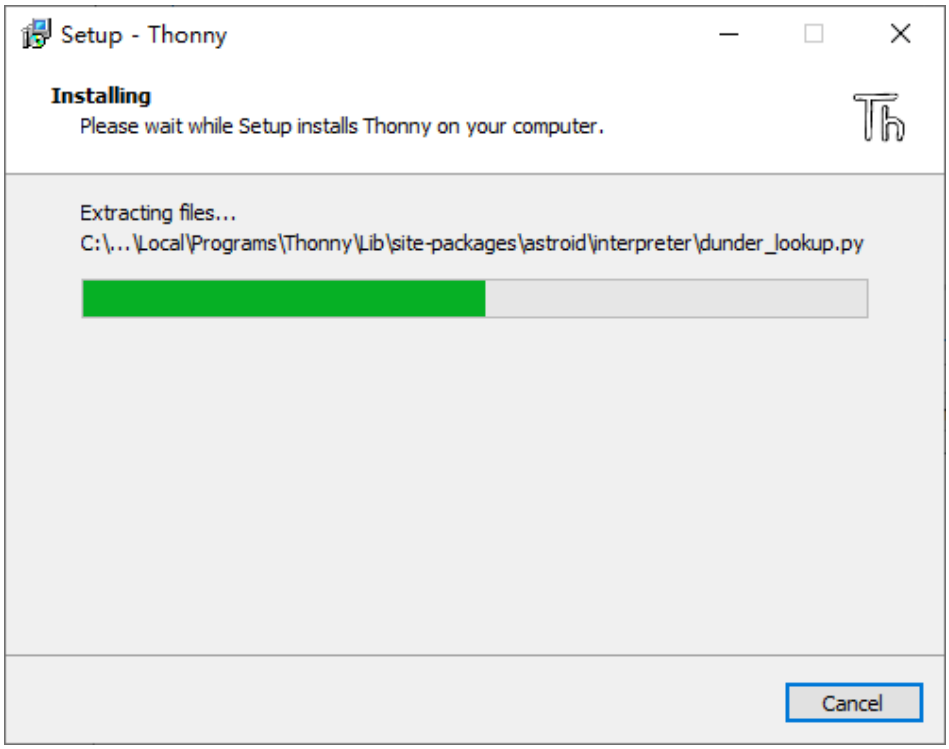
Check "Create desktop icon" and then it will generate a shortcut on your desktop to facilitate you to open Thonny later.



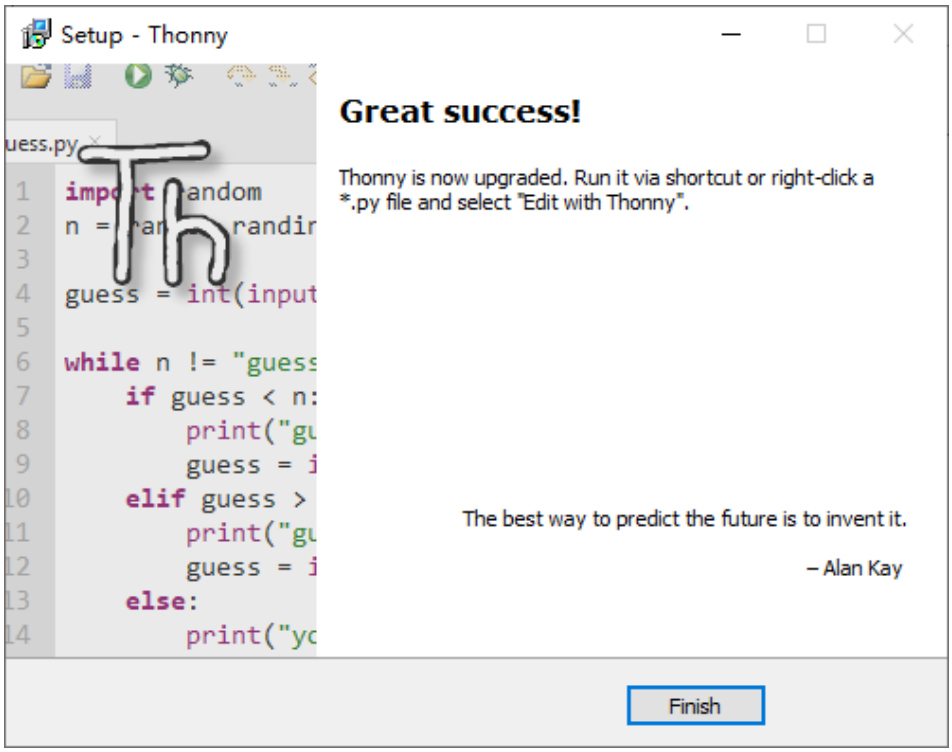
Click "install" to install the software.



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



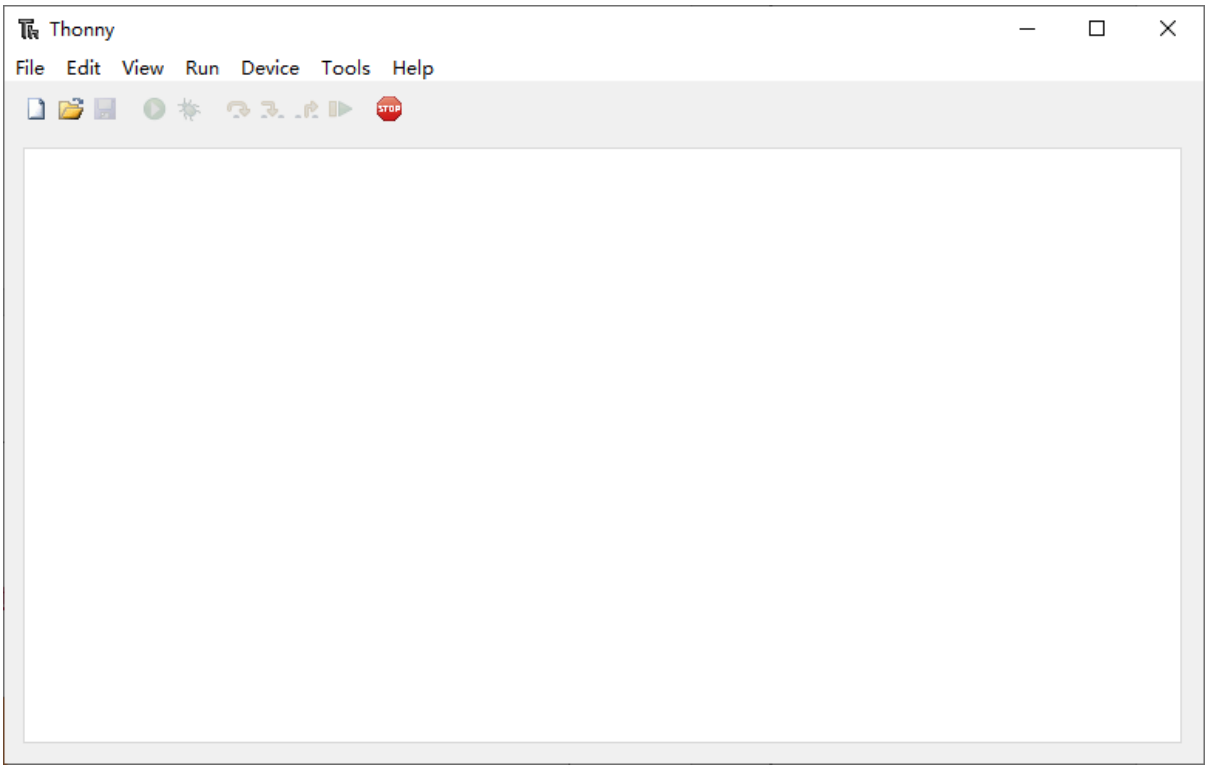
Once you see the interface as below, Thonny has been installed successfully.



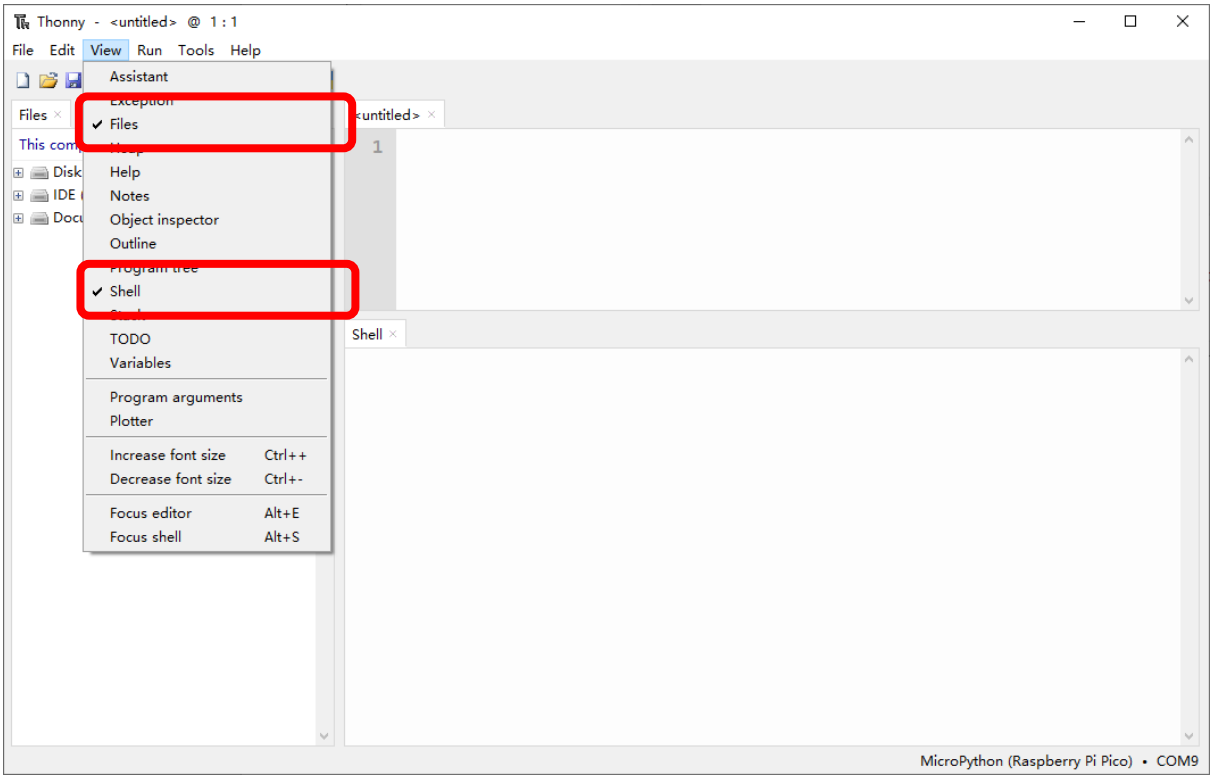
If you've check "Create desktop icon" during the installation process, you can see the below icon on your desktop.。

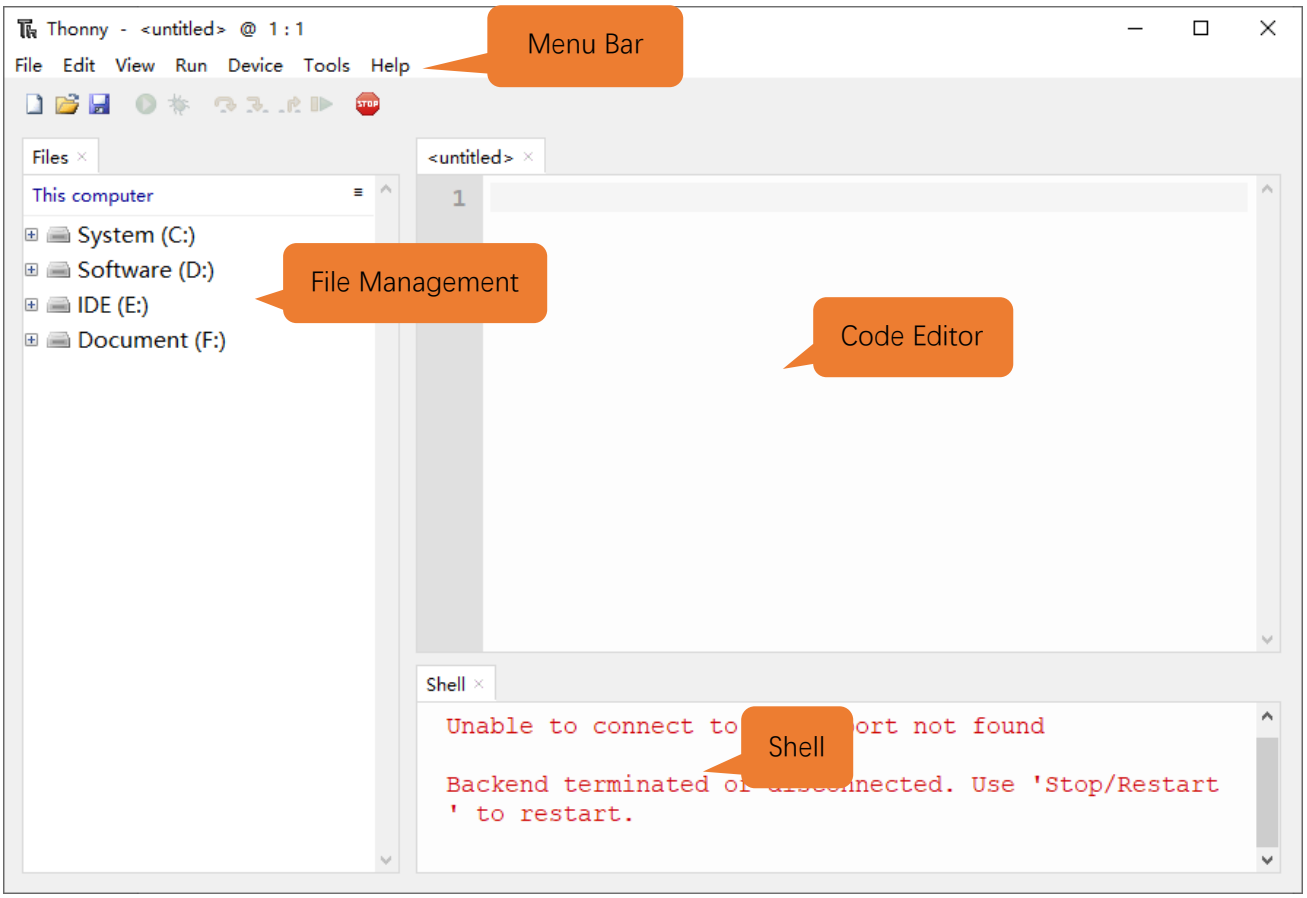
Basic Configuration of Thonny

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



Select "View"→ "Files" and "Shell".



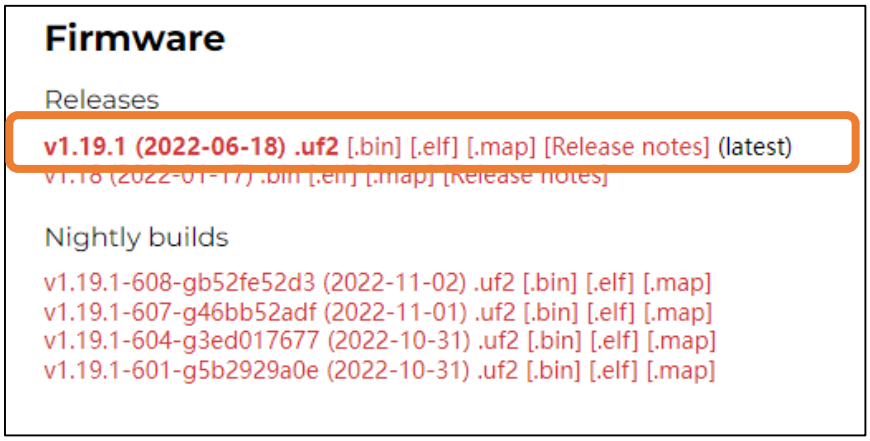

Thonny - <untitled> @ 1 : 1
File Edit View Run Device Tools Help
Menu Bar
Files
This computer
System (C:)
Software (D:)
IDE (E:)
Document (F:)
File Management
<untitled>
1
Code Editor
Shell
Unable to connect to
ort not found
Backend terminated or disconnected. Use 'Stop/Restart
' to restart.
Shell

Burning Micropython Firmware (Important)

To run Python programs on ESP32S3, we need to burn a firmware to ESP32-S3 first.

Downloading Micropython Firmware

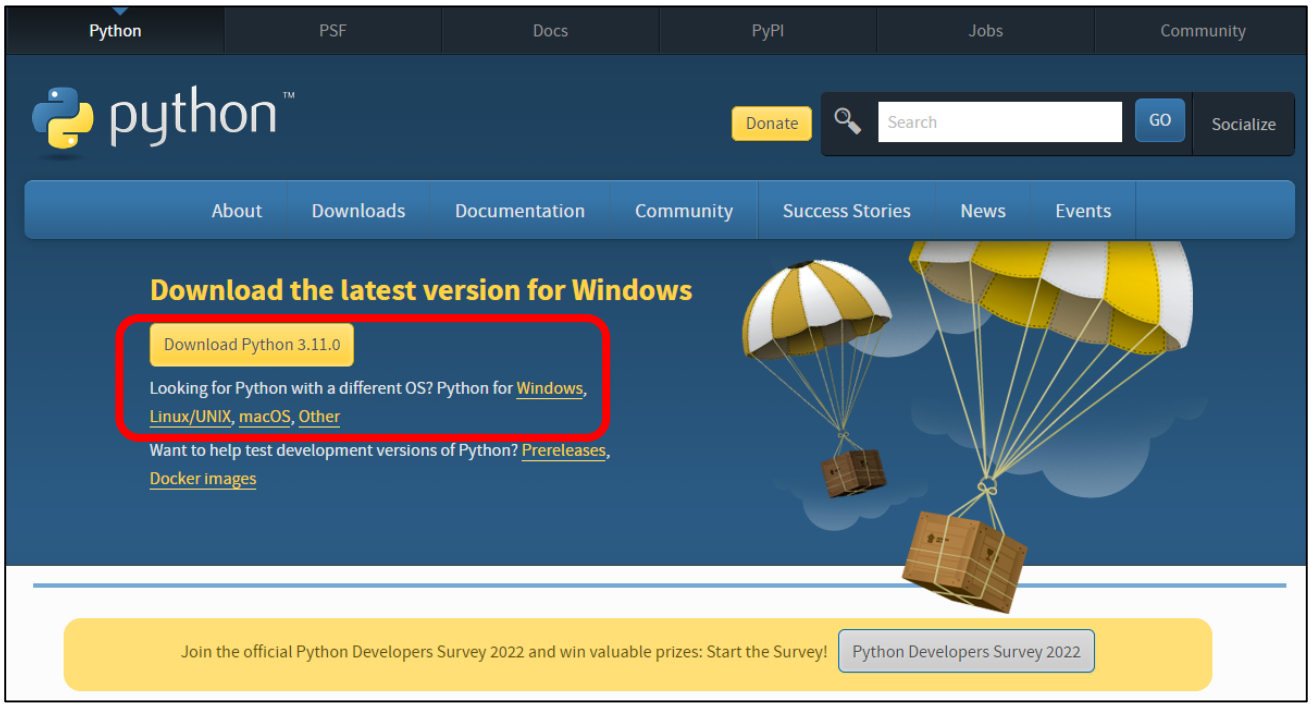
Official website of microPython: http://micropython.org/
Webpage listing firmware of microPython for ESP32S3:
https://micropython.org/download/ESP32_GENERIC_S3/



Firmware used in this tutorial is **GENERIC_S3-20220618-v1.19.1.bin**
This file is also provided in our data folder "**Freenove_ESP32_S3_WROVER_Board /Python/Python_Firmware**".。

Install python3

Before burning the firmware to ESP32S3, we need to ensure that Python 3 has been installed on the computer. If you have not already installed it, please install it first. Python Official download address is: https://www.python.org/downloads/

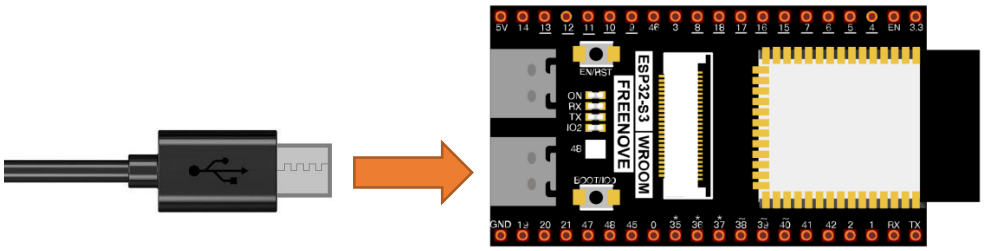


Please follow the official instructions to download and install.

Burning a Micropython Firmware

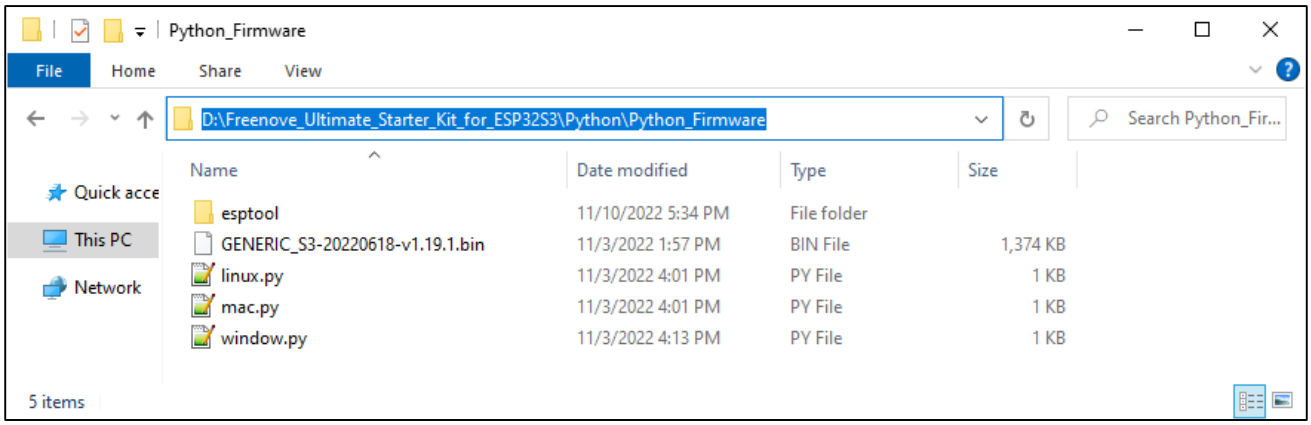
Window

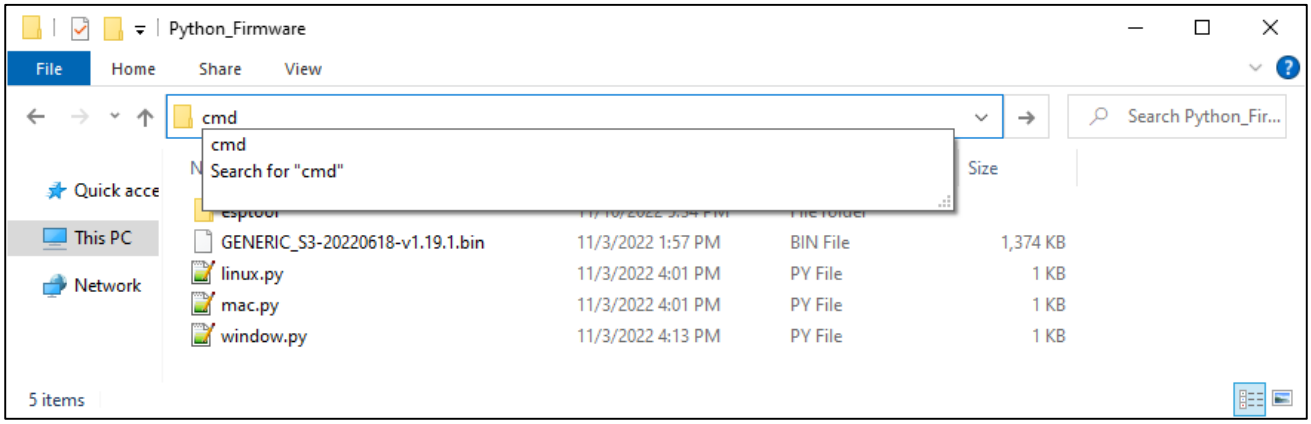
Connect your computer and ESP32-S3 with a USB cable.



Open **Freenove_ESP32_S3_WROVER_Board**/Python/**Python_Firmware**

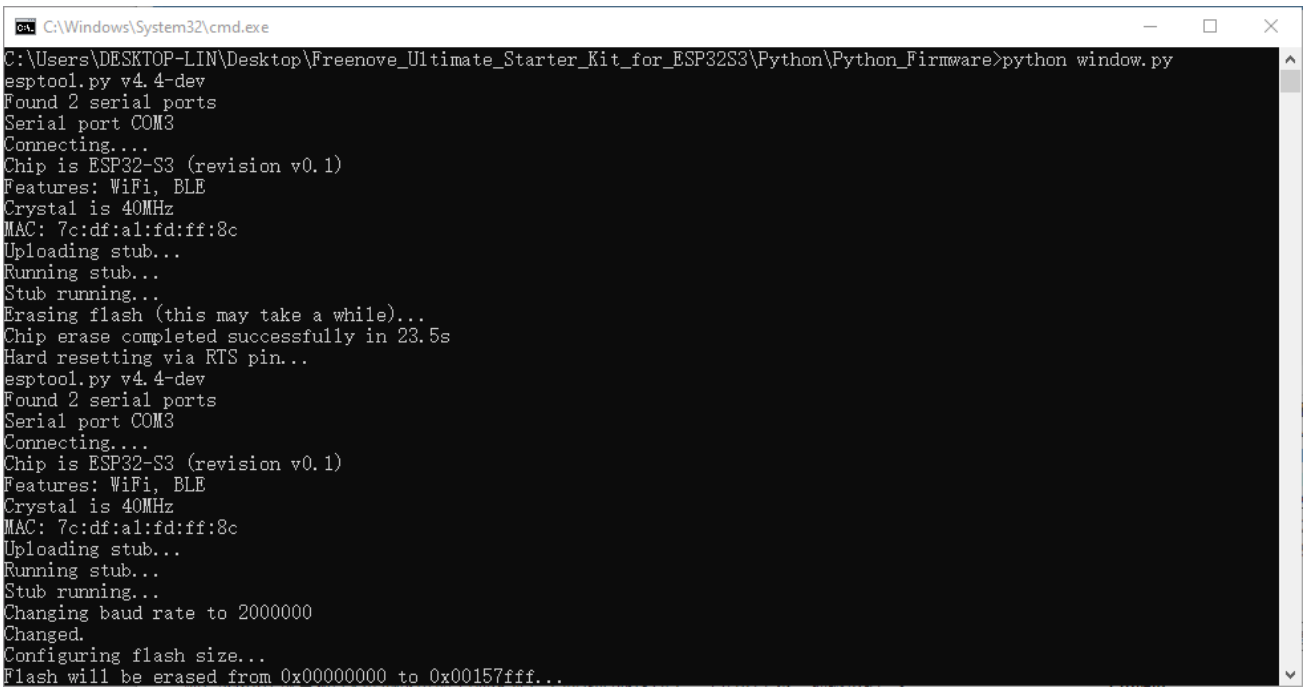
Enter cmd on path bar then press Enter.



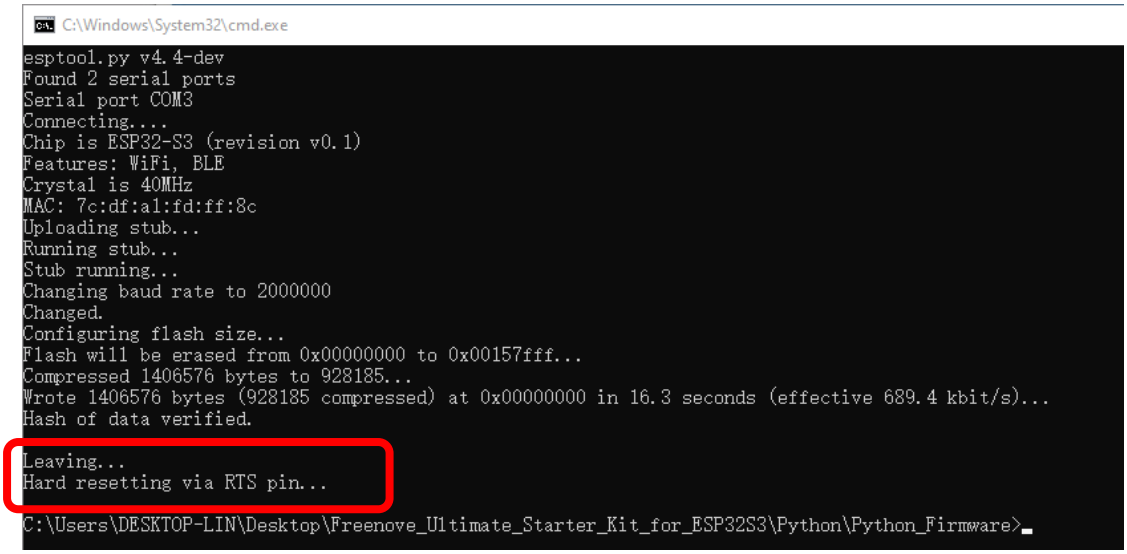


Here my python3 version is 3.8.1.

Enter “python window.py”. Micropython firmware will be automatically burned to ESP32S3.

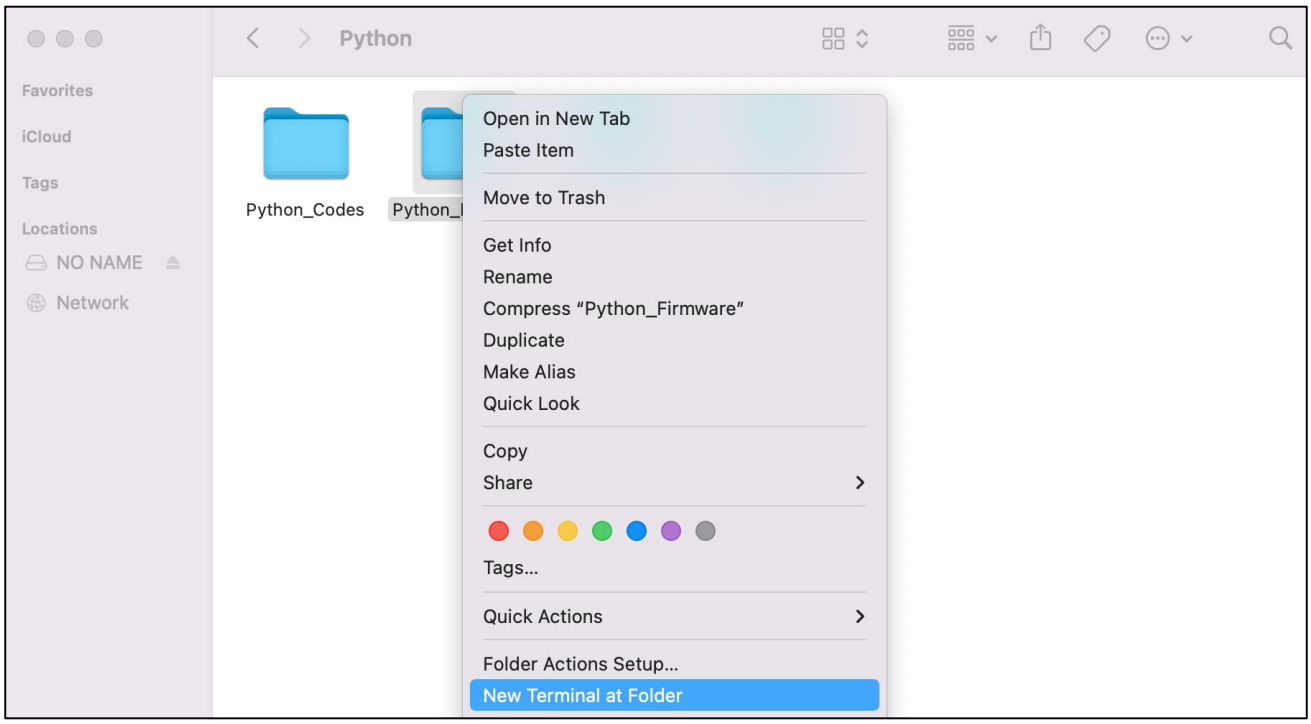

C:\Windows\System32\cmd.exe
C:\Users\DESKTOP-LIN\Desktop\Freenove_Ultimate_Starter_Kit_for_ESP32S3\Python\Python_Firmware>python window.py
esptool.py v4.4-dev
Found 2 serial ports
Serial port COM3
Connecting....
Chip is ESP32-S3 (revision v0.1)
Features: WiFi, BLE
Crystal is 40MHz
MAC: 7c:df:a1:fd:ff:8c
Uploading stub...
Running stub...
Stub running...
Erasing flash (this may take a while)...
Chip erase completed successfully in 23.5s
Hard resetting via RTS pin...
esptool.py v4.4-dev
Found 2 serial ports
Serial port COM3
Connecting....
Chip is ESP32-S3 (revision v0.1)
Features: WiFi, BLE
Crystal is 40MHz
MAC: 7c:df:a1:fd:ff:8c
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 2000000
Changed.
Configuring flash size...
Flash will be erased from 0x00000000 to 0x00157fff...

As shown in the figure below after completion.

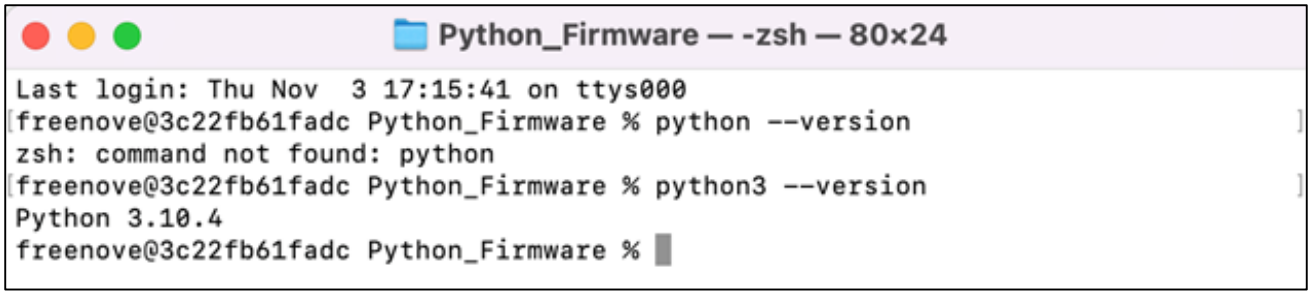


Mac

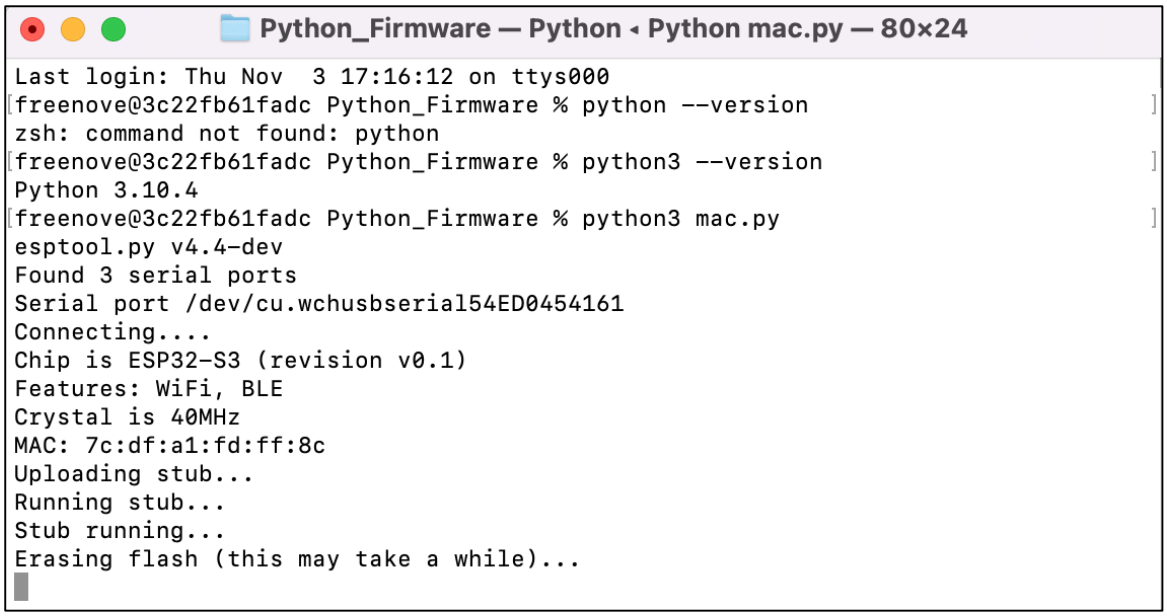
Open **Freenove_ESP32_S3_WROVER_Board/Python/Python_Firmware**. Right- click and select New Terminal at Folder.



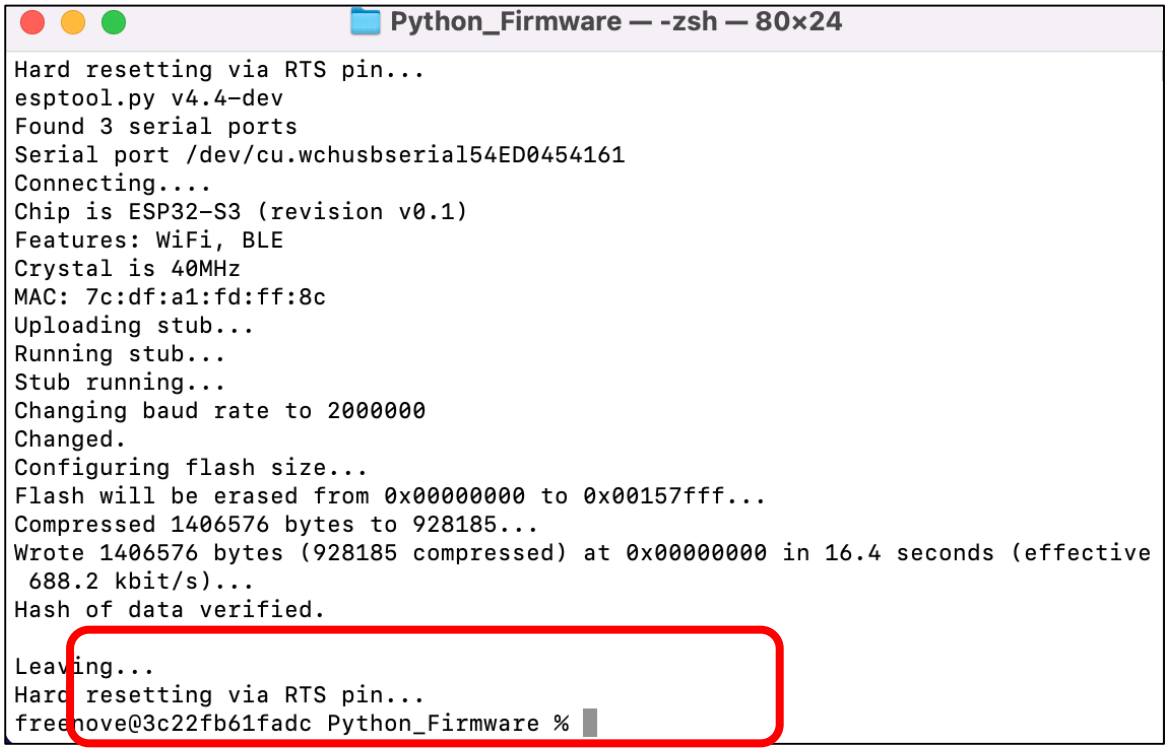
Here, my python3 version is 3.10.4



Enter "python3 mac. py" in the terminal, press Enter, and wait for the code to automatically burn the microython firmware into ESP32S3.



After completion, it is shown below.

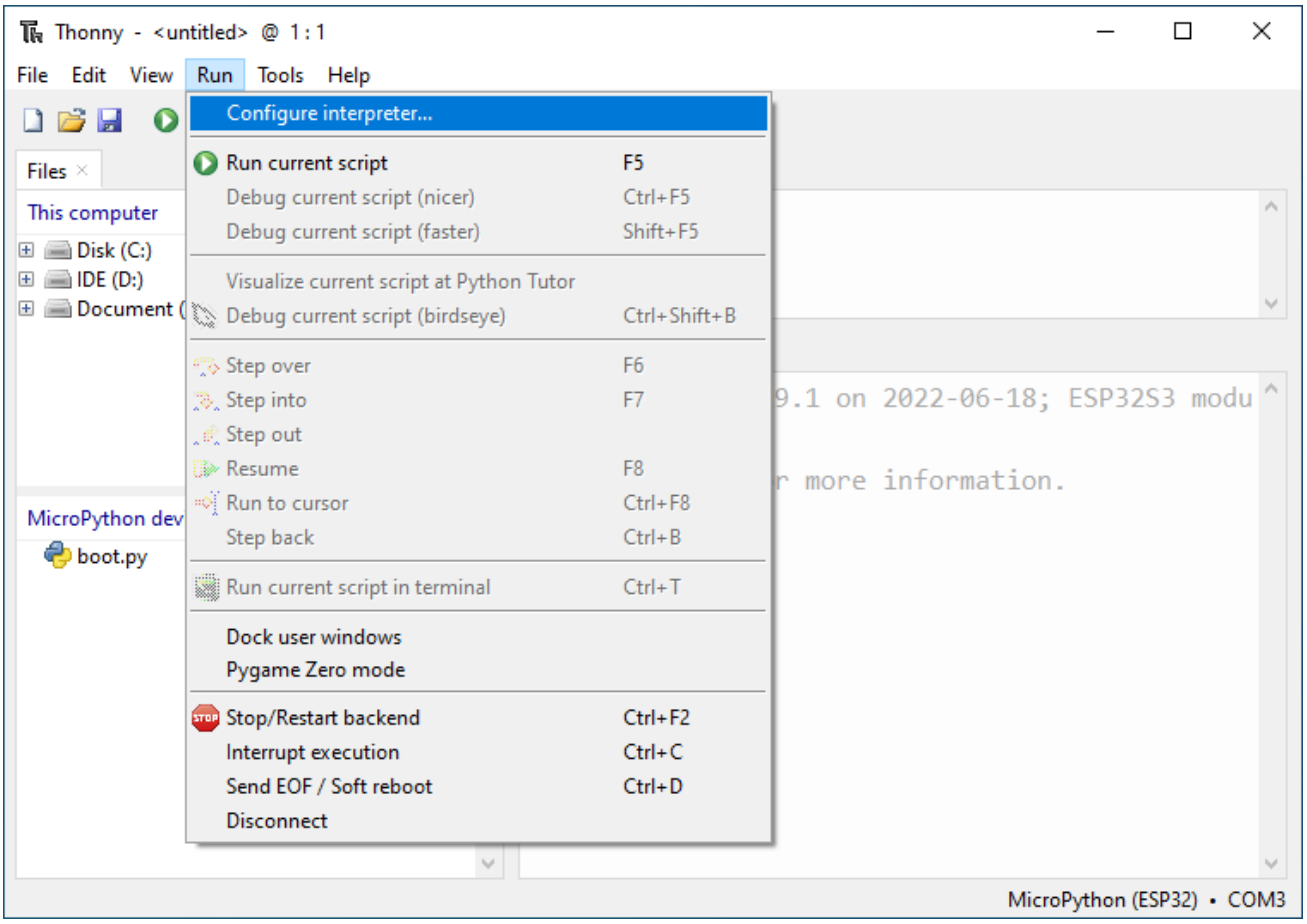


Note: The operation of the Linux system is similar to that of the Mac system. Please refer to the Mac system.

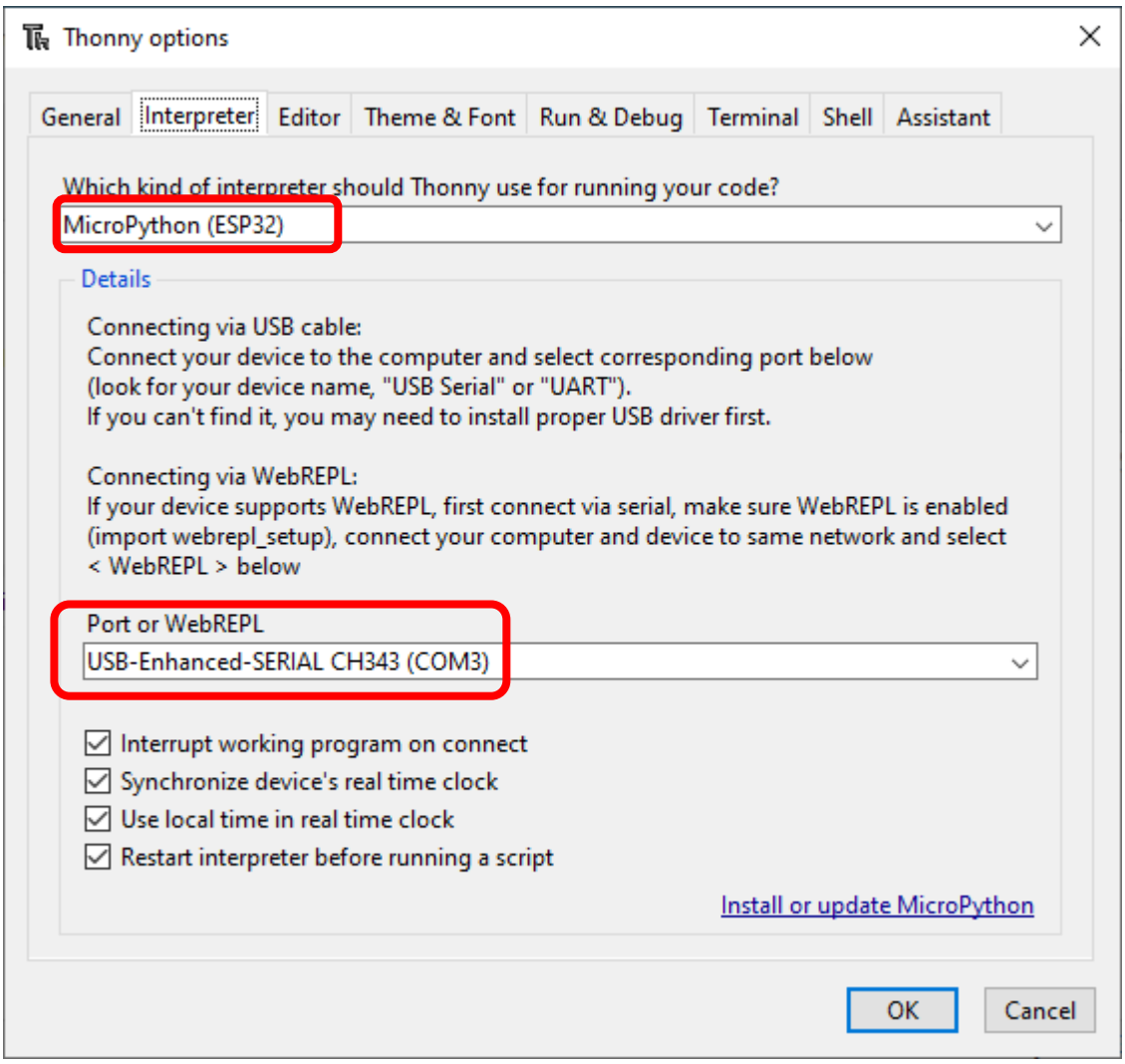
Testing codes (Important)

Testing Shell Command

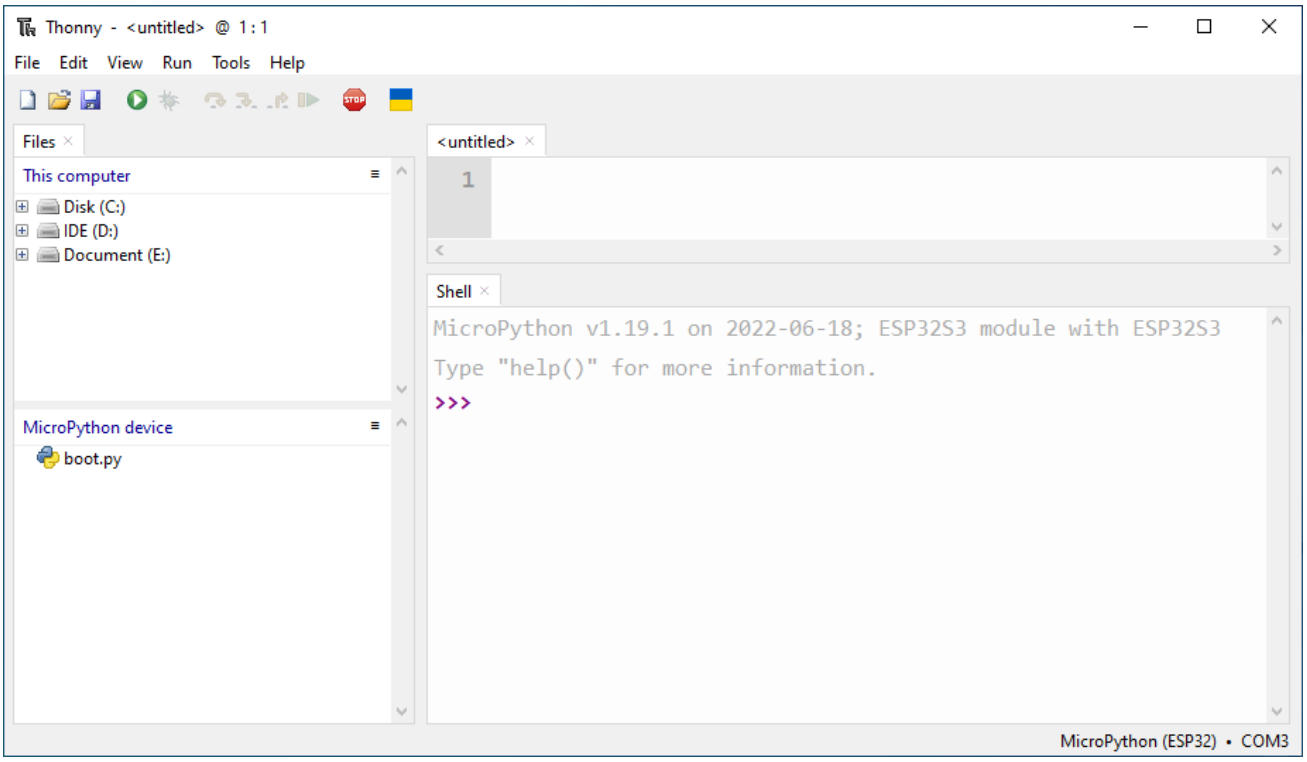
Make sure that the ESP 32S3 has burned the firmware and is connected to the computer through the data cable. Run Thonny. Click Run and select Configure interpreter.



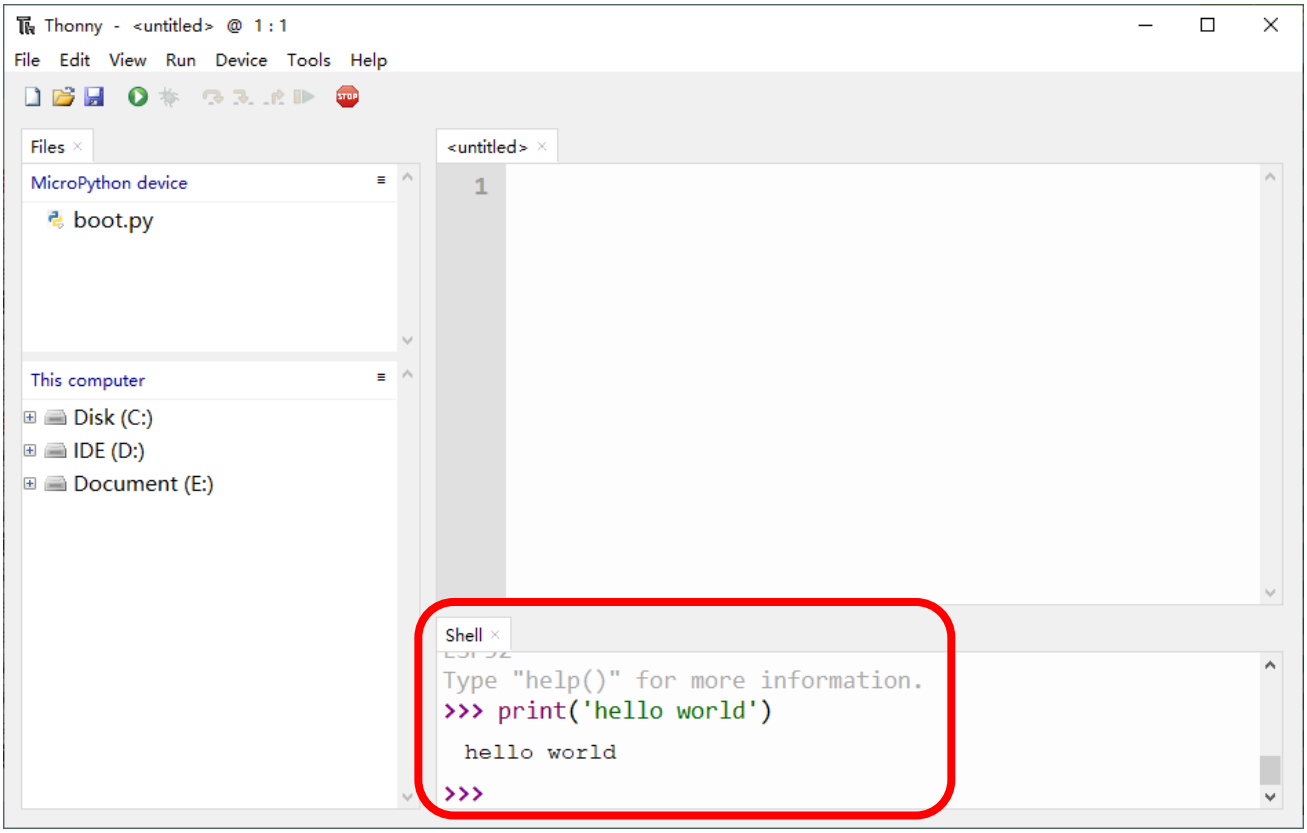
Please configure according to the following figure. Note that the port numbers of USB Enhanced SERIAL may be different for different systems. Please select according to the actual situation. After configuration, click OK.



After configuration, every time you open Thonny, it will communicate with ESP32S3. The interface is shown below.



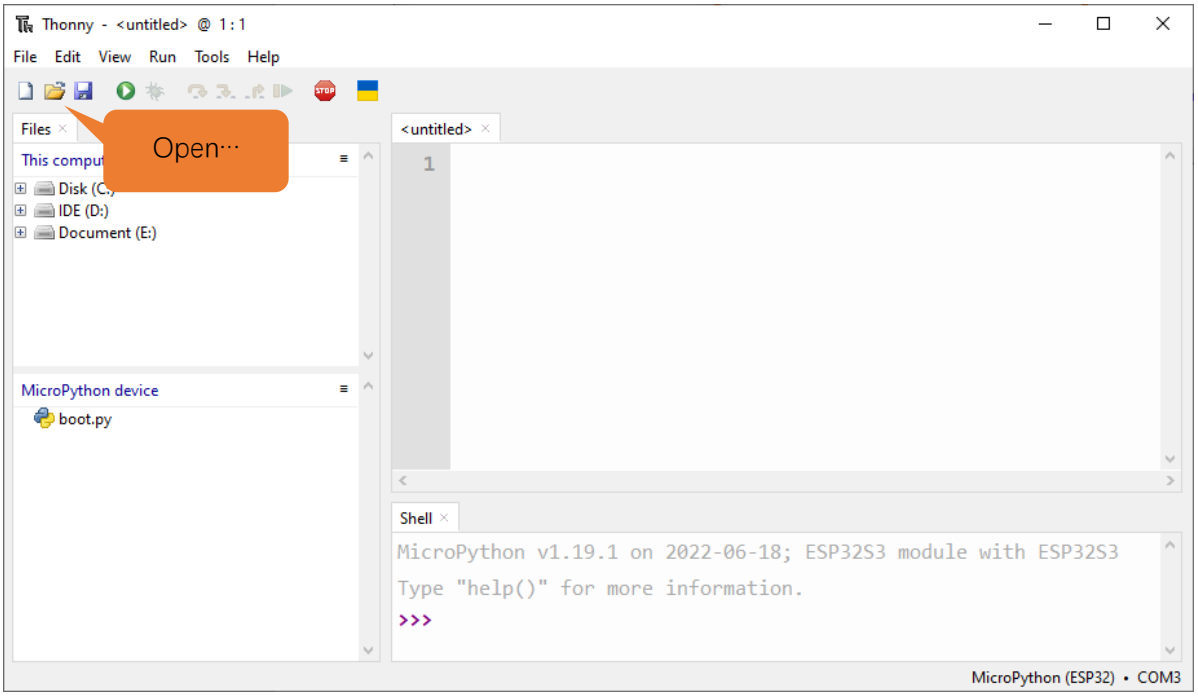
Enter "print('hello world')" in "Shell" and press Enter.



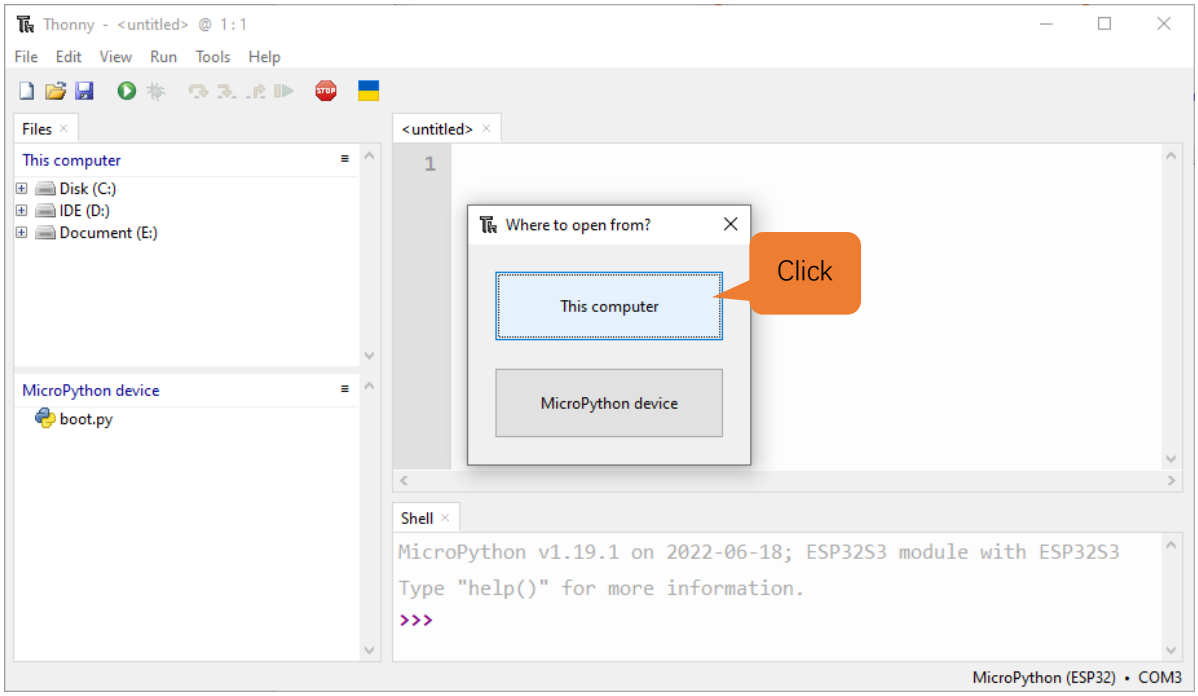
Running Online

ESP32-S3 needs to be connected to a computer when it is run online. Users can use Thonny to writer and debug programs.

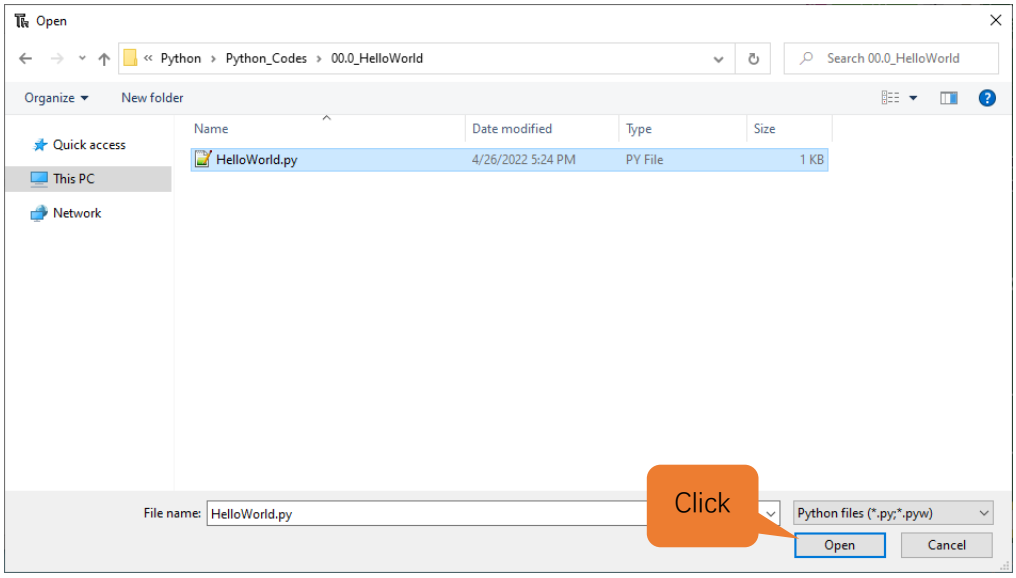
1. Open Thonny and click “Open···”.



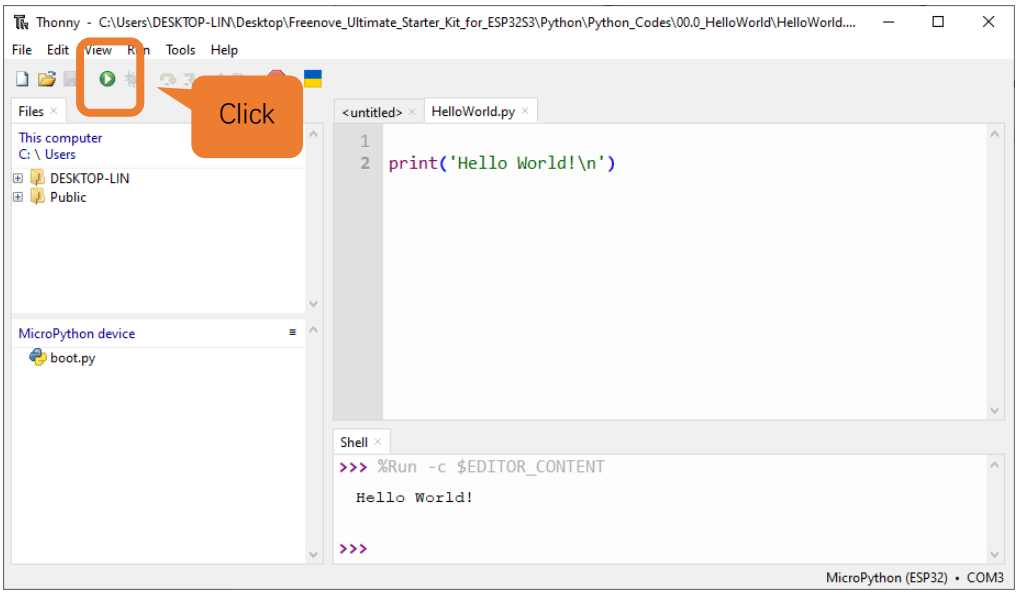
2. On the newly pop-up window, click “This computer”.



In the new dialog box, select "**HelloWorld.py**" in "**Freenove_ESP32_S3_WROVER_Board/Python/Python_Codes/00.0_HelloWorld**" folder.



Click "Run current script" to execute the program and "Hello World" will be printed in "Shell".

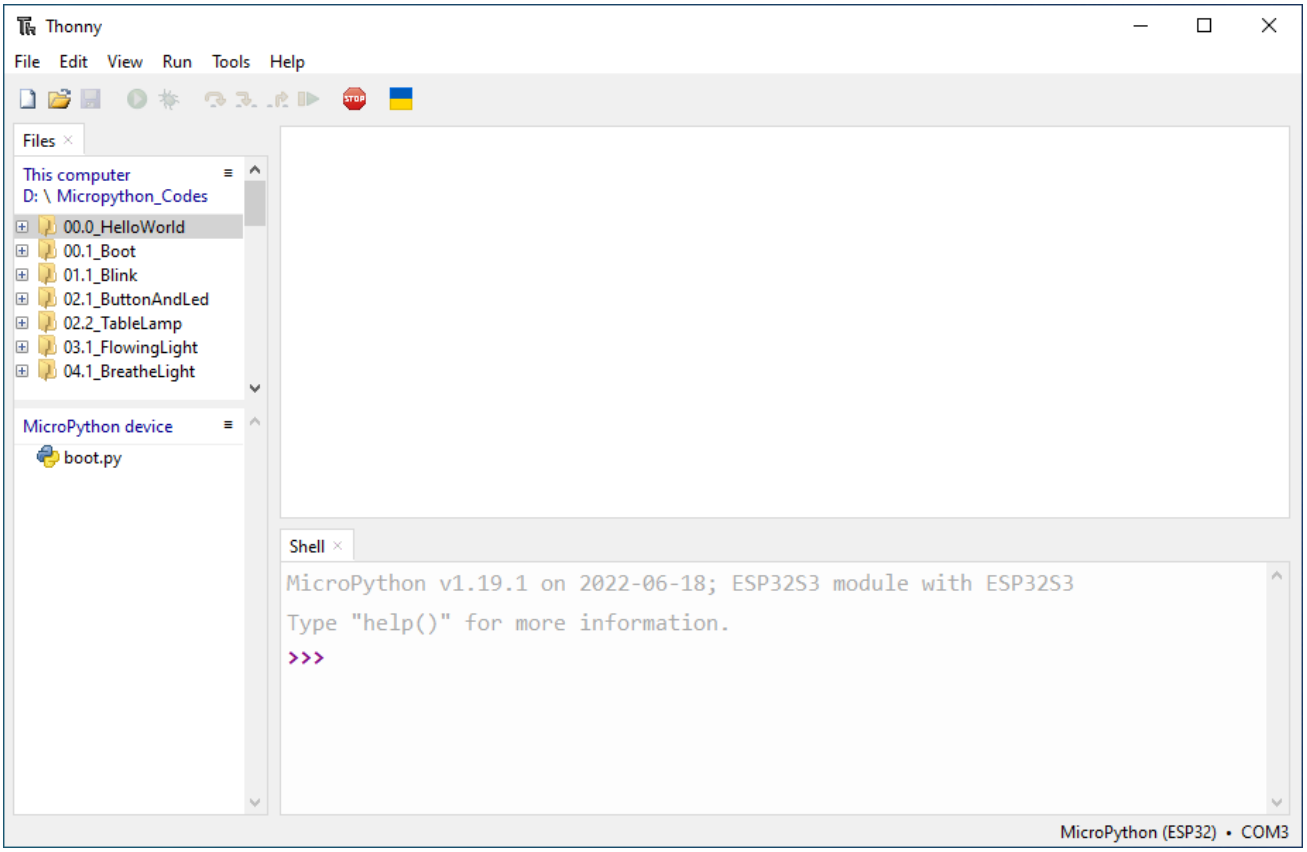


Note: When running online, if you press the reset key of ESP32S3, user's code will not be executed again. If you wish to run the code automatically after resetting the code, please refer to the following Running Offline.

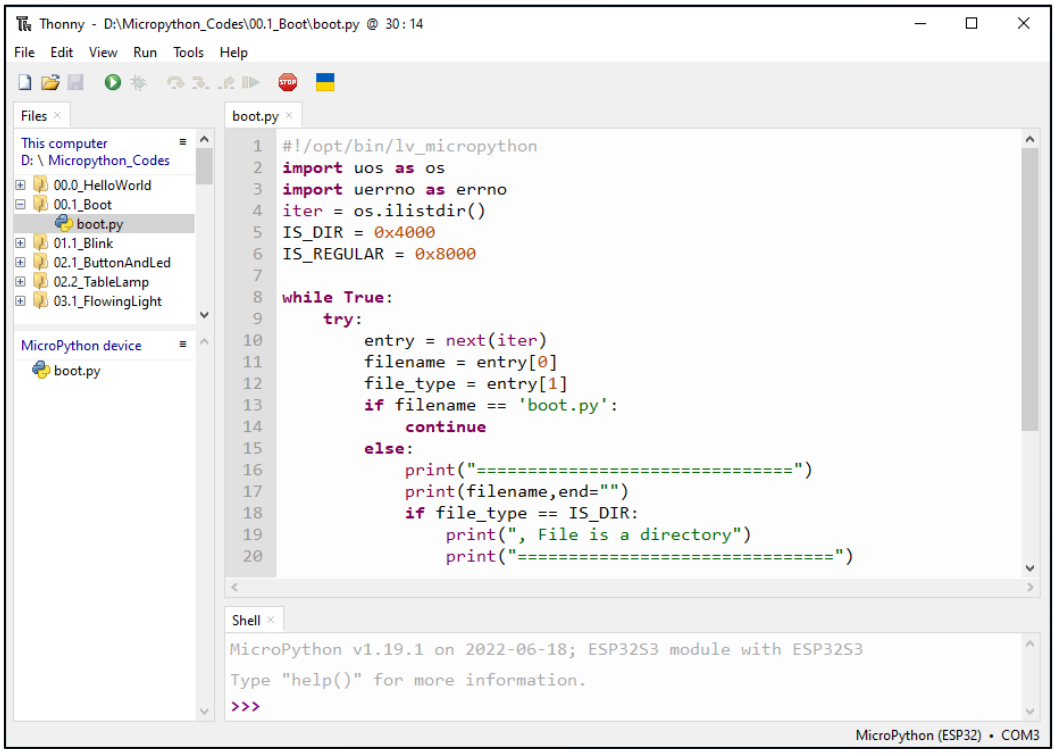
Running Offline（Importance）

After ESP32-S3 is reset, it runs the file boot.py in root directory first and then runs file main.py, and finally, it enters "Shell". Therefore, to make ESP32-S3 execute user's programs after resetting, we need to add a guiding program in boot.py to execute user's code.

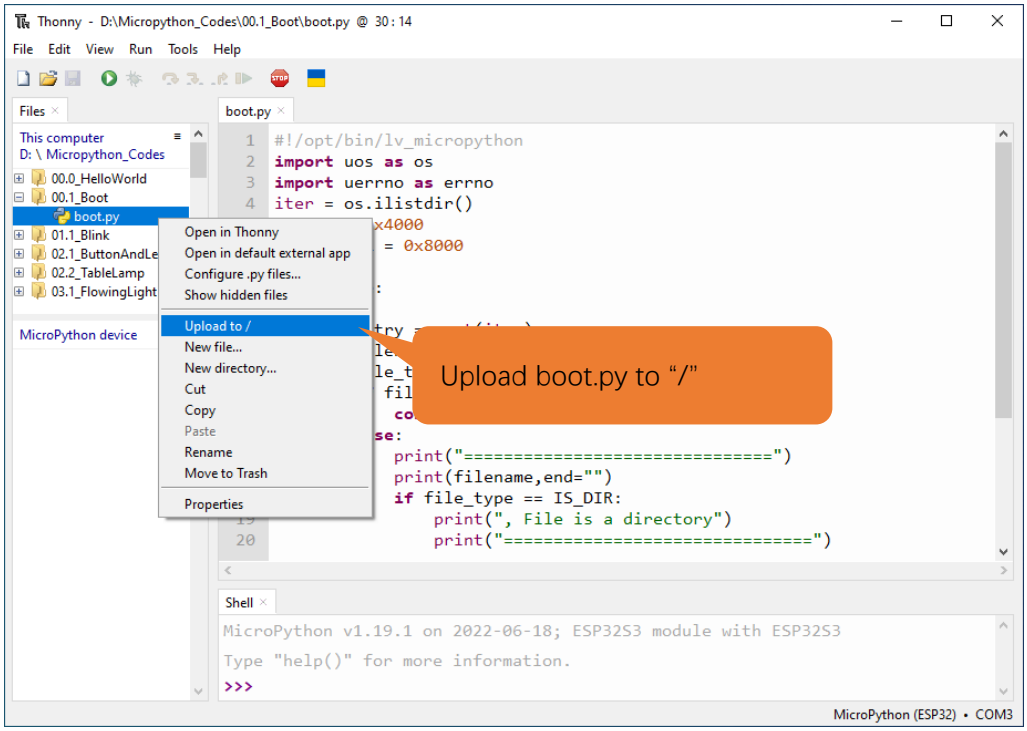
Move the program folder "**Freenove_ESP32_S3_WROVER_Board/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**". Open "Thonny"。



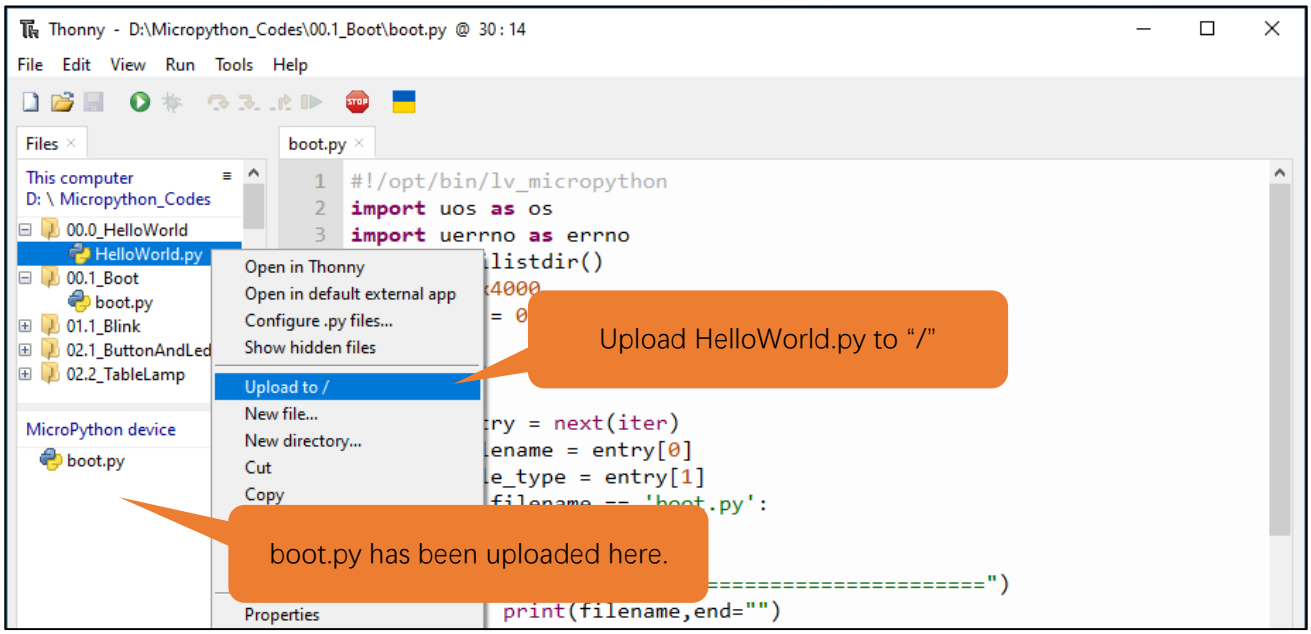
Expand "00.1_Boot" in the "Micropython_Codes" in the directory of disk(D), and double-click boot.py, which is provided by us to enable programs in "MicroPython device" to run offline.



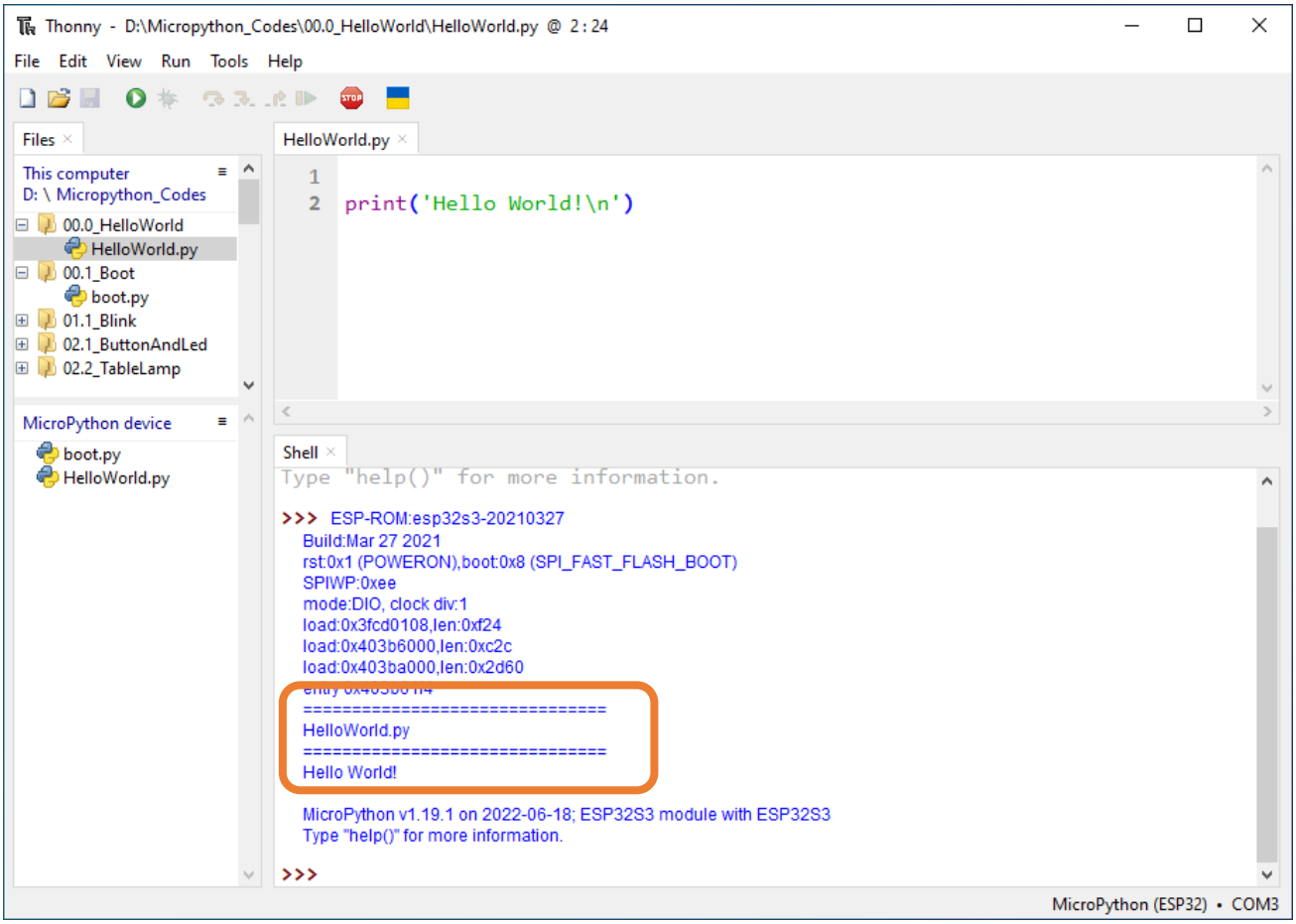
If you want your written programs to run offline, you need to upload boot.py we provided and all your codes to "MicroPython device" and press ESP32S3's reset key. Here we use programs 00.0 and 00.1 as examples. Select "boot.py", right-click to select "Upload to /".



Similarly, upload "HelloWorld.py" to "MicroPython device".



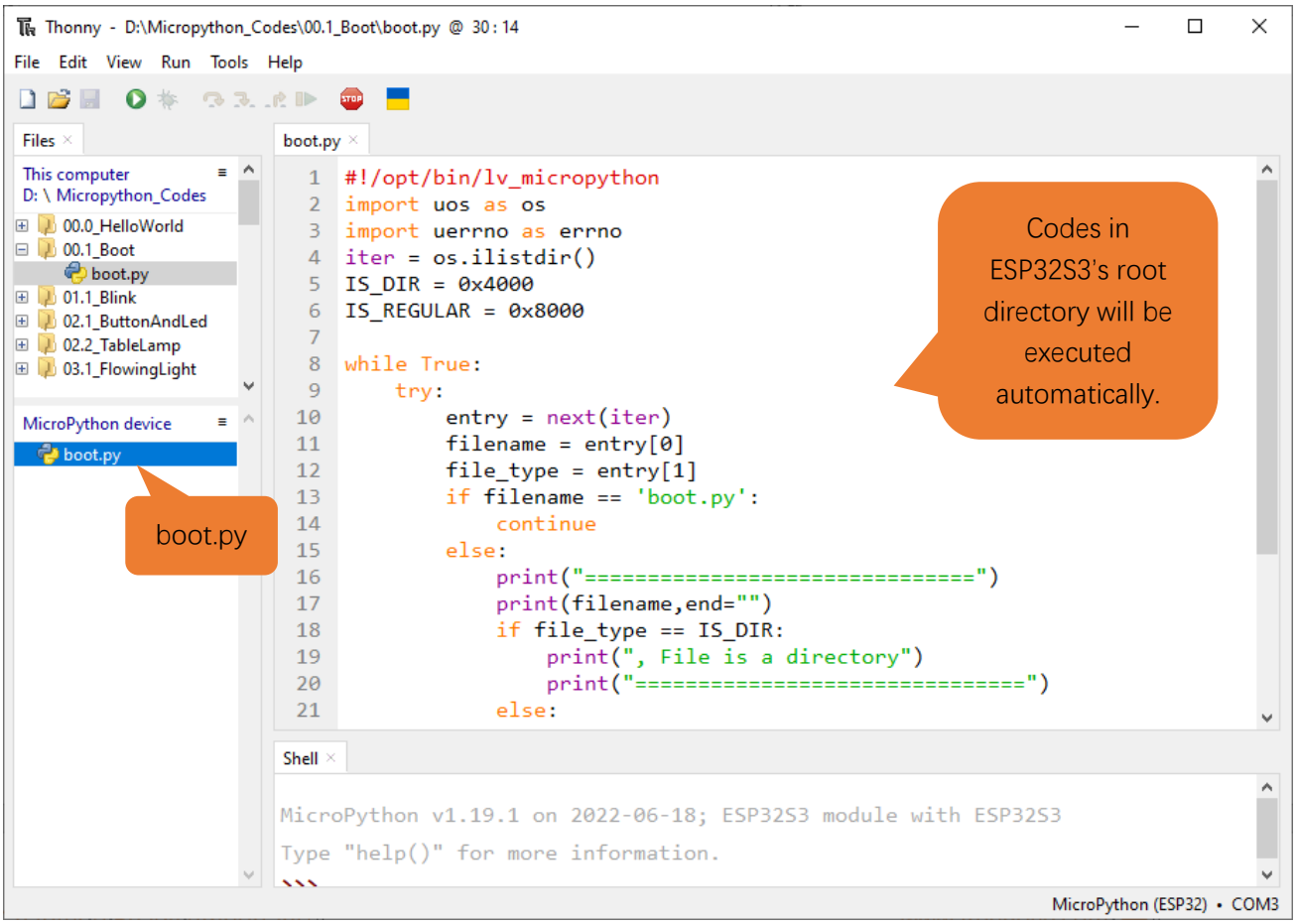
Press the reset key and in the box of the illustration below, you can see the code is executed.



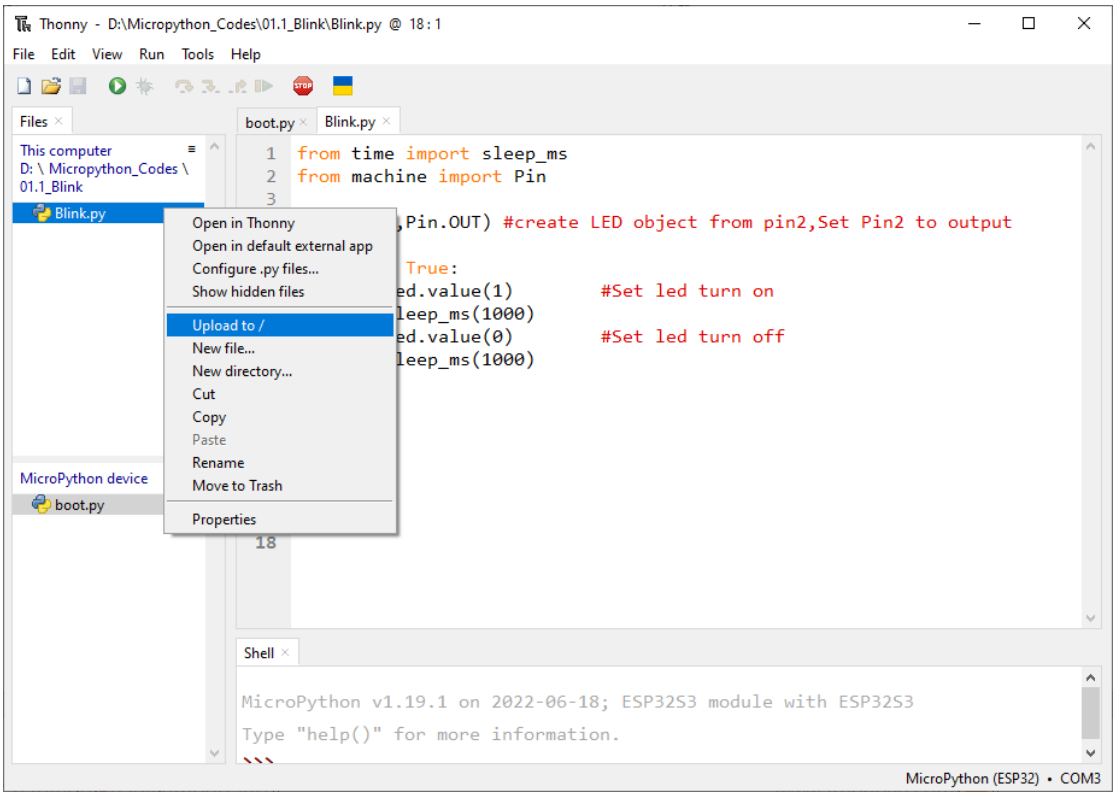
Thonny Common Operation

Uploading Code to ESP32S3

Each time when ESP32-S3 restarts, if there is a "boot.py" in the root directory, it will execute this code first.

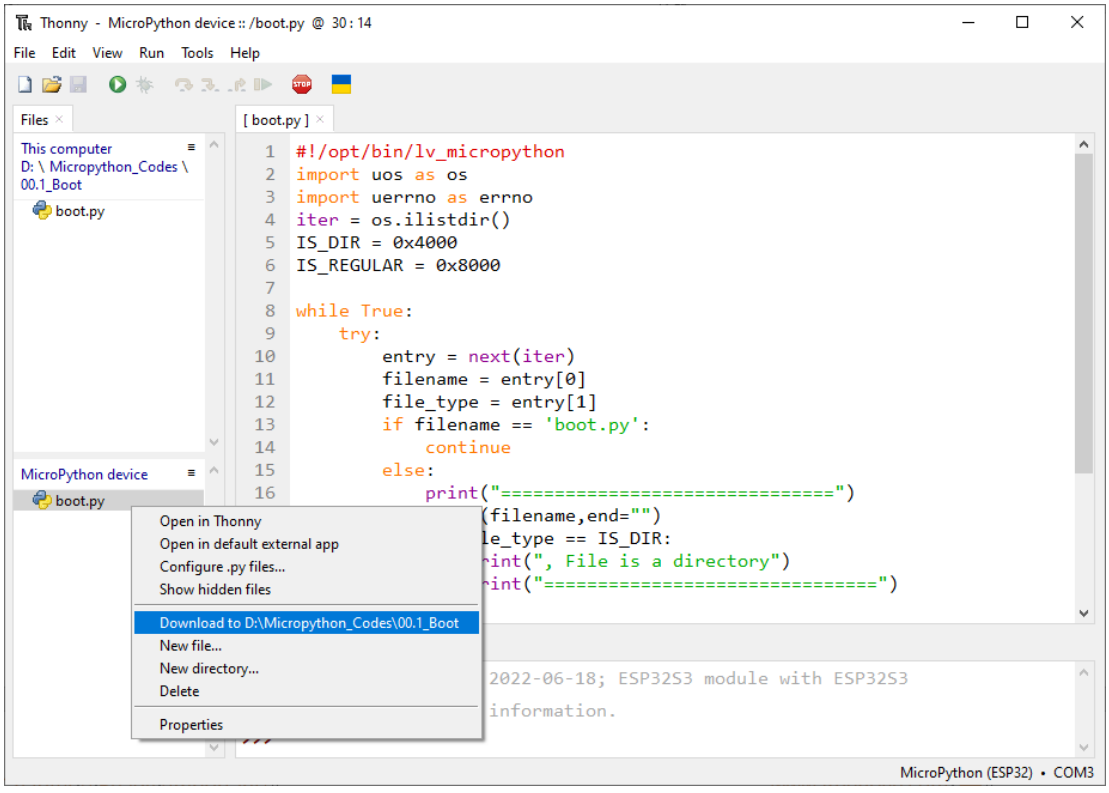


Select "Blink.py" in "01.1_Blink", right-click your mouse and select "Upload to /" to upload code to ESP32S3's root directory.



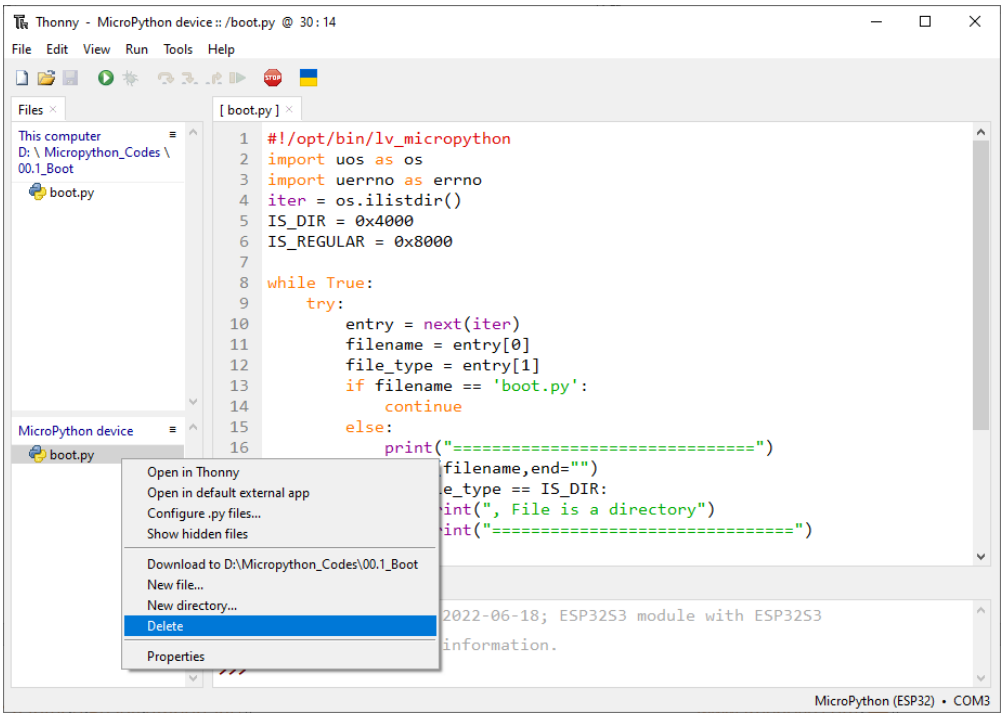
Downloading Code to Computer

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



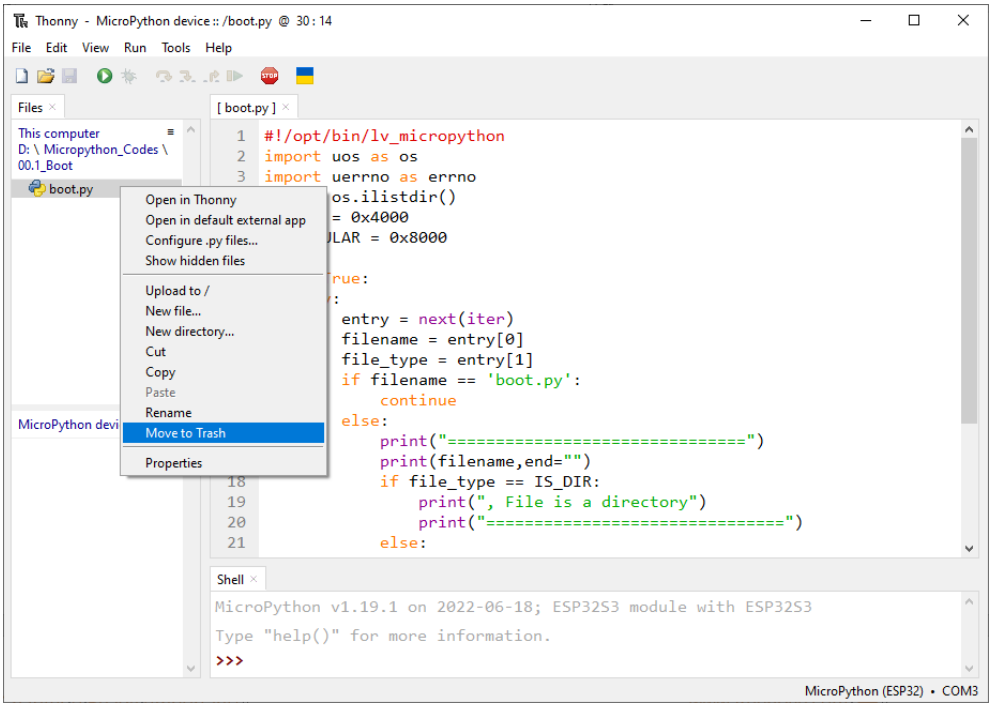
Deleting Files from ESP32S3's Root Directory

Select "boot.py" in "MicroPython device", right-click it and select "Delete" to delete "boot.py" from ESP32S3's root directory.



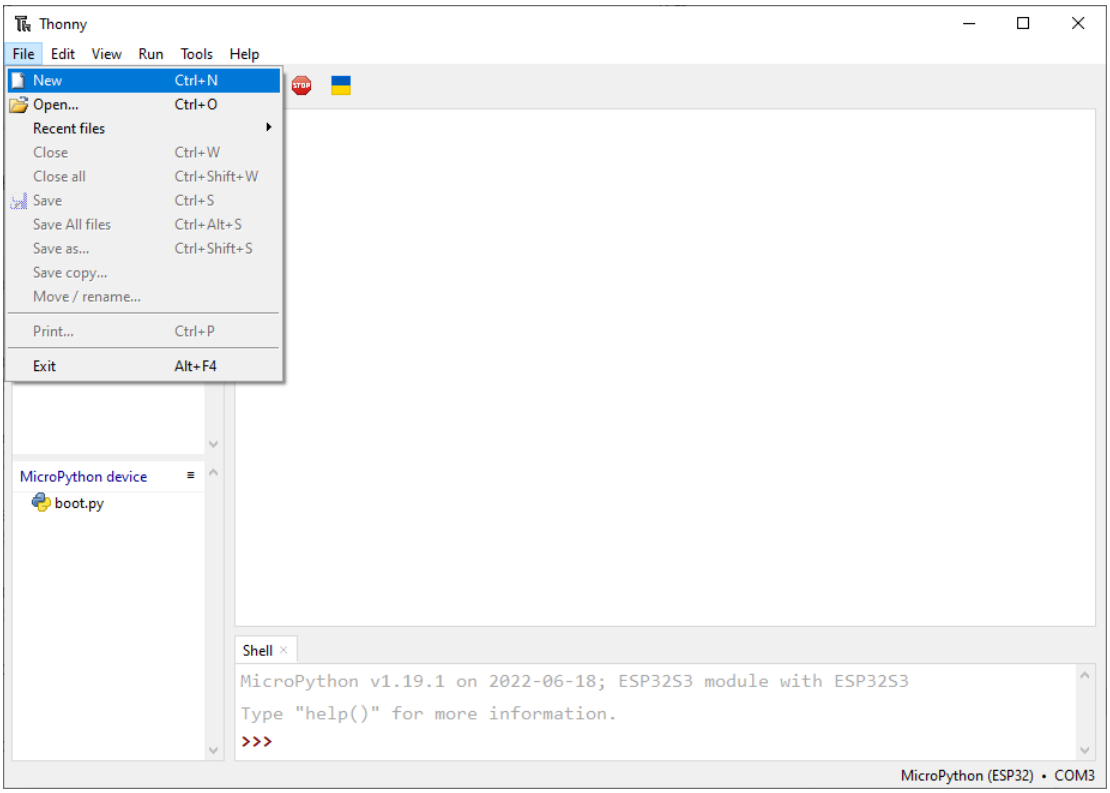
Deleting Files from your Computer Directory

Select "boot.py" in "00.1_Boot", right-click it and select "Move to Recycle Bin" to delete it from "00.1_Boot".

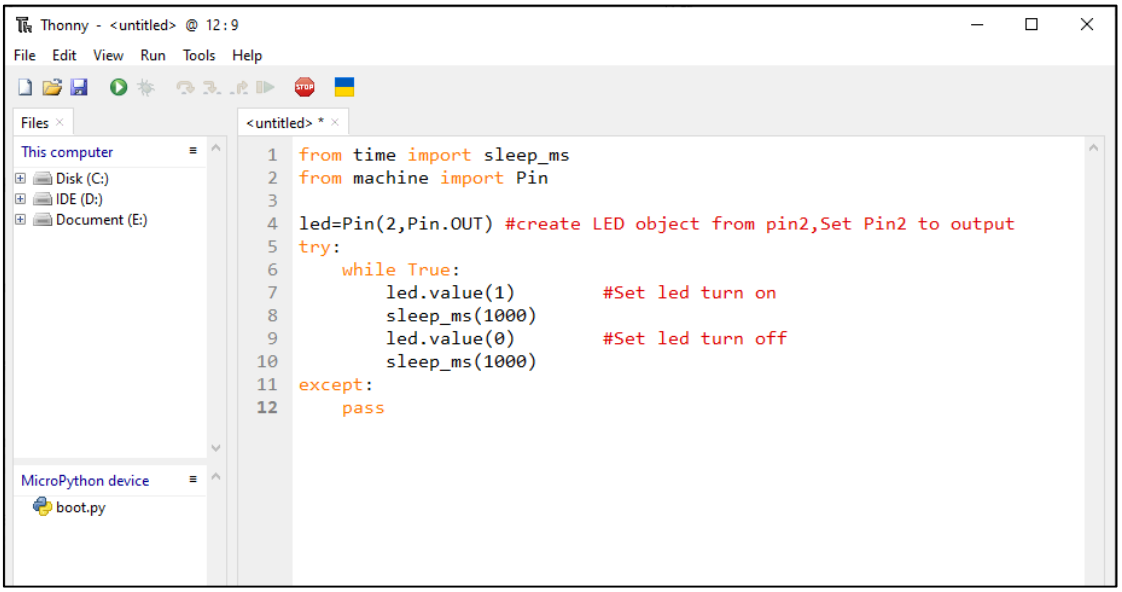


Creating and Saving the code

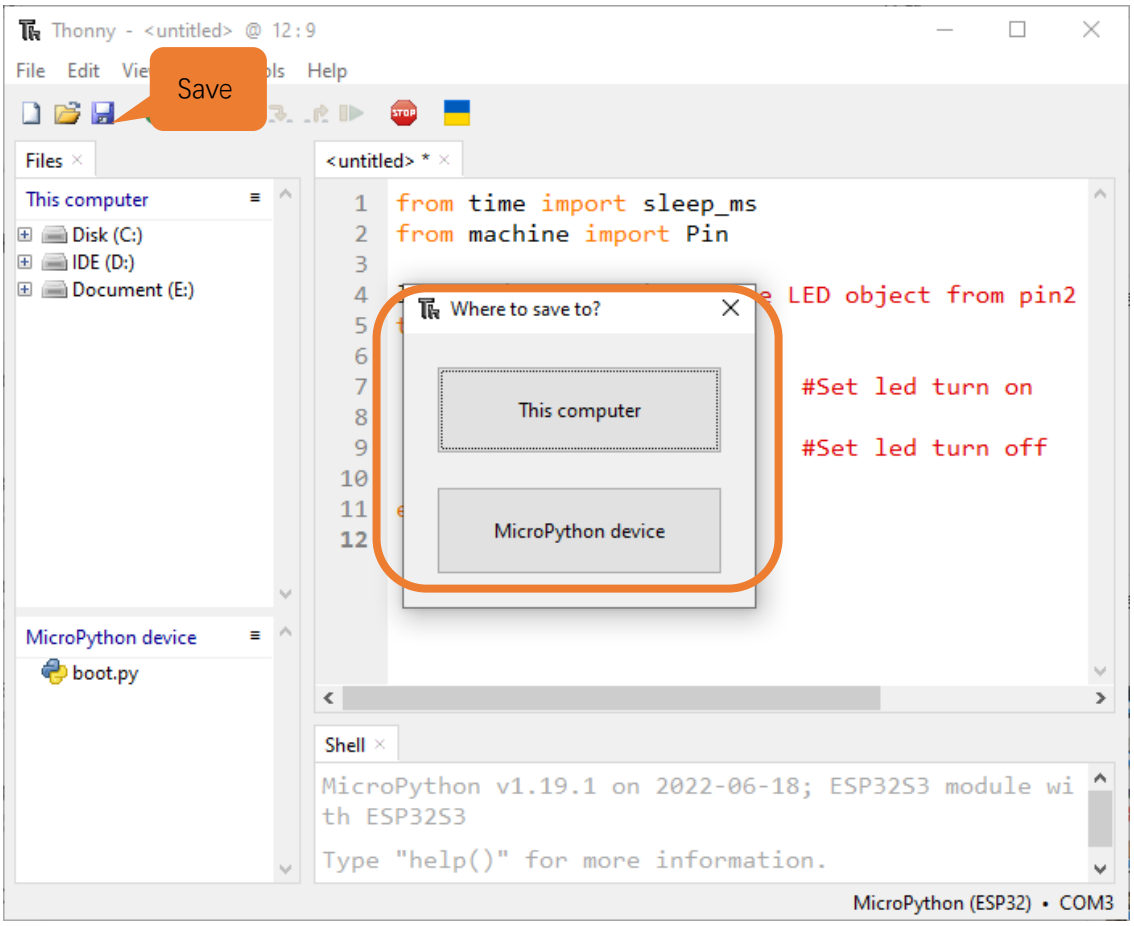
Click "File"→"New" to create and write codes.



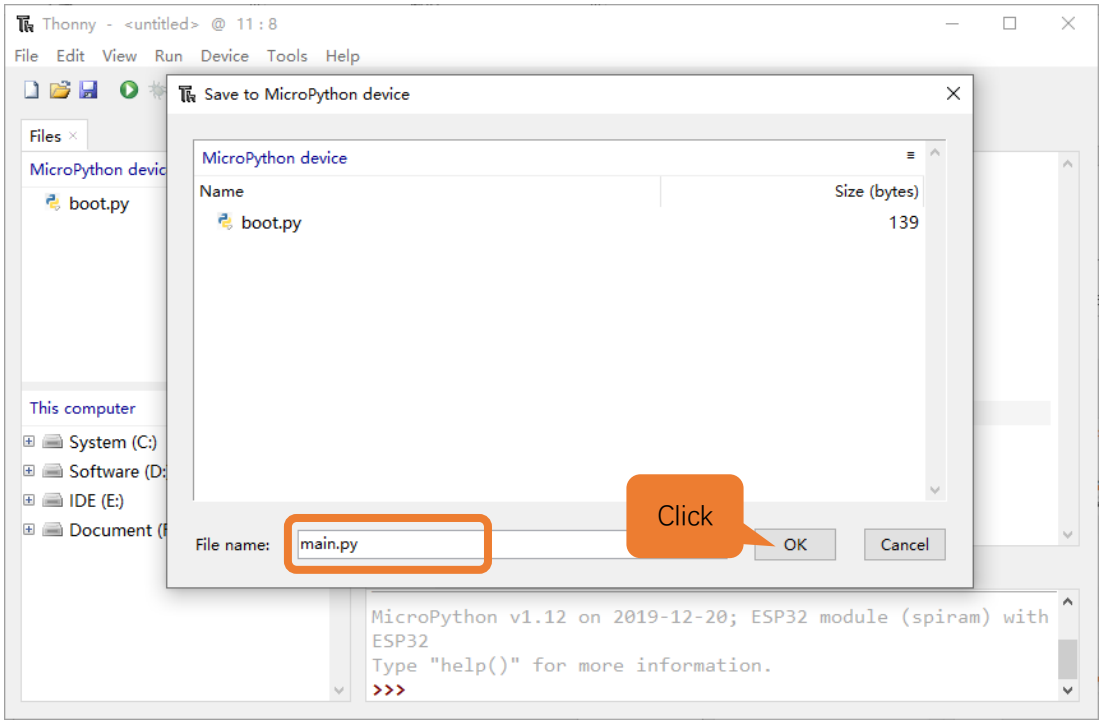
Enter codes in the newly opened file. Here we use codes of "01.1_Blink.py" as an example.



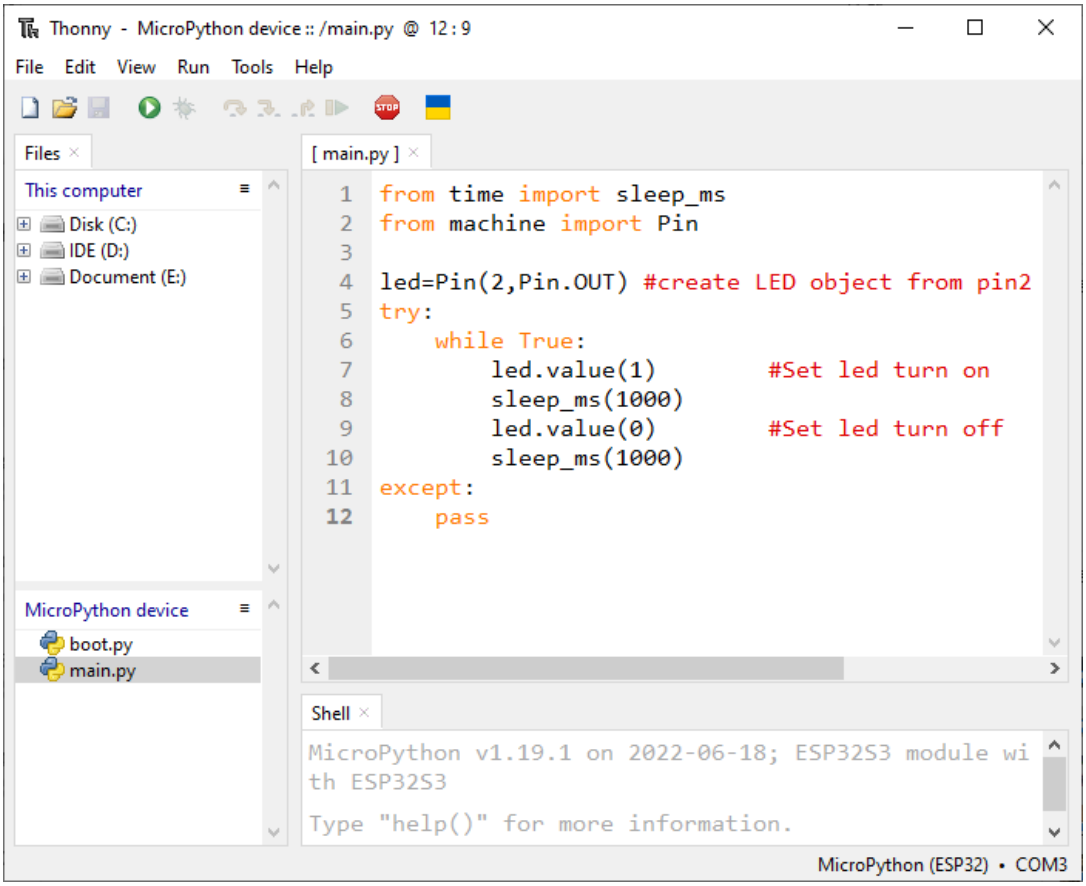
Click “Save” on the menu bar. You can save the codes either to your computer or to ESP32S3.



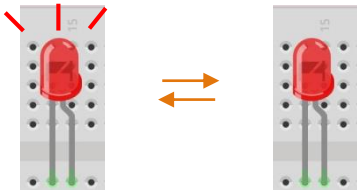
Select "MicroPython device", enter "main.py" in the newly pop-up window and click "OK".



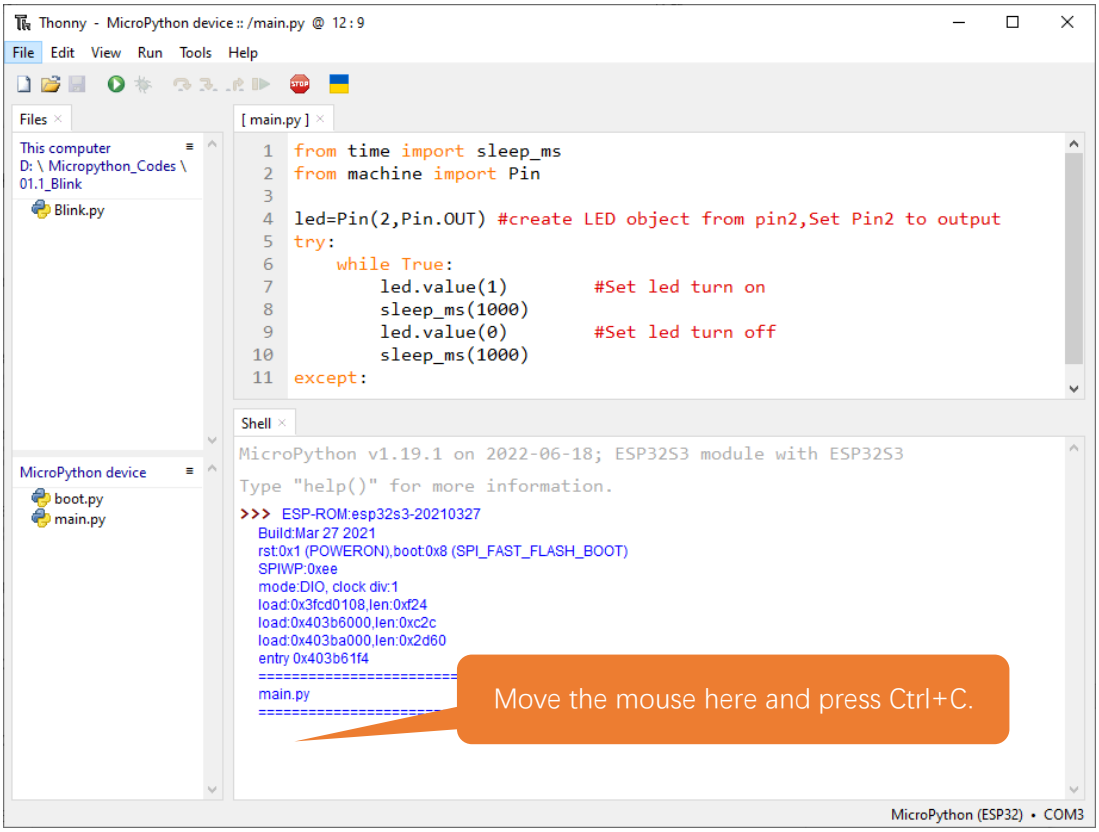
You can see that codes have been uploaded to ESP32S3.



Disconnect and reconnect USB cable, and you can see that LED is ON for one second and then OFF for one second, which repeats in an endless loop.



If you want to exit the offline operation mode, you can press Ctrl+C at the same time in the shell to let the ESP32-S3 exit the offline operation mode.



If there is no response after pressing, it is recommended to press again until exiting.

Notes for GPIO

Strapping Pin

There are four Strapping pins for ESP32S3: GPIO0、GPIO45、GPIO46、GPIO3。
With the release of the chip's system reset (power-on reset, RTC watchdog reset, undervoltage reset), the strapping pins sample the level and store it in the latch as "0" or "1" ", and keep it until the chip is powered off or turned off.
Each Strapping pin is connecting to internal pull-up/pull-down. Connecting to high-impedance external circuit or without an external connection, a strapping pin's default value of input level will be determined by internal weak pull-up/pull-down. To change the value of the Strapping, users can apply an external pull-down/pull-up resistor, or use the GPIO of the host MCU to control the level of the strapping pin when the ESP32-S3's power on reset is released.
When releasing the reset, the strapping pin has the same function as a normal pin.
The followings are default configurations of these four strapping pins at power-on and their functions under the corresponding configuration.

VDD_SPI Voltage			
Pin	Default	3.3 V	1.8 V
GPIO45	Pull-down	0	1
Booting Mode [1]			
Pin	Default	SPI Boot	Download Boot
GPIO0	Pull-up	1	0
GPIO46	Pull-down	Don't care	0
Enabling/Disabling ROM Messages Print During Booting [2]			
Pin	Default	Enabled	Disabled
GPIO46	Pull-down	See the 2nd note	See the 2nd note
JTAG Signal Selection			
Pin	Default	EFUSE_DIS_USB_JTAG = 0, EFUSE_DIS_PAD_JTAG = 0, EFUSE_STRAP_JTAG_SEL=1	
GPIO3	N/A	0: JTAG signal from on-chip JTAG pins 1: JTAG signal from USB Serial/JTAG controller	

Note:

1. The strapping combination of GPIO46 = 1 and GPIO0 = 0 is invalid and will trigger unexpected behavior.
2. By default, the ROM boot messages are printed over UART0 (U0TXD pin) and USB Serial/JTAG controller together. The ROM code printing can be disabled through configuration register and eFuse. For detailed information, please refer to Chapter Chip Boot Control in *ESP32-S3 Technical Reference Manual*.

If you have any difficulties or questions with this tutorial or toolkit, feel free to ask for our quick and free technical support through support@freenove.com at any time.
or check: https://www.espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf

Any concerns? ✉ support@freenove.com

PSRAM Pin

The modules on the ESP32-S3 WROOM board use the ESP32-S3R8 chip with 8MB external Flash. When using OPI PSRAM, please note that GPIO35-GPIO37 on the ESP32-S3 WROOM board cannot be used for other purposes. When OPI PSRAM is not used, GPIO35-GPIO37 on the board can be used as a common GPIO.

ESP32-S3R8 / ESP32-S3R8V	In-package PSRAM (8 MB, Octal SPI)
SPICLK	CLK
SPICS1	CE#
SPID	DQ0
SPIQ	DQ1
SPIWP	DQ2
SPIHD	DQ3
GPIO33	DQ4
GPIO34	DQ5
GPIO35	DQ6
GPIO36	DQ7
GPIO37	DQS/DM

SDcard Pin

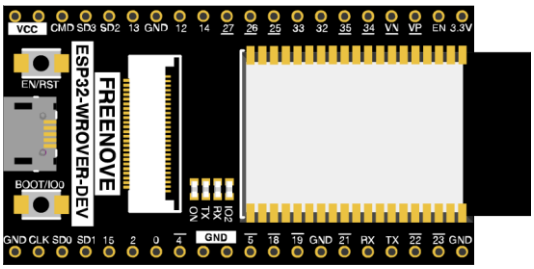
An SD card slot is integrated on the back of the ESP32-S3 WROOM board. We can use GPIO38-GPIO40 of ESP32-S3 WROOM to drive the SD card.

USB Pin

In Micropython, GPIO19 and GPIO20 are used for the USB function of ESP32S3, so they cannot be used as other functions!

Cam Pin

When using the camera of our ESP32-S3 WROOM, please check the pins of it. Pins with underlined numbers are used by the camera function, if you want to use other functions besides it, please avoid using them.



CAM_Pin	GPIO_pin
SIOD	GPIO4
SIOC	GPIO5
CSI_VYSNC	GPIO6
CSI_HREF	GPIO7
CSI_Y9	GPIO16
XCLK	GPIO15
CSI_Y8	GPIO17
CSI_Y7	GPIO18
CSI_PCLK	GPIO13
CSI_Y6	GPIO12
CSI_Y2	GPIO11
CSI_Y5	GPIO10
CSI_Y3	GPIO9
CSI_Y4	GPIO8

If you have any questions about the information of GPIO, you can click here to go back to ESP32-S3 WROOM to view specific information about GPIO.
or check: https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf.

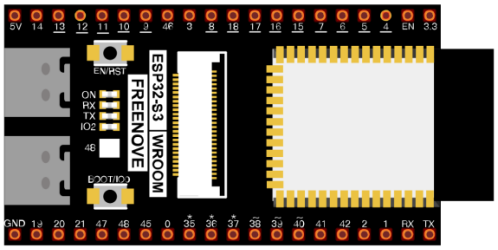
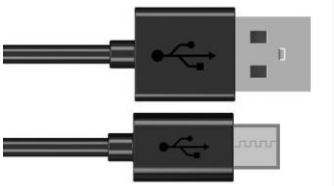
Chapter 1 LED (Important)

This chapter is the Start Point in the journey to build and explore ESP32-S3 WROOM electronic projects. We will start with simple “Blink” project.

Project 1.1 Blink

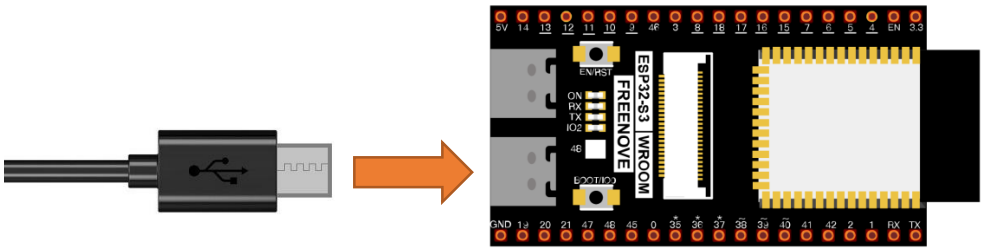
In this project, we will use ESP32-S3 WROOM to control blinking a common LED.

Component List

ESP32-S3 WROOM x1	USB cable x1
	

Power

ESP32-S3 WROOM needs 5v power supply. In this tutorial, we need connect ESP32-S3 WROOM to computer via USB cable to power it and program it. We can also use other 5v power source to power it.



In the following projects, we only use USB cable to power ESP32-S3 WROOM by default.

In the whole tutorial, we don’t use T extension to power ESP32-S3 WROOM. So 5V and 3.3V (includeing EXT 3.3V) on the extension board are provided by ESP32-S3 WROOM.

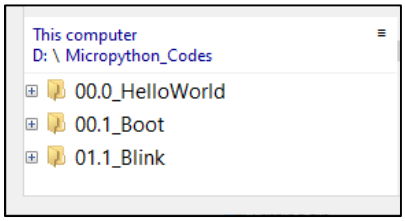
We can also use DC jack of extension board to power ESP32-S3 WROOM. In this way, 5v and EXT 3.3v on extension board are provided by external power resource.

Code

Codes used in this tutorial are saved in "**Freenove_ESP32_S3_WROVER_Board/Python/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**".

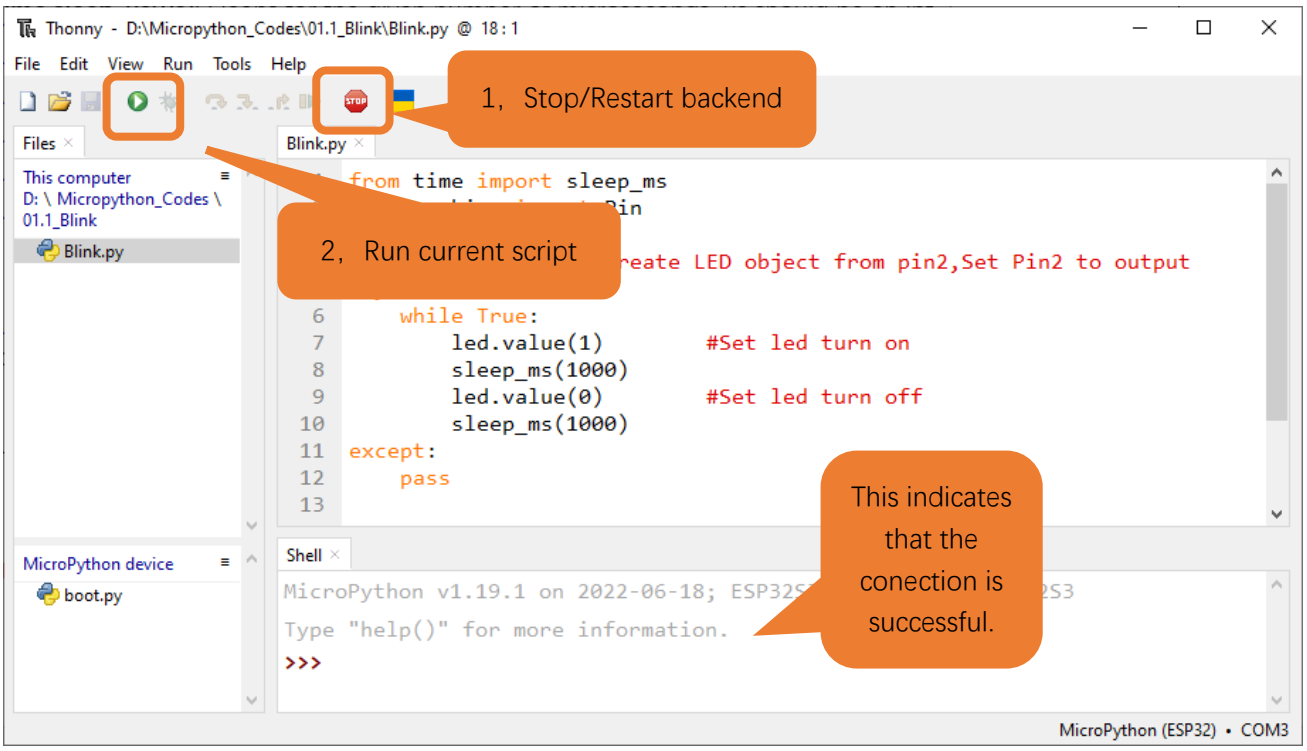
01.1_Blink

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

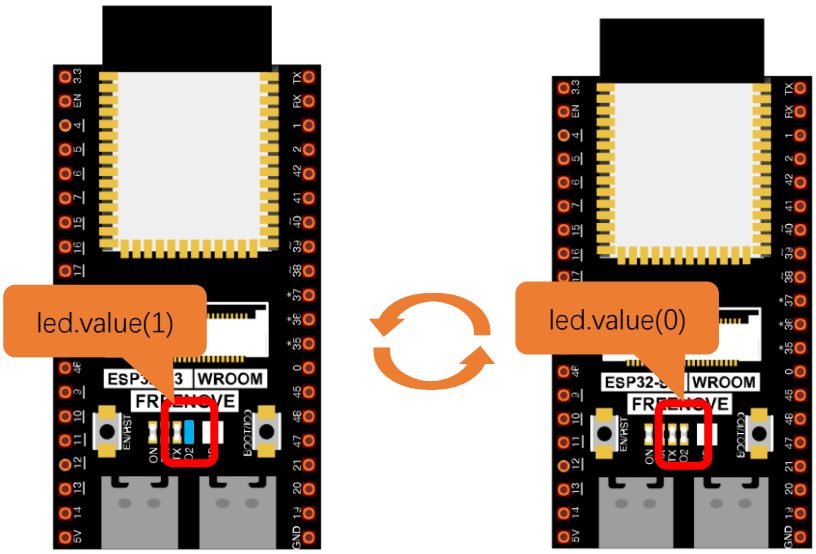


Expand folder "01.1_Blink" and double click "Blink.py" to open it. As shown in the illustration below.

Make sure ESP32-S3 has been connected with the computer with ESP32-S3 correctly. Click "Stop/Restart backend" or press the reset button, and then wait to see what interface will show up.

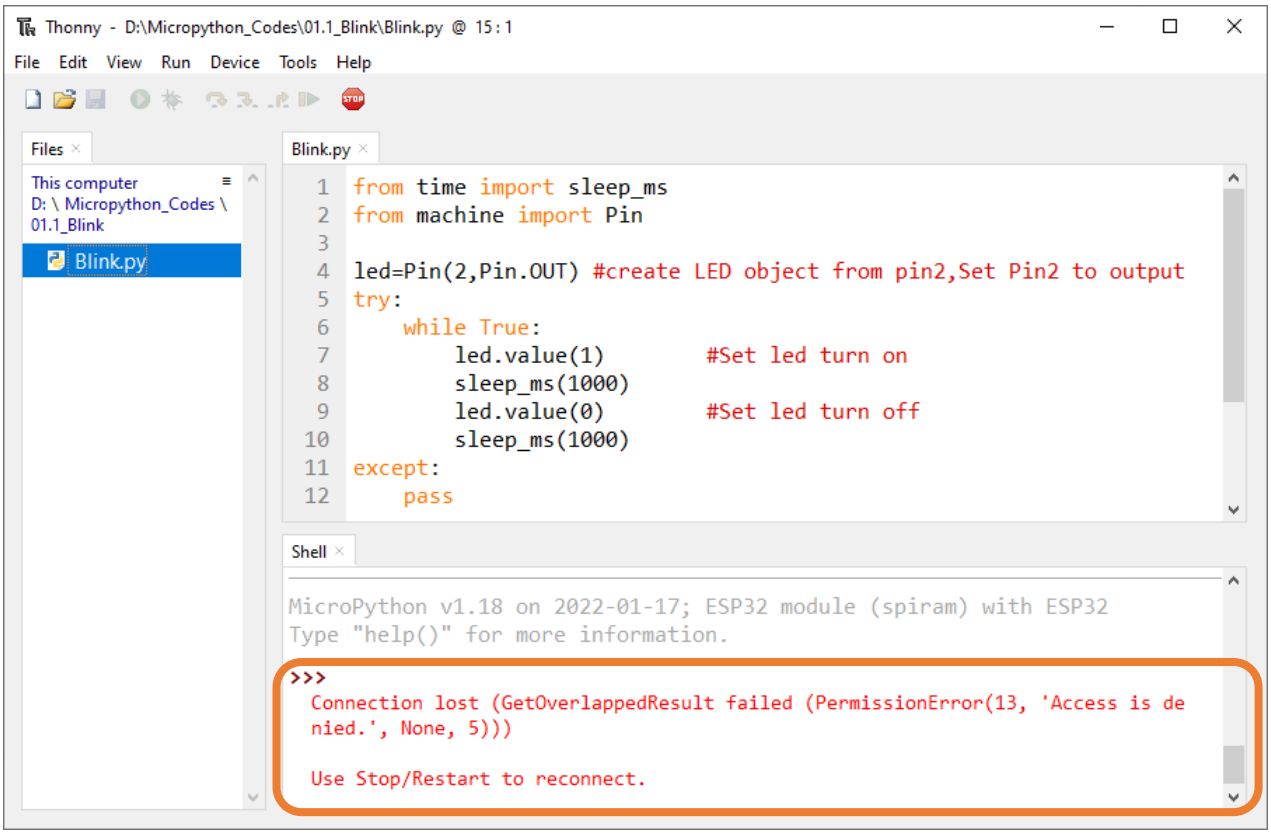


Click "Run current script" shown in the box above, the code starts to be executed and the LED in the circuit starts to blink.



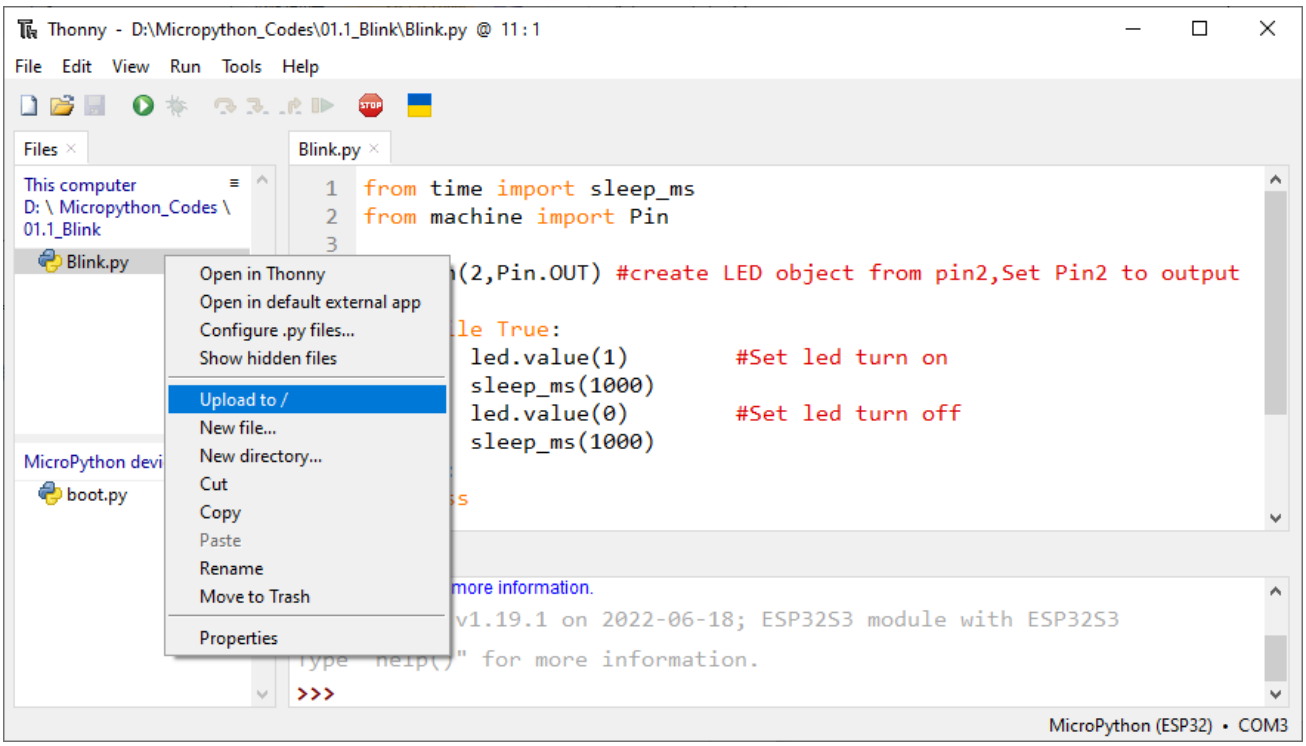
Note:

This is the code running online. If you disconnect USB cable and repower ESP32-S3 or press its reset key, LED stops blinking and the following messages will be displayed in Thonny.

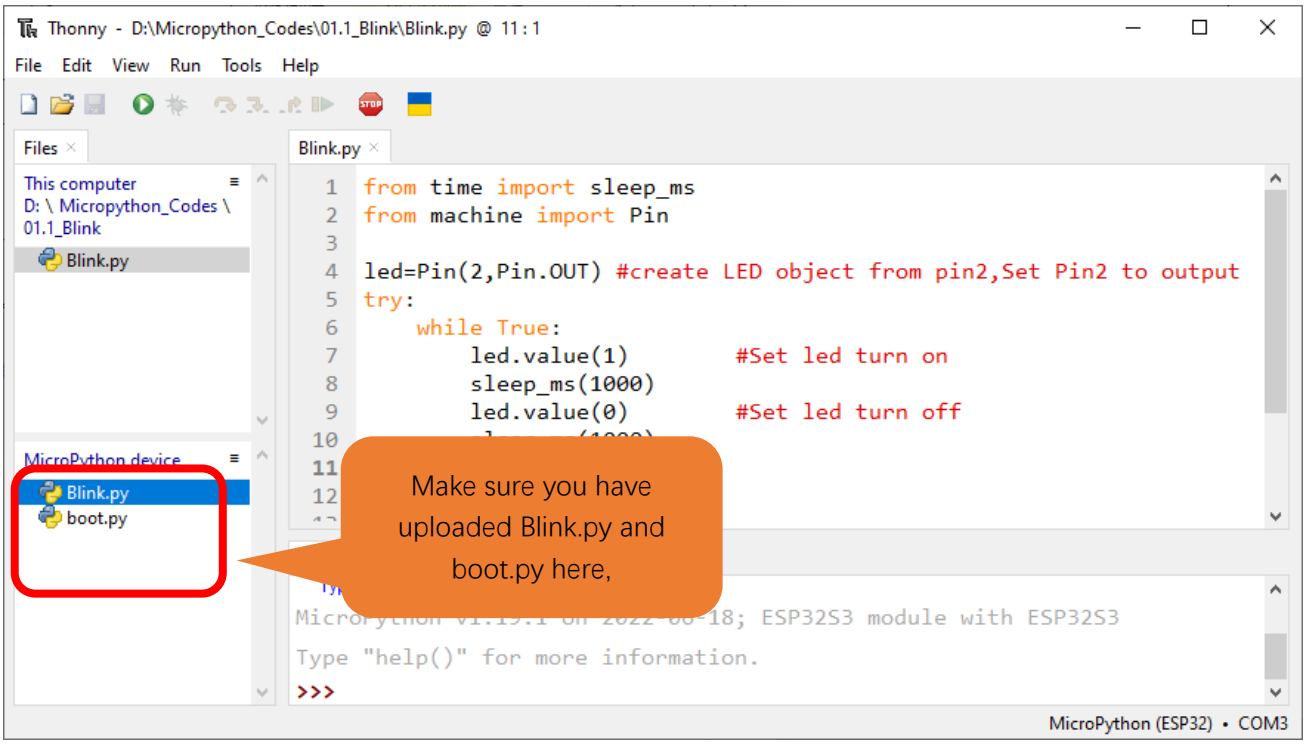


Uploading code to ESP32S3

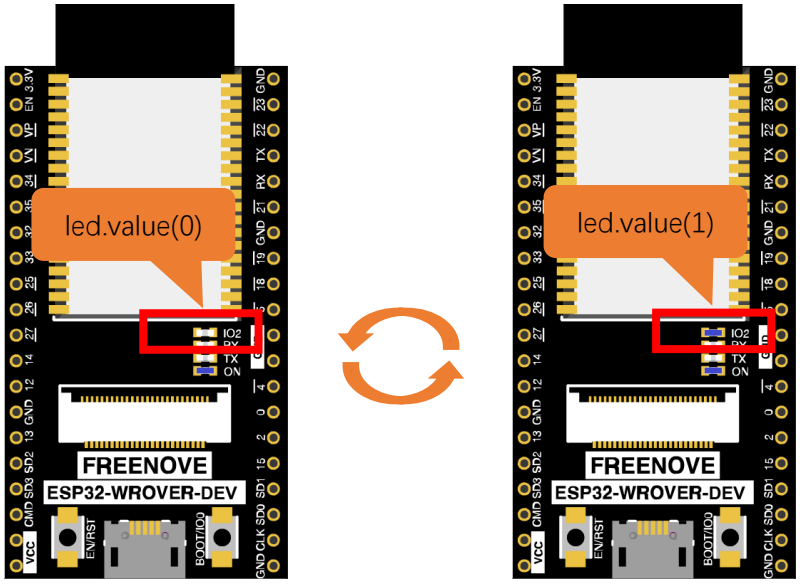
As shown in the following illustration, right-click the file Blink.py and select "Upload to /" to upload code to ESP32S3.



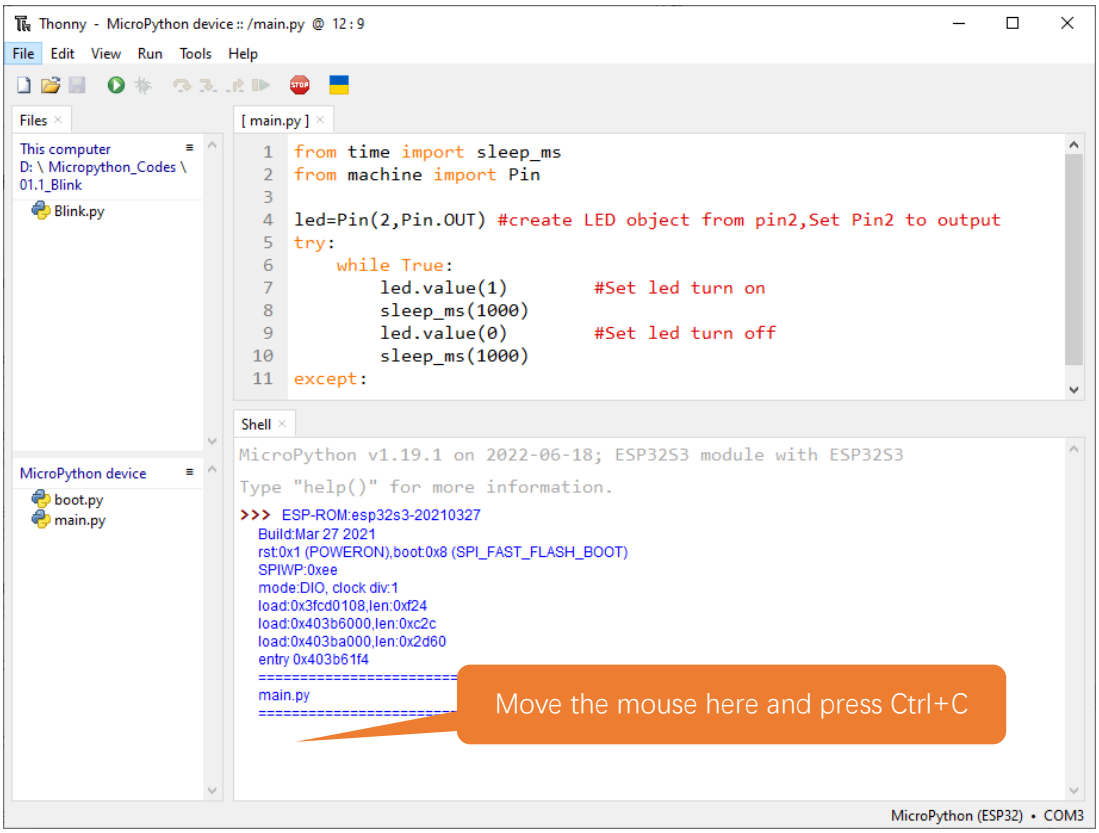
Upload boot.py in the same way.



Press the reset key of ESP32-S3 and you can see LED is ON for one second and then OFF for one second, which repeats in an endless loop.



If you want to exit the offline operation mode, you can press Ctrl+C at the same time in the shell to let the ESP32-S3 exit the offline operation mode.



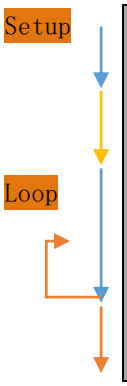
If there is no response after pressing, it is recommended to press again until exiting.

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

The following is the program code:

```
1  from time import sleep_ms
2  from machine import Pin
3
4  led=Pin(2,Pin.OUT) #create LED object from pin2,Set Pin2 to output
5  try:
6      while True:
7          led.value(1) #Set led turn on
8          sleep_ms(1000)
9          led.value(0) #Set led turn off
10         sleep_ms(1000)
11 except:
12     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  from time import sleep_ms
2  from machine import Pin
3
4  led=Pin(2,Pin.OUT) #create LED object from pin2,Set Pin2 to output
5  try:
6      while True:
…          ...
11 except:
12     pass
```

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 ESP32S3, you need to import modules corresponding to those functions: Import sleep_ms module of time module and Pin module of machine module.

```
1  from time import sleep_ms
2  from machine import Pin
```

Configure GPIO2 of ESP32-S3 to output mode and assign it to an object named "led".

```
4  led=Pin(2,Pin.OUT) #create LED object from pin2,Set Pin2 to output
```

It means that from now on, LED represents GPIO2 that is in output mode.

Set the value of LED to 1 and GPIO2 will output high level.

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

Set the value of LED to 0 and GPIO2 will output low level.

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

Execute codes in a while loop.

```
6      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 ESP32-S3 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.

```
5   try:
…       ...
11  except:
12      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 ignore comments.

```
9   #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.

```
6       while True:
7           led.value(1) #Set led turn on
8           sleep_ms(1000)
9           led.value(0) #Set led turn off
10          sleep_ms(1000)
```

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: 80000000(80MHz)、160000000(160MHz)、240000000(240MHz)

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.

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

3: interrupt on both edges

Handler: callback function

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

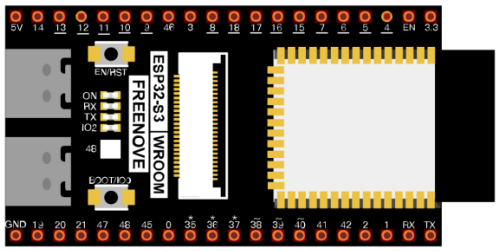
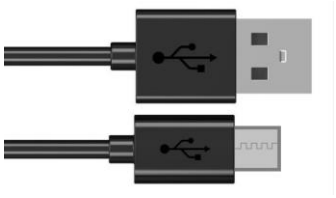
new_t: Ending time

Chapter 2 Bluetooth

This chapter mainly introduces how to make simple data transmission through Bluetooth of ESP32-S3 WROOM and mobile phones.

Project 2.1 Bluetooth Low Energy Data Passthrough

Component List

ESP32-S3 WROOM x1 	USB cable x1 

Component knowledge

ESP32S3's integrated Bluetooth function Bluetooth is a short-distance communication system, which can be divided into two types, namely Bluetooth Low Energy(BLE) and Classic Bluetooth. There are two modes for simple data transmission: master mode and slave mode.

Master mode

In this mode, works are done in the master device and it can connect with a slave device. And we can search and select slave devices nearby to connect with. When a device initiates connection request in master mode, it requires information of the other Bluetooth devices including their address and pairing passkey. After finishing pairing, it can connect with them directly.

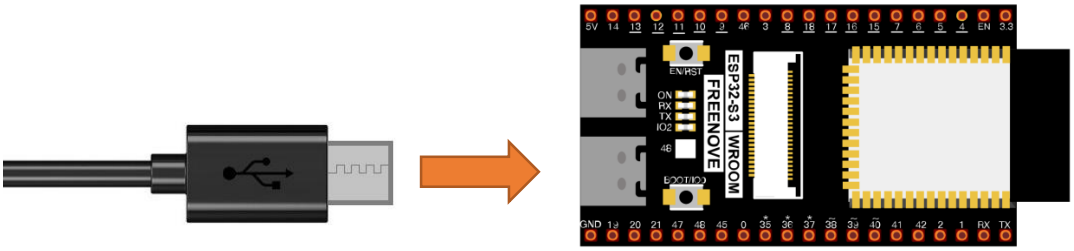
Slave mode

The Bluetooth module in slave mode can only accept connection request from a host computer, but cannot initiate a connection request. After connecting with a host device, it can send data to or receive from the host device.

Bluetooth devices can make data interaction with each other, as one is in master mode and the other in slave mode. When they are making data interaction, the Bluetooth device in master mode searches and selects devices nearby to connect to. When establishing connection, they can exchange data. When mobile phones exchange data with ESP32S3, they are usually in master mode and ESP32-S3 in slave mode.

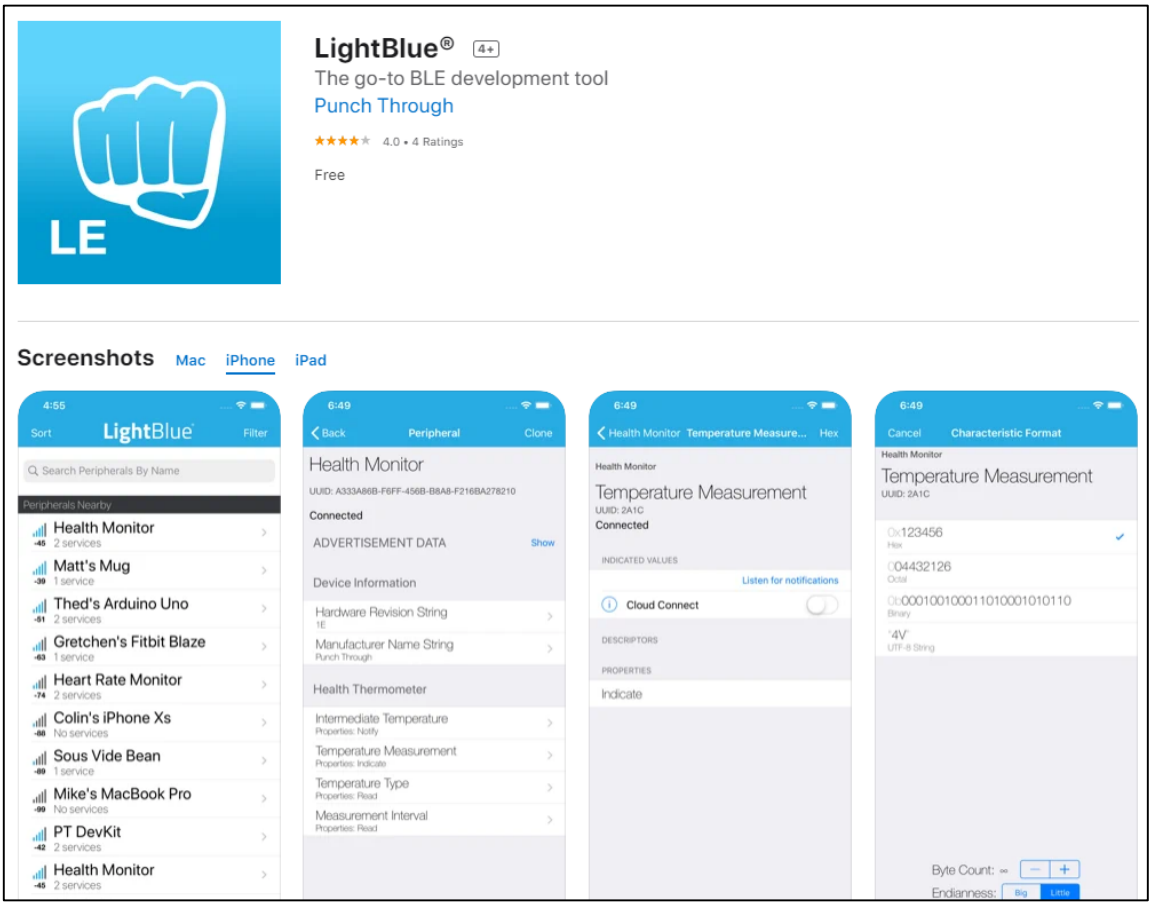
Circuit

Connect Freenove ESP32-S3 to the computer using the USB cable.



Lightblue

If you can't install Serial Bluetooth 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.

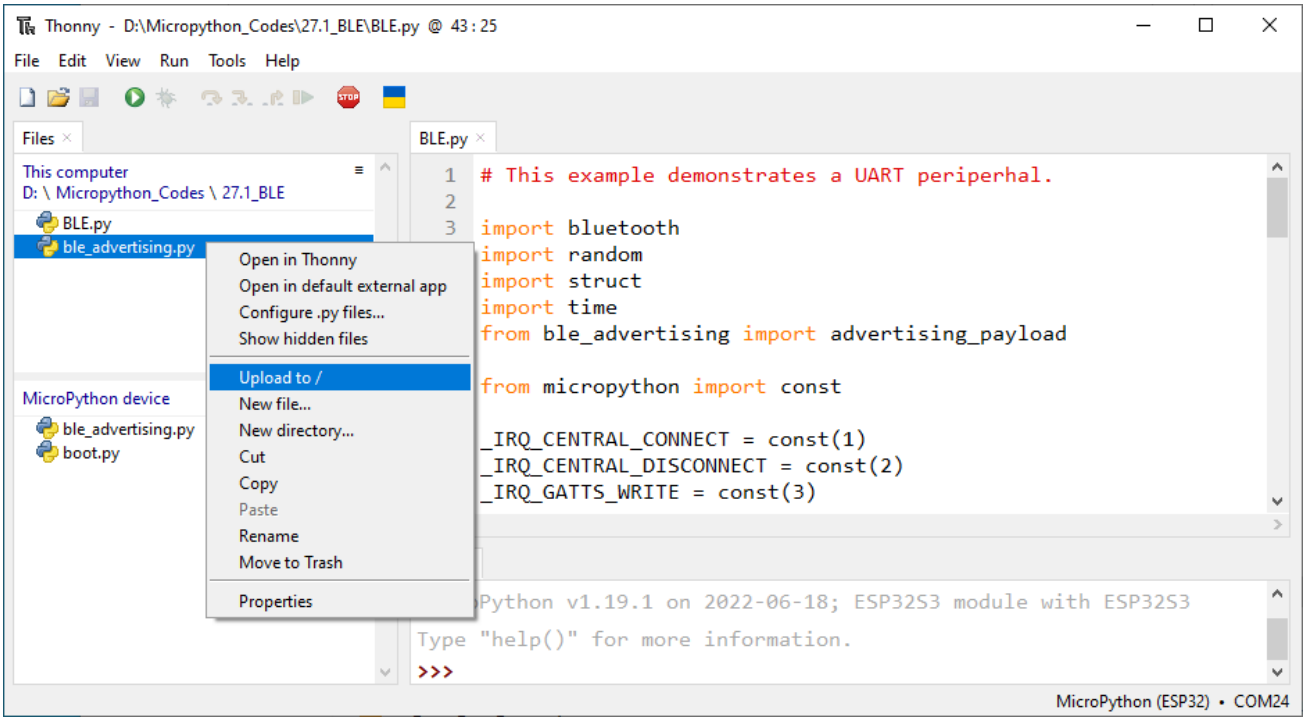


Code

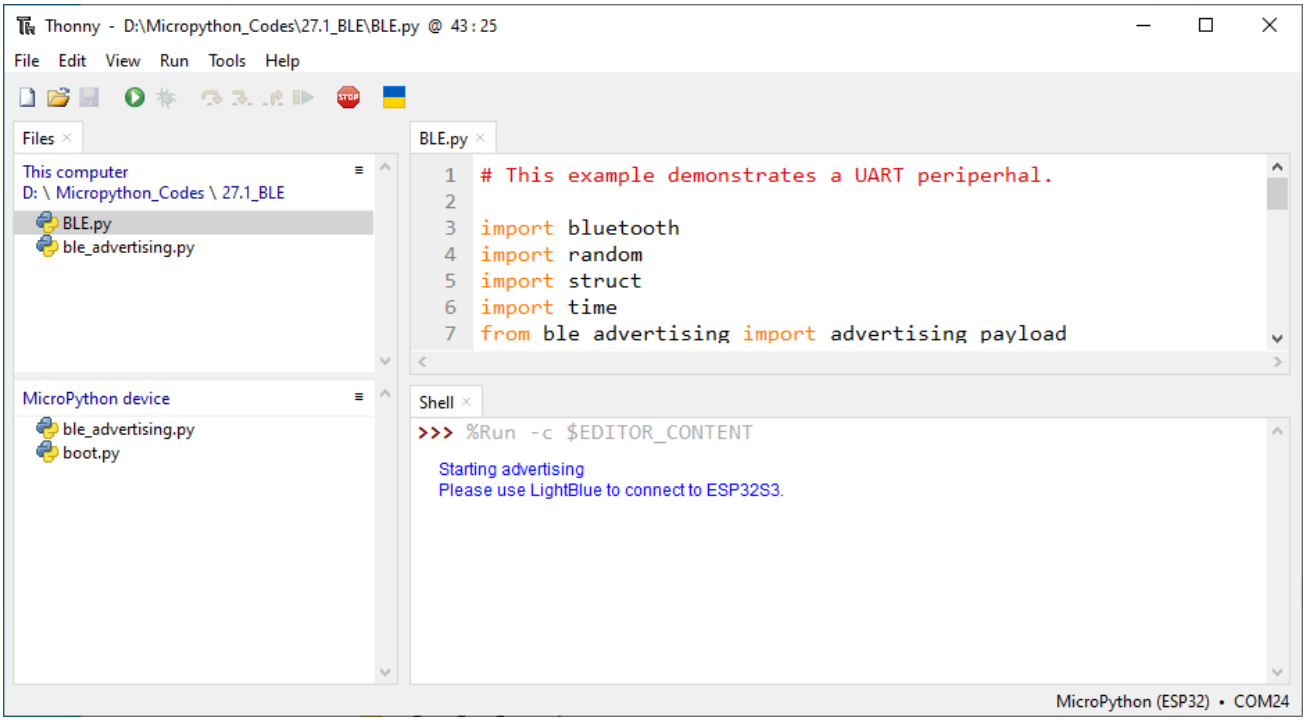
Move the program folder "**Freenove_ESP32_S3_WROVER_Board/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "02.1_BLE". Select "ble_advertising.py", right click your mouse to select "Upload to /", wait for "ble_advertising.py" to be uploaded to ESP32-S3 and then double click "BLE.py".

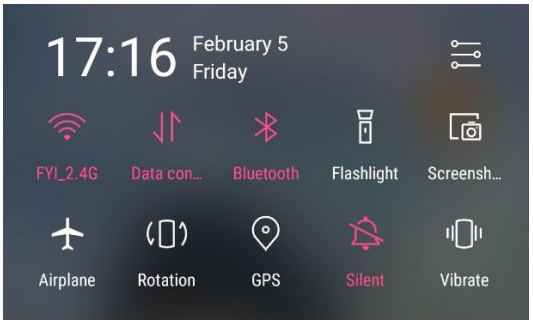
02.1_BLE



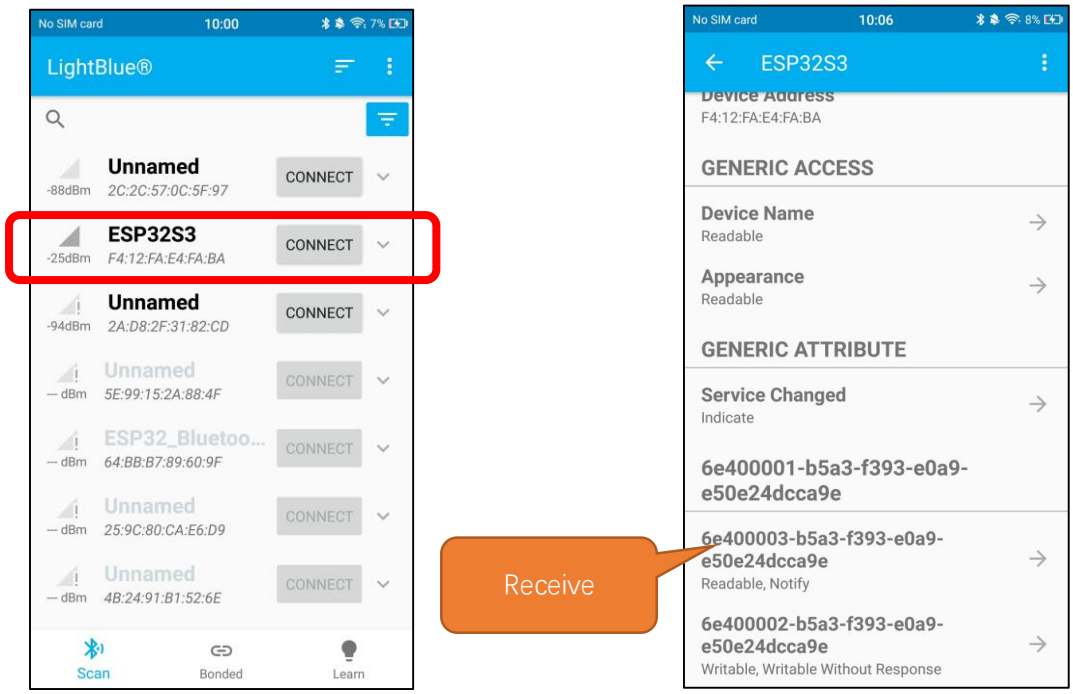
Click run for BLE.py.



Turn ON Bluetooth on your phone, and open the Lightblue APP.

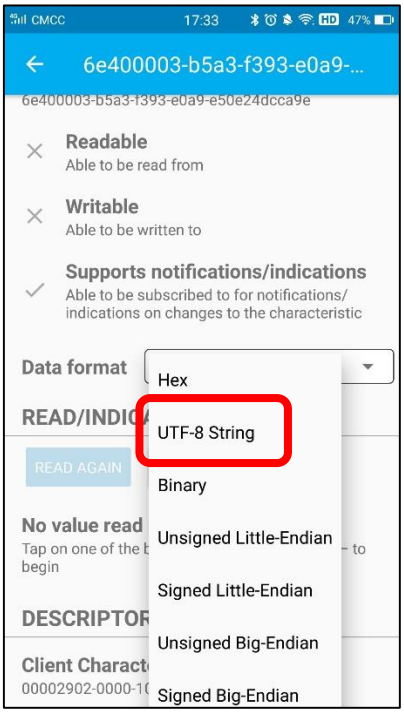


In the Scan page, swipe down to refresh the name of Bluetooth that the phone searches for. Click ESP32S3.

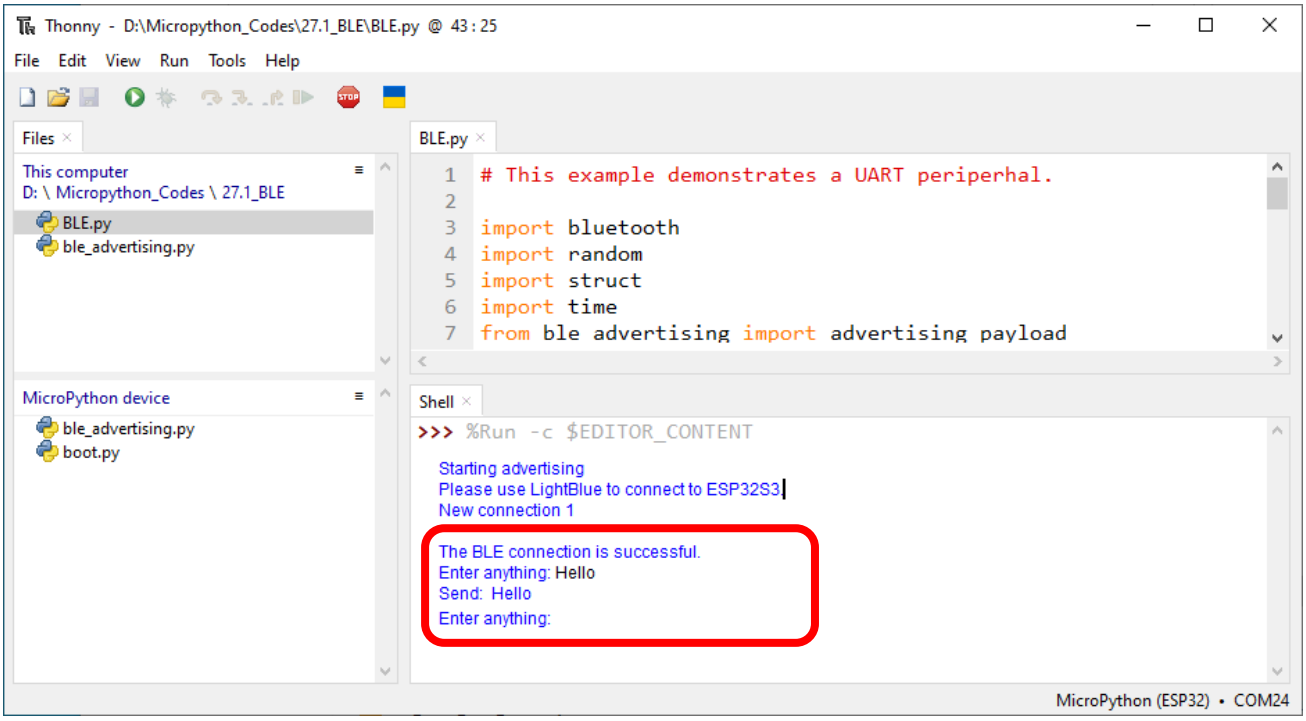


After Bluetooth is connect successfully, Shell will printer the information.

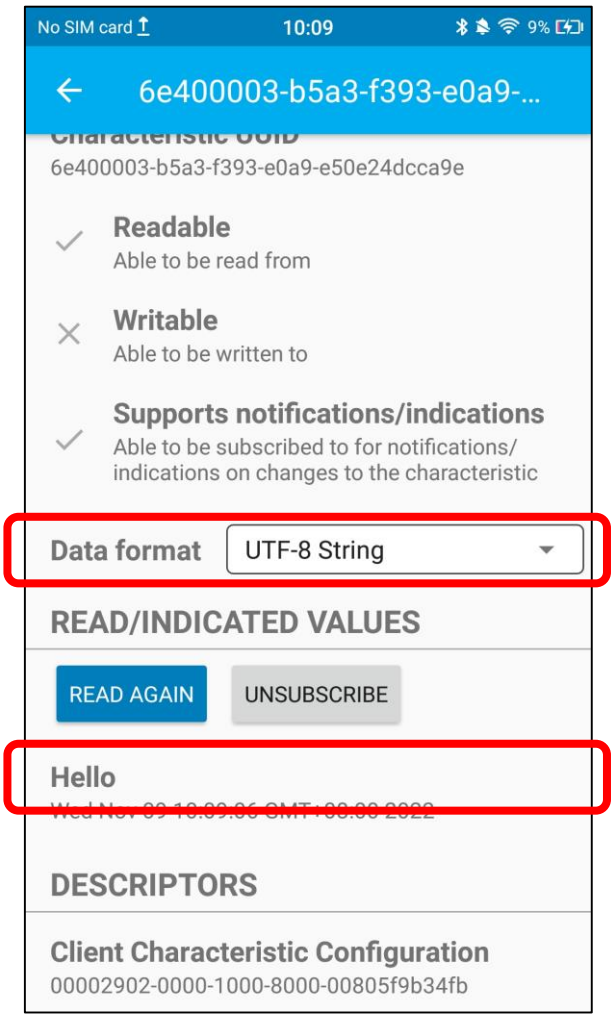
Click "Receive". Select the appropriate Data format in the box to the right of Data Format. For example, HEX for hexadecimal, utf-string for character, Binary for Binary, etc. Then click SUBSCRIBE.



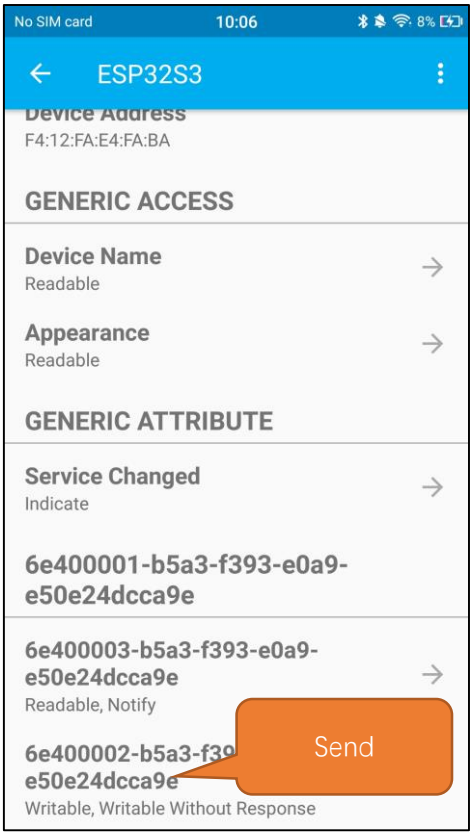
You can type "Hello" in Shell and press "Enter" to send.

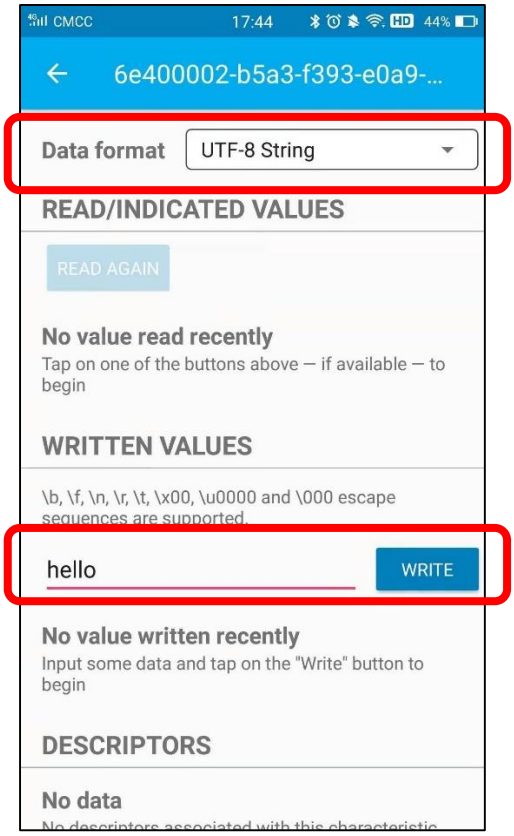


And then you can see the mobile Bluetooth has received the message.

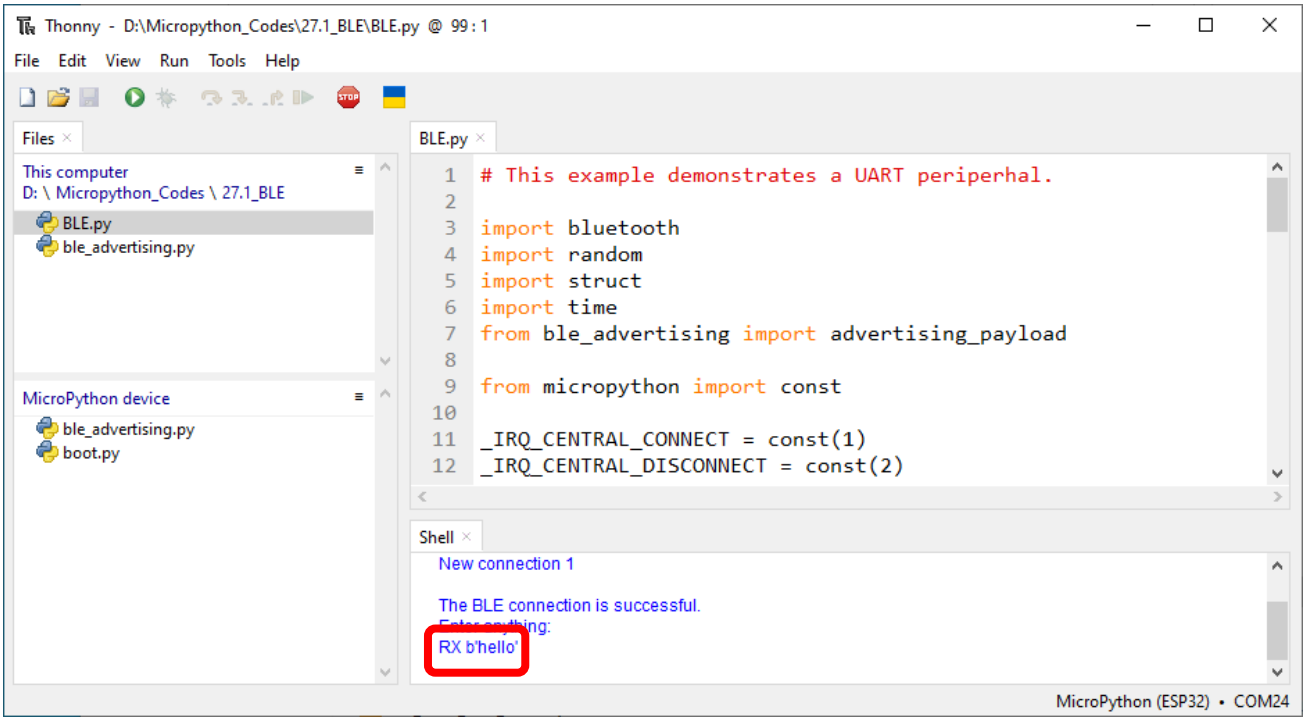


Similarly, you can select "Send" on your phone. Set Data format, and then enter anything in the sending box and click Write to send.





You can check the message from Bluetooth in "Shell".



And now data can be transferred between your mobile phone and computer via ESP32S3.
The following is the program code:

```
import bluetooth
import random
import struct
import time
from ble_advertising import advertising_payload
from micropython import const

_IRQ_CENTRAL_CONNECT = const(1)
_IRQ_CENTRAL_DISCONNECT = const(2)
_IRQ_GATTS_WRITE = const(3)
_FLAG_READ = const(0x0002)
_FLAG_WRITE_NO_RESPONSE = const(0x0004)
_FLAG_WRITE = const(0x0008)
_FLAG_NOTIFY = const(0x0010)

_UART_UUID = bluetooth.UUID("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")
_UART_TX = (
    bluetooth.UUID("6E400003-B5A3-F393-E0A9-E50E24DCCA9E"),
    _FLAG_READ | _FLAG_NOTIFY,
)
_UART_RX = (
    bluetooth.UUID("6E400002-B5A3-F393-E0A9-E50E24DCCA9E"),
    _FLAG_WRITE | _FLAG_WRITE_NO_RESPONSE,
)
_UART_SERVICE = (
    _UART_UUID,
    (_UART_TX, _UART_RX),
)
class BLESimplePeripheral:
    def __init__(self, ble, name="ESP32S3"):
        self._ble = ble
        self._ble.active(True)
        self._ble.irq(self._irq)
        ((self._handle_tx, self._handle_rx),) =
self._ble.gatts_register_services((_UART_SERVICE,))
        self._connections = set()
        self._write_callback = None
        self._payload = advertising_payload(name=name, services=[_UART_UUID])
        self._advertise()
    def _irq(self, event, data):
        # Track connections so we can send notifications.
        if event == _IRQ_CENTRAL_CONNECT:
```

```
            conn_handle, _, _ = data
            print("New connection", conn_handle)
            print("\nThe BLE connection is successful.")
            self._connections.add(conn_handle)
        elif event == _IRQ_CENTRAL_DISCONNECT:
            conn_handle, _, _ = data
            print("Disconnected", conn_handle)
            self._connections.remove(conn_handle)
            # Start advertising again to allow a new connection.
            self._advertise()
        elif event == _IRQ_GATTS_WRITE:
            conn_handle, value_handle = data
            value = self._ble.gatts_read(value_handle)
            if value_handle == self._handle_rx and self._write_callback:
                self._write_callback(value)
    def send(self, data):
        for conn_handle in self._connections:
            self._ble.gatts_notify(conn_handle, self._handle_tx, data)
    def is_connected(self):
        return len(self._connections) > 0
    def _advertise(self, interval_us=500000):
        print("Starting advertising")
        self._ble.gap_advertise(interval_us, adv_data=self._payload)
    def on_write(self, callback):
        self._write_callback = callback
def demo():
    ble = bluetooth.BLE()
    p = BLESimplePeripheral(ble)
    def on_rx(rx_data):
        print("RX", rx_data)
    p.on_write(on_rx)
    print("Please use LightBlue to connect to ESP32S3.")
    while True:
        if p.is_connected():
            # Short burst of queued notifications.
            tx_data = input("Enter anything: ")
            print("Send: ", tx_data)
            p.send(tx_data)
if __name__ == "__main__":
    demo()
```

Define the specified UUID number for BLE vendor.

```
_UART_UUID = bluetooth.UUID("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")
_UART_TX = (
    bluetooth.UUID("6E400003-B5A3-F393-E0A9-E50E24DCCA9E"),
```

```
    _FLAG_READ | _FLAG_NOTIFY,
)
_UART_RX = (
    bluetooth.UUID("6E400002-B5A3-F393-E0A9-E50E24DCCA9E"),
    _FLAG_WRITE | _FLAG_WRITE_NO_RESPONSE,
)
```

Write an _irq function to manage BLE interrupt events.

```
    def _irq(self, event, data):
        # Track connections so we can send notifications.
        if event == _IRQ_CENTRAL_CONNECT:
            conn_handle, _, _ = data
            print("New connection", conn_handle)
            print("\nThe BLE connection is successful.")
            self._connections.add(conn_handle)
        elif event == _IRQ_CENTRAL_DISCONNECT:
            conn_handle, _, _ = data
            print("Disconnected", conn_handle)
            self._connections.remove(conn_handle)
            # Start advertising again to allow a new connection.
            self._advertise()
        elif event == _IRQ_GATTS_WRITE:
            conn_handle, value_handle = data
            value = self._ble.gatts_read(value_handle)
            if value_handle == self._handle_rx and self._write_callback:
                self._write_callback(value)
```

Initialize the BLE function and name it.

```
def __init__(self, ble, name="ESP32S3"):
```

When the mobile phone send data to ESP32-S3 via BLE Bluetooth, it will print them out with serial port;
When the serial port of ESP32-S3 receive data, it will send them to mobile via BLE Bluetooth.

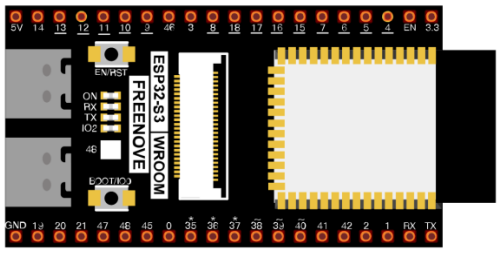
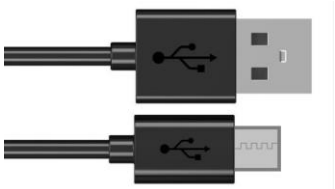
```
  def demo():
    ble = bluetooth.BLE()
    p = BLESimplePeripheral(ble)
    def on_rx(rx_data):
        print("RX", rx_data)
    p.on_write(on_rx)
    print("Please use LightBlue to connect to ESP32S3.")
    while True:
        if p.is_connected():
            # Short burst of queued notifications.
            tx_data = input("Enter anything: ")
            print("Send: ", tx_data)
            p.send(tx_data)
    lastMsg = now;
  }
```

Chapter 3 WiFi Working Modes

In this chapter, we'll focus on the WiFi infrastructure for ESP32-S3 WROOM.
ESP32-S3 WROOM has 3 different WiFi operating modes: station mode, AP mode and AP+station mode. All WiFi programming projects must be configured with WiFi operating mode before using WiFi, otherwise WiFi cannot be used.

Project 3.1 Station mode

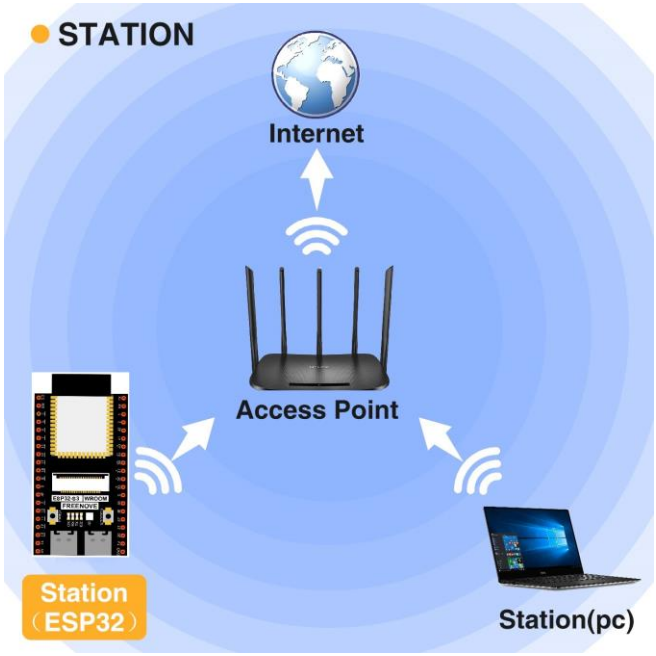
Component List

ESP32-S3 WROOM x1 	USB cable x1 

Component knowledge

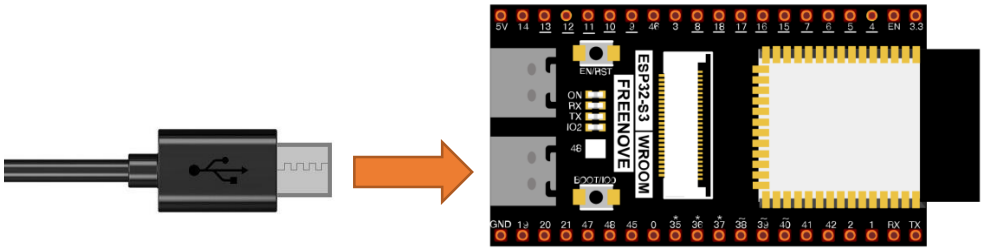
Station mode

When ESP32-S3 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 ESP32-S3 wants to communicate with the PC, it needs to be connected to the router.



Circuit

Connect Freenove ESP32-S3 to the computer using the USB cable.

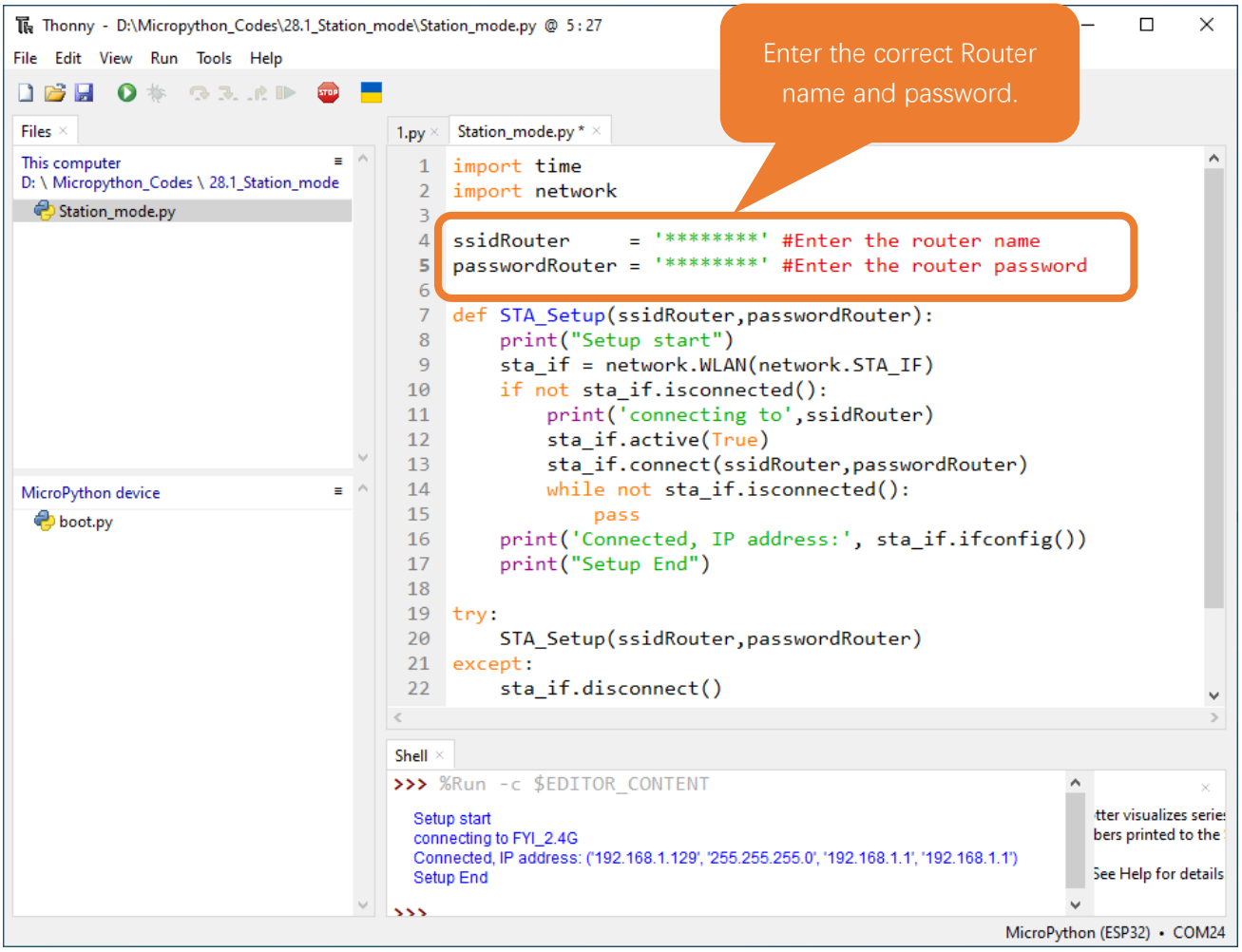


Code

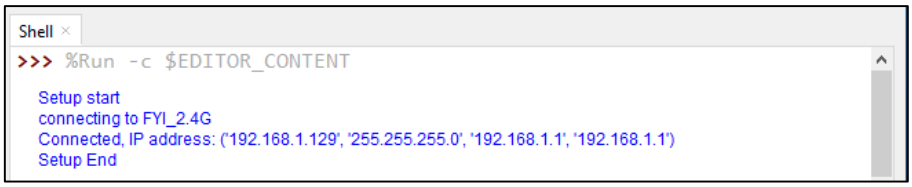
Move the program folder "**Freenove_ESP32_S3_WROVER_Board/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "03.1_Station_mode" and double click "Station_mode.py".

03.1_Station_mode



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 ESP32S3, wait for ESP32-S3 to connect to your router and print the IP address assigned by the router to ESP32-S3 in "Shell".



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.

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

Set ESP32-S3 in Station mode.

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

Activate ESP32-S3'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 ESP32-S3 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 ESP32-S3 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 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 ESP32-S3 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: WiFiname

password: WiFipassword

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

Project 3.2 AP mode

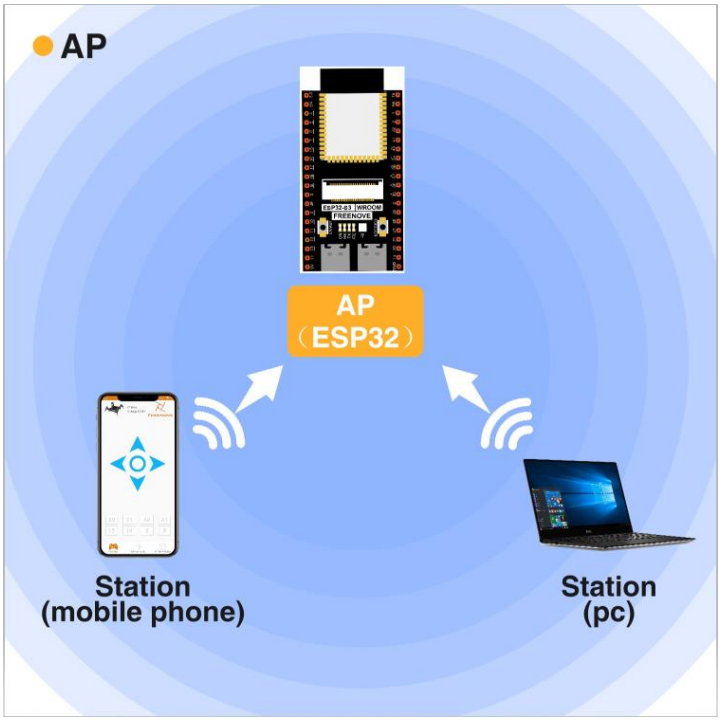
Component List & Circuit

Component List & Circuit are the same as in Project 3.1.

Component knowledge

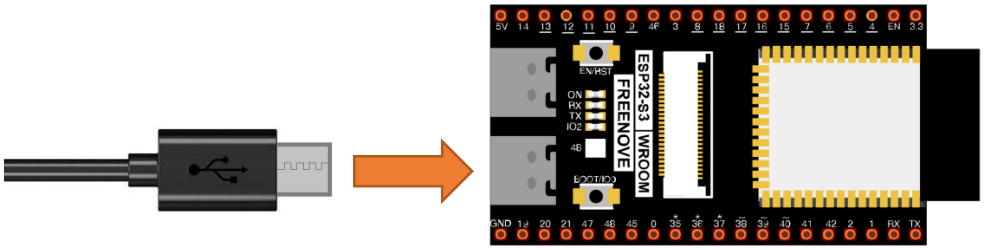
AP mode

When ESP32-S3 selects AP mode, it creates a hotspot network that is separate from the Internet and waits for other WiFi devices to connect. As shown in the figure below, ESP32-S3 is used as a hotspot. If a mobile phone or PC wants to communicate with ESP32S3, it must be connected to the hotspot of ESP32S3. Only after a connection is established with ESP32-S3 can they communicate.



Circuit

Connect Freenove ESP32-S3 to the computer using the USB cable.

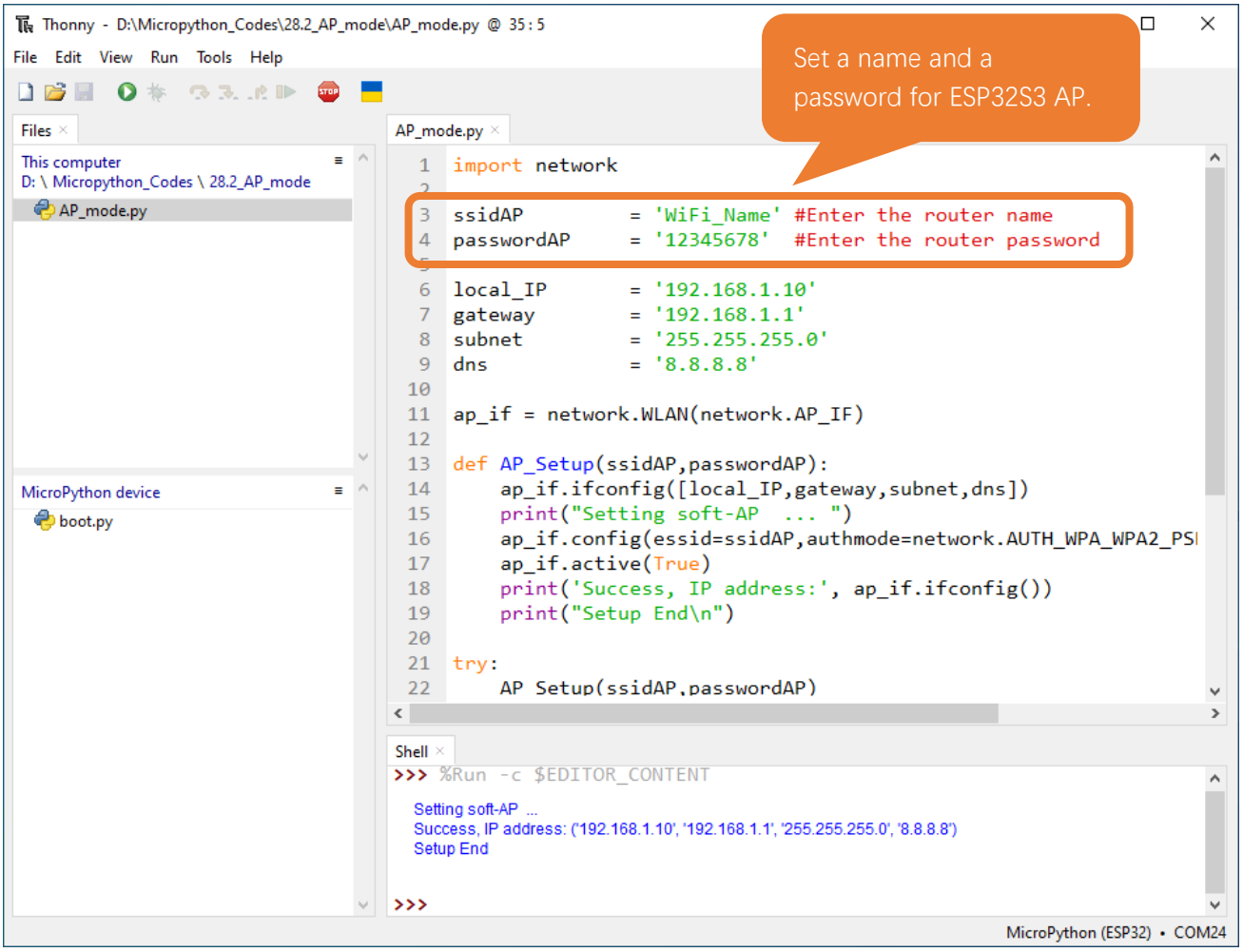


Code

Move the program folder "**Freenove_ESP32_S3_WROVER_Board/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

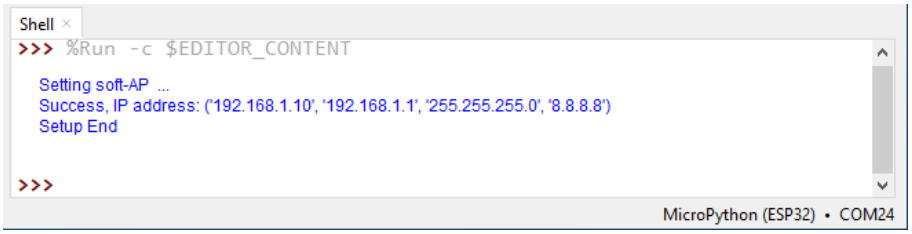
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "03.2_AP_mode". and double click "AP_mode.py".

03.2_AP_mode

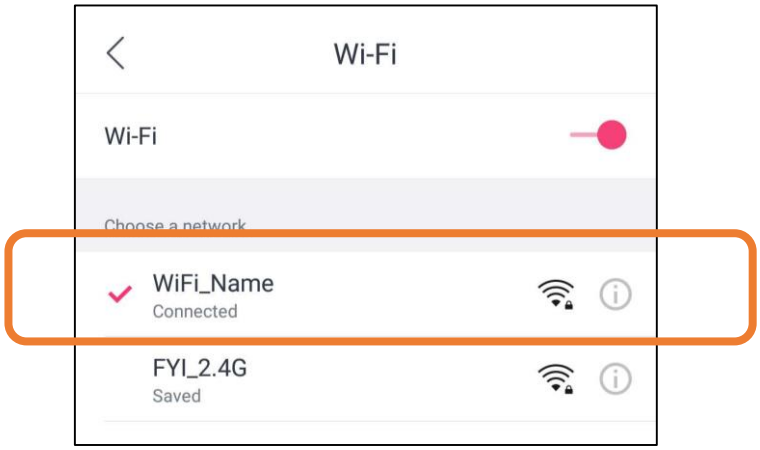


Before the Code runs, you can make any changes to the AP name and password for ESP32-S3 in the box as shown in the illustration above. Of course, you can leave it alone by default.

Click "Run current script", open the AP function of ESP32-S3 and print the access point information.



Turn on the WiFi scanning function of your phone, and you can see the ssid_AP on ESP32S3, 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:

```
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.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.active(True)
    print('Success, IP address:', ap_if.ifconfig())
    print("Setup End\n")

try:
    AP_Setup(ssidAP,passwordAP)
except:
    ap_if.disconnect()
```

Import network module.

```
import network
```

Enter correct AP name and password.

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

Set ESP32-S3 in AP mode.

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

Configure IP address, gateway and subnet mask for ESP32S3.

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

Turn on an AP in ESP32S3, whose name is set by ssid_AP and password is set by password_AP.

```
16     ap_if.config(essid=ssidAP,authmode=network.AUTH_WPA_WPA2_PSK, 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: WiFiname

password: WiFipassword

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: WiFichannel

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: IPaddress

subnet_mask: subnet mask

gateway: gateway

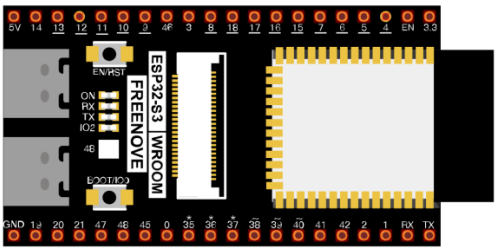
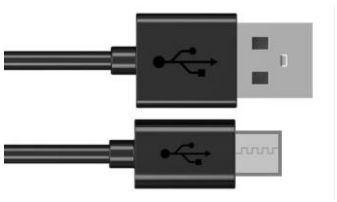
DNS_server: DNSserver

disconnect(): Disconnect from the currently connected wireless network

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

Project 3.3 AP+Station mode

Component List

ESP32-S3 WROOM x1 	USB cable x1 

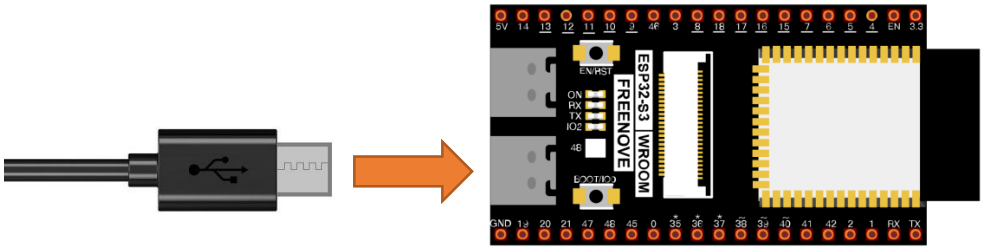
Component knowledge

AP+Station mode

In addition to AP mode and station mode, ESP32-S3 can also use AP mode and station mode at the same time. This mode contains the functions of the previous two modes. Turn on ESP32S3's station mode, connect it to the router network, and it can communicate with the Internet via the router. At the same time, turn on its AP mode to create a hotspot network. Other WiFi devices can choose to connect to the router network or the hotspot network to communicate with ESP32S3.

Circuit

Connect Freenove ESP32-S3 to the computer using the USB cable.

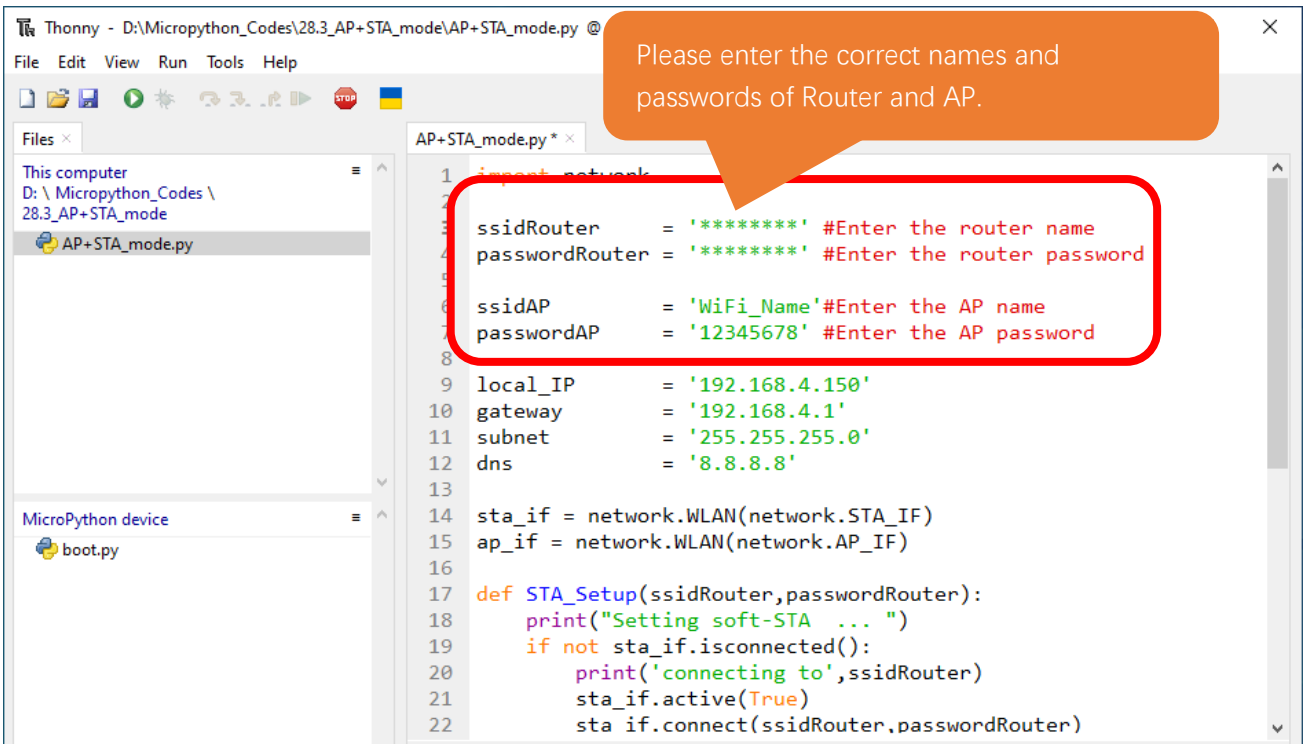


Code

Move the program folder "**Freenove_ESP32_S3_WROVER_Board/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "03.3_AP+STA_mode"and double click "AP+STA_mode.py".

03.3_AP+STA_mode



It is analogous to Project 3.1 and Project 3.2. 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:

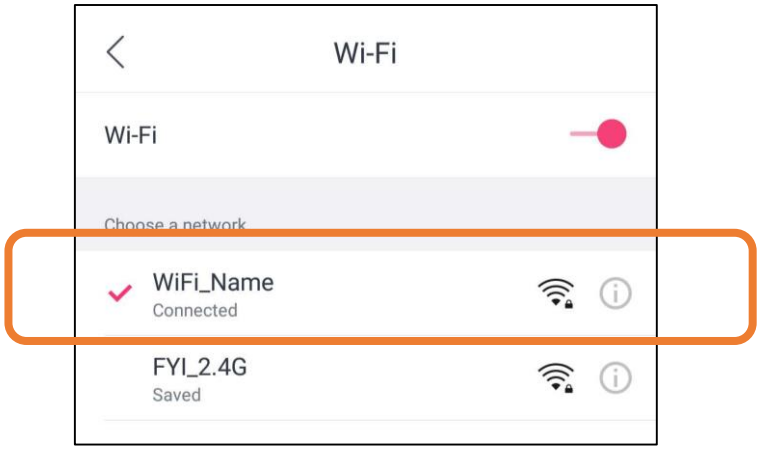
```
Shell
>>> %Run -c $EDITOR_CONTENT

    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  ...
    connecting to FYI_2.4G
    Connected, IP address: ('192.168.1.129', '255.255.255.0', '192.168.1.1', '192.168.1.1')
    Setup End

>>>
                                                    MicroPython (ESP32) • COM24
```

Turn on the WiFi scanning function of your phone, and you can see the ssidAP on ESP32-S3.



The following is the program code:

```
import network

ssidRouter     = '********' #Enter the router name
passwordRouter = '********' #Enter the router password

ssidAP         = 'WiFi_Name'#Enter the AP name
passwordAP     = '12345678' #Enter the AP password

local_IP       = '192.168.4.150'
gateway        = '192.168.4.1'
subnet         = '255.255.255.0'
dns            = '8.8.8.8'

sta_if = network.WLAN(network.STA_IF)
ap_if = network.WLAN(network.AP_IF)

def STA_Setup(ssidRouter,passwordRouter):
    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  ... ")
31      ap_if.config(essid=ssidAP,authmode=network.AUTH_WPA_WPA2_PSK, 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.idsconnect()
```

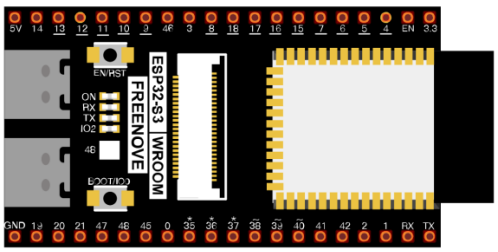
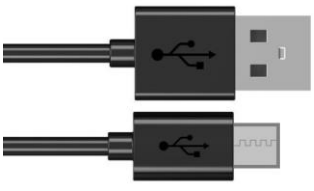
Chapter 4 TCP/IP

In this chapter, we will introduce how ESP32-S3 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 4.1 As Client

In this section, ESP32-S3 is used as Client to connect Server on the same LAN and communicate with it.

Component List

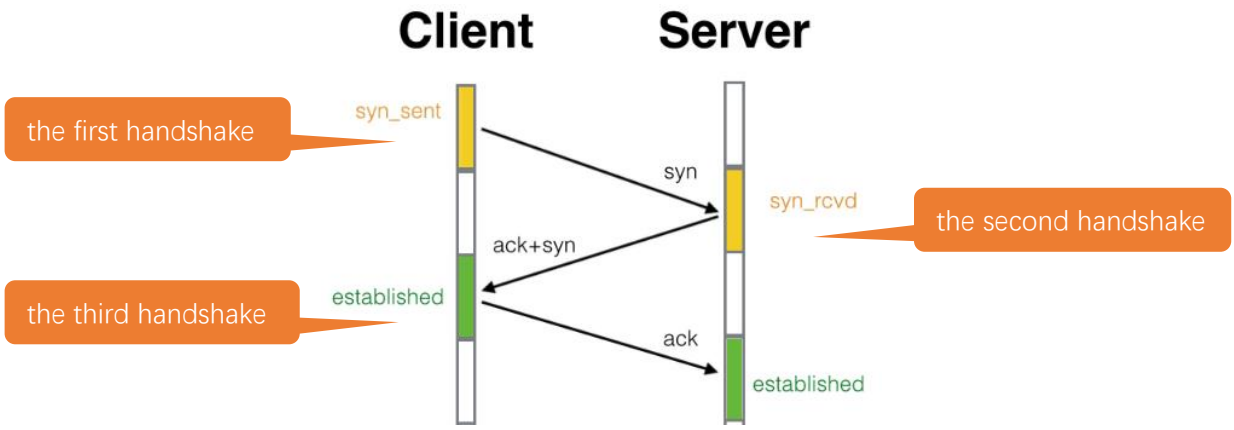
ESP32-S3 WROOM x1 	USB cable x1 

Component knowledge

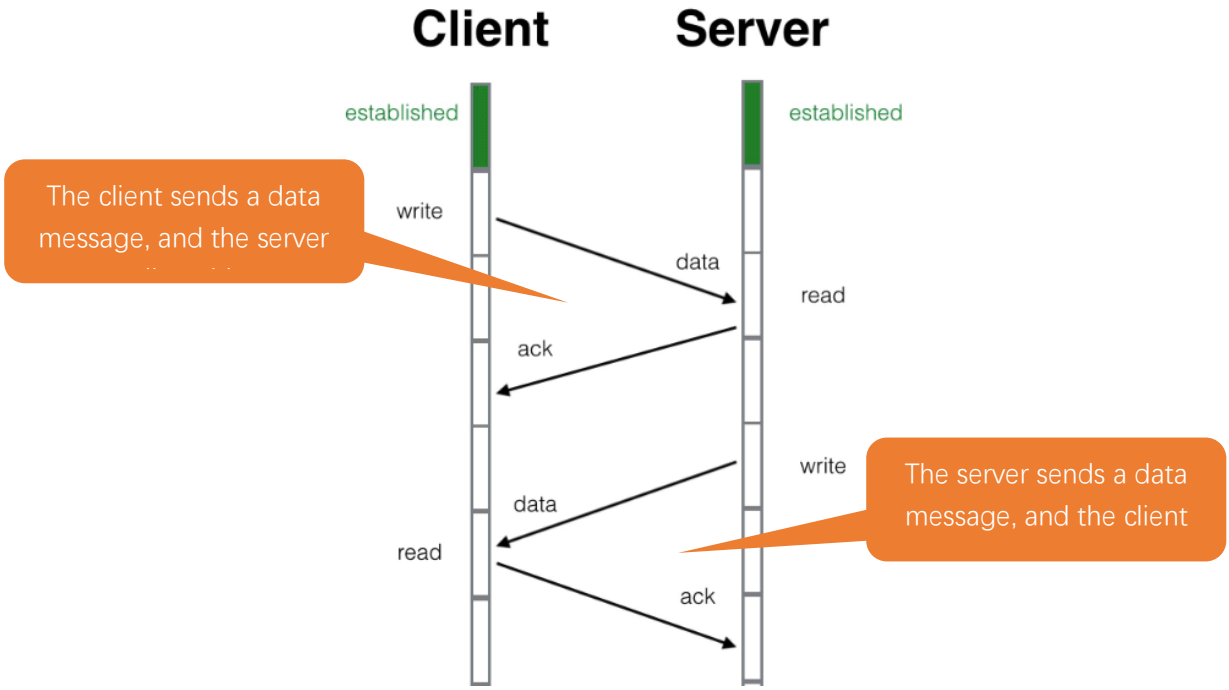
TCP connection

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-times handshake" is required.

Three-times 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-times 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

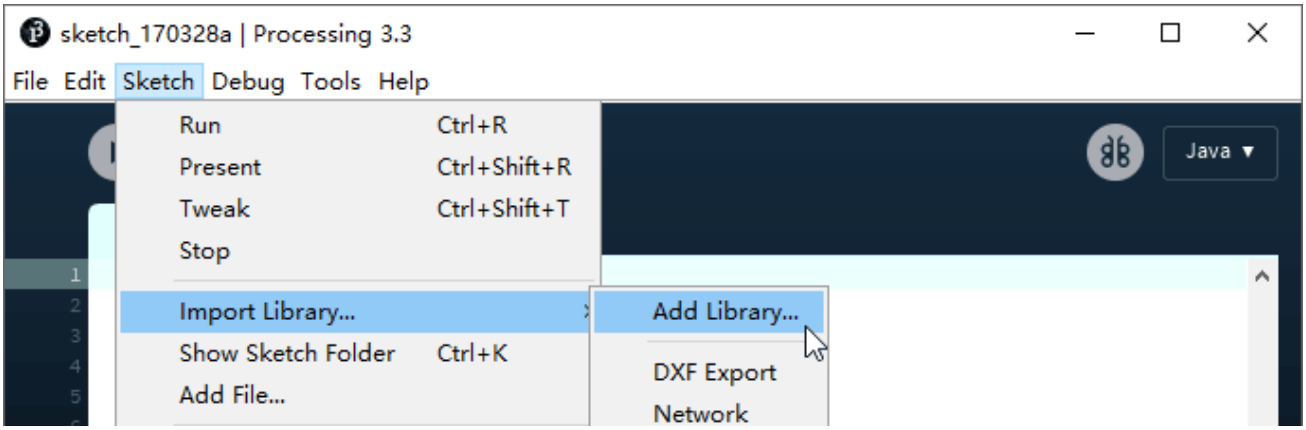
In this tutorial, we use Processing to build a simple TCP/IP communication platform.

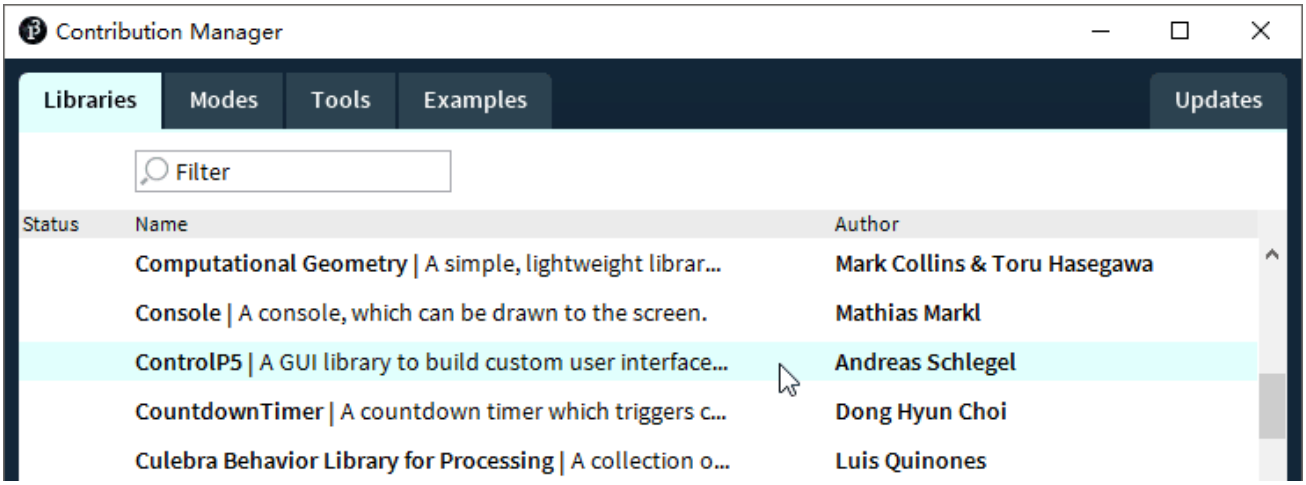
If you've not installed Processing, you can download it by clicking https://processing.org/download/. You can choose an appropriate version to download according to your PC system.

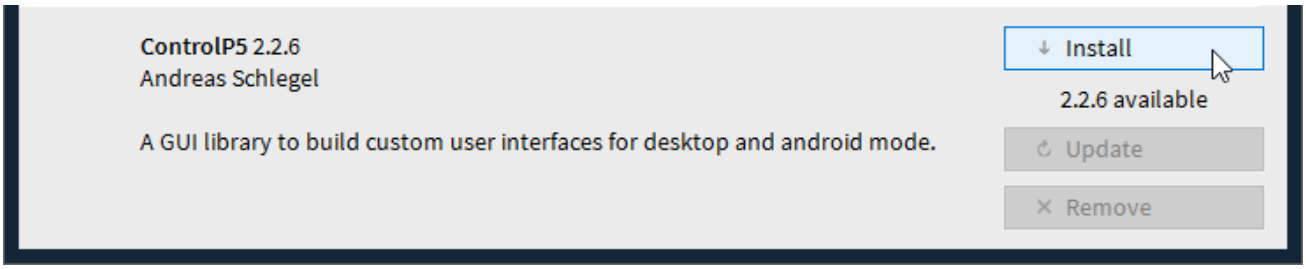
Unzip the downloaded file to your computer. Click "processing.exe" as the figure below to run this software.

Use Server mode for communication

Install ControlP5.

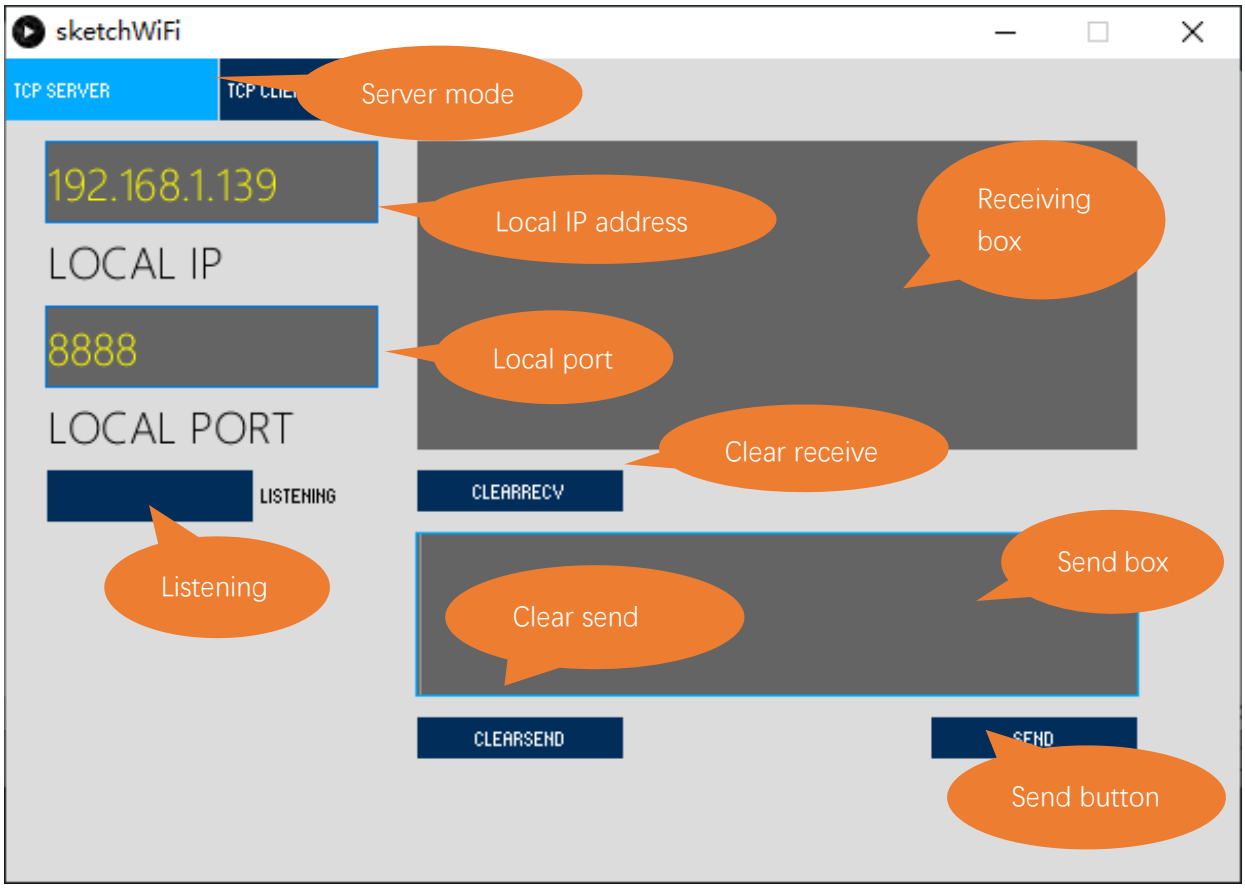






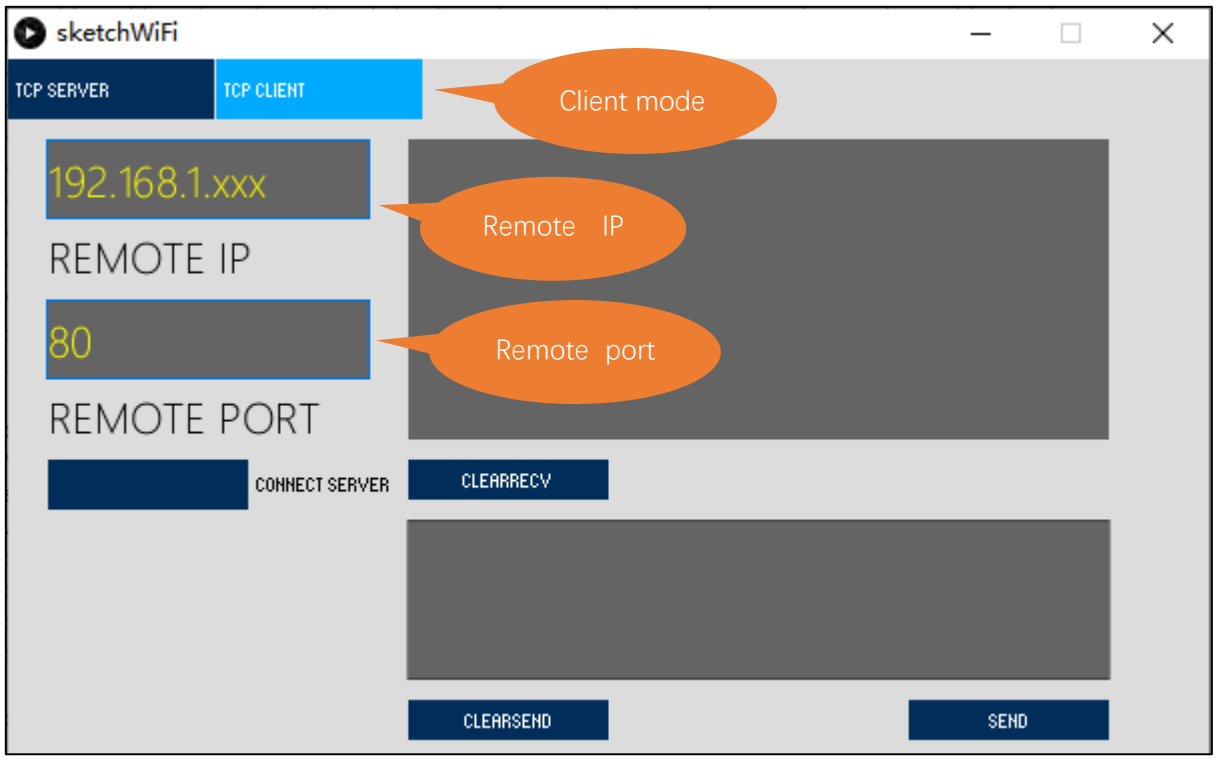
Open the "**Freenove_ESP32_S3_WROVER_Board\Sketches\Sketches\Sketch_03.1_WiFiClient\sketchWiFi\sketchWiFi.pde**", and click "Run".

The new pop-up interface is as follows. If ESP32-S3 is used as client, select TCP SERVER mode for sketchWiFi.



When sketchWiFi selects TCP SERVER mode, ESP32-S3 Sketch needs to be changed according to sketchWiFi's displaying of LOCAL IP or LOCAL PORT.

If ESP32-S3 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, 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.

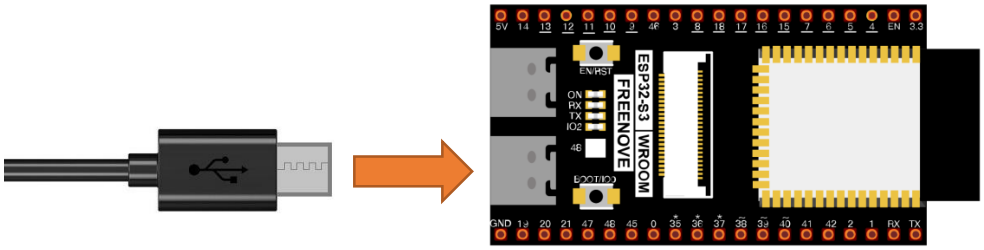
clear receive: clear out the content in the receiving text box

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.

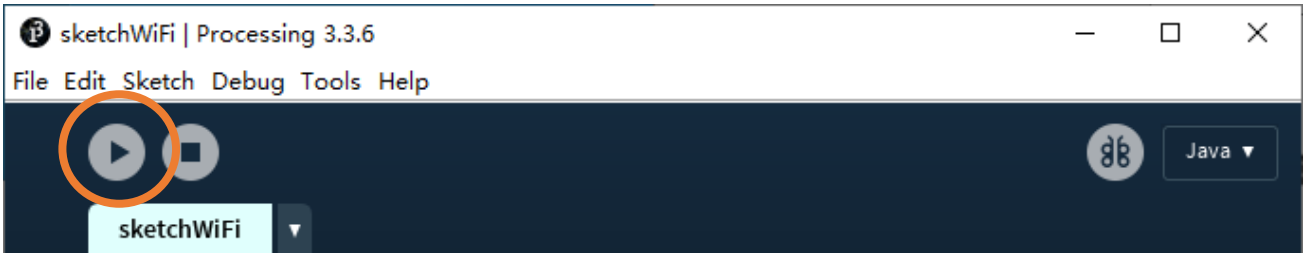
Circuit

Connect Freenove ESP32-S3 to the computer using the USB cable.

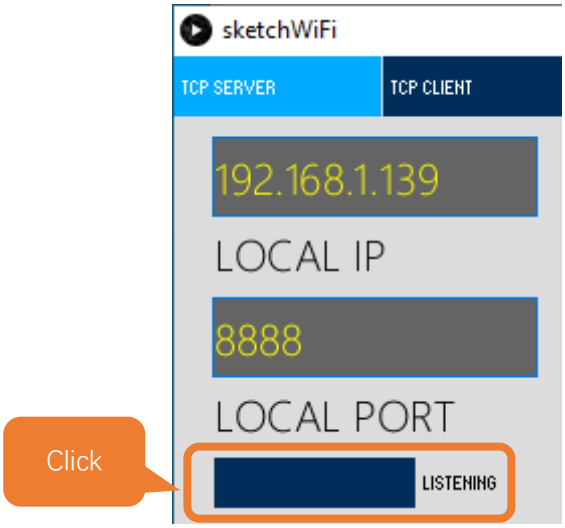


Code

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"。

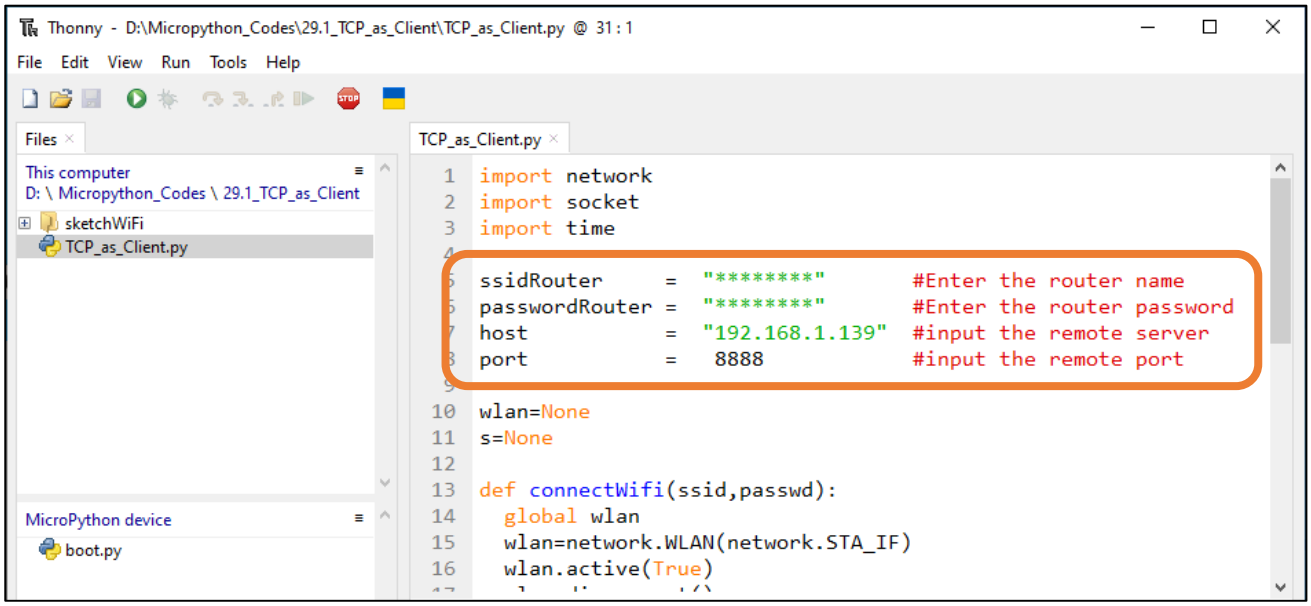


Move the program folder "**Freenove_ESP32_S3_WROVER_Board/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "04.1_TCP_as_Client" and double click "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 shown in the box below:

04.1_TCP_as_Client

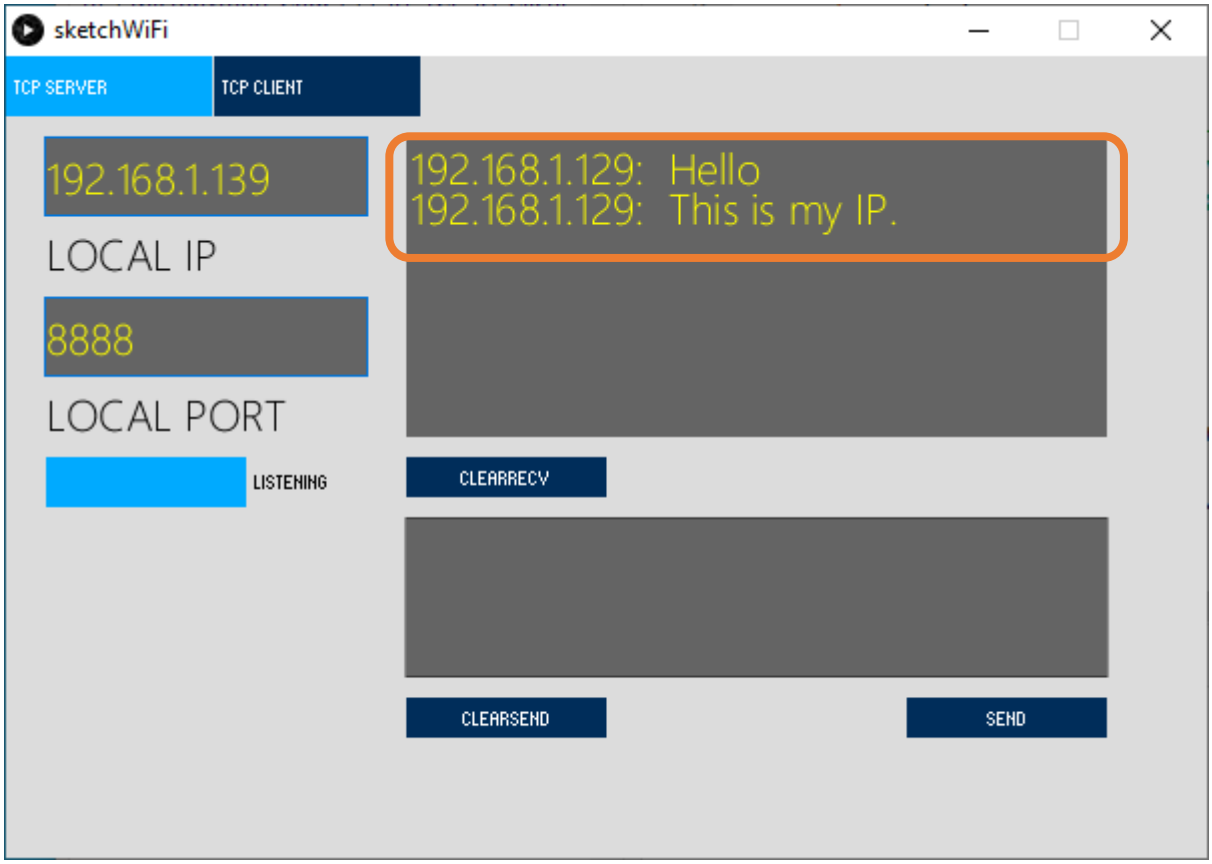


Click "Run current script" and in "Shell", you can see ESP32-S3 automatically connects to sketchWiFi.

If you don't click "Listening" for sketchWiFi, ESP32-S3 will fail to connect and will print information as follows:

```
Shell
  Close socket
>>> %Run -c $EDITOR_CONTENT
  TCP close, please reset!
>>>
```

ESP32-S3 connects with TCP SERVER, and TCP SERVER receives messages from ESP32S3, as shown in the figure below. You can enter any content in TCP SERVER, click SEND, and ESP32-S3 will receive this message



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.

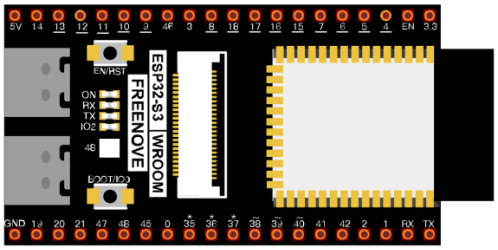
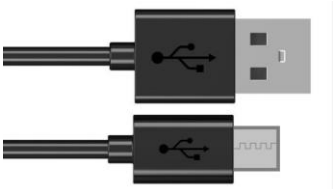
close(): Close socket.

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

Project 4.2 As Server

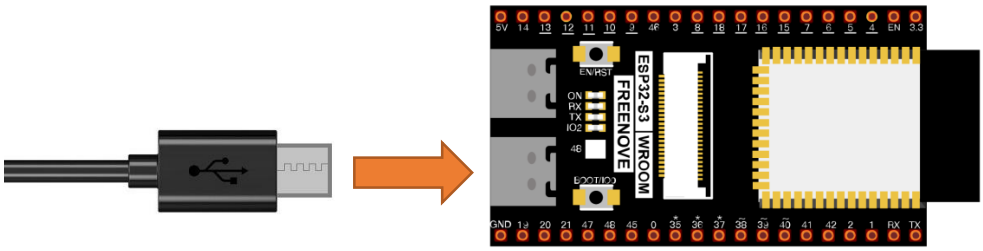
In this section, ESP32-S3 is used as a server to wait for the connection and communication of client on the same LAN.

Component List

ESP32-S3 WROOM x1 	USB cable x1 

Circuit

Connect Freenove ESP32-S3 to the computer using a USB cable.



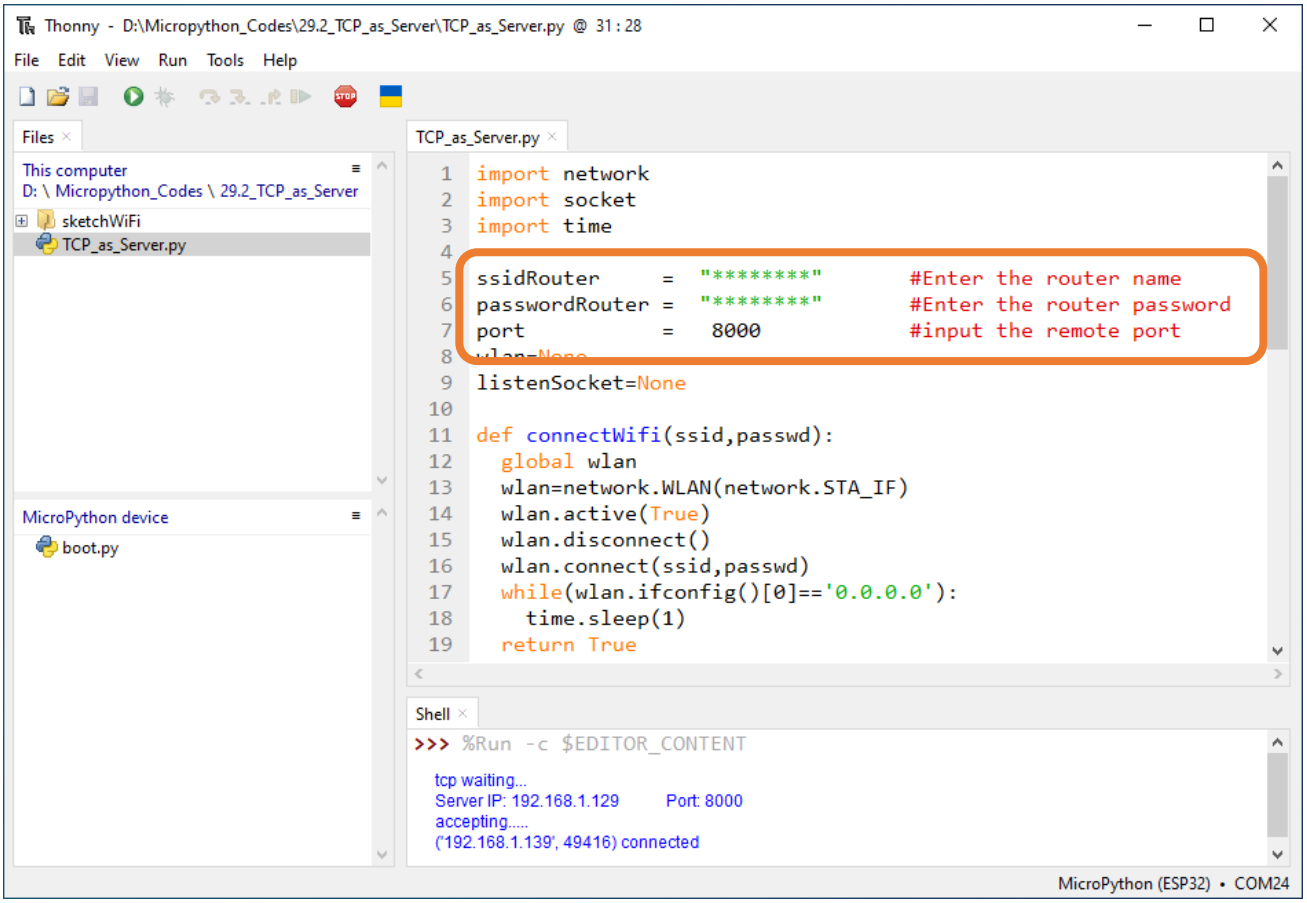
Code

Move the program folder "**Freenove_ESP32_S3_WROVER_Board/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "04.2_TCP_as_Server" and double click "TCP_as_Server.py".

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

04.2_TCP_as_Server

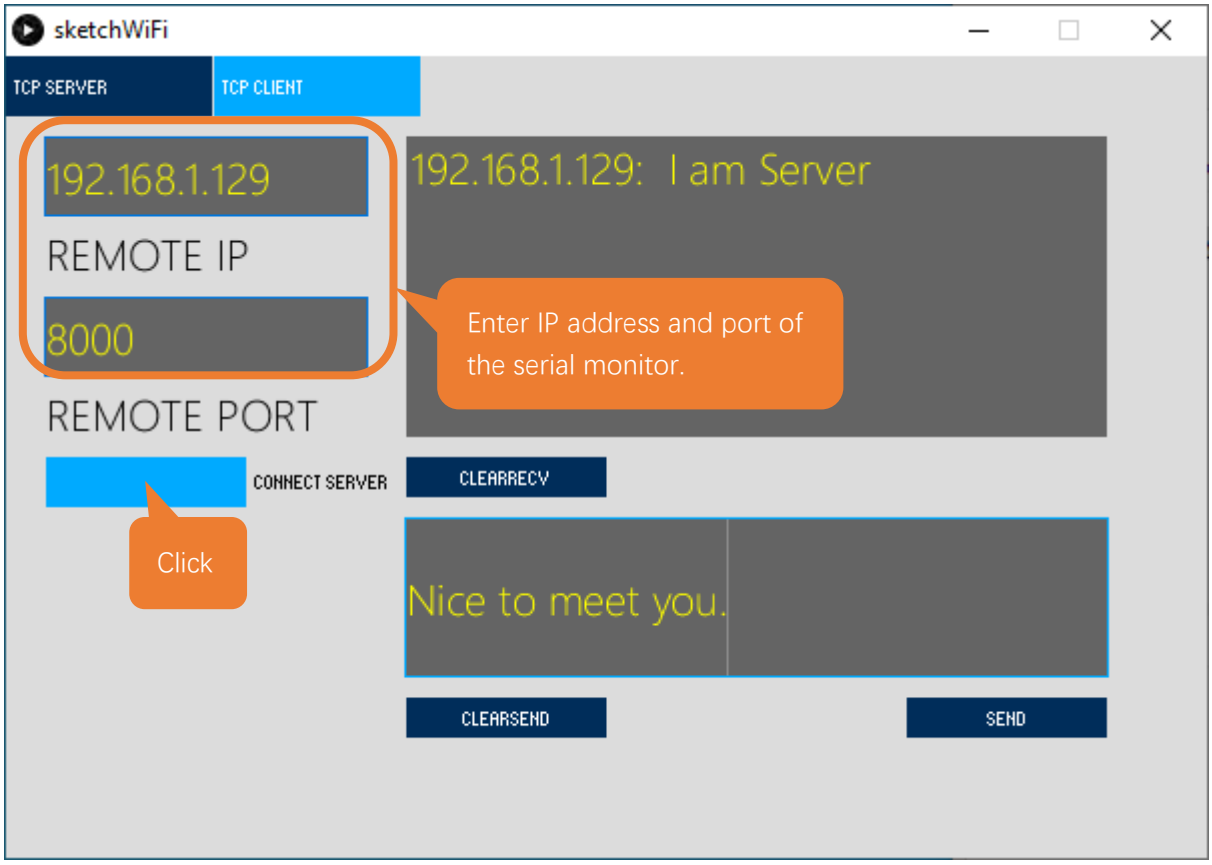


After making sure that the router's name and password are correct, click "Run current script" and in "Shell", you can see a server opened by the ESP32-S3 waiting to connecting to other network devices.

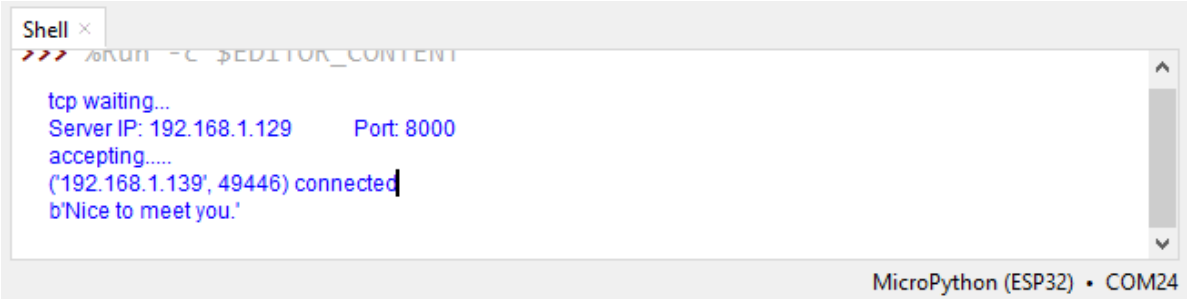
Processing:

Open the "**Freenove_ESP32_S3_WROVER_Board/Codes/MicroPython_Codes/04.2_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 ESP32-S3 to communicate.



You can enter any information in the "Send Box" of sketchWiFi. Click "Send" and ESP32-S3 will print the received messages to "Shell" and send them back to sketchWiFi.



The following is the program code:

```
import network
import socket
import time

ssidRouter     =  "********"       #Enter the router name
passwordRouter =  "********"       #Enter the router password
port           =   8000            #input the remote port
wlan           =  None
listenSocket   =  None

def connectWifi(ssid,passwd):
  global wlan
  wlan=network.WLAN(network.STA_IF)
  wlan.active(True)
  wlan.disconnect()
  wlan.connect(ssid,passwd)
  while(wlan.ifconfig()[0]=='0.0.0.0'):
    time.sleep(1)
  return True

try:
  connectWifi(ssidRouter,passwordRouter)
  ip=wlan.ifconfig()[0]
  listenSocket = socket.socket()
  listenSocket.bind((ip,port))
  listenSocket.listen(1)
  listenSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
  print('tcp waiting...')
  while True:
    print("Server IP:",ip,"\tPort:",port)
    print("accepting.....")
    conn,addr = listenSocket.accept()
    print(addr,"connected")
    break
  conn.send('I am Server')
  while True:
    data = conn.recv(1024)
    if(len(data) == 0):
      print("close socket")
      listenSocket.close()
      wlan.disconnect()
      wlan.active(False)
      break
```

```
    else:
      print(data)
      ret = conn.send(data)
except:
  print("Close TCP-Server, please reset.")
  if(listenSocket):
    listenSocket.close()
  wlan.disconnect()
  wlan.active(False)
```

Call function connectWifi() to connect to router and obtain the dynamic IP that it assigns to ESP32-S3.

```
  connectWifi(ssidRouter,passwordRouter)
  ip=wlan.ifconfig()[0]
```

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

```
  listenSocket = socket.socket()
  listenSocket.bind((ip,port))
  listenSocket.listen(1)
  listenSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

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

```
  while True:
    print("Server IP:",ip,"\tPort:",port)
    print("accepting.....")
    conn,addr = listenSocket.accept()
    print(addr,"connected")
    break
```

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

```
  while True:
    data = conn.recv(1024)
    if(len(data) == 0):
      print("close socket")
      listenSocket.close()
      wlan.disconnect()
      wlan.active(False)
      break
    else:
      print(data)
      ret = conn.send(data)
```

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

```
except:
  print("Close TCP-Server, please reset.")
  if(listenSocket):
    listenSocket.close()
  wlan.disconnect()
  wlan.active(False)
```

What's next?

Thanks for your reading. This tutorial is all over here. If you find any mistakes, omissions or you have other ideas and questions about contents of this tutorial or the kit and etc., please feel free to contact us:

support@freenove.com

We will check and correct it as soon as possible.

If you want learn more about ESP32S3, you view our ultimate tutorial:
https://github.com/Freenove/Freenove_Ultimate_Starter_Kit_for_ESP32_S3/archive/master.zip

If you want to learn more about Arduino, Raspberry Pi, smart cars, robots and other interesting products in science and technology, please continue to focus on our website. We will continue to launch cost-effective, innovative and exciting products.

http://www.freenove.com/

End of the Tutorial

Thank you again for choosing Freenove products.