

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.

Any concerns?  support@freenove.com



About Freenove

Freenove provides open source electronic products and services worldwide.

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

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

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

<http://www.freenove.com>

sale@freenove.com

Copyright

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



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

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



Contents

Important Information	1
Contents.....	1
Preface.....	2
ESP32-S3 WROOM	3
CH343 (Importance).....	5
Programming Software	17
Basic Configuration of Thonny	23
Burning Micropython Firmware (Important).....	24
Testing codes (Important).....	30
Thonny Common Operation.....	38
Notes for GPIO.....	45
Chapter 1 LED (Important).....	48
Project 1.1 Blink	48
Chapter 2 Bluetooth.....	58
Project 2.1 Bluetooth Low Energy Data Passthrough.....	58
Chapter 3 WiFi Working Modes.....	70
Project 3.1 Station mode	70
Project 3.2 AP mode	75
Project 3.3 AP+Station mode	79
Chapter 4 TCP/IP.....	83
Project 4.1 As Client	83
Project 4.2 As Server	95
Chapter 5 Camera Web Server	100
Project 05.1 Camera Web Server.....	100
What's next?	111
End of the Tutorial.....	111

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 Microcontroller(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.

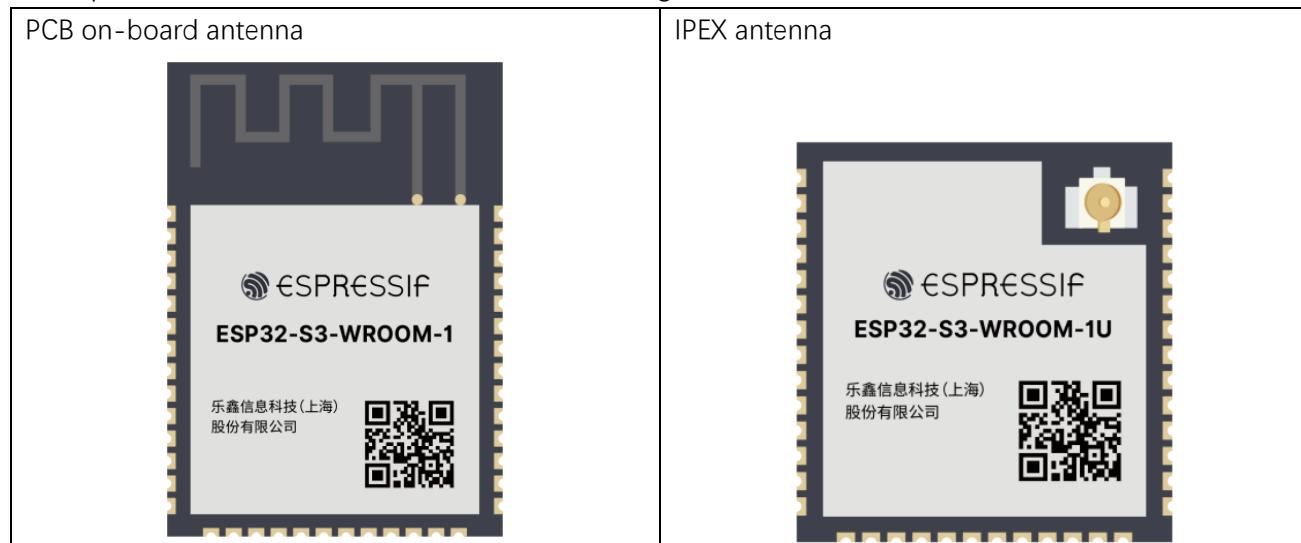
Generally, ESP32-S3 projects consist of code and circuits. Don't worry even if you've never learned code and circuits, because we will gradually introduce the basic knowledge of C programming language and electronic circuits, from easy to difficult. Our products contain all the electronic components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

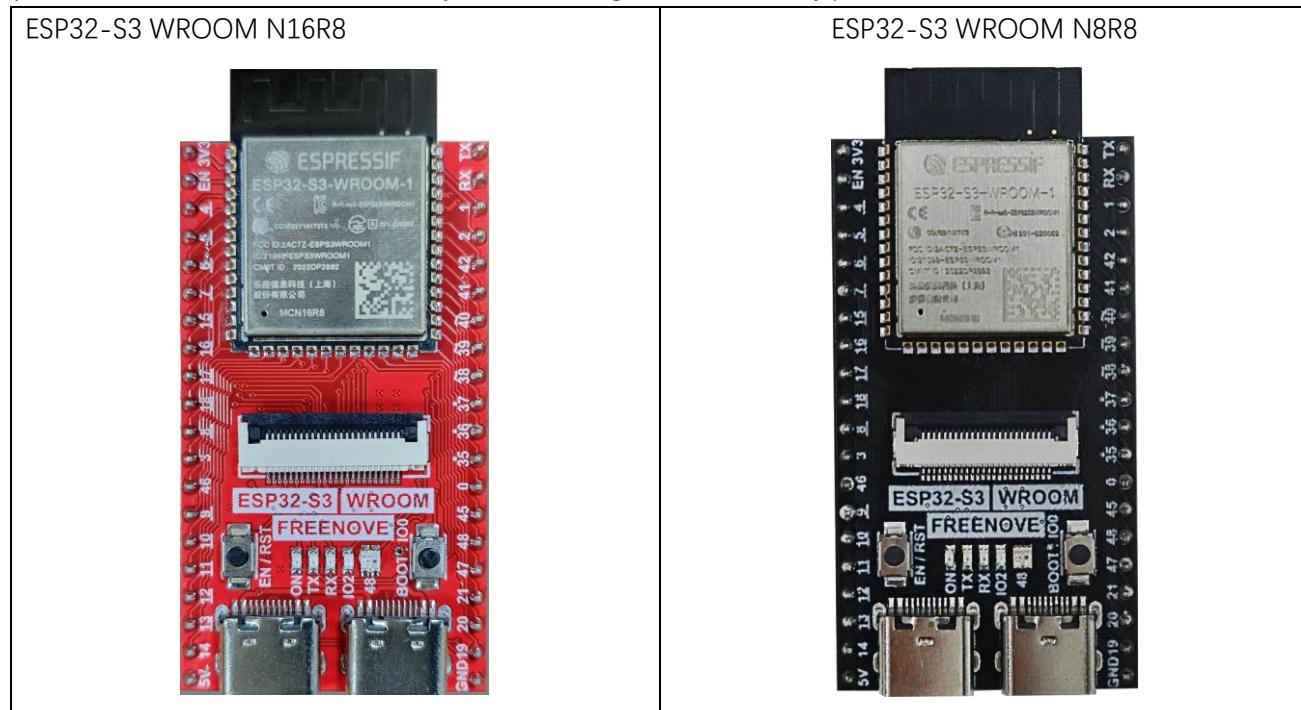
ESP32-S3-WROOM-1 has launched a total of two antenna packages, PCB on-board antenna and IPEX antenna respectively. The PCB on-board antenna is an integrated antenna in the chip module itself, so it is convenient to carry and design. The IPEX antenna is a metal antenna derived from the integrated antenna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



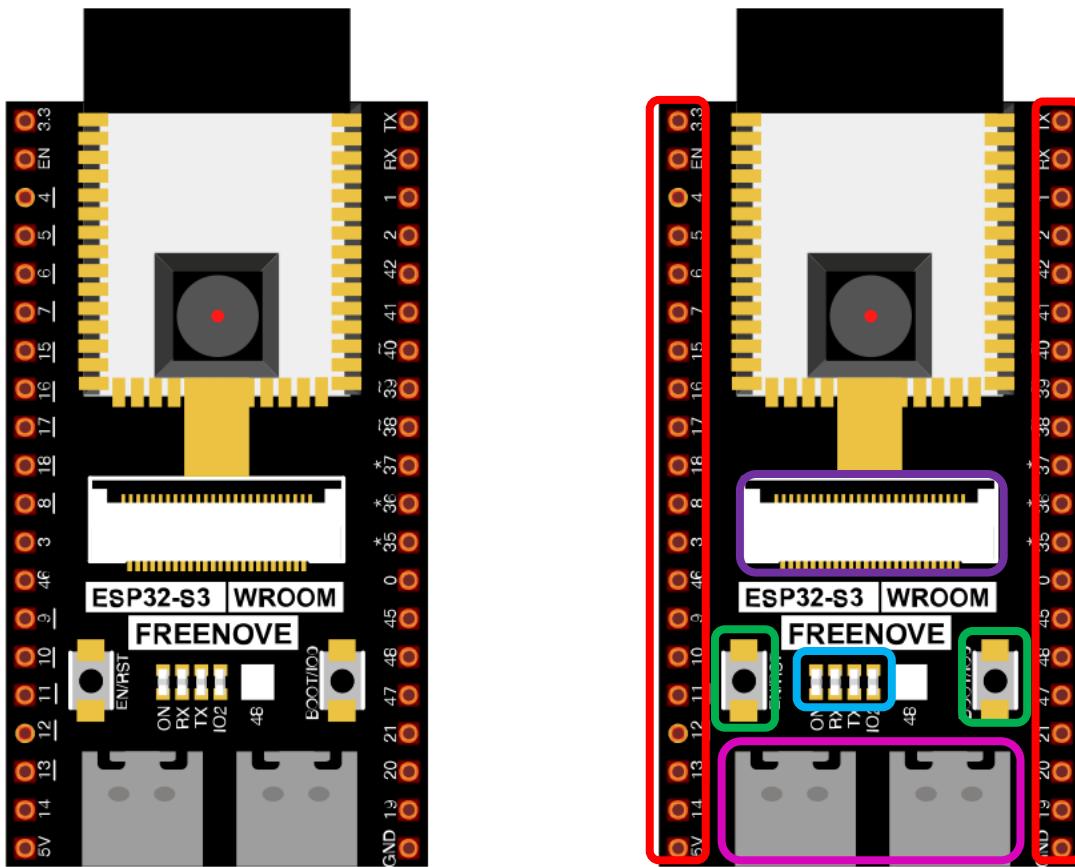
In this tutorial, the ESP32-S3 WROOM is designed based on the PCB on-board antenna-packaged ESP32-S3-WROOM-1 module.

Currently, we have two development board options available: the N8R8 version with a black PCB and the N16R8 version with a red PCB. The only difference between these two boards lies in their built-in Flash storage capacity, while all other features, functionalities, and peripheral circuits remain identical.

(Note: The N16R8 version is currently sold as a single-board kit only.)



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.

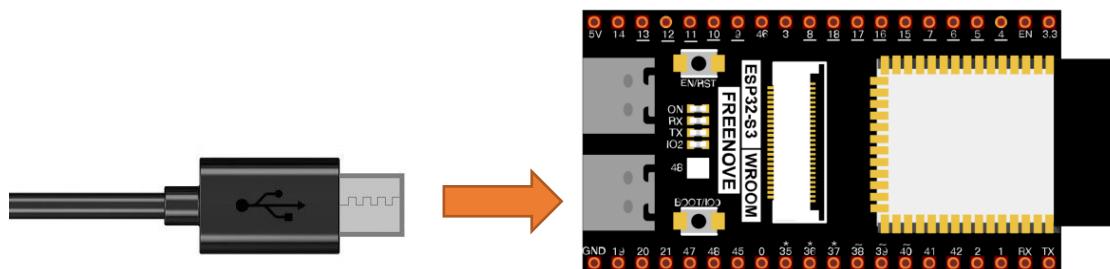
CH343 (Importance)

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

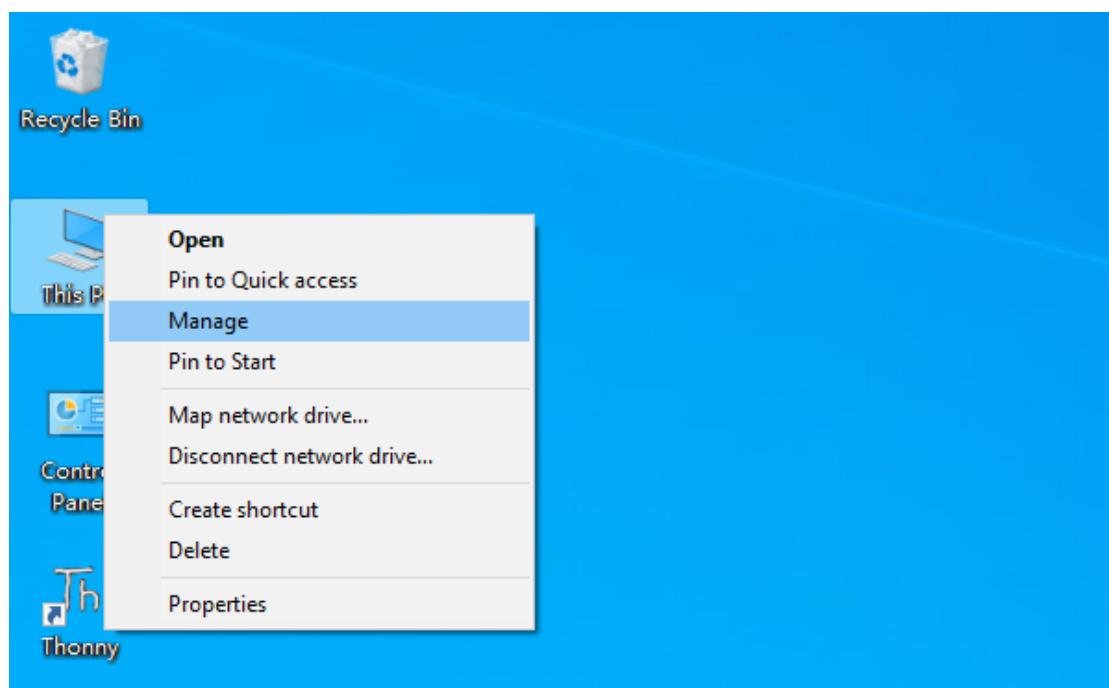
Windows

Check whether CH343 has been installed

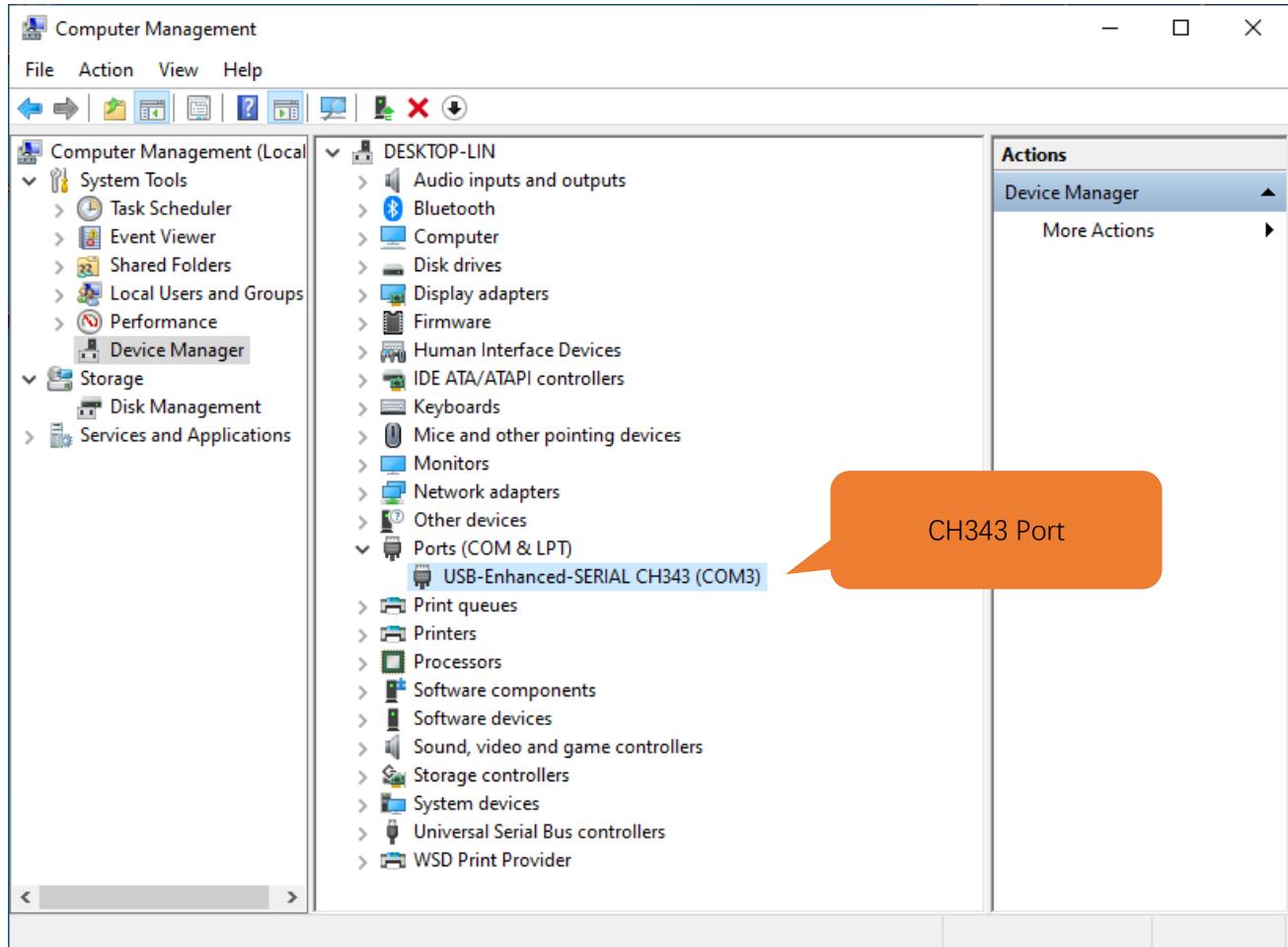
1. Connect your computer and ESP32-S3 WROOM with a USB cable.



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-Enhanced-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.

keyword ch343

Downloads(8)

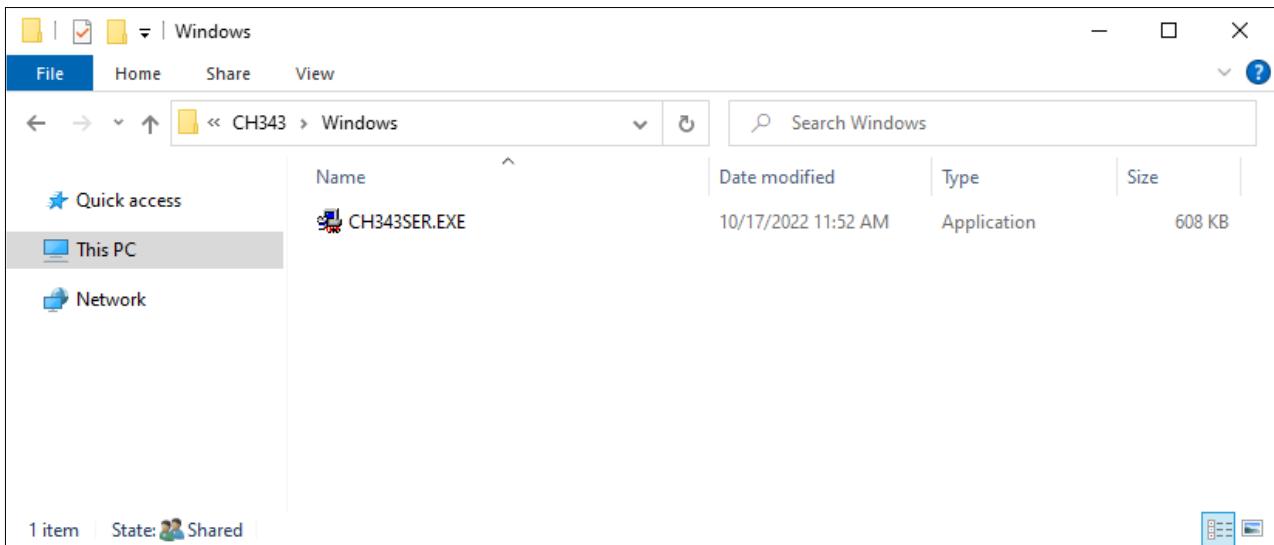
file category	file content	version	upload time
DataSheet			
CH343DS1.PDF	CH343 datasheet, USB to single serial port, supports up to 6M baud rate, serial port signals support 5V/3.3V/2.5V/1.8V, built-in crystal oscillator. CH343 supports built-in CDC driver in operating system or multi-functional high-speed VCP manufacture driver.	1.5	2021-11-18
Driver&Tools			
CH343SER.ZIP	For CH342/CH343/CH344/CH347/CH9101/CH9102/CH9103/CH9143, USB to high-speed serial port VCP vendor driver, supports Windows 11/10/8.1/8/7/VISTA/XP/2000	1.61	2022-05-13
CH343CDC.ZIP	For CH342/CH343/CH344/CH347/CH910X/CH9143/CH9340, USB to CDC serial port driver, supports Windows 11/10/8.1/8/7/VISTA/XP/2000	1.4	2022-05-13
CH343SER.EXE	For CH342/CH343/CH344/CH347/CH9101/CH9102/CH9103/CH9143, USB to high-speed serial port VCP vendor driver, supports Windows 11/10/8.1/8/7/VISTA/XP/2000	1.61	2022-05-13
CH34XSER_MAC.ZI...	For CH340/CH341/CH342/CH343/CH344/CH347/CH9101/CH9102/CH9103/CH9143, USB to serial port VCP vendor driver of macOS	1.7	2022-05-13
CH343CDC.EXE	For CH342/CH343/CH344/CH347/CH910X/CH9143/CH9340, USB to CDC serial port driver, supports Windows 11/10/8.1/8/7/VISTA/XP/2000	1.4	2022-05-13
Application			
CH34xSerCfg.ZIP	USB configuration tool of Windows for CH340/CH342/CH343/CH344/CH347/CH348/CH9101/CH9102/CH9103. Via this tool, the chip's Vendor ID, product ID, maximum current value, BCD version	1.2	2022-05-24

If you would not like to download the installation package, you can open

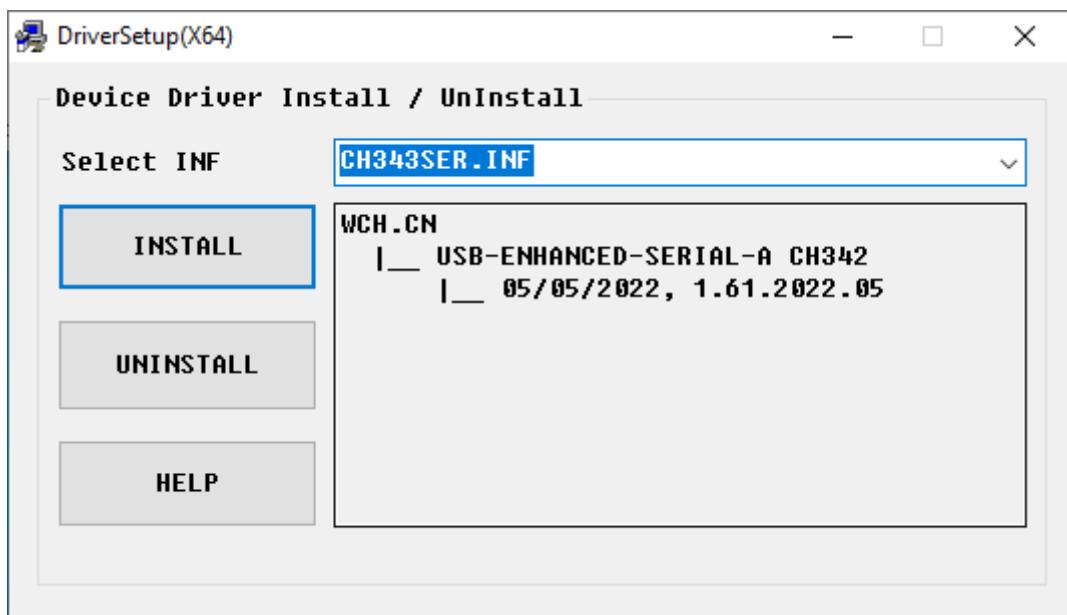
"Freenove_ESP32_S3_WROVER_Board/CH343", we have prepared the installation package.

 Linux	10/17/2022 1:30 PM	File folder
 MAC	10/17/2022 1:30 PM	File folder
 Windows	10/17/2022 1:30 PM	File folder

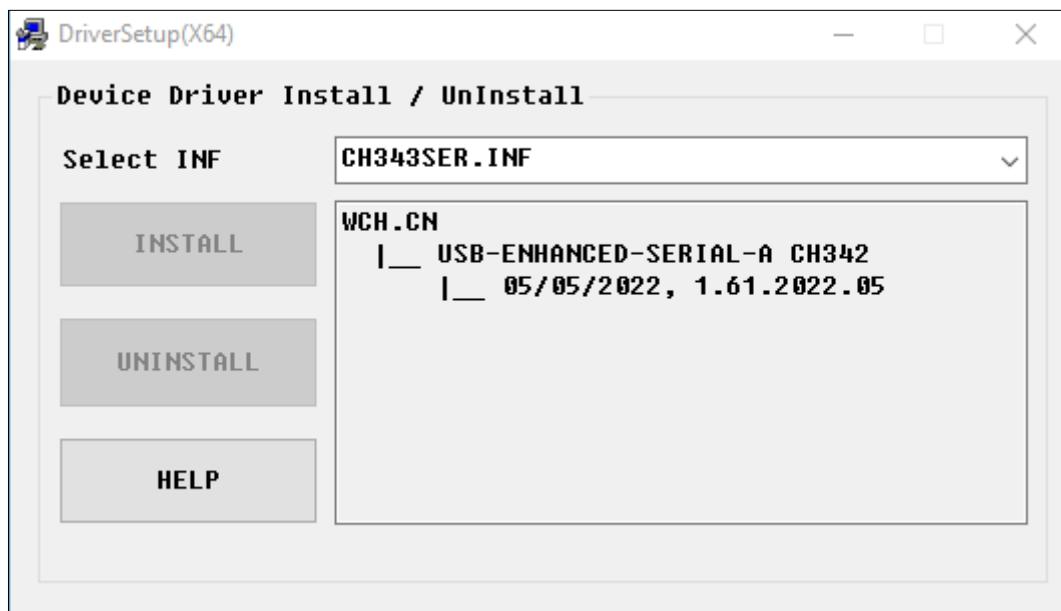
2. Open the folder “Freenove_ESP32_S3_WROVER_Board/CH343/Windows/”



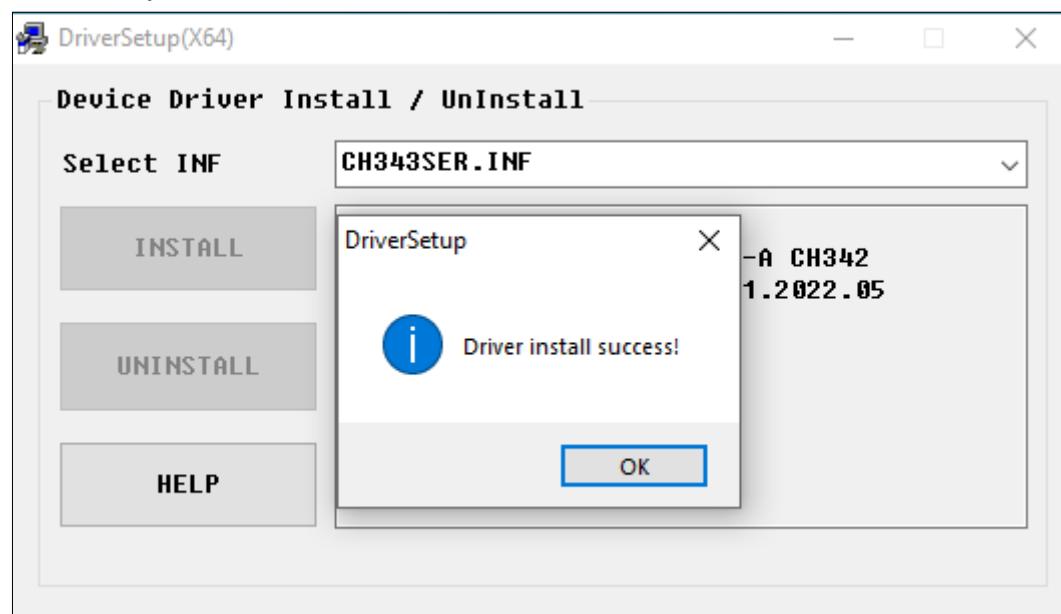
3. Double click “CH343SER.EXE”.



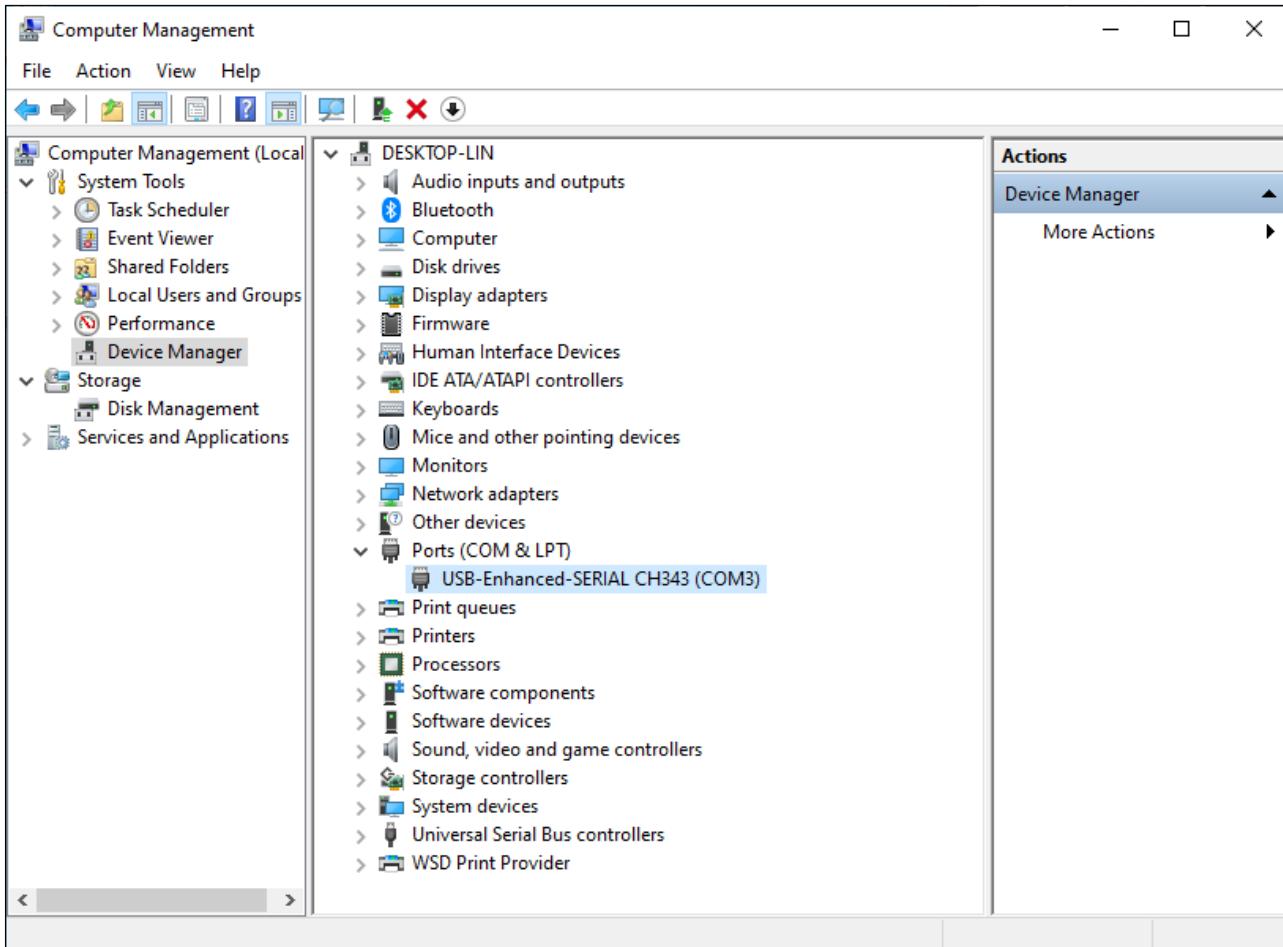
4. Click “INSTALL” and wait for the installation to complete.



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

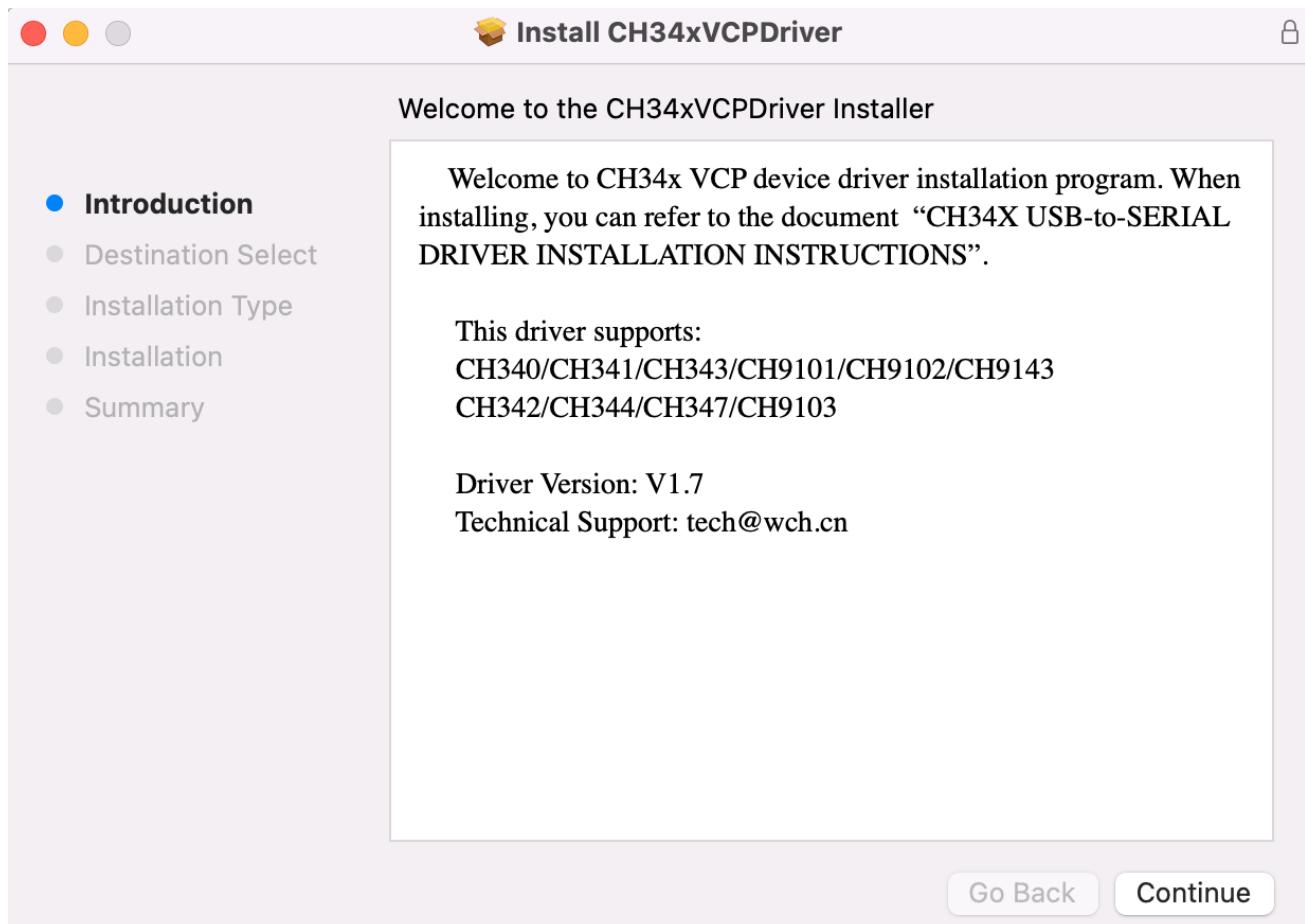
First, download CH343 driver, click <http://www.wch-ic.com/search?t=all&q=ch343> to download the appropriate one based on your operating system.

keyword ch343				
Downloads(8)				
file category	file content	version	upload time	
DataSheet				
CH343DS1.PDF	CH343 datasheet, USB to single serial port, supports up to 6M baud rate, serial port signals support 5V/3.3V/2.5V/1.8V, built-in crystal oscillator. CH343 supports built-in CDC driver in operating system or multi-functional high-speed VCP manufacture driver.	1.5	2021-11-18	
Driver&Tools				
CH343SER.ZIP	For CH342/CH343/CH344/CH347/CH9101/CH9102/CH9103/CH9143, USB to high-speed serial port VCP vendor driver, supports Windows 11/10/8.1/8/7/VISTA/XP/2000	1.61	2022-05-13	
CH343CDC.ZIP	For CH342/CH343/CH344/CH347/CH910X/CH9143/CH9340, USB to CDC serial port driver, supports Windows 11/10/8.1/8/7/VISTA/XP/2000	1.4	2022-05-13	
CH343SER.EXE	For CH343SER.EXE, CH347/CH9101/CH9102/CH9103/CH9143, USB to high-speed serial port VCP vendor driver, supports Windows 11/10/8.1/8/7/VISTA/XP/2000	1.61	2022-05-13	
CH34XSER_MAC.ZIP	For CH34XSER_MAC.ZIP, CH344/CH347/CH9101/CH9102/CH9103/CH9143, USB to serial port VCP vendor driver of macOS	1.7	2022-05-13	
CH343CDC.EXE	For CH342/CH343/CH344/CH347/CH910X/CH9143/CH9340, USB to CDC serial port driver, supports Windows 11/10/8.1/8/7/VISTA/XP/2000	1.4	2022-05-13	
Application				
CH34xSerCfg.ZIP	USB configuration tool of Windows for CH340/CH342/CH343/CH344/CH347/CH348/CH9101/CH9102/CH9103. Via this tool, the chip's Vendor ID, product ID, maximum current value, BCD version	1.2	2022-05-24	

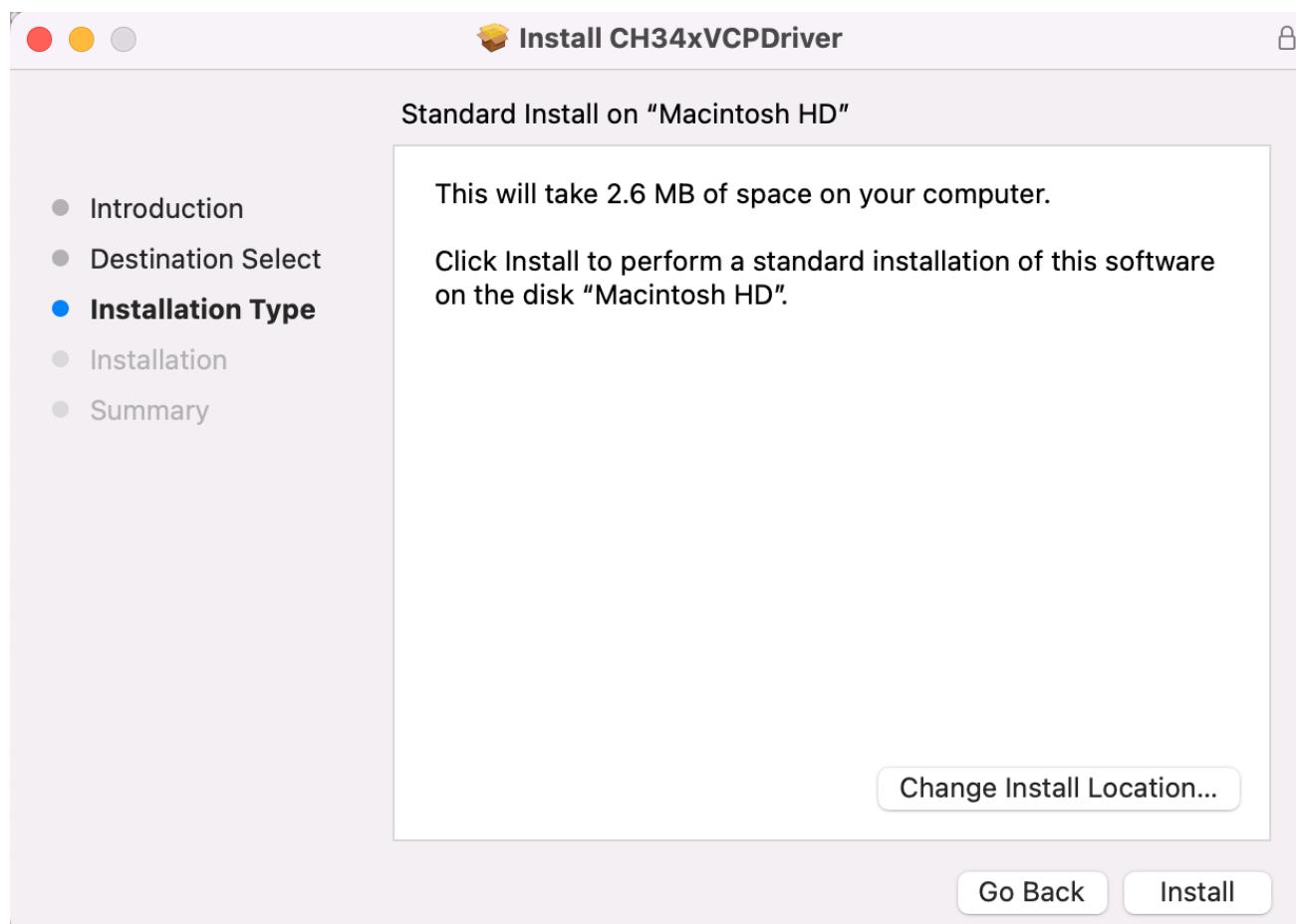
If you would not like to download the installation package, you can open “**Freenove_ESP32_S3_WROVER_Board/CH343**”, we have prepared the installation package. Second, open the folder “**Freenove_ESP32_S3_WROVER_Board/CH343/MAC/**”



Third, click Continue.

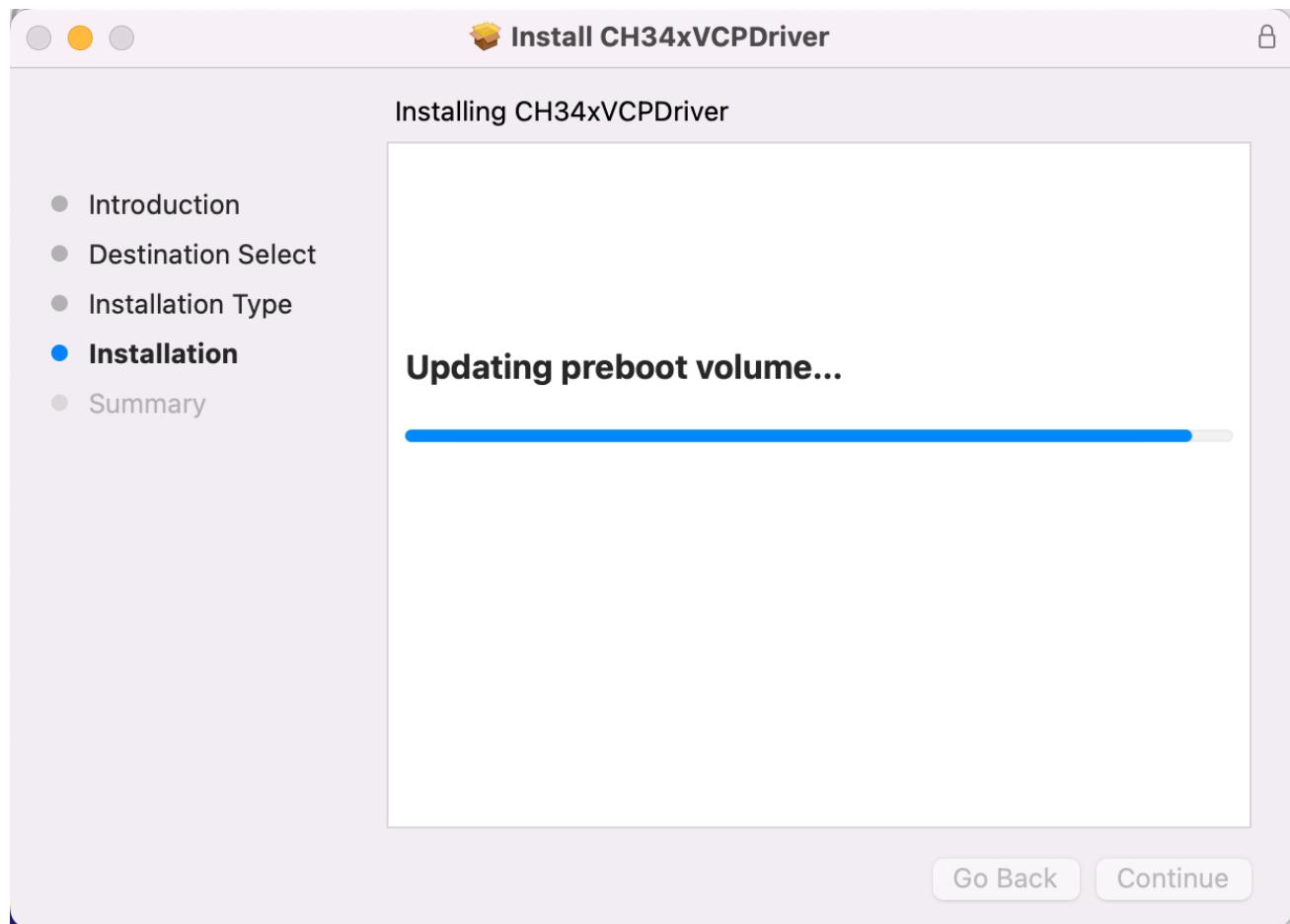


Fourth, click Install.

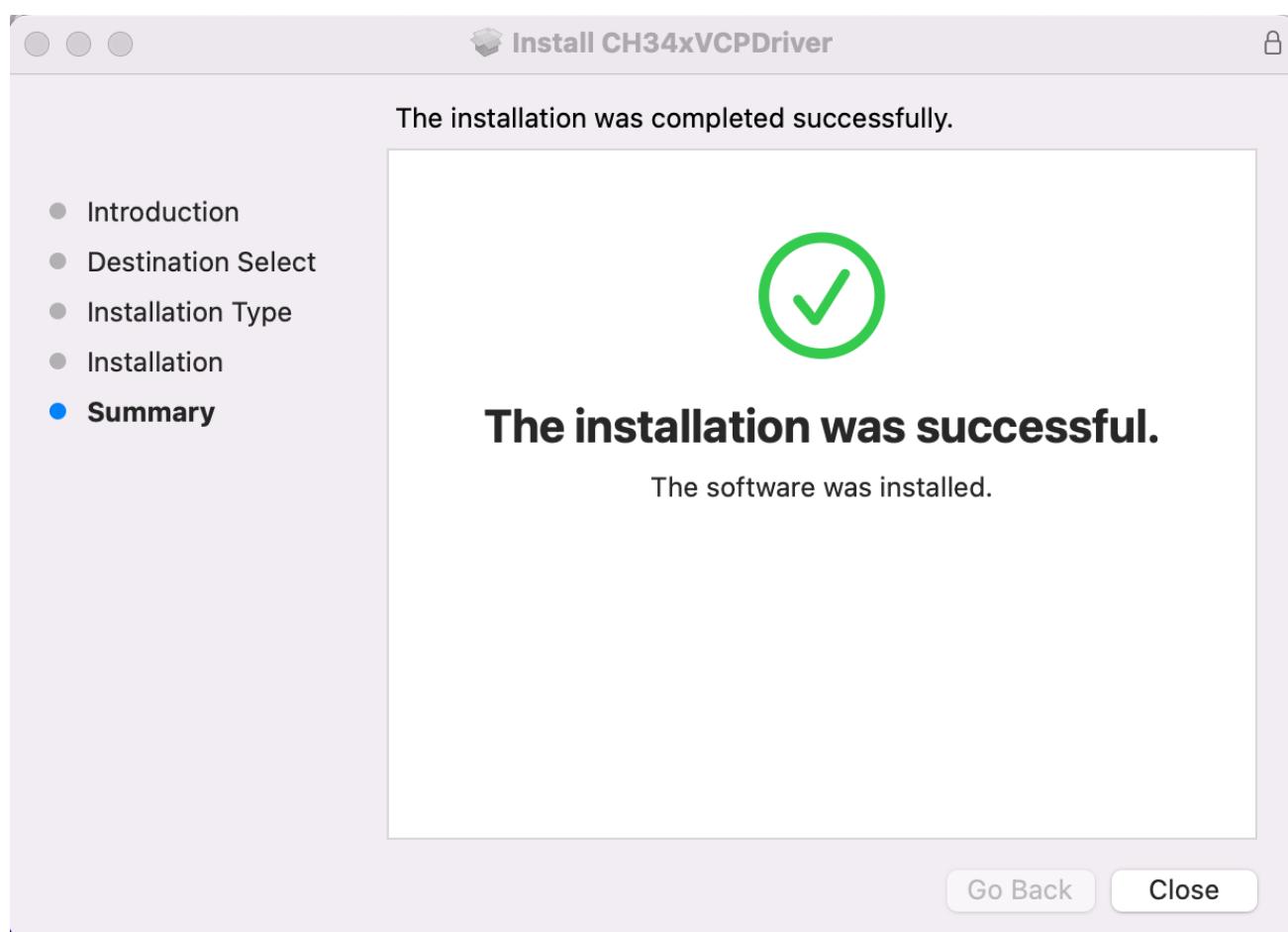




Then, waiting Finsh.

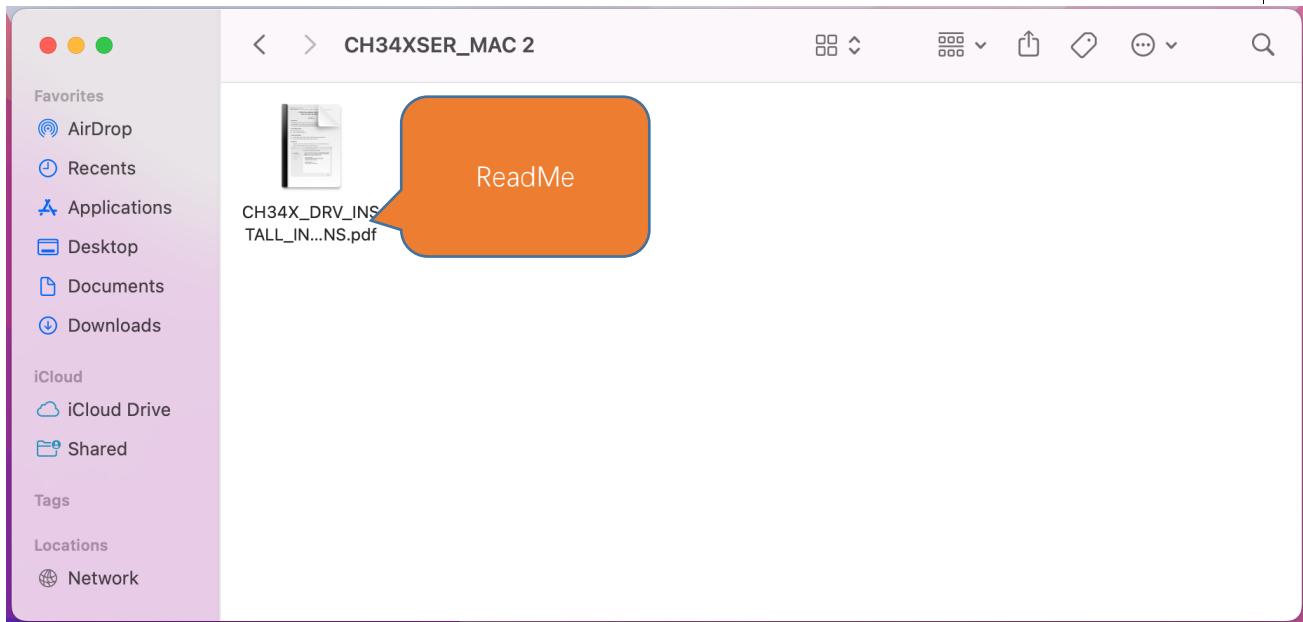


Finally, restart your PC.





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

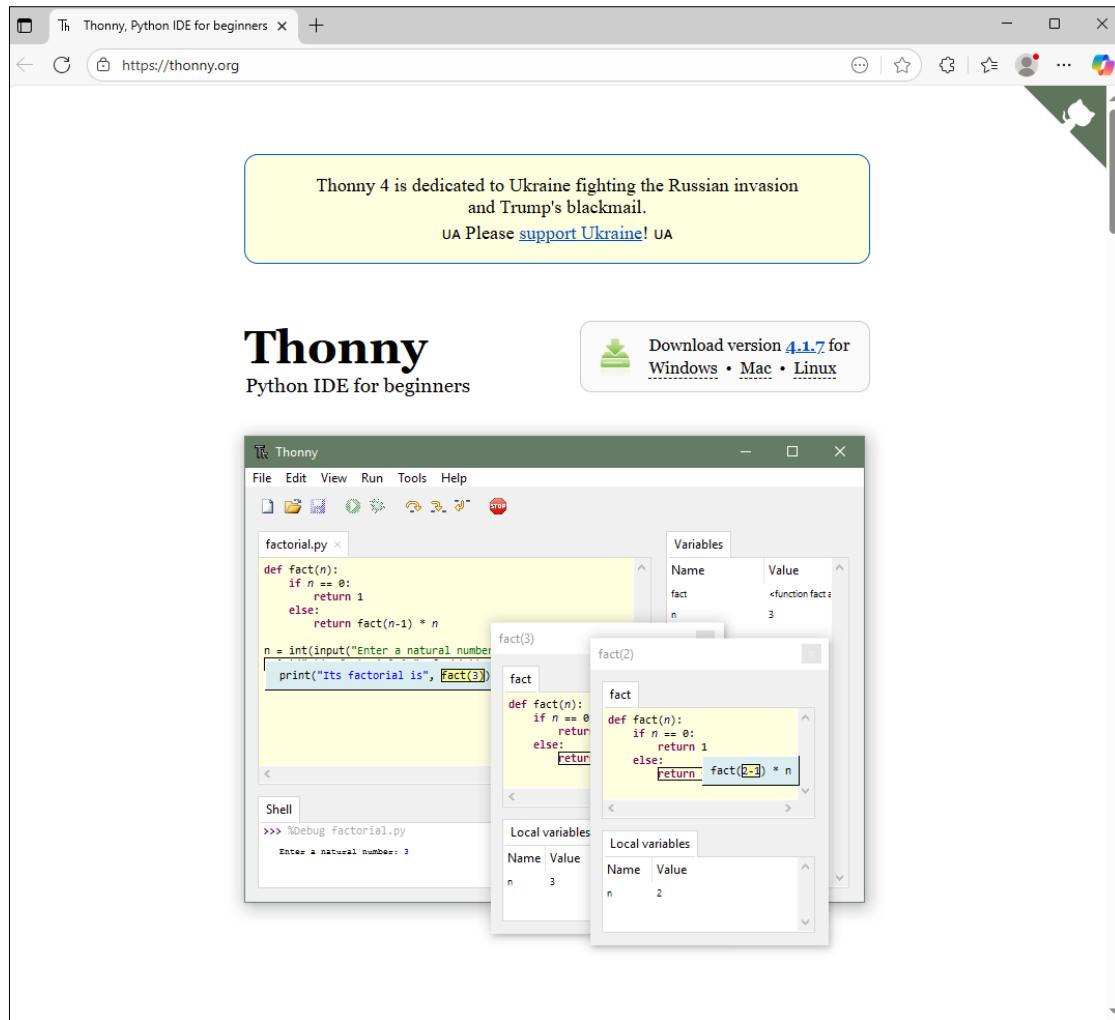
Official website of Thonny: <https://thonny.org>

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.)

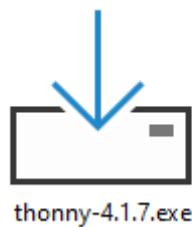
Operating System	Download links/methods
Windows	https://github.com/thonny/thonny/releases/download/v4.1.7/thonny-4.1.7.exe
Mac OS	https://github.com/thonny/thonny/releases/download/v4.1.7/thonny-4.1.7.pkg
Linux	Official downloads for Linux Installer (installs private Python 3.10 on x86_64, uses existing 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 Fedor sudo dnf install thonny

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

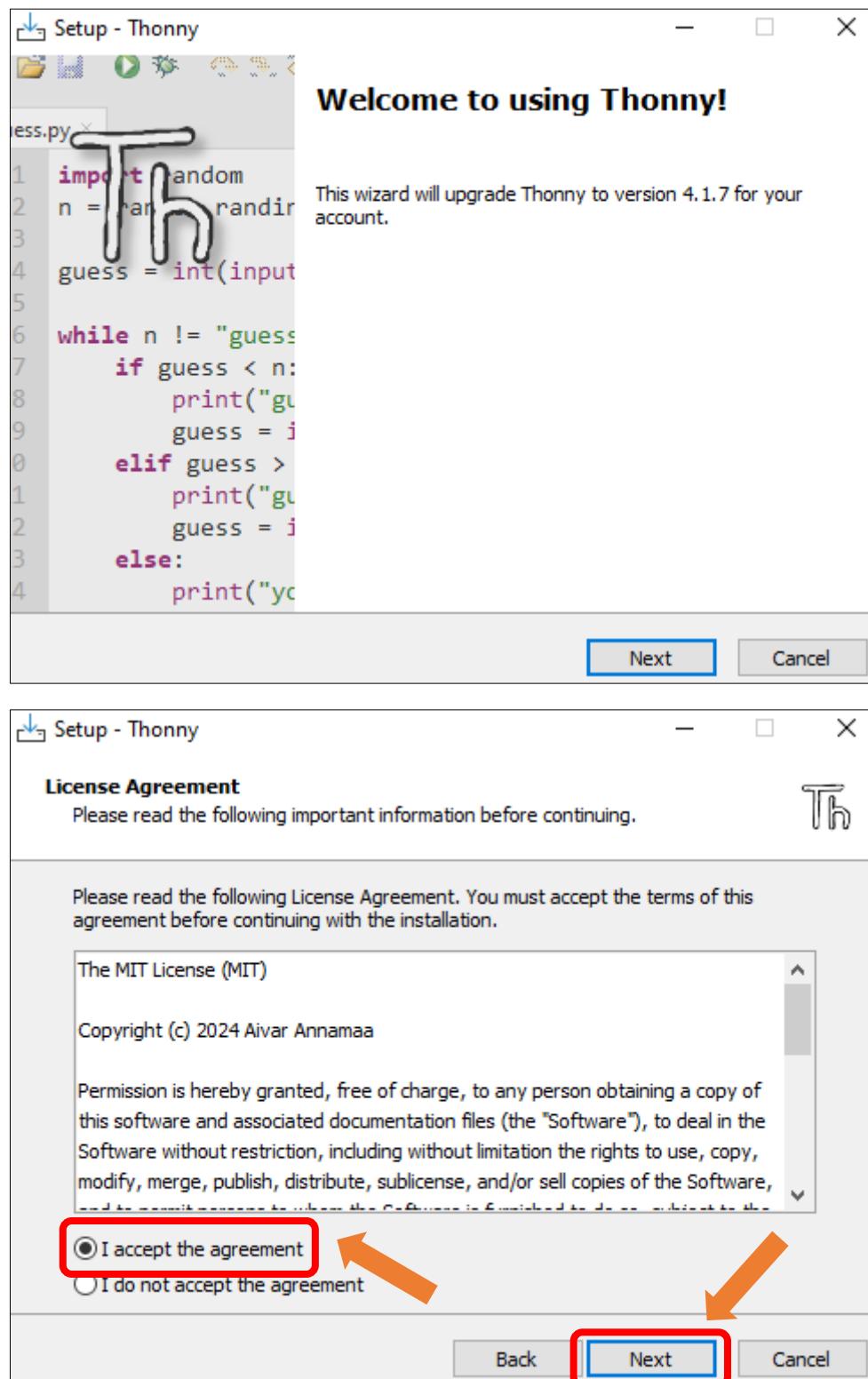


Installing on Windows

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

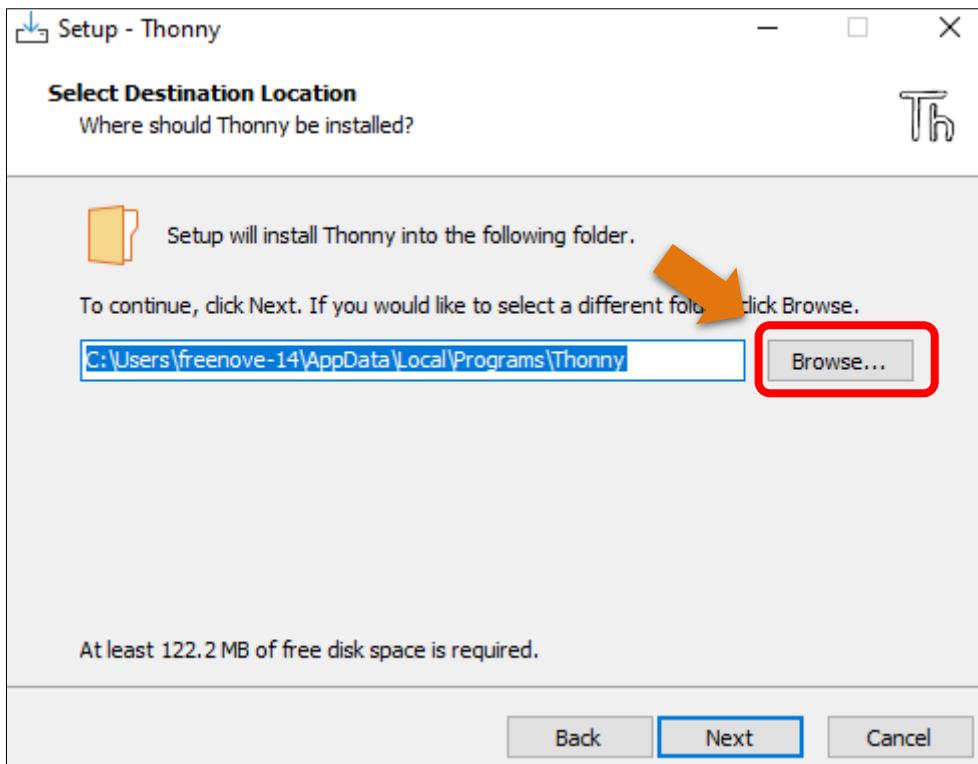


If you're not familiar with computer software installation, you can simply keep clicking "Next" until the installation completes.

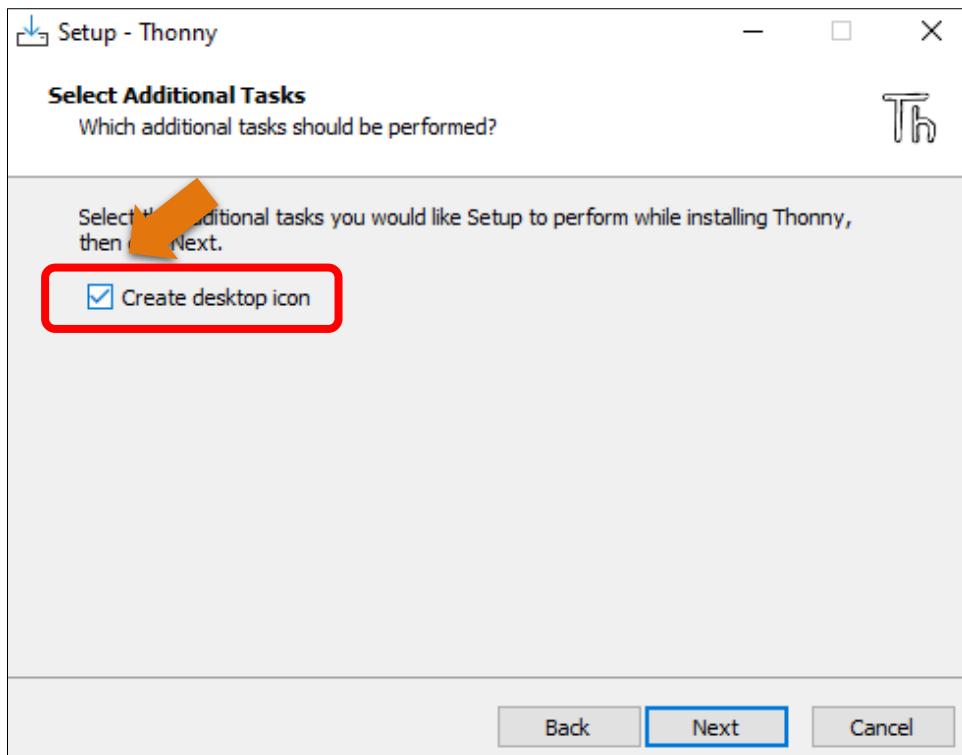


If you want to change Thonny's installation path, you can click "Browse" to modify it. After selecting installation path, click "OK".

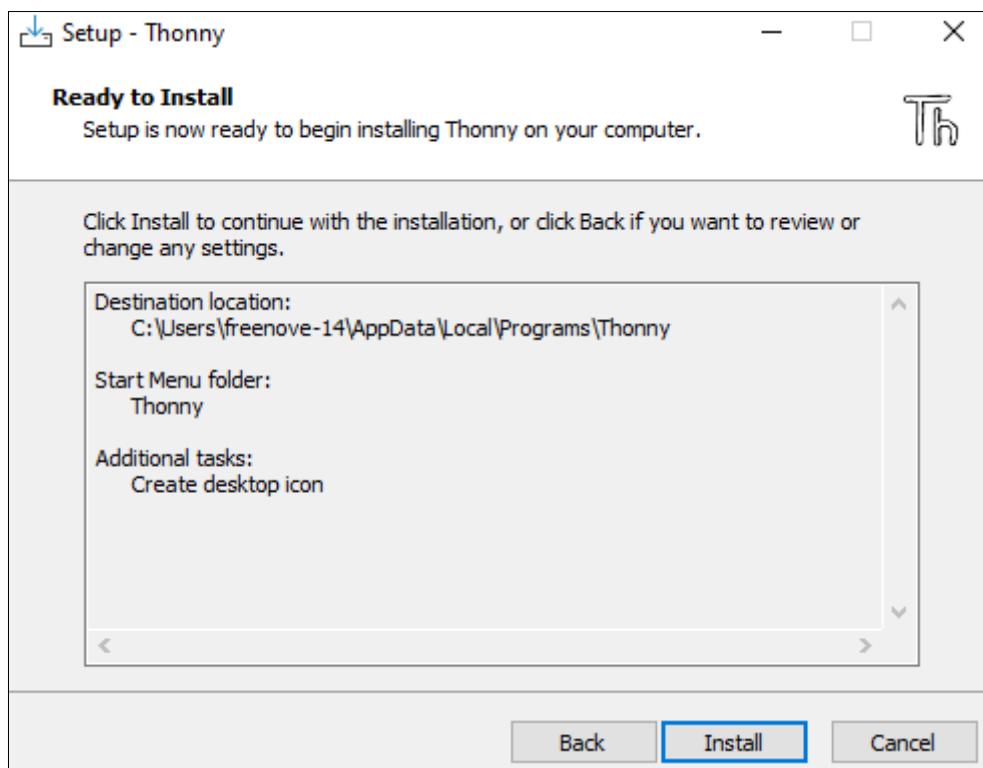
If you do not want to change it, just click "Next".



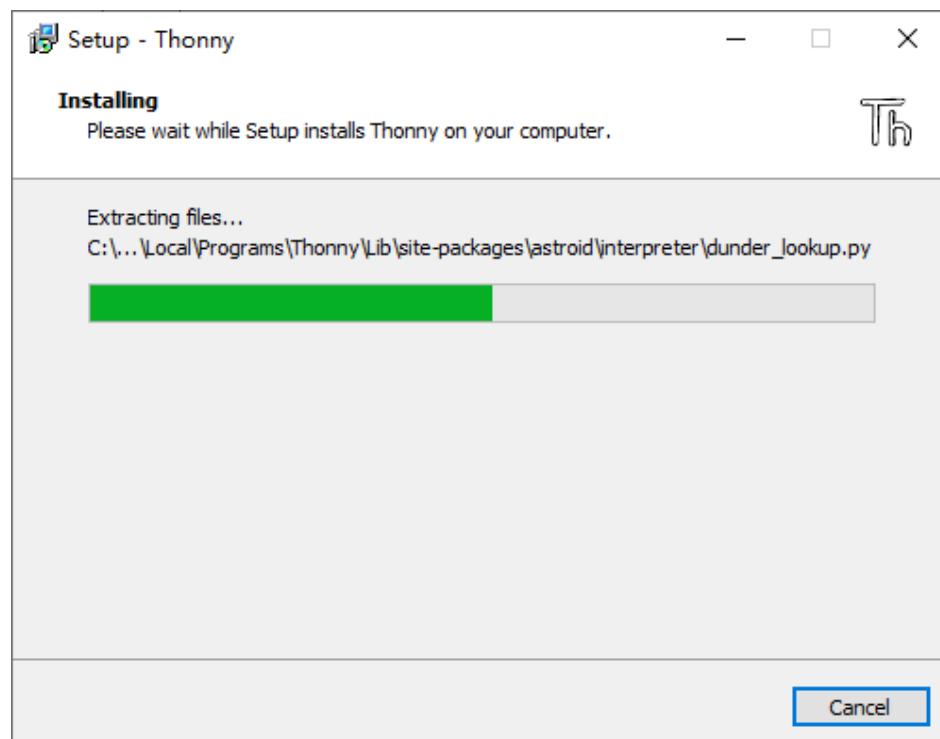
Check "Create desktop icon" and then it will generate a shortcut on your desktop to facilitate you to open Thonny later.



Click "install" to install the software.

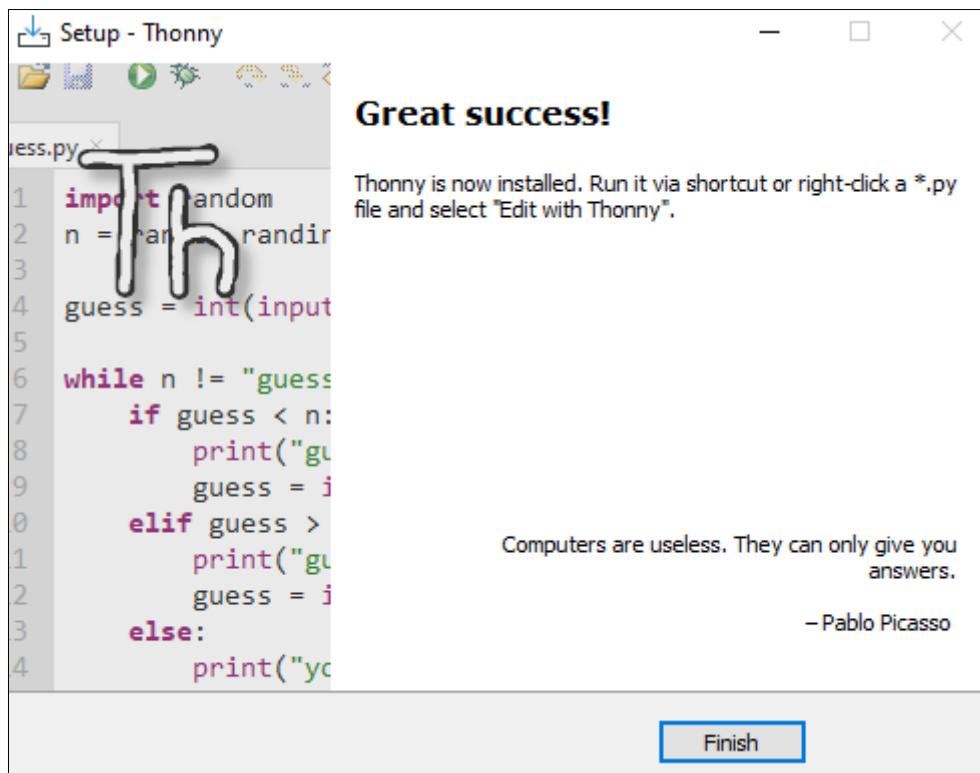


During the installation process, you only need to wait for the installation to complete, and you must not click "Cancel", otherwise Thonny will fail to be installed.





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

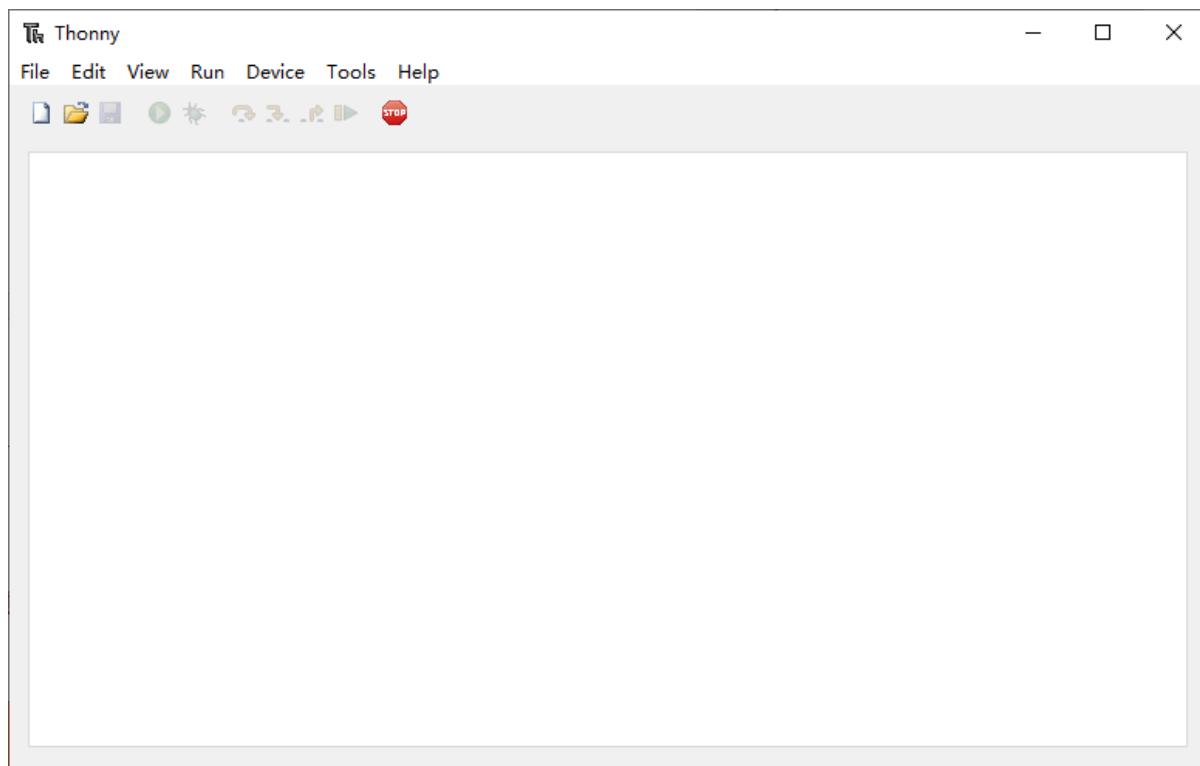


If you've checked "Create desktop icon" during the installation process, you can see the below icon on your desktop..

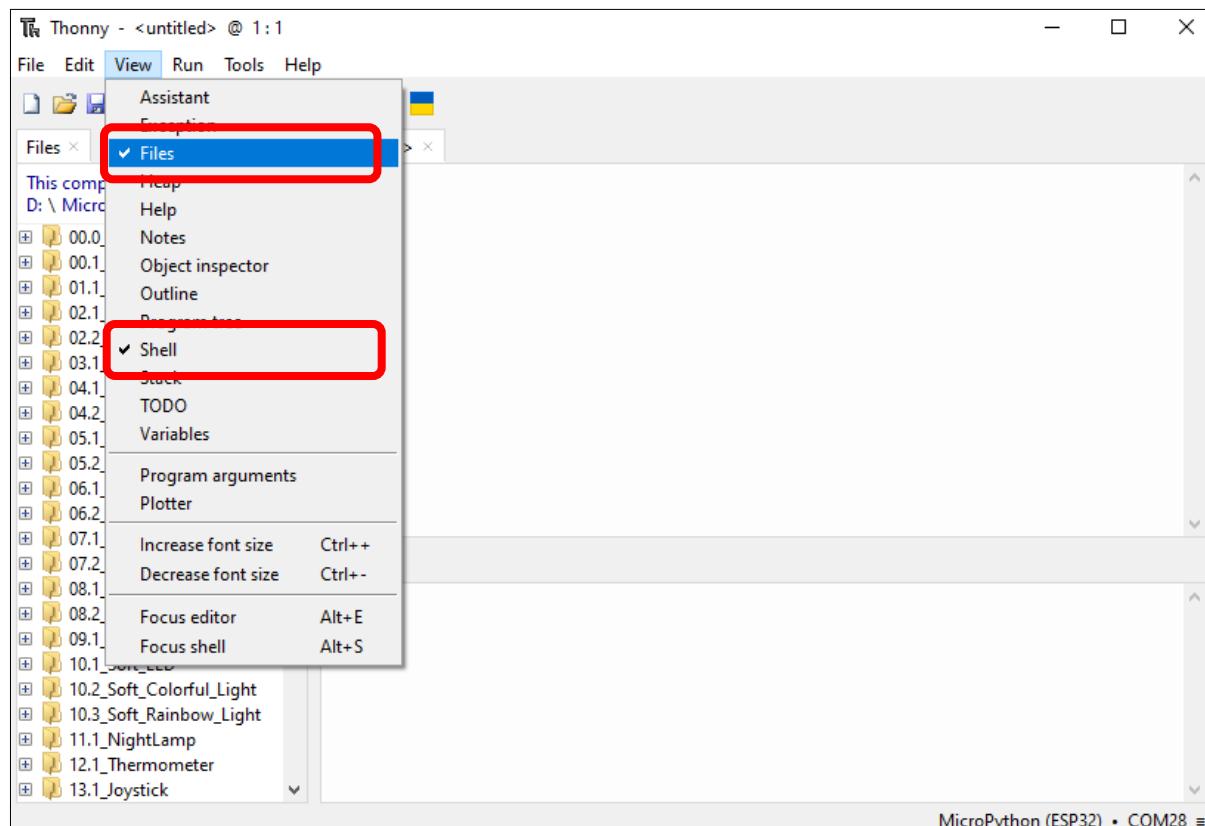


Basic Configuration of Thonny

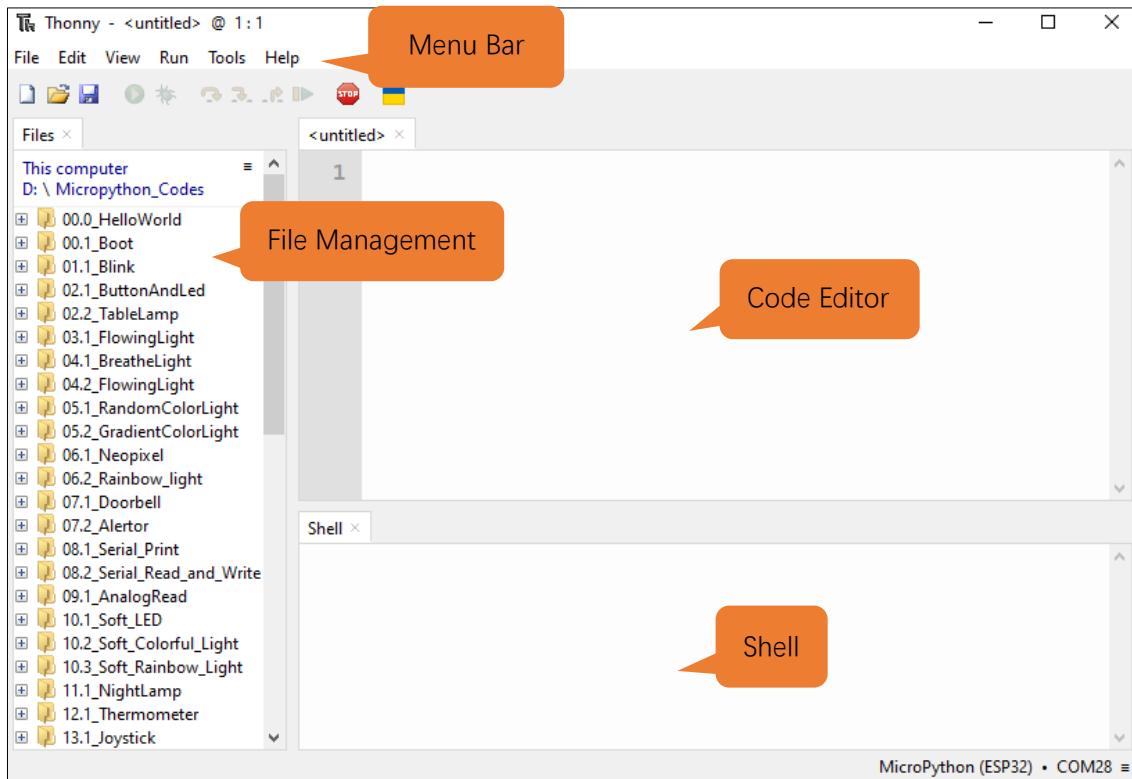
Click the desktop icon of Thonny and you can see the interface of it as follows:



Select "View" → "Files" and "Shell".



Any concerns? ✉ support@freenove.com



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

Releases

[**v1.26.0 \(2025-08-09\) .uf2**](#) / [.app-bin] / [.bin] / [.elf] / [.map] / [Release notes] (latest)
[v1.25.0 \(2025-04-15\) .uf2](#) / [.app-bin] / [.bin] / [.elf] / [.map] / [Release notes]
[v1.24.1 \(2024-11-29\) .uf2](#) / [.app-bin] / [.bin] / [.elf] / [.map] / [Release notes]
[v1.24.0 \(2024-10-25\) .uf2](#) / [.app-bin] / [.bin] / [.elf] / [.map] / [Release notes]
[v1.23.0 \(2024-06-02\) .uf2](#) / [.app-bin] / [.bin] / [.elf] / [.map] / [Release notes]
[v1.22.2 \(2024-02-22\) .uf2](#) / [.app-bin] / [.bin] / [.elf] / [.map] / [Release notes]
[v1.22.1 \(2024-01-05\) .uf2](#) / [.app-bin] / [.bin] / [.elf] / [.map] / [Release notes]
[v1.22.0 \(2023-12-27\) .uf2](#) / [.app-bin] / [.bin] / [.elf] / [.map] / [Release notes]
[v1.21.0 \(2023-10-05\) .uf2](#) / [.app-bin] / [.bin] / [.elf] / [.map] / [Release notes]
[v1.20.0 \(2023-04-26\) .uf2](#) / [.bin] / [.elf] / [.map] / [Release notes]
[v1.19.1 \(2022-06-18\) .uf2](#) / [.bin] / [.elf] / [.map] / [Release notes]
[v1.18 \(2022-01-17\) .bin](#) / [.elf] / [.map] / [Release notes]

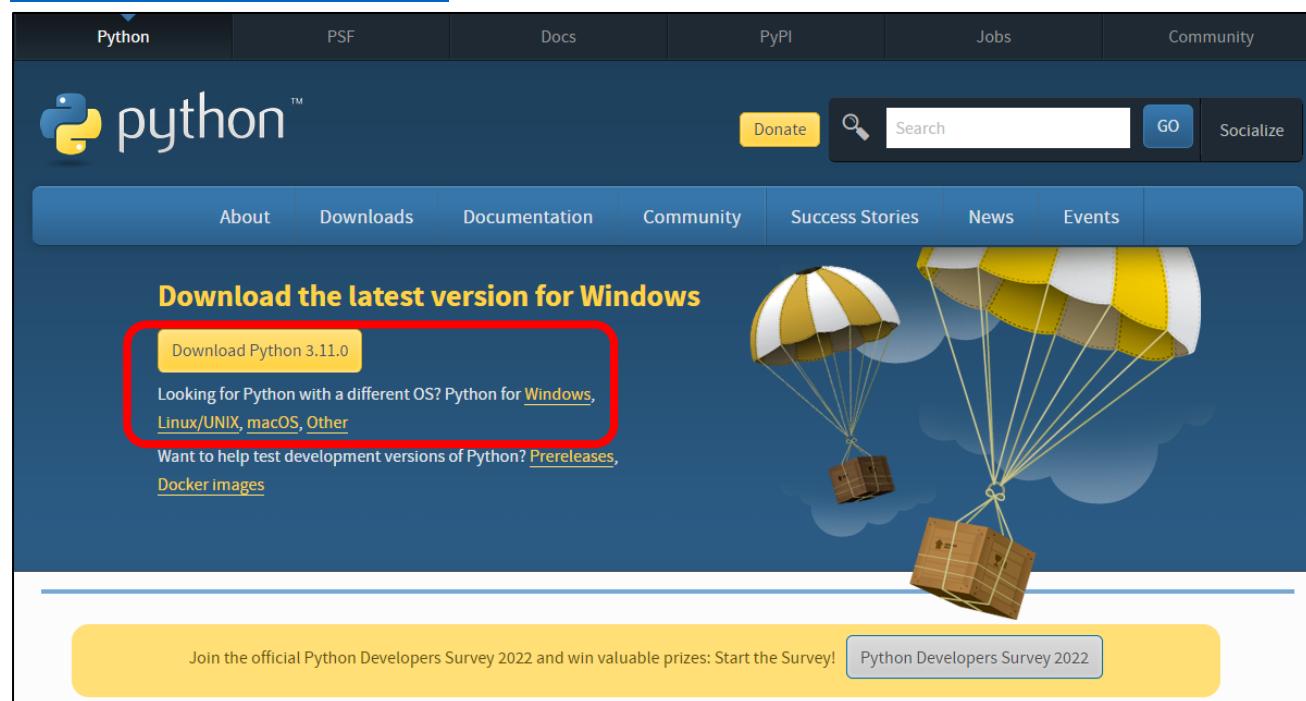
Any concerns? ✉ support@freenove.com

Firmware used in this tutorial is **ESP32_GENERIC_S3-20250809-v1.26.0.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/>

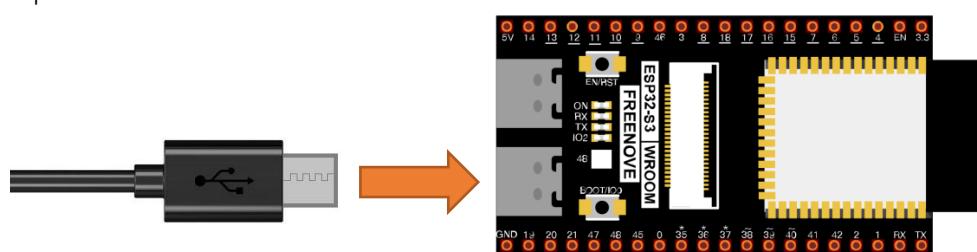


Please follow the official instructions to download and install.

Burning a Micropython Firmware

Window

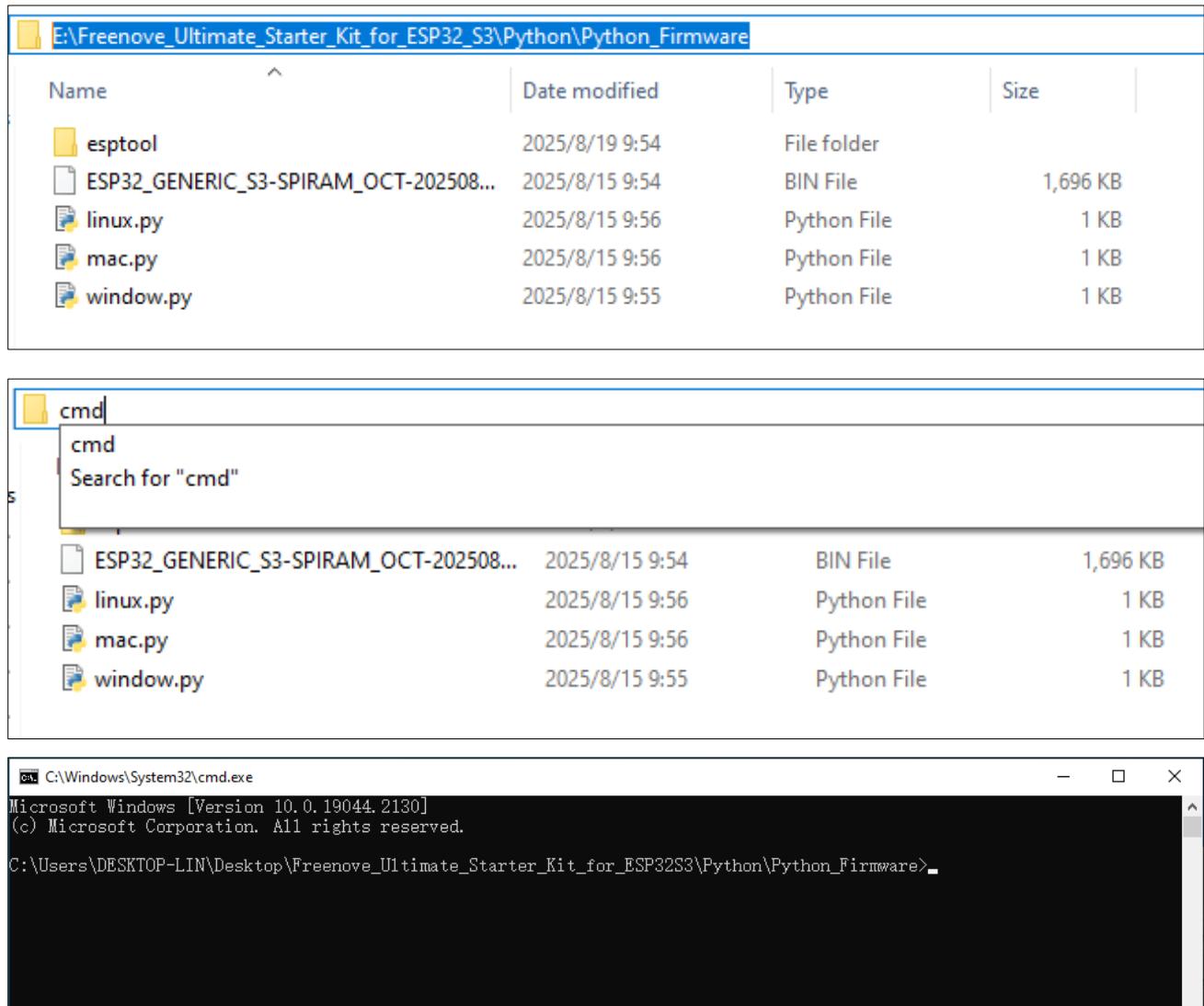
Connect your computer and ESP32-S3 with a USB cable.



Any concerns? ✉ support@freenove.com

Open Freenove_ESP32_S3_WROVER_Board/Python/Python_Firmware

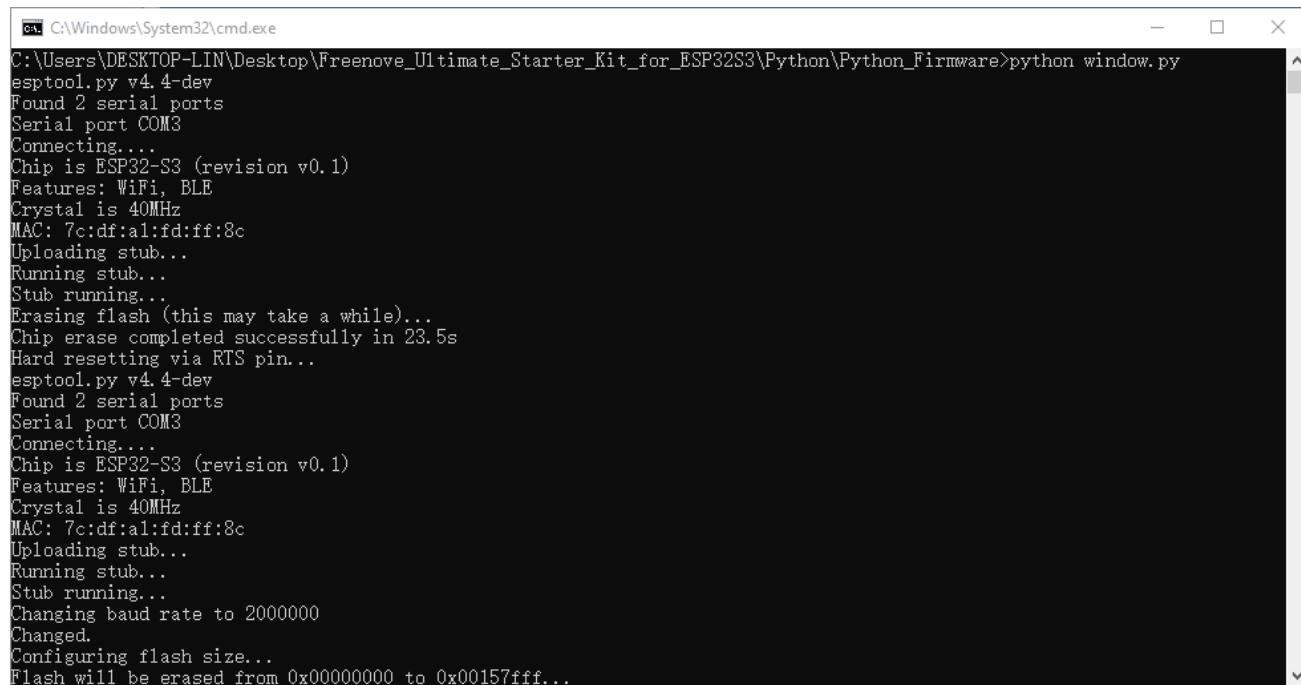
Enter cmd on path bar then press Enter.



Here my python3 version is 3.8.1.

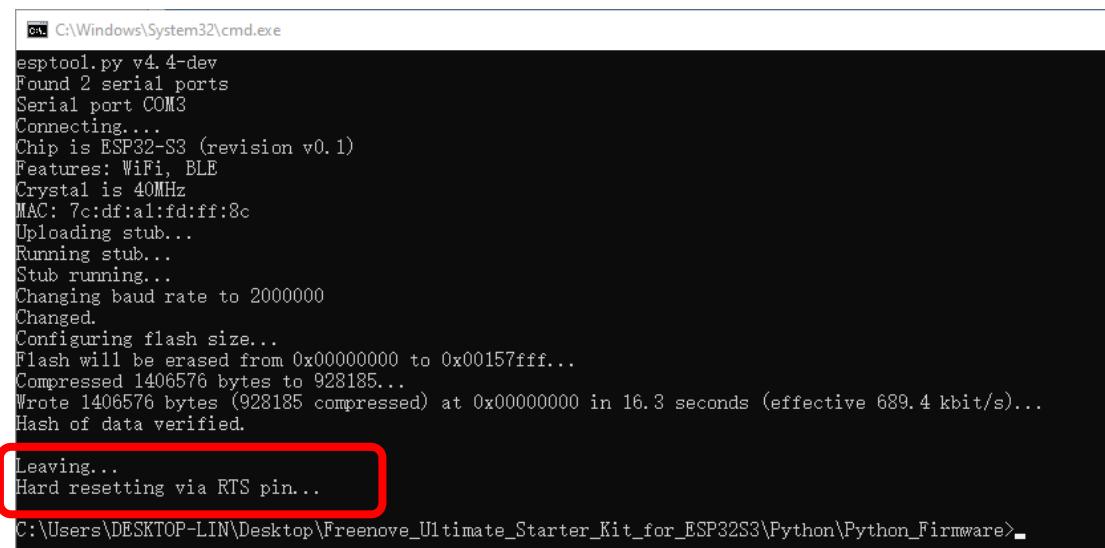


Enter “python window.py”. Micropython firmware will be automatically burned to ESP32S3.



```
C:\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:al: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:al: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 0x0015ffff...
```

As shown in the figure below after completion.



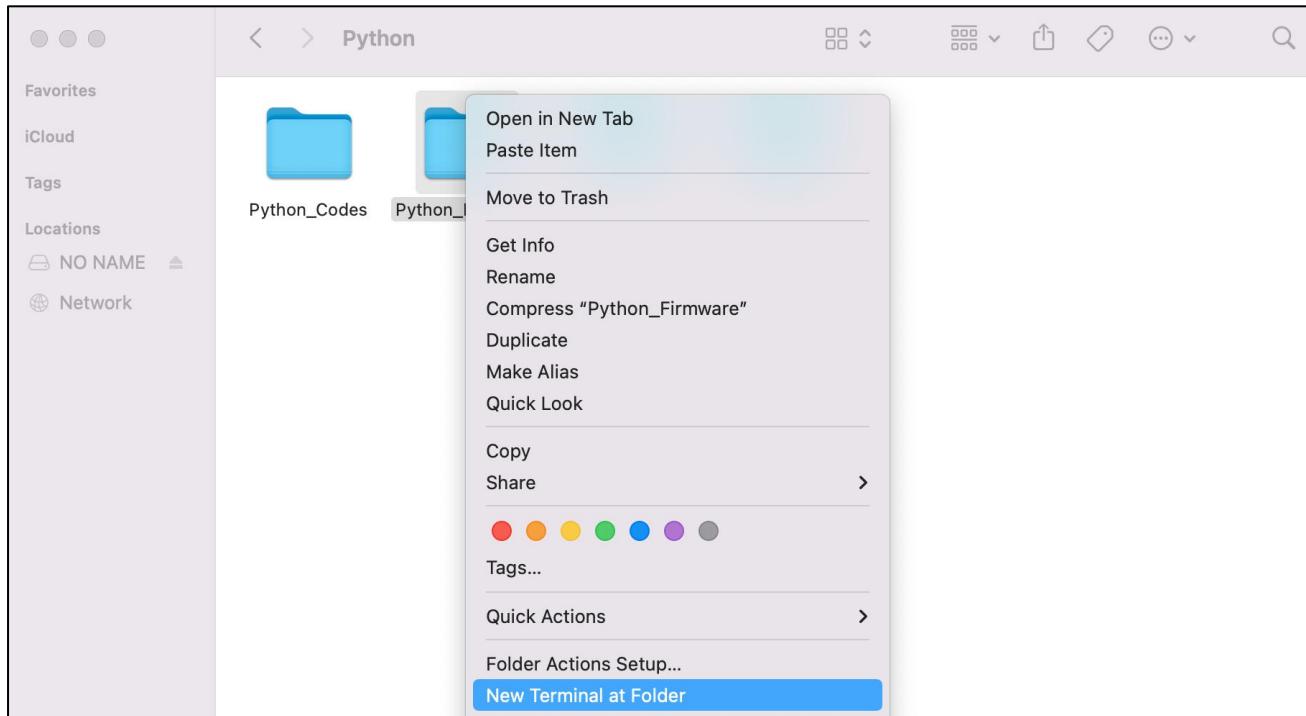
```
C:\Windows\System32\cmd.exe
esptool.py v4.4-dev
Found 2 serial ports
Serial port COM3
Connecting....
Chip is ESP32-S3 (revision v0.1)
Features: WiFi, BLE
Crystal is 40MHz
MAC: 7c:df:al:fd:ff:8c
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 2000000
Changed.
Configuring flash size...
Flash will be erased from 0x00000000 to 0x0015ffff...
Compressed 1406576 bytes to 928185...
Wrote 1406576 bytes (928185 compressed) at 0x00000000 in 16.3 seconds (effective 689.4 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...

C:\Users\DESKTOP-LIN\Desktop\Freenove_Ultimate_Starter_Kit_for_ESP32S3\Python\Python_Firmware>
```

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

```
Last login: Thu Nov  3 17:15:41 on ttys000
freenove@3c22fb61fad Python_Firmware % python --version
zsh: command not found: python
[freenove@3c22fb61fad Python_Firmware % python3 --version
Python 3.10.4
freenove@3c22fb61fad Python_Firmware % ]
```

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

```
Python_Firmware — Python < Python mac.py — 80x24
Last login: Thu Nov  3 17:16:12 on ttys000
[freenove@3c22fb61fad Python_Firmware % python --version
zsh: command not found: python
[freenove@3c22fb61fad Python_Firmware % python3 --version
Python 3.10.4
[freenove@3c22fb61fad Python_Firmware % python3 mac.py
esptool.py v4.4-dev
Found 3 serial ports
Serial port /dev/cu.wchusbserial54ED0454161
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)...
```

After completion, it is shown below.

```
Python_Firmware — -zsh — 80x24
Hard resetting via RTS pin...
esptool.py v4.4-dev
Found 3 serial ports
Serial port /dev/cu.wchusbserial54ED0454161
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...
Compressed 1406576 bytes to 928185...
Wrote 1406576 bytes (928185 compressed) at 0x00000000 in 16.4 seconds (effective
 688.2 kbit/s)...
Hash of data verified.

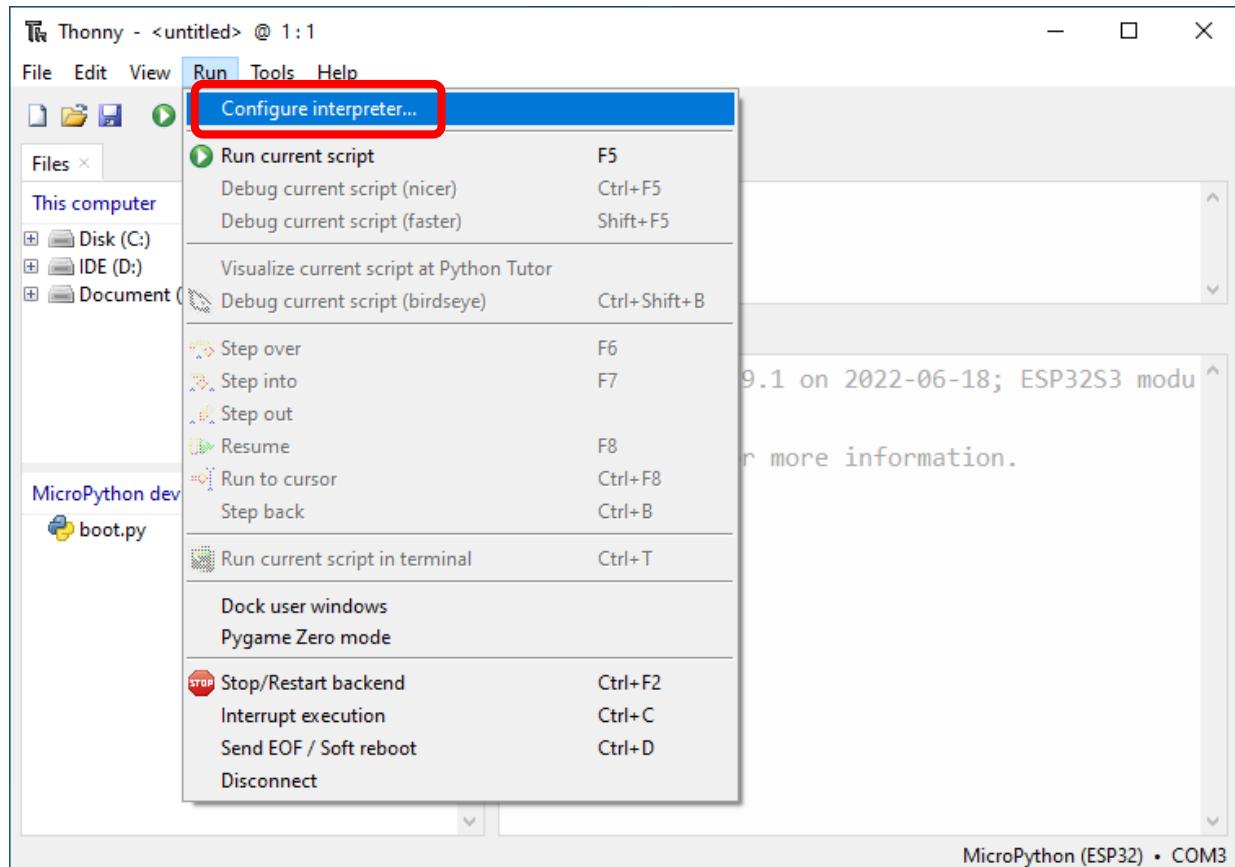
Leaving...
Hard resetting via RTS pin...
freenove@3c22fb61fad Python_Firmware %
```

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

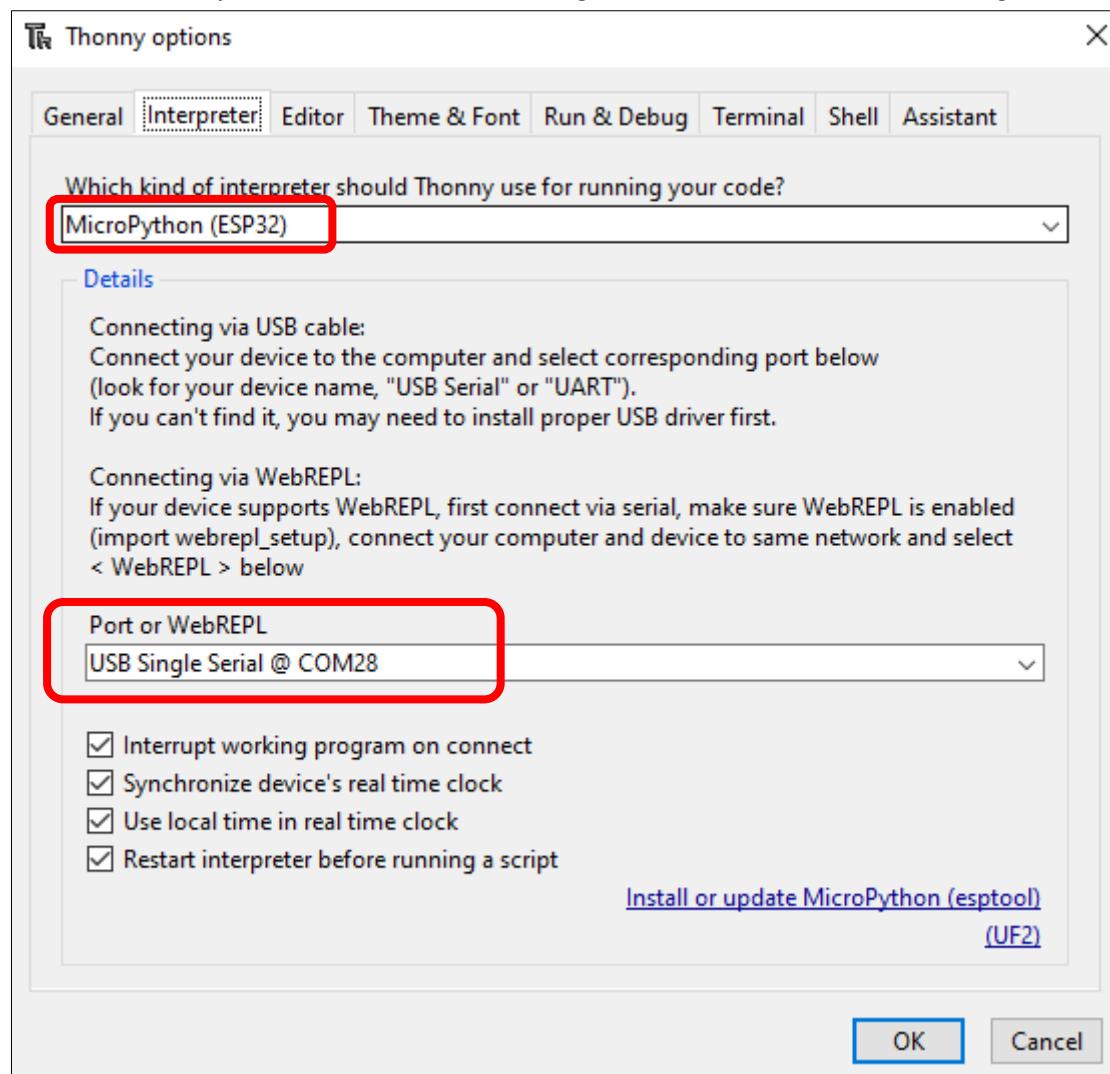
Testing codes (Important)

Testing Shell Command

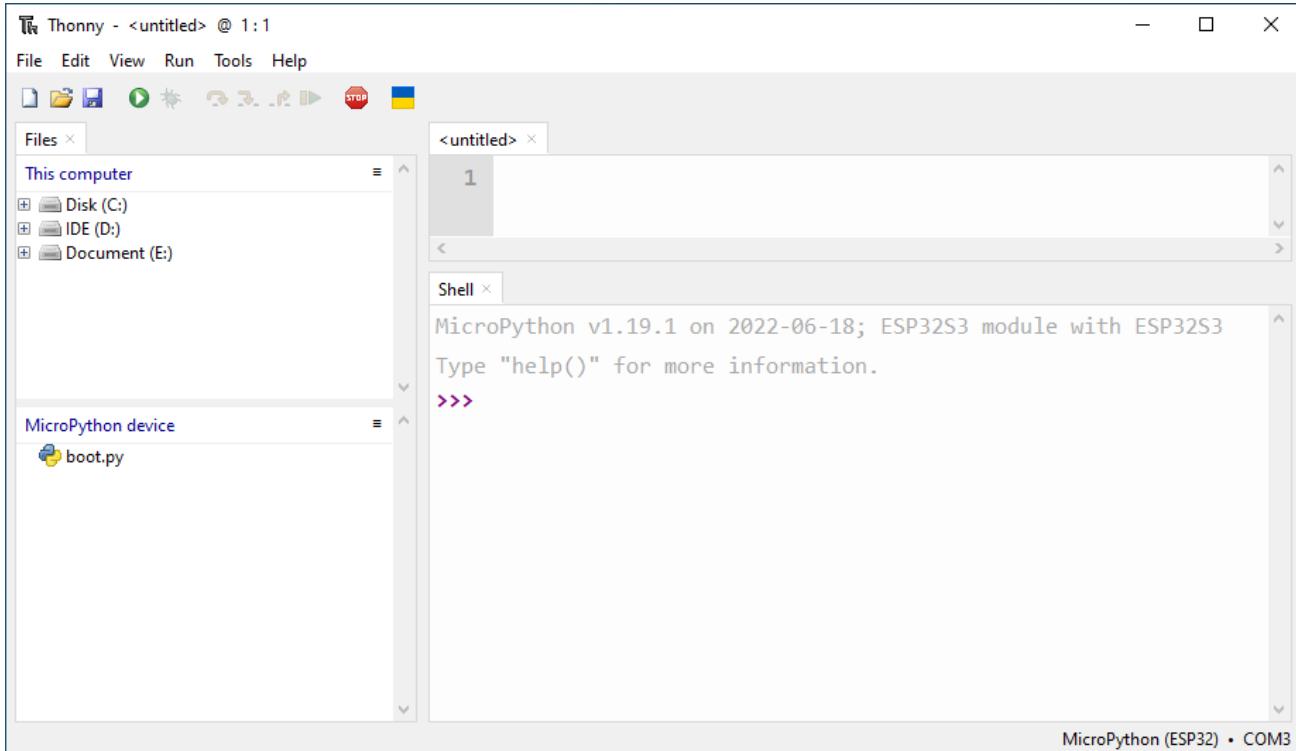
Make sure that the ESP 32S3 has burned the firmware and is connected to the computer through the data cable. Run Thonny. Click Run and select Configure interpreter.



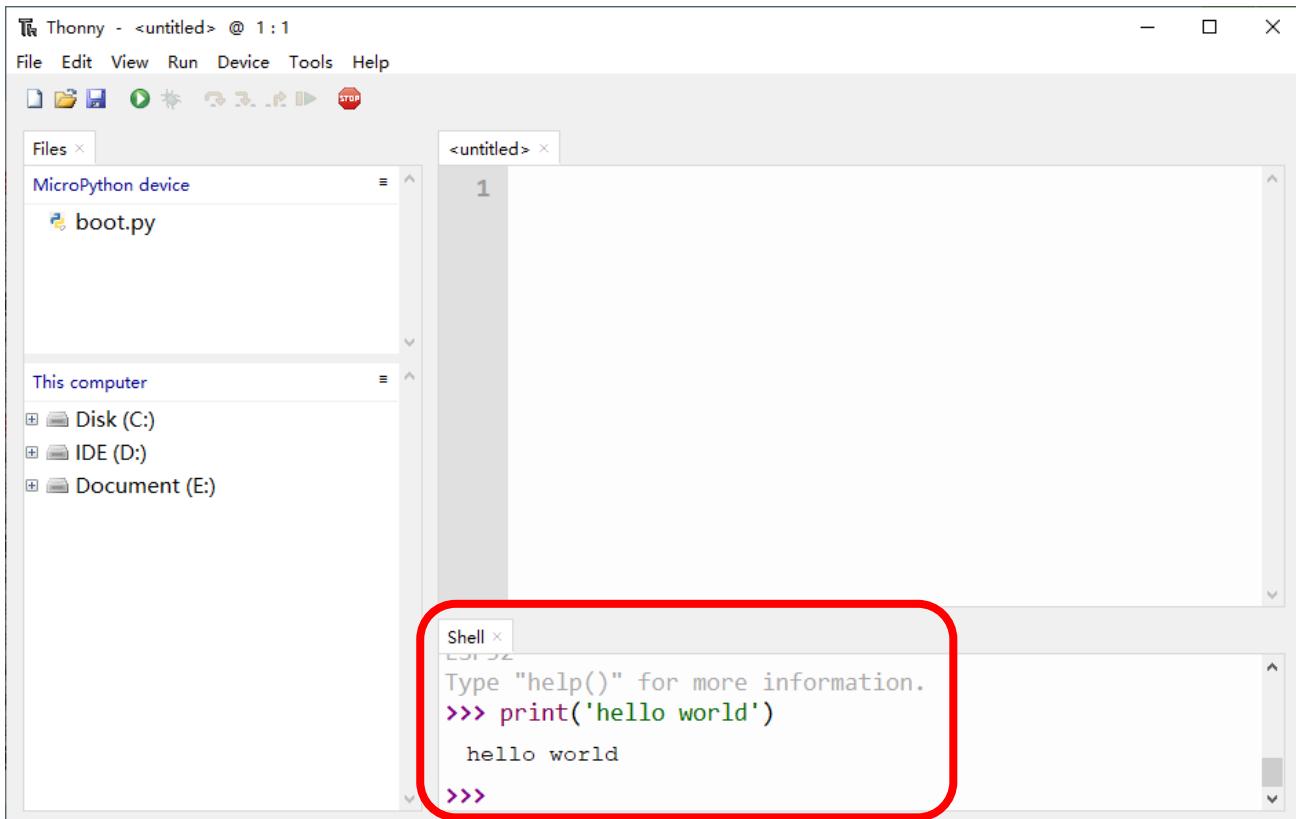
Please configure according to the following figure. Note that the port numbers of USB Enhanced SERIAL may be different for different systems. Please select according to the actual situation. After configuration, click OK.



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



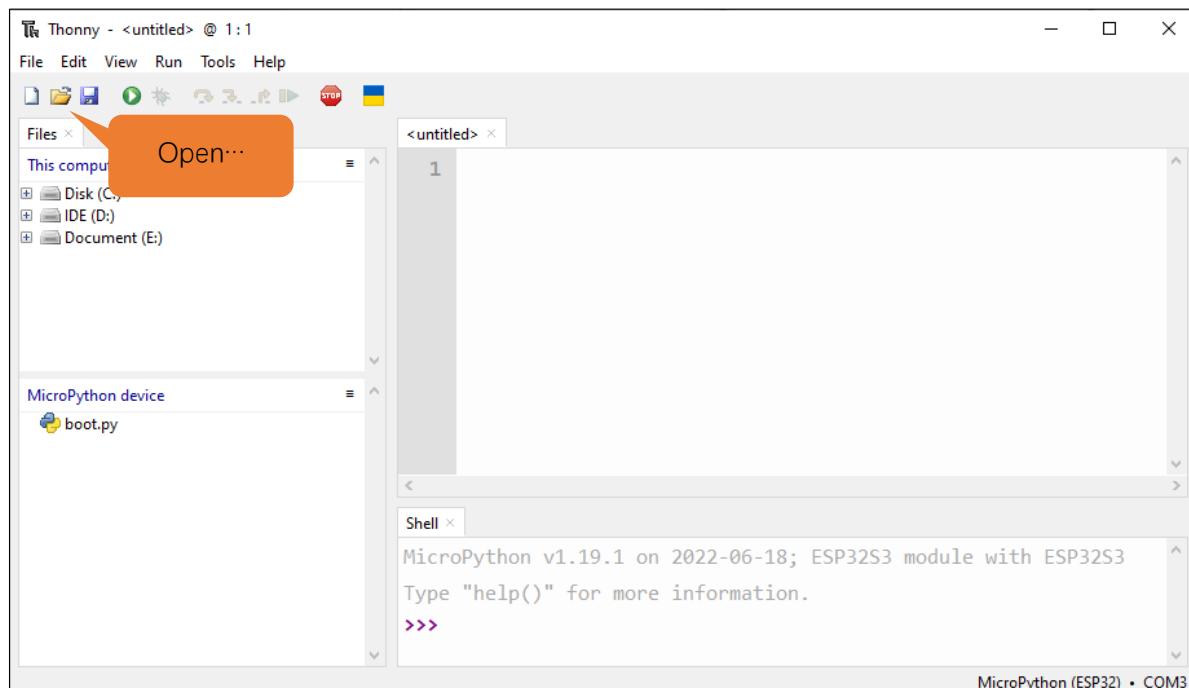
Enter "print('hello world')" in "Shell" and press Enter.



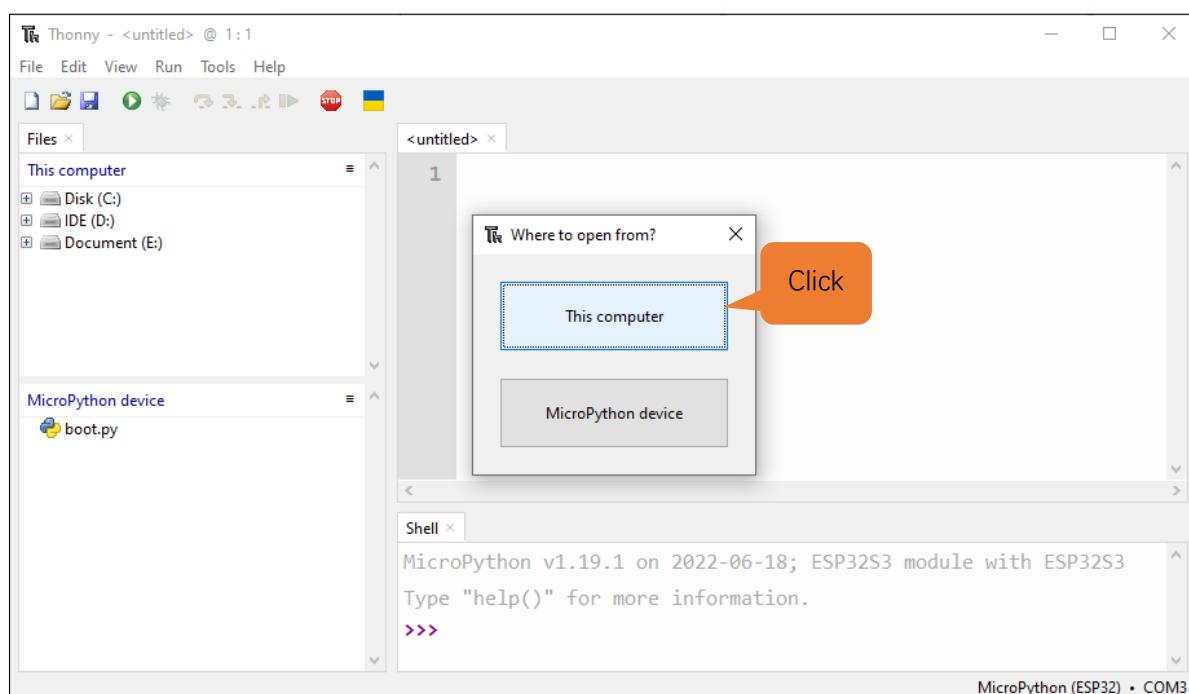
Running Online

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

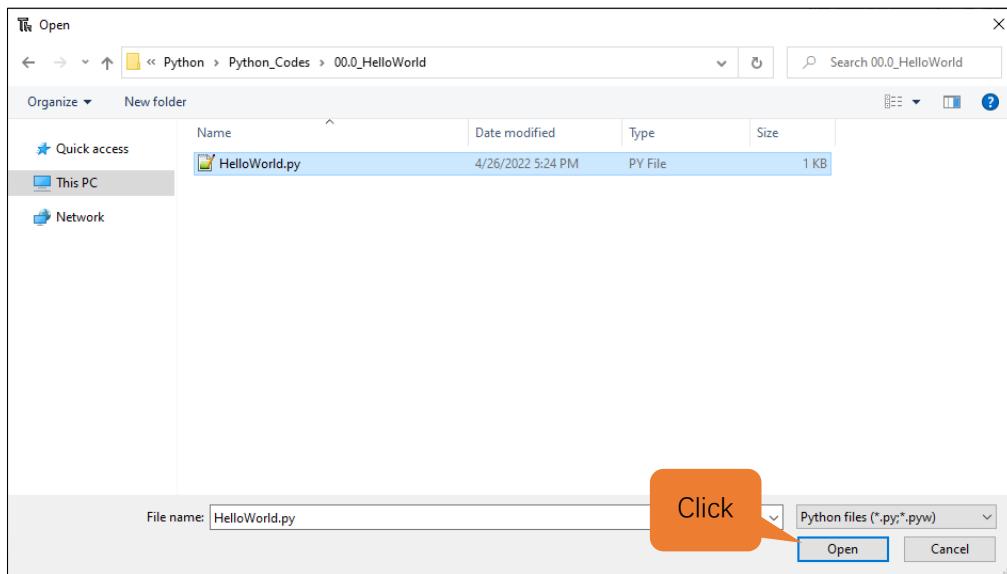
1. Open Thonny and click “Open…”.



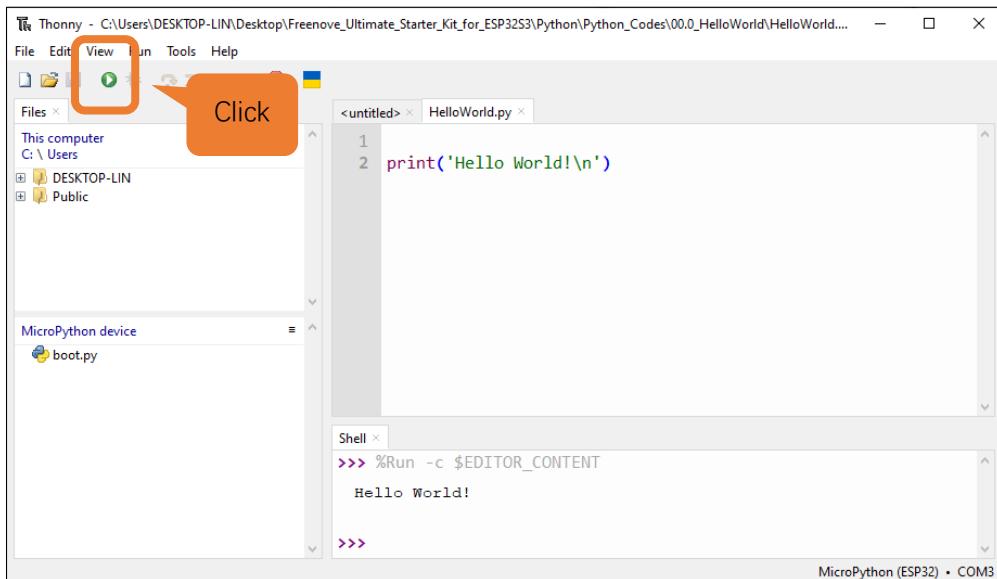
2. On the newly pop-up window, click “This computer”.



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”.

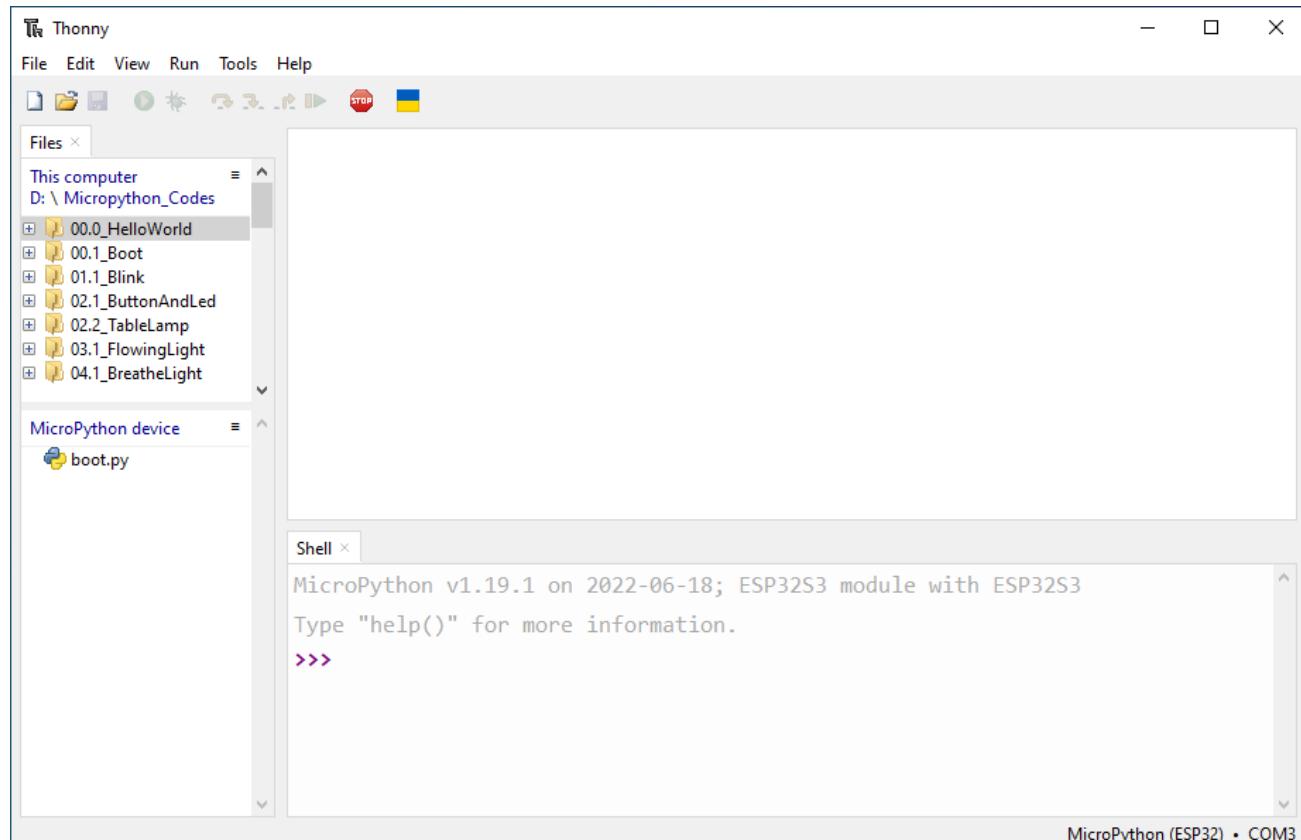


Note: When running online, if you press the reset key of ESP32S3, user's code will not be executed again. If you wish to run the code automatically after resetting the code, please refer to the following [Running Offline](#).

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.

The screenshot shows the Thonny IDE interface. The left sidebar displays a file tree under "This computer" and a "MicroPython device" section. The main area shows the code for `boot.py`:

```

1  #!/opt/bin/lv_micropython
2  import uos as os
3  import uerrno as errno
4  iter = os.ilistdir()
5  IS_DIR = 0x4000
6  IS_REGULAR = 0x8000
7
8  while True:
9      try:
10         entry = next(iter)
11         filename = entry[0]
12         file_type = entry[1]
13         if filename == 'boot.py':
14             continue
15         else:
16             print("====")
17             print(filename,end="")
18             if file_type == IS_DIR:
19                 print(", File is a directory")
20             print("====")

```

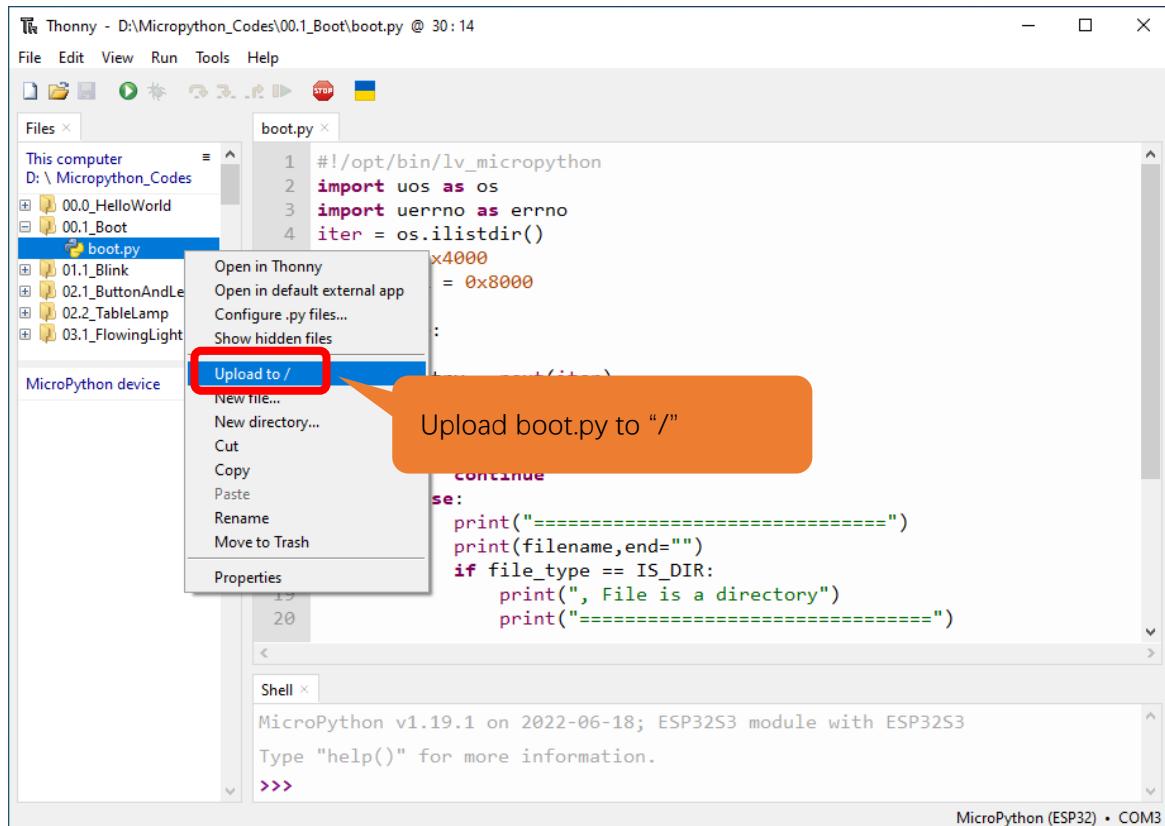
The "Shell" tab at the bottom shows the MicroPython environment:

```

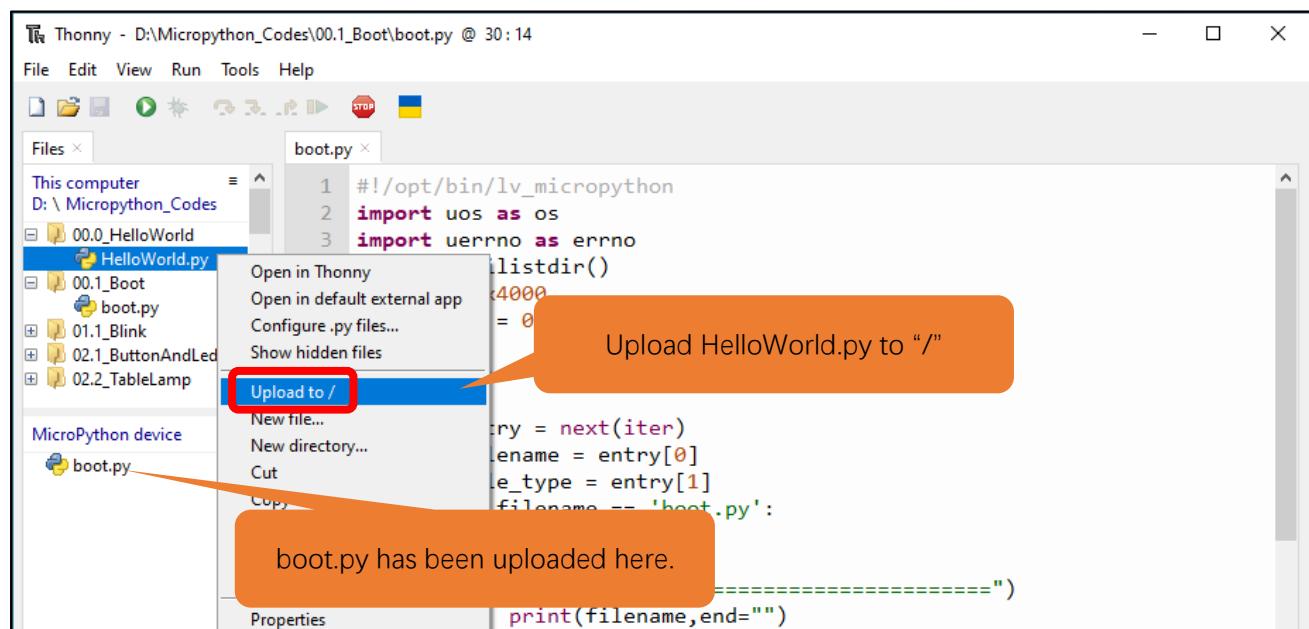
MicroPython v1.19.1 on 2022-06-18; ESP32S3 module with ESP32S3
Type "help()" for more information.
>>>

```

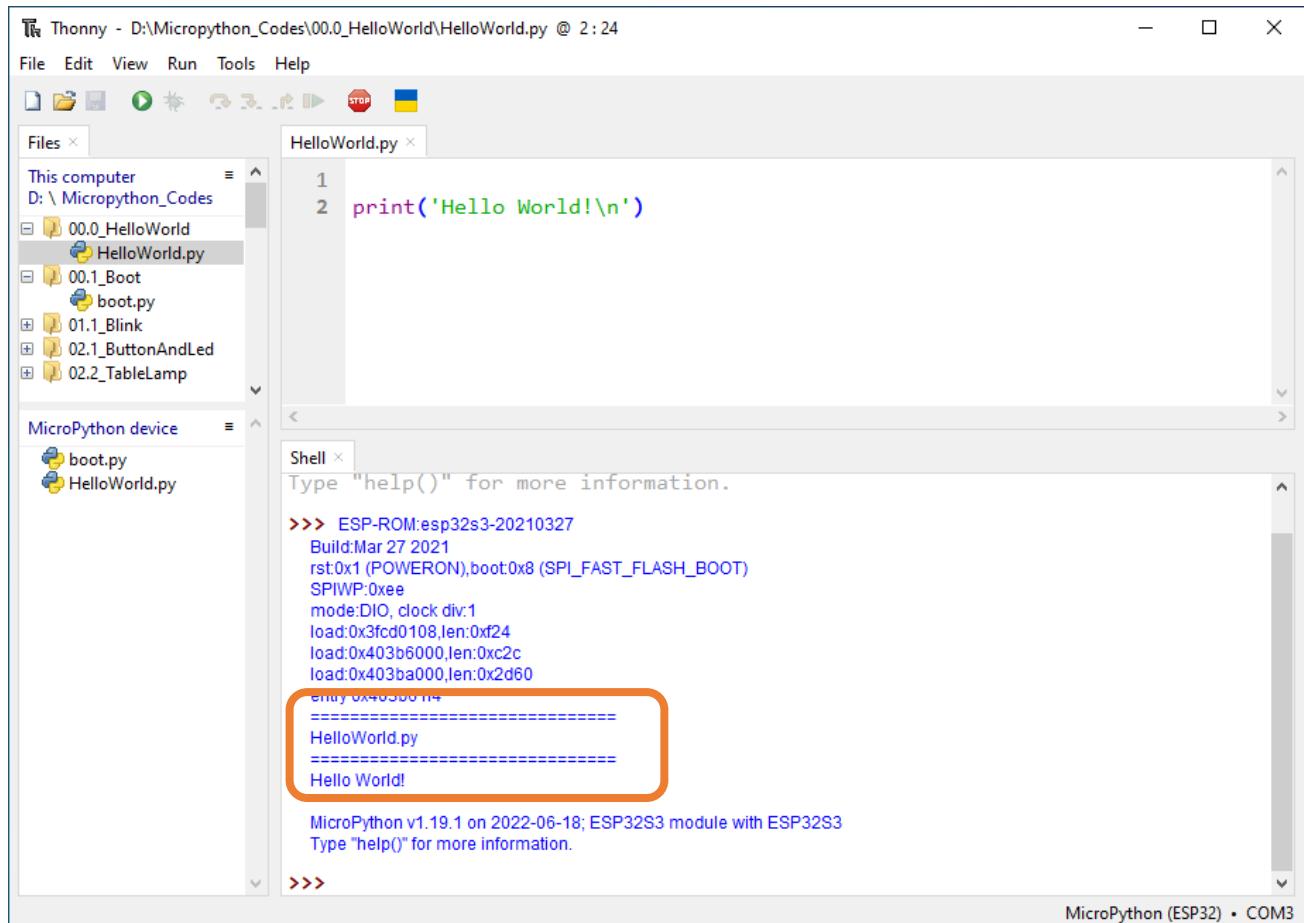
If you want your written programs to run offline, you need to upload `boot.py` we provided and all your codes to “MicroPython device” and press ESP32S3’s reset key. Here we use programs 00.0 and 00.1 as examples. Select “boot.py”, right-click to select “Upload to /”.



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.

The screenshot shows the Thonny IDE interface. In the top menu bar, it says "Thonny - D:\Micropython_Codes\00.1_Boot\boot.py @ 30:14". The "File" menu is highlighted. Below the menu is a toolbar with icons for file operations like Open, Save, Run, and Stop. The main window has two panes: "Files" on the left and "Shell" on the right. The "Files" pane shows a tree view of files on "This computer" (D:\Micropython_Codes) and a list of files on the "MicroPython device". A "boot.py" file is selected in the device list, and its contents are displayed in the "boot.py" editor pane. The code in the editor is:

```
1 #!/opt/bin/lv_micropython
2 import uos as os
3 import uerrno as errno
4 iter = os.ilistdir()
5 IS_DIR = 0x4000
6 IS_REGULAR = 0x8000
7
8 while True:
9     try:
10         entry = next(iter)
11         filename = entry[0]
12         file_type = entry[1]
13         if filename == 'boot.py':
14             continue
15         else:
16             print("====")
17             print(filename,end="")
18             if file_type == IS_DIR:
19                 print(", File is a directory")
20                 print("====")
21             else:
```

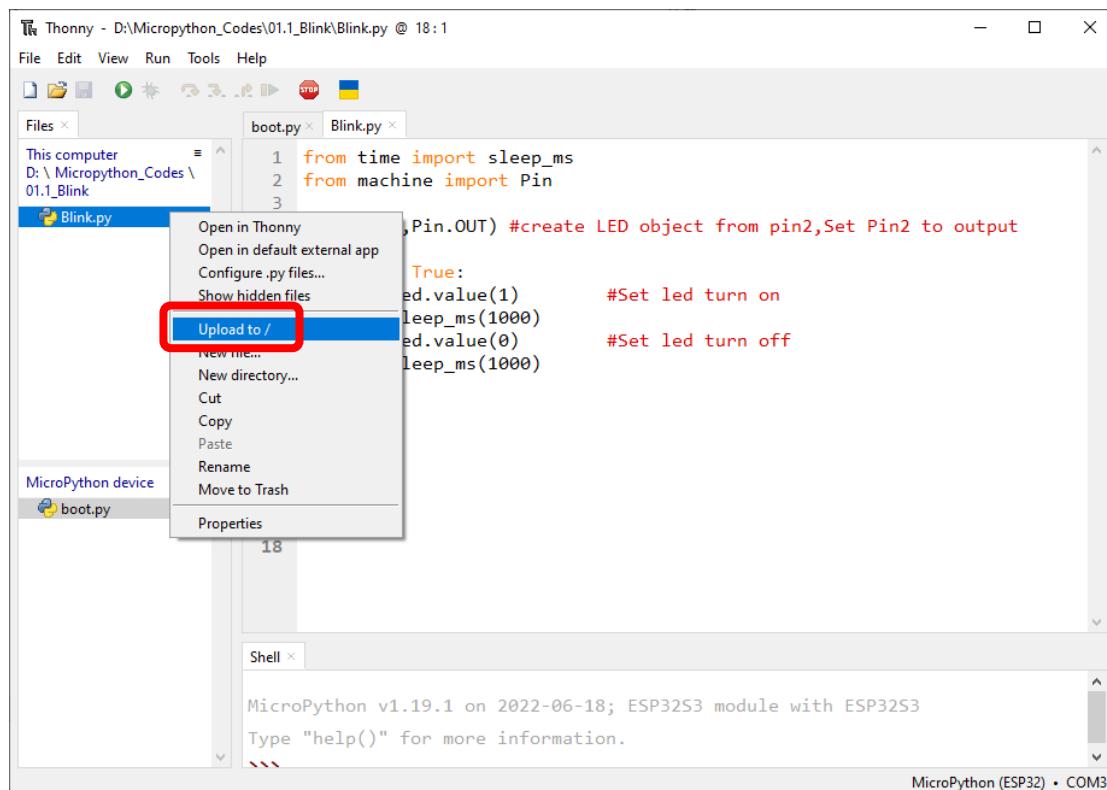
A callout bubble from the "boot.py" file in the "Files" pane points to the "boot.py" file in the "MicroPython device" list. Another callout bubble from the "boot.py" file in the "Files" pane contains the text: "Codes in ESP32S3's root directory will be executed automatically."

In the "Shell" pane, the MicroPython version and connection information are shown:

```
MicroPython v1.19.1 on 2022-06-18; ESP32S3 module with ESP32S3
Type "help()" for more information.
>>>
```

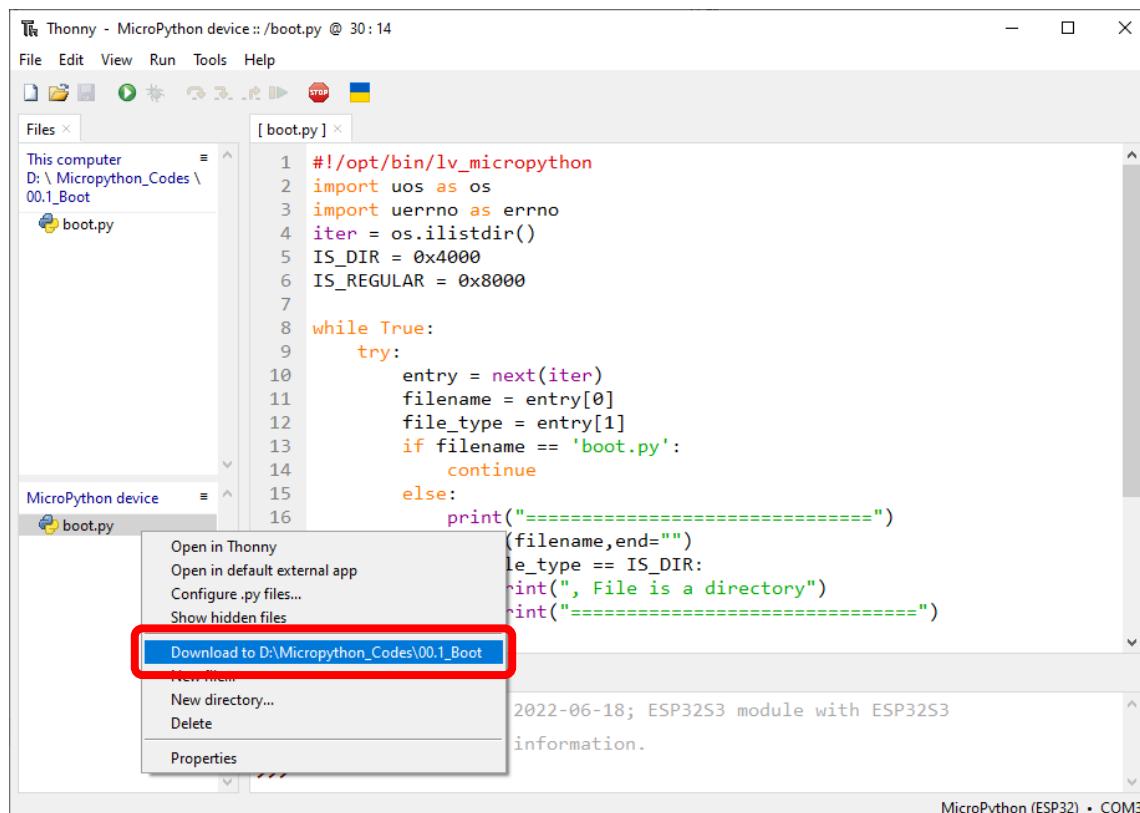
At the bottom right of the shell pane, it says "MicroPython (ESP32) • COM3".

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

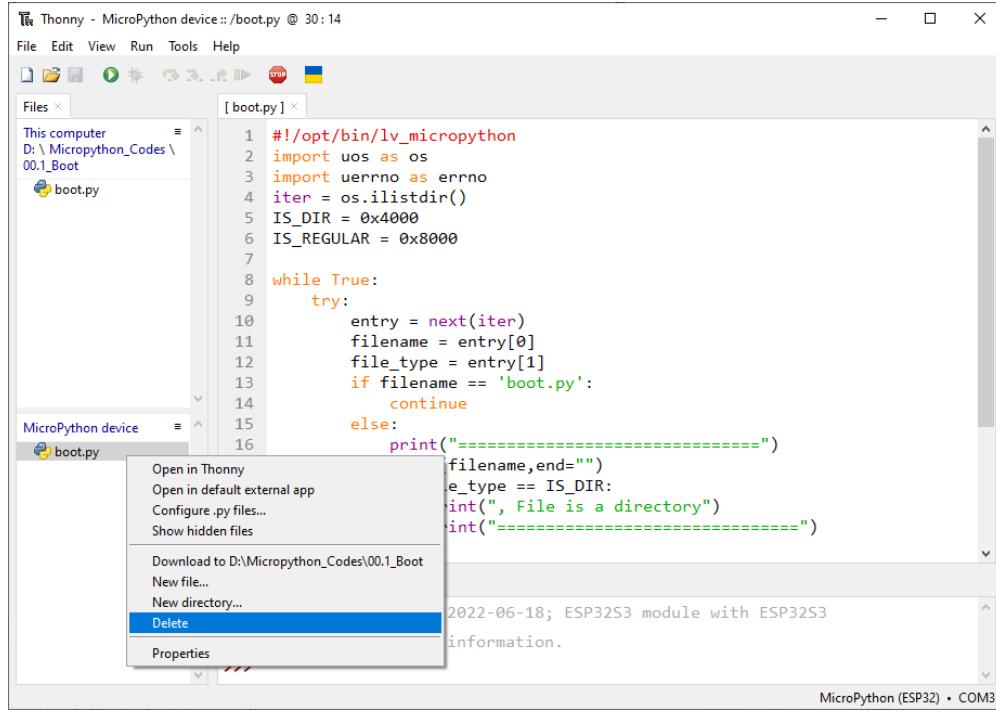
Select “boot.py” in “MicroPython device”, right-click to select “Download to ...” to download the code to your computer.



Any concerns? ✉ support@freenove.com

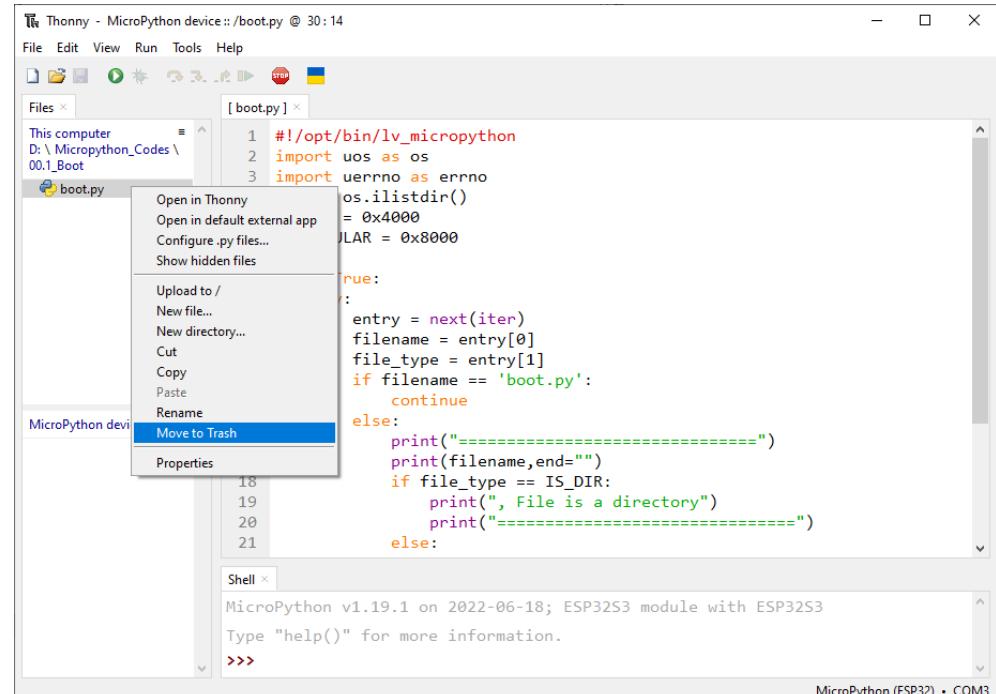
Deleting Files from ESP32S3's Root Directory

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

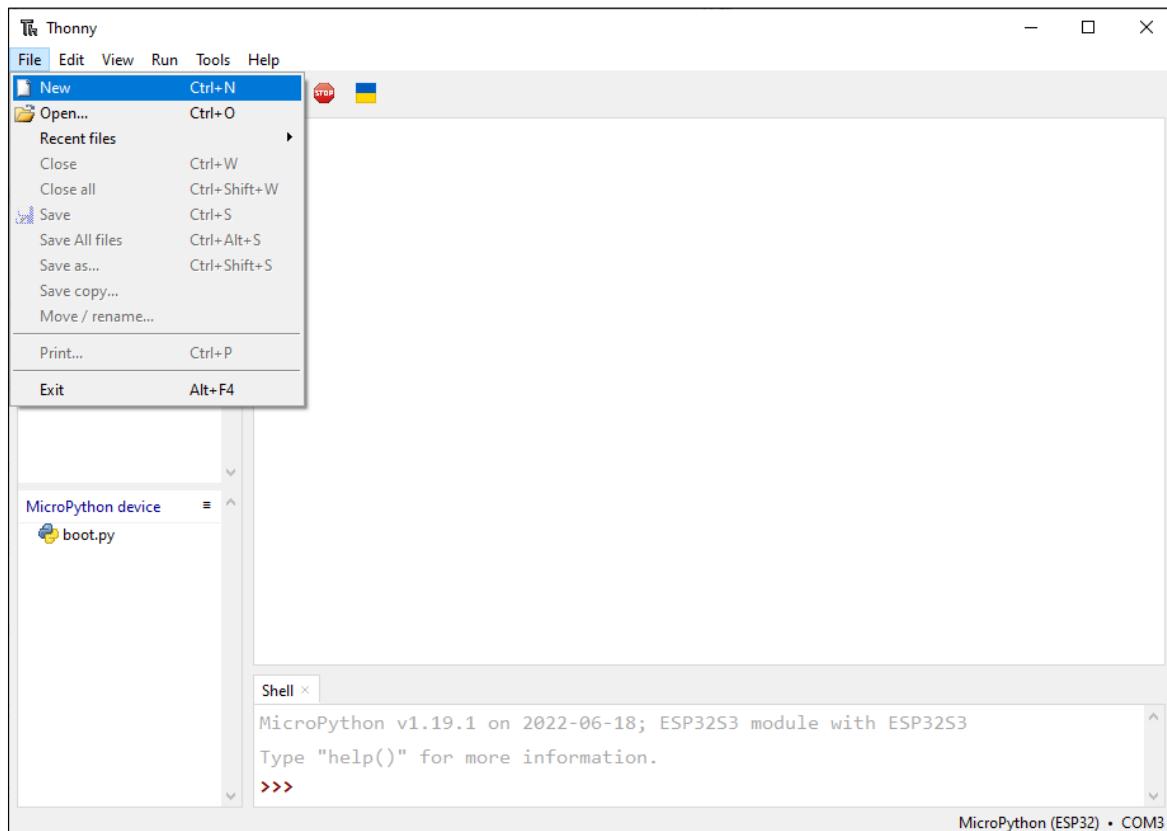
Select “boot.py” in “00.1_Boot”, right-click it and select “Move to Recycle Bin” to delete it from “00.1_Boot”.



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

Creating and Saving the code

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

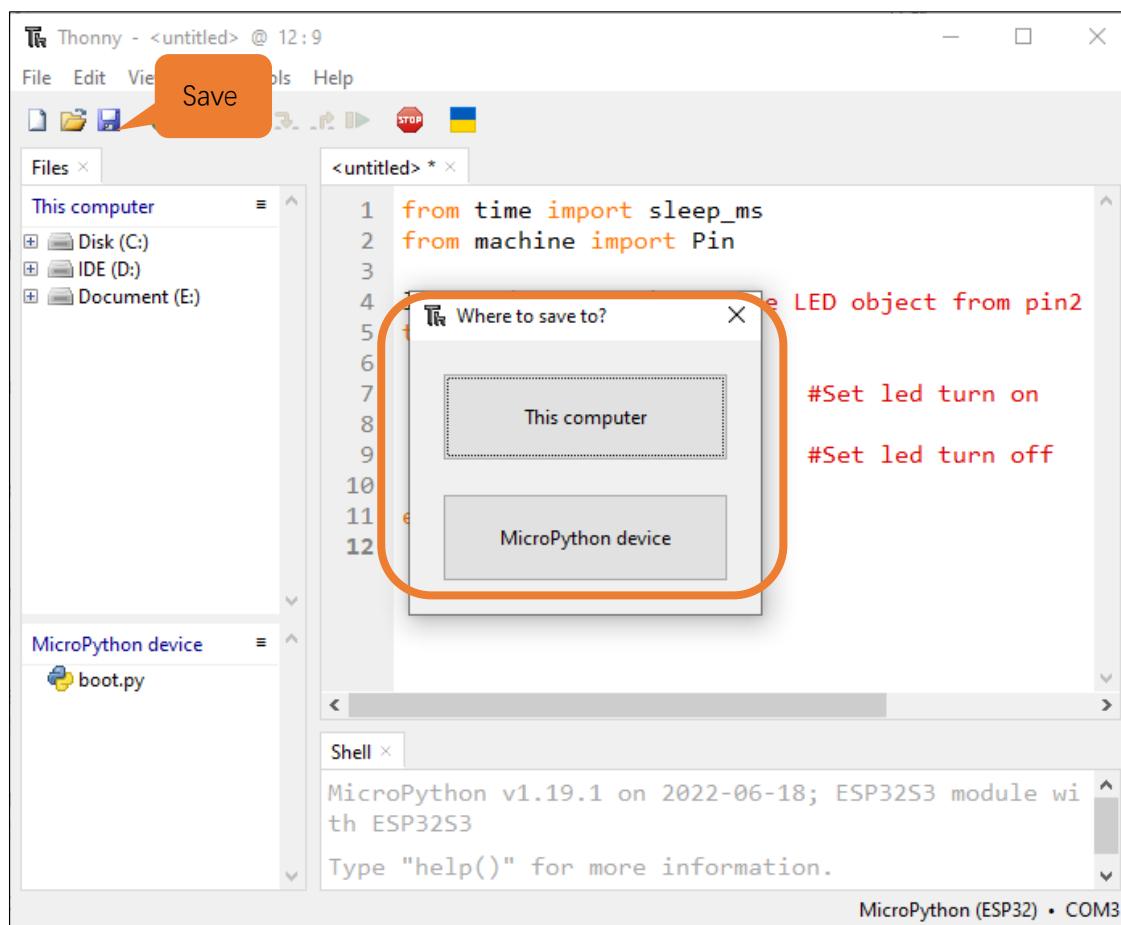


Enter codes in the newly opened file. Here we use codes of "01.1_Blink.py" as an example.

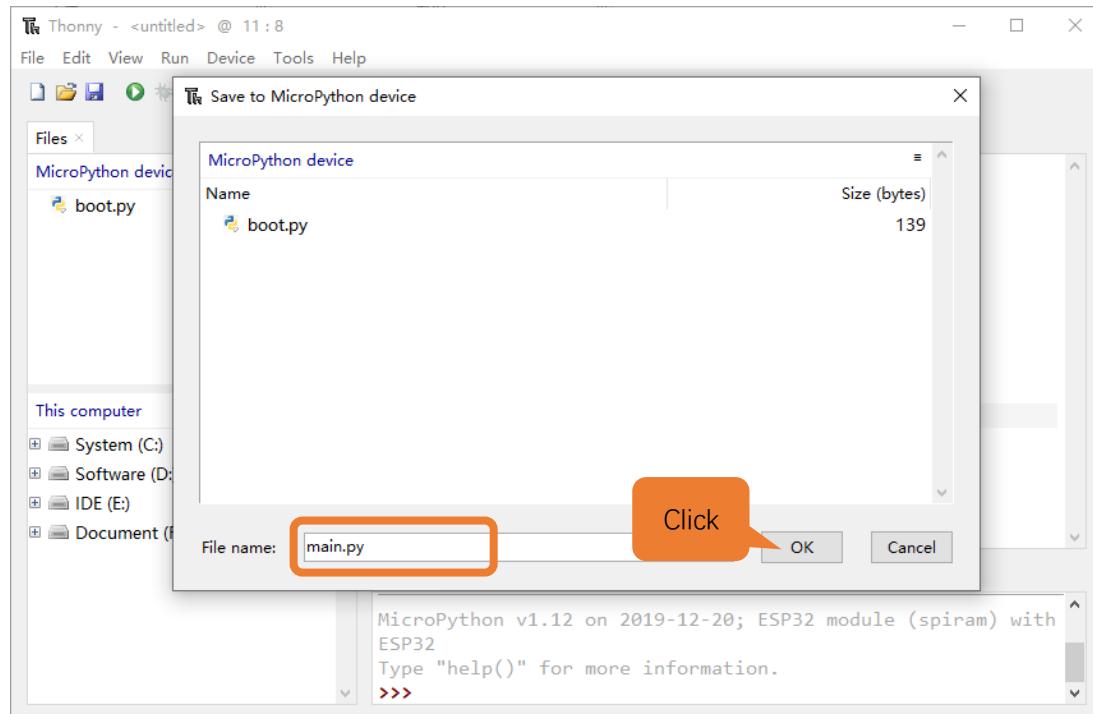
```
from time import sleep_ms
from machine import Pin

led=Pin(2,Pin.OUT) #create LED object from pin2,Set Pin2 to output
try:
    while True:
        led.value(1)          #Set led turn on
        sleep_ms(1000)
        led.value(0)           #Set led turn off
        sleep_ms(1000)
except:
    pass
```

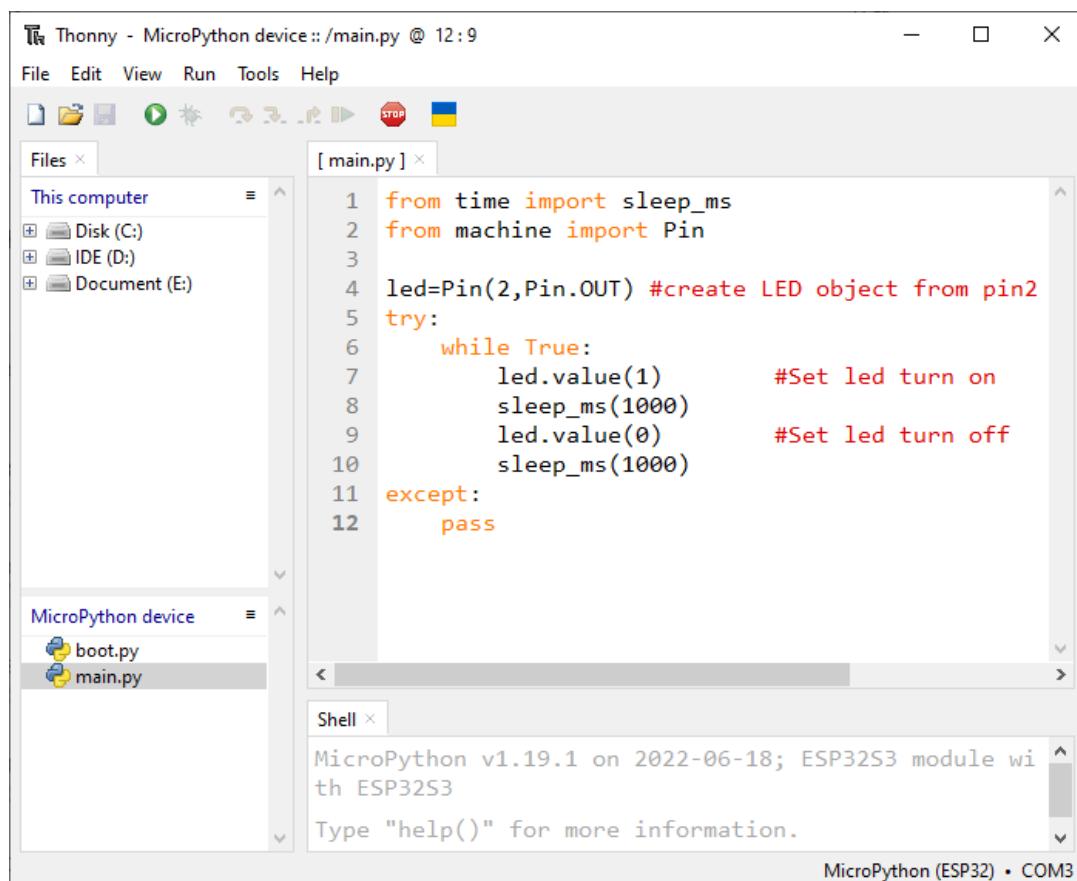
Click “Save” on the menu bar. You can save the codes either to your computer or to ESP32S3.



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.

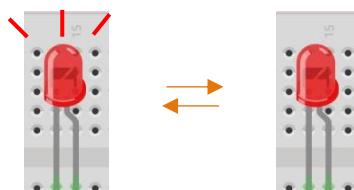


The screenshot shows the Thonny IDE interface. The title bar says "Thonny - MicroPython device :: /main.py @ 12:9". The menu bar includes File, Edit, View, Run, Tools, Help. The toolbar has icons for file operations and a stop button. The left sidebar shows "Files" with "This computer" and "MicroPython device" sections. The "MicroPython device" section lists "boot.py" and "main.py". The main area displays the Python code for a LED blink script:

```
1 from time import sleep_ms
2 from machine import Pin
3
4 led=Pin(2,Pin.OUT) #create LED object from pin2
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
```

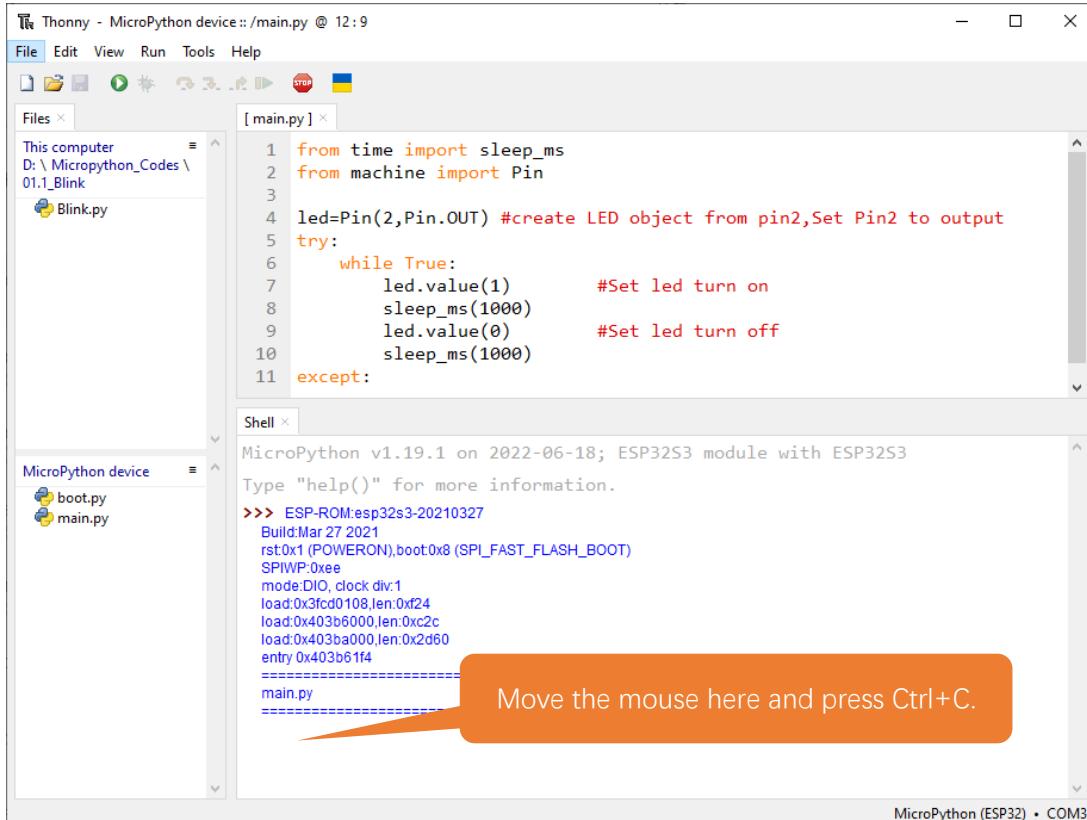
The bottom right shows "MicroPython (ESP32) • COM3".

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#
SPIID	DQ0
SPIQ	DQ1
SPIWP	DQ2
SPIHD	DQ3
GPIO33	DQ4
GPIO34	DQ5
GPIO35	DQ6
GPIO36	DQ7
GPIO37	DQS/DM

SDcard Pin

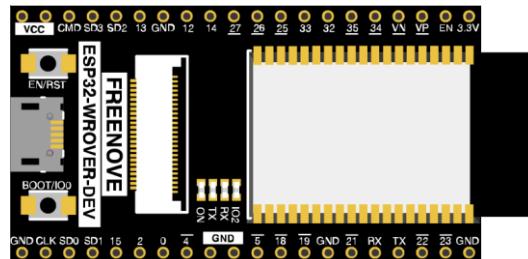
An SD card slot is integrated on the back of the ESP32-S3 WROOM board. We can use GPIO38-GPIO40 of ESP32-S3 WROOM to drive the SD card.

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_VSYNC	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.

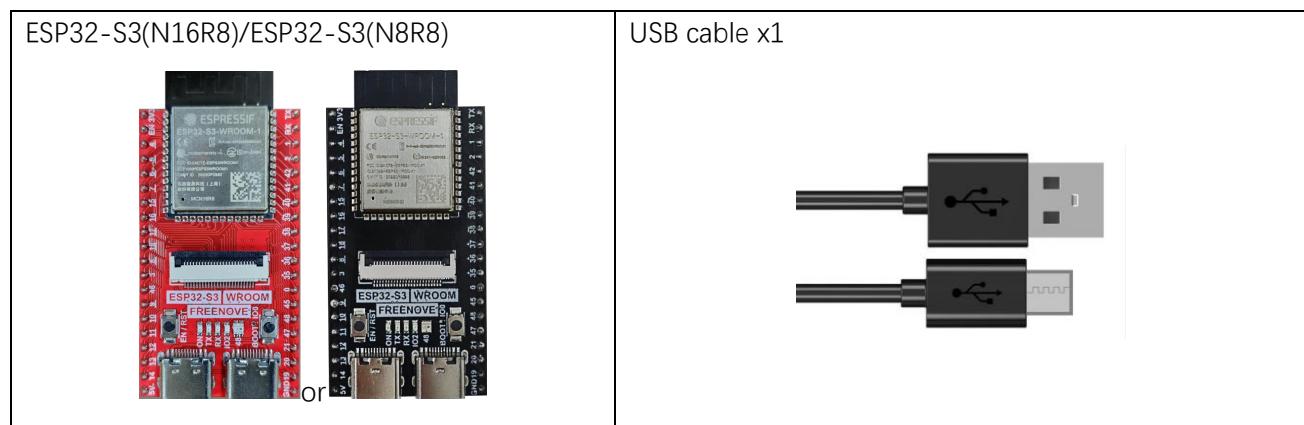
Chapter 1 LED (Important)

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

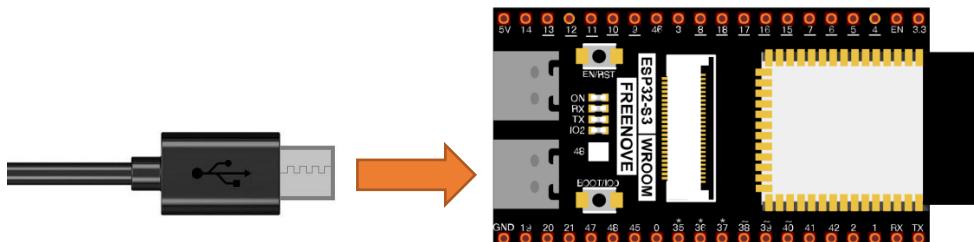
In this project, we will use ESP32-S3 WROOM to control blinking a common LED.

Component List



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 (including EXT 3.3V) on the extension board are provided by ESP32-S3 WROOM.

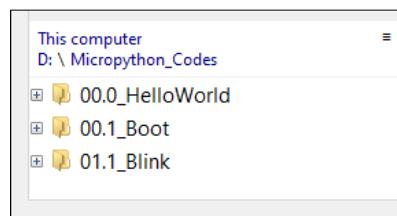
We can also use DC jack of extension board to power ESP32-S3 WROOM. In this way, 5v and EXT 3.3v on extension board are provided by external power resource.

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.

```

Thonny - D:\Micropython_Codes\01.1_Blink\Blink.py @ 18:1
File Edit View Run Tools Help
Blink.py x
This computer D:\ Micropython 01.1_Blink
Blink.py
import sleep_ms
from machine import Pin
Pin(2, Pin.OUT) #create LED object from pin2, Set Pin2 to output
try:
    while True:
        led.value(1)      #Set led turn on
        sleep_ms(1000)
        led.value(0)      #Set led turn off
        sleep_ms(1000)
except:
    pass

```

MicroPython device

boot.py

Shell

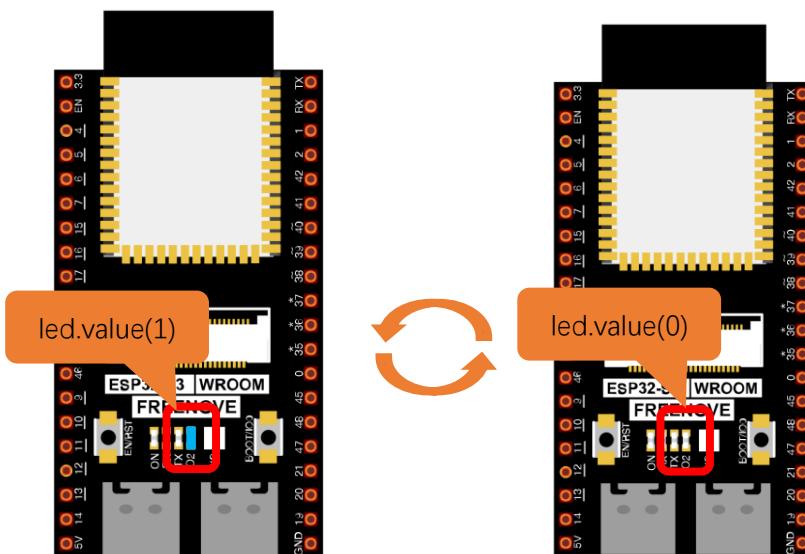
MicroPython v1.19.1 on 2022-06-18; ESP32S3

Type "help()" for more information.

>>>

MicroPython (ESP32) • COM3

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.

The screenshot shows the Thonny IDE interface. In the top menu bar, it says "Thonny - D:\Micropython_Codes\01.1_Blink\Blink.py @ 15:1". Below the menu is a toolbar with icons for file operations and device connection. The left sidebar has a "Files" tab showing "This computer" and "D:\Micropython_Codes\01.1_Blink", with "Blink.py" selected. The main area is titled "Blink.py" and contains the following 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

```

Below the code editor is a "Shell" tab showing the MicroPython prompt and an error message:

```

MicroPython v1.18 on 2022-01-17; ESP32 module (spiram) with ESP32
Type "help()" for more information.

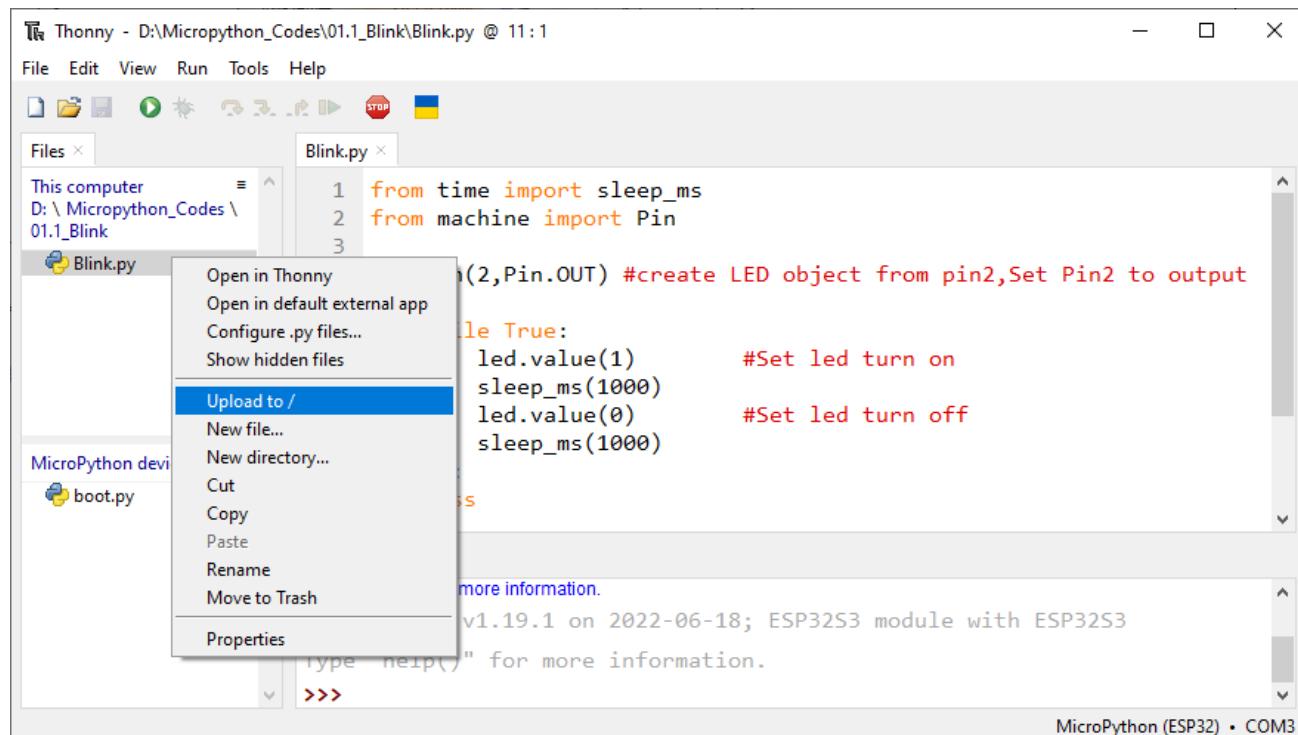
>>>
Connection lost (GetOverlappedResult failed (PermissionError(13, 'Access is denied.', None, 5)))

Use Stop/Restart to reconnect.

```

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.

Make sure you have uploaded Blink.py and boot.py here,

```

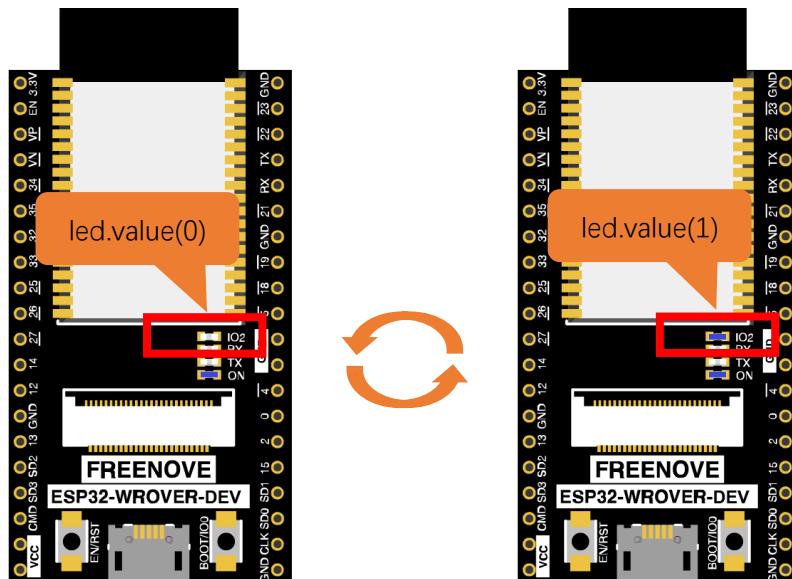
1 from time import sleep_ms
2 from machine import Pin
3
4 led=Pin(2,Pin.OUT) #create LED object from pin2,Set Pin2 to output
5 try:
6     while True:
7         led.value(1)           #Set led turn on
8         sleep_ms(1000)
9         led.value(0)          #Set led turn off
10        sleep_ms(1000)

```

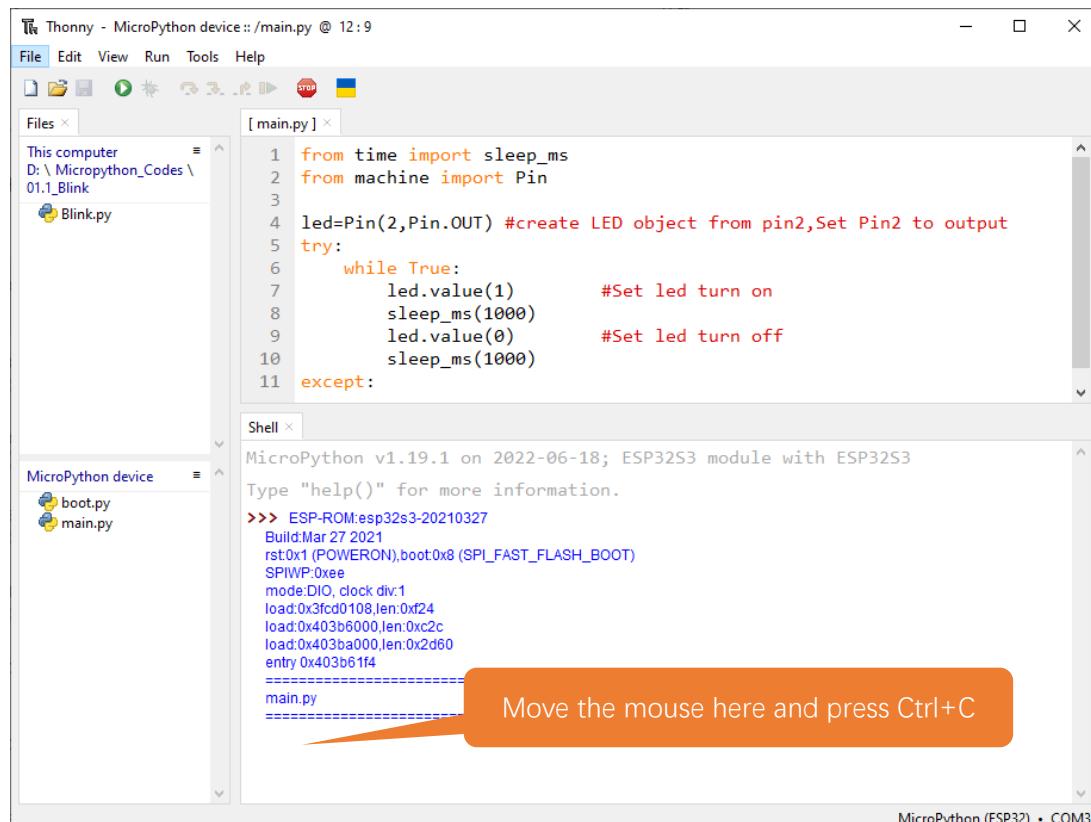
MicroPython v1.19.1 on 2022-06-18; ESP32S3 module with ESP32S3
Type "help()" for more information.
>>>

MicroPython (ESP32) • COM3

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

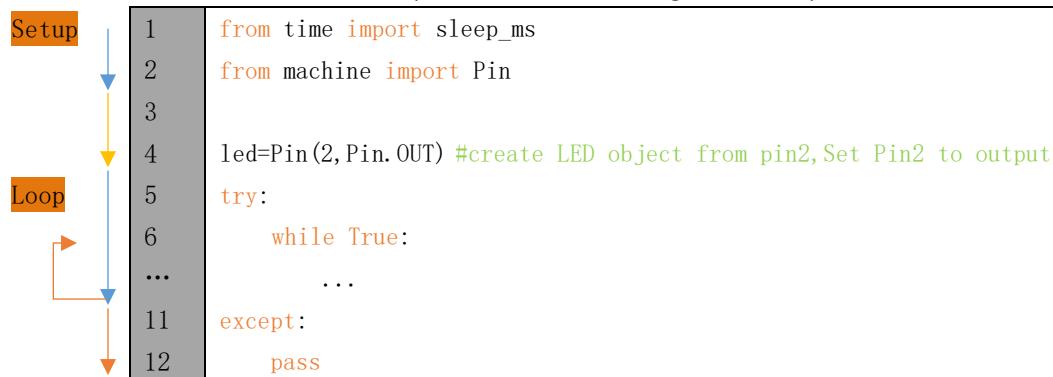
The following is the program code:

```

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.



`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  ...
   ...
```

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  ...  
12  except:  
    pass
```

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

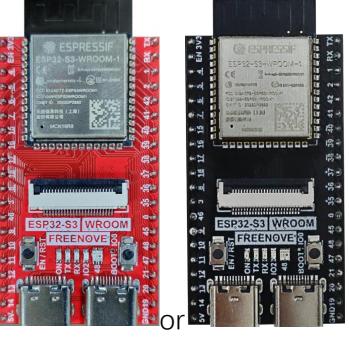
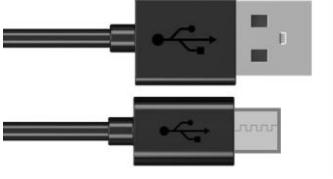
new_t: Ending time

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(N16R8)/ESP32-S3(N8R8)	USB cable x1
 or 	

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

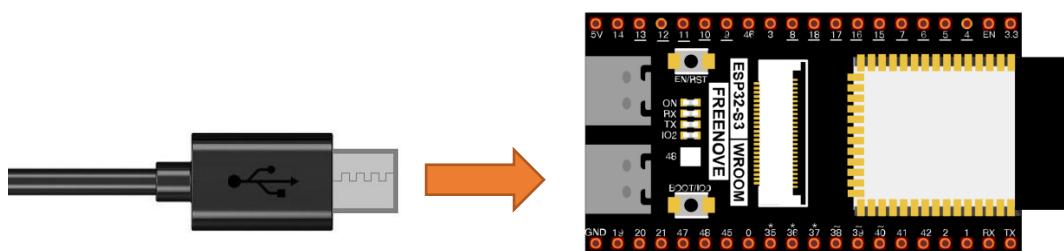
The Bluetooth module in slave mode can only accept connection request from a host computer, but cannot initiate a connection request. After connecting with a host device, it can send data to or receive from the host device.

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

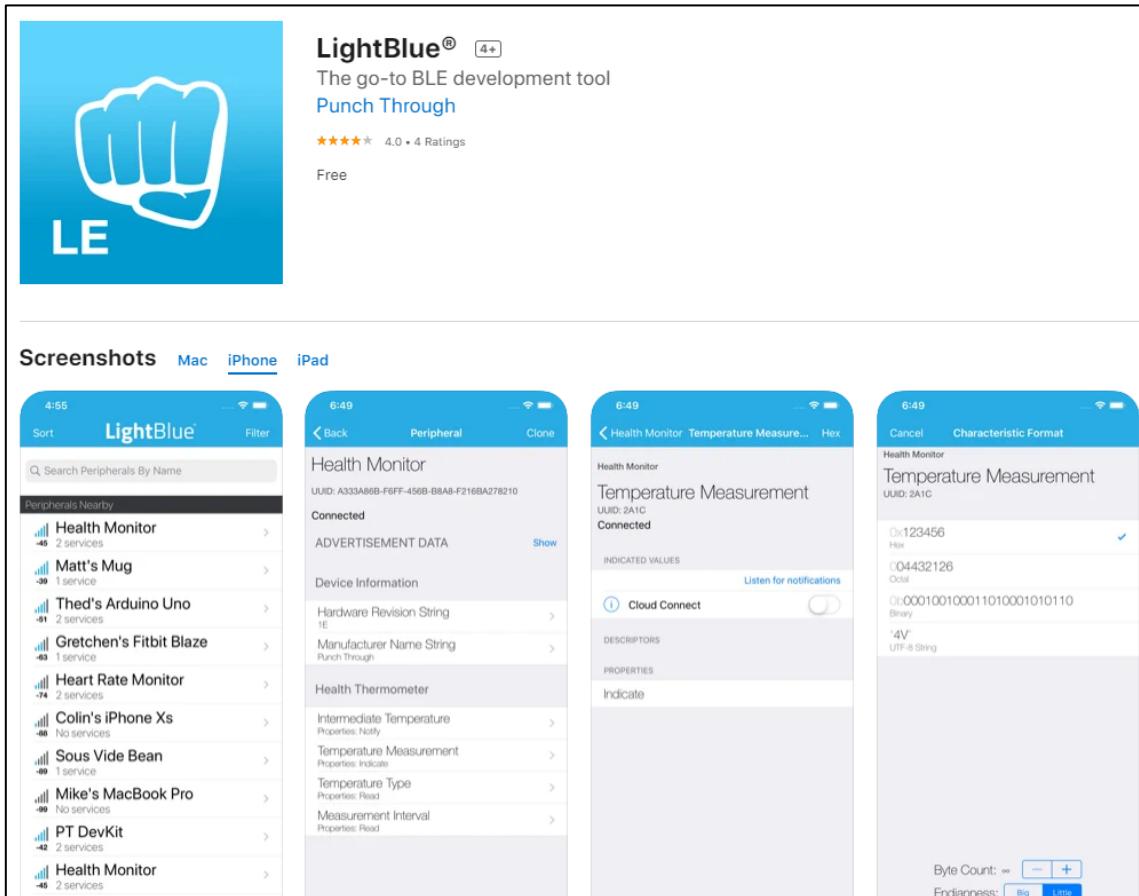
Connect Freenove ESP32-S3 to the computer using the USB cable.



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.>

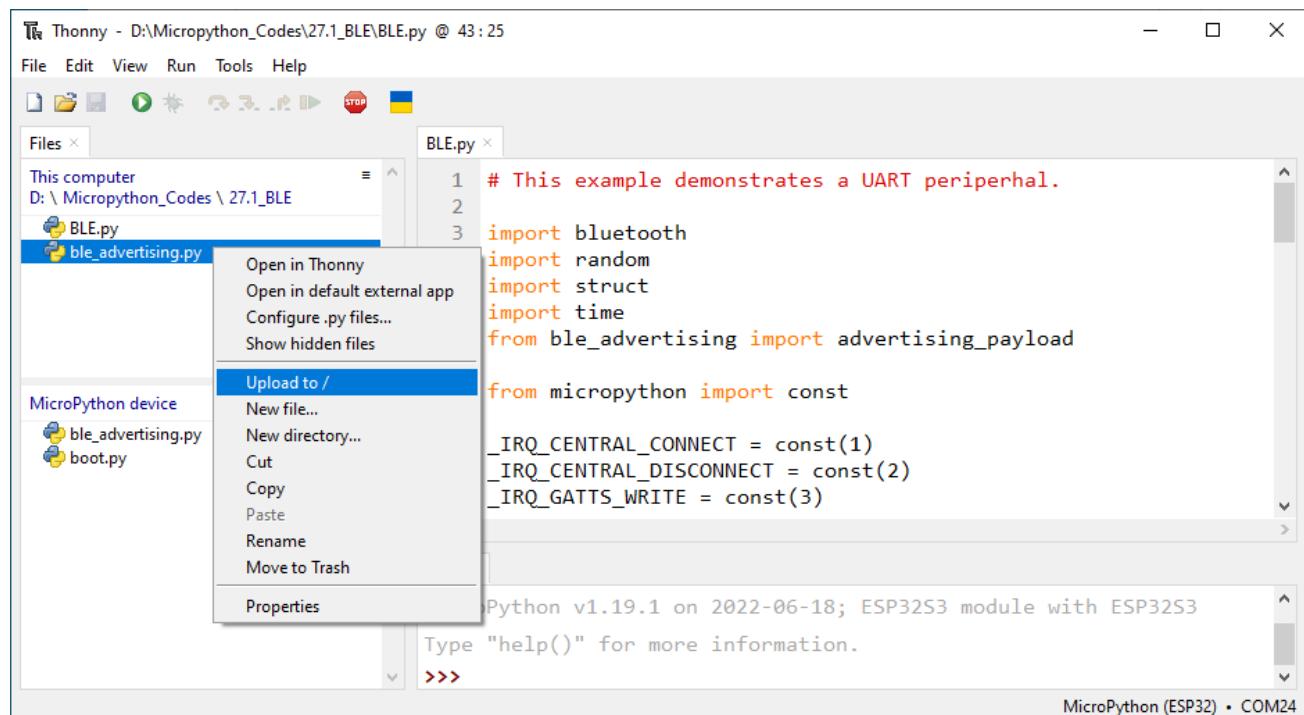


Code

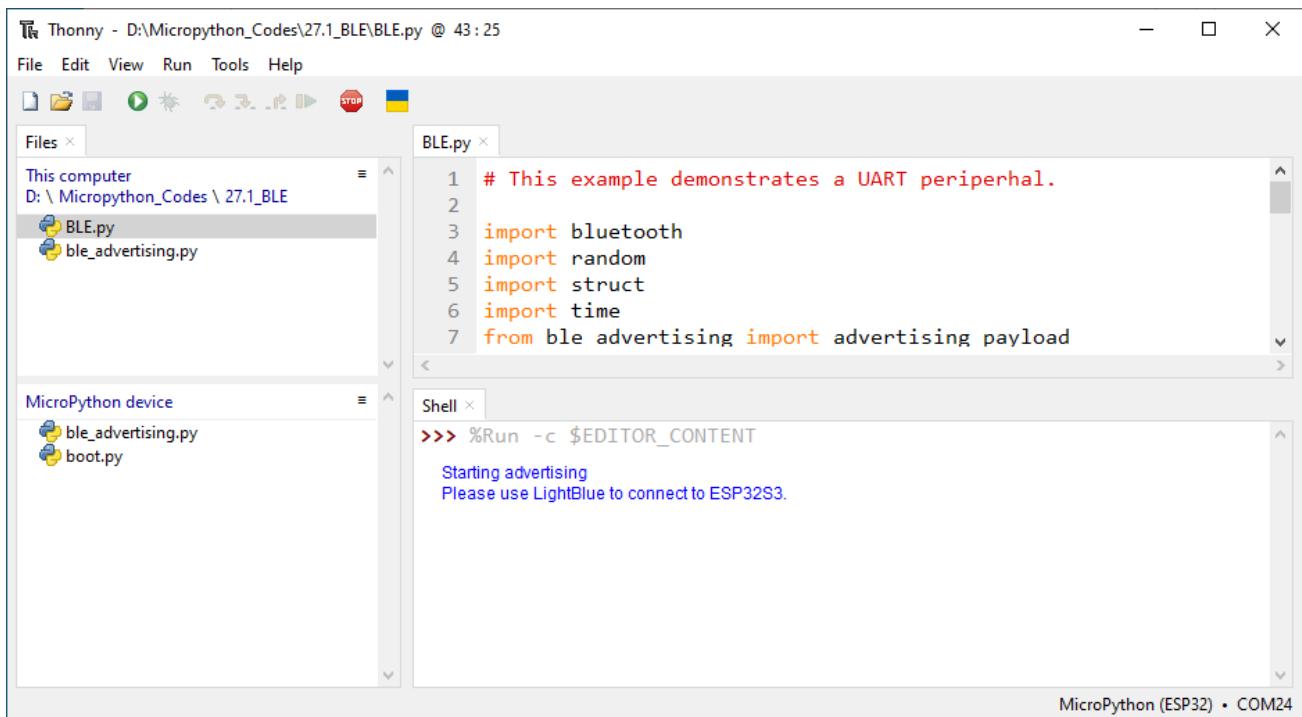
Move the program folder “**Freenove_ESP32_S3_WROVER_Board/Python/Python_Codes**” to disk(D) in advance with the path of “**D:/Micropython_Codes**”.

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.



The screenshot shows the Thonny IDE interface. The title bar says "Thonny - D:\Micropython_Codes\27.1_BLE\BLE.py @ 43:25". The menu bar includes File, Edit, View, Run, Tools, Help. The toolbar has icons for file operations and a stop button. The left sidebar shows "Files" with "This computer" containing "D:\Micropython_Codes\27.1_BLE\BLE.py" and "ble_advertising.py", and "MicroPython device" containing "ble_advertising.py" and "boot.py". The main area shows the code for BLE.py:

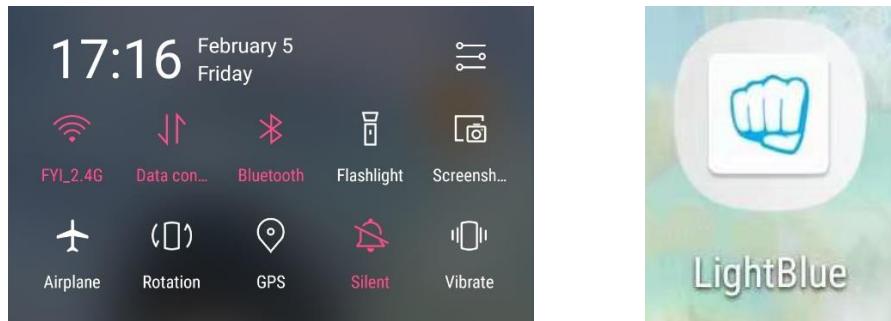
```

1 # This example demonstrates a UART peripheral.
2
3 import bluetooth
4 import random
5 import struct
6 import time
7 from ble_advertising import advertising_payload

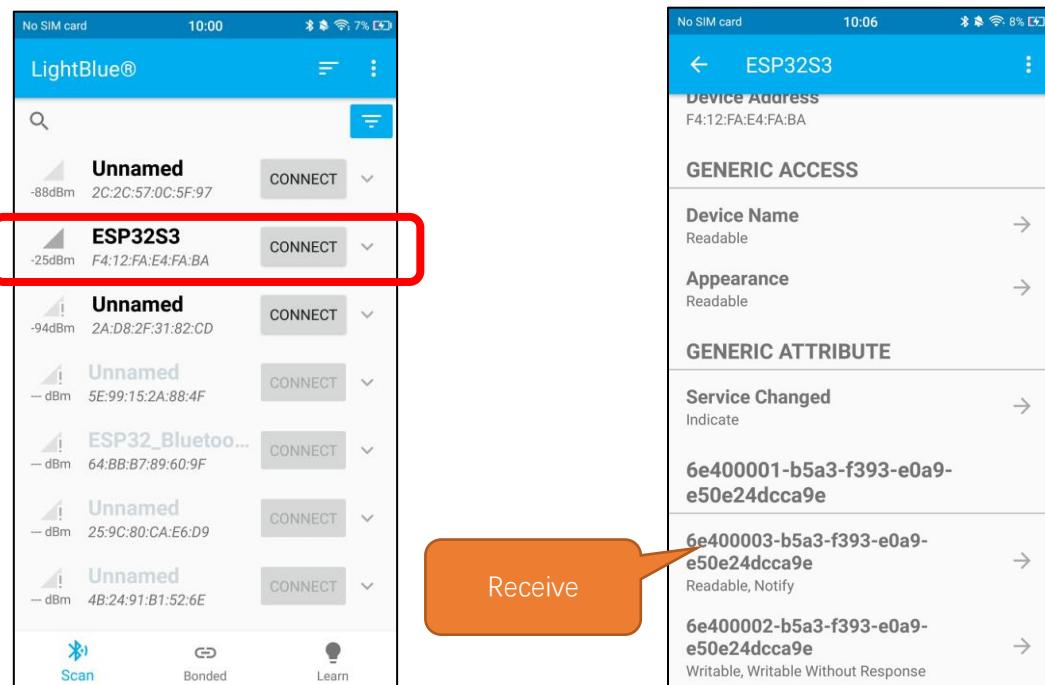
```

The shell window below shows the command ">>> %Run -c \$EDITOR_CONTENT" and the output "Starting advertising Please use LightBlue to connect to ESP32S3." The status bar at the bottom right says "MicroPython (ESP32) • COM24".

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.

```

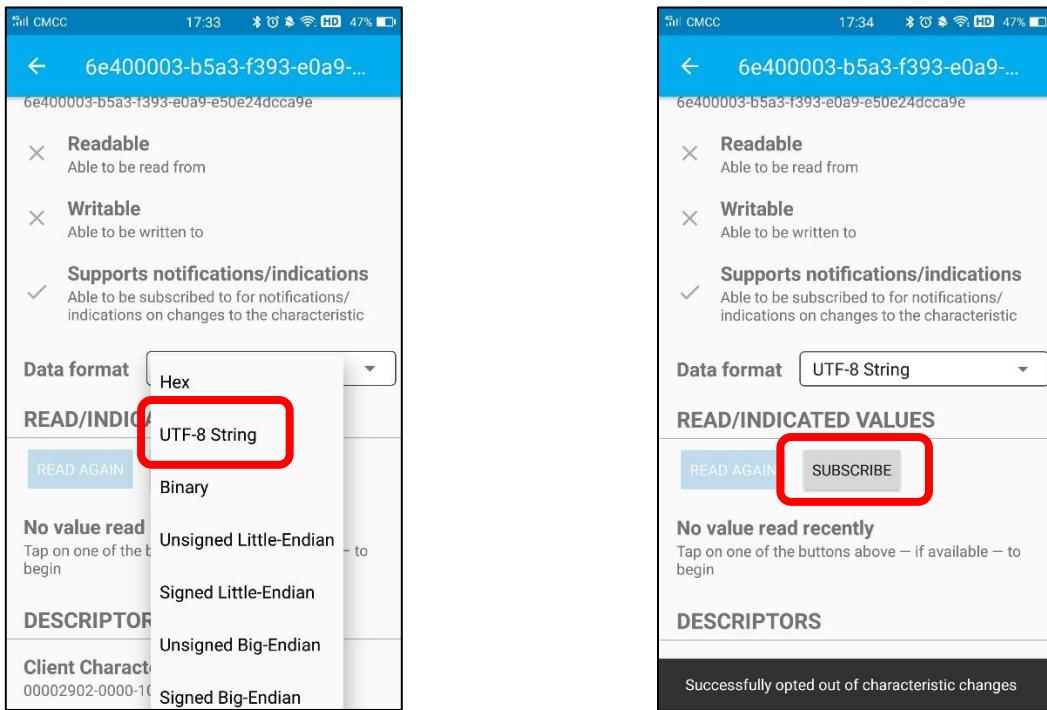
Shell >
>>> %Run -c $EDITOR_CONTENT
Starting advertising
Please use LightBlue to connect to ESP32S3.
New connection 1

The BLE connection is successful.
Enter anything:

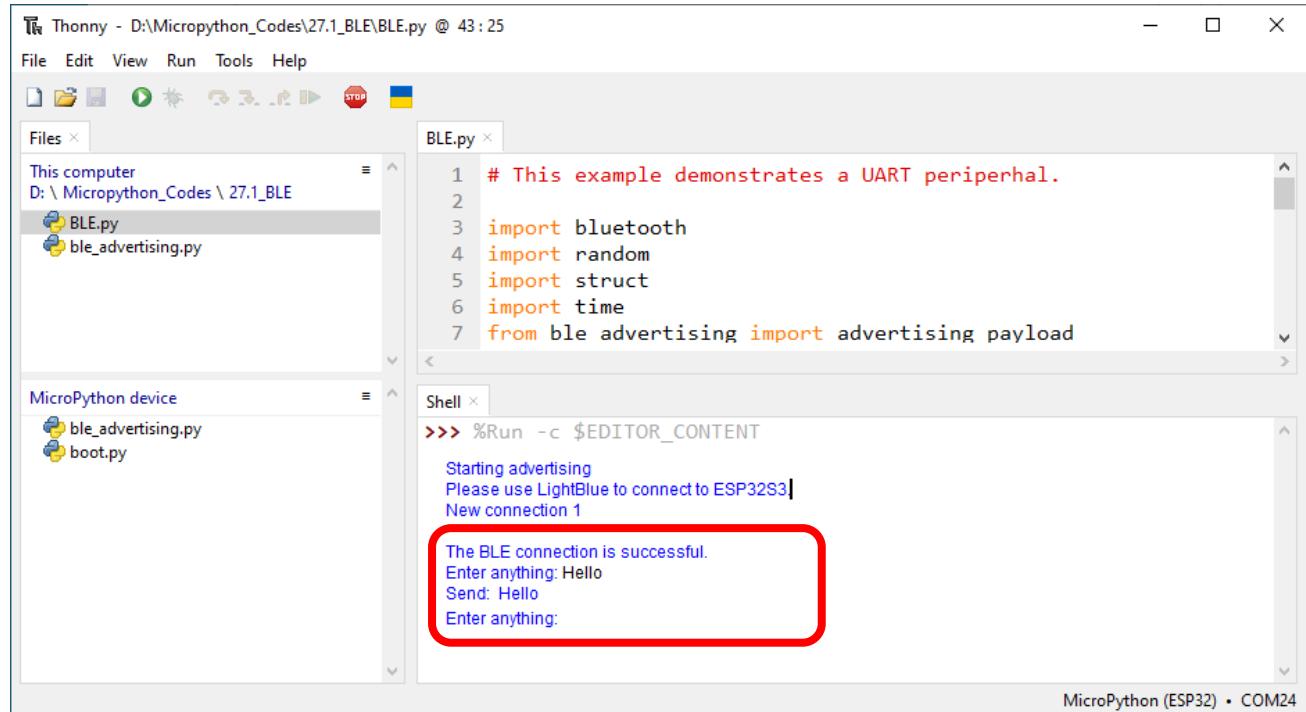
```

MicroPython (ESP32) • COM24

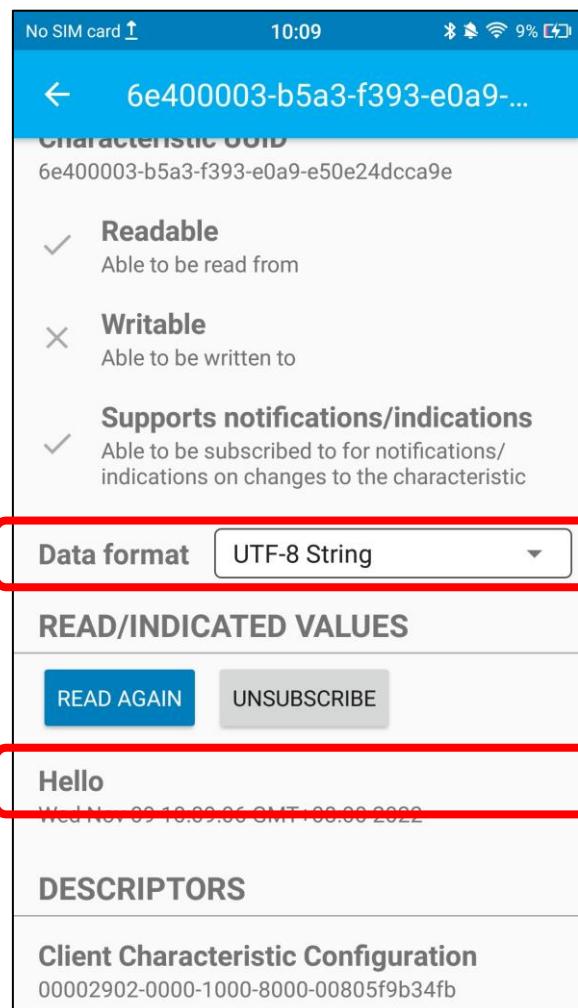
Click "Receive". Select the appropriate Data format in the box to the right of Data Format. For example, HEX for hexadecimal, utf-string for character, Binary for Binary, etc. Then click SUBSCRIBE.



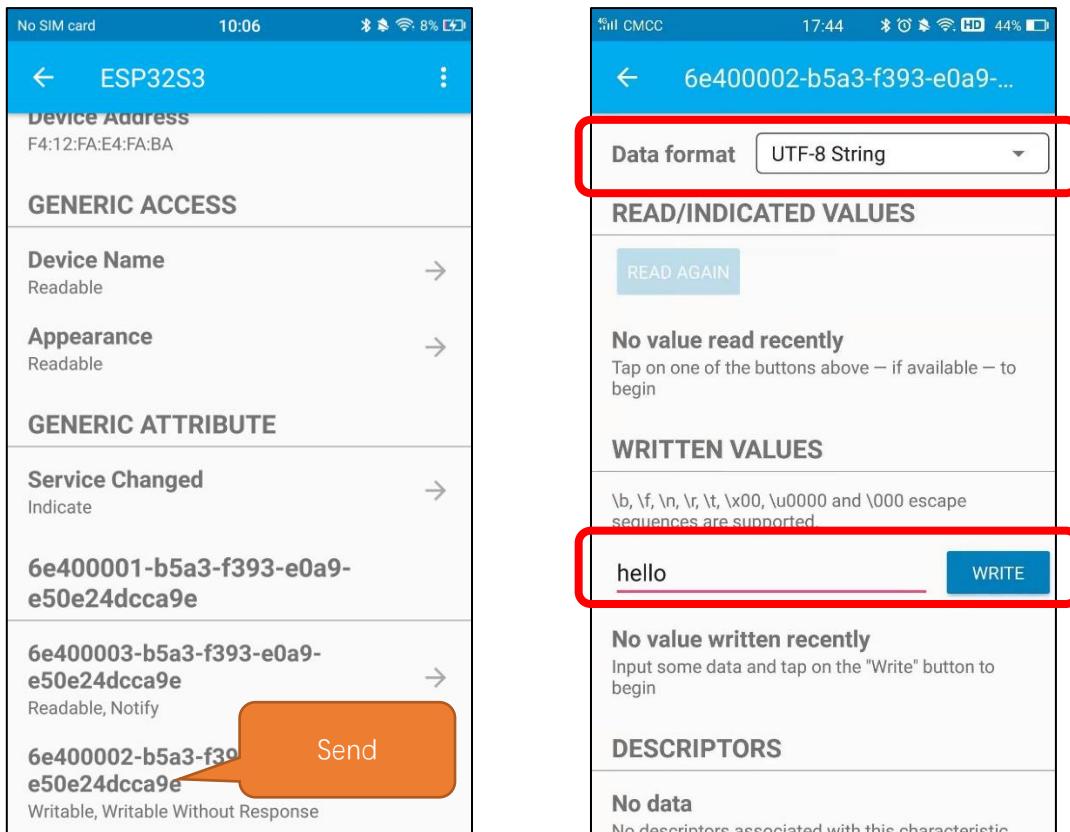
You can type “Hello” in Shell and press “Enter” to send.



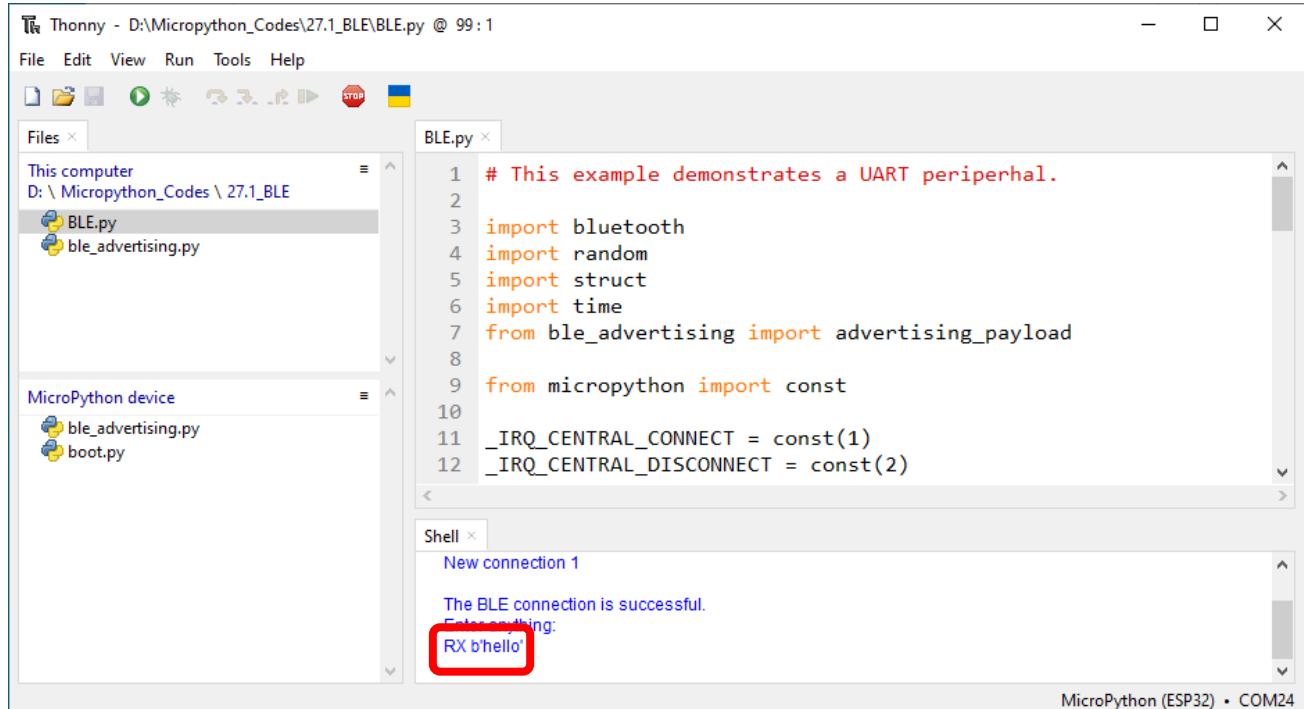
And then you can see the mobile Bluetooth has received the message.



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:

```
1 import bluetooth
2 import random
3 import struct
4 import time
5 from ble_advertising import advertising_payload
6 from micropython import const
7
8 _IRQ_CENTRAL_CONNECT = const(1)
9 _IRQ_CENTRAL_DISCONNECT = const(2)
10 _IRQ_GATTS_WRITE = const(3)
11 _FLAG_READ = const(0x0002)
12 _FLAG_WRITE_NO_RESPONSE = const(0x0004)
13 _FLAG_WRITE = const(0x0008)
14 _FLAG_NOTIFY = const(0x0010)
15
16 _UART_UUID = bluetooth.UUID("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")
17 _UART_TX = (
18     bluetooth.UUID("6E400003-B5A3-F393-E0A9-E50E24DCCA9E"),
19     _FLAG_READ | _FLAG_NOTIFY,
20 )
21 _UART_RX = (
22     bluetooth.UUID("6E400002-B5A3-F393-E0A9-E50E24DCCA9E"),
23     _FLAG_WRITE | _FLAG_WRITE_NO_RESPONSE,
24 )
25 _UART_SERVICE = (
26     _UART_UUID,
27     (_UART_TX, _UART_RX),
28 )
29 class BLESimplePeripheral:
30     def __init__(self, ble, name="ESP32S3"):
31         self._ble = ble
32         self._ble.active(True)
33         self._ble.irq(self._irq)
34         ((self._handle_tx, self._handle_rx),) =
35         self._ble.gatts_register_services((_UART_SERVICE,))
36         self._connections = set()
37         self._write_callback = None
38         self._payload = advertising_payload(name=name, services=[_UART_UUID])
39         self._advertise()
40     def _irq(self, event, data):
41         # Track connections so we can send notifications.
42         if event == _IRQ_CENTRAL_CONNECT:
```

Any concerns? ✉ support@freenove.com

```

43         conn_handle, _, _ = data
44         print("New connection", conn_handle)
45         print("\nThe BLE connection is successful.")
46         self._connections.add(conn_handle)
47     elif event == _IRQ_CENTRAL_DISCONNECT:
48         conn_handle, _, _ = data
49         print("Disconnected", conn_handle)
50         self._connections.remove(conn_handle)
51         # Start advertising again to allow a new connection.
52         self._advertise()
53     elif event == _IRQ_GATTS_WRITE:
54         conn_handle, value_handle = data
55         value = self._ble.gatts_read(value_handle)
56         if value_handle == self._handle_rx and self._write_callback:
57             self._write_callback(value)
58     def send(self, data):
59         for conn_handle in self._connections:
60             self._ble.gatts_notify(conn_handle, self._handle_tx, data)
61     def is_connected(self):
62         return len(self._connections) > 0
63     def _advertise(self, interval_us=500000):
64         print("Starting advertising")
65         self._ble.gap_advertise(interval_us, adv_data=self._payload)
66     def on_write(self, callback):
67         self._write_callback = callback
68     def demo():
69         ble = bluetooth.BLE()
70         p = BLESimplePeripheral(ble)
71         def on_rx(rx_data):
72             print("RX", rx_data)
73         p.on_write(on_rx)
74         print("Please use LightBlue to connect to ESP32S3.")
75         while True:
76             if p.is_connected():
77                 # Short burst of queued notifications.
78                 tx_data = input("Enter anything: ")
79                 print("Send: ", tx_data)
80                 p.send(tx_data)
81             if __name__ == "__main__":
82                 demo()

```

Define the specified UUID number for BLE vendor.

```

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

```

```

21     _FLAG_READ | _FLAG_NOTIFY,
22 )
23 _UART_RX = (
24     bluetooth.UUID("6E400002-B5A3-F393-E0A9-E50E24DCCA9E"),
25     _FLAG_WRITE | _FLAG_WRITE_NO_RESPONSE,
26 )

```

Write an _irq function to manage BLE interrupt events.

```

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

```

Initialize the BLE function and name it.

```

33     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.

```

70     def demo():
71         ble = bluetooth.BLE()
72         p = BLESimplePeripheral(ble)
73         def on_rx(rx_data):
74             print("RX", rx_data)
75             p.on_write(on_rx)
76             print("Please use LightBlue to connect to ESP32S3.")
77         while True:
78             if p.is_connected():
79                 # Short burst of queued notifications.
80                 tx_data = input("Enter anything: ")
81                 print("Send: ", tx_data)
82                 p.send(tx_data)
83             lastMsg = now;
84     }

```



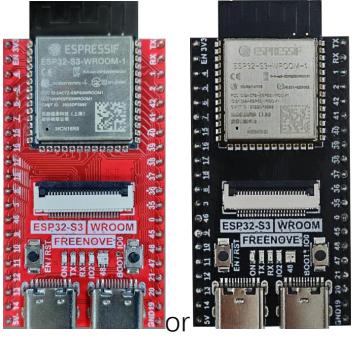
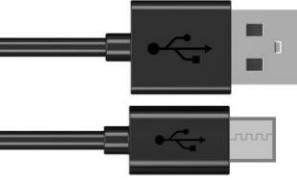
Chapter 3 WiFi Working Modes

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(N16R8)/ESP32-S3(N8R8)	USB cable x1
	

Component knowledge

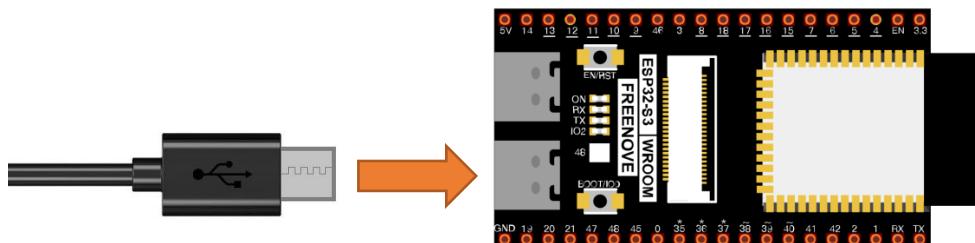
Station mode

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

```

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()

```

The screenshot shows the Thonny IDE interface with the file `Station_mode.py` open. The code is written in Python for MicroPython. It defines a function `STA_Setup` that attempts to connect to a Wi-Fi router using the provided SSID and password. The `ssidRouter` and `passwordRouter` variables are currently set to five asterisks each, indicated by a red box. An orange callout bubble with the text "Enter the correct Router name and password." points to these variables. The Thonny shell at the bottom shows the output of running the code, which includes the setup process and a successful connection with IP details.

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".

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

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)
```

Any concerns? ✉ support@freenove.com



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: WiFi name

password: WiFi password

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

Project 3.2 AP mode

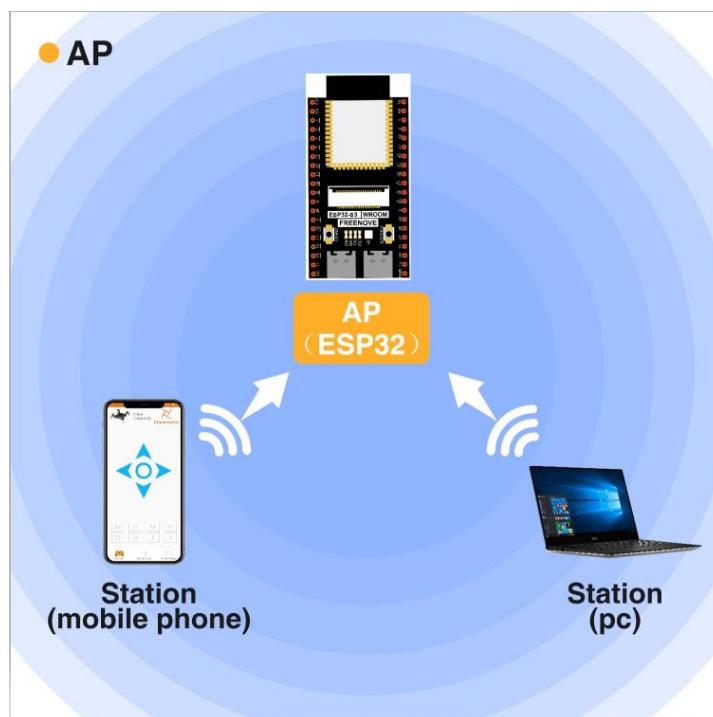
Component List & Circuit

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

Component knowledge

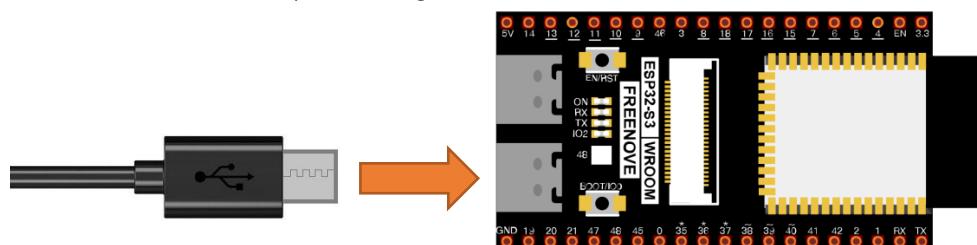
AP mode

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.



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

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

```

Thonny - D:\Micropython_Codes\28.2_AP_mode\AP_mode.py @ 35:5
File Edit View Run Tools Help
AP_mode.py x
1 import network
2
3 ssidAP      = 'WiFi_Name' #Enter the router name
4 passwordAP   = '12345678' #Enter the router password
5
6 local_IP     = '192.168.1.10'
7 gateway      = '192.168.1.1'
8 subnet       = '255.255.255.0'
9 dns          = '8.8.8.8'
10
11 ap_if = network.WLAN(network.AP_IF)
12
13 def AP_Setup(ssidAP,passwordAP):
14     ap_if.ifconfig([local_IP,gateway,subnet,dns])
15     print("Setting soft-AP ... ")
16     ap_if.config(essid=ssidAP,authmode=network.AUTH_WPA_WPA2_PSK)
17     ap_if.active(True)
18     print('Success, IP address:', ap_if.ifconfig())
19     print("Setup End\n")
20
21 try:
22     AP_Setup(ssidAP,passwordAP)
    
```

Set a name and a password for ESP32S3 AP.

Shell x

```

>>> %Run -c $EDITOR_CONTENT
Setting soft-AP ...
Success, IP address: ('192.168.1.10', '192.168.1.1', '255.255.255.0', '8.8.8.8')
Setup End
    
```

MicroPython (ESP32) • COM24

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.

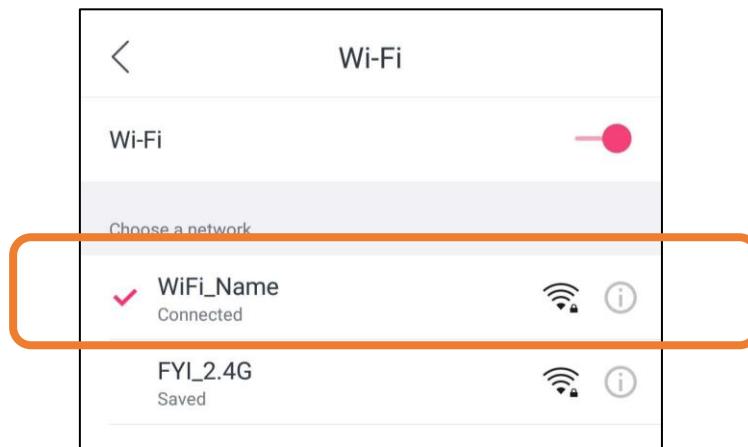
Click “Run current script”, open the AP function of ESP32-S3 and print the access point information.

```

Shell x
>>> %Run -c $EDITOR_CONTENT
Setting soft-AP ...
Success, IP address: ('192.168.1.10', '192.168.1.1', '255.255.255.0', '8.8.8.8')
Setup End
    
```

MicroPython (ESP32) • COM24

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:

```

1 import network
2
3 ssidAP      = 'WiFi_Name' #Enter the router name
4 passwordAP  = '12345678' #Enter the router password
5
6 local_IP    = '192.168.1.10'
7 gateway     = '192.168.1.1'
8 subnet      = '255.255.255.0'
9 dns         = '8.8.8.8'
10
11 ap_if = network.WLAN(network.AP_IF)
12
13 def AP_Setup(ssidAP, passwordAP):
14     ap_if.ifconfig([local_IP, gateway, subnet, dns])
15     print("Setting soft-AP ... ")
16     ap_if.config(essid=ssidAP, authmode=network.AUTH_WPA_WPA2_PSK, password=passwordAP)
17     ap_if.active(True)
18     print(' Success, IP address:', ap_if.ifconfig())
19     print("Setup End\n")
20
21 try:
22     AP_Setup(ssidAP, passwordAP)
23 except:
24     ap_if.disconnect()

```

Import network module.

```
1 import network
```

Enter correct AP name and password.

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

Set 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: WiFi name

password: WiFi password

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

ssid: WiFi account name

channel: WiFi channel

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

ip: IP address

subnet_mask: subnet mask

gateway: gateway

DNS_server: DNS server

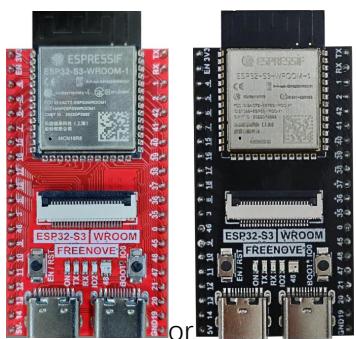
disconnect(): Disconnect from the currently connected wireless network

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

Project 3.3 AP+Station mode

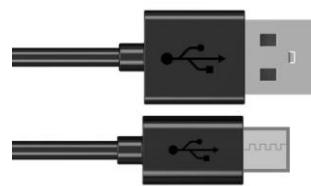
Component List

ESP32-S3(N16R8)/ESP32-S3(N8R8)



or

USB cable x1



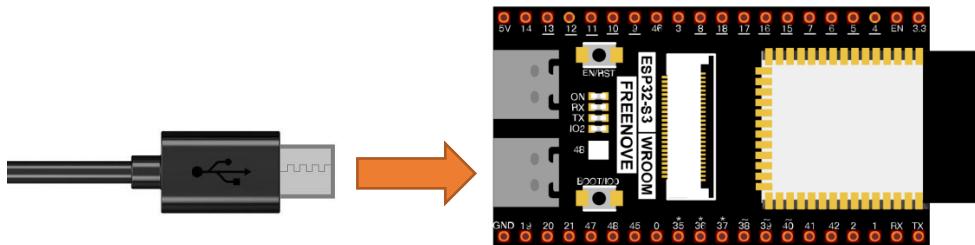
Component knowledge

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

```

1 import network
2
3 ssidRouter      = '*****' #Enter the router name
4 passwordRouter = '*****' #Enter the router password
5
6 ssidAP          = 'WiFi_Name'#Enter the AP name
7 passwordAP      = '12345678' #Enter the AP password
8
9 local_IP        = '192.168.4.150'
10 gateway         = '192.168.4.1'
11 subnet          = '255.255.255.0'
12 dns             = '8.8.8.8'
13
14 sta_if = network.WLAN(network.STA_IF)
15 ap_if = network.WLAN(network.AP_IF)
16
17 def STA_Setup(ssidRouter,passwordRouter):
18     print("Setting soft-STA ... ")
19     if not sta_if.isconnected():
20         print('connecting to',ssidRouter)
21         sta_if.active(True)
22         sta_if.connect(ssidRouter,passwordRouter)

```

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 x
>>> %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

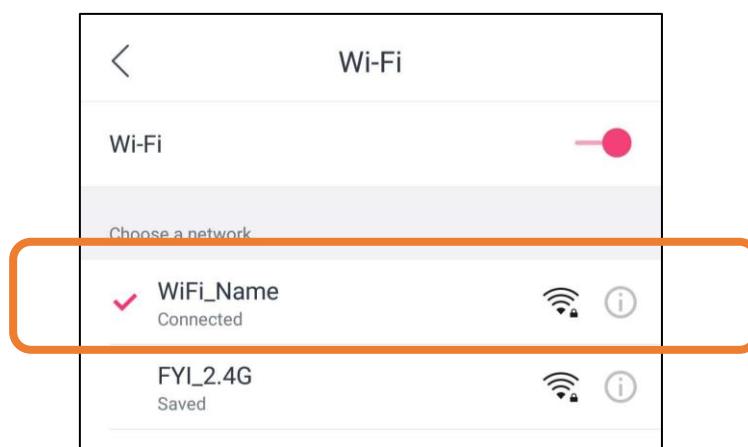
Setting soft-STA ...
connecting to FYI_2.4G
Connected, IP address: ('192.168.1.129', '255.255.255.0', '192.168.1.1', '192.168.1.1')
Setup End

>>>

```

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:

```

1 import network
2
3 ssidRouter      = '*****' #Enter the router name
4 passwordRouter = '*****' #Enter the router password
5
6 ssidAP          = 'WiFi_Name'#Enter the AP name
7 passwordAP      = '12345678' #Enter the AP password
8
9 local_IP        = '192.168.4.150'
10 gateway         = '192.168.4.1'
11 subnet          = '255.255.255.0'
12 dns             = '8.8.8.8'
13
14 sta_if = network.WLAN(network.STA_IF)
15 ap_if = network.WLAN(network.AP_IF)
16
17 def STA_Setup(ssidRouter, passwordRouter):
18     print("Setting soft-STA ... ")

```



```
19     if not sta_if.isconnected():
20         print('connecting to', ssidRouter)
21         sta_if.active(True)
22         sta_if.connect(ssidRouter, passwordRouter)
23         while not sta_if.isconnected():
24             pass
25         print('Connected, IP address:', sta_if.ifconfig())
26         print("Setup End")
27
28 def AP_Setup(ssidAP, passwordAP):
29     ap_if.ifconfig([local_IP, gateway, subnet, dns])
30     print("Setting soft-AP ... ")
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.disconnect()
```

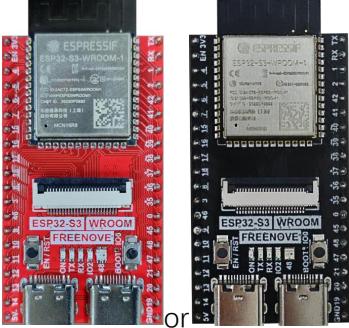
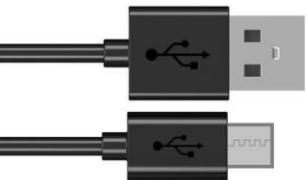
Chapter 4 TCP/IP

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(N16R8)/ESP32-S3(N8R8)	USB cable x1
 or	

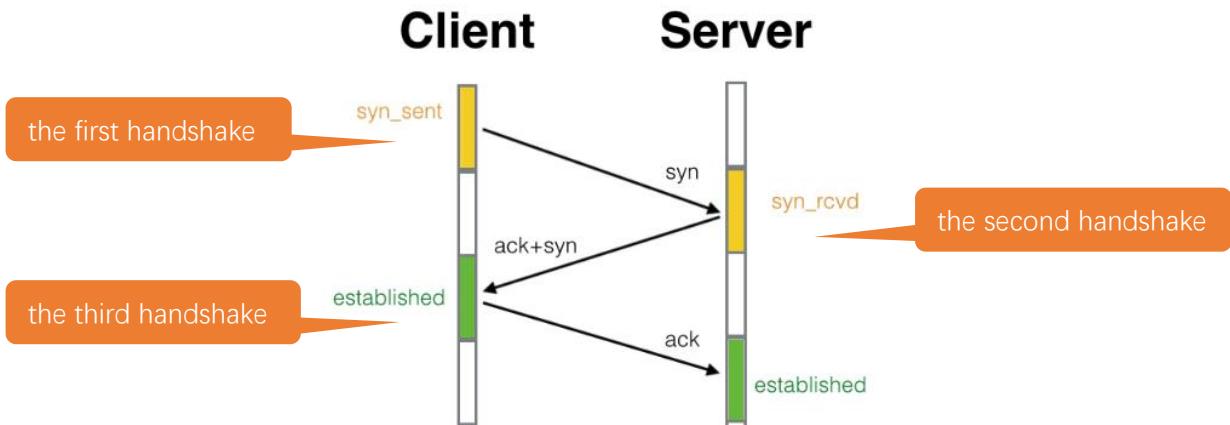
Component knowledge

TCP connection

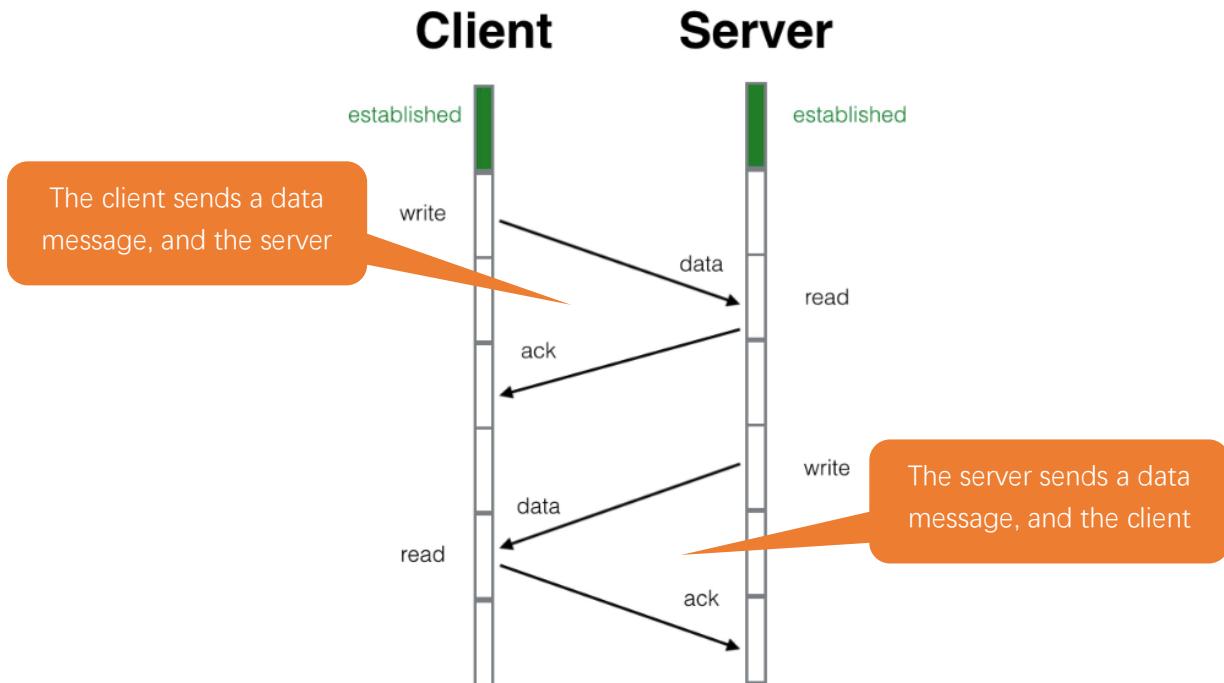
Before transmitting data, TCP needs to establish a logical connection between the sending end and the receiving end. It provides reliable and error-free data transmission between the two computers. In the TCP connection, the client and the server must be clarified. The client sends a connection request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

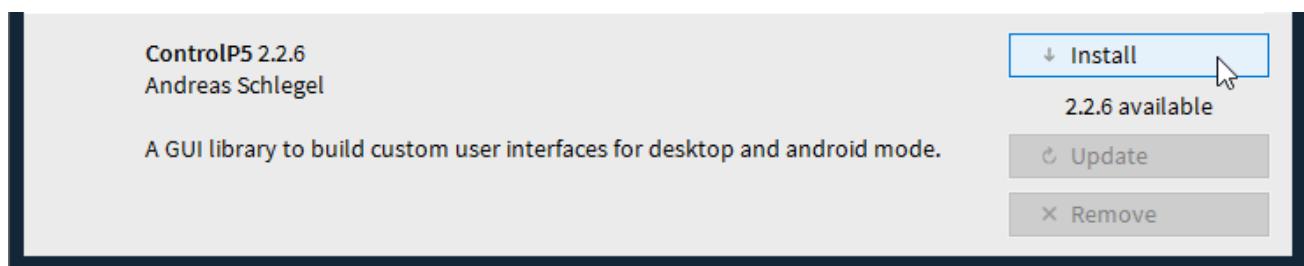
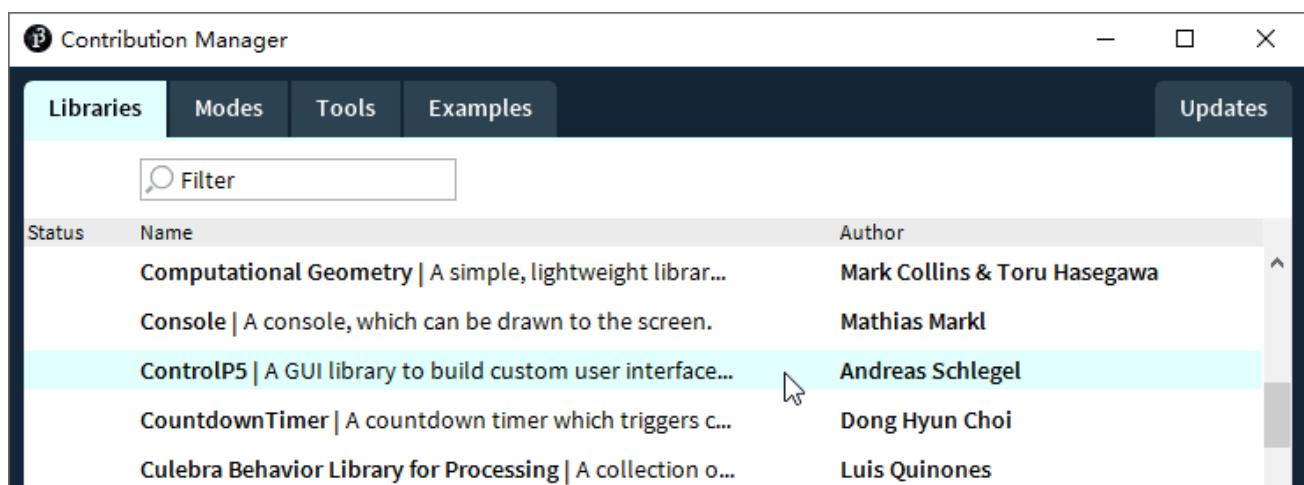
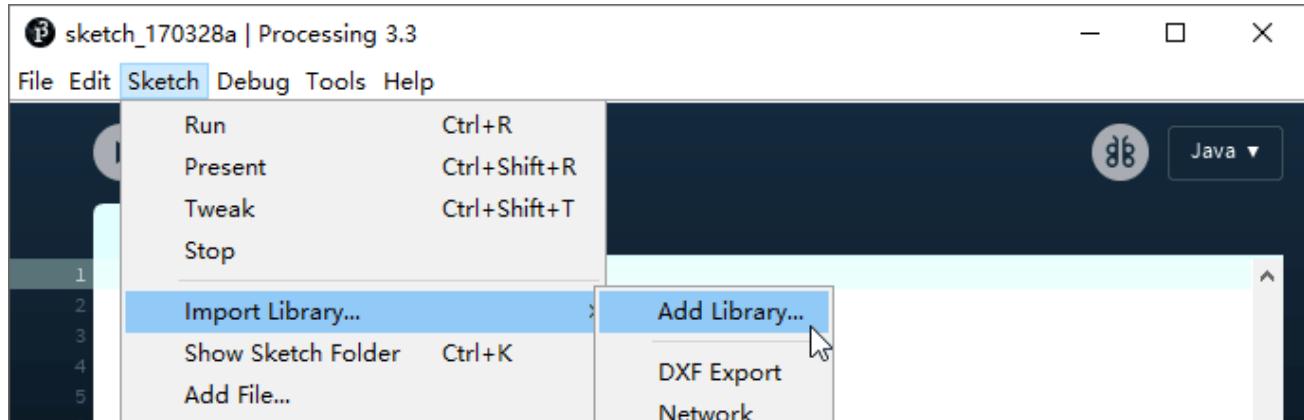
The screenshot shows the official Processing website. At the top, there's a navigation bar with links for "Processing", "p5.js", "Processing.py", "Processing for Android", "Processing for Pi", and "Processing Foundation". Below the navigation bar, the main header features the word "Processing" in large letters next to a stylized geometric logo. To the right is a search bar with a magnifying glass icon. On the left side of the page, there's a sidebar with links for "Cover", "Download", "Donate", "Exhibition", "Reference", "Libraries", "Tools", "Environment", "Tutorials", "Examples", "Books", "Overview", and "People". The main content area is titled "Download Processing". It states that Processing is available for Linux, Mac OS X, and Windows. A large "3.5.4 (17 January 2020)" version number is displayed. Below this, there are download links for "Windows 64-bit", "Windows 32-bit", "Linux 64-bit", and "Mac OS X". Further down, there are links for "» Github", "» Report Bugs", "» Wiki", "» Supported Platforms", and a link to "Read about the changes in 3.0. The list of revisions covers the differences between releases in detail."

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

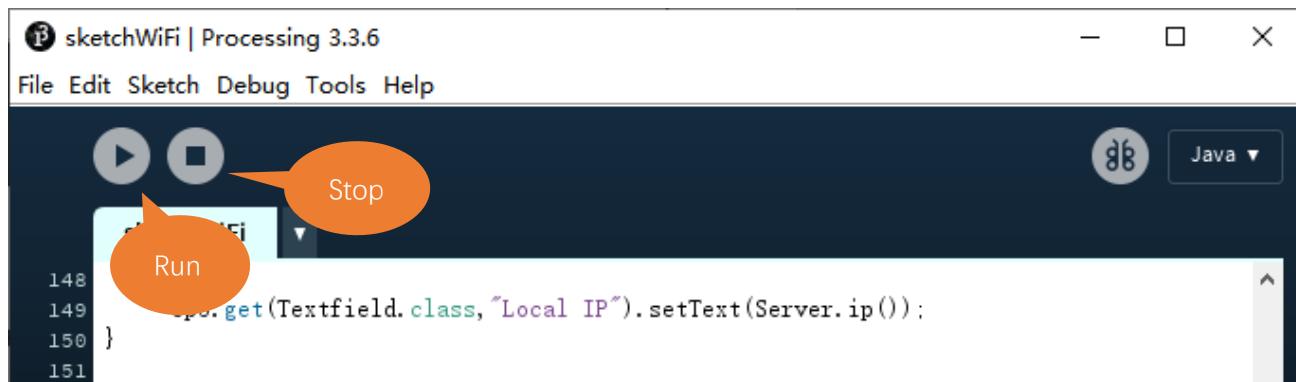
	core	2020/1/17 12:16
	java	2020/1/17 12:17
	lib	2020/1/17 12:16
	modes	2020/1/17 12:16
	tools	2020/1/17 12:16
	processing.exe	2020/1/17 12:16
	processing-java.exe	2020/1/17 12:16
	revisions.txt	2020/1/17 12:16

Use Server mode for communication

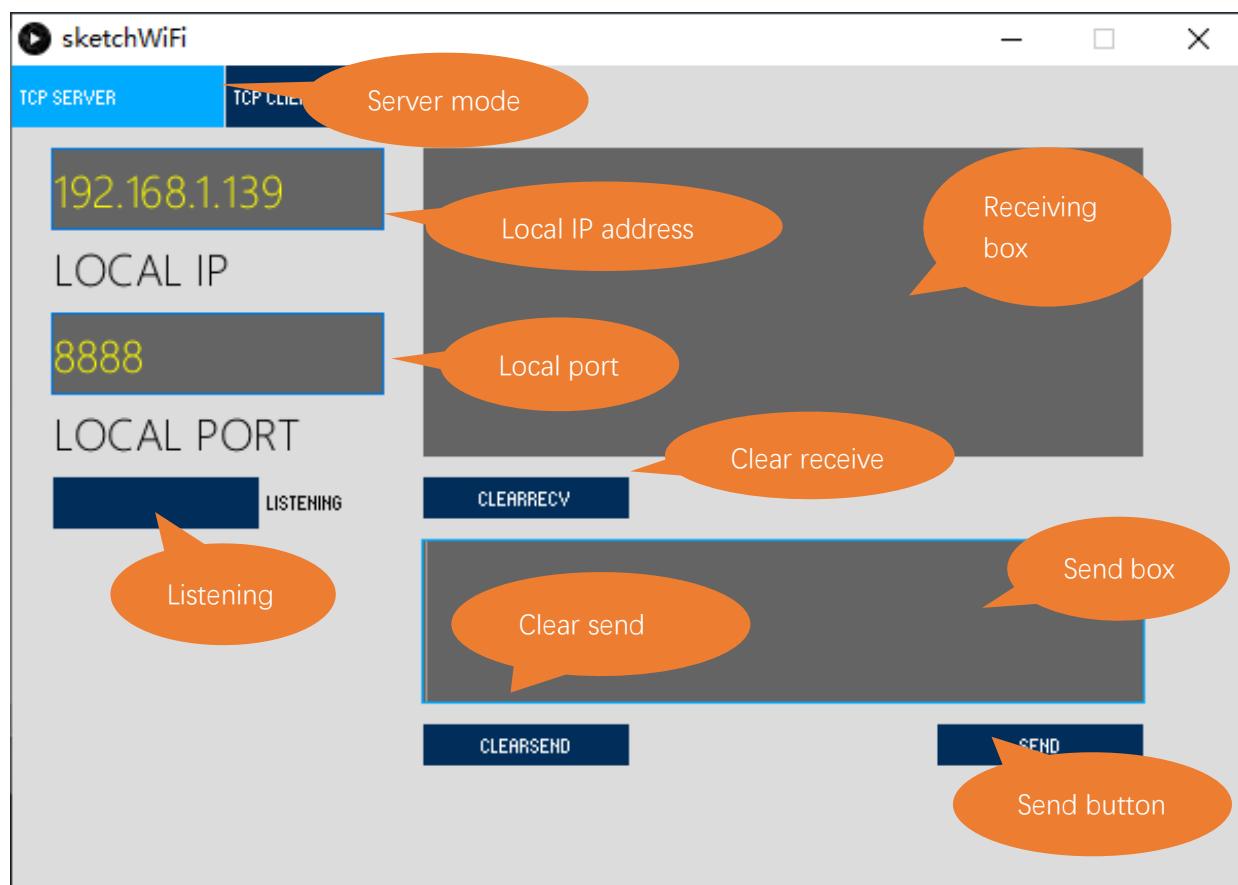
Install ControlP5.



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

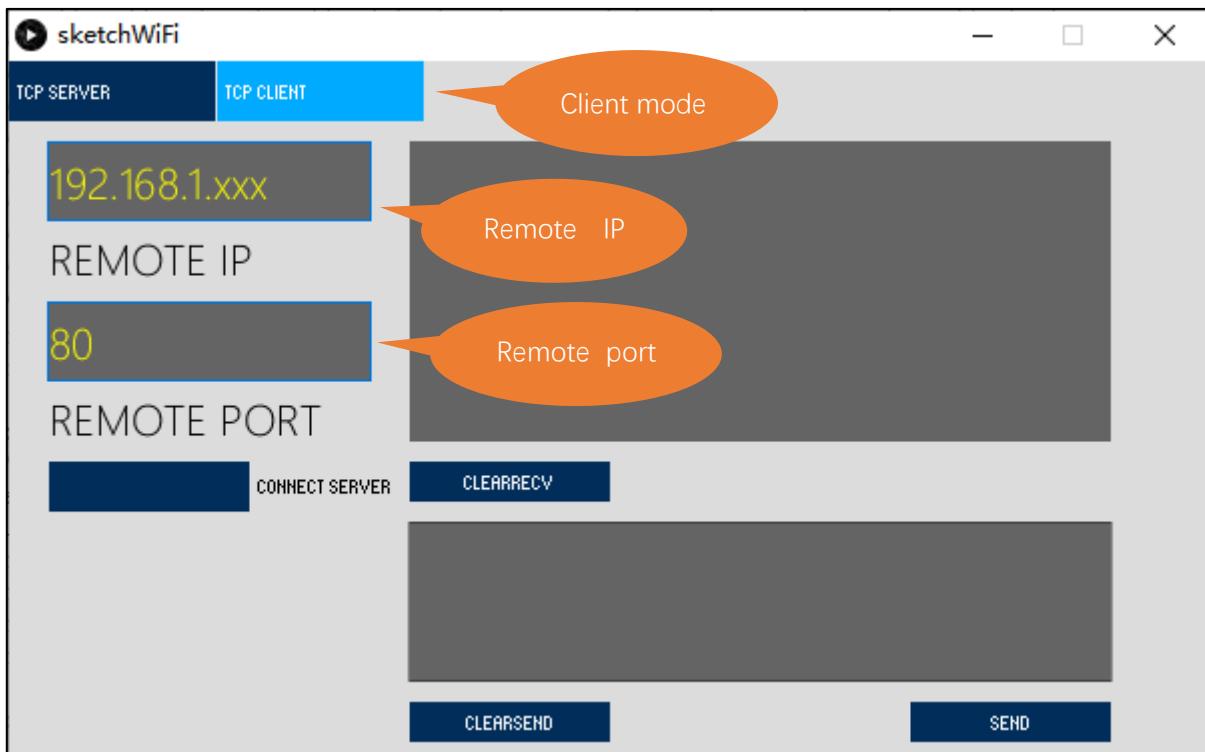


The new pop-up interface is as follows. If ESP32-S3 is used as client, select TCP SERVER mode for sketchWiFi.



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.

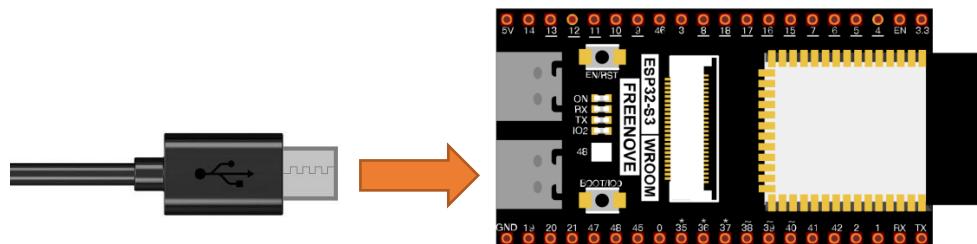
clear receive: clear out the content in the receiving text box

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.

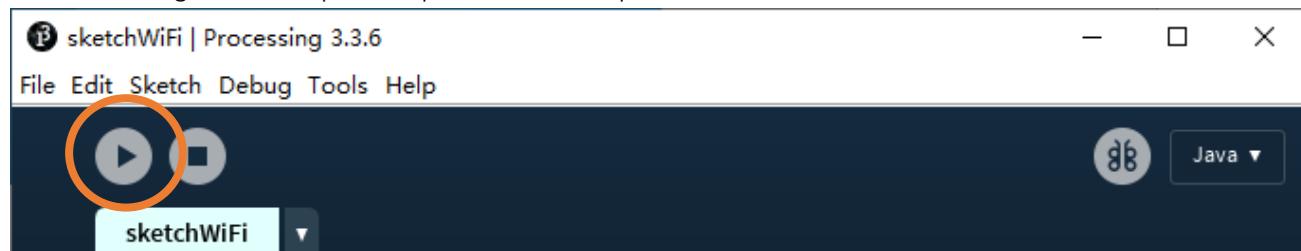
Circuit

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

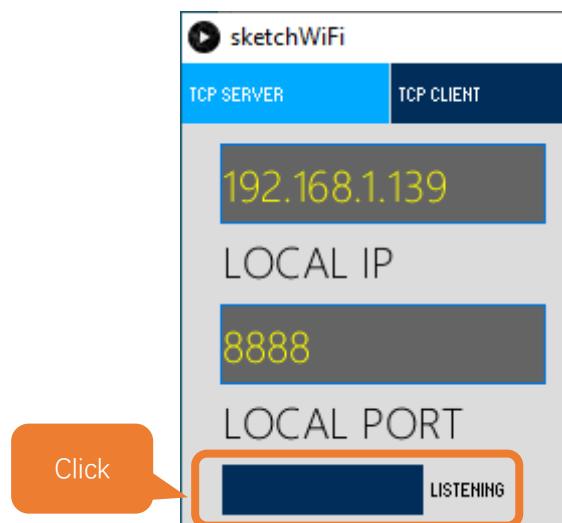


Code

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”.

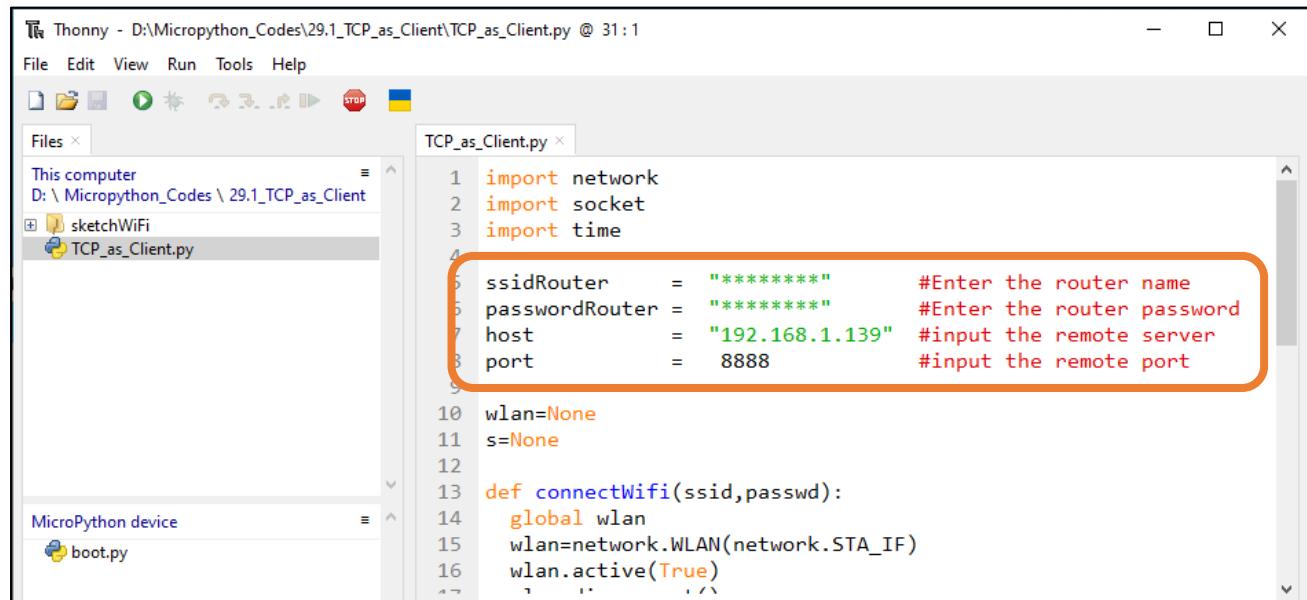


Move the program folder “**Freenove_ESP32_S3_WROVER_Board/Python/Python_Codes**” to disk(D) in advance with the path of “**D:/Micropython_Codes**”.

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



```

import network
import socket
import time

ssidRouter      = "*****"          #Enter the router name
passwordRouter = "*****"          #Enter the router password
host            = "192.168.1.139"    #input the remote server
port            = 8888              #input the remote port

wlan=None
s=None

def connectWifi(ssid,passwd):
    global wlan
    wlan=network.WLAN(network.STA_IF)
    wlan.active(True)

```

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



```

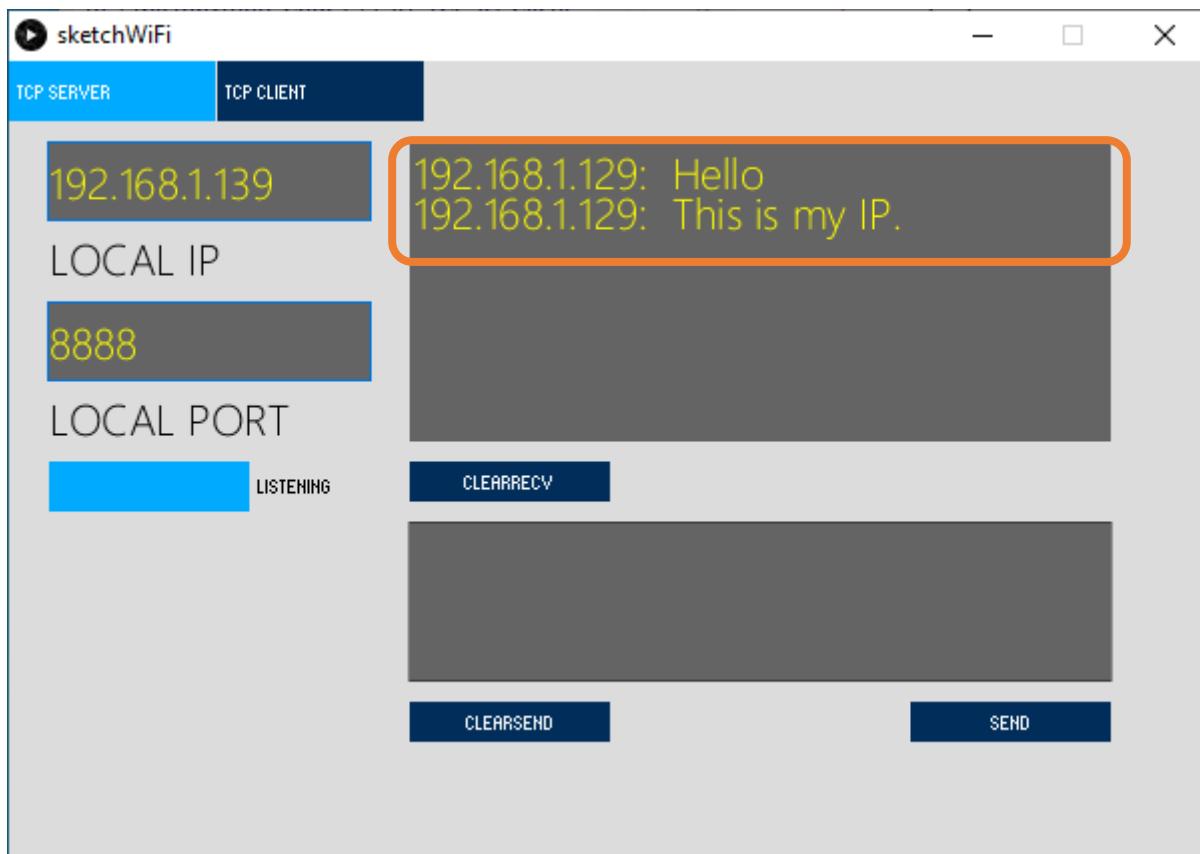
>>> %Run -c $EDITOR_CONTENT
TCP Connected to: 192.168.1.139 : 8888

```

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

```
Shell x
    for connected to: 192.168.1.142 : 8888
    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
```



```

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.

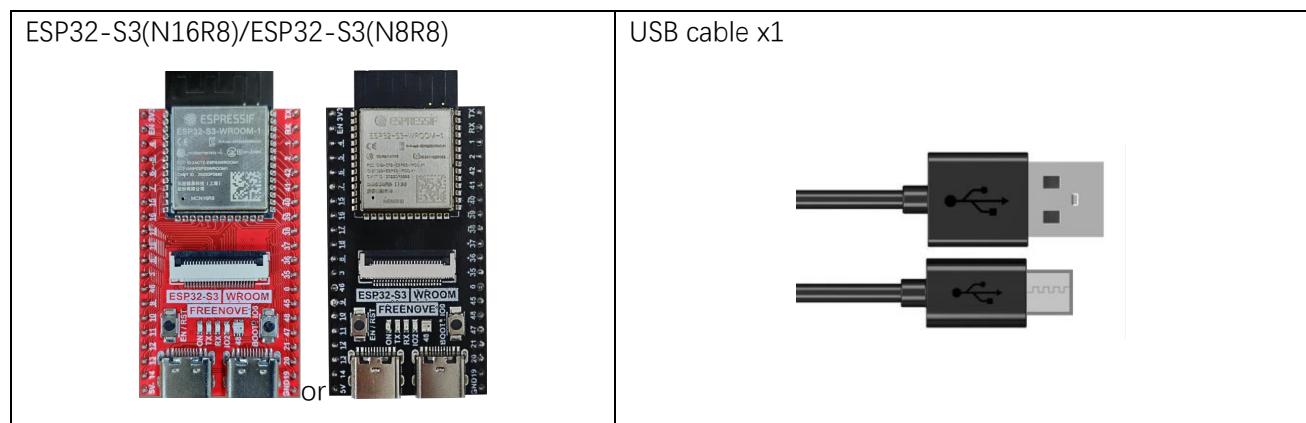
close(): Close socket.

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

Project 4.2 As Server

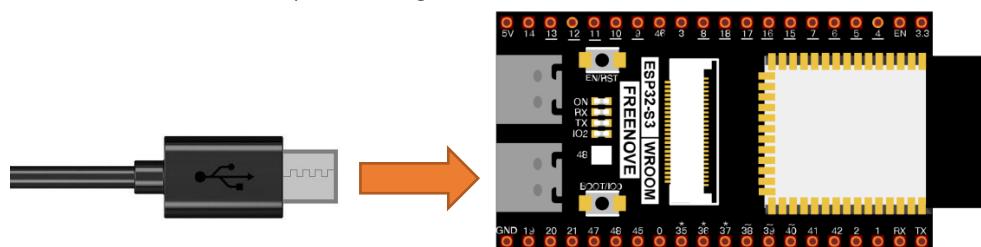
In this section, ESP32-S3 is used as a server to wait for the connection and communication of client on the same LAN.

Component List



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

```

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

```

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.

```

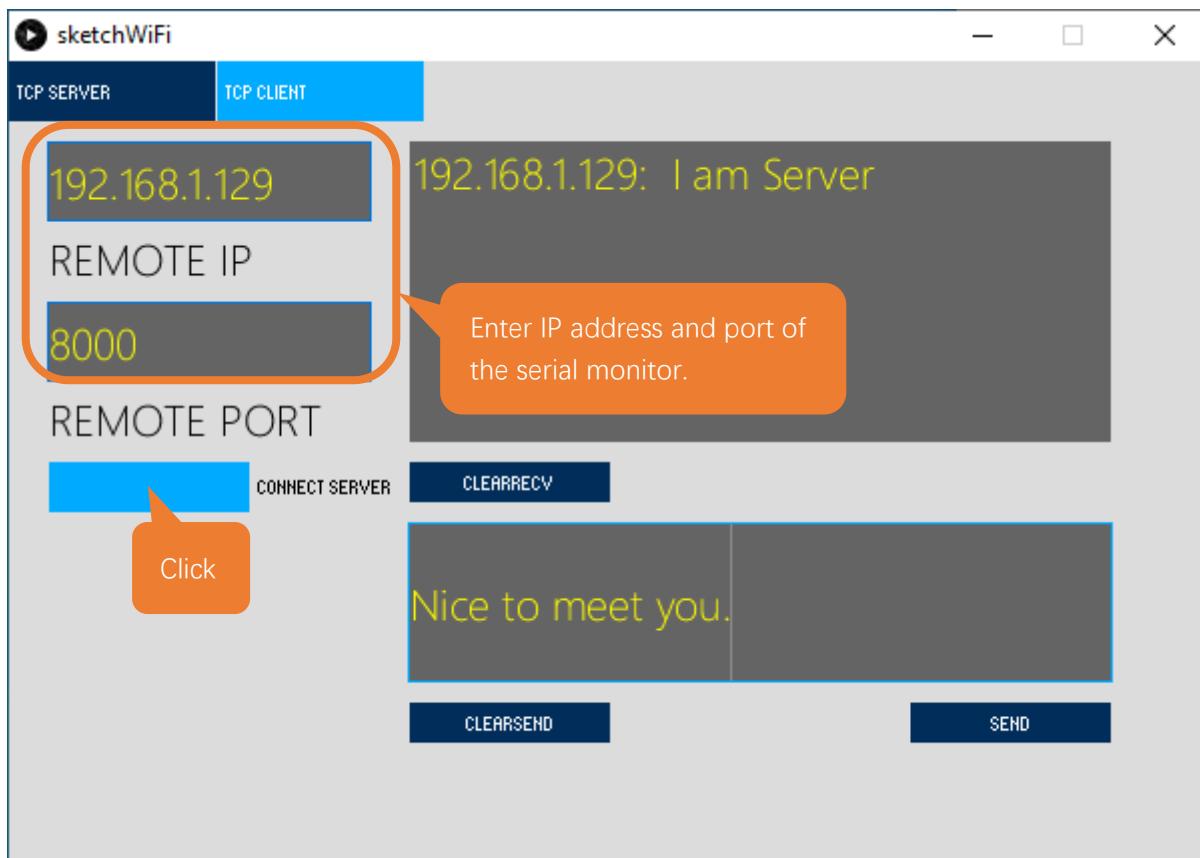
>>> %Run -c $EDITOR_CONTENT
tcp waiting...
Server IP: 192.168.1.129      Port: 8000
accepting....
(192.168.1.139, 49416) connected

```

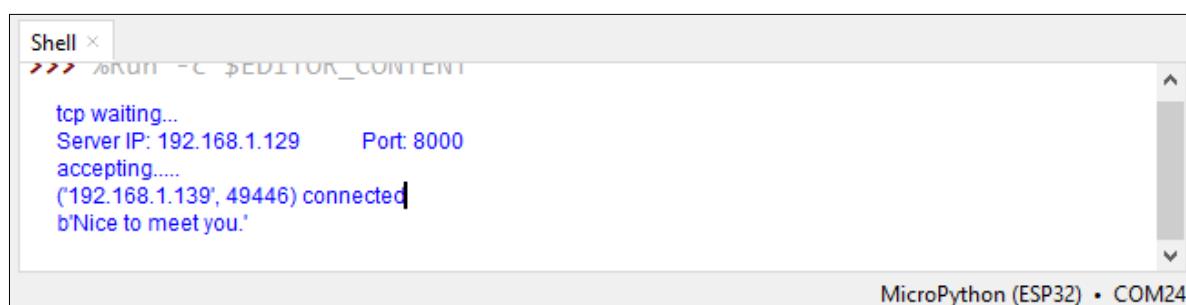
Processing:

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:

```
1 import network
2 import socket
3 import time
4
5 ssidRouter      = "*****"          #Enter the router name
6 passwordRouter = "*****"          #Enter the router password
7 port           = 8000             #input the remote port
8 wlan            = None
9 listenSocket    = None
10
11 def connectWifi(ssid,passwd):
12     global wlan
13     wlan=network.WLAN(network.STA_IF)
14     wlan.active(True)
15     wlan.disconnect()
16     wlan.connect(ssid,passwd)
17     while(wlan.ifconfig()[0]=='0.0.0.0'):
18         time.sleep(1)
19     return True
20
21 try:
22     connectWifi(ssidRouter,passwordRouter)
23     ip=wlan.ifconfig()[0]
24     listenSocket = socket.socket()
25     listenSocket.bind((ip, port))
26     listenSocket.listen(1)
27     listenSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
28     print('tcp waiting... ')
29     while True:
30         print("Server IP:", ip, "\tPort:", port)
31         print("accepting.... ")
32         conn,addr = listenSocket.accept()
33         print(addr, "connected")
34         break
35     conn.send('I am Server')
36     while True:
37         data = conn.recv(1024)
38         if(len(data) == 0):
39             print("close socket")
40             listenSocket.close()
41             wlan.disconnect()
42             wlan.active(False)
43             break
```

```

44     else:
45         print(data)
46         ret = conn.send(data)
47     except:
48         print("Close TCP-Server, please reset.")
49         if(listenSocket):
50             listenSocket.close()
51             wlan.disconnect()
52             wlan.active(False)

```

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

```

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

```

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

```

24     listenSocket = socket.socket()
25     listenSocket.bind((ip, port))
26     listenSocket.listen(1)
27     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.

```

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

```

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

```

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

```

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

```

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

```

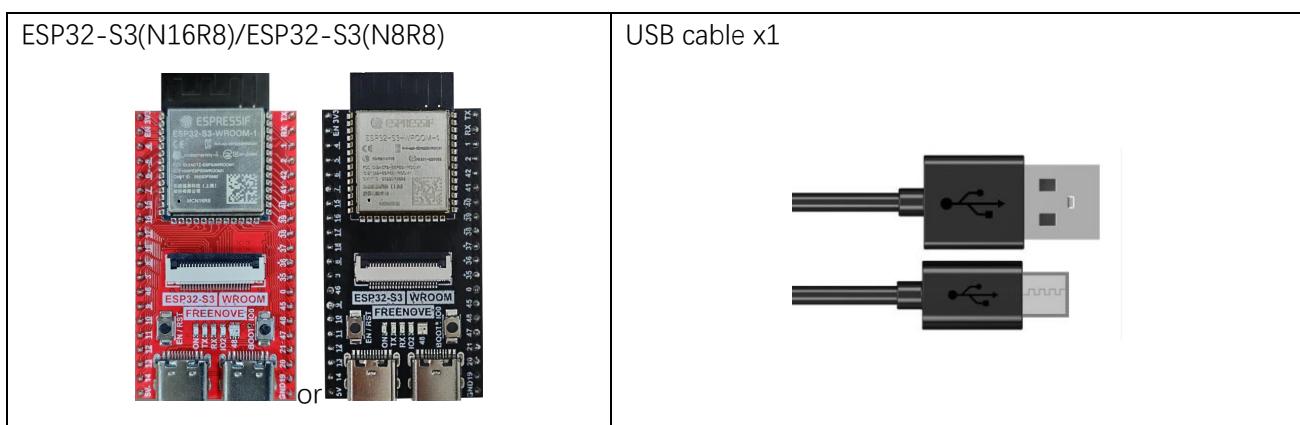
Chapter 5 Camera Web Server

In this section, we'll use ESP32's video function as an example to study.

Project 05.1 Camera Web Server

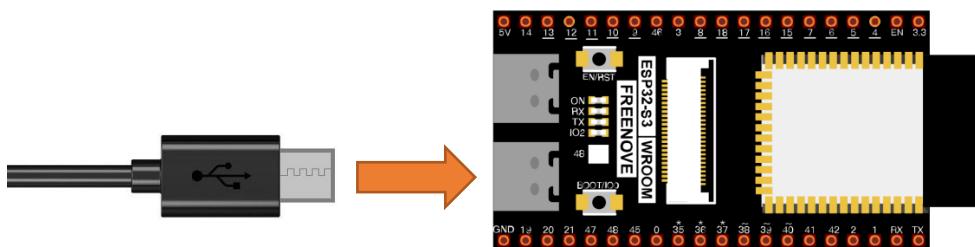
Connect ESP32 using USB and check its IP address through serial monitor. Use web page to access IP address to obtain video and image data.

Component List



Circuit

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



Code

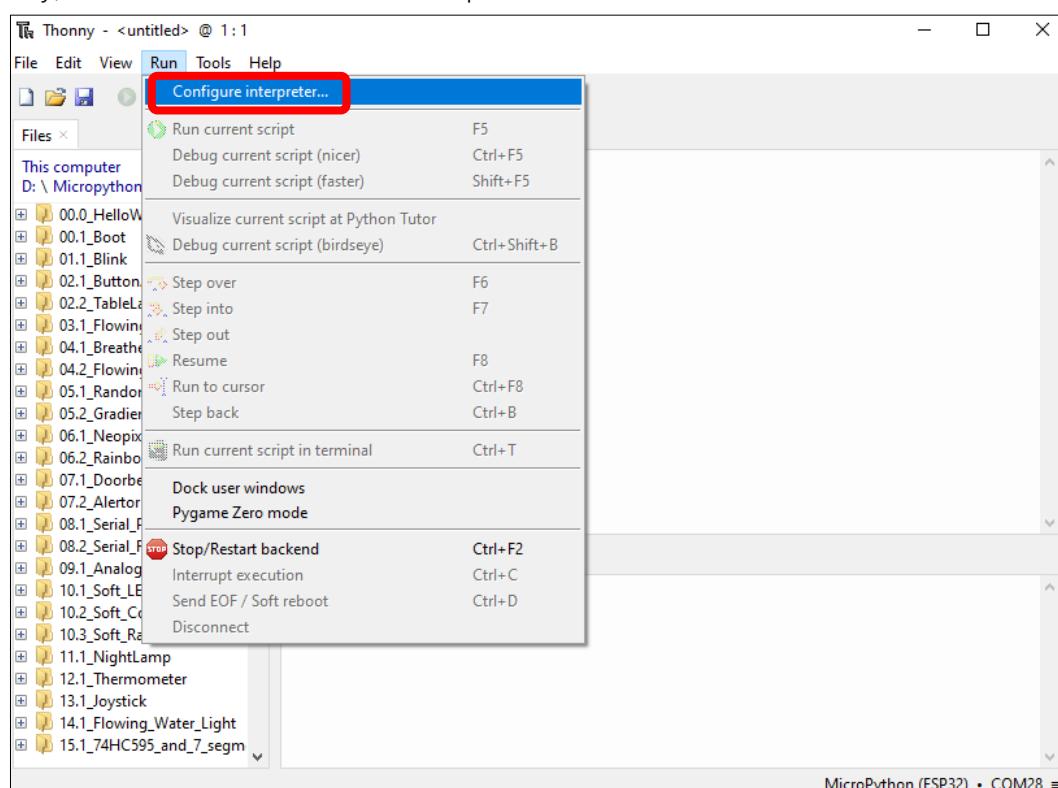
Move the program folder “**Freenove_ESP32_S3_WROVER_Board/Python/Python_Codes**” to disk(D) in advance with the path of “**D:/Micropython_Codes**”.

Since Micropython does not provide firmware including camera module, in this chapter, we will use the camera based on the firmware in lemariva's Github project, micropython-camera-driver.

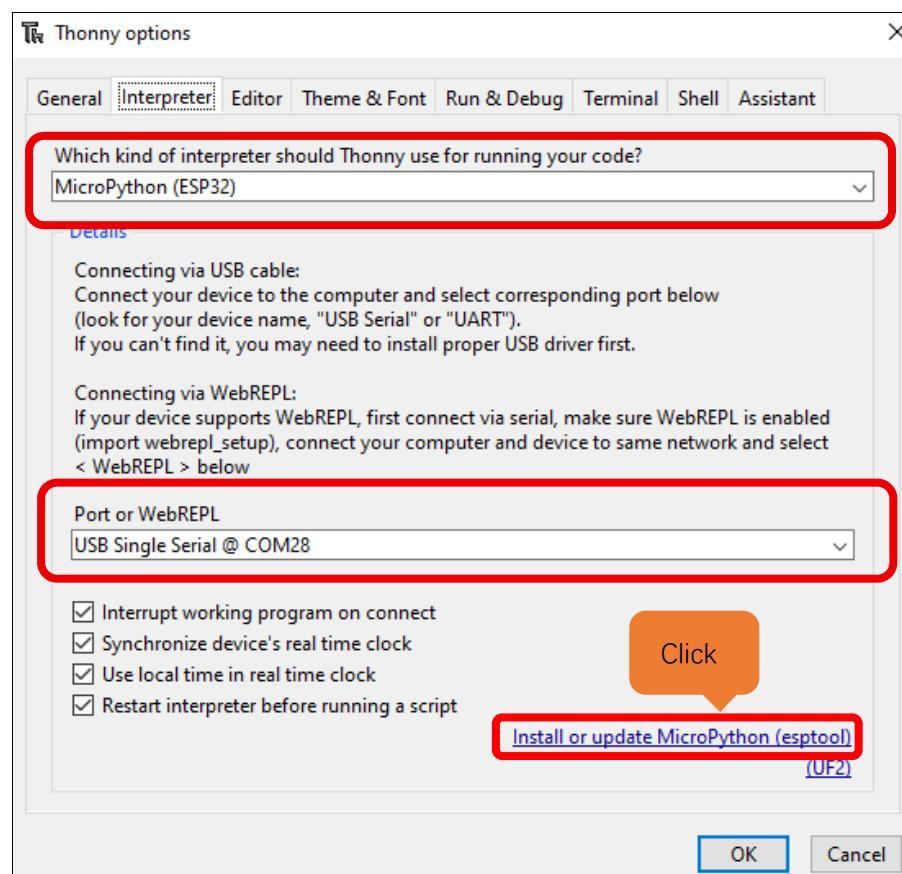
Project link: <https://github.com/lemariva/micropython-camera-driver>

Before starting the project, we need to re-upload the firmware with the camera module via steps below.

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

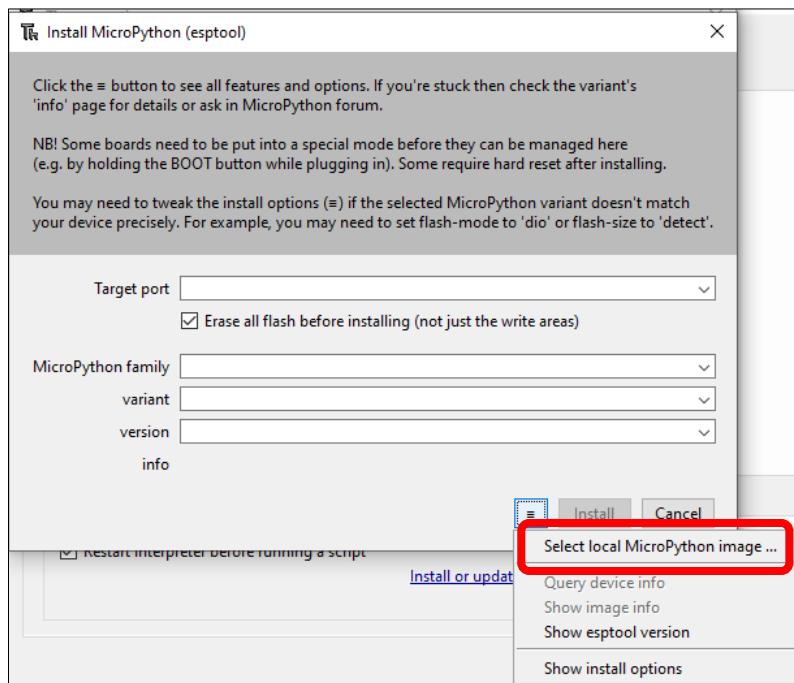


Select "MicroPython (ESP32)", select "USB Single Serial @ COM28", and then click the long button under "Firmware".

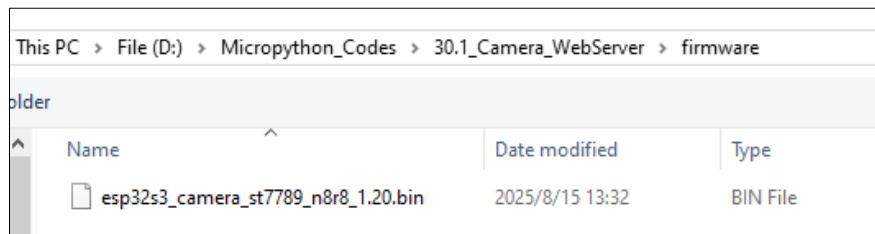


Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

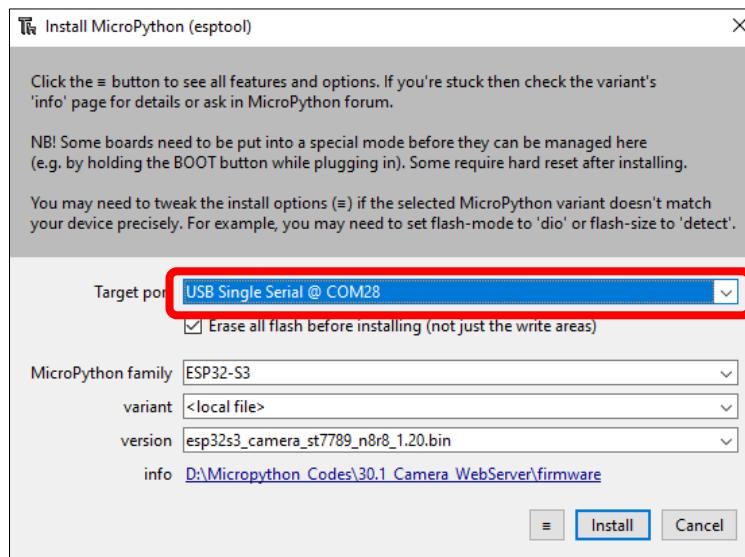
Click “Select local MicroPython image …”.



Choose “esp32s3_camera_st7789_n8r8_1.20.bin”



Select “USB Single Serial @ COM28”

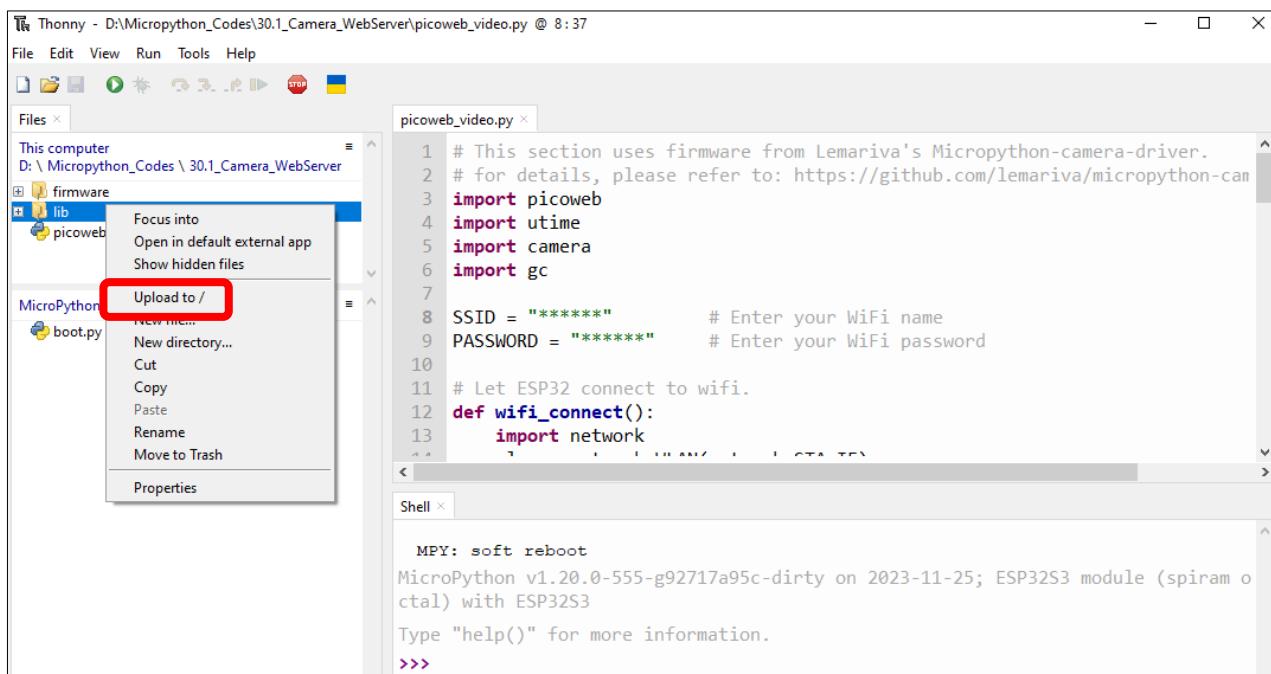


Click “Install”, Wait for completion.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “05.1_Camera_WebServer”. Select folder “lib”, right click your mouse to select “Upload to /”, wait for “lib” to be uploaded to ESP32-WROVER and then double click “picoweb_video.py”.

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

05.1_Camera_WebServer



Before running the program, please modify your router's name and password in the box shown in the illustration above to make sure that your code can compile and work successfully.

Click "run" to run the code "picoweb_video.py", then you can see the following content in the shell area.

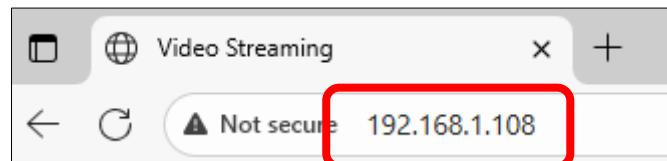
```
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
network config: '192.168.1.108' '255.255.255.0', '192.168.1.1', '192.168.1
.1')
* Running on http://0.0.0.0:80/
```

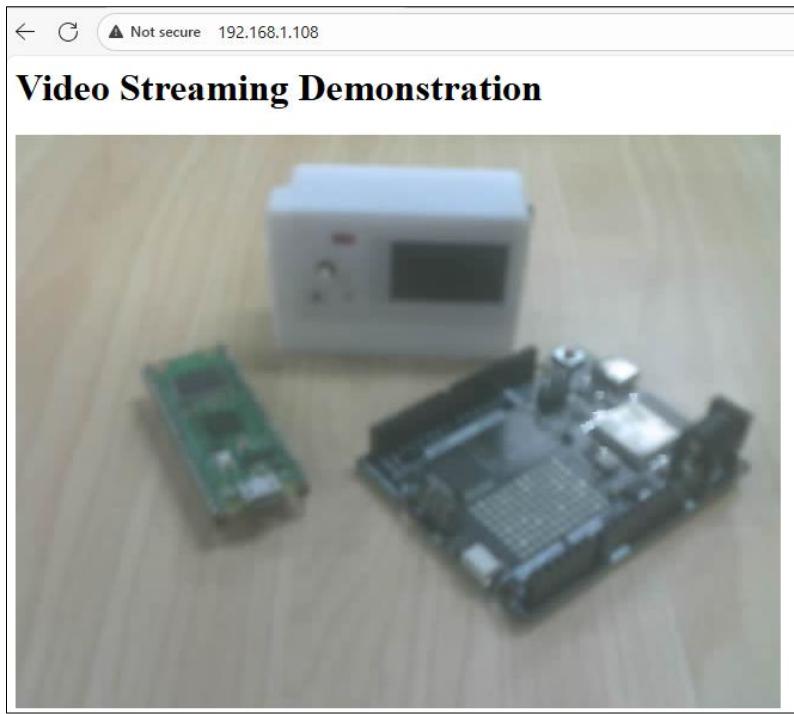
If your ESP32S3 has been in the process of connecting to router, but the information above has not been printed out, please re-check whether the router name and password have been entered correctly and press the reset key on ESP32S3 to wait for a successful connection prompt.

Open a web browser, enter the IP address printed by the serial monitor in the address bar, and access it.

Taking the Google browser as an example, here's what the browser prints out after successful access to ESP32S3's IP.



The effect is shown in the image below.



Please note:

If the shell area prompts an error when you click to run the code, please press the rst button on the ESP32S3, wait for the system reset to complete, and then re-run the code.

The following is the program code.

```
1 # This section uses firmware from Lemariva's Micropython-camera-driver.
2 # for details, please refer to: https://github.com/lemariva/micropython-camera-driver
3 import picoweb
4 import utime
5 import camera
6 import gc
7
8 SSID = "*****"          # Enter your WiFi name
9 PASSWORD = "*****"      # Enter your WiFi password
10
11 # Let ESP32 connect to wifi.
12 def wifi_connect():
13     import network
14     wlan = network.WLAN(network.STA_IF)
15     wlan.active(True)
16     if not wlan.isconnected():
17         print('connecting to network... ')
18         wlan.connect(SSID, PASSWORD)
19         start = utime.time()
20         while not wlan.isconnected():
21             utime.sleep(1)
```

```
22     if utime.time()-start > 5:
23         print("connect timeout!")
24         break
25     if wlan.isconnected():
26         print(' network config:', wlan.ifconfig())
27
28 # Initializing the Camera
29 def camera_init():
30     # Disable camera initialization
31     camera.deinit()
32     # Enable camera initialization
33     camera.init(0, d0=11, d1=9, d2=8, d3=10, d4=12, d5=18, d6=17, d7=16,
34             format=camera.JPEG, framesize=camera.FRAME_VGA,
35             xclk_freq=camera.XCLK_10MHz,
36             href=7, vsync=6, reset=-1, pwn=1,
37             sioc=5, siod=4, xclk=15, pclk=13, fb_location=camera.PSRAM)
38
39     camera.framesize(camera.FRAME_VGA) # Set the camera resolution
40     # The options are the following:
41     # FRAME_96X96 FRAME_QQVGA FRAME_QCIF FRAME_HQVGA FRAME_240X240
42     # FRAME_QVGA FRAME_CIF FRAME_HVGA FRAME_VGA FRAME_SVGA
43     # FRAME_XGA FRAME_HD FRAME_SXGA FRAME_UXGA
44     # Note: The higher the resolution, the more memory is used.
45     # Note: And too much memory may cause the program to fail.
46
47     camera.flip(0)                      # Flip up and down window: 0-1
48     camera.mirror(0)                    # Flip window left and right: 0-1
49     camera.saturation(0)                # saturation: -2,2 (default 0). -2 grayscale
50     camera.brightness(0)                # brightness: -2,2 (default 0). 2 brightness
51     camera.contrast(0)                 # contrast: -2,2 (default 0). 2 highcontrast
52     camera.quality(10)                 # quality: # 10-63 lower number means higher quality
53
54     # Note: The smaller the number, the sharper the image. The larger the number, the more
55     # blurry the image
56
57     camera.speffect(camera.EFFECT_NONE) # special effects:
58     # EFFECT_NONE (default) EFFECT_NEG EFFECT_BW EFFECT_RED EFFECT_GREEN EFFECT_BLUE
59     # EFFECT_RETRO
60     camera.whitebalance(camera.WB_NONE) # white balance
61     # WB_NONE (default) WB_SUNNY WB_CLOUDY WB_OFFICE WB_HOME
62
63     # HTTP Response Content
64     index_web="""
65     HTTP/1.0 200 OK\r\n
66     <html>
```



```

64 <head>
65   <title>Video Streaming</title>
66 </head>
67 <body>
68   <h1>Video Streaming Demonstration</h1>
69   
70 </body>
71 </html>
72 """
73
74 # HTTP Response
75 def index(req, resp):
76     # You can construct an HTTP response completely yourself, having
77     yield from resp.awrite(index_web)
78
79 # Send camera pictures
80 def send_frame():
81     buf = camera.capture()
82     yield b'--frame\r\n'
83     yield b'Content-Type: image/jpeg\r\n\r\n'
84     yield buf
85     del buf
86     gc.collect()
87
88 # Video transmission
89 def video(req, resp):
90     yield from picoweb.start_response(resp, content_type="multipart/x-mixed-replace;
91 boundary=frame")
92     while True:
93         yield from resp.awrite(next(send_frame()))
94         gc.collect()
95
96
97 ROUTES = [
98     # You can specify exact URI string matches...
99     ("/", index),
100    ("/video", video),
101 ]
102
103
104 if __name__ == '__main__':
105
106     import ulogging as logging
107     logging.basicConfig(level=logging.INFO)

```

```

108     camera_init()
109     wifi_connect()

110
111     #Create an app object that contains two decorators
112     app = picoweb.WebApp(__name__, ROUTES)

113
114     app.run(debug=1, port=80, host="0.0.0.0")
115     # debug values:
116     # -1 disable all logging
117     # 0 (False) normal logging: requests and errors
118     # 1 (True) debug logging
119     # 2 extra debug logging

```

Import picoweb、utime、camera、gc modules.

```

1 import picoweb
2 import utime
3 import camera
4 import gc

```

Before running the code, please modify the WiFi name and password in the code to ensure that the ESP32S3 can connect to the network.

```

8 SSID = "*****"          # Enter your WiFi name
9 PASSWORD = "*****"      # Enter your WiFi password

```

Define the WiFi connection function, set the ESP32S3 to STA mode, and let the ESP32S3 connect to the nearby WiFi. If the connection is successful, the WiFi configuration information of the ESP32S3 will be printed; if the connection fails, the connection timeout will be printed.

```

12 def wifi_connect():
13     import network
14     wlan = network.WLAN(network.STA_IF)
15     wlan.active(True)
16     if not wlan.isconnected():
17         print('connecting to network... ')
18         wlan.connect(SSID, PASSWORD)
19         start = utime.time()
20     while not wlan.isconnected():
21         utime.sleep(1)
22         if utime.time() - start > 5:
23             print("connect timeout!")
24             break
25     if wlan.isconnected():
26         print('network config:', wlan.ifconfig())

```

The deinit() is used to disable the configuration of the camera to prevent the previous configuration from interfering with the following configuration.

The init() is used to configure the camera's pin driver, image data format, resolution and other information. By default, please do not modify this function, otherwise the camera initialization fails and the image cannot be obtained.

Any concerns? ✉ support@freenove.com

```

31 camera.deinit()
32 # Enable camera initialization
33 camera.init(0, d0=11, d1=9, d2=8, d3=10, d4=12, d5=18, d6=17, d7=16,
34             format=camera.JPEG, framesize=camera.FRAME_VGA,
35             xclk_freq=camera.XCLK_10MHz,
36             href=7, vsync=6, reset=-1, pwn=1,
37             sioc=5, siod=4, xclk=15, pclk=13, fb_location=camera.PSRAM)

```

This function can set the resolution of the camera individually, you can refer to the notes below to select the appropriate resolution size.

```

39 camera.framesize(camera.FRAME_VGA) # Set the camera resolution
40 # The options are the following:
41 # FRAME_96X96 FRAME_QQVGA FRAME_QCIF FRAME_HQVGA FRAME_240X240
42 # FRAME_QVGA FRAME_CIF FRAME_HVGA FRAME_VGA FRAME_SVGA
43 # FRAME_XGA FRAME_HD FRAME_SXGA FRAME_UXGA
44 # Note: The higher the resolution, the more memory is used.
45 # Note: And too much memory may cause the program to fail.

```

The following functions can modify the image information obtained by the camera.

```

47 camera.flip(0)                      # Flip up and down window: 0-1
48 camera.mirror(0)                    # Flip window left and right: 0-1
49 camera.saturation(0)                # saturation: -2, 2 (default 0). -2 grayscale
50 camera.brightness(0)                # brightness: -2, 2 (default 0). 2 brightness
51 camera.contrast(0)                  # contrast: -2, 2 (default 0). 2 highcontrast
52 camera.quality(10)                 # quality: # 10-63 lower number means higher quality
53 # Note: The smaller the number, the sharper the image. The larger the number, the more blurry
54 the image
55
56 camera.speffect(camera.EFFECT_NONE) # special effects:
57 # EFFECT_NONE (default) EFFECT_NEG EFFECT_BW EFFECT_RED EFFECT_GREEN EFFECT_BLUE EFFECT_RETRO
58 camera.whitebalance(camera.WB_NONE)  # white balance
59 # WB_NONE (default) WB_SUNNY WB_CLOUDY WB_OFFICE WB_HOME

```

This is the code for a simple web interface, used here as an example.

```

61 index_web"""
62 HTTP/1.0 200 OK\r\n
63 <html>
64   <head>
65     <title>Video Streaming</title>
66   </head>
67   <body>
68     <h1>Video Streaming Demonstration</h1>
69     
70   </body>
71 </html>
72 """

```

Web page response function. When a user visits the webpage "/" built by ESP32S3, ESP32S3 calls this function,

allowing the user to observe a display interface in the browser.

```
75 def index(req, resp):
76     # You can construct an HTTP response completely yourself, having
77     yield from resp.awrite(index_web)
```

send_frame() can send the image obtained by ESP32S3 in web page format. When someone visits the webpage "/video" built by the ESP32S3, the video(req, resp) function is used to continuously fetch images and send them to the browser.

```
79 # Send camera pictures
80 def send_frame():
81     buf = camera.capture()
82     yield (b'--frame\r\n'
83             b'Content-Type: image/jpeg\r\n\r\n'
84             + buf + b'\r\n')
85     del buf
86     gc.collect()
87
88 # Video transmission
89 def video(req, resp):
90     yield from picoweb.start_response(resp, content_type="multipart/x-mixed-replace;
91 boundary=frame")
92     while True:
93         yield from resp.awrite(next(send_frame()))
94         gc.collect()
```

Create two route decorators and declare their listening strings and corresponding response handlers respectively.

```
97 ROUTES = [
98     # You can specify exact URI string matches...
99     ("/", index),
100    ("/video", video),
101 ]
```

This is the main part of the program. First initialize the ESP32S3 camera, and then configure WiFi to connect the ESP32S3 to the network. Call the picoweb library, build a webserver, and run it.

```
103 import ulogging as logging
104 logging.basicConfig(level=logging.INFO)
105 camera_init()
106 wifi_connect()
107
108 #Create an app object that contains two decorators
109 app = picoweb.WebApp(__name__, ROUTES)
110
111 app.run(debug=1, port=80, host="0.0.0.0")
```



Reference

Image resolution	Sharpness	Image resolution	Sharpness
FRAMESIZE_96x96	96x96	FRAMESIZE_HVGA	480x320
FRAMESIZE_QQVGA	160x120	FRAMESIZE_VGA	640x480
FRAMESIZE_QCIF	176x144	FRAMESIZE_SVGA	800x600
FRAMESIZE_HQVGA	240x176	FRAMESIZE_XGA	1024x768
FRAMESIZE_240x240	240x240	FRAMESIZE_HD	1280x720
FRAMESIZE_QVGA	320x240	FRAMESIZE_SXGA	1280x1024
FRAMESIZE_CIF	400x296	FRAMESIZE_UXGA	1600x1200

We recommend that the resolution not exceed VGA(640x480).

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.

Any concerns? ✉ support@freenove.com