Welcome

Thank you for choosing Freenove products!

How to Start

When reading this, you should have downloaded the ZIP file for this product.
Unzip it and you will get a folder containing tutorials and related files. Please start with this PDF tutorial.

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

Get Support

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

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

support@freenove.com

Attention

Pay attention to safety when using and storing this product:

- This product is not suitable for children under 12 years of age because of small parts and sharp parts.
- Minors should use this product under the supervision and guidance of adults.
- This product contains small and sharp parts. Do not swallow, prick and scratch to avoid injury.
- This product contains conductive parts. Do not hold them to touch power supply and other circuits.
- To avoid personal injury, do not touch parts rotating or moving while working.
- The wrong operation may cause overheat. Do not touch and disconnect the power supply immediately.
- Operate in accordance with the requirements of the tutorial. Fail to do so may damage the parts.
- Store this product in a dry and dark environment. Keep away from children.
- Turn off the power of the circuit before leaving.

About

Freenove provides open source electronic products and services.

Freenove is committed to helping customers learn programming and electronic knowledge, quickly implement product prototypes, realize their creativity and launch innovative products. Our services include:

- Kits for learning programming and electronics
- Kits compatible with Arduino®, Raspberry Pi®, micro:bit®, ESP32®, etc.
- Kits for robots, smart cars, drones, etc.
- Components, modules and tools
- Design and customization

To learn more about us or get our latest information, please visit our website:

http://www.freenove.com

Copyright

All the files provided in the ZIP file are released under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License. You can find a copy of the license in the ZIP file.

It means you can use these files on your own derived works, in part or completely. But not for commercial use.

Freenove® brand and logo are trademarks of Freenove Creative Technology Co., Ltd. Must not be used without permission.

FREENOVE™

Other registered trademarks and their owners appearing in this document:

Arduino® is a trademark of Arduino LLC (https://www.arduino.cc/).
Raspberry Pi® is a trademark of Raspberry Pi Foundation (https://www.raspberrypi.org/).
micro:bit® is a trademark of Micro:bit Educational Foundation (https://www.microbit.org/).
ESPRESSIF® and ESP32® are trademarks of ESPRESSIF Systems (Shanghai) Co., Ltd (https://www.espressif.com/).

Any concerns? ✉ support@freenove.com

Contents

Welcome i
Contents 1
Prepare 3
ESP32-WROVER 4
Extension board of the ESP32-WROVER 6
Chapter 0 Ready (Important) 7
0.1 Installing Thonny (Important) 7
0.2 Basic Configuration of Thonny 12
0.3 Installing CH340 (Important) 14
0.4 Burning Micropython Firmware (Important) 25
0.5 Testing codes (Important) 30
0.6 Thonny Common Operation 37
0.7 Note 42
Chapter 1 LED (Important) 44
Project 1.1 Blink 44
Project 1.2 Blink 53
Chapter 2 Button & LED 61
Project 2.1 Button & LED 61
Project 2.2 MINI table lamp 68
Chapter 3 LED Bar 73
Project 3.1 Flowing Light 73
Chapter 4 Analog & PWM 79
Project 4.1 Breathing LED 79
Project 4.2 Meteor Flowing Light 86
Chapter 5 RGBLED 92
Project 5.1 Random Color Light 92
Project 5.2 Gradient Color Light 98
Chapter 6 Buzzer 100
Project 6.1 Doorbell 100
Project 6.2 Alertor 106

Chapter 7 Serial Communication 108
Project 7.1 Serial Print 108
Project 7.2 Serial Read and Write 113
Chapter 8 AD/DA Converter 114
Project 8.1 Read the Voltage of Potentiometer 114
Chapter 9 TouchSensor 122
Project 9.1 Read Touch Sensor 122
Project 9.2 TouchLamp 127
Chapter 10 Potentiometer & LED 132
Project 10.1 Soft Light 132
Chapter 11 Photoresistor & LED 135
Project 11.1 NightLamp 135
Chapter 12 Thermistor 139
Project 12.1 Thermometer 139
Chapter 13 Bluetooth 145
Project 13.1 Bluetooth Low Energy Data Passthrough 145
Project 13.2 Bluetooth Control LED 155
Chapter 14 WiFi Working Modes 161
Project 14.1 Station mode 161
Project 14.2 AP mode 165
Project 14.3 AP+Station mode 169
Chapter 15 TCP/IP 173
Project 15.1 As Client 173
Project 15.2 As Server 184
Chapter 16 Camera Web Server 189
Project 16.1 Camera Web Server 189
What's next? 200
End of the Tutorial 200

Prepare

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

ESP32 can be developed both either with C/C++ language or micropython language. In this tutorial, we use micropython. With Micropython is as easy to learn as Python with little code, making it ideal for beginners. Moreover, the code of ESP32 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.

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 ESP32 and its 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

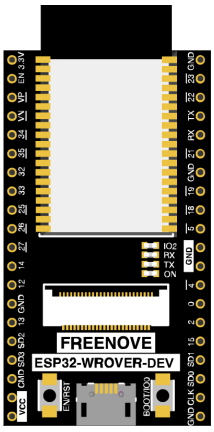
ESP32-WROVER

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

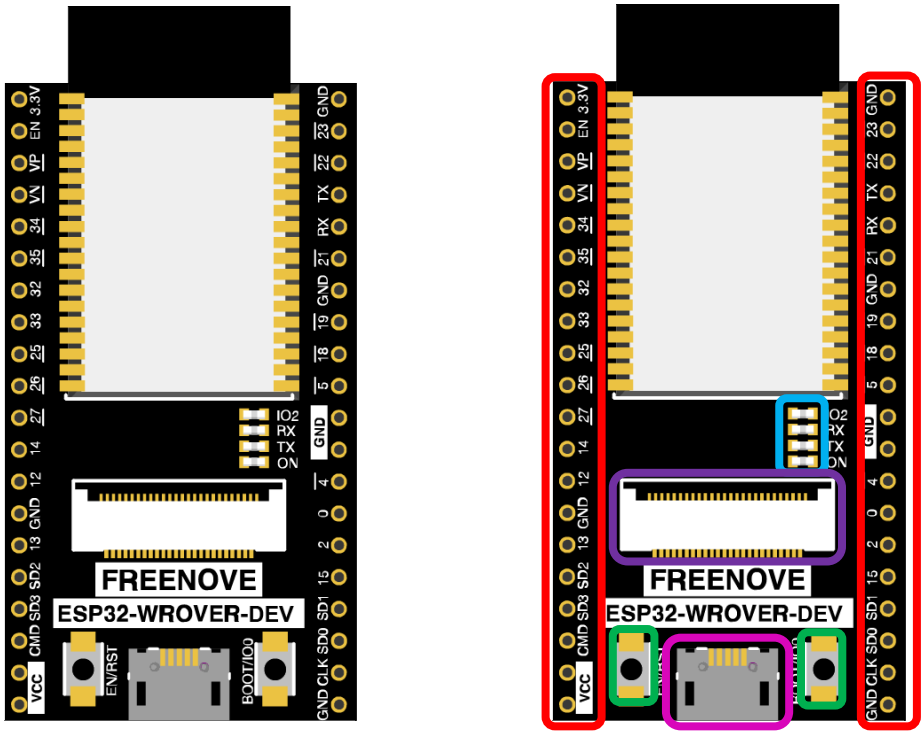
PCB on-board antenna	IPEX antenna

In this tutorial, the ESP32-WROVER is designed based on the PCB on-board antenna package.

ESP32-WROVER



The hardware interfaces of ESP32-WROVER are distributed as follows:



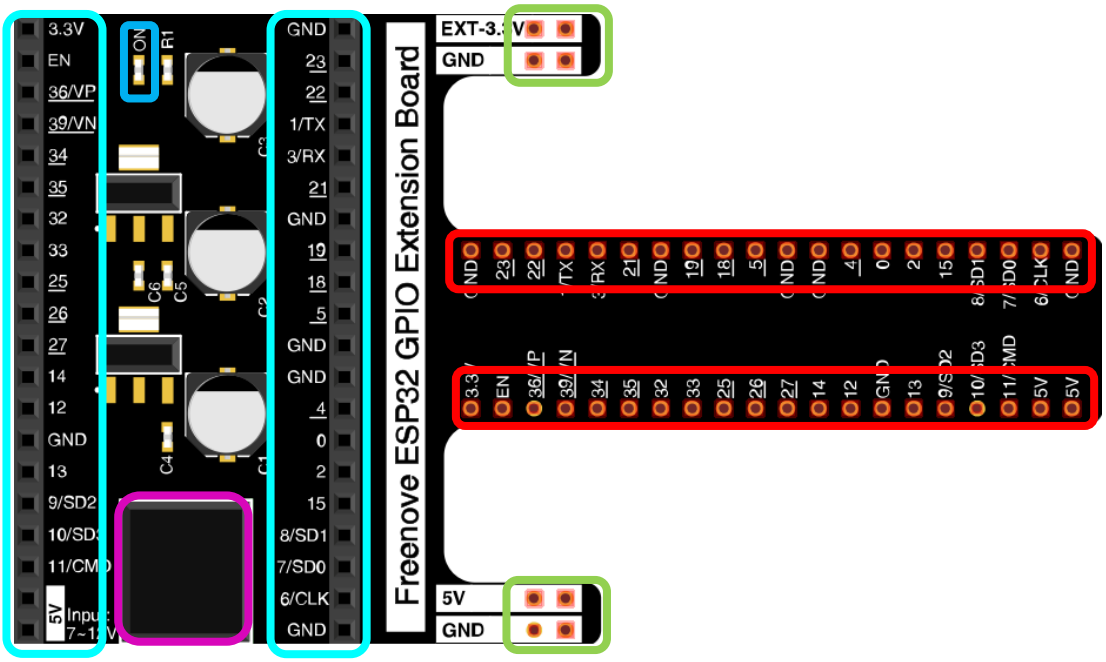
Compare the left and right images. We've boxed off the resources on the ESP32-WROVER in different colors to facilitate your understanding of the ESP32-WROVER.

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

Extension board of the ESP32-WROVER

And we also design an extension board, so that you can use the ESP32 more easily in accordance with the circuit diagram provided. The followings are their photos. All the projects in this tutorial are studied with this ESP32-WROVER.

The hardware interfaces of ESP32-WROVER are distributed as follows:



We've boxed off the resources on the ESP32-WROVER in different colors to facilitate your understanding of the ESP32-WROVER.

Box color	Corresponding resources introduction
	GPIO pin
	LED indicator
	GPIO interface of development board
	Power supplied by the extension board
	External power supply

In ESP32, GPIO is an interface to control peripheral circuit. For beginners, it is necessary to learn the functions of each GPIO. The following is an introduction to the GPIO resources of the ESP32-WROVER development board.

Later, we only use USB cable to power ESP32-WROVER in default.

In the whole tutorial, we don't use T extension to power ESP32-WROVER. So 5V and 3.3V (include EXT 3.3V) on the extension board are from ESP32-WROVER.

We can also use DC jack of extension board to power ESP32-WROVER. Then 5v and EXT 3.3v on extension board are from external power resource.

For more information, please visit:

https://www.espressif.com/sites/default/files/documentation/esp32-wrover_datasheet_en.pdf

Chapter 0 Ready (Important)

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

0.1 Installing Thonny (Important)

Thonny is a free, open-source software platform with compact size, simple interface, simple operation and rich functions, making it a Python IDE for beginners. In this tutorial, we use this IDE to develop ESP32 during the whole process.
Thonny supports various operating system, including Windows、Mac OS、Linux.

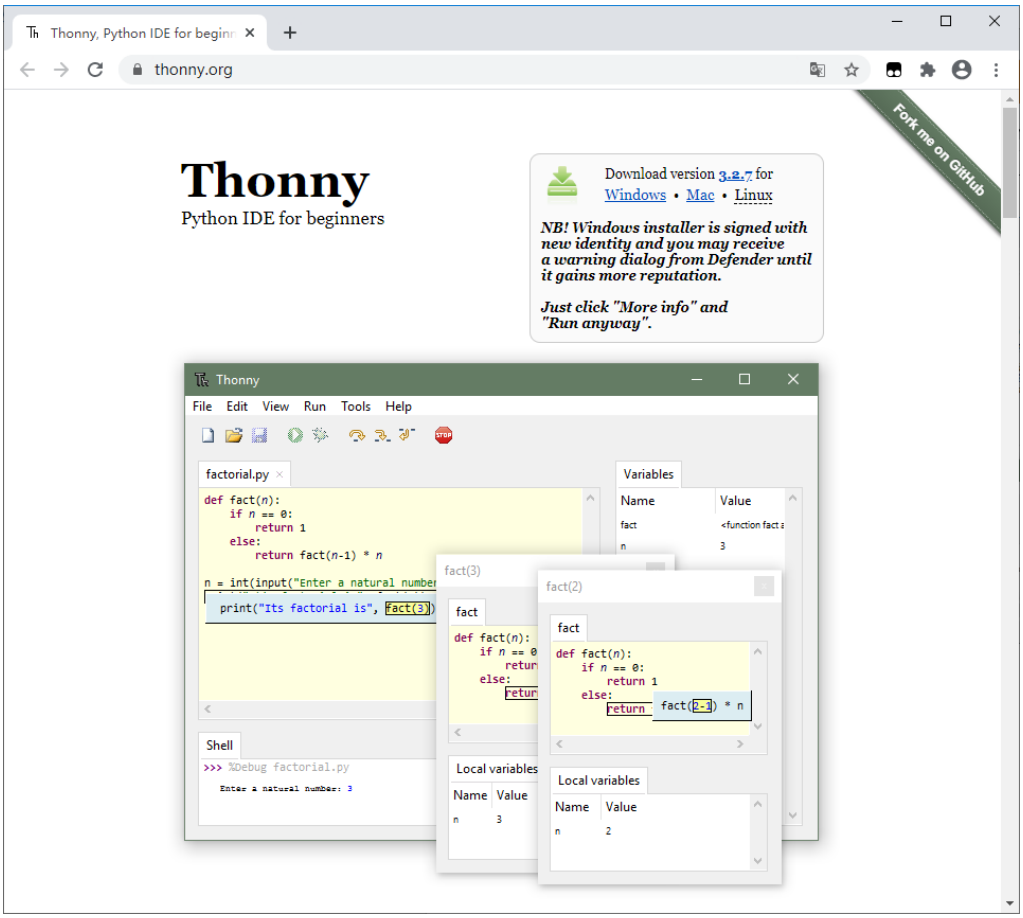
Downloading Thonny

Official website of Thonny: https://thonny.org
Open-source code repositories of Thonny: https://github.com/thonny/thonny

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

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

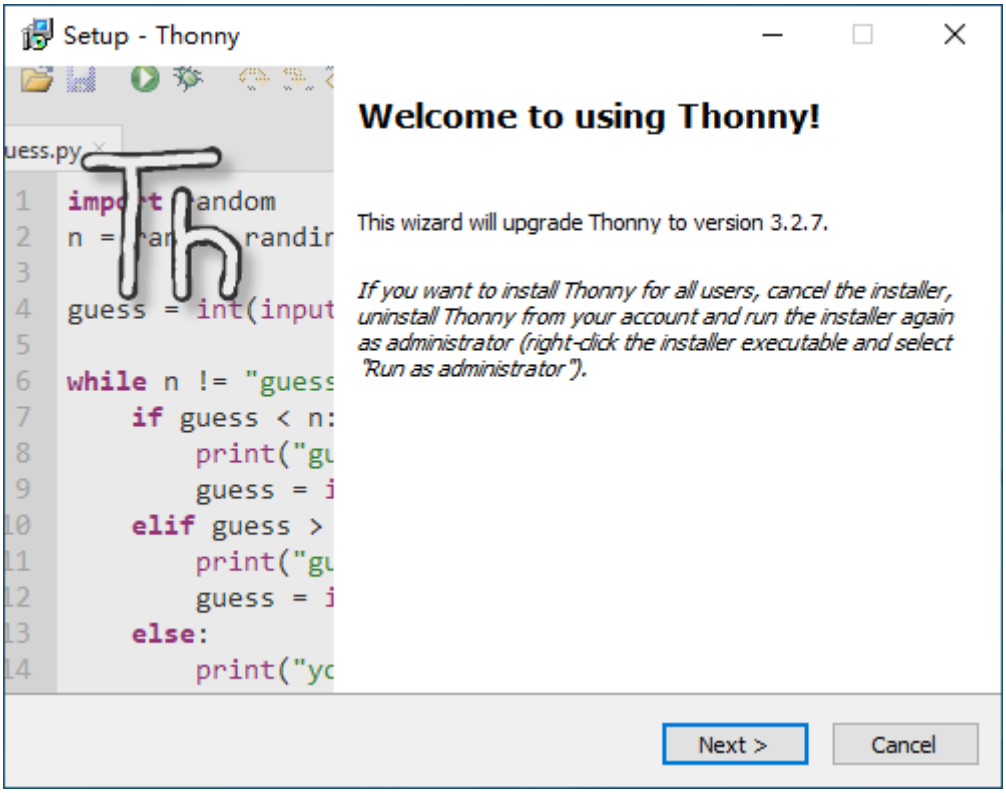
You can also open "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Software**", we have prepared it in advance.



Installing on Windows

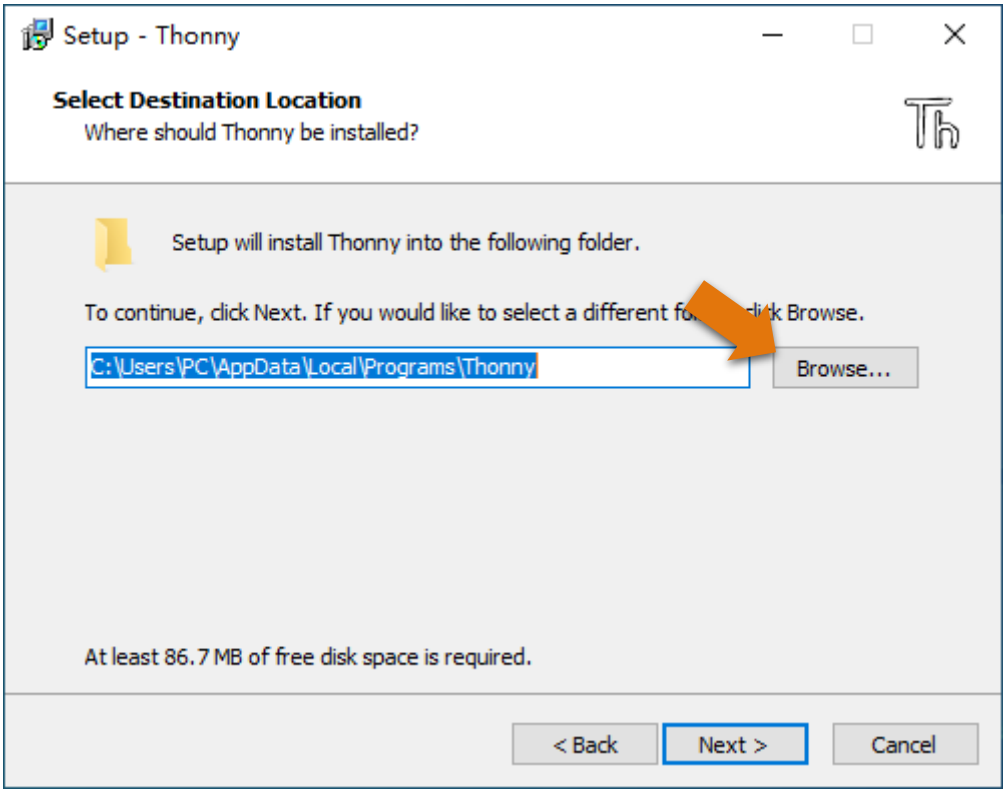
The icon of Thonny after downloading is as below. Double click “thonny-3.2.7.exe”.

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

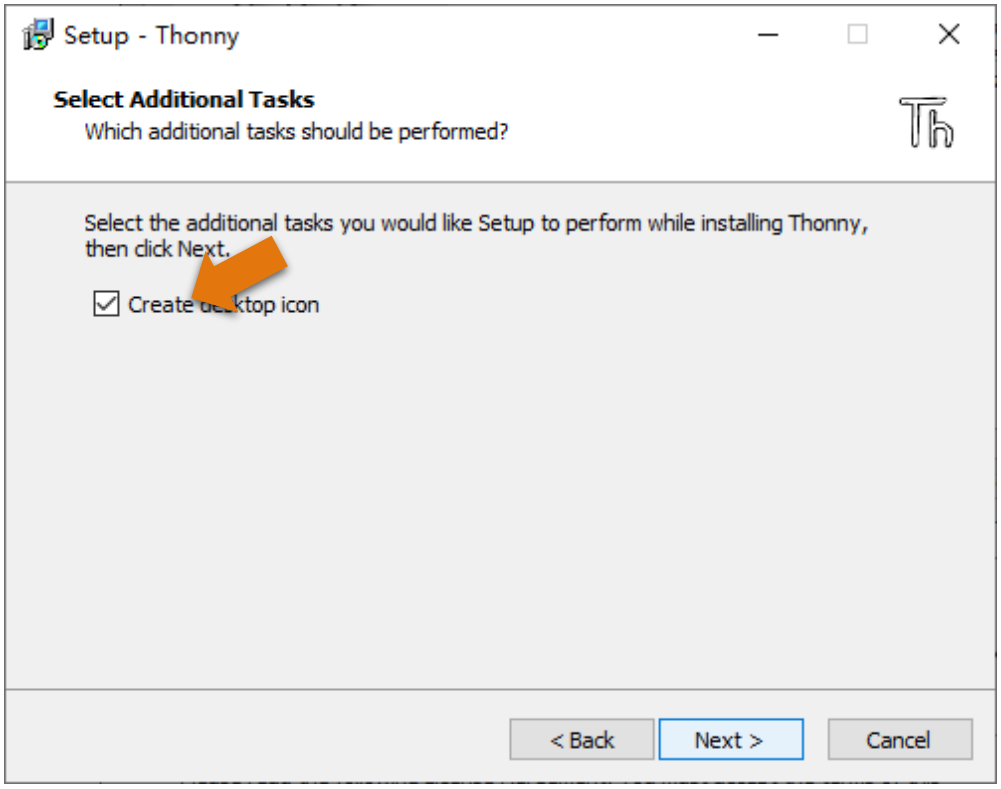


If you want to change Thonny's installation path, you can click "Browse" to modify it. After selecting installation path, click "OK".

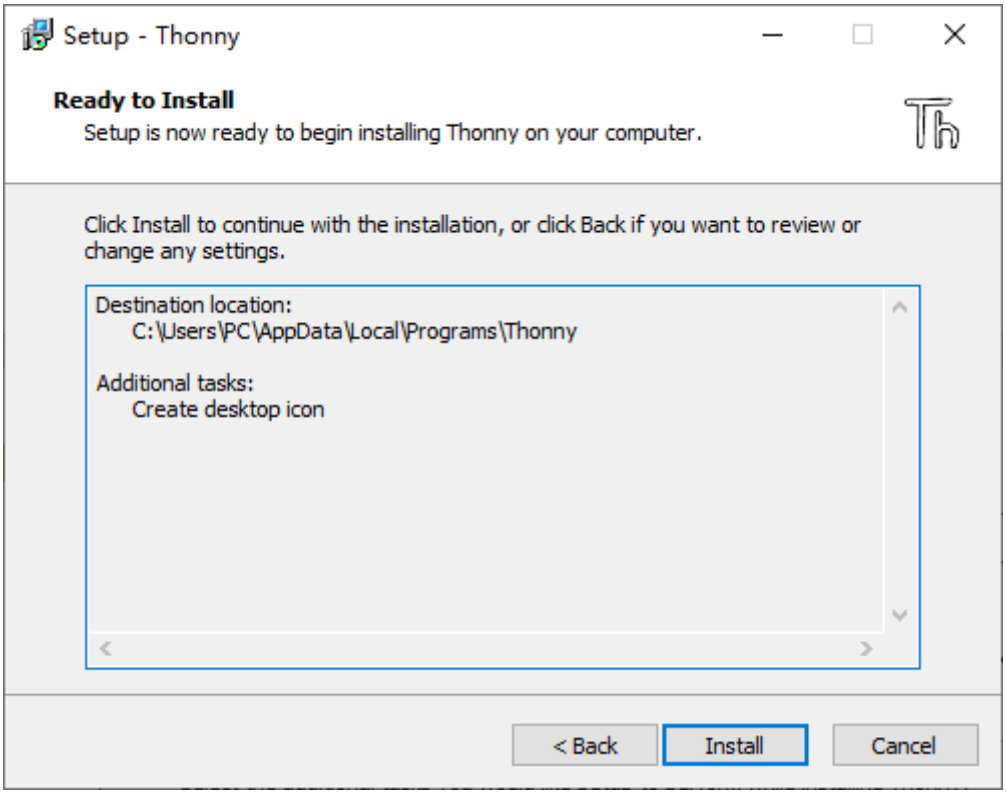
If you do not want to change it, just click "Next".



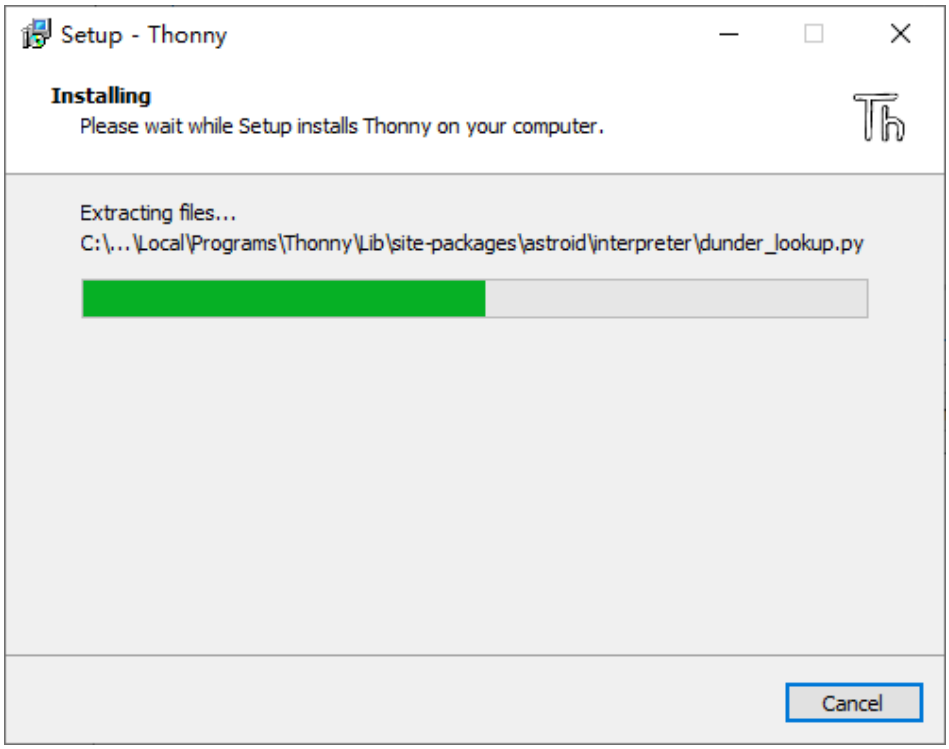
Check "Create desktop icon" and then it will generate a shortcut on your desktop to facilitate you to open Thonny later.



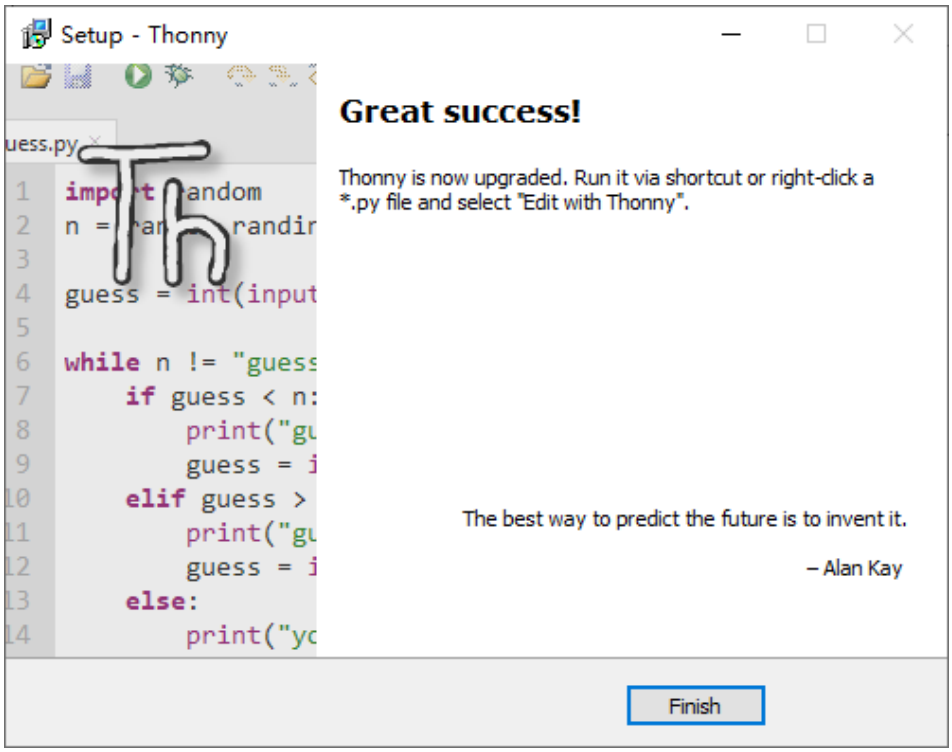
Click "install" to install the software.



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



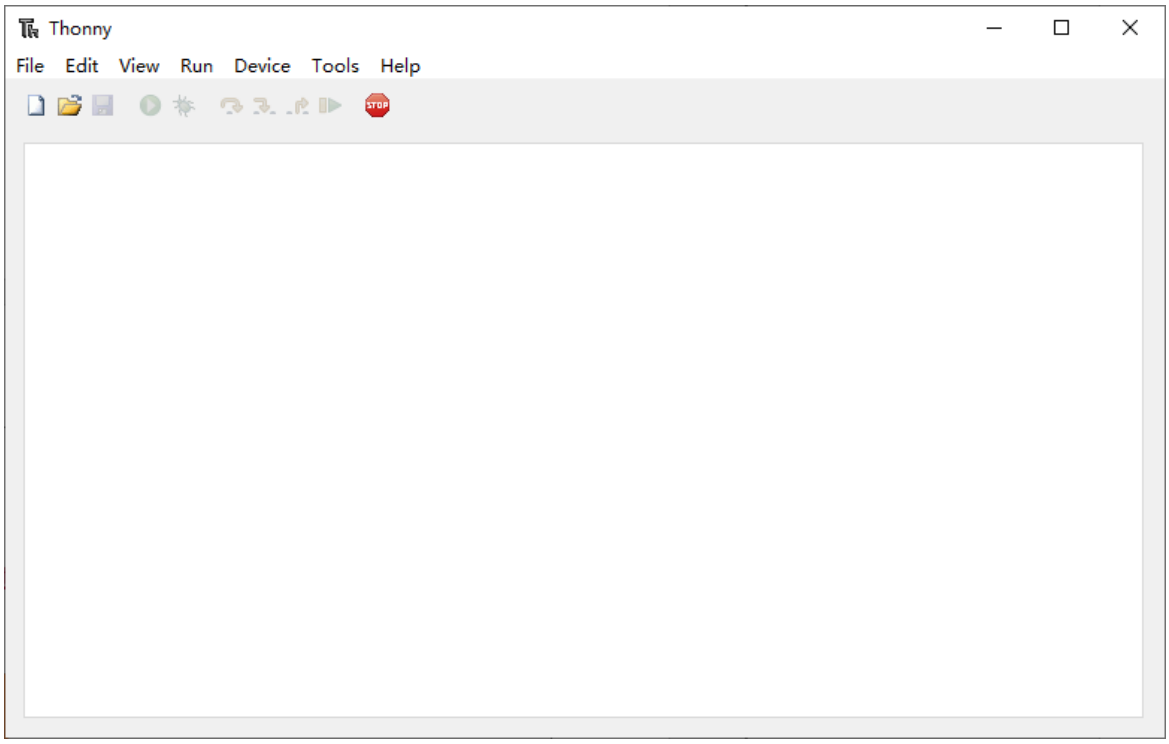
Once you see the interface as below, Thonny has been installed successfully.



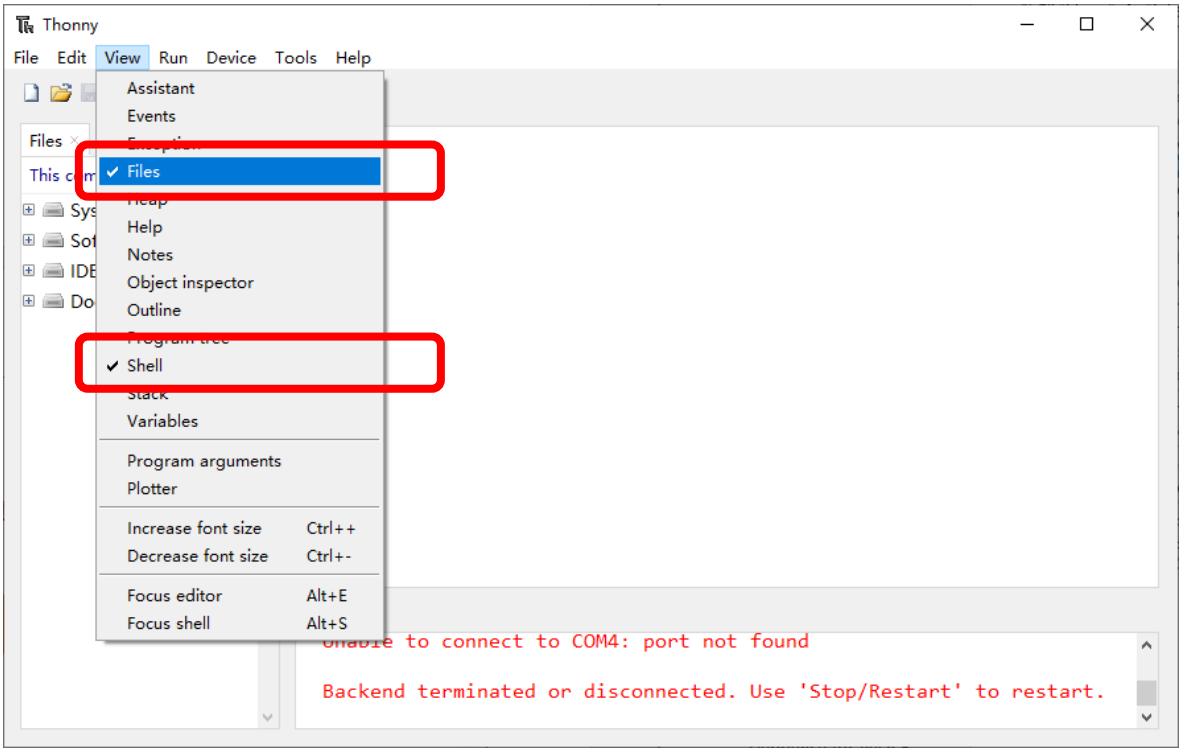
If you've check "Create desktop icon" during the installation process, you can see the below icon on your desktop.。

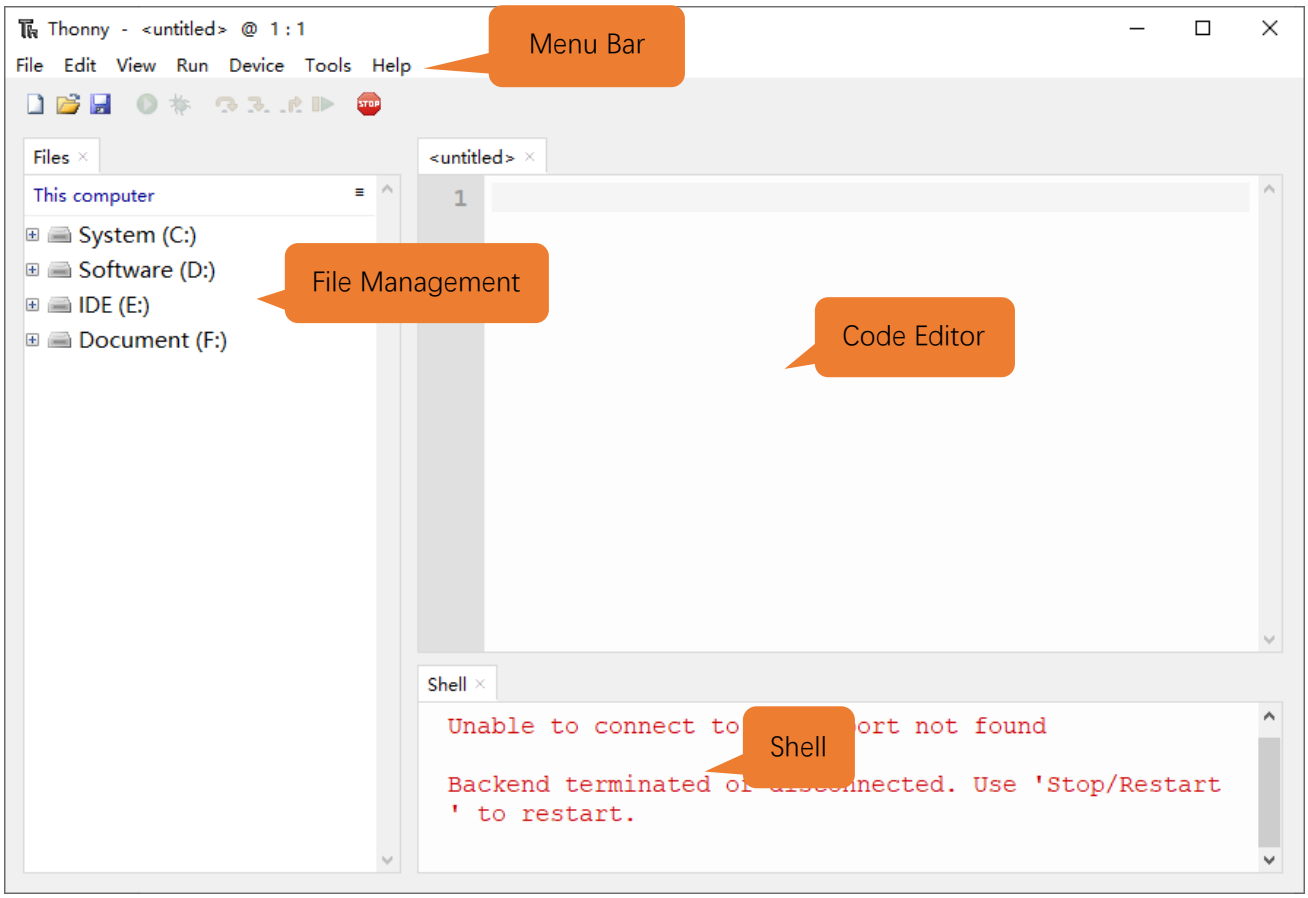
0.2 Basic Configuration of Thonny

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



Select "View"→ "Files" and "Shell".




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

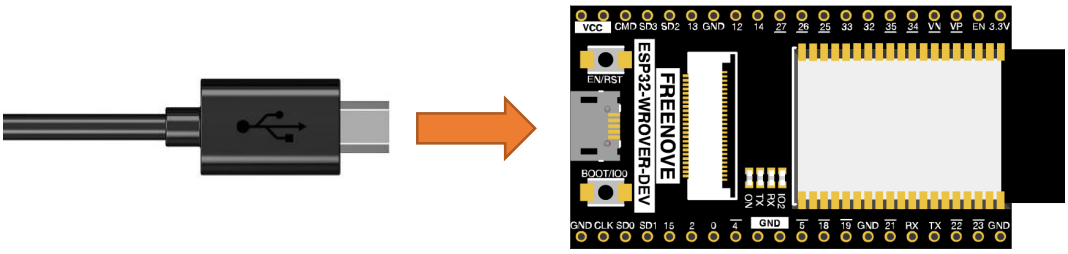
0.3 Installing CH340 (Important)

ESP32 uses CH340 to download codes. So before using it, we need to install CH340 driver in our computers.

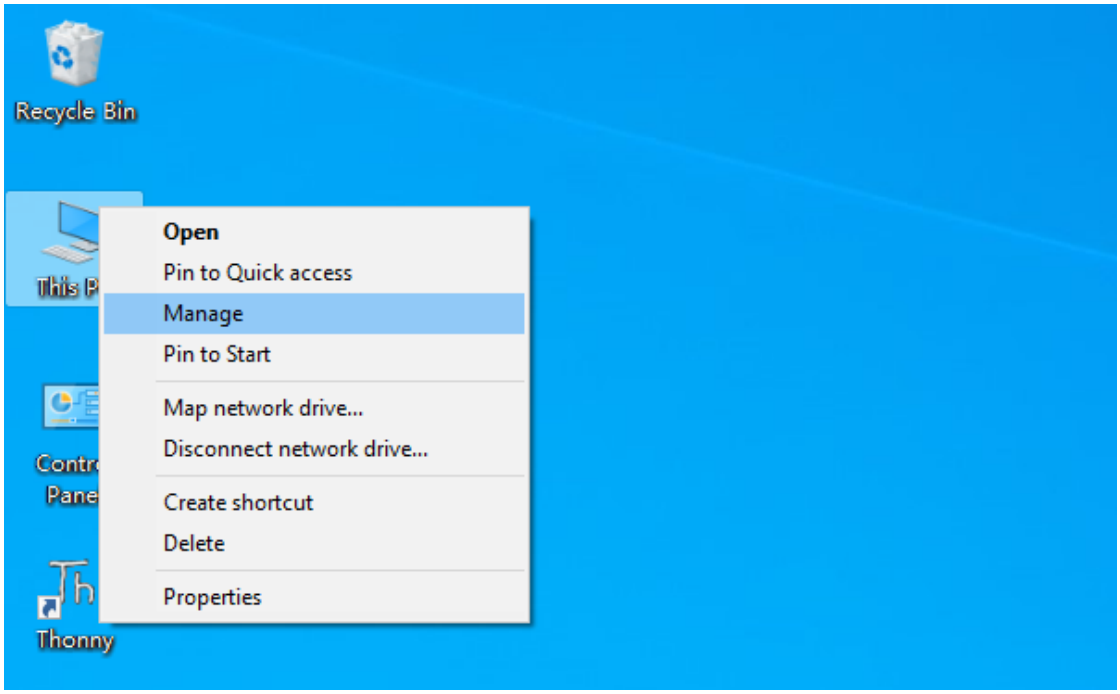
Windows

Check whether CH340 has been installed

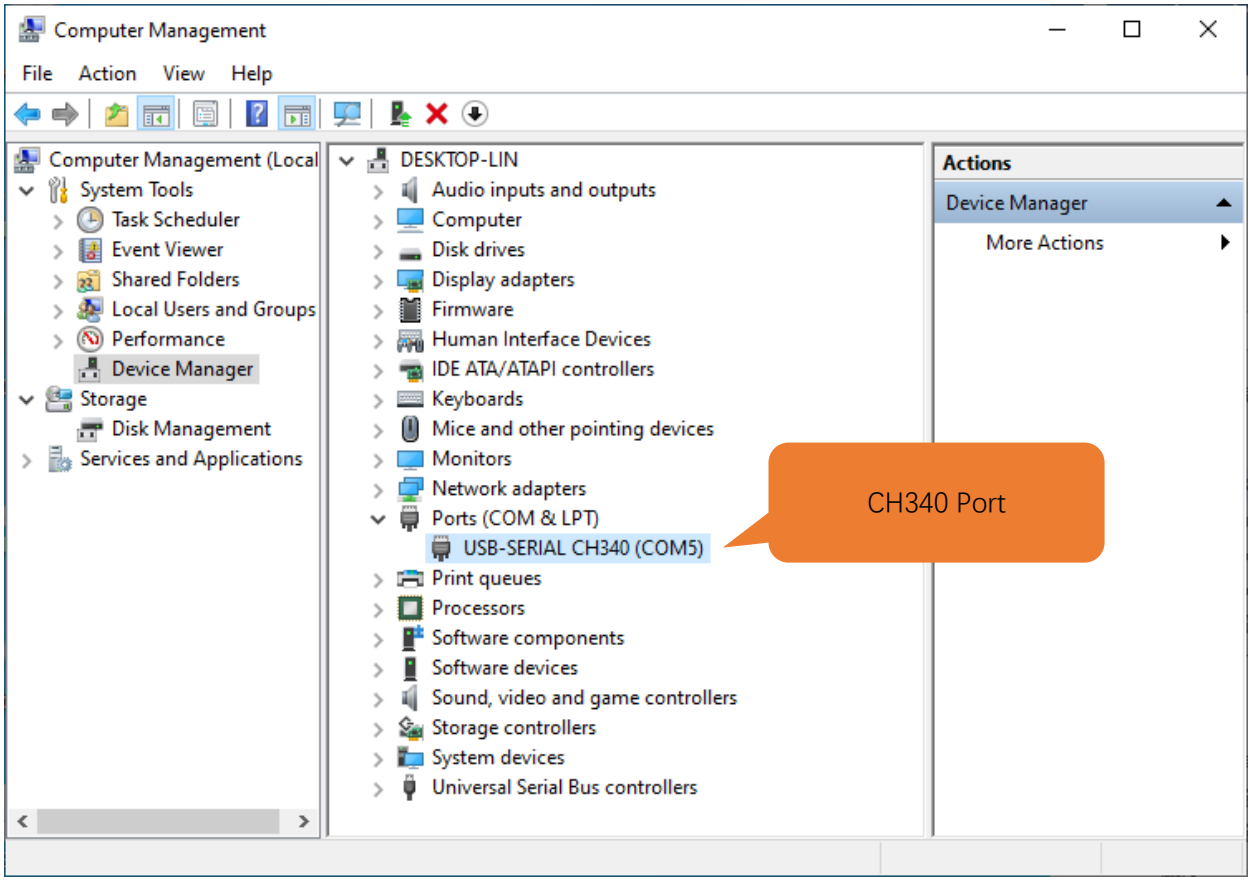
1. Connect your computer and ESP32 with a USB cable.



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

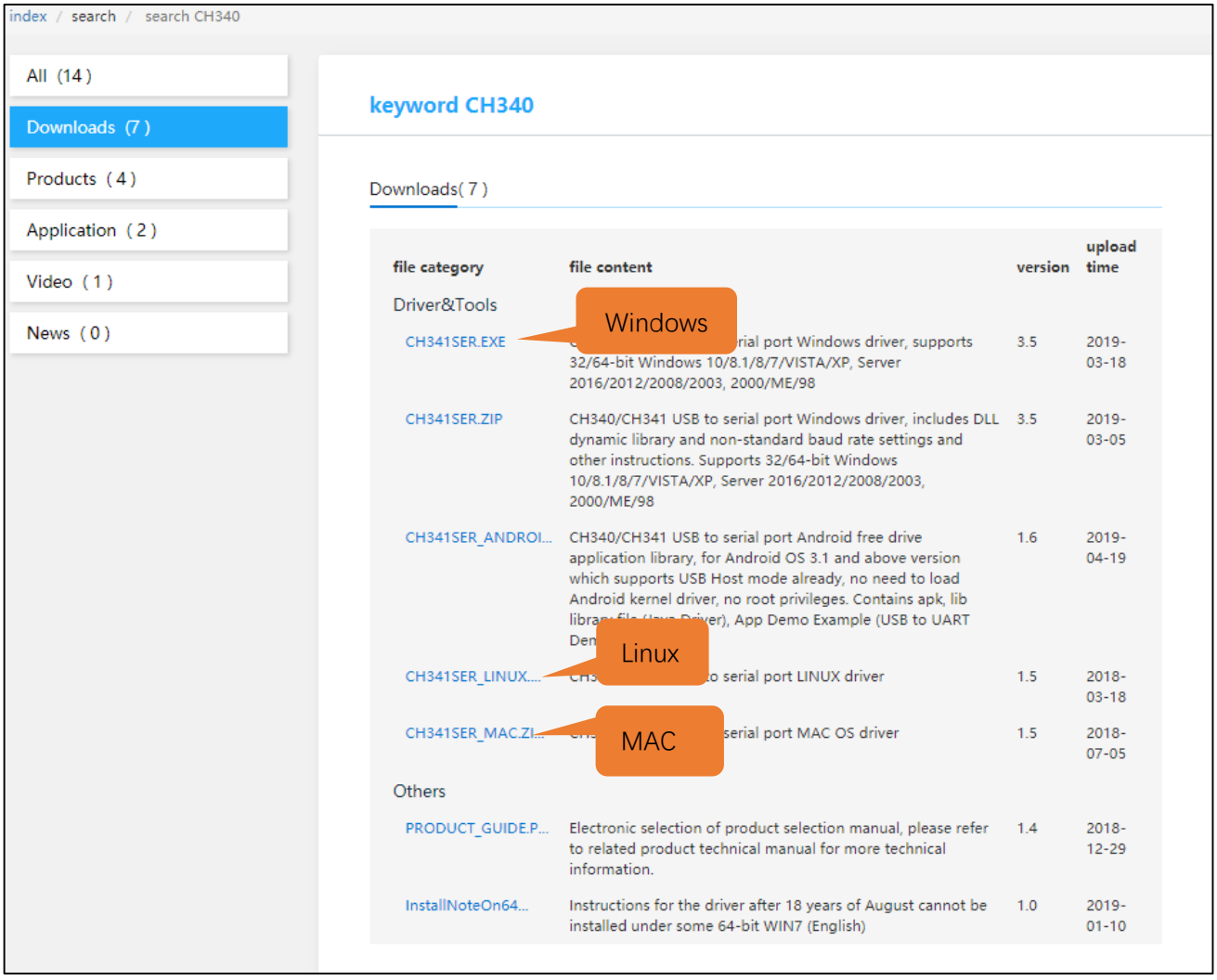


3. Click "Device Manager". If your computer has installed CH340, you can see"USB-SERIAL CH340 (COMx)". And you can click here to move to the next step.

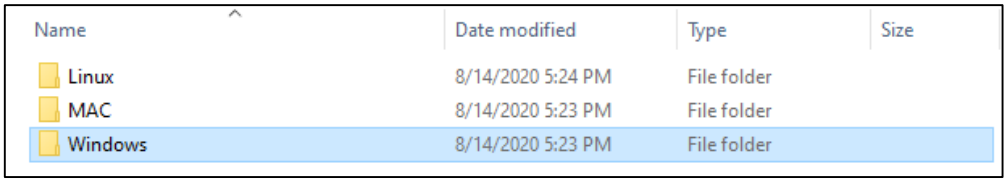


Installing CH340

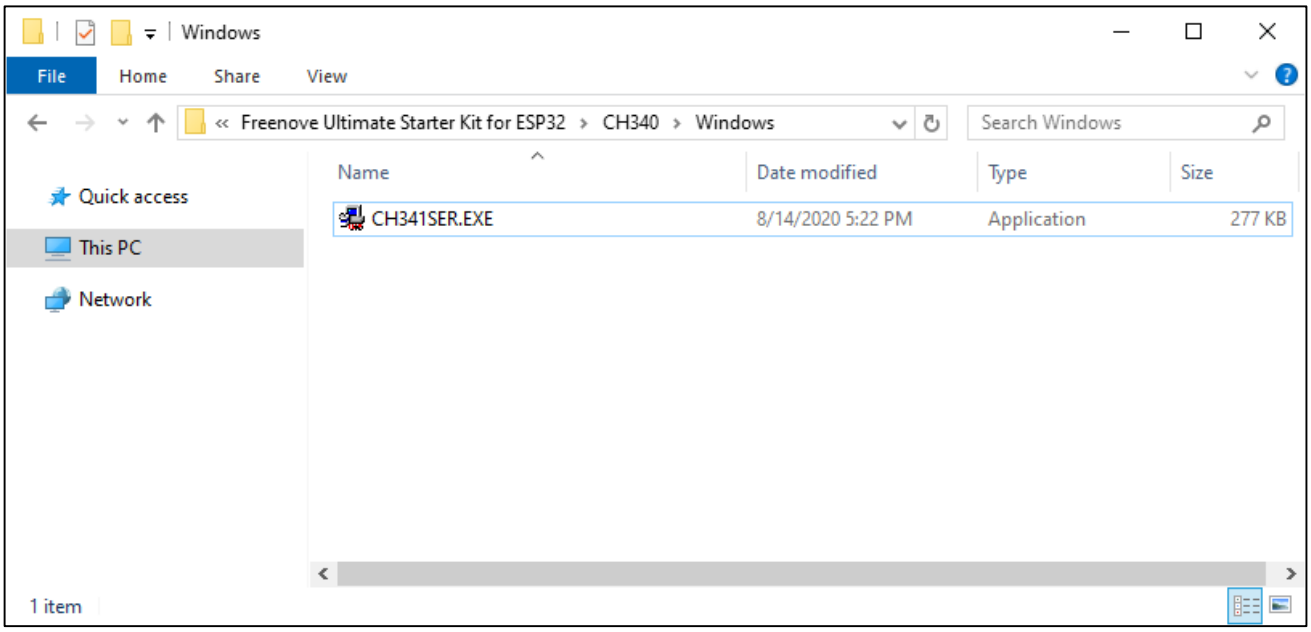
1. First, download CH340 driver, click http://www.wch-ic.com/search?q=CH340&t=downloads to download the appropriate one based on your operating system.



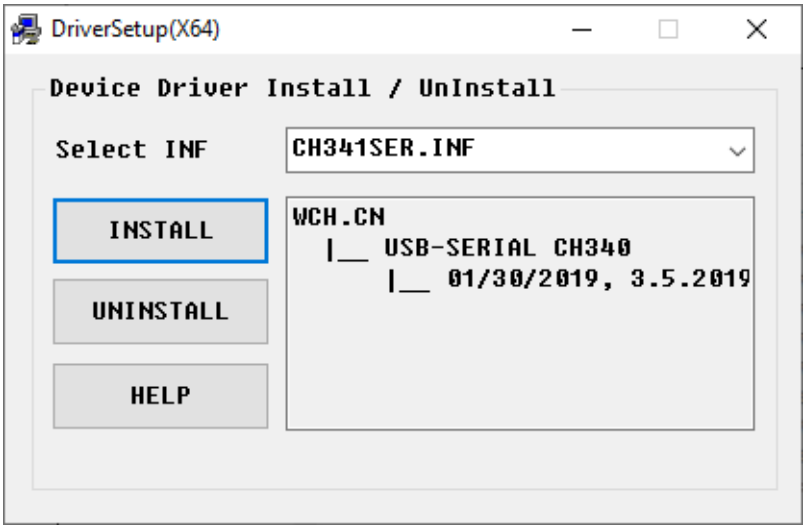
You can also open **"Freenove_Basic_Starter_Kit_for_ESP32/CH340"**, we have prepared the installation package.



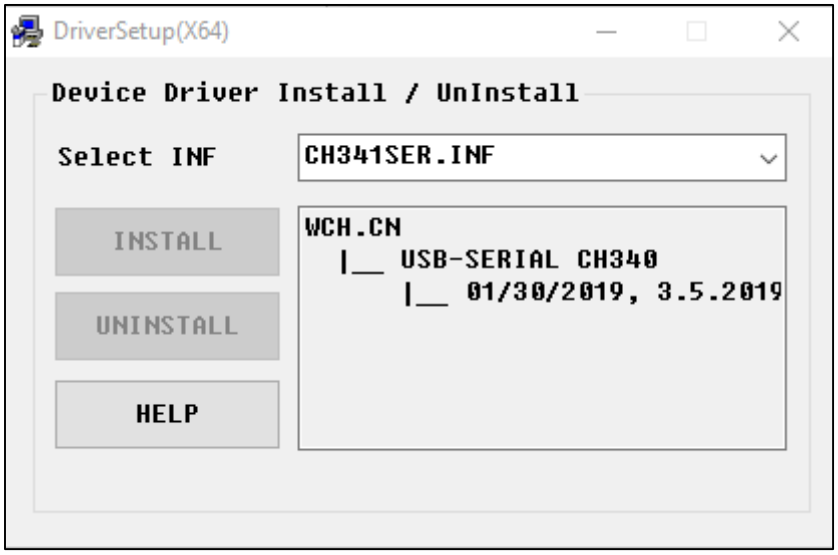
2. Open the folder "**Freenove_Basic_Starter_Kit_for_ESP32/CH340/Windows/ch341ser**"



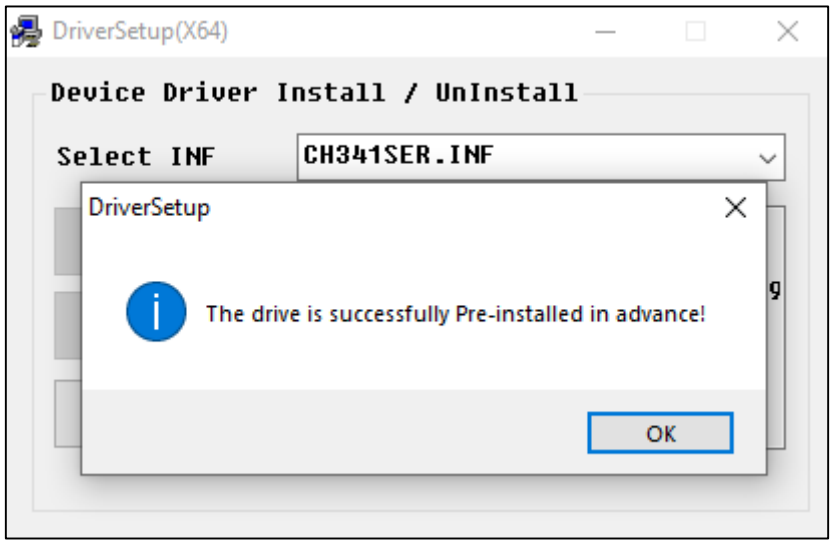
3. Double click "CH341SER.EXE".



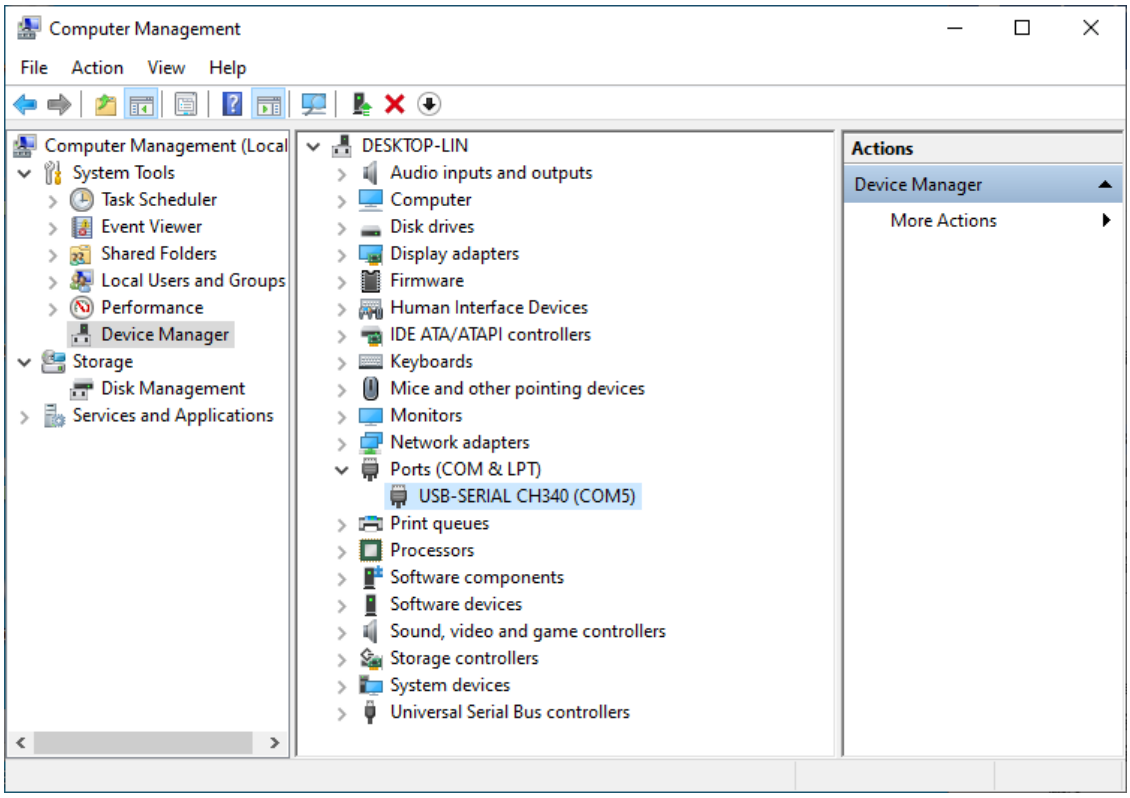
4. Click “INSTALL” and wait for the installation to complete.



5. Install successfully. Close all interfaces.



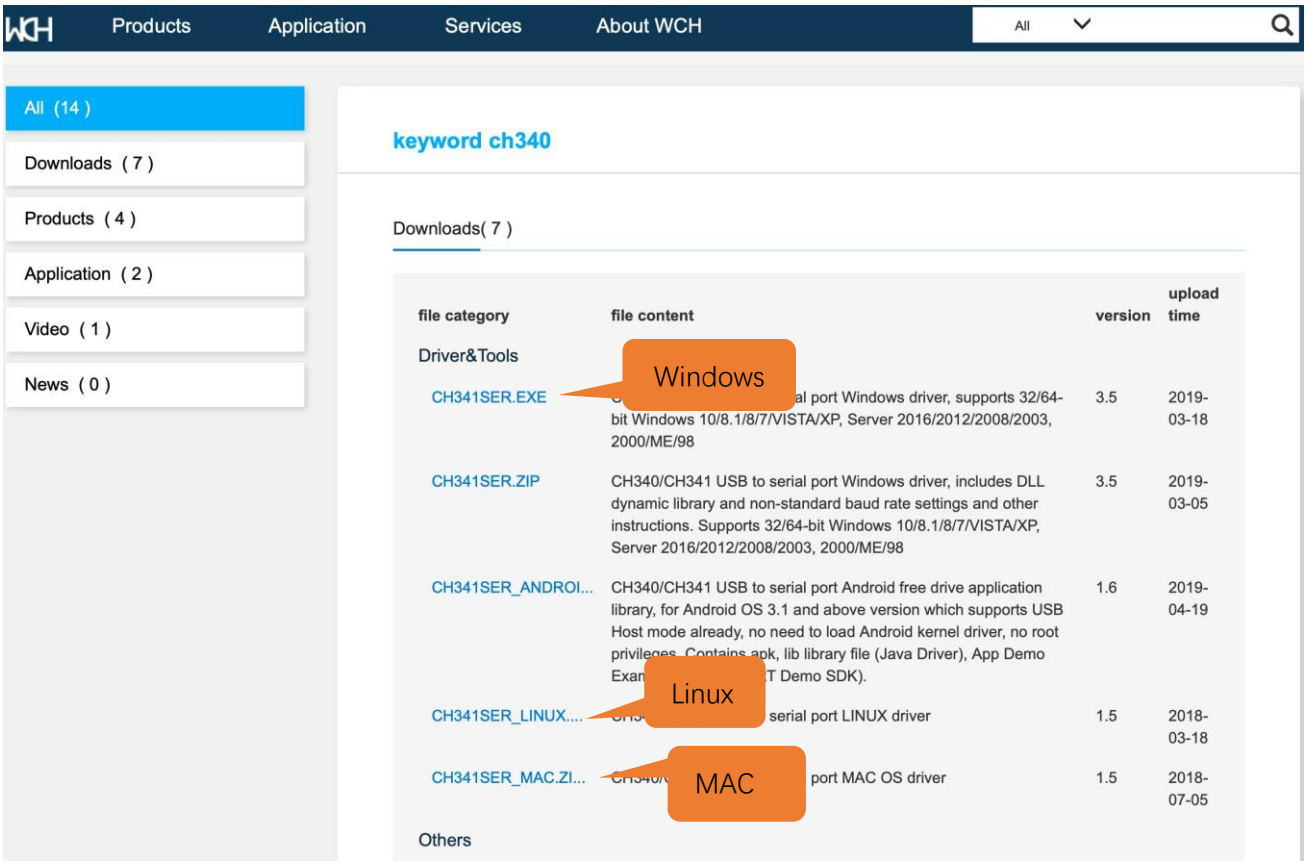
6. When ESP32 is connected to computer, select “This PC”, right-click to select “Manage” and click “Device Manager” in the newly pop-up dialog box, and you can see the following interface.



7. So far, CH340 has been installed successfully. Close all dialog boxes.

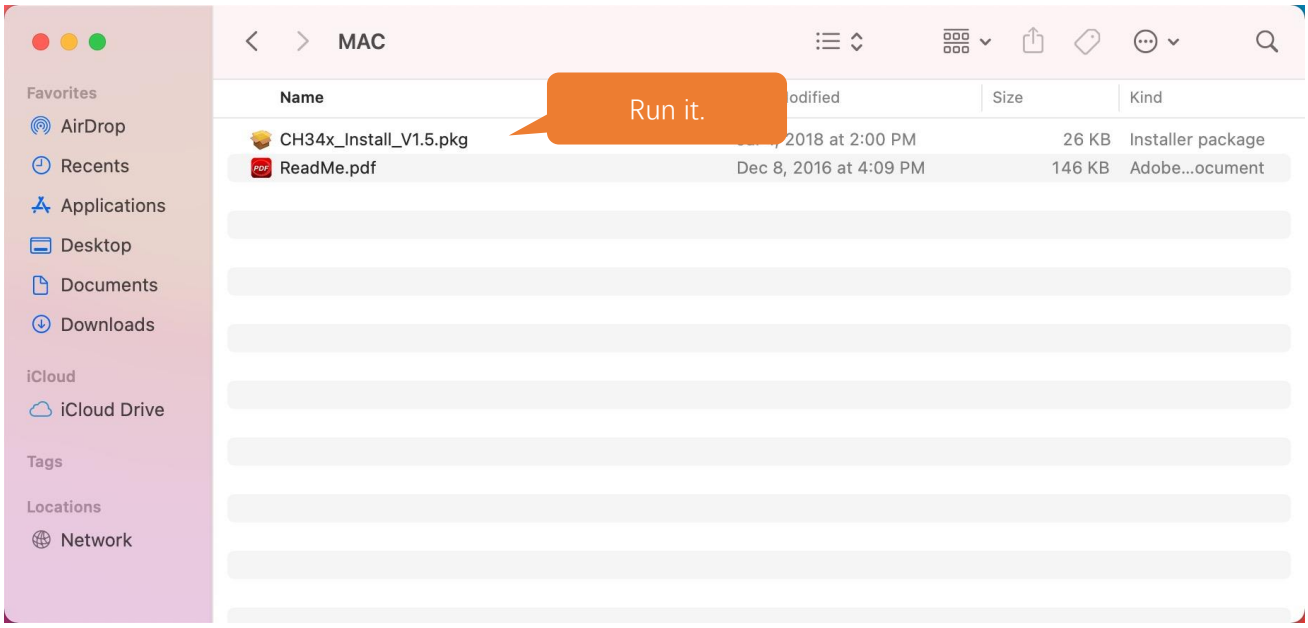
MAC

First, download CH340 driver, click http://www.wch-ic.com/search?q=CH340&t=downloads to download the appropriate one based on your operating system.

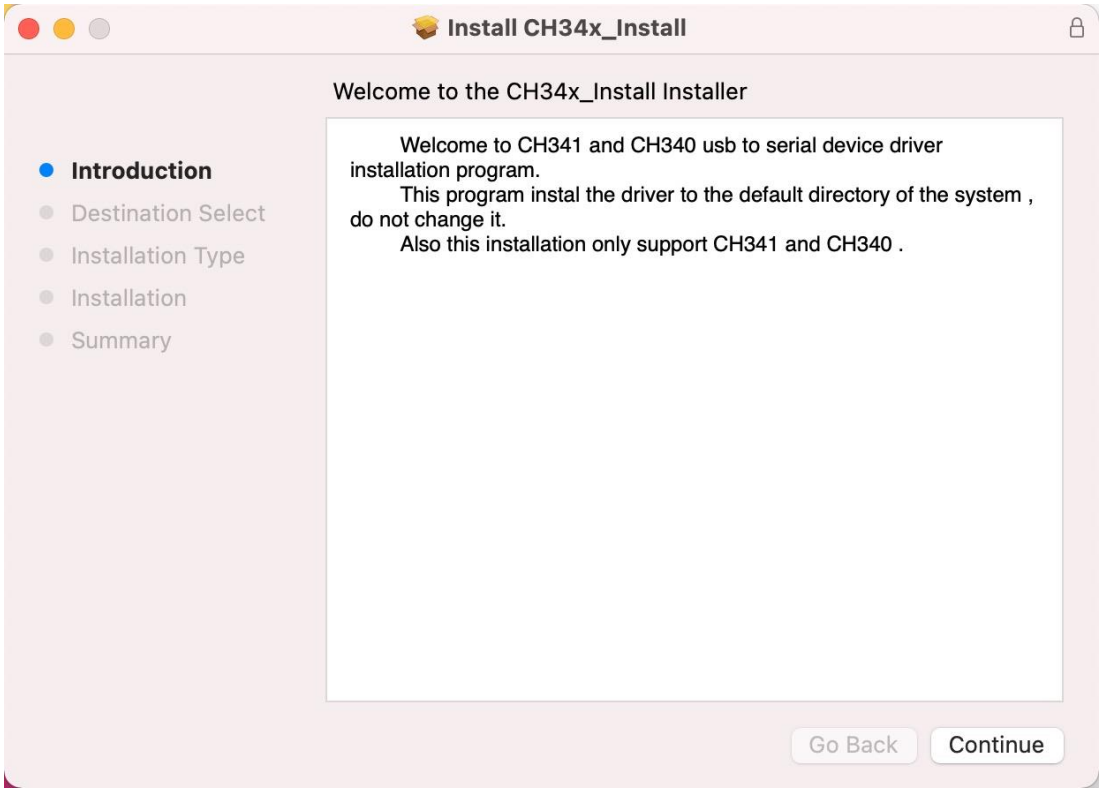


If you would not like to download the installation package, you can open **"Freenove_Basic_Starter_Kit_for_ESP32/CH340"**, we have prepared the installation package.

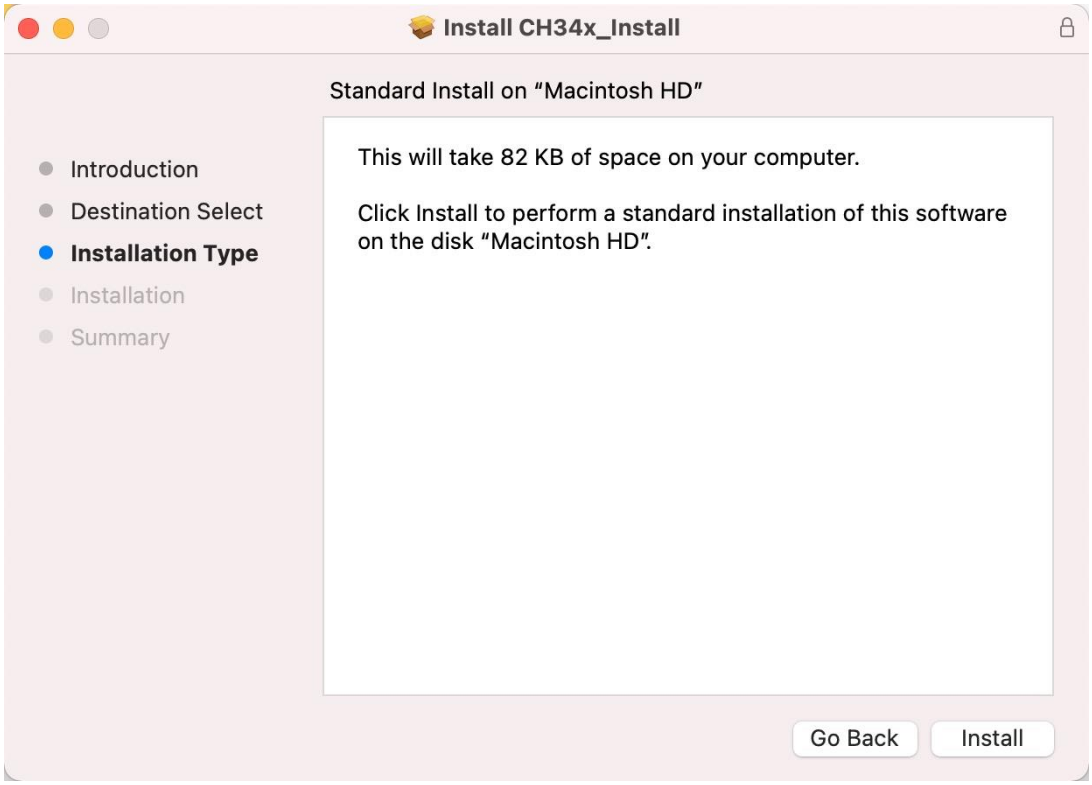
Second, open the folder "**Freenove_Basic_Starter_Kit_for_ESP32/CH340/MAC/**"



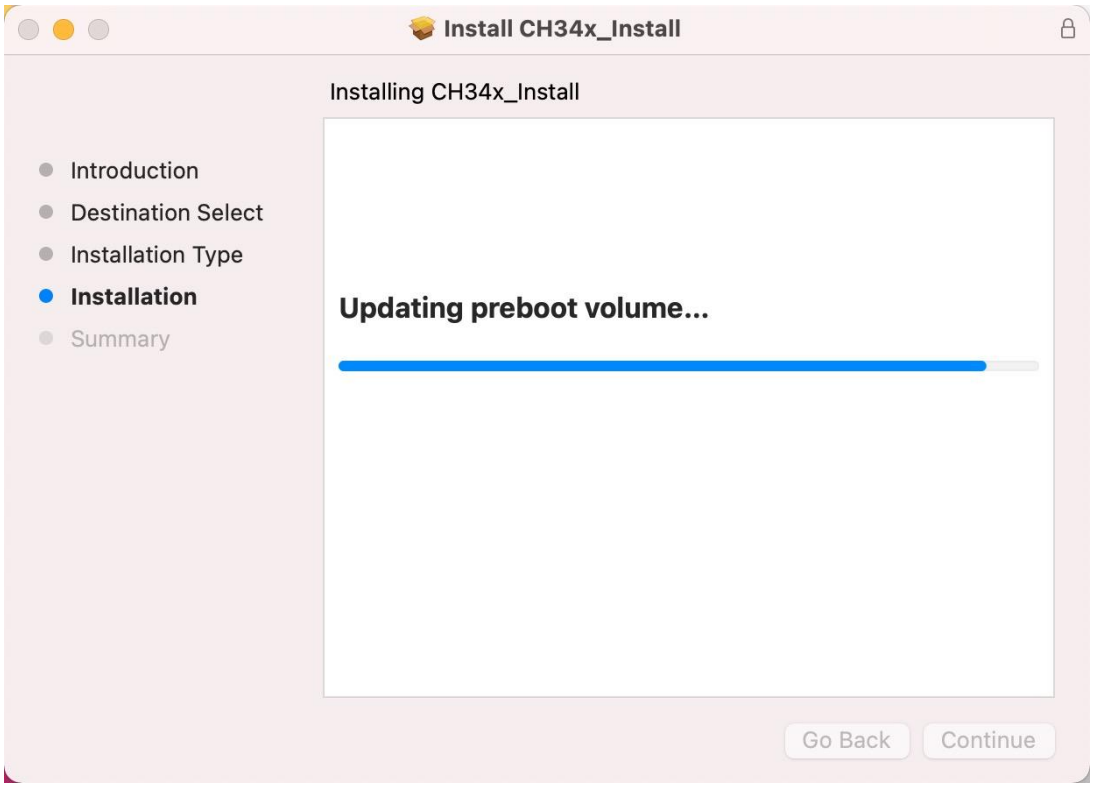
Third, click Continue.



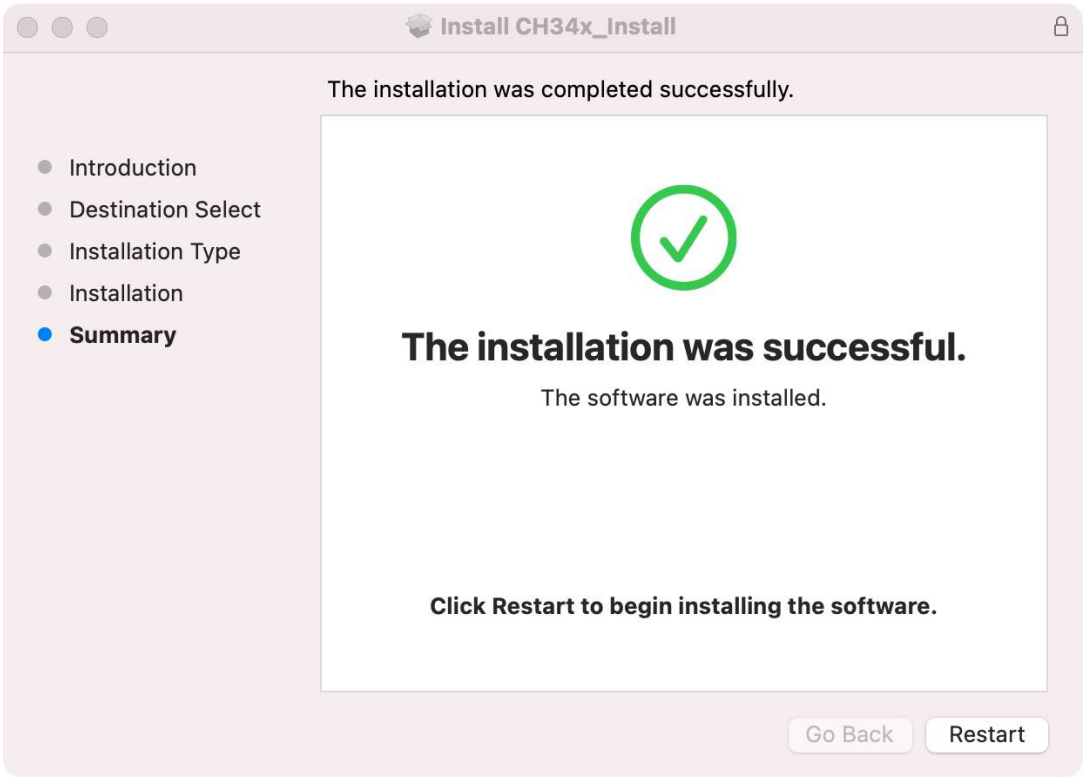
Fourth, click Install.


Install CH34x_Install
Standard Install on "Macintosh HD"
Introduction
Destination Select
Installation Type
Installation
Summary
This will take 82 KB of space on your computer.
Click Install to perform a standard installation of this software on the disk "Macintosh HD".
Go Back
Install

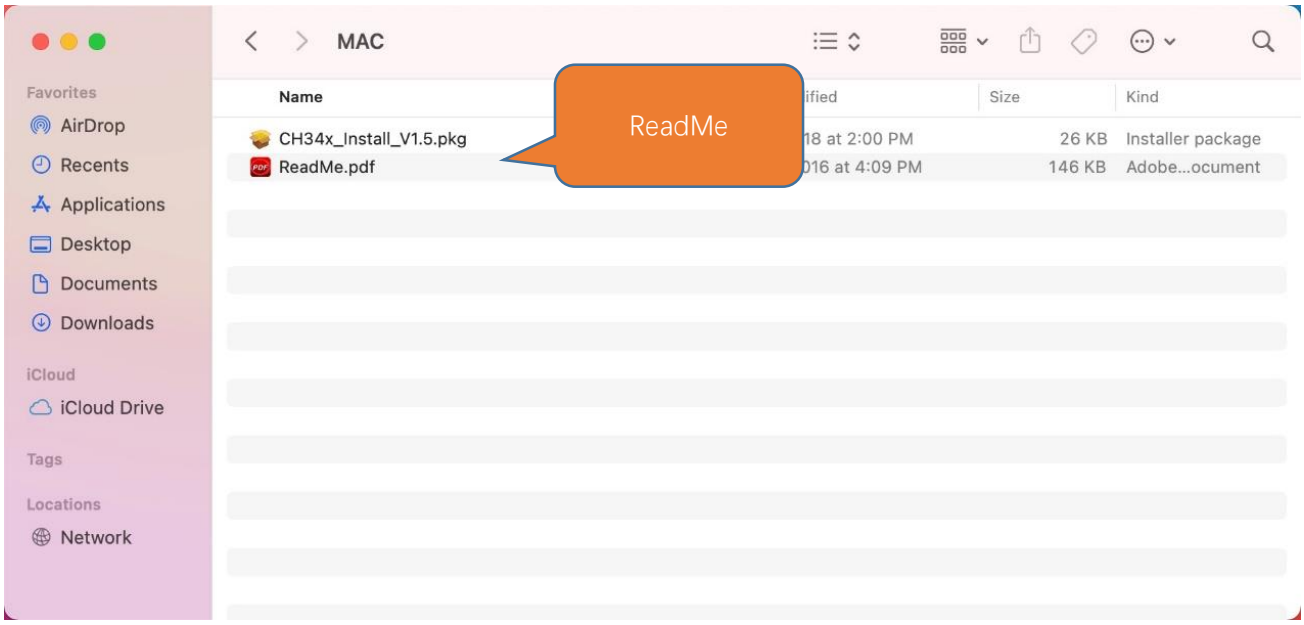
Then, waiting Finsh.



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.

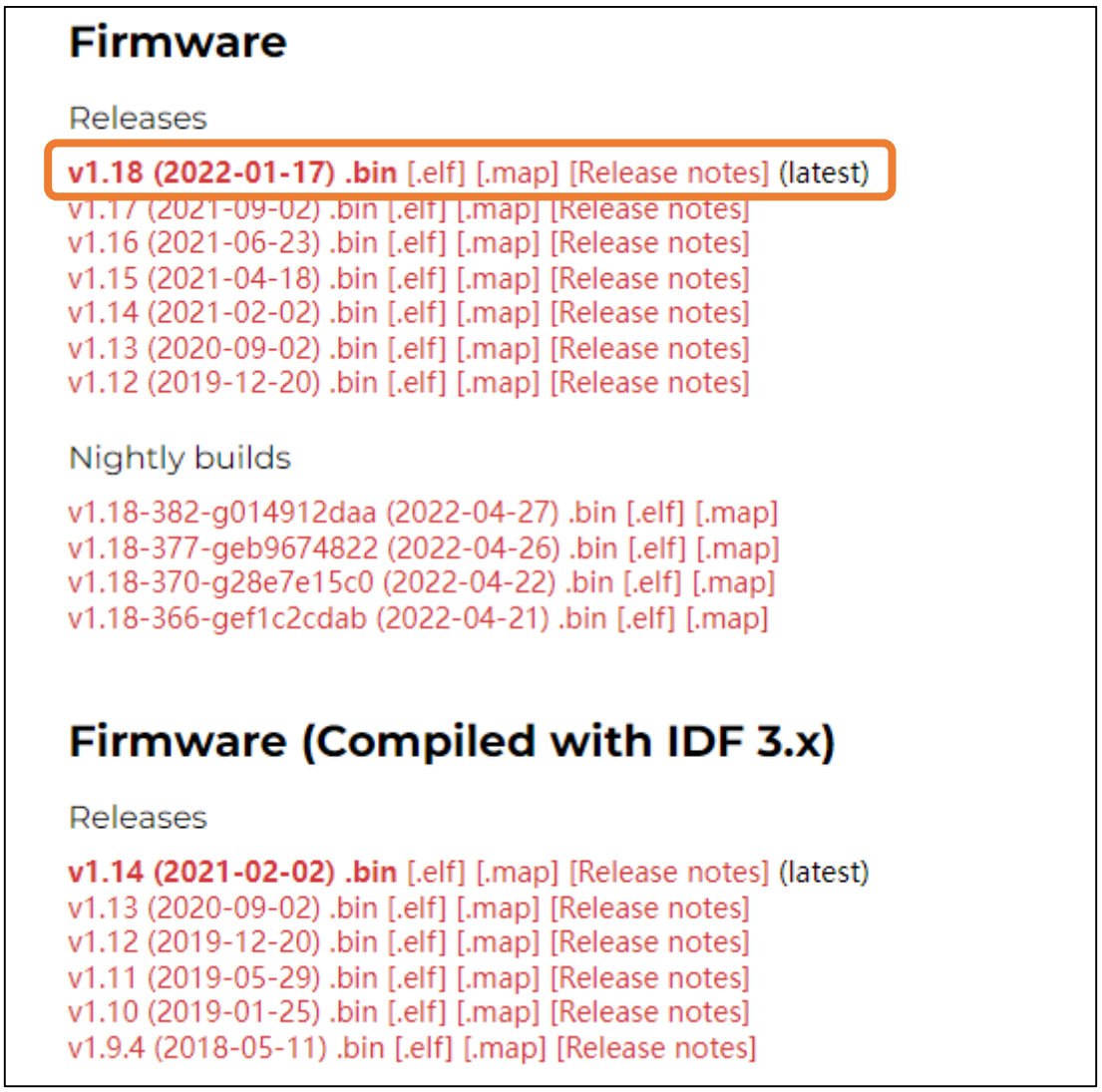


0.4 Burning Micropython Firmware (Important)

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

Downloading Micropython Firmware

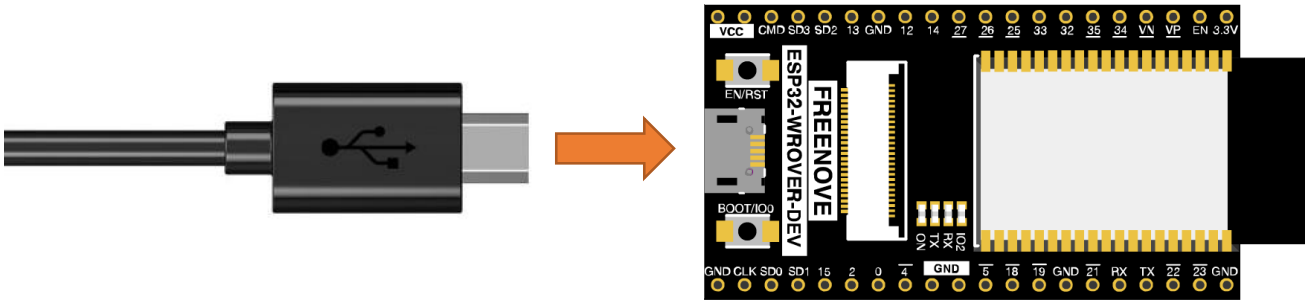
Official website of microPython: http://micropython.org/
Webpage listing firmware of microPython for ESP32: https://micropython.org/download/esp32spiram/



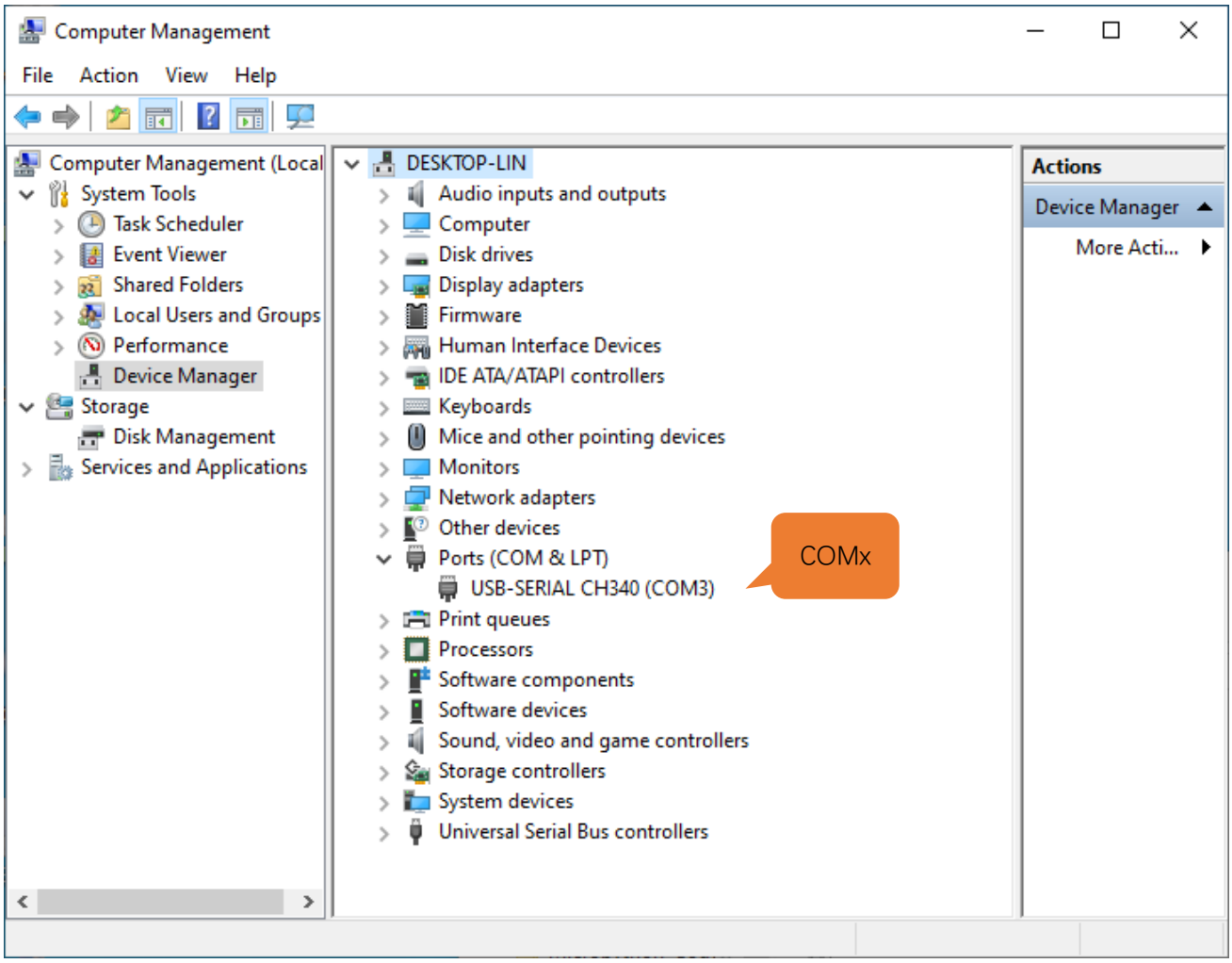
Firmware used in this tutorial is **esp32spiram-20220117-v1.18.bin**
This file is also provided in our data folder "**Freenove_Basic_Starter_Kit_for_ESP32 /Python/Python_Firmware**".。

Burning a Micropython Firmware

Connect your computer and ESP32 with a USB cable.

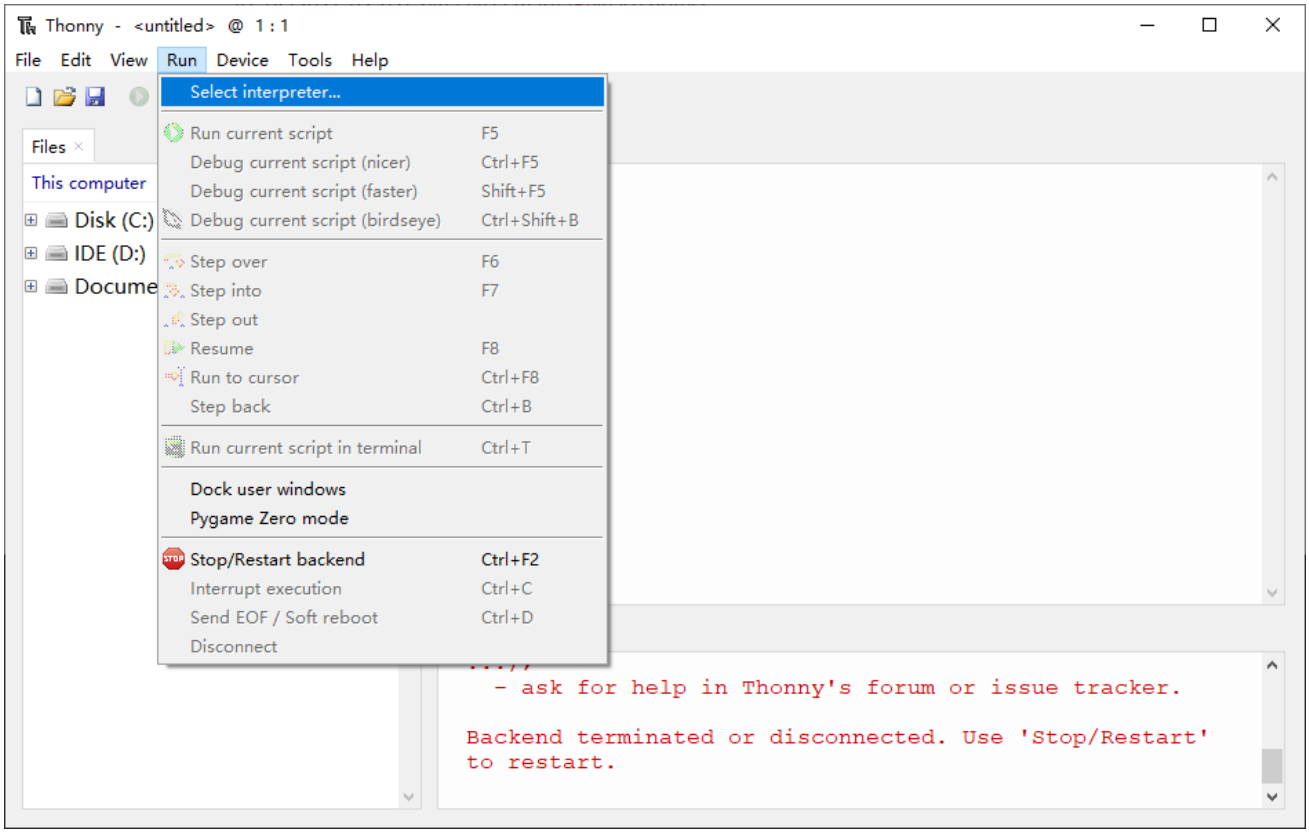


Make sure that the driver has been installed successfully and that it can recognize COM port correctly. Open device manager and expand "Ports".

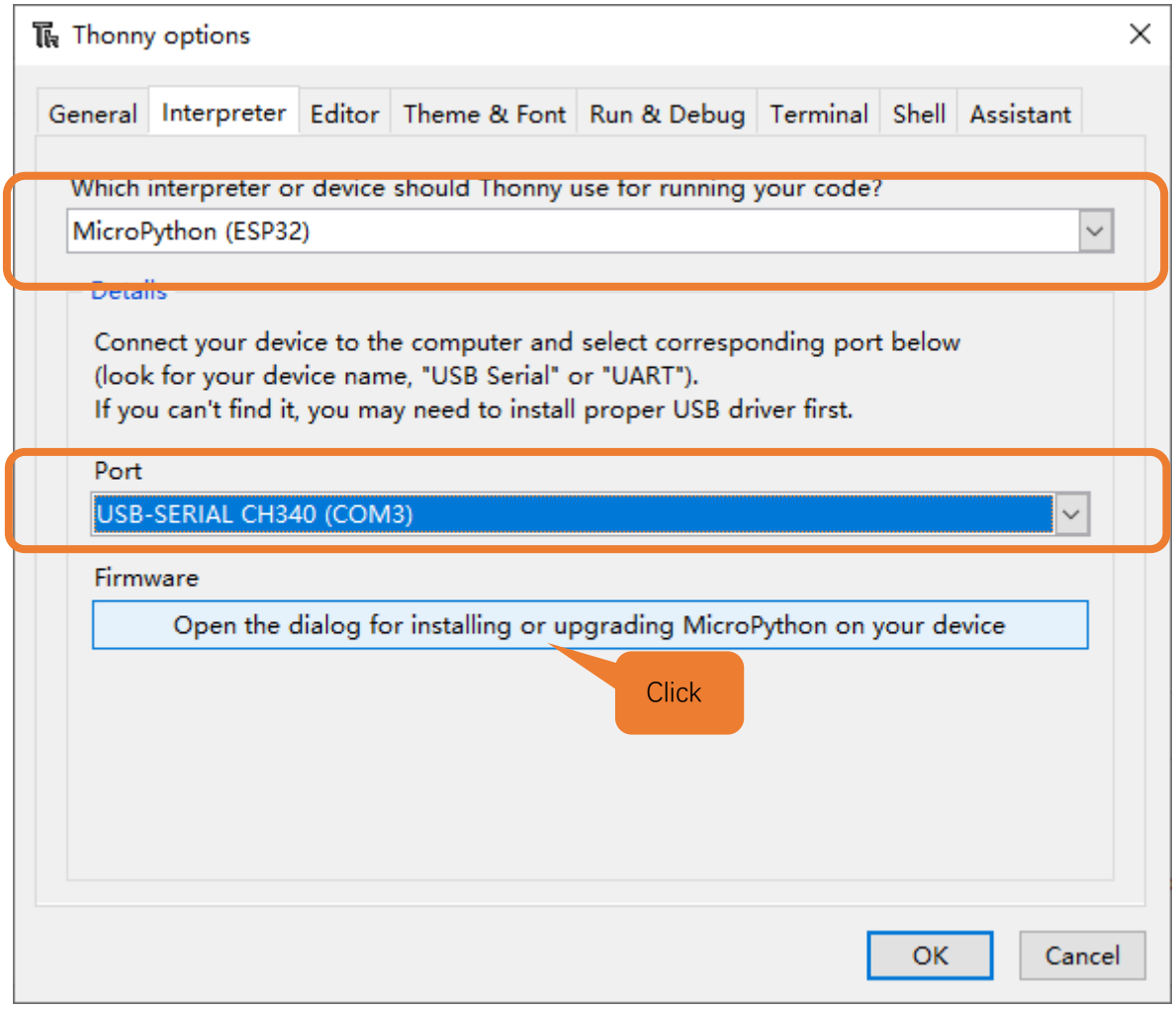


Note: the port of different people may be different, which is a normal situation.

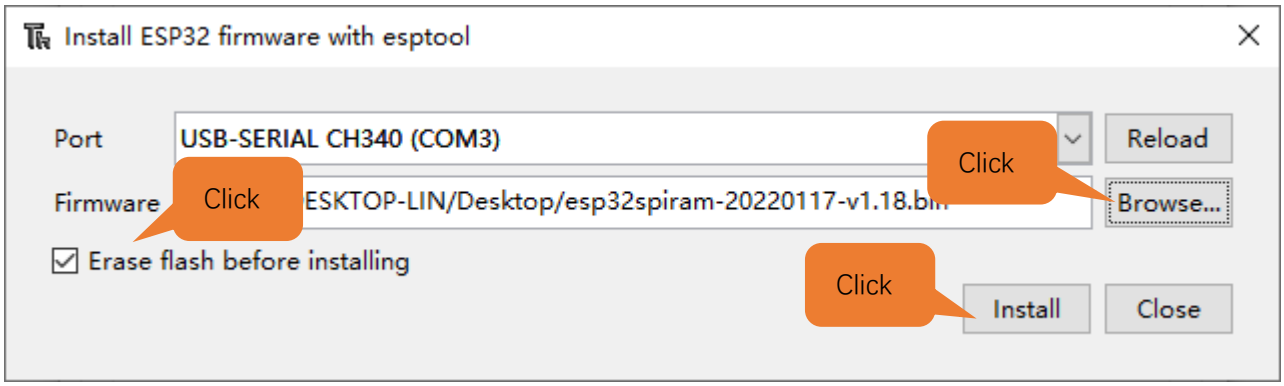
1. Open Thonny, click “run” and select “Select interpreter...””



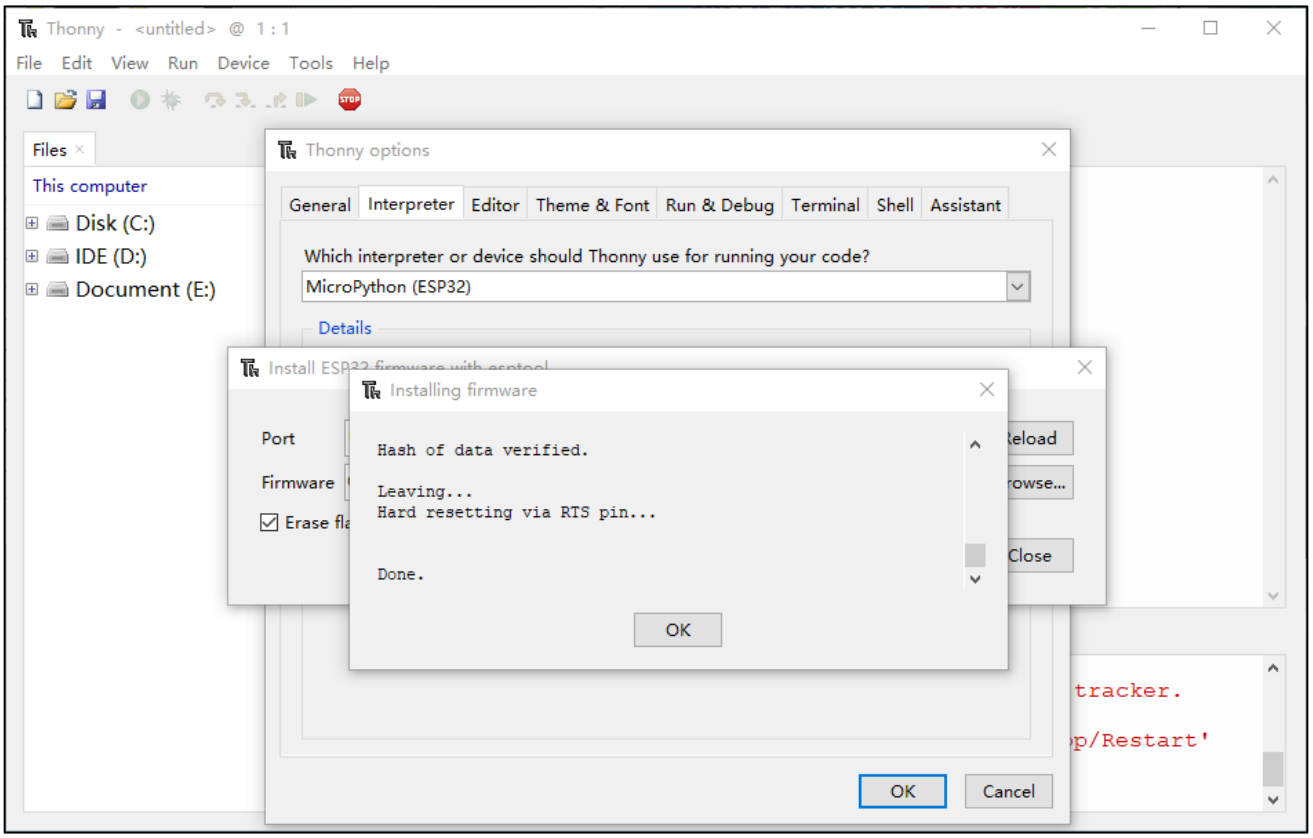
2. Select "Micropython (ESP32)", select "USB-SERIAL CH340 (COM4)", and then click the long button under "Firmware".



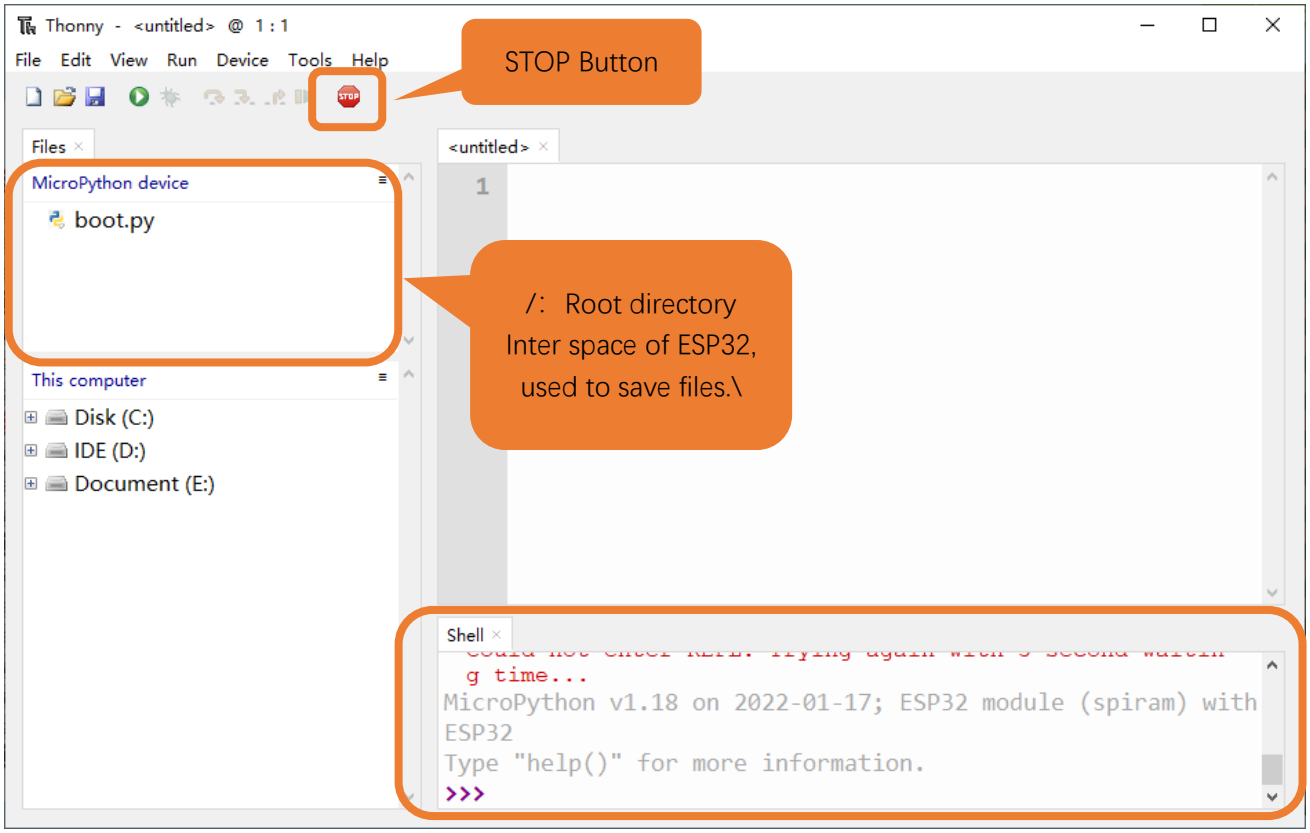
3. The following dialog box pops up. Select "USB-SERIAL CH340 (COM3)" for "Port" and then click "Browse...". Select the previous prepared microPython firmware "**esp32spiram-20220117-v1.18.bin**". Check "Erase flash before installing" and click "install" to wait for the prompt of finishing installation.



4. Wait for the installation to be done.



5. Close all dialog boxes, turn to main interface and click “STOP”. As shown in the illustration below

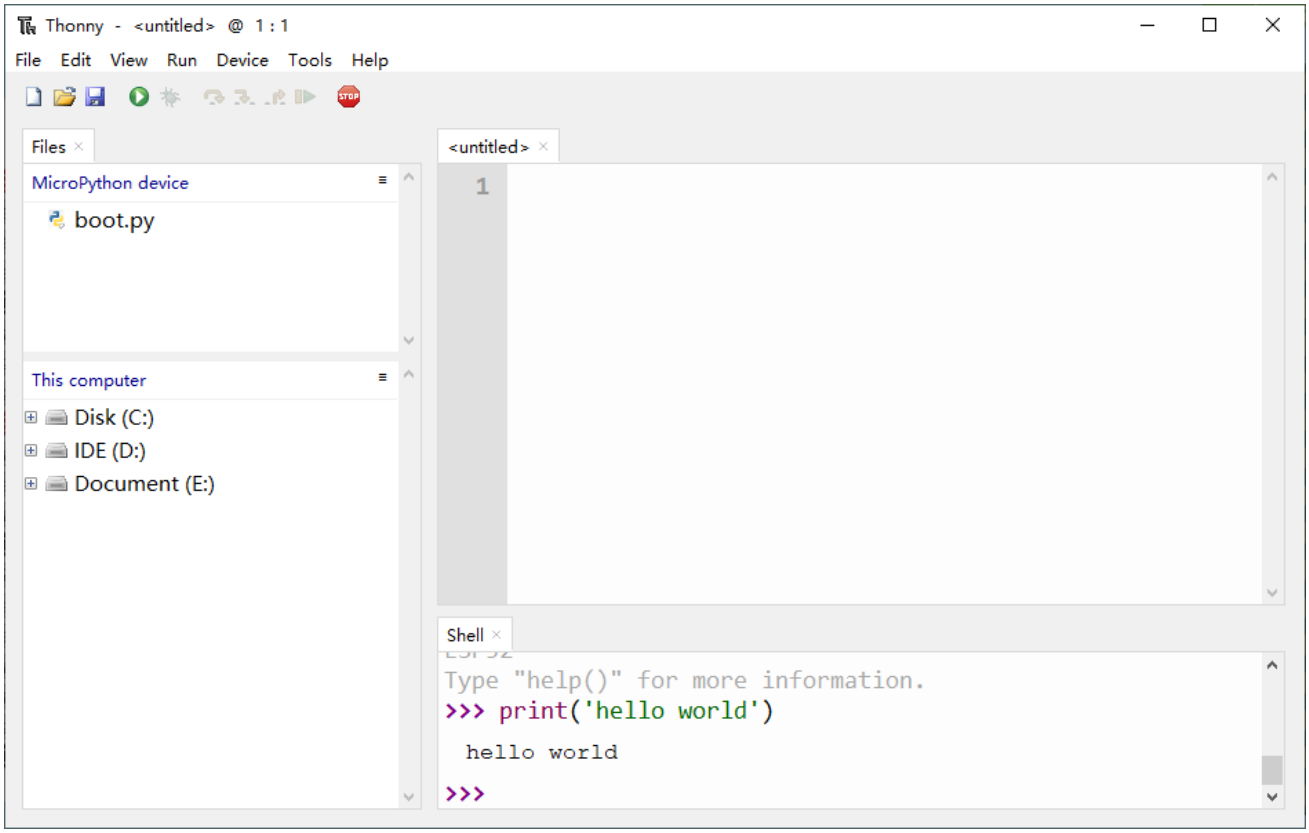


6. So far, all the preparations have been made.

0.5 Testing codes (Important)

Testing Shell Command

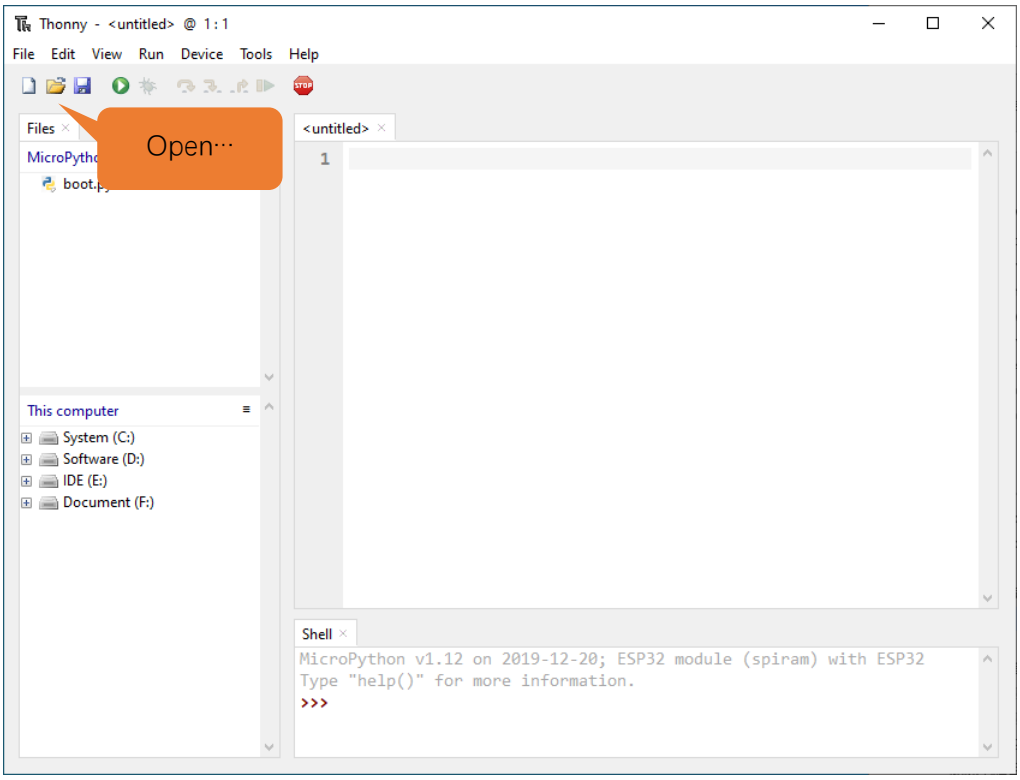
Enter "print('hello world')" in "Shell" and press Enter.



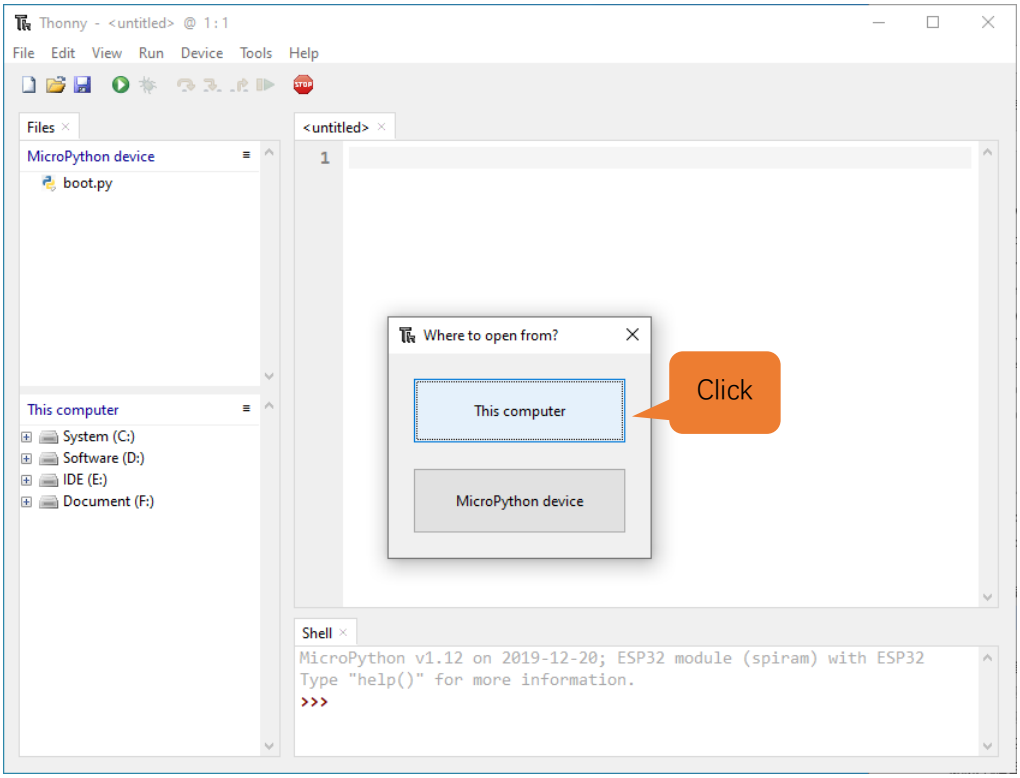
Running Online

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

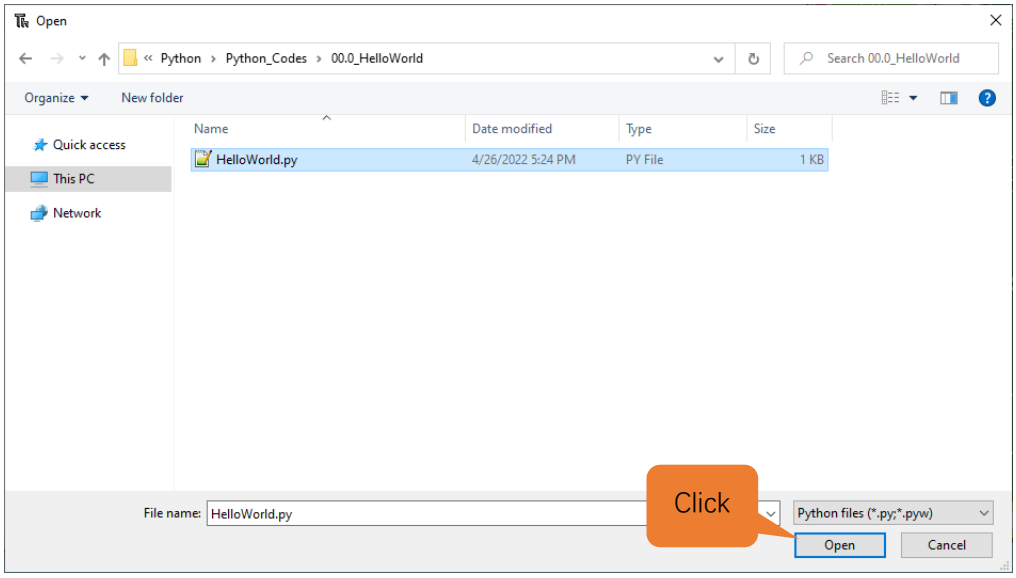
1. Open Thonny and click “Open···”.



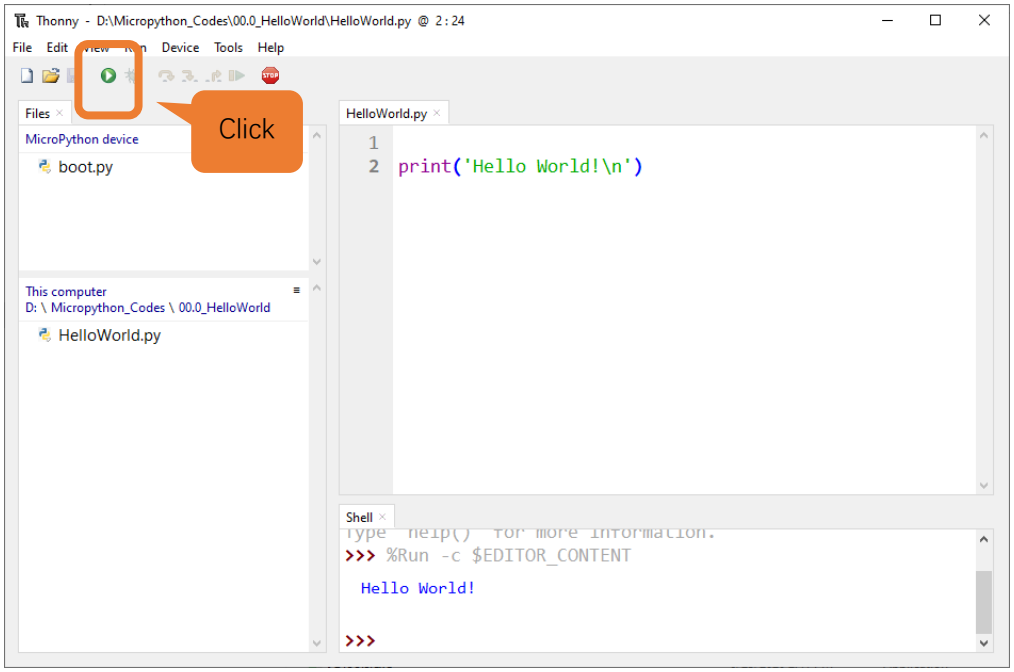
2. On the newly pop-up window, click “This computer”.



In the new dialog box, select "**HelloWorld.py**" in "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes/00.0_HelloWorld**" folder.



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

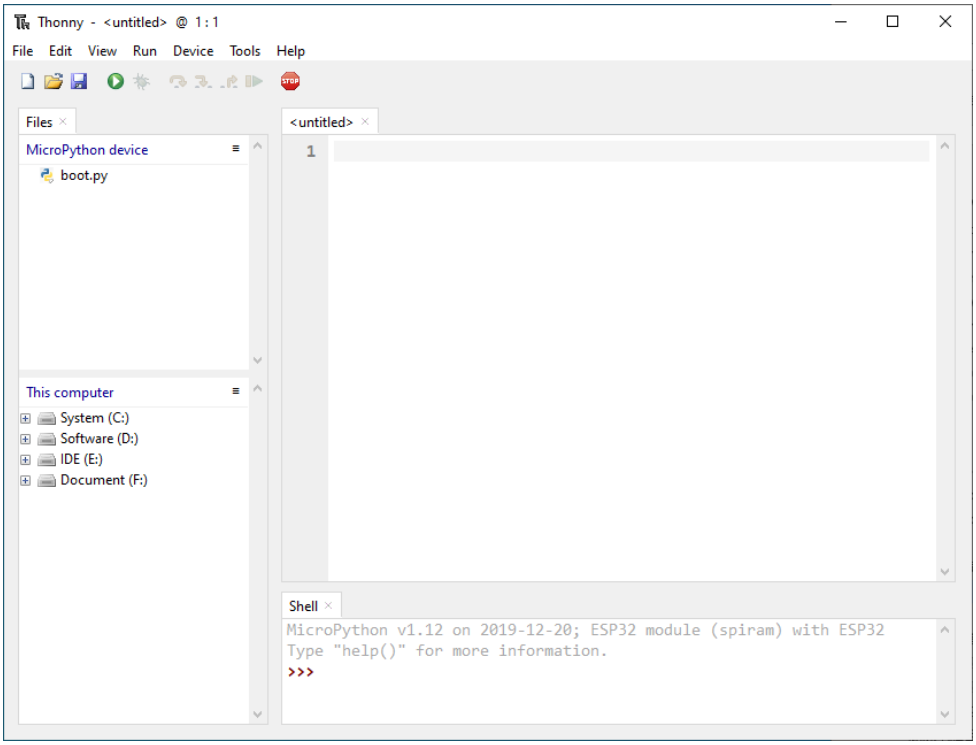


Note: When running online, if you press the reset key of ESP32, user's code will not be executed again. If you wish to run the code automatically after resetting the code, please refer to the following Running Offline.

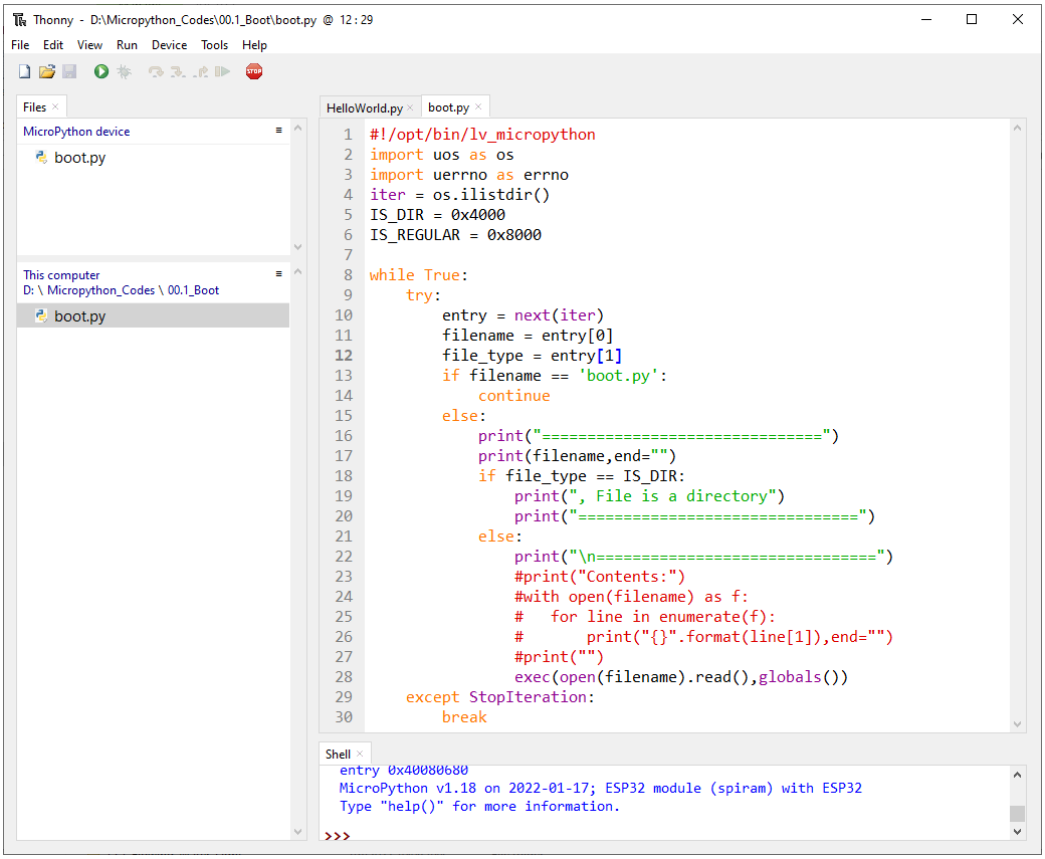
Running Offline （Importance）

After ESP32 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 execute user’s programs after resetting, we need to add a guiding program in boot.py to execute user’s code.

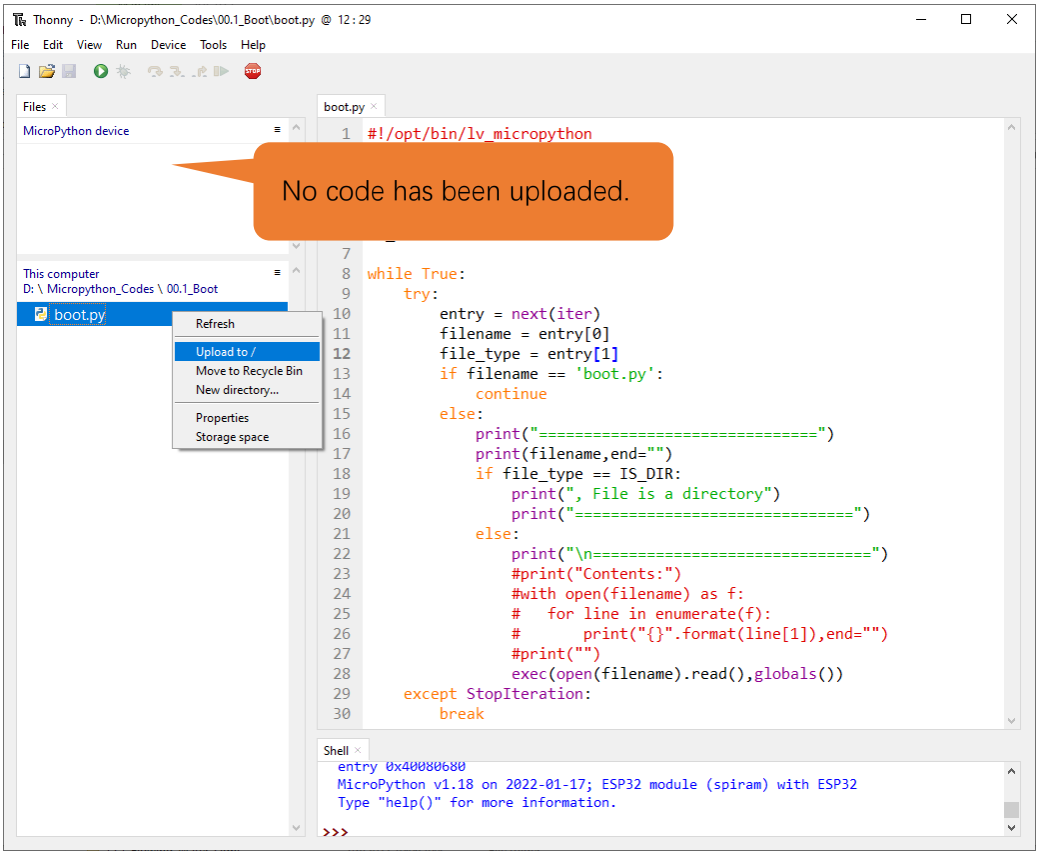
1. Move the program folder “**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**” to disk(D) in advance with the path of “**D:/Micropython_Codes**”. Open “Thonny”。



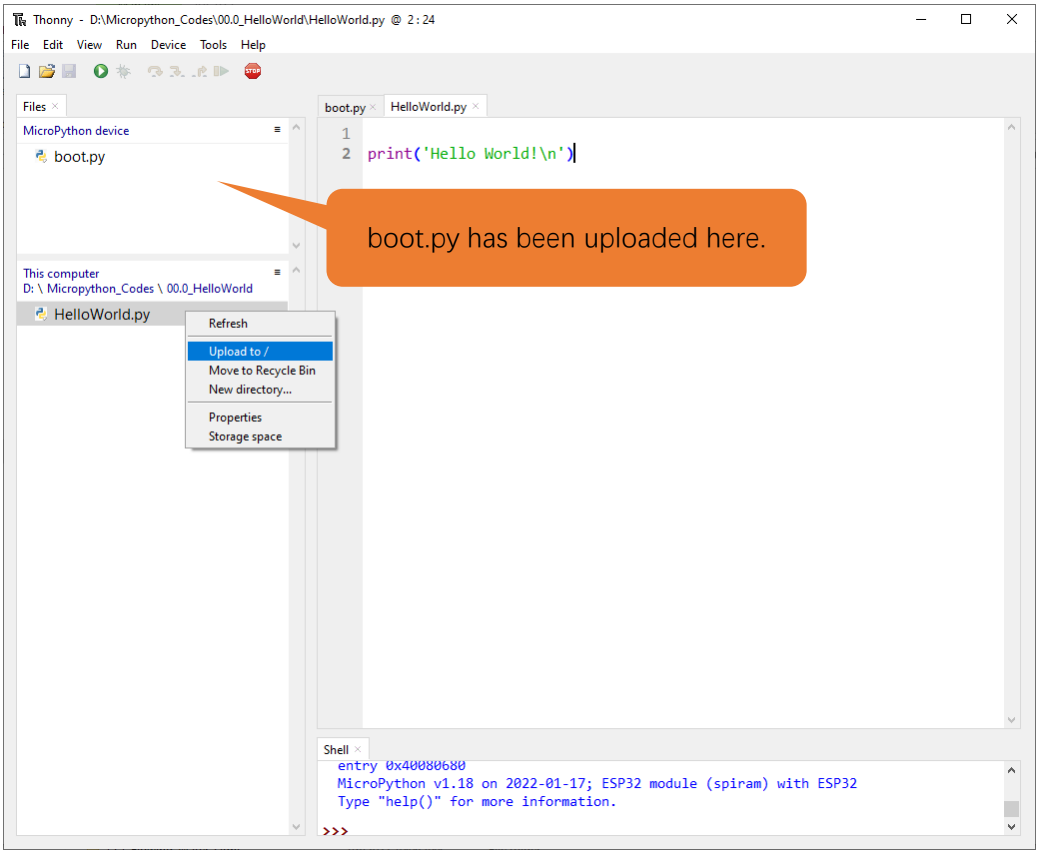
2. Expand “00.1_Boot” in the “Micropython_Codes” in the directory of disk(D), and double-click boot.py, which is provided by us to enable programs in “MicroPython device” to run offline.



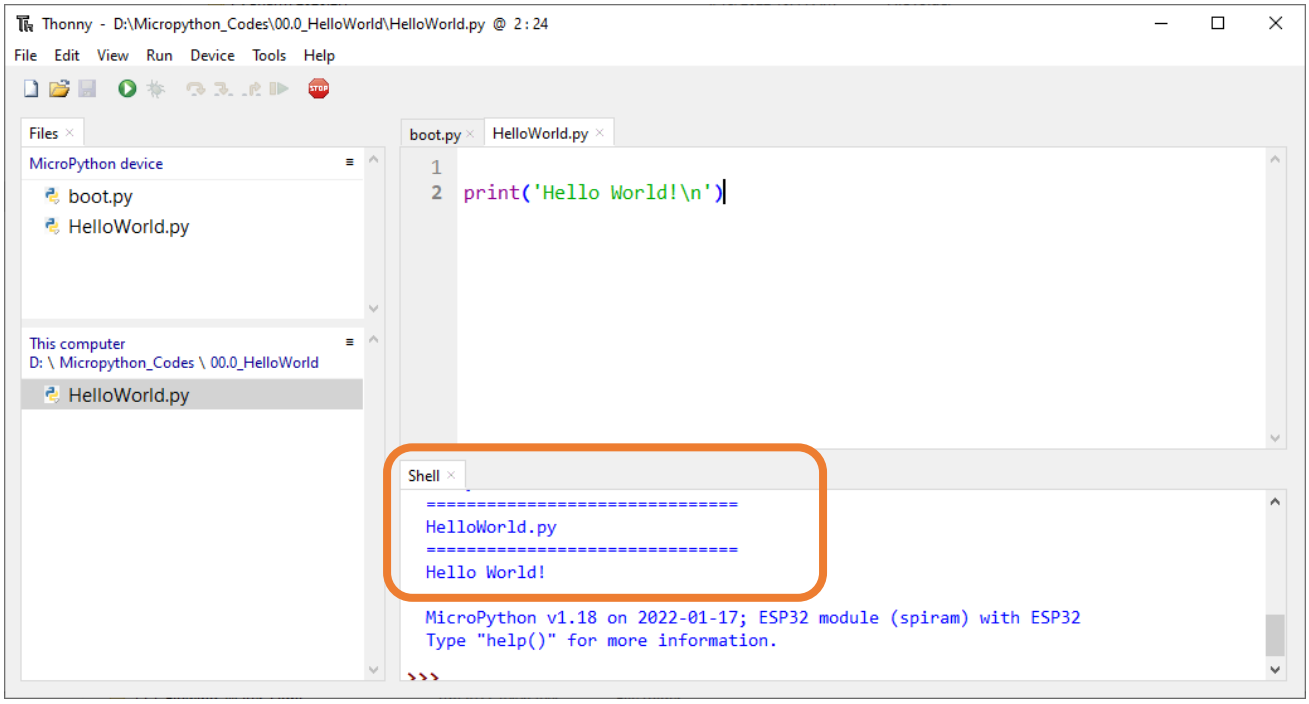
If you want your written programs to run offline, you need to upload boot.py we provided and all your codes to "MicroPython device" and press ESP32's reset key. Here we use programs 00.0 and 00.1 as examples. Select "boot.py", right-click to select "Upload to /".



Similarly, upload “HelloWorld.py” to “MicroPython device”.



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

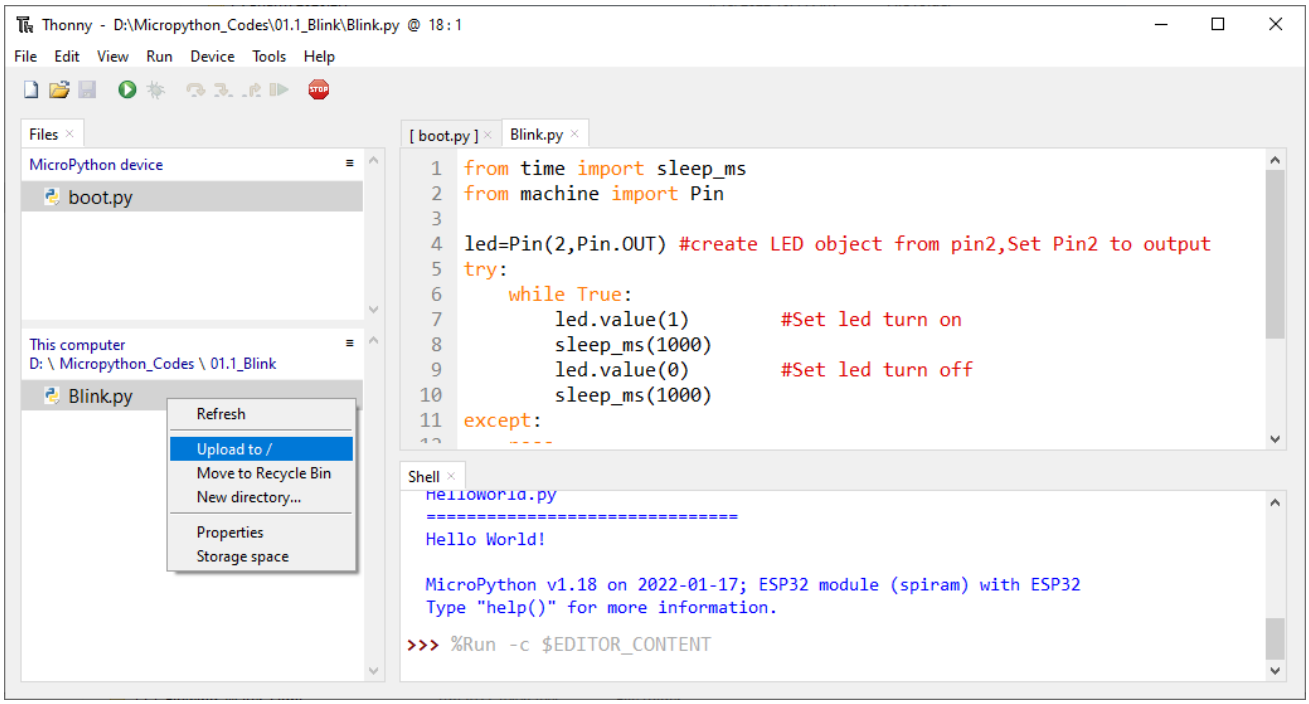


0.6 Thonny Common Operation

Uploading Code to ESP32

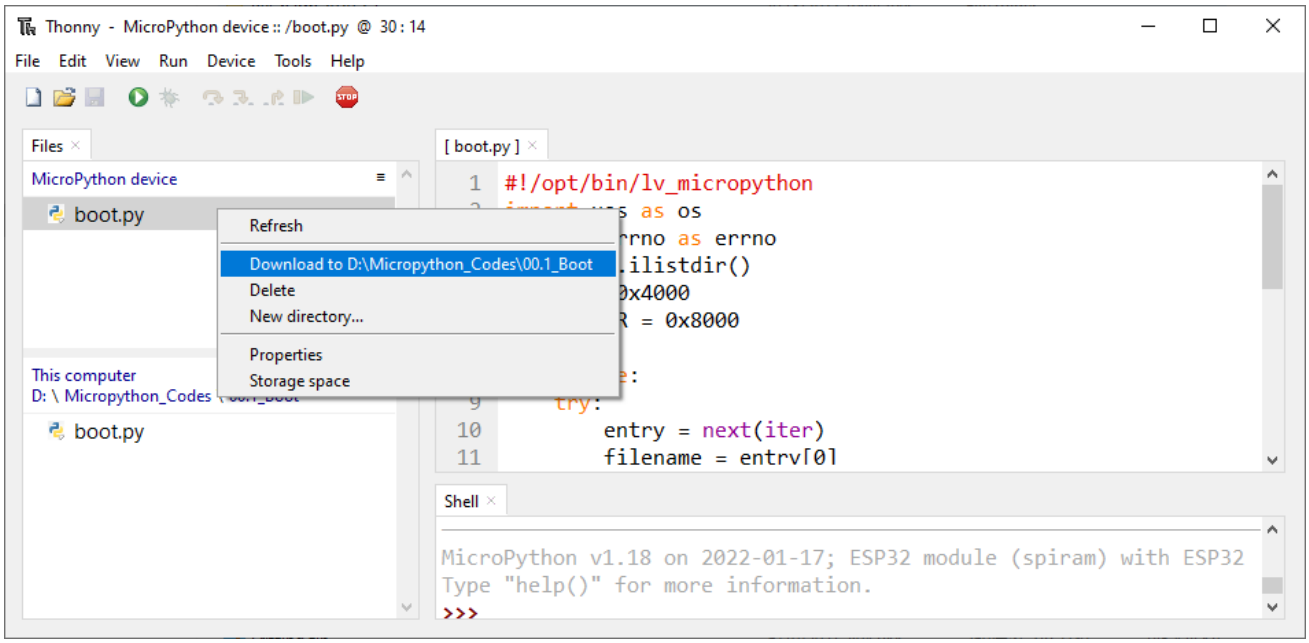
Each time when ESP32 restarts, if there is a "boot.py" in the root directory, it will execute this code first.

Select "Blink.py" in "01.1_Blink", right-click your mouse and select "Upload to /" to upload code to ESP32's root directory.



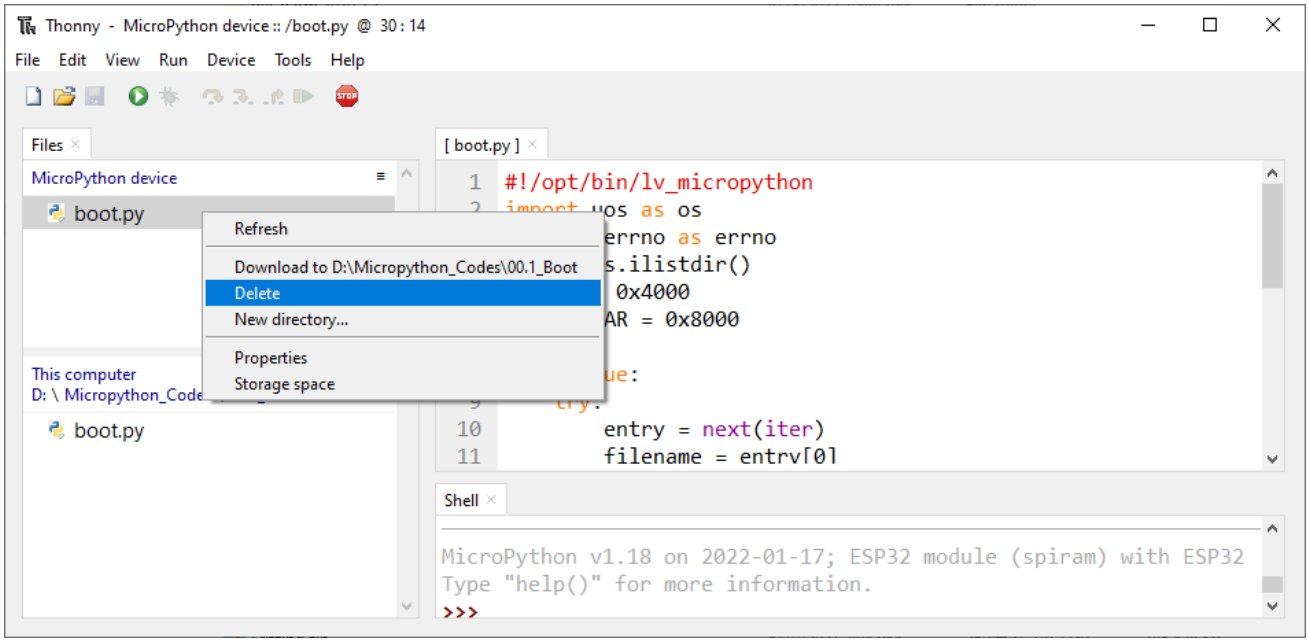
Downloading Code to Computer

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



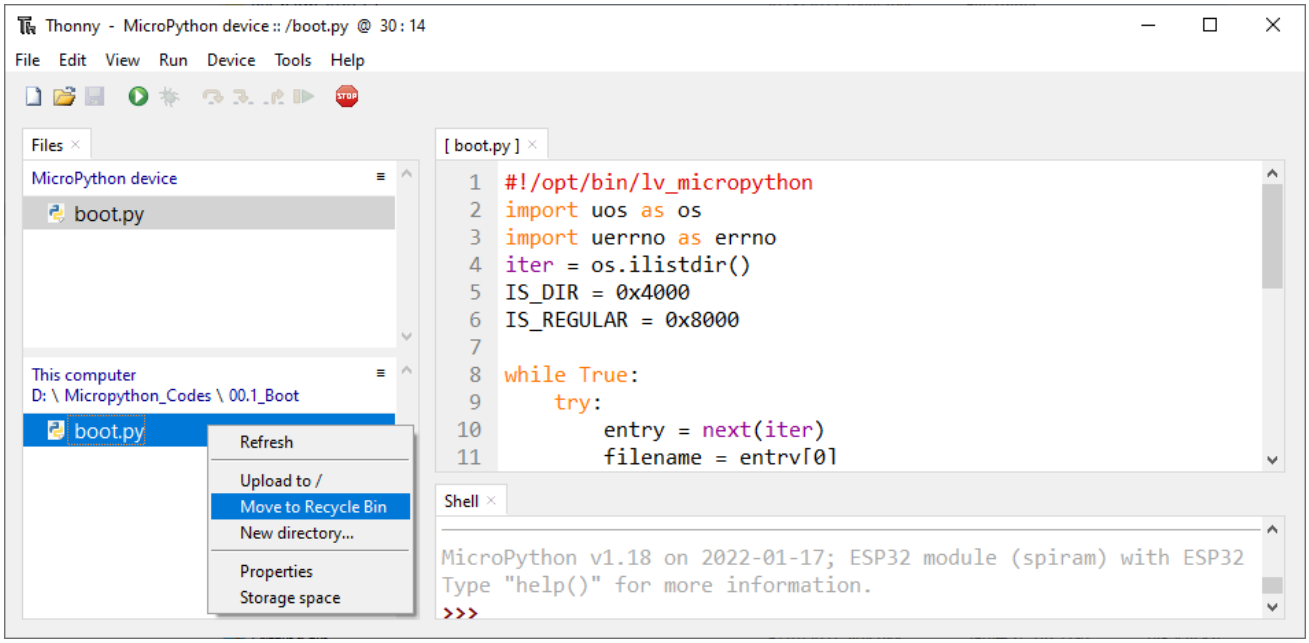
Deleting Files from ESP32’s Root Directory

Select “boot.py” in “MicroPython device”, right-click it and select “Delete” to delete “boot.py” from ESP32’s root directory.



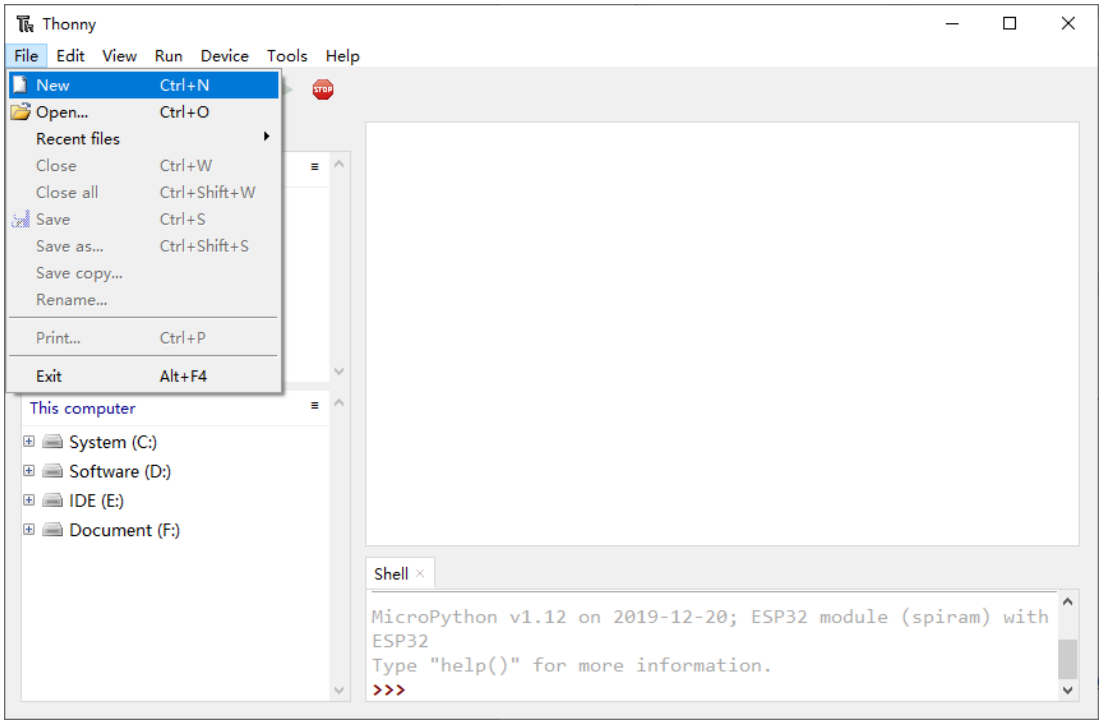
Deleting Files from your Computer Directory

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

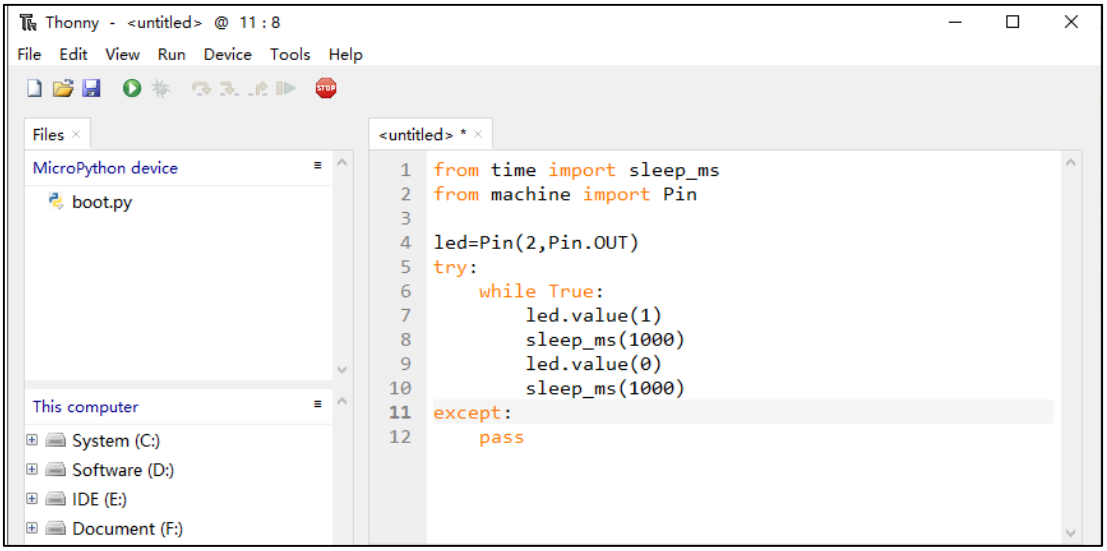


Creating and Saving the code

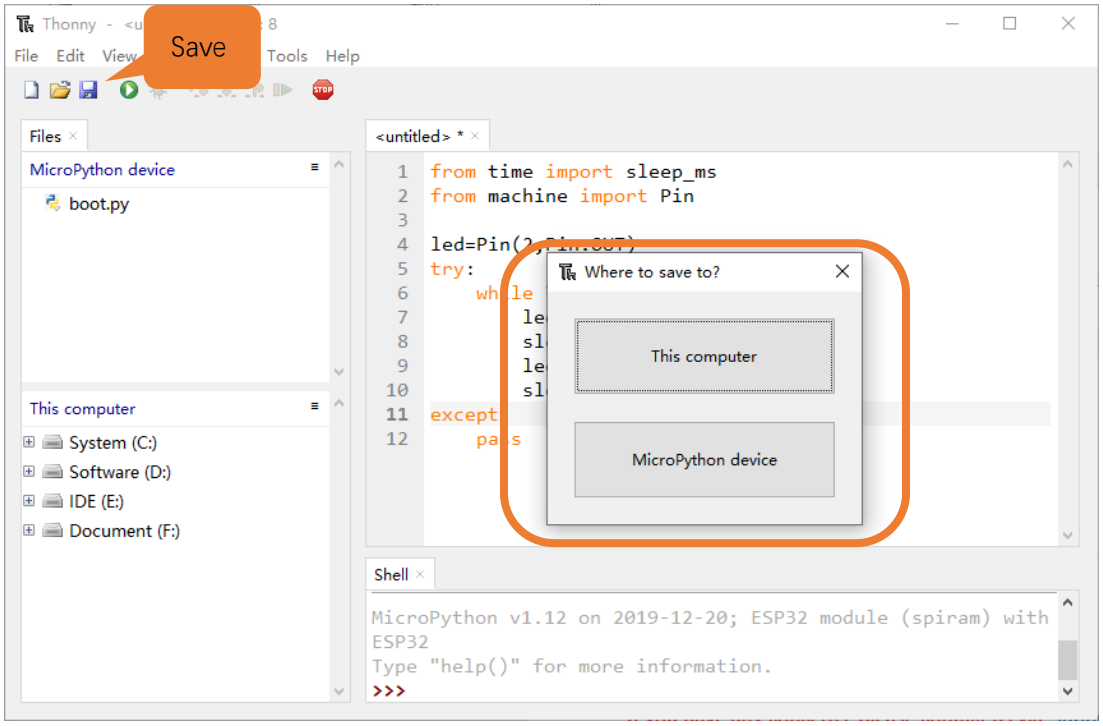
Click "File"→"New" to create and write codes.



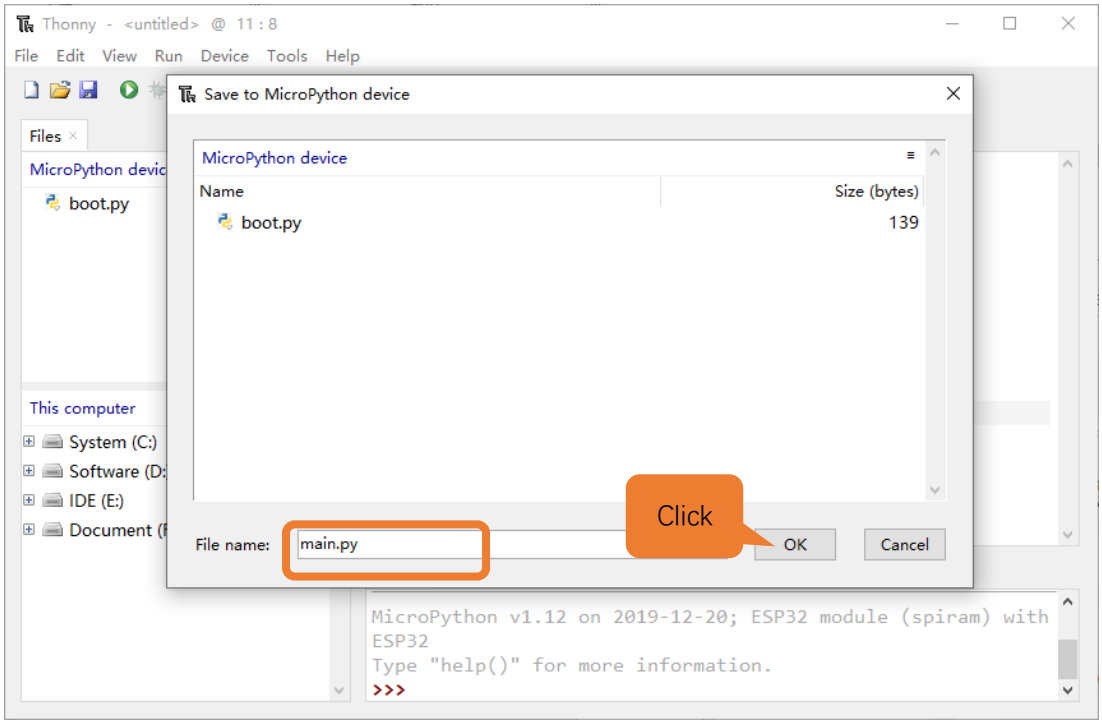
Enter codes in the newly opened file. Here we use codes of "01.1_Blink.py" as an example.



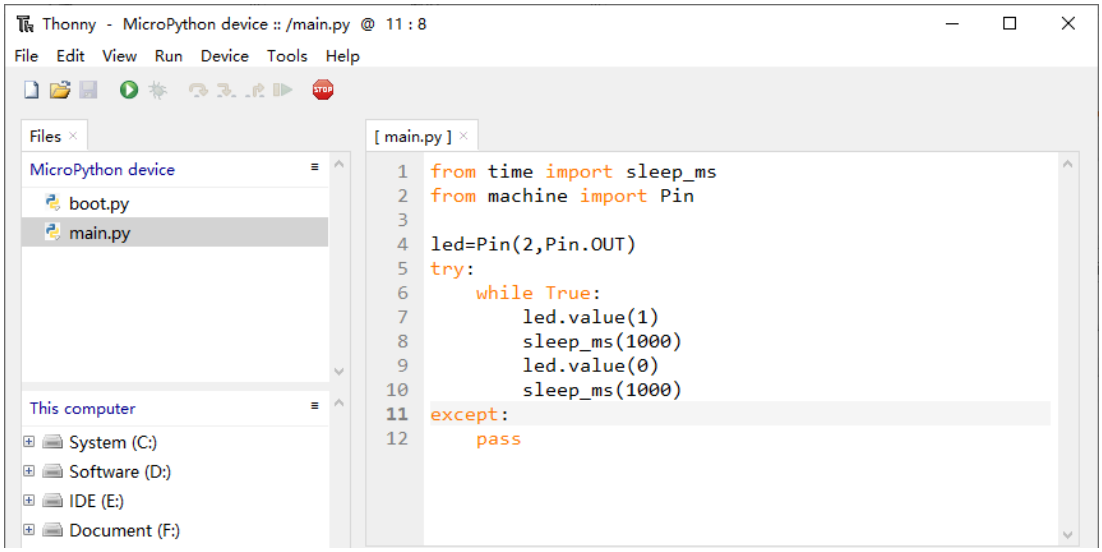
Click "Save" on the menu bar. You can save the codes either to your computer or to ESP32-WROVER.



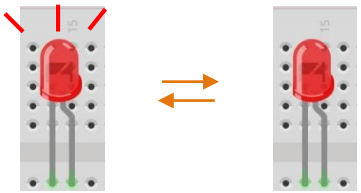
Select "MicroPython device", enter "main.py" in the newly pop-up window and click "OK".



You can see that codes have been uploaded to ESP32-WROVER.



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.



0.7 Note

Though there are many pins available on ESP32, some of them have been connected to peripheral equipment, so we should avoid using such pins to prevent pin conflicts. For example, when downloading programs, make sure that the pin state of Strapping Pin, when resetting, is consistent with the default level; do NOT use Flash Pin; Do NOT use Cam Pin when using Camera function.

Strapping Pin

The state of Strapping Pin can affect the functions of ESP32 after it is reset, as shown in the table below.

Voltage of Internal LDO (VDD_SDIO)			
Pin	Default	3.3 V	1.8 V
MTDI	Pull-down	0	1
Booting Mode			
Pin	Default	SPI Boot	Download Boot
GPIO0	Pull-up	1	0
GPIO2	Pull-down	Don't-care	0
Enabling/Disabling Debugging Log Print over U0TXD During Booting			
Pin	Default	U0TXD Active	U0TXD Silent
MTDO	Pull-up	1	0

Timing of SDIO Slave					
Pin	Default	Falling-edge Sampling Falling-edge Output	Falling-edge Sampling Rising-edge Output	Rising-edge Sampling Falling-edge Output	Rising-edge Sampling Rising-edge Output
MTDO	Pull-up	0	0	1	1
GPIO5	Pull-up	0	1	0	1

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-wrover_datasheet_en.pdf

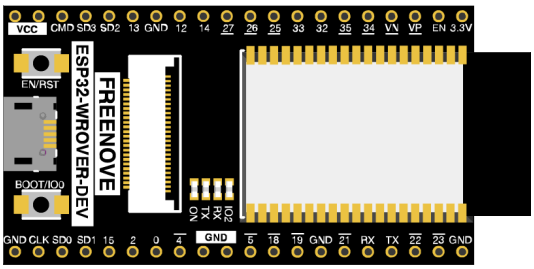
Flash Pin

GPIO6-11 has been used to connect the integrated SPI flash on the module, and is used when GPIO 0 is power on and at high level. Flash is related to the operation of the whole chip, so the external pin GPIO6-11 cannot be used as an experimental pin for external circuits, otherwise it may cause errors in the operation of the program.
GPIO16-17 has been used to connect the integrated PSRAM on the module.
Because of external pull-up, MTDI pin is not suggested to be used as a touch sensor. For details, please refer to Peripheral Interface and Sensor chapter in "ESP32_Data_Sheet".
For more relevant information, please click:
https://www.espressif.com/sites/default/files/documentation/esp32-wrover_datasheet_en.pdf.

Any concerns? ✉ support@freenove.com

Cam Pin

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



CAM_Pin	GPIO_pin
I2C_SDA	GPIO26
I2C_SCL	GPIO27
CSI_VYSNC	GPIO25
CSI_HREF	GPIO23
CSI_Y9	GPIO35
XCLK	GPIO21
CSI_Y8	GPIO34
CSI_Y7	GPIO39
CSI_PCLK	GPIO22
CSI_Y6	GPIO36
CSI_Y2	GPIO4
CSI_Y5	GPIO19
CSI_Y3	GPIO5
CSI_Y4	GPIO18

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

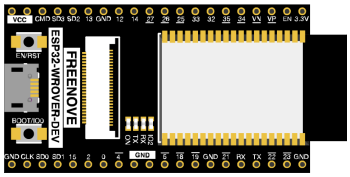
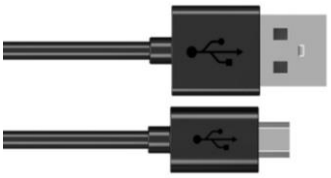
Chapter 1 LED (Important)

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

Project 1.1 Blink

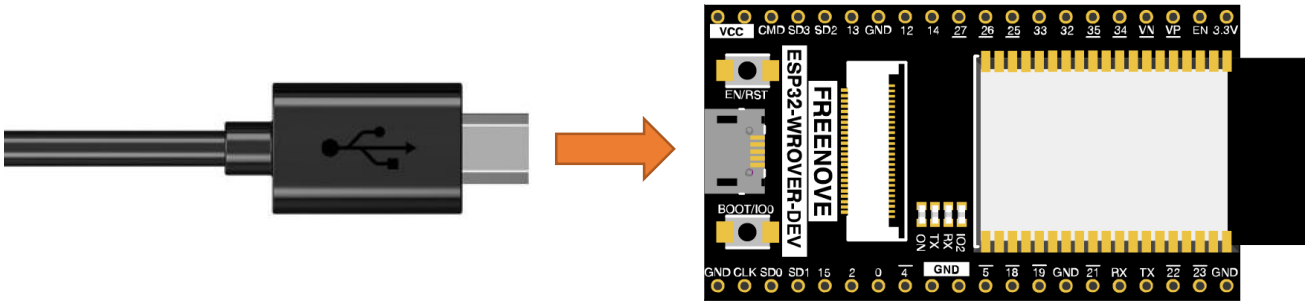
In this project, we will use ESP32 to control blinking a common LED.
If you have not yet installed Thonny, click here.
If you have not yet downloaded Micropython Firmware, click here.
If you have not yet loaded Micropython Firmware, click here.

Component List

ESP32-WROVER x1	USB cable
	

Power

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



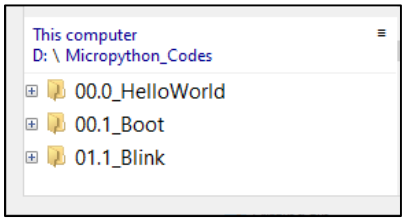
In the following projects, we only use USB cable to power ESP32-WROVER by default.
In the whole tutorial, we don’t use T extension to power ESP32-WROVER. So 5V and 3.3V (includeing EXT 3.3V) on the extension board are provided by ESP32-WROVER.
We can also use DC jack of extension board to power ESP32-WROVER.In 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_Basic_Starter_Kit_for_ESP32/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**".

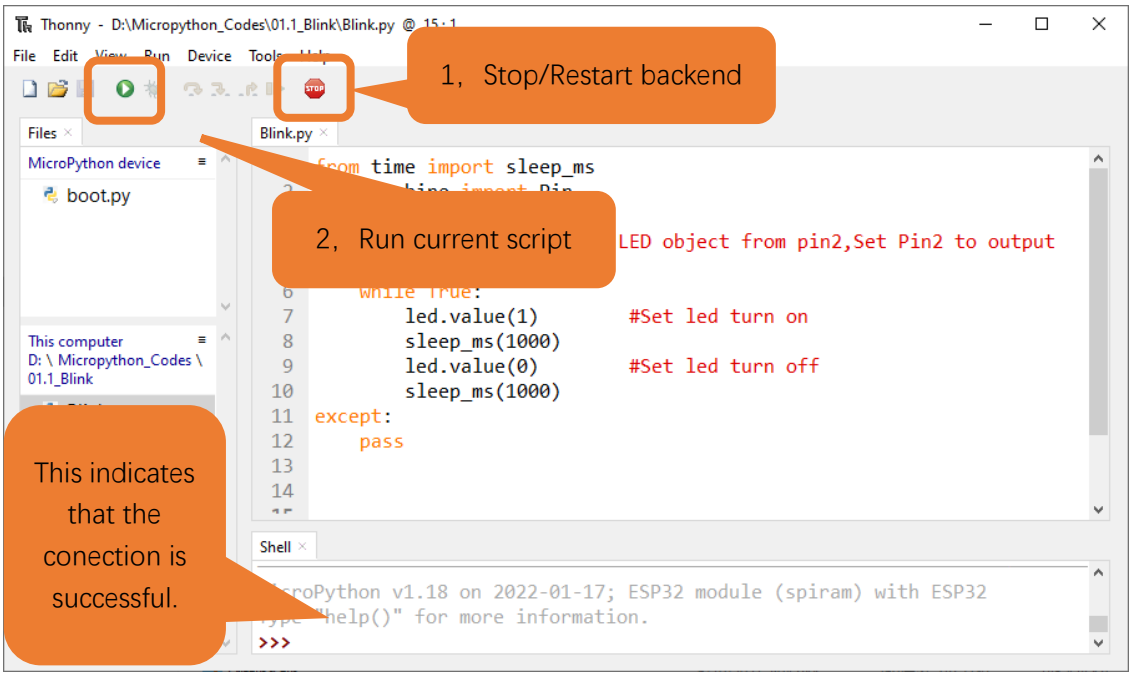
01.1_Blink

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

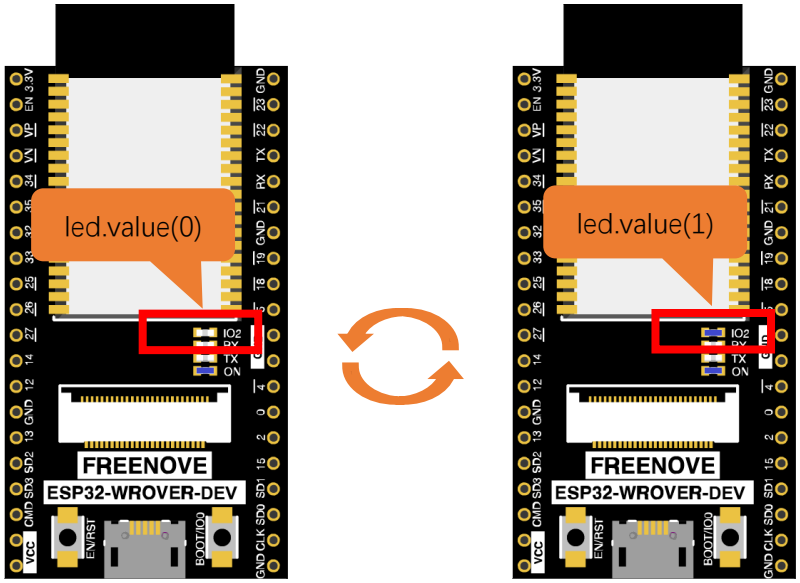


Expand folder "01.1_Blink" and double click "Blink.py" to open it. As shown in the illustration below.

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

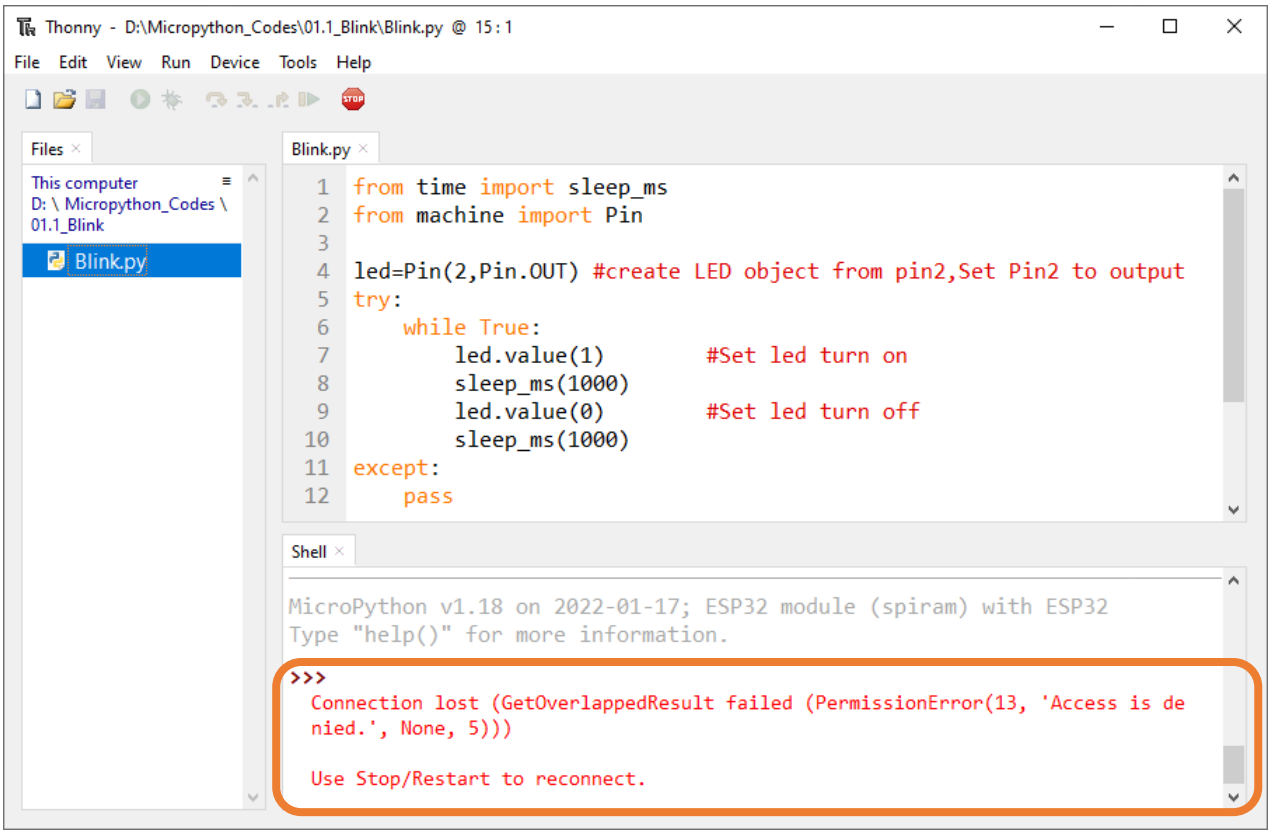


Click "Run current script" shown in the box above, the code starts to be executed and the LED in the circuit starts to blink.



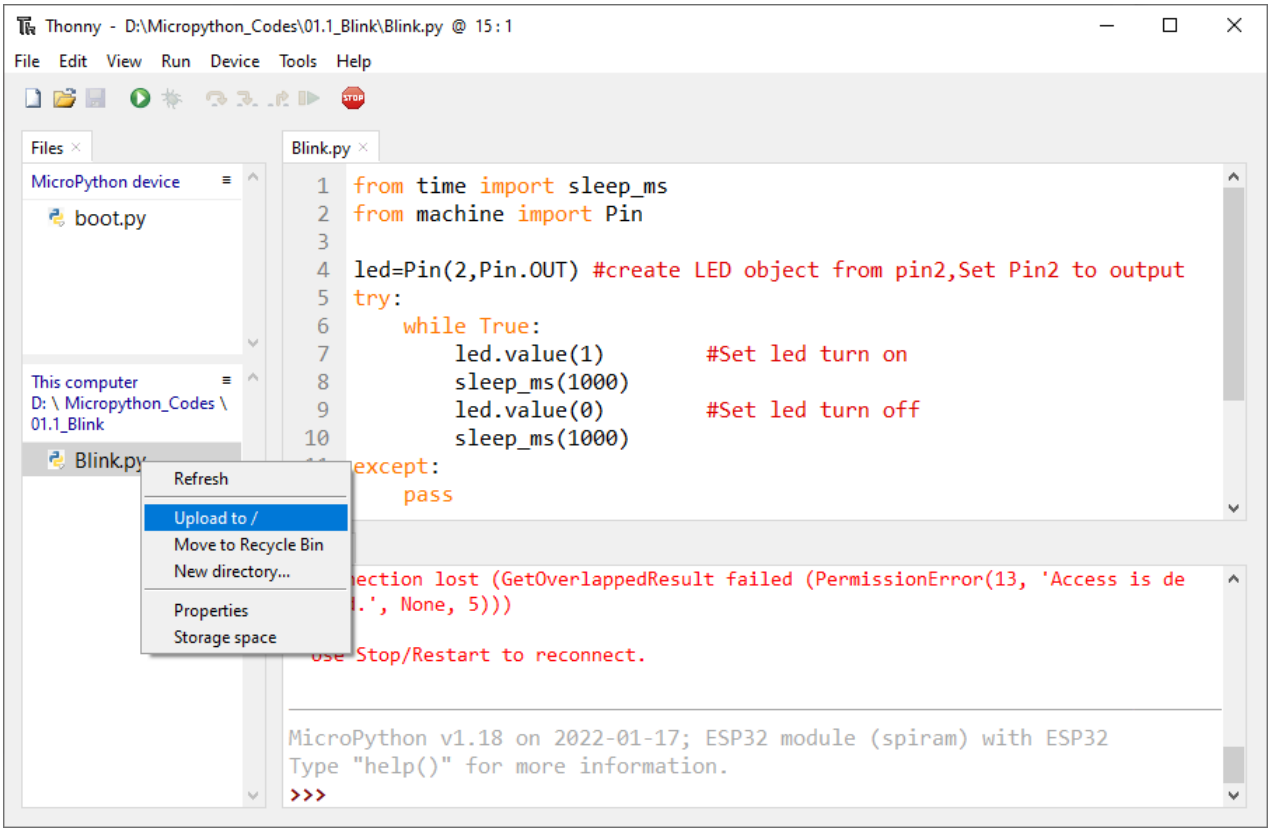
Note:

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

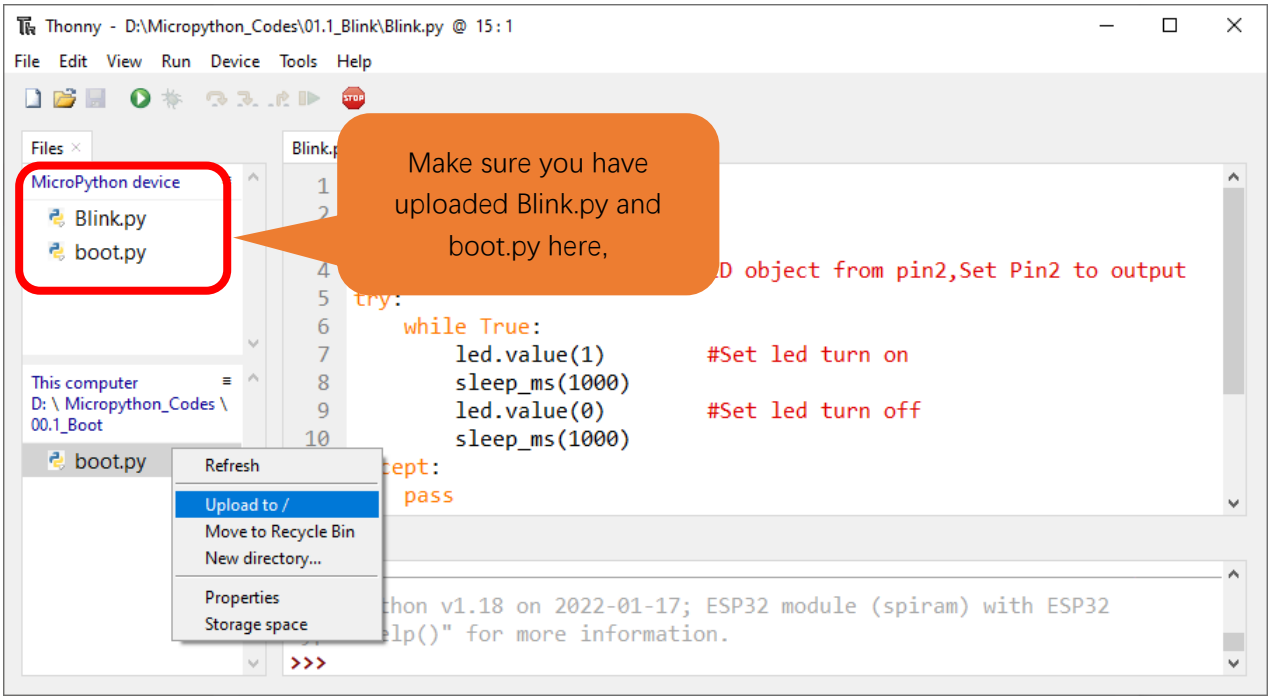


Uploading code to ESP32

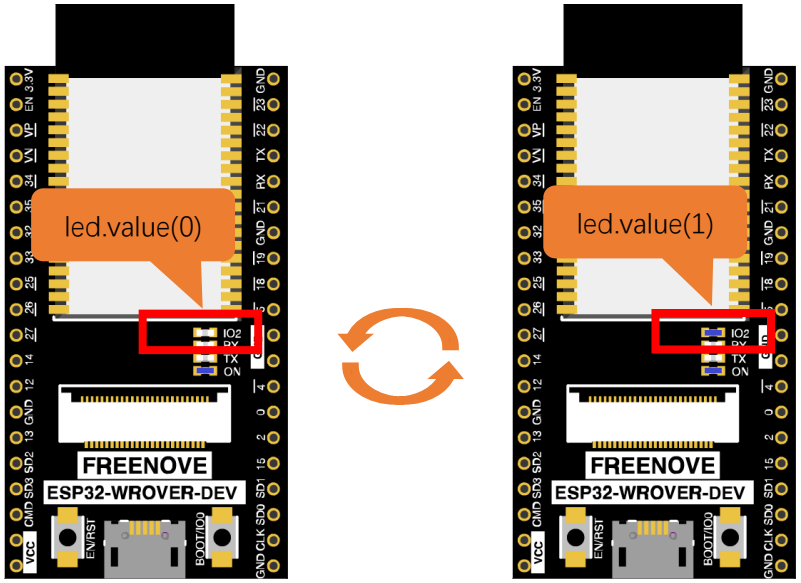
As shown in the following illustration, right-click the file Blink.py and select "Upload to /" to upload code to ESP32.



Upload boot.py in the same way.

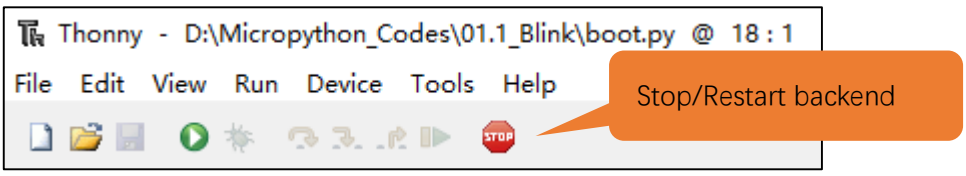


Press the reset key of ESP32 and you can see LED is ON for one second and then OFF for one second, which repeats in an endless loop.



Note:

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



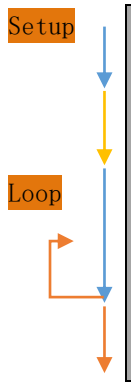
If you have any concerns, please contact us via: support@freenove.com

The following is the program code:

```
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
```

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.




```
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:
        ...
except:
    pass
```

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

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

Each time when using the functions of ESP32, you need to import modules corresponding to those functions: Import sleep_ms module of time module and Pin module of machine module.

```
from time import sleep_ms
from machine import Pin
```

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

```
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.

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

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

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

Execute codes in a while loop.

```
    while True:
        ...
```

Put statements that may cause an error in “try” block and the executing statements when an error occurs in “except” block. In general, when the program executes statements, it will execute those in “try” block. However, when an error occurs to ESP32 due to some interference or other reasons, it will execute statements in “except” block.

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

```
5    try:
...      ...
11   except:
12       pass
```

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

```
9    #Set led turn on
```

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

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

How to import python files

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

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

```
import random

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

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

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

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

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

Reference

Class machine

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

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

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

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

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

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

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

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

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

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

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

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

Class Pin(id[, mode, pull, value])

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

id: Arbitrary pin number

mode: Mode of pins

Pin.IN: Input Mode

Pin.OUT: Output Mode

Pin.OPEN_DRAIN: Open-drain Mode

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

None: No pull up or pull down resistors

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

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

Value: State of the pin level, 0/1

Pin.init(mode, pull): Initialize pins

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

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

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

trigger:

Pin.IRQ_FALLING: interrupt on falling edge

Pin.IRQ_RISING: interrupt on rising edge

3: interrupt on both edges

Handler: callback function

Class time

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

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

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

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

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

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

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

time.ticks_us(): Returns microsecond

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

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

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

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

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

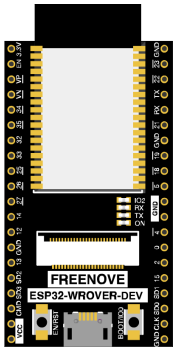
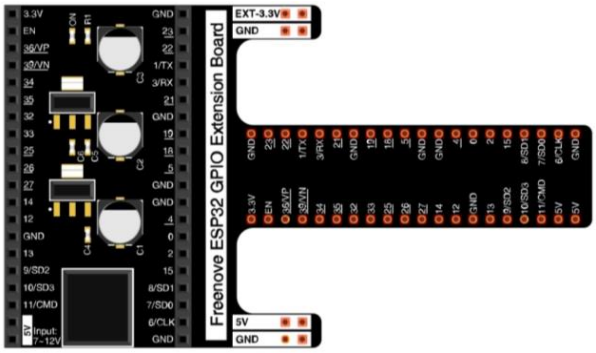
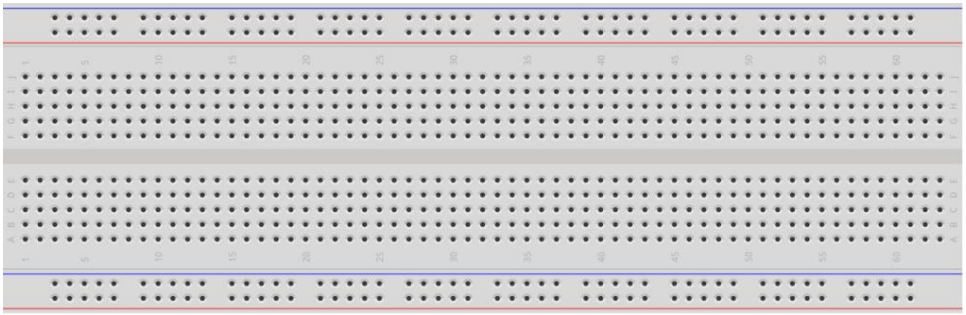
old_t: Starting time

new_t: Ending time

Project 1.2 Blink

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

Component List

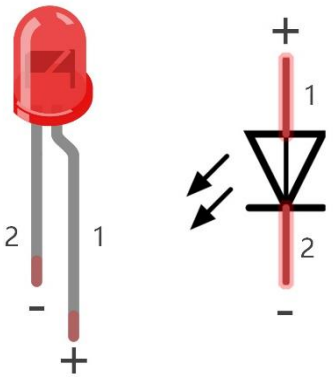
ESP32-WROVER x1 	GPIO Extension Board x1 
Breadboard x1 	
LED x1	Resistor 220Ω x1
Jumper M/M x2	

Component knowledge

LED

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

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



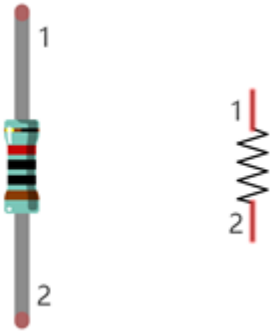
LED	Voltage	Maximum current	Recommended current
Red	1.9-2.2V	20mA	10mA
Green	2.9-3.4V	10mA	5mA
Blue	2.9-3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

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

Resistor

Resistors use Ohms (Ω) as the unit of measurement of their resistance (R). 1MΩ=1000kΩ, 1kΩ=1000Ω.

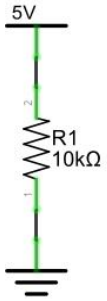
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



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

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

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

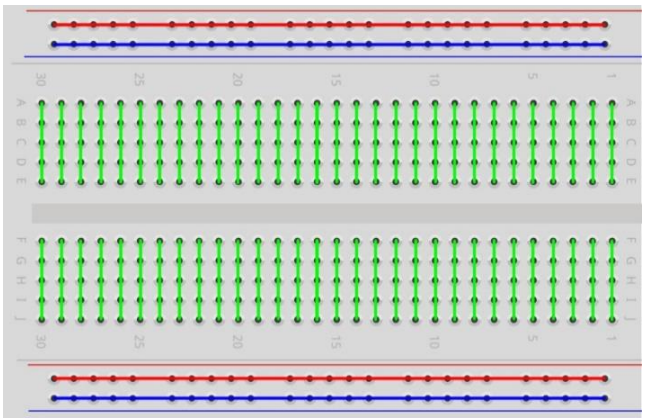


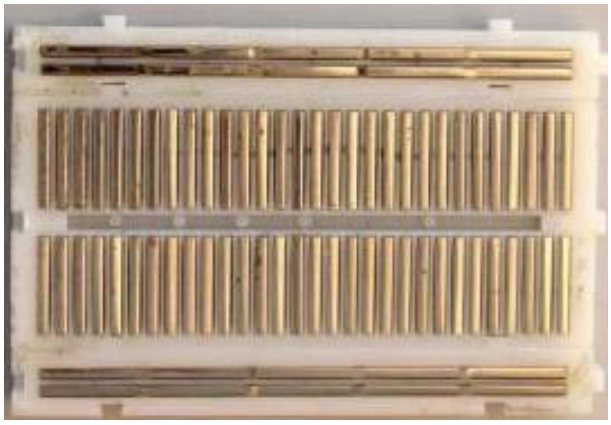
WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

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

Breadboard

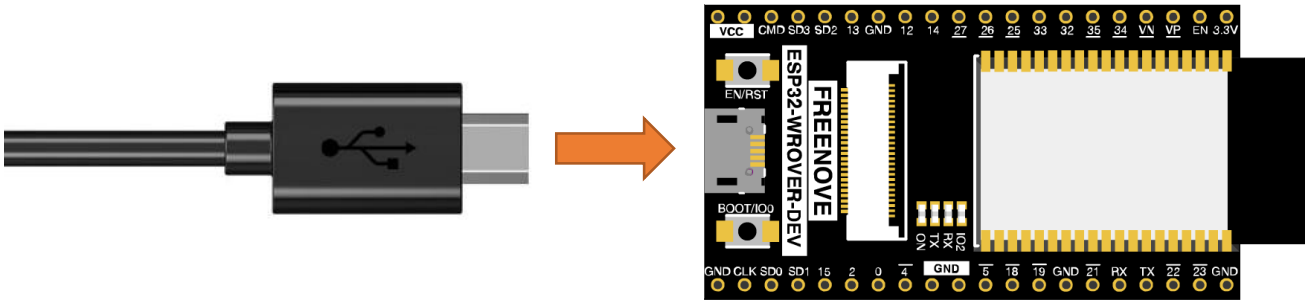
Here we have a small breadboard as an example of how the rows of holes (sockets) are electrically attached. The left picture shows the way to connect pins. The right picture shows the practical internal structure.





Power

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



Later, we only use USB cable to power ESP32-WROVER in default.

In the whole tutorial, we don't use T extension to power ESP32-WROVER. So 5V and 3.3V (include EXT 3.3V) on the extension board are from ESP32-WROVER.

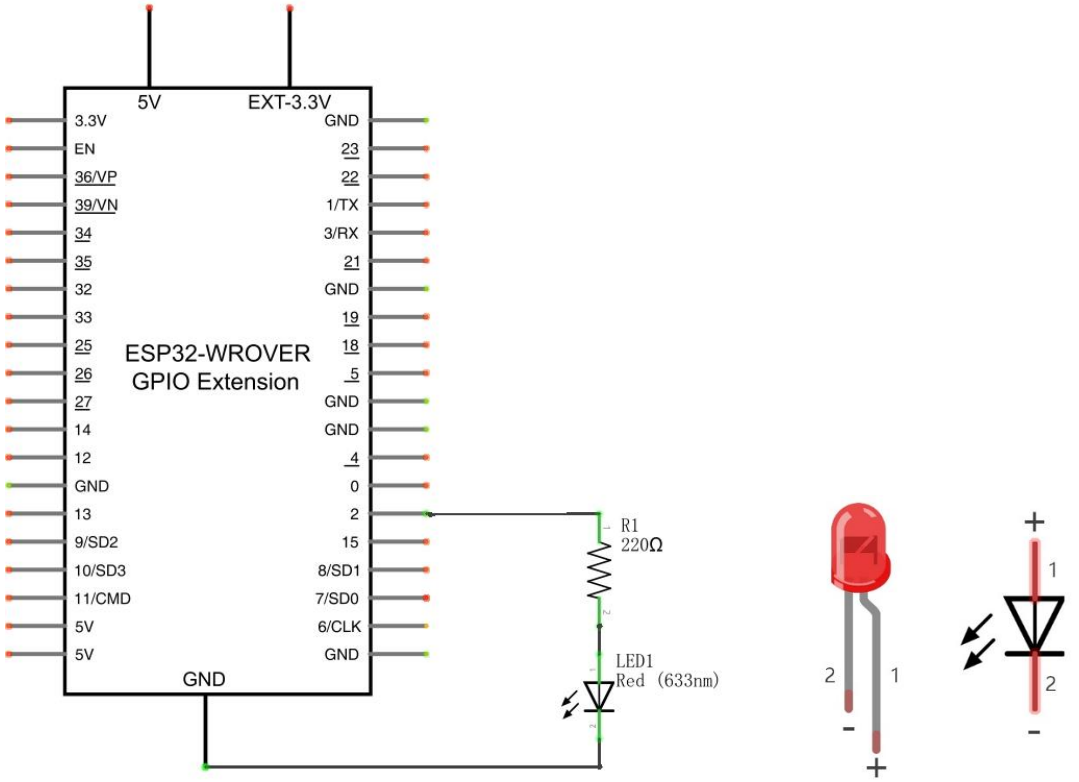
We can also use DC jack of extension board to power ESP32-WROVER. Then 5v and EXT 3.3v on extension board are from external power resource.

Circuit

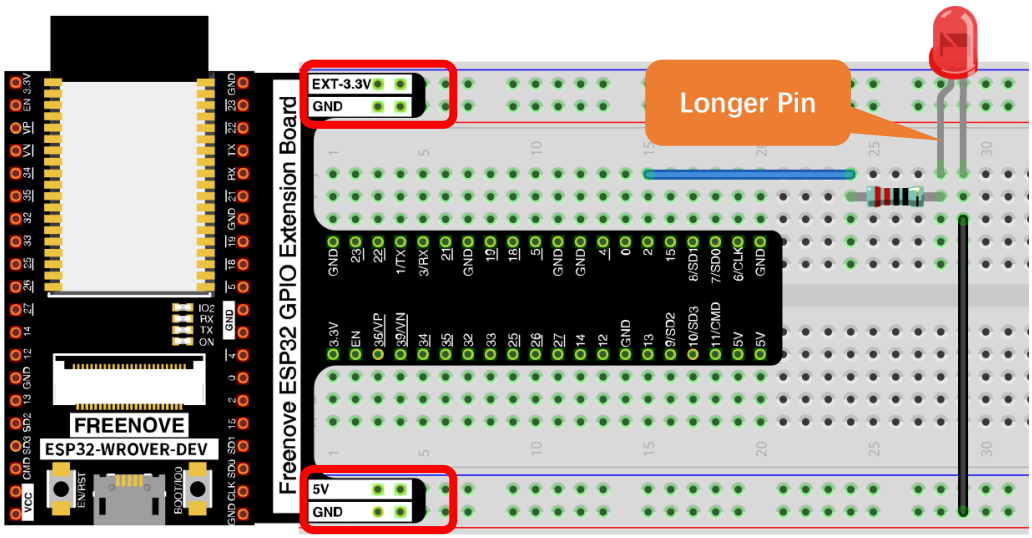
First, disconnect all power from the ESP32-WROVER. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the PC to ESP32-WROVER.

CAUTION: Avoid any possible short circuits (especially connecting 5V or GND, 3.3V and GND)! WARNING: A short circuit can cause high current in your circuit, create excessive component heat and cause permanent damage to your hardware!

Schematic diagram



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



Don't rotate ESP32-WROVER 180° for connection.

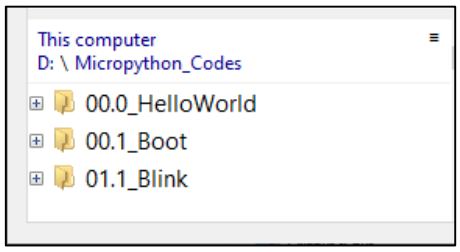
Any concerns? ✉ support@freenove.com

Code

Codes used in this tutorial are saved in "**Freenove_Basic_Starter_Kit_for_ESP32/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**".

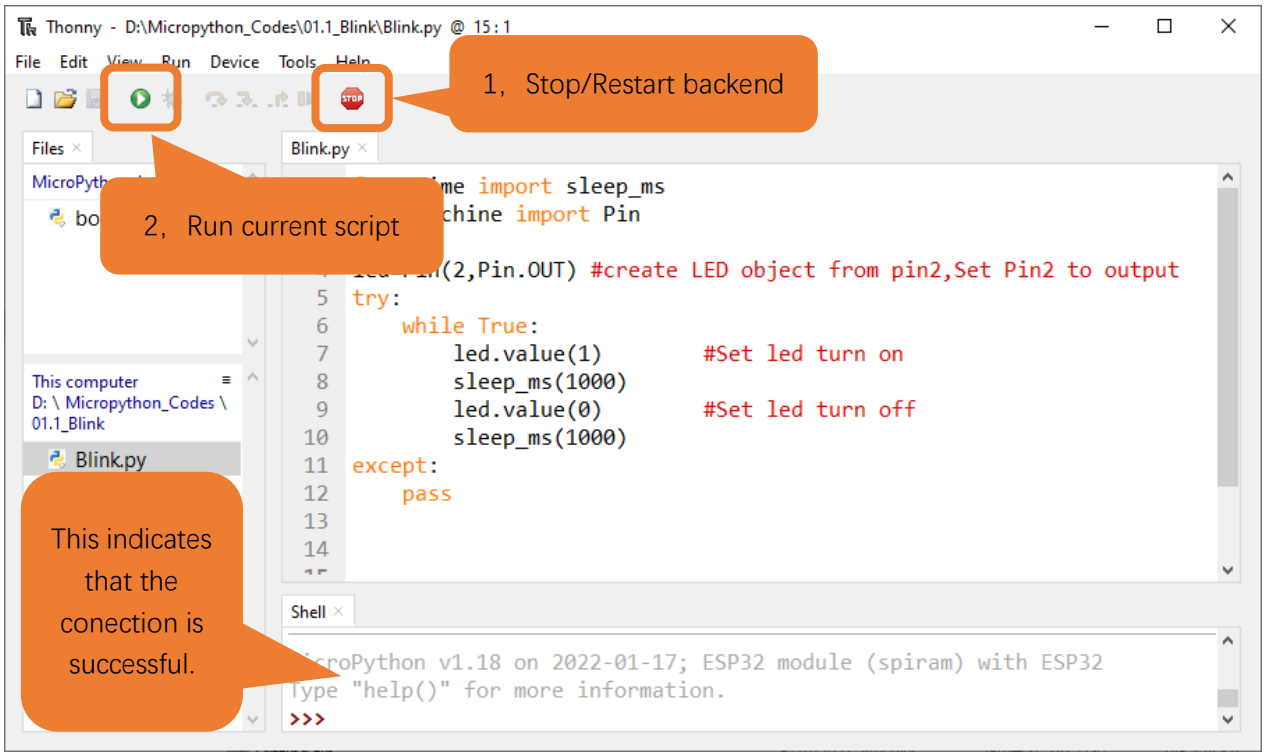
01.1_Blink

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



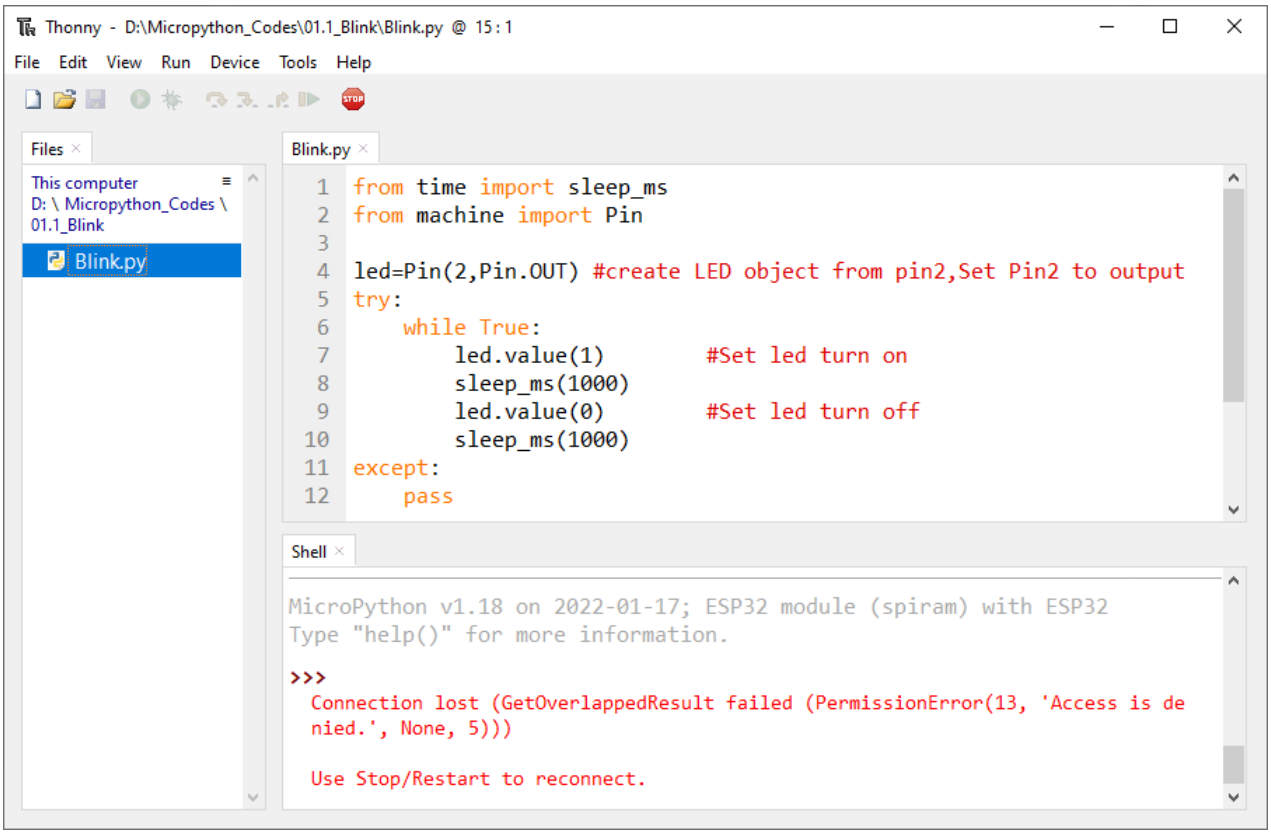

```
me import sleep_ms
chine import Pin

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

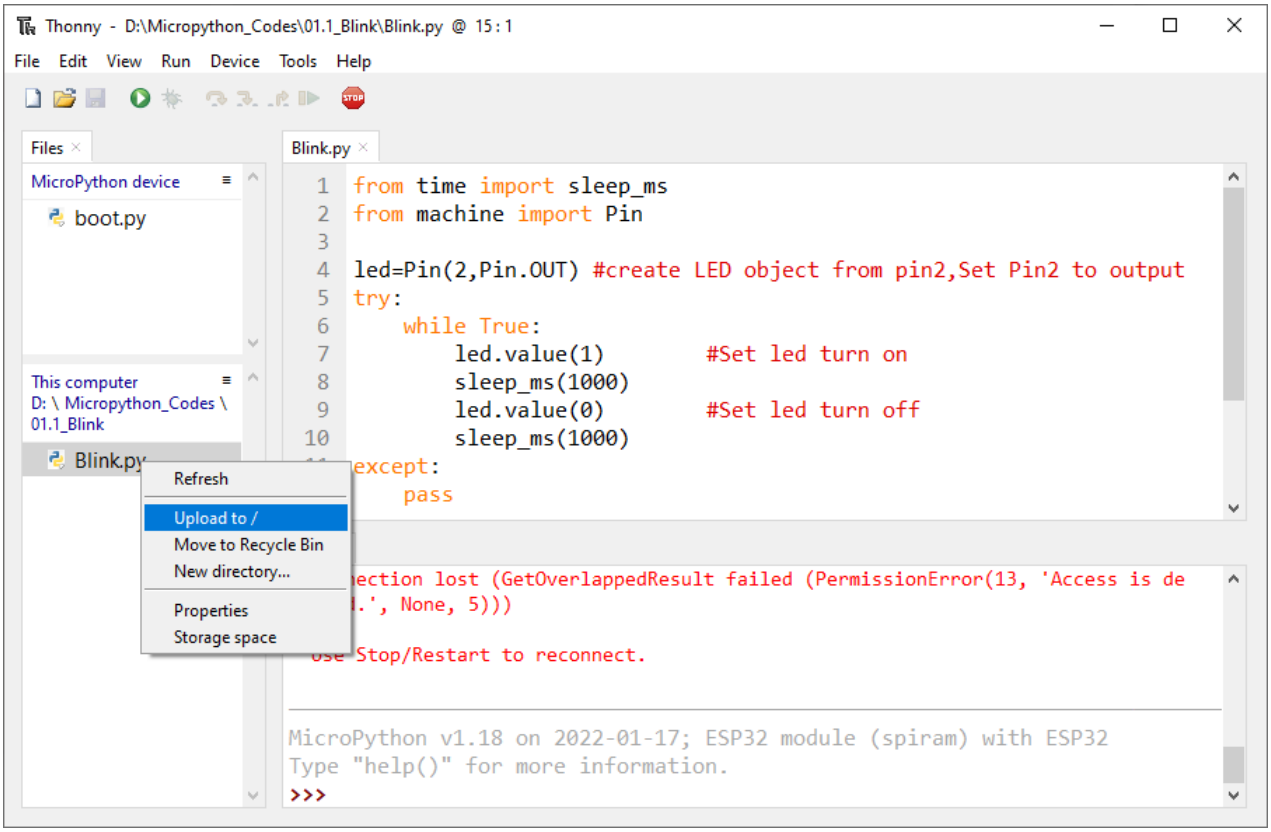
Note:

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

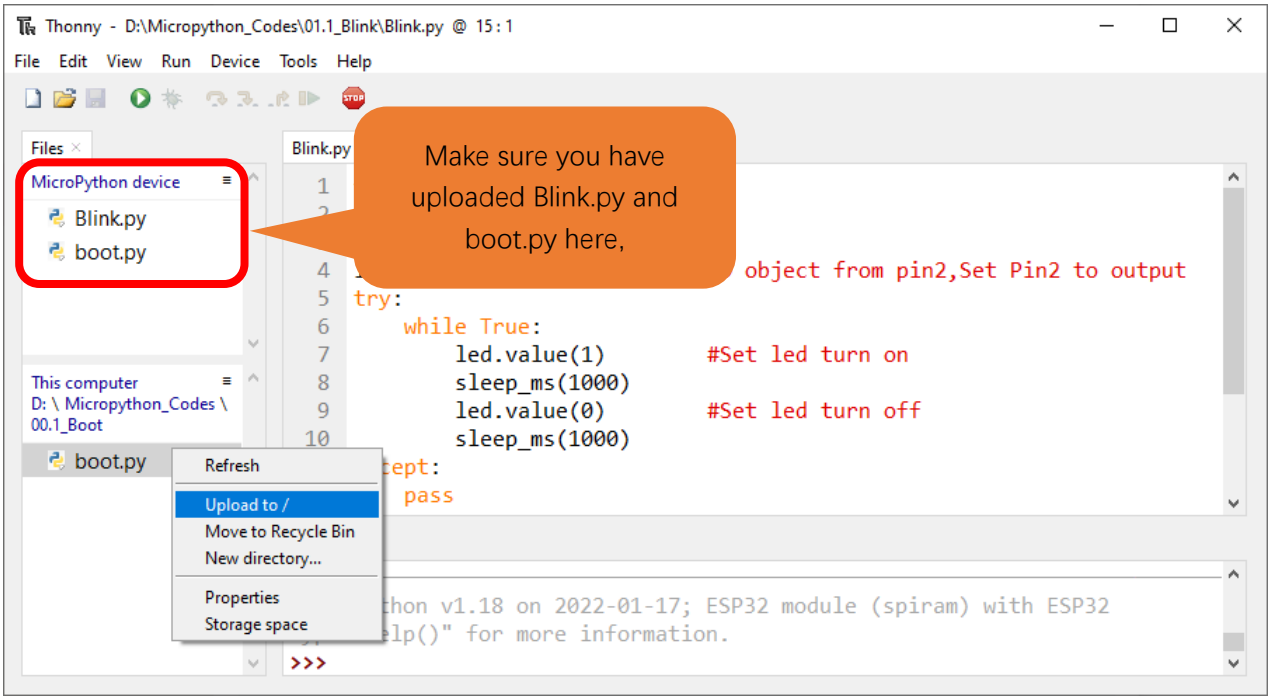


Uploading code to ESP32

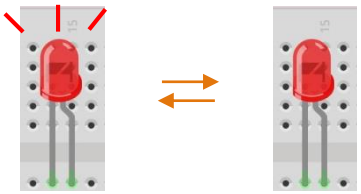
As shown in the following illustration, right-click the file Blink.py and select "Upload to /" to upload code to ESP32.



Upload boot.py in the same way.

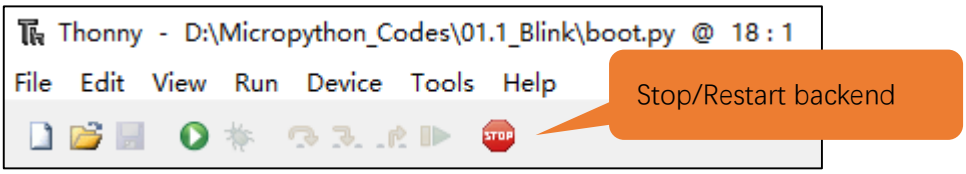


Press the reset key of ESP32 and you can see LED is ON for one second and then OFF for one second, which repeats in an endless loop.



Note:

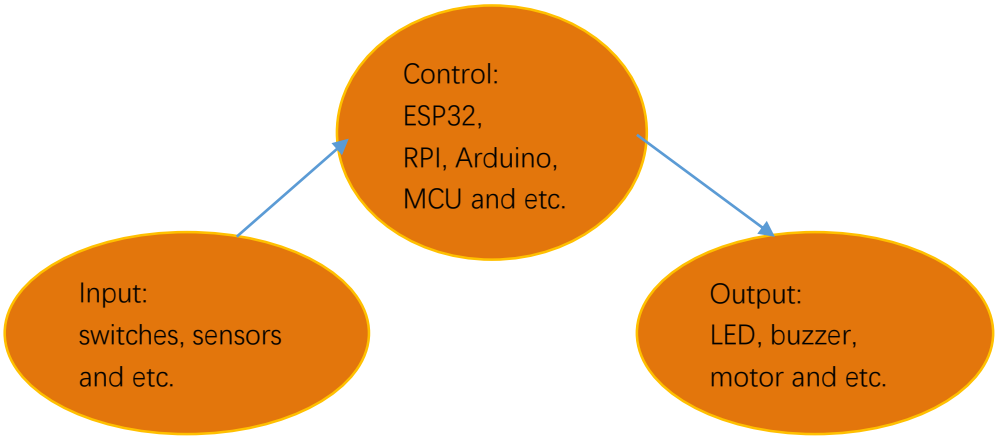
Codes here is run offline. If you want to stop running offline and enter Shell, just click "Stop" in Thonny.



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

Chapter 2 Button & LED

Usually, there are three essential parts in a complete automatic control device: INPUT, OUTPUT, and CONTROL. In last section, the LED module was the output part and ESP32 was the control part. In practical applications, we not only make LEDs flash, but also make a device sense the surrounding environment, receive instructions and then take the appropriate action such as LEDs light up, make a buzzer turn ON and so on.

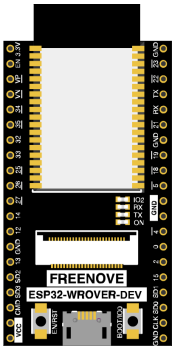
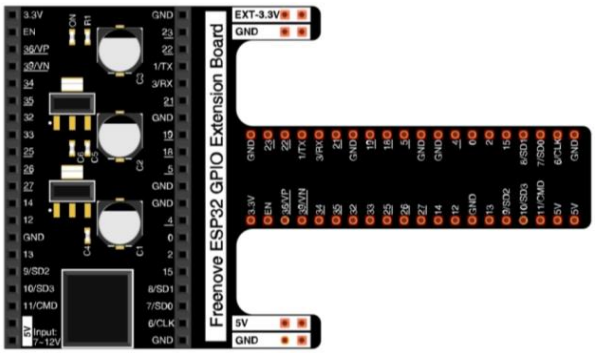
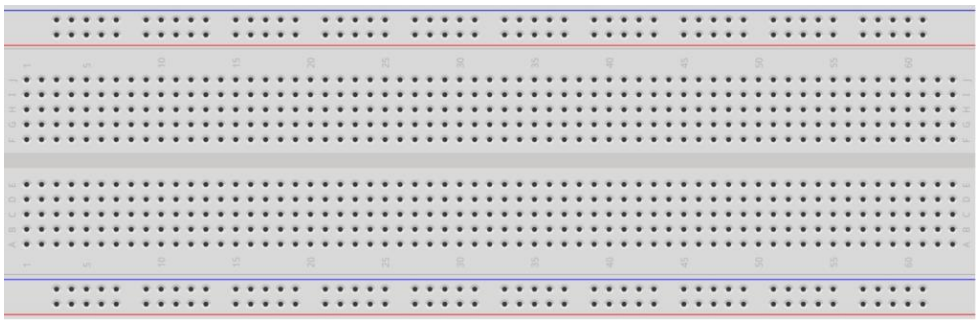


Next, we will build a simple control system to control an LED through a push button switch.

Project 2.1 Button & LED

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

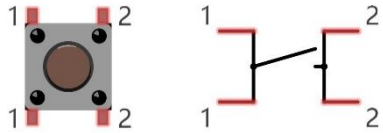
Component List

ESP32-WROVER x1 	GPIO Extension Board x1 			
Breadboard x1 				
Jumper M/M x4 	LED x1 	Resistor 220Ω x1 	Resistor 10kΩ x2 	Push button x1

Component knowledge

Push button

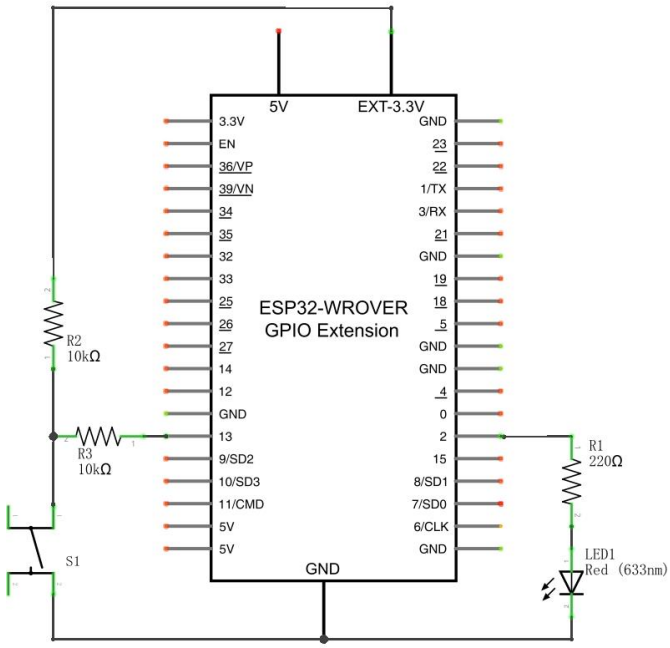
This type of Push Button Switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



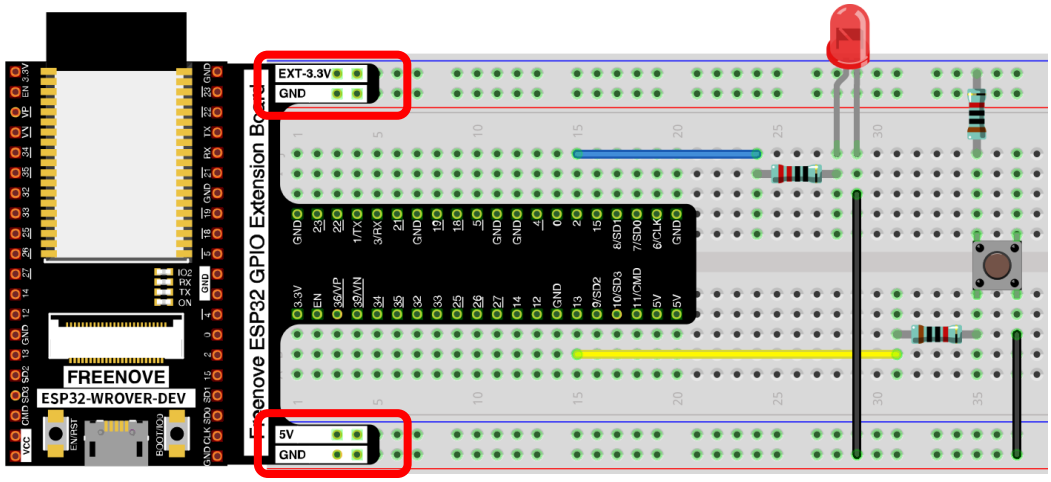
When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

Circuit

Schematic diagram



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



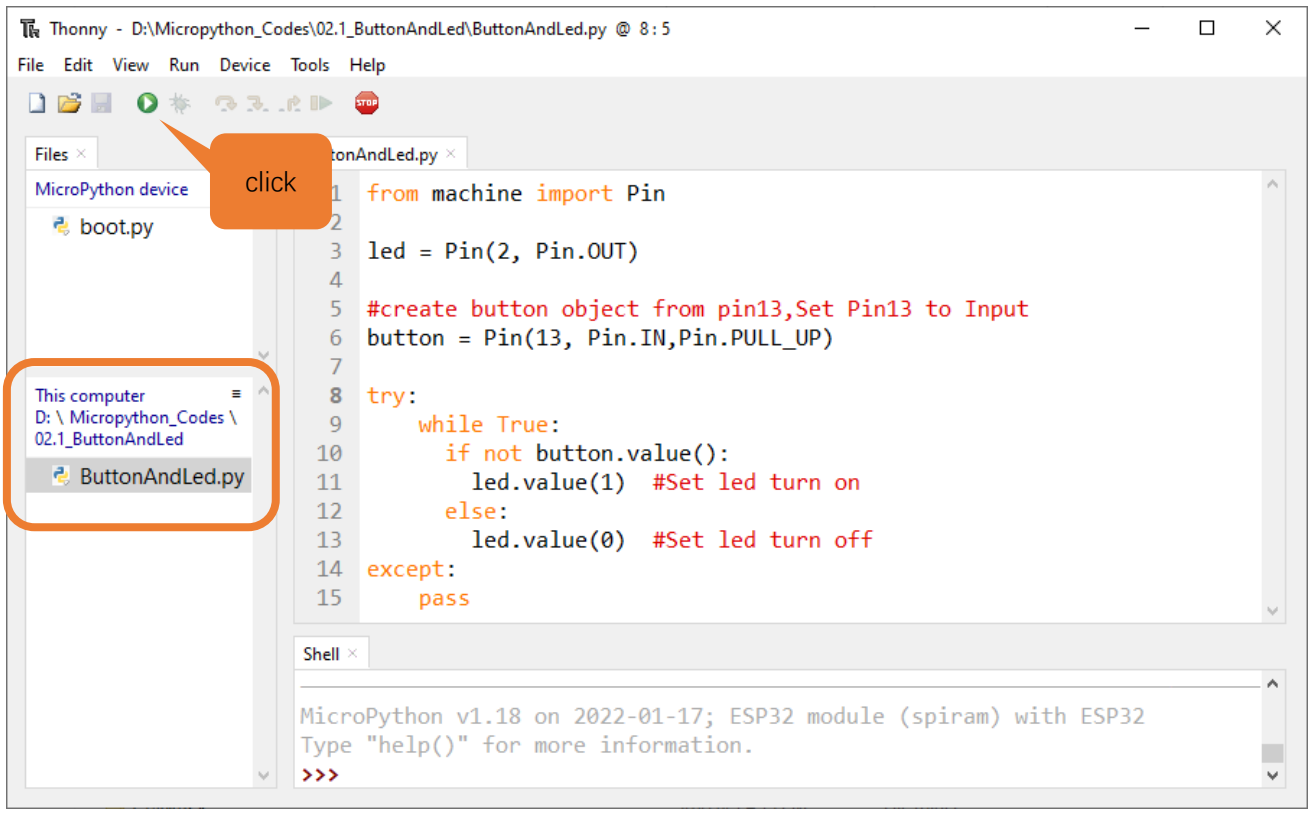
Code

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

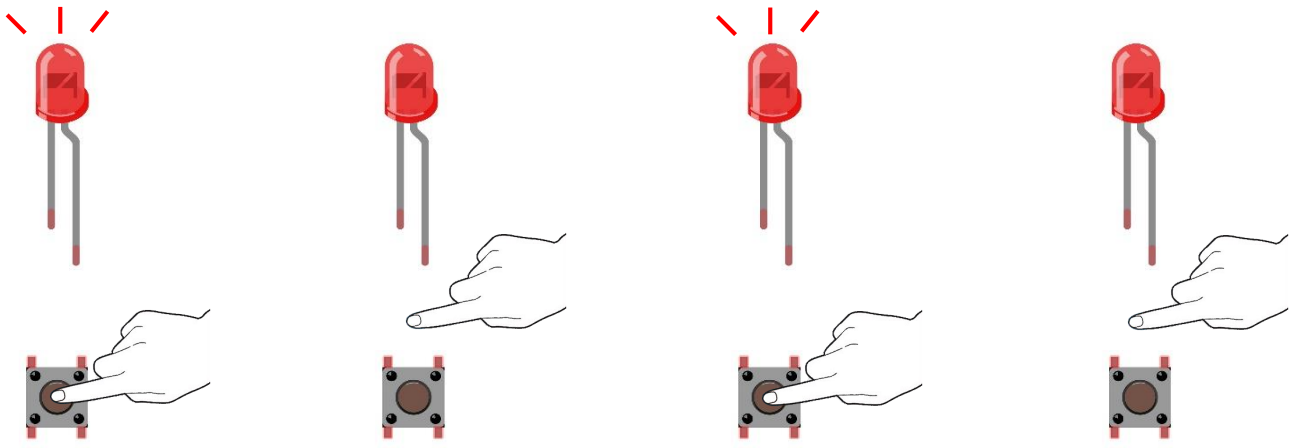
Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

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

02.1_ButtonAndLed

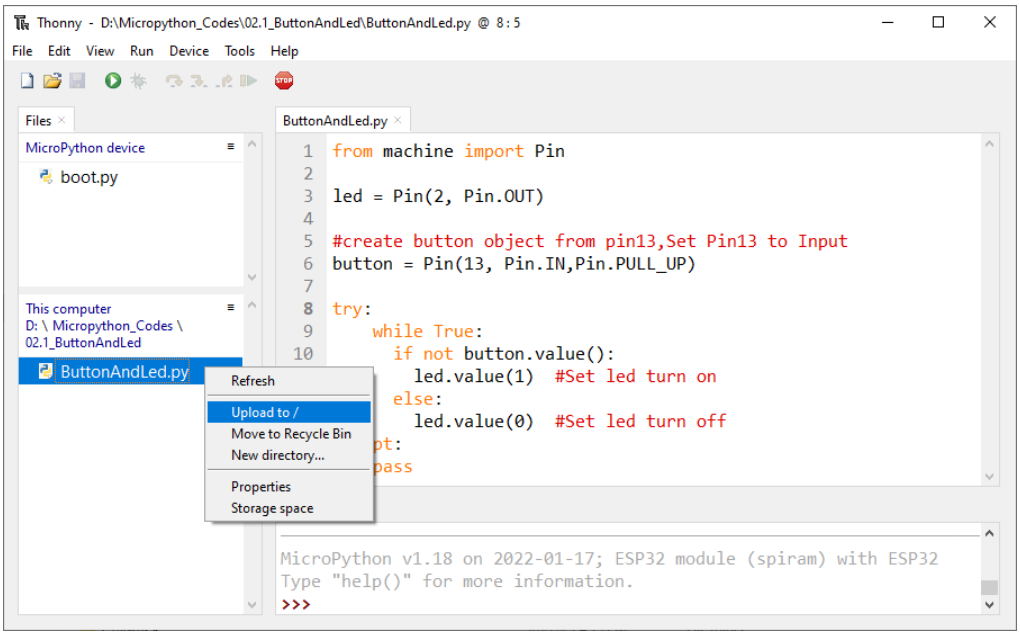


Click "Run current script" shown in the box of the above illustration, press the push button switch, LED turns ON; release the switch, LED turns OFF.

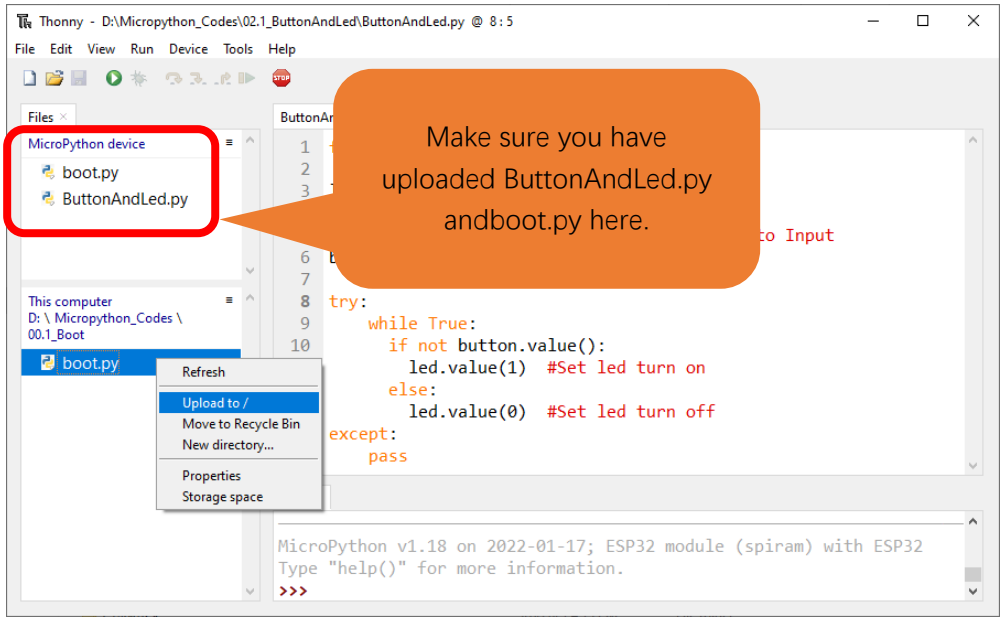


Upload Code to ESP32

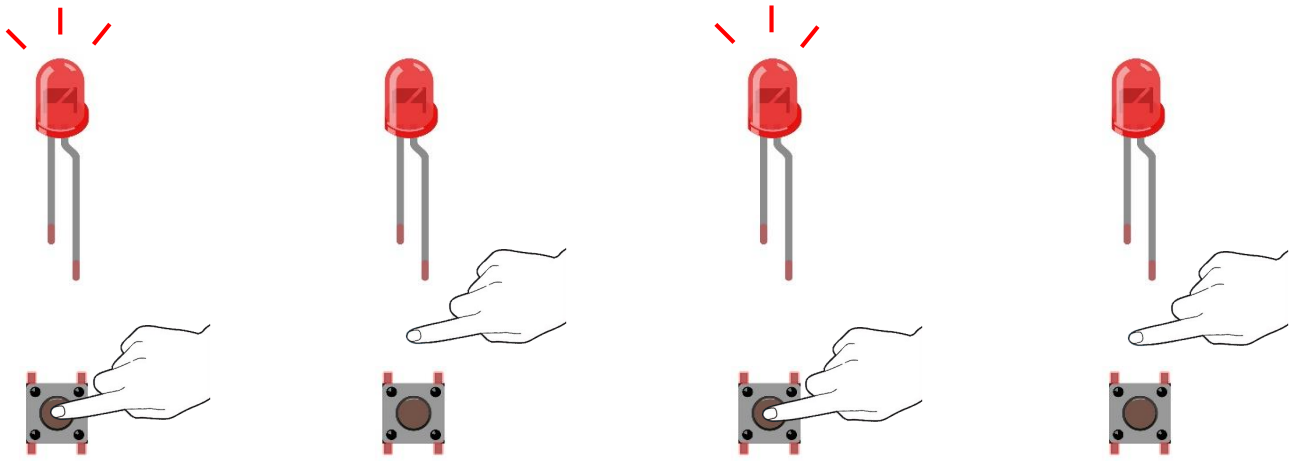
As shown in the following illustration, right-click file 02.1_ButtonAndLed and select "Upload to /" to upload code to ESP32.



Upload boot.py in the same way.



Press ESP32's reset key, and then push the button switch, LED turns ON; Push the button again, LED turns OFF.



The following is the program code:

```
1  from machine import Pin
2
3  led = Pin(2, Pin.OUT)
4
5  #create button object from pin13,Set Pin13 to Input
6  button = Pin(13, Pin.IN,Pin.PULL_UP)
7
8  try:
9      while True:
10       if not button.value():
11         led.value(1)  #Set led turn on
12       else:
13         led.value(0)  #Set led turn off
14 except:
15     pass
```

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

```
1  from machine import Pin
```

In the circuit connection, LED and Button are connected with GPIO2 and GPIO13 respectively, so define led and button as 2 and 13 respectively.

```
3  led = Pin(2, Pin.OUT)
4
5  #create button object from pin13,Set Pin13 to Input
6  button = Pin(13, Pin.IN,Pin.PULL_UP)
```

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

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

If statement is used to execute the next statement when a certain condition is proved to be true (or non0). It is often used together with “else” statement, which judges other statements except the if statement. If you need to judge if the result of a condition is 0, you can use if not statement.

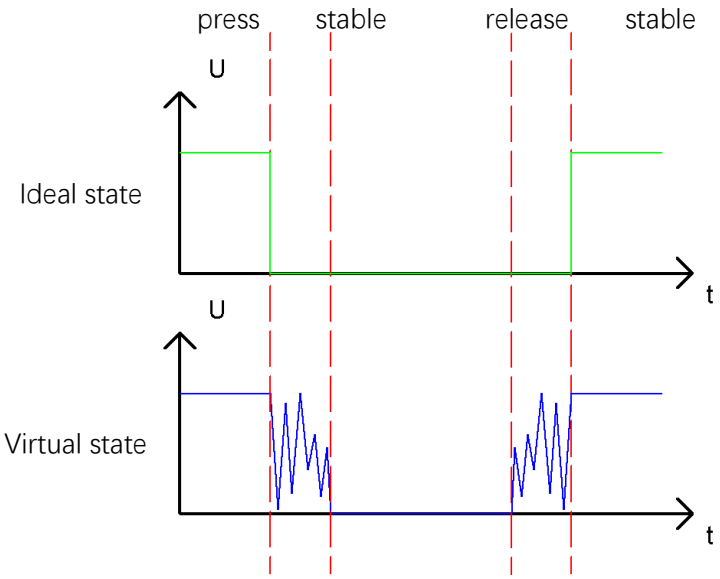
```
10  if not button.value():
11      …
12  else:
13      …
```

Project 2.2 MINI table lamp

We will also use a Push Button Switch, LED and ESP32 to make a MINI Table Lamp but this will function differently: Press the button, the LED will turn ON, and pressing the button again, the LED turns OFF. The ON switch action is no longer momentary (like a door bell) but remains ON without needing to continually press on the Button Switch.
First, let us learn something about the push button switch.

Debounce for Push Button

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



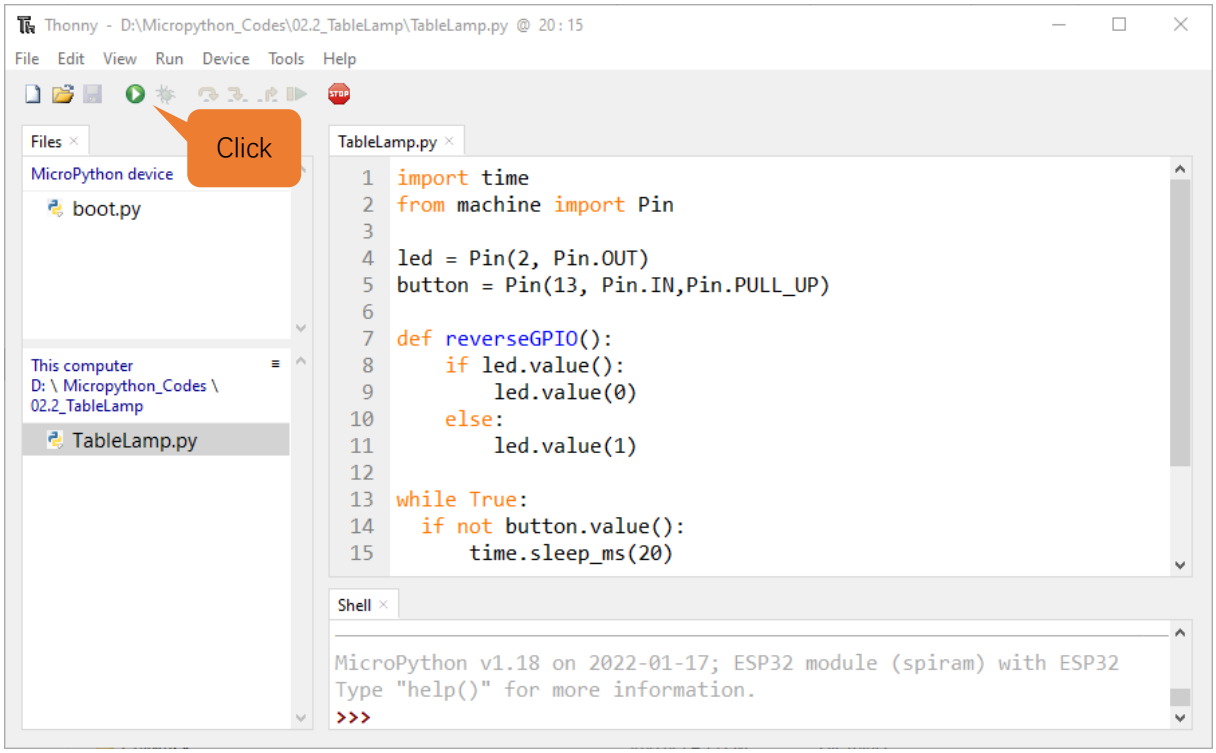
Therefore, if we can directly detect the state of the Push Button Switch, there are multiple pressing and releasing actions in one pressing cycle. This buffeting will mislead the high-speed operation of the microcontroller to cause many false decisions. Therefore, we need to eliminate the impact of buffeting.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

Code

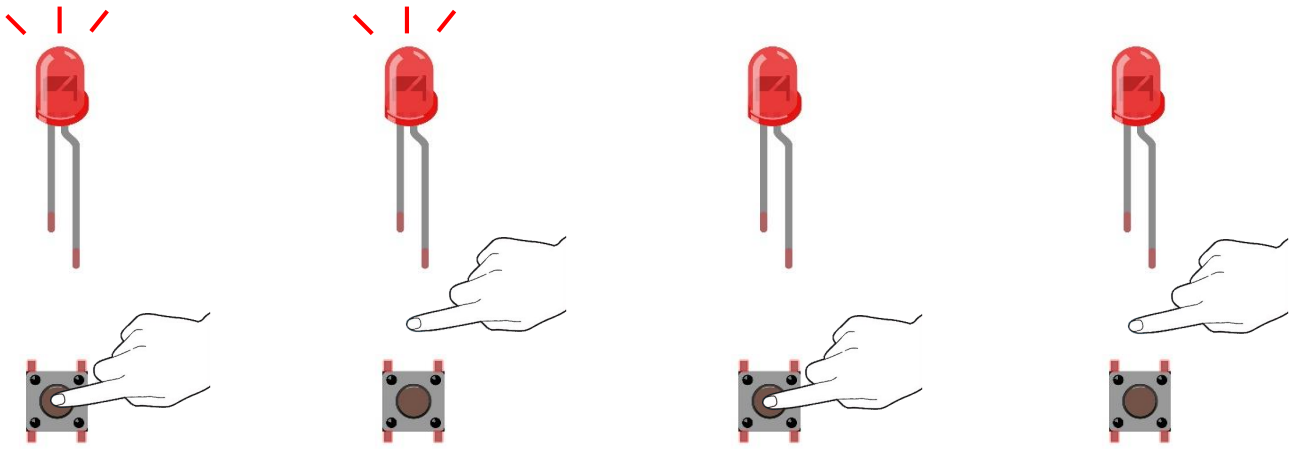
02.2_Tablelamp

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

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



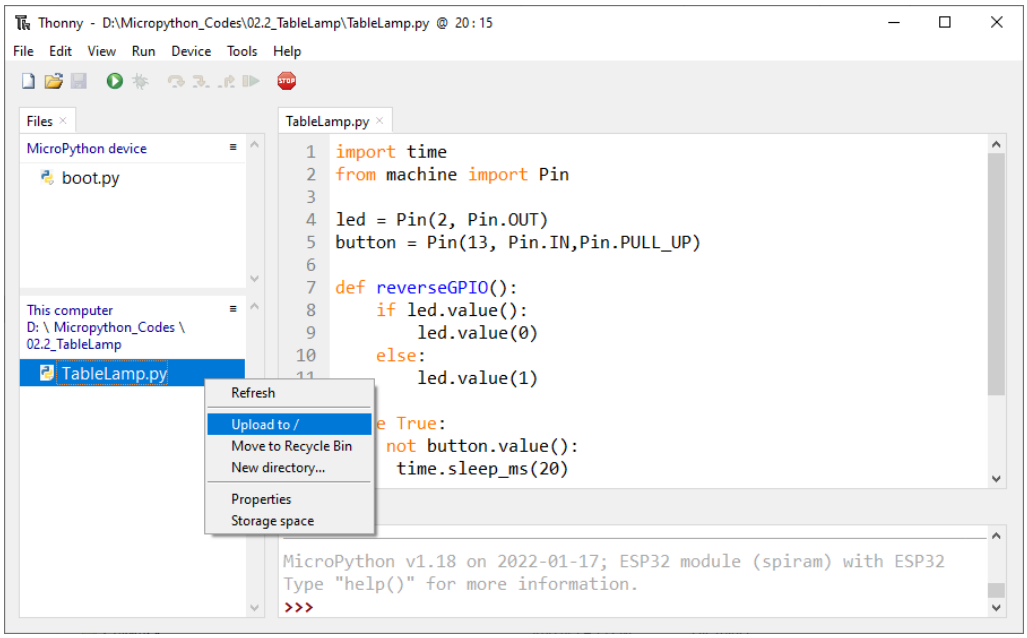
Click "Run current script" shown in the box of the above illustration, press the push button switch, LED turns ON; press it again, LED turns OFF.



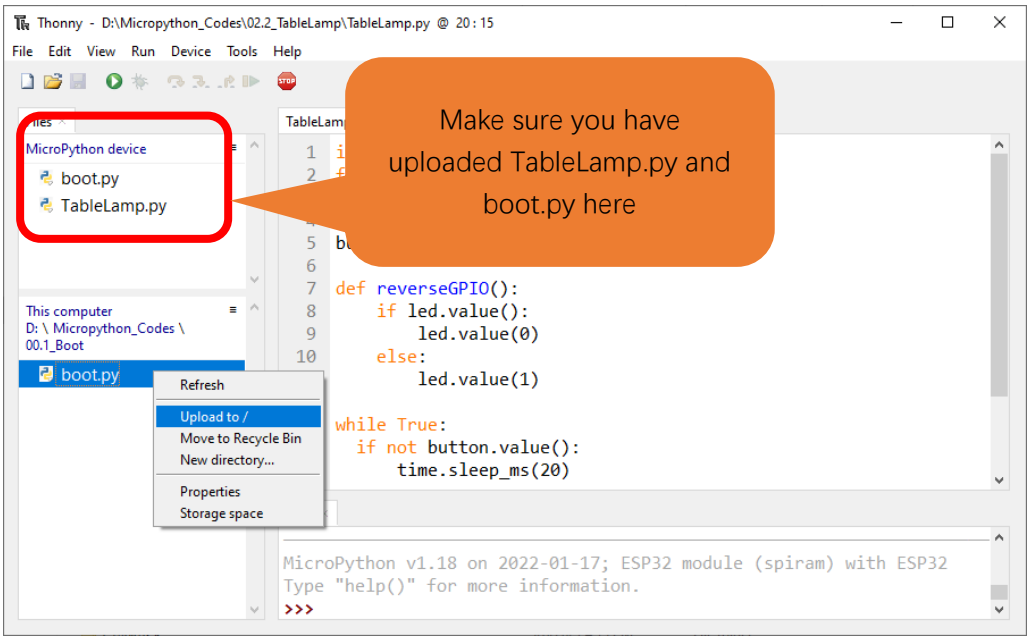
If you have any concerns, please contact us via: support@freenove.com

Upload code to ESP32

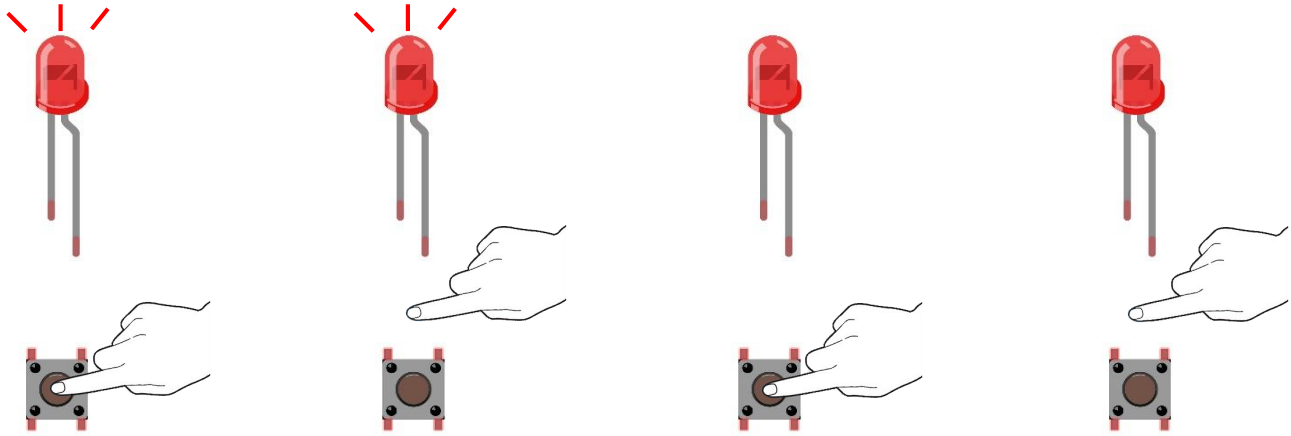
As shown in the following illustration, right-click file 02.2_TableLamp and select "Upload to /" to upload code to ESP32.



Upload boot.py in the same way.



Press ESP32's reset key, and then push the button switch, LED turns ON; Push the button again, LED turns OFF.



The following is the program code:

```
import time
from machine import Pin

led = Pin(2, Pin.OUT)
button = Pin(13, Pin.IN, Pin.PULL_UP)

def reverseGPIO():
    if led.value():
        led.value(0)
    else:
        led.value(1)

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

```

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

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

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

```
7   def reverseGPIO():
8       if led.value():
9           led.value(0)
10      else:
11          led.value(1)
```

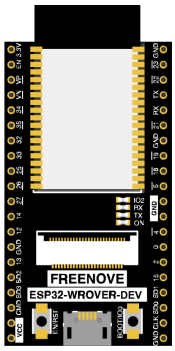
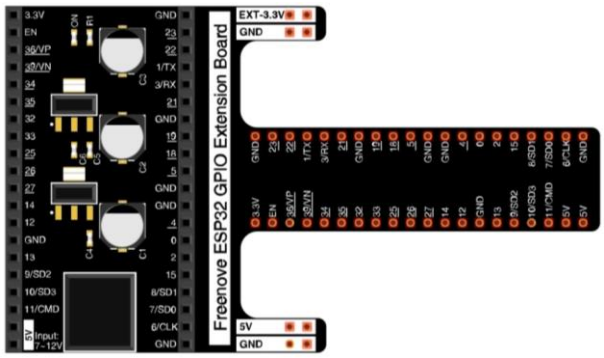
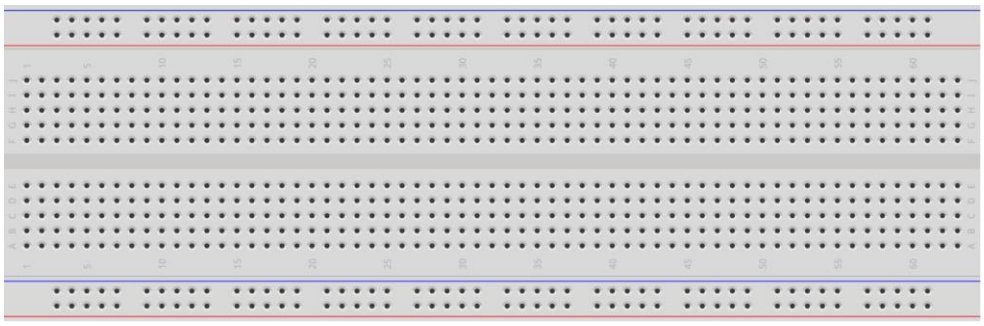
Chapter 3 LED Bar

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

Project 3.1 Flowing Light

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

Component List

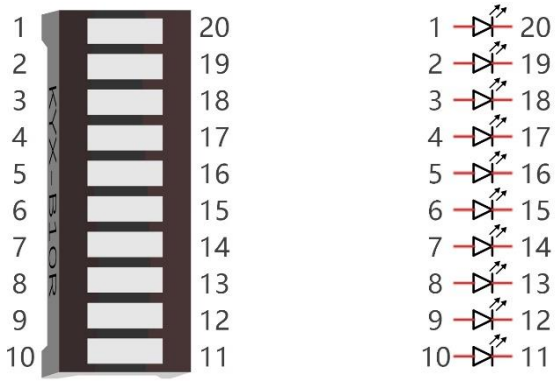
ESP32-WROVER x1 	GPIO Extension Board x1 	
Breadboard x1 		
Jumper M/M x10 	LED bar graph x1 	Resistor 220Ω x10

Component knowledge

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

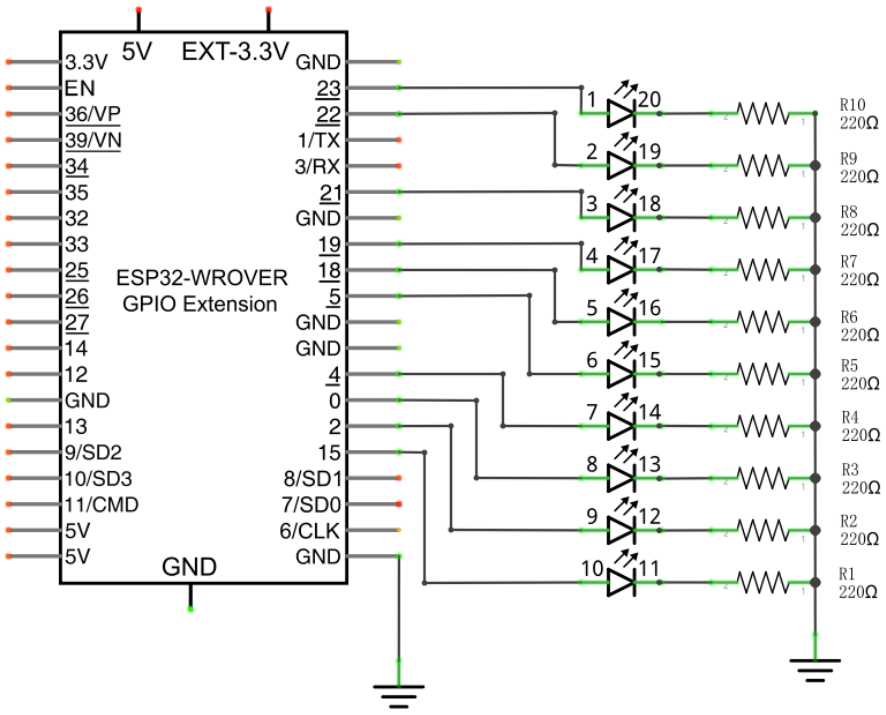
LED bar

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

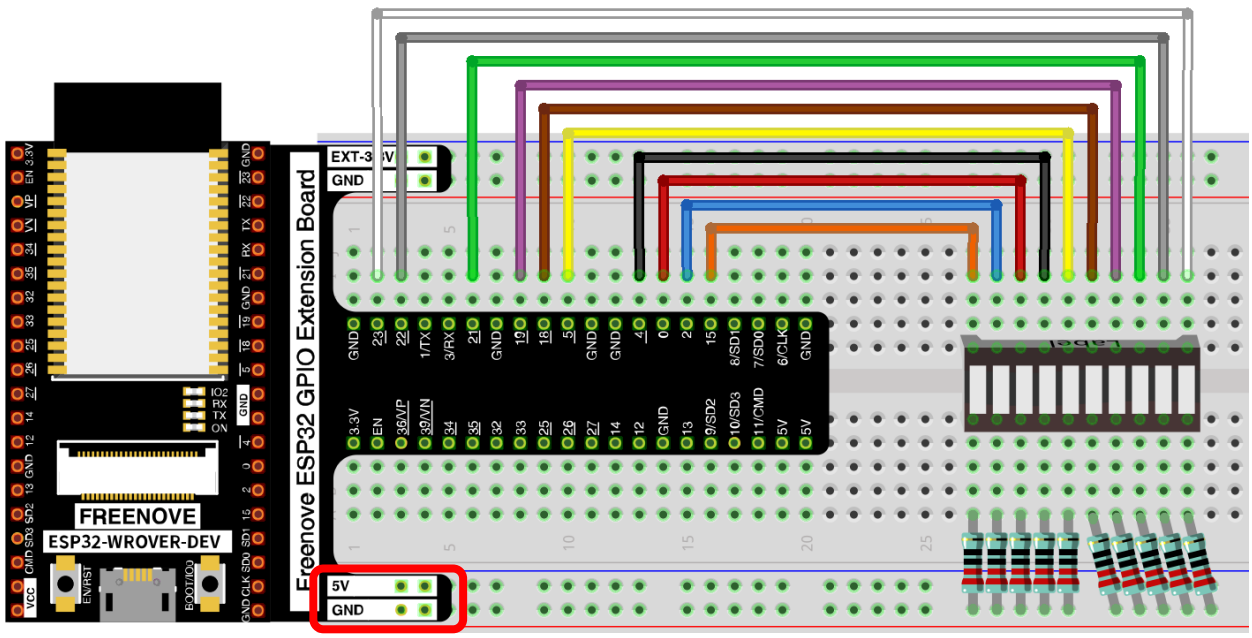


Circuit

Schematic diagram



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



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

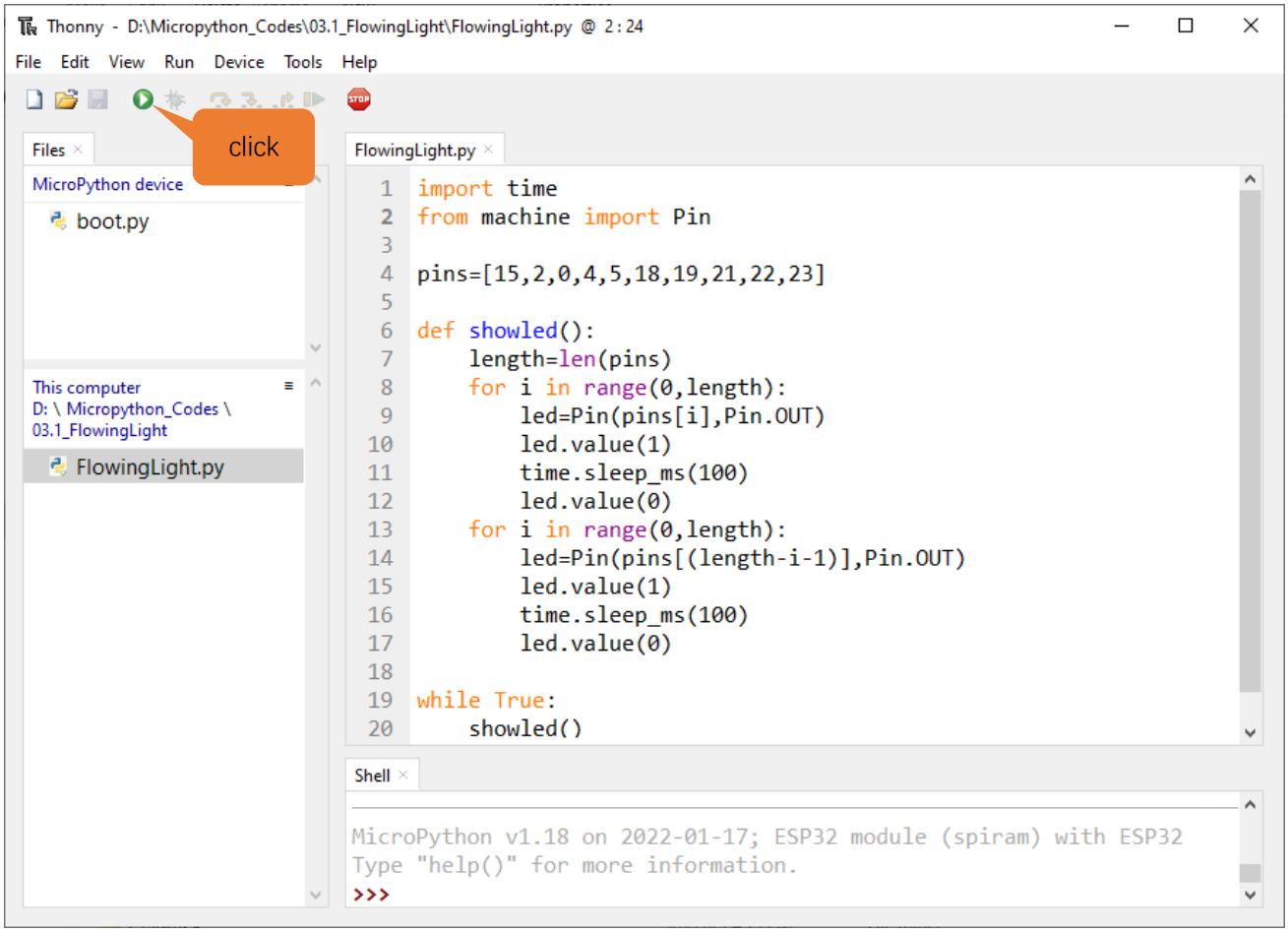
Code

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

03.1_FlowingLight

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/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_FlowingLight" and double click "FlowingLight.py".



Click "Run current script" shown in the box above, LED Bar Graph will light up from left to right and then back from right to left.

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

The following is the program code:

```
1  import time
2  from machine import Pin
3
4  pins=[15,2,0,4,5,18,19,21,22,23]
5
6  def showled():
7      length=len(pins)
8      for i in range(0,length):
9          led=Pin(pins[i],Pin.OUT)
10         led.value(1)
11         time.sleep_ms(100)
12         led.value(0)
13     for i in range(0,length):
14         led=Pin(pins[(length-i-1)],Pin.OUT)
15         led.value(1)
16         time.sleep_ms(100)
17         led.value(0)
18
19 while True:
20     showled()
```

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

```
4  pins=[15,2,0,4,5,18,19,21,22,23]
```

Use len() function to obtain the amount of elements in the list and use a for loop to configure pins as output mode.

```
7      length=len(pins)
8      for i in range(0,length):
9          led=Pin(pins[i],Pin.OUT)
```

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

```
8      for i in range(0,length):
9          led=Pin(pins[i],Pin.OUT)
10         led.value(1)
11         time.sleep_ms(100)
12         led.value(0)
13     for i in range(0,length):
14         led=Pin(pins[(length-i-1)],Pin.OUT)
15         led.value(1)
16         time.sleep_ms(100)
17         led.value(0)
```

Reference

for i in range(start,end,num: int=1)
For loop is used to execute a program endlessly and iterate in the order of items (a list or a string) in the sequence start: The initial value, the for loop starts with it end: The ending value, the for loop end with it num: Num is automatically added each time to the data. The default value is 1

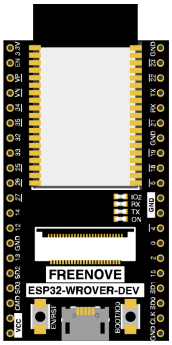
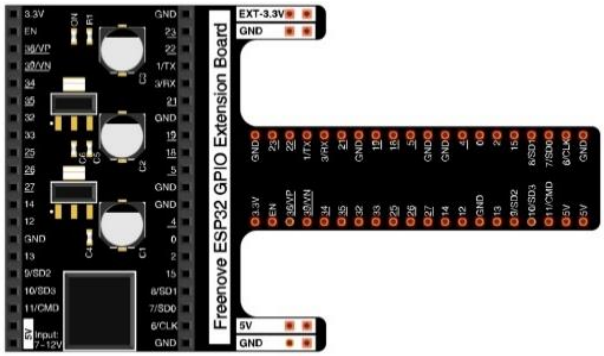
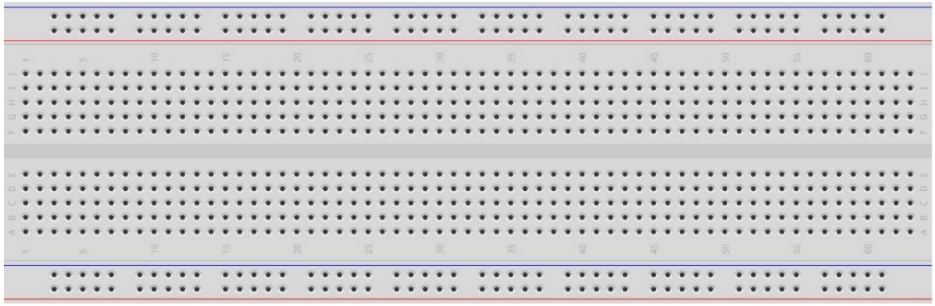
Chapter 4 Analog & PWM

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

Project 4.1 Breathing LED

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

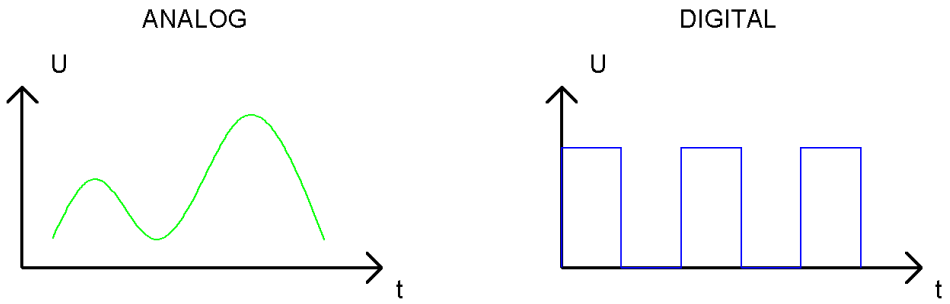
Component List

<table>
<tr><td colspan="2">ESP32-WROVER x1
  </td><td>GPIO Extension Board x1
  </td></tr>
<tr><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>LED x1
 </td><td>Resistor 220Ω x1
 </td><td>Jumper M/M x2
 </td></tr>
</table>

Related knowledge

Analog & Digital

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



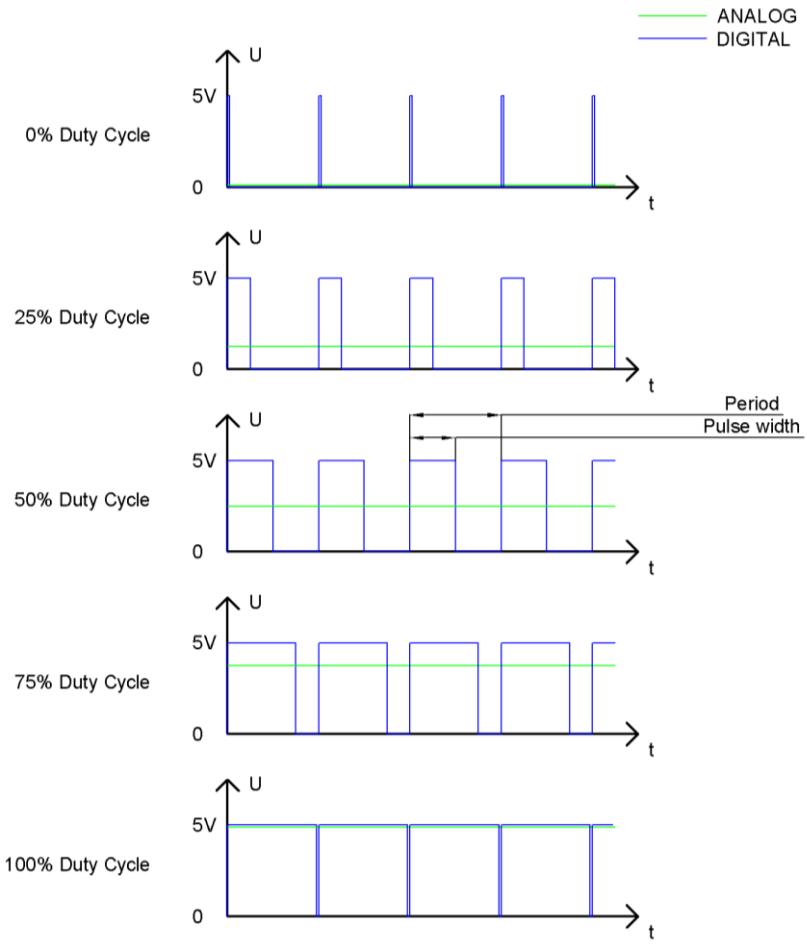
In practical application, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1), it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

PWM

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

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

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



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

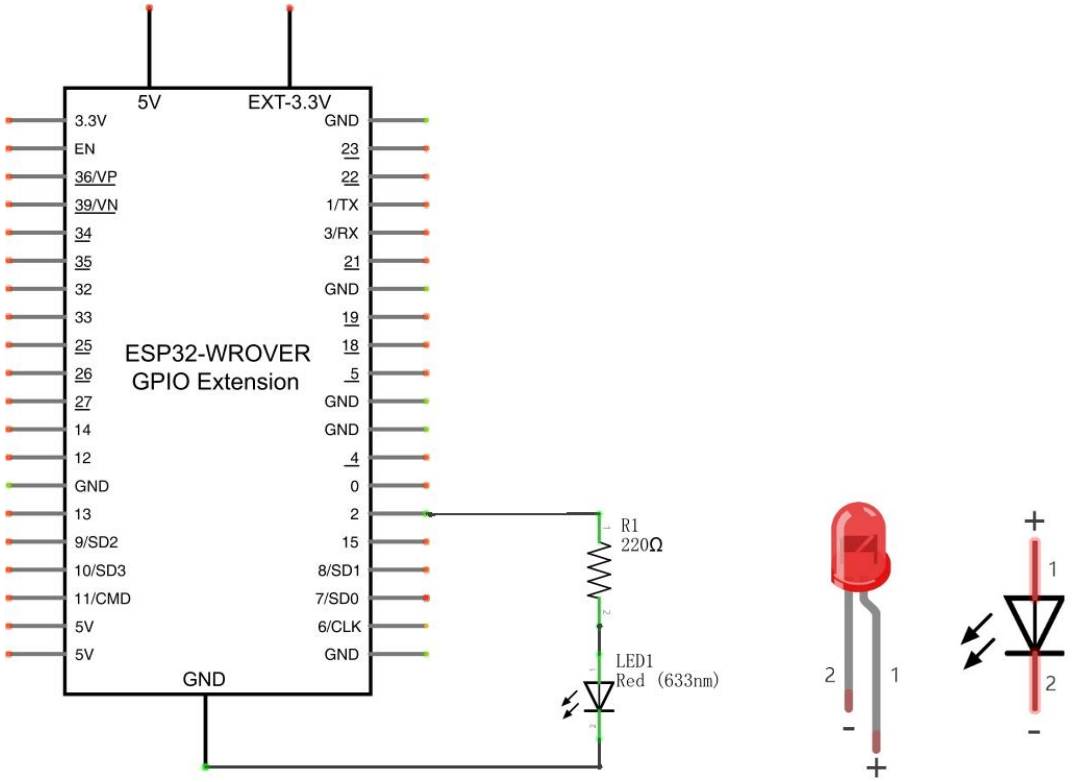
ESP32 and PWM

The ESP32 PWM controller has 8 independent channels, each of which can independently control frequency, duty cycle, and even accuracy. Unlike traditional PWM pins, the PWM output pins of ESP32 are configurable and they can be configured to PWM.

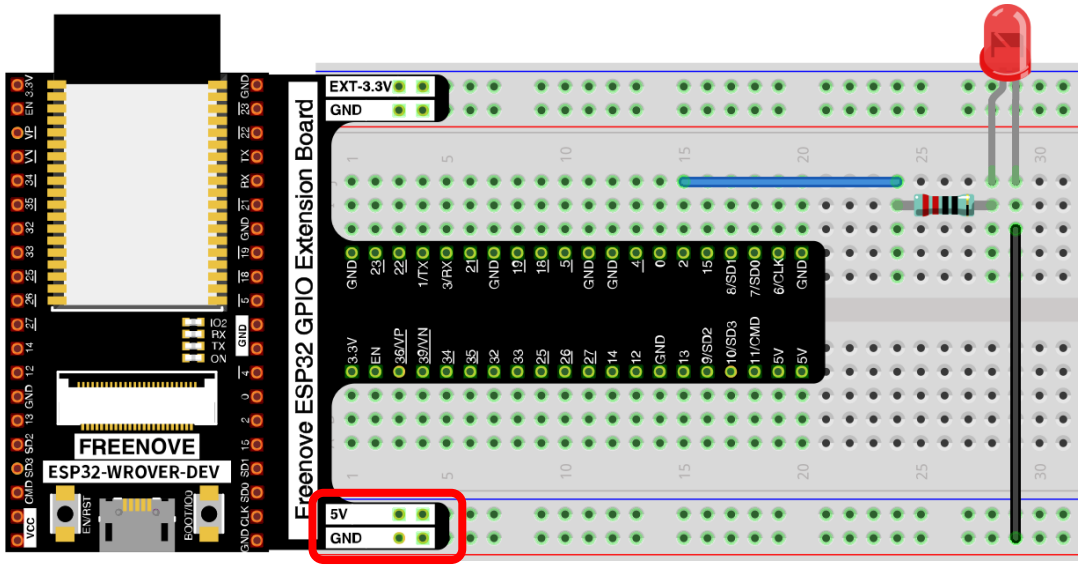
Circuit

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

Schematic diagram



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



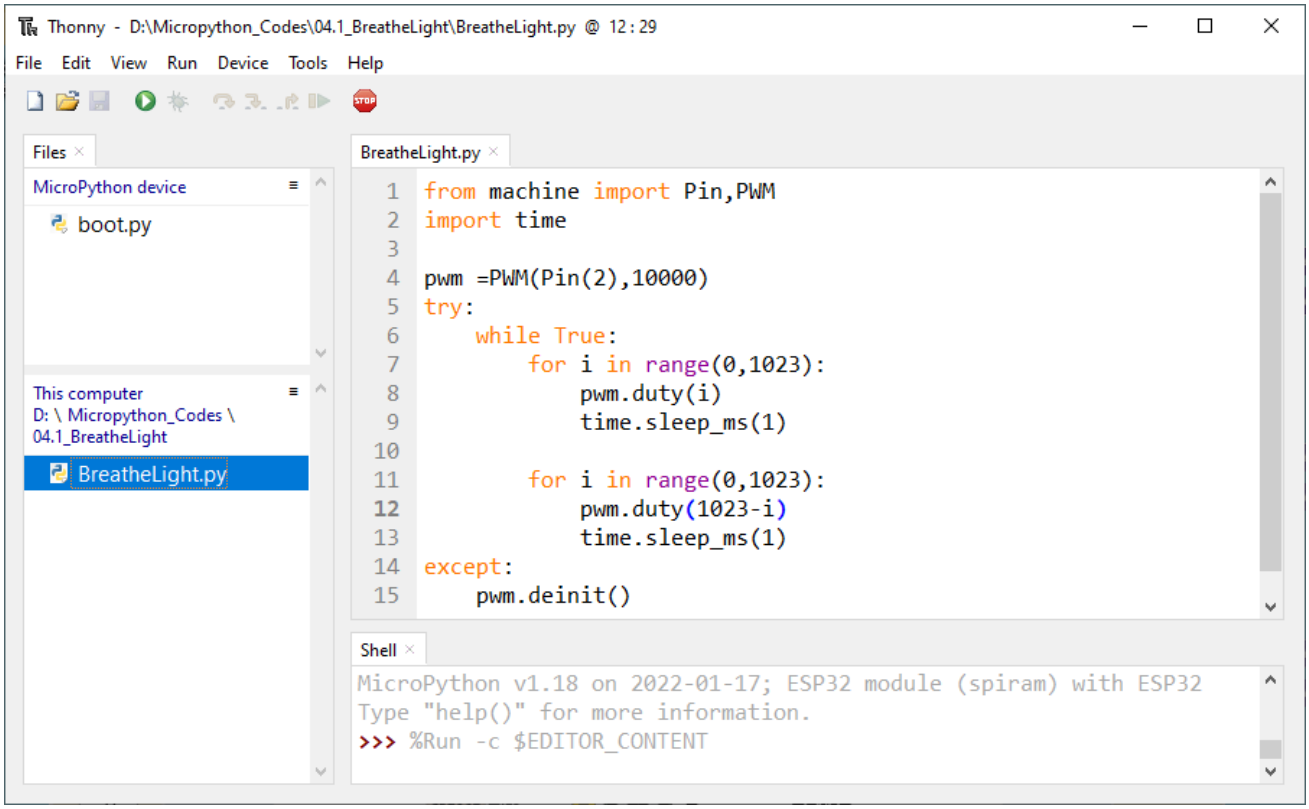
Code

This project is designed to make PWM output GPIO2 with pulse width increasing from 0% to 100%, and then reducing from 100% to 0% gradually.

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/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_BreatheLight" and double click "BreatheLight.py".

04.1_BreatheLight



Click "Run current script", and you'll see that LED is turned from ON to OFF and then back from OFF to ON gradually like breathing.

The following is the program code:

```
1  from machine import Pin,PWM
2  import time
3
4  pwm =PWM(Pin(2,Pin.OUT),10000)
5  try:
6      while True:
7          for i in range(0,1023):
8              pwm.duty(i)
9              time.sleep_ms(1)
10
11         for i in range(0,1023):
12             pwm.duty(1023-i)
13             time.sleep_ms(1)
14 except:
15     pwm.deinit()
```

The way that the ESP32 PWM pins output is different from traditionally controllers. It can change frequency and duty cycle by configuring PWM's parameters at the initialization stage. Define GPIO2's output frequency as 10000Hz, and assign them to PWM.

```
4  pwm =PWM(Pin(2,Pin.OUT),10000)
```

The range of duty cycle is 0-1023, so we use the first for loop to control PWM to change the duty cycle value, making PWM output 0% -100%; Use the second for loop to make PWM output 100%-0%.

```
7          for i in range(0,1023):
8              pwm.duty(i)
9              time.sleep_ms(1)
10
11         for i in range(0,1023):
12             pwm.duty(1023-i)
13             time.sleep_ms(1)
```

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

```
15 pwm.deinit()
```

Reference

Class PWM(pin, freq)

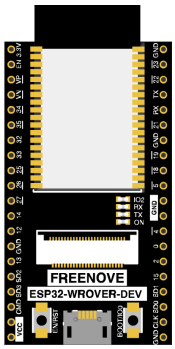
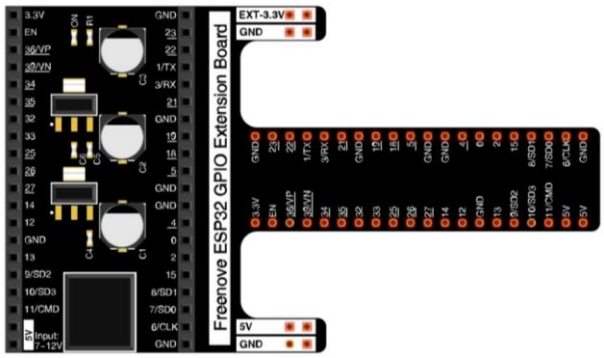
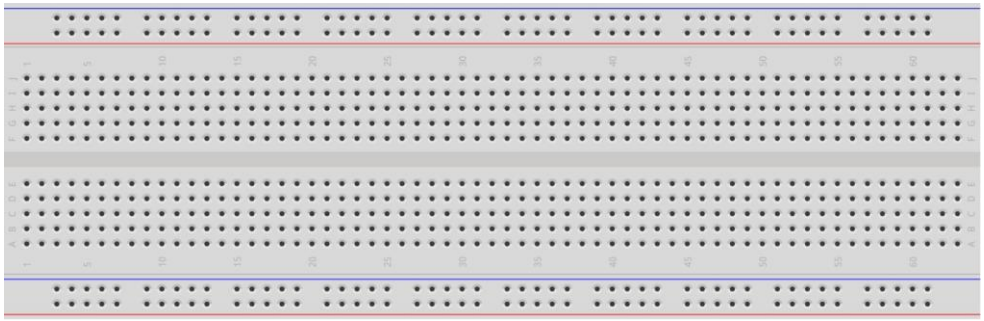
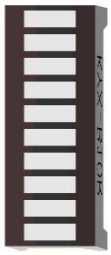
Before each use of PWM module, please add the statement "**from machine import PWM**" to the top of the python file.

pin: PWM pins are supported, such as Pin(0)、Pin(2)、Pin(4)、Pin(5)、Pin(10)、Pin(12~19)、Pin(21)、Pin(22)、Pin(23)、Pin(25~27).
freq: Output frequency, with the range of 0-78125 Hz
duty: Duty cycle, with the range of 0-1023.
PWM.init(freq, duty): Initialize PWM, parameters are the same as above.
PWM.freq([freq_val]): When there is no parameter, the function obtains and returns PWM frequency; When parameters are set, the function is used to set PWM frequency and returns nothing.
PWM.duty([duty_val]): When there is no parameter, the function obtains and returns PWM duty cycle; When parameters are set, the function is used to set PWM duty cycle.
PWM.deinit(): Turn OFF PWM.

Project 4.2 Meteor Flowing Light

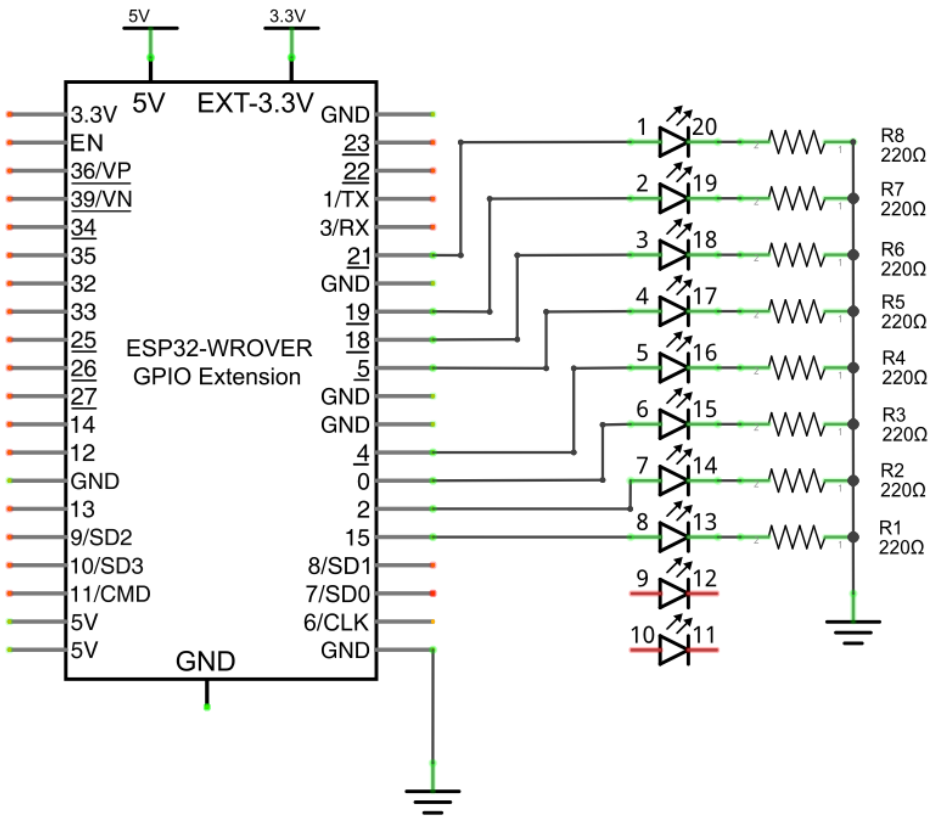
After learning about PWM, we can use it to control LED Bar Graph and realize a cooler Flowing Light.

Component List

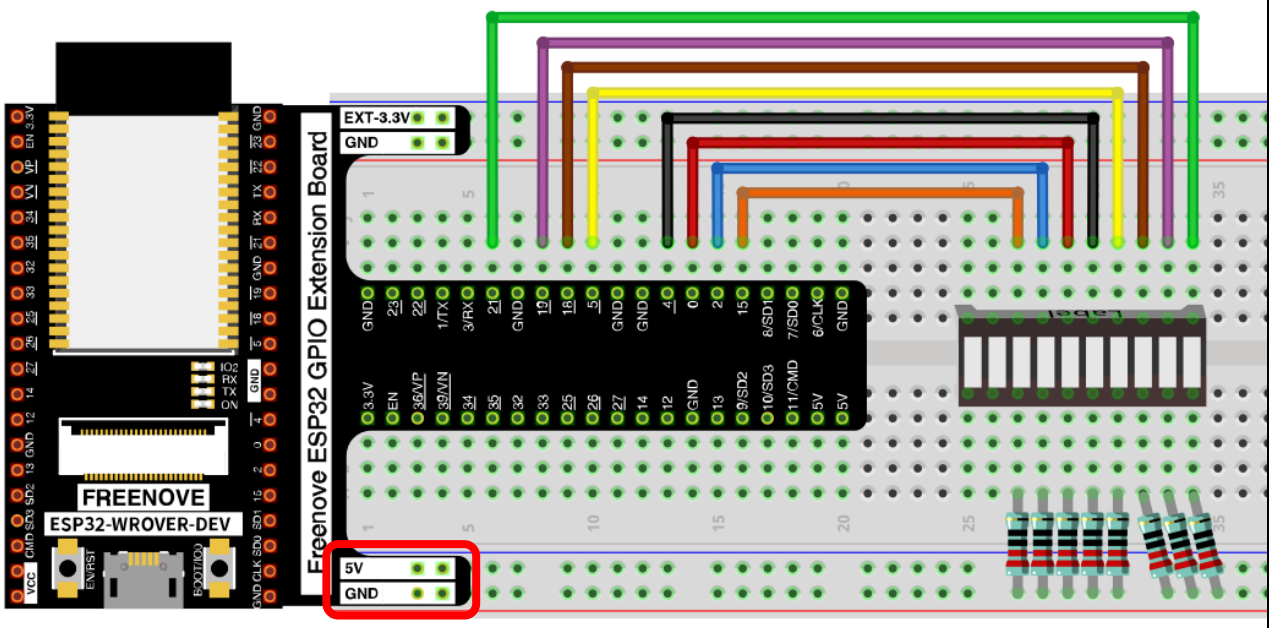
<table>
<tr><td>ESP32-WROVER x1
  </td><td colspan="2">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>Jumper M/M x10
 </td><td>LED bar graph x1
  </td><td>Resistor 220Ω x10
 </td></tr>
</table>

Circuit

Schematic diagram



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



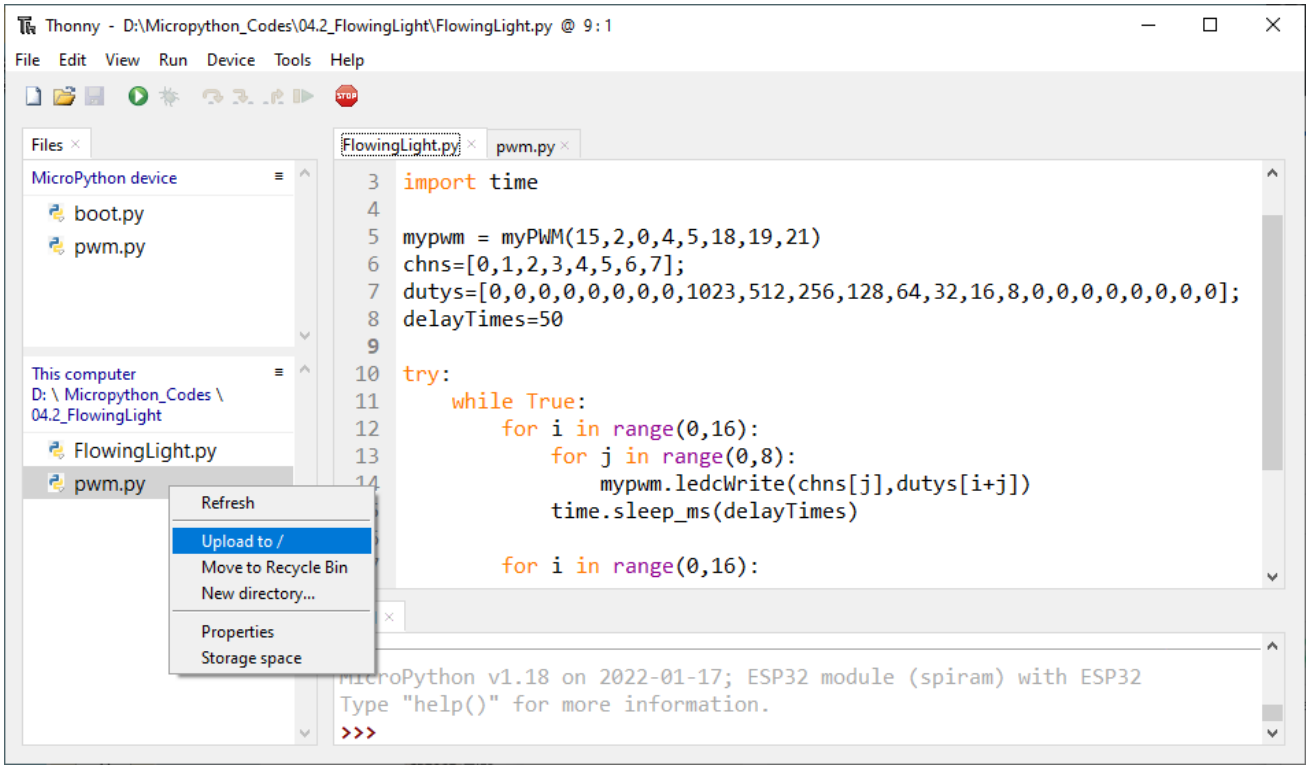
If LEDbar doesn't work, try to rotate LEDbar for 180°. The label is random.

Code

Flowing Light with tail was implemented with PWM.

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "04.2_FlowingLight". Select "pwm.py", right click to select "Upload to /", wait for "pwm.py" to be uploaded to ESP32-WROVER and then double click "FlowingLight.py"

04.2_FlowingLight



Click "Run current script", and LED Bar Graph will gradually light up and out from left to right, then light up and out from right to left.

The following is the program code:

```
1  from machine import Pin,PWM
2  from pwm import myPWM
3  import time
4
5  mypwm = myPWM(15,2,0,4,5,18,19,21)
6  chns=[0,1,2,3,4,5,6,7];
7  dutys=[0,0,0,0,0,0,0,0,1023,512,256,128,64,32,16,8,0,0,0,0,0,0,0,0];
8  delayTimes=50
9
10 try:
11     while True:
12         for i in range(0,16):
13             for j in range(0,8):
14                 mypwm.ledcWrite(chns[j],dutys[i+j])
15             time.sleep_ms(delayTimes)
16
17         for i in range(0,16):
18             for j in range(0,8):
19                 mypwm.ledcWrite(chns[7-j],dutys[i+j])
20             time.sleep_ms(delayTimes)
21 except:
22     mypwm.deinit()
```

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

```
2  from pwm import myPWM
…  …
5  mypwm = myPWM(15,2,0,4,5,18,19,21)
```

First we defined 8 GPIO, 8 PWM channels, and 24 pulse width values.

```
5  mypwm = myPWM(15,2,0,4,5,18,19,21)
6  chns=[0,1,2,3,4,5,6,7];
7  dutys=[0,0,0,0,0,0,0,0,1023,512,256,128,64,32,16,8,0,0,0,0,0,0,0,0];
```

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

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

Close the PWM of the object myPWM.

```
14 mypwm.deinit()
```

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

```
12          for i in range(0,16):
13              for j in range(0,8):
14                  mypwm.ledcWrite(chns[j],dutys[i+j])
15              time.sleep_ms(delayTimes)
16
17          for i in range(0,16):
18              for j in range(0,8):
19                  mypwm.ledcWrite(chns[7-j],dutys[i+j])
20              time.sleep_ms(delayTimes)
```

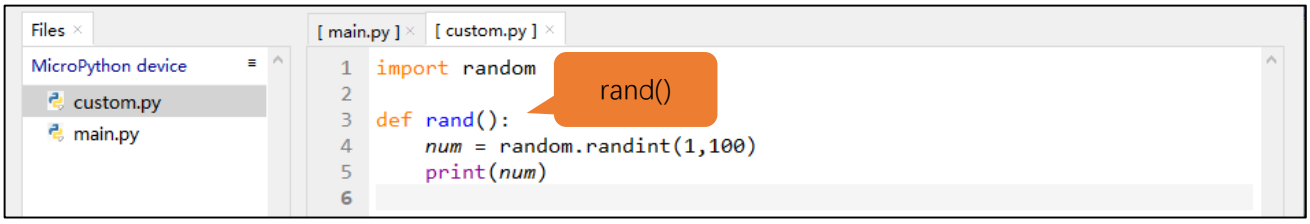
In the main function, a nested for loop is used to control the pulse width of the PWM. Every time i in the first for loop increases by 1, the LED Bar Graph will move one grid, and gradually change according to the value in the array dutys. As shown in the following table, the value in the second row is the value of the array dutys, and the 8 green grids in each row below represent the 8 LEDs on the LED Bar Graph. Each time i increases by 1, the value of the LED Bar Graph will move to the right by one grid, and when it reaches the end, it will move from the end to the starting point, achieving the desired effect.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
d i	0	0	0	0	0	0	0		1023	512	256	128	64	32	16	8	0	0	0	0	0	0	0	0
0																								
1																								
...																								
14																								
15																								
16																								

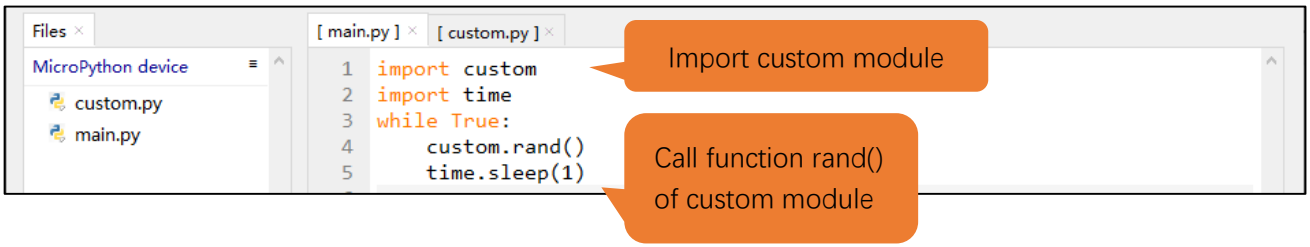
How to import a custom python module

Each Python file, as long as it's stored on the file system of ESP32, is a module. To import a custom module, the module file needs to be located in the MicroPython environment variable path or in the same path as the currently running program.

First, customize a python module "custom.py". Create a new py file and name it "custom.py". Write code to it and save it to ESP32.



Second, import custom module “custom” to main.py



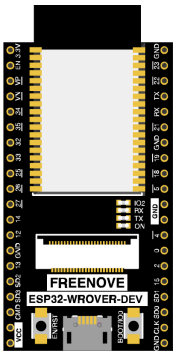
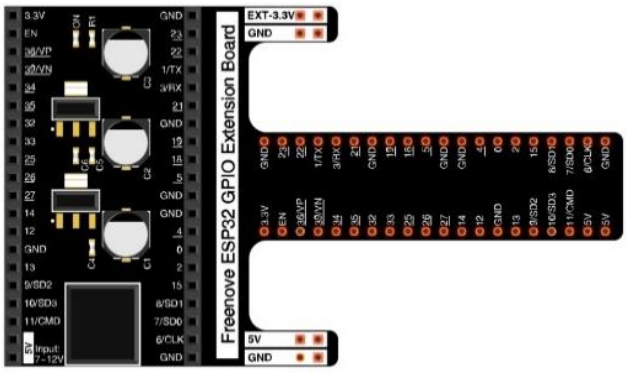
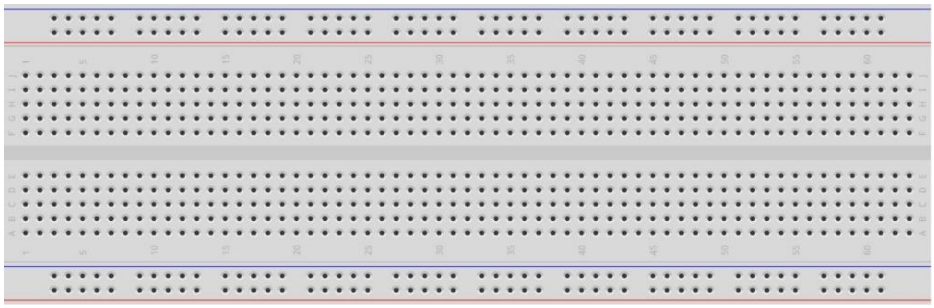
Chapter 5 RGBLED

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

Project 5.1 Random Color Light

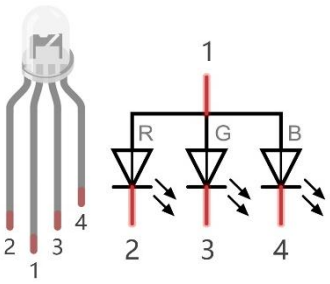
In this project, we will make a multicolored LED. And we can control RGBLED to switch different colors automatically.

Component List

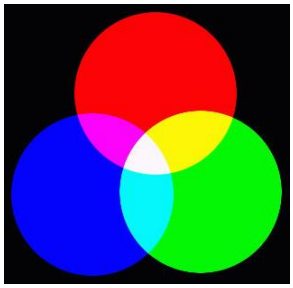
<table>
<tr><td>ESP32-WROVER x1
  </td><td colspan="2">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="3">Breadboard x1
  </td></tr>
<tr><td>RGBLED x1
 </td><td>Resistor 220Ω x3
 </td><td>Jumper M/M x4
 </td></tr>
</table>

Related knowledge

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



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

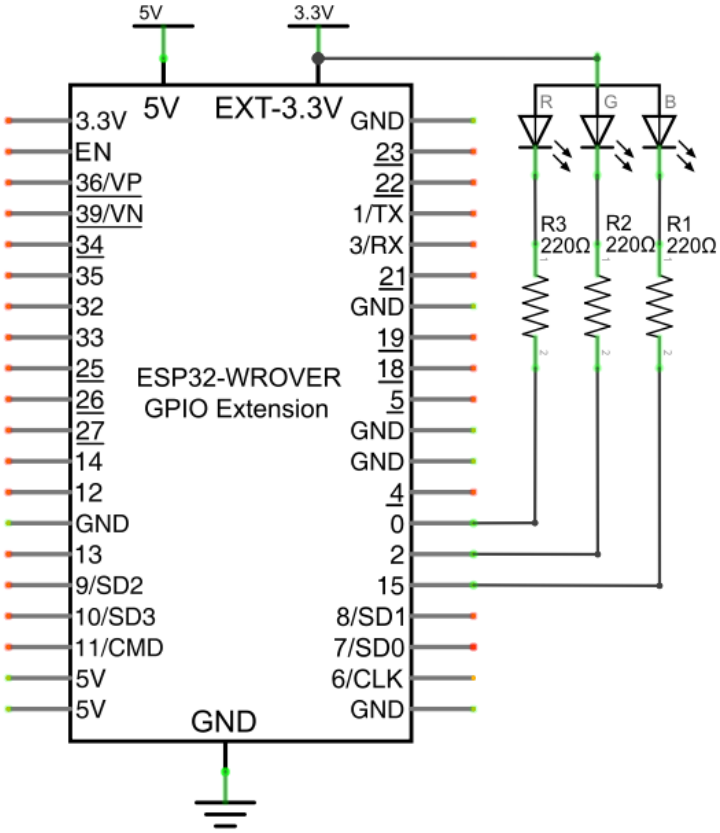


RGB

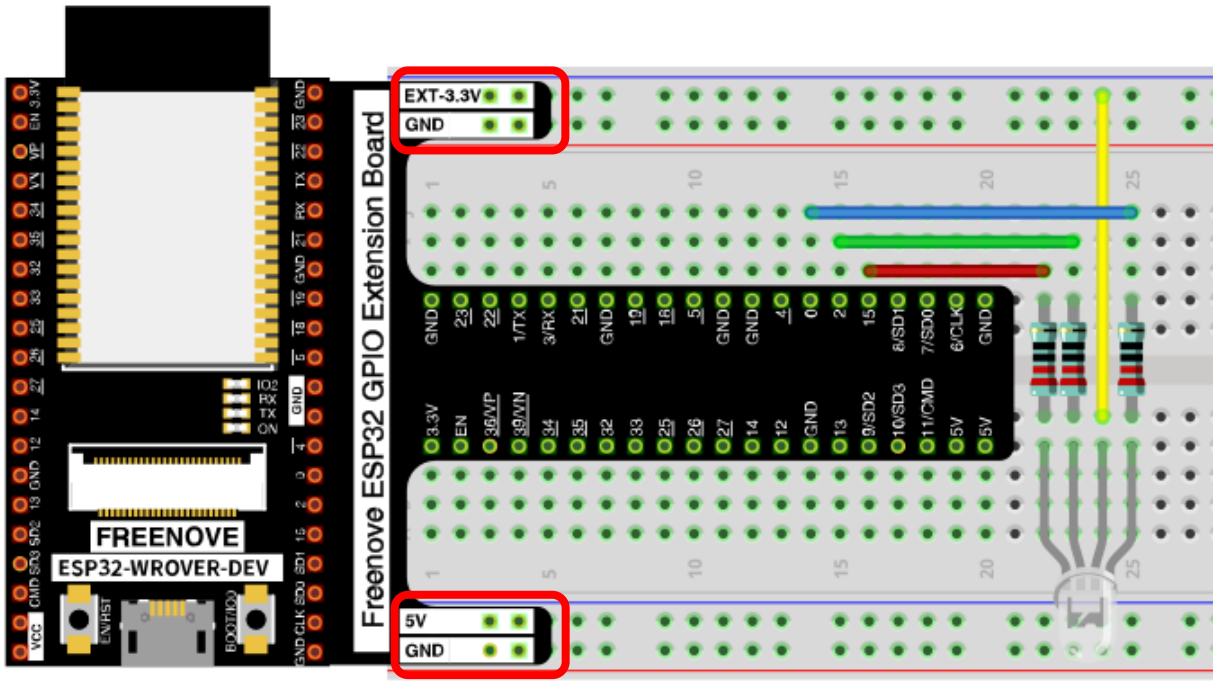
If we use three 10-bit PWM to control the RGBLED, in theory, we can create $2^{10}*2^{10}*2^{10}$= 1,073,741,824(1 billion) colors through different combinations.

Circuit

Schematic diagram



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



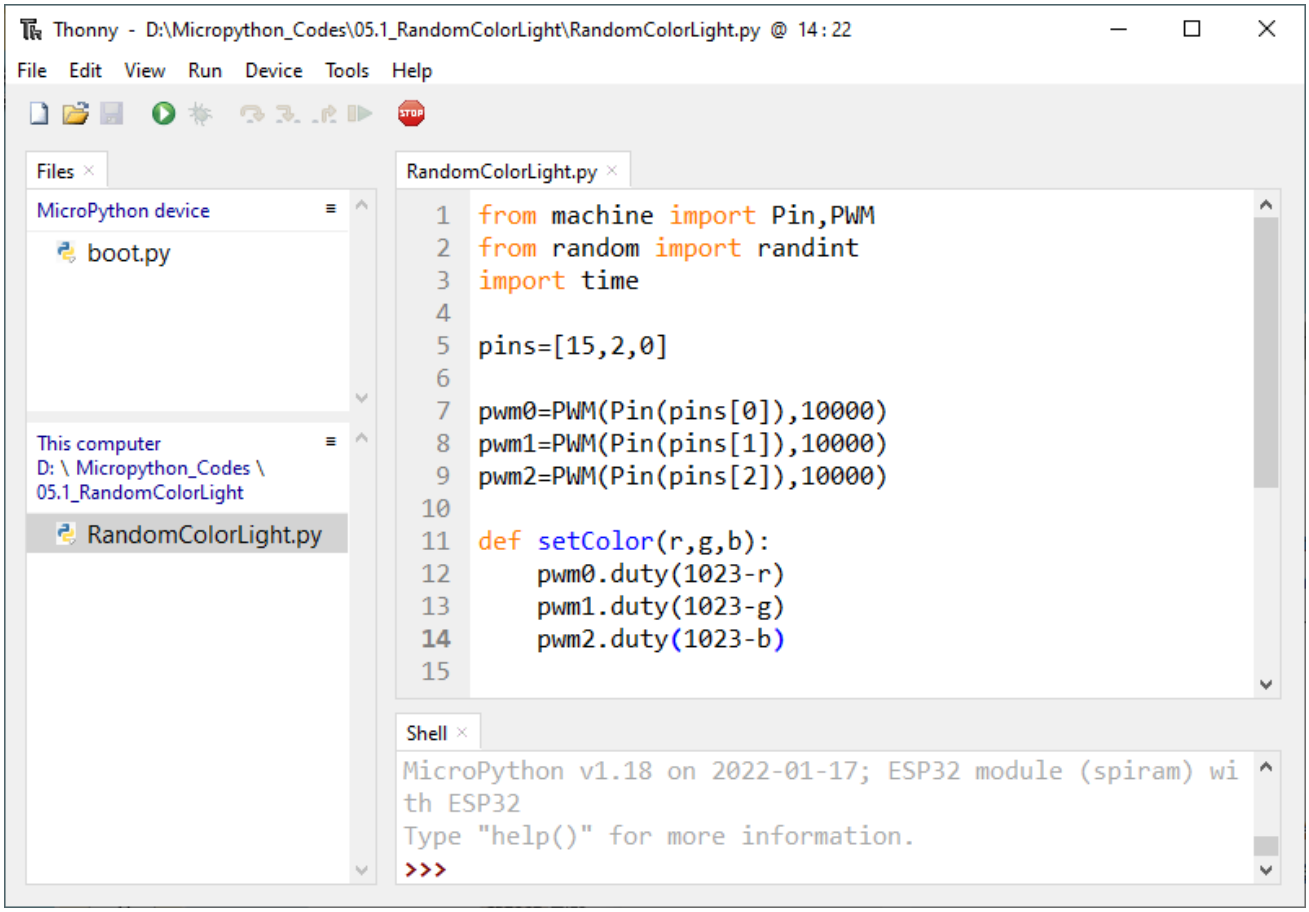
Code

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

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

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

05.1_RandomColorLight



Click "Run current script", RGBLED begins to display random colors.

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

The following is the program code:

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

pins=[15,2,0]

pwm0=PWM(Pin(pins[0]),10000)
pwm1=PWM(Pin(pins[1]),10000)
pwm2=PWM(Pin(pins[2]),10000)

def setColor(r,g,b):
    pwm0.duty(1023-r)
    pwm1.duty(1023-g)
    pwm2.duty(1023-b)

try:
    while True:
        red   = randint(0,1023)
        green = randint(0,1023)
        blue  = randint(0,1023)
        setColor(red,green,blue)
        time.sleep_ms(200)
except:
    pwm0.deinit()
    pwm1.deinit()
    pwm2.deinit()
```

Import Pin, PWM and Randon Function modules.

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

Configure ouput mode of GPIO15, GPIO2 and GPIO0 as PWM output and PWM frequency as 10000Hz

```
pins=[15,2,0]

pwm0=PWM(Pin(pins[0]),10000)
pwm1=PWM(Pin(pins[1]),10000)
pwm2=PWM(Pin(pins[2]),10000)
```

Define a function to set the color of RGBLED.

```
def setColor(r,g,b):
    pwm0.duty(1023-r)
    pwm1.duty(1023-g)
    pwm2.duty(1023-b)
```

Call random function randint()to generate a random number in the range of 0-1023 and assign the value to red.

```
18  red   = randint(0,1023)
```

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

```
17      while True:
18          red   = randint(0,1023)
19          green = randint(0,1023)
20          blue  = randint(0,1023)
21          setColor(red,green,blue)
22          time.sleep_ms(200)
```

Reference

Class random

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

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

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

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

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

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

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

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

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

For example:

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

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

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

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

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

step: An integer specifying the incrementation.

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

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

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

obj: list of elements

Project 5.2 Gradient Color Light

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

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

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

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

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

05.2_GradientColorLight

The following is the program code:

```
from machine import Pin,PWM
import time

pins=[15,2,0];

pwm0=PWM(Pin(pins[0]),1000)
pwm1=PWM(Pin(pins[1]),1000)
pwm2=PWM(Pin(pins[2]),1000)

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

def setColor():
    pwm0.duty(red)
    pwm1.duty(green)
    pwm2.duty(blue)

def wheel(pos):
    global red,green,blue
    WheelPos=pos%1023
    print(WheelPos)
    if WheelPos<341:
        red=1023-WheelPos*3
        green=WheelPos*3
        blue=0
```

```
27
28      elif WheelPos>=341 and WheelPos<682:
29          WheelPos -= 341;
30          red=0
31          green=1023-WheelPos*3
32          blue=WheelPos*3
33      else :
34          WheelPos -= 682;
35          red=WheelPos*3
36          green=0
37          blue=1023-WheelPos*3
38
39  try:
40      while True:
41          for i in range(0,1023):
42              wheel(i)
43              setColor()
44              time.sleep_ms(15)
45  except:
46      pwm0.deinit()
47      pwm1.deinit()
48      pwm2.deinit()
```

The function wheel() is a color selection method of the color model introduced earlier. The value range of the parameter pos is 0-1023. The function will return a data containing the duty cycle values of 3 pins.

```
19  def wheel(pos):
20      global red,green,blue
21      WheelPos=pos%1023
22      print(WheelPos)
23      if WheelPos<341:
24          red=1023-WheelPos*3
25          green=WheelPos*3
26          blue=0
27
28      elif WheelPos>=341 and WheelPos<682:
29          WheelPos -= 341;
30          red=0
31          green=1023-WheelPos*3
32          blue=WheelPos*3
33      else :
34          WheelPos -= 682;
35          red=WheelPos*3
36          green=0
37          blue=1023-WheelPos*3
```

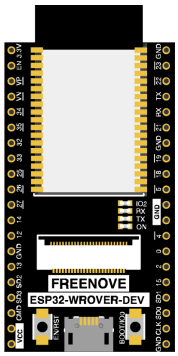
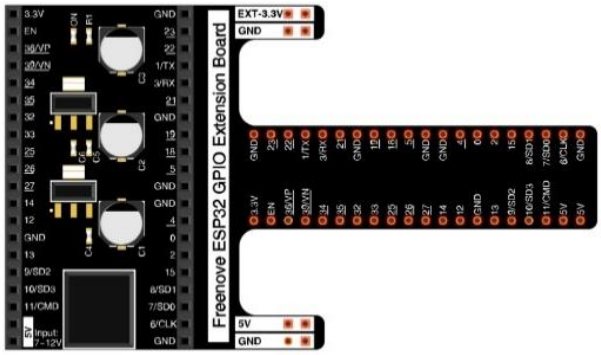
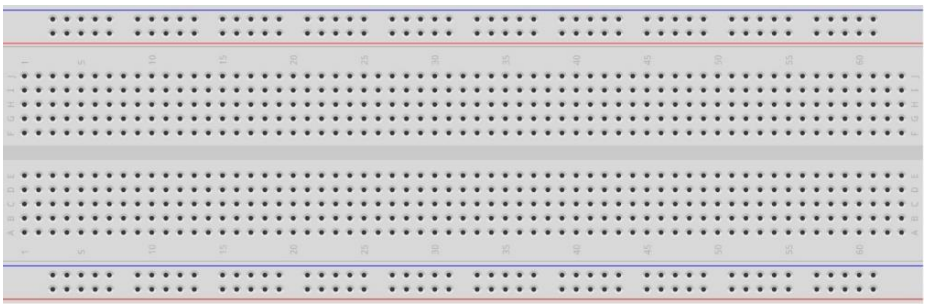
Chapter 6 Buzzer

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

Project 6.1 Doorbell

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

Component List

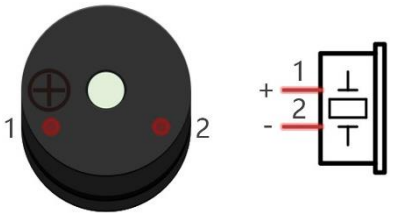
<table>
<tr><td colspan="3">ESP32-WROVER x1
  </td><td colspan="2">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="5">Breadboard x1
  </td></tr>
<tr><td colspan="5">Jumper M/M x6
 </td></tr>
<tr><td>NPN transistorx1
(S8050)
 </td><td>Active buzzer x1
 </td><td>Push button x1
 </td><td>Resistor 1kΩ x1
 </td><td>Resistor 10kΩ x2
 </td></tr>
</table>

Component knowledge

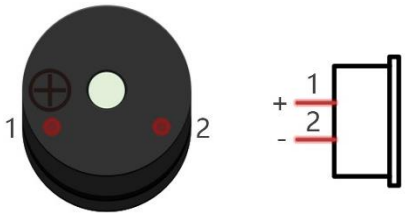
Buzzer

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

Active buzzer



Passive buzzer



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

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

How to identify active and passive buzzer?

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

Active buzzer

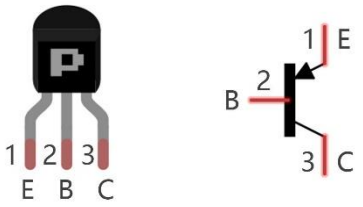
Passive buzzer

Transistor

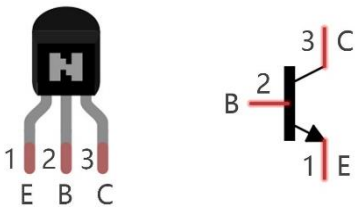
Because the buzzer requires such large current that GPIO of ESP32 output capability cannot meet the requirement, a transistor of NPN type is needed here to amplify the current.

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

PNP transistor



NPN transistor

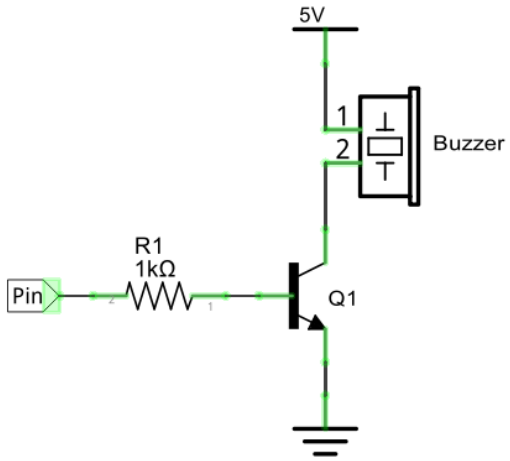
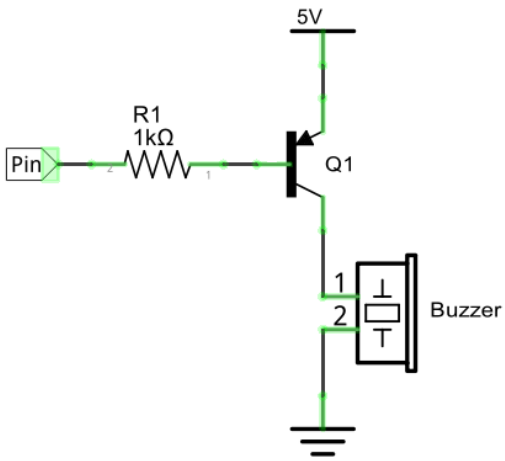


In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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

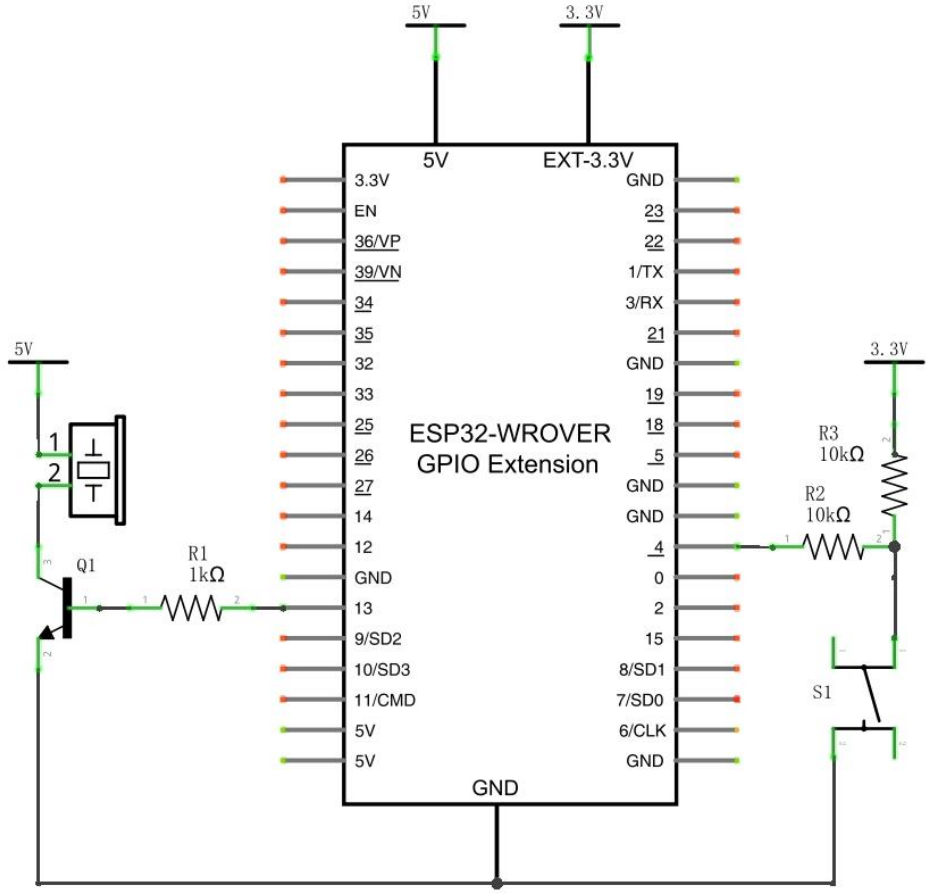
When using NPN transistor to drive buzzer, we often adopt the following method. If GPIO outputs high level, current will flow through R1, the transistor will get conducted, and the buzzer will sound. If GPIO outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer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

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

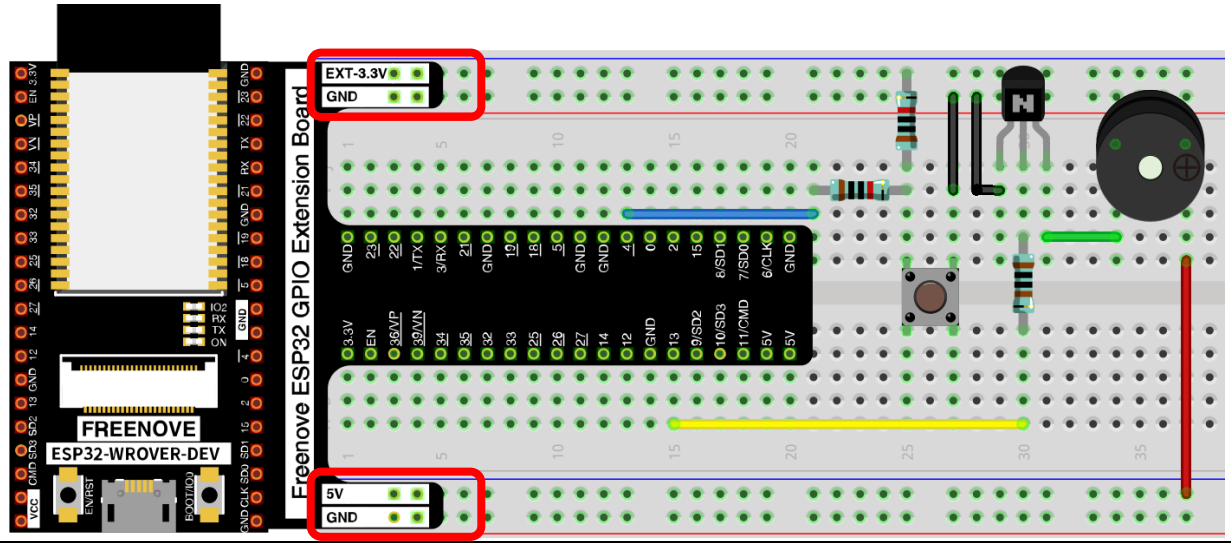
NPN transistor to drive buzzer	PNP transistor to drive buzzer
	

Circuit

Schematic diagram



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



Note: in this circuit, the power supply for buzzer is 5V, and pull-up resistor of the button connected to the power 3.3V. The buzzer can work when connected to power 3.3V, but it will reduce the loudness.

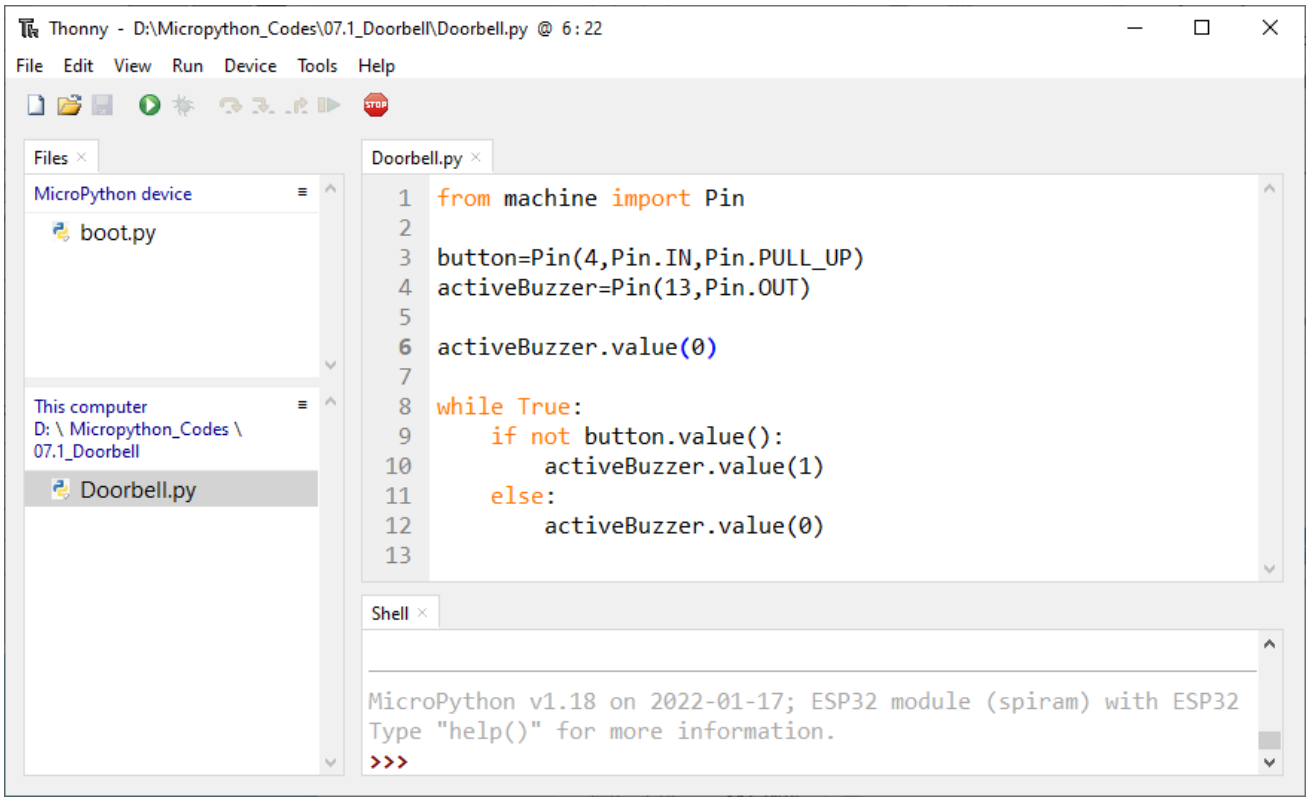
Code

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

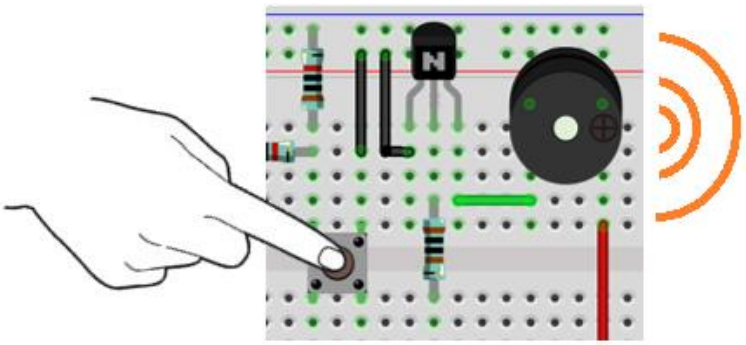
Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "06.1_Doorbell" and double click "Doorbell.py".

06.1_Doorbell



Click "Run current script", press the push button switch and the buzzer will sound. Release the push button switch and the buzzer will stop.



The following is the program code:

```
from machine import Pin

button=Pin(4,Pin.IN,Pin.PULL_UP)
activeBuzzer=Pin(13,Pin.OUT)

activeBuzzer.value(0)

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

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

Project 6.2 Alertor

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

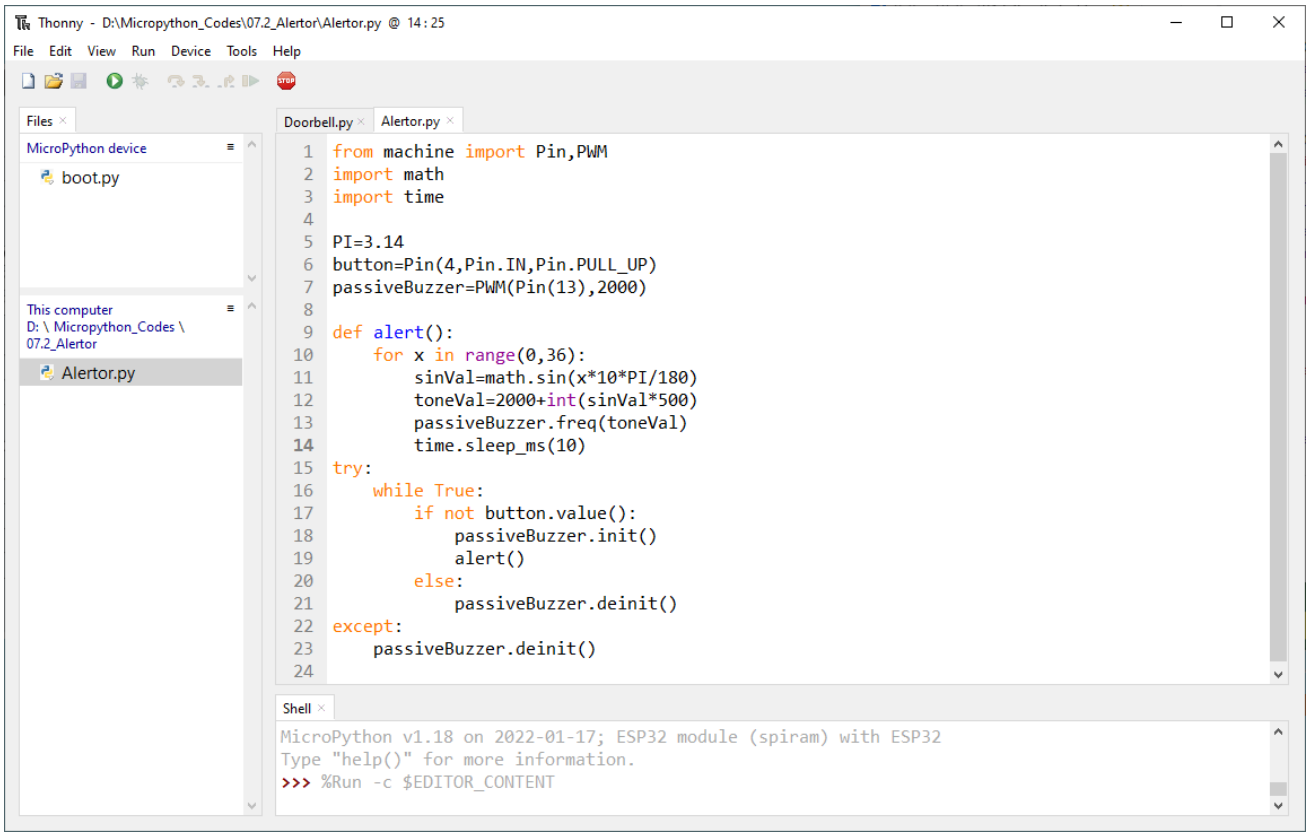
Component list and the circuit part is similar to last section. In the Doorbell circuit only the **active buzzer** needs to be **replaced** with a **passive buzzer.**

Code

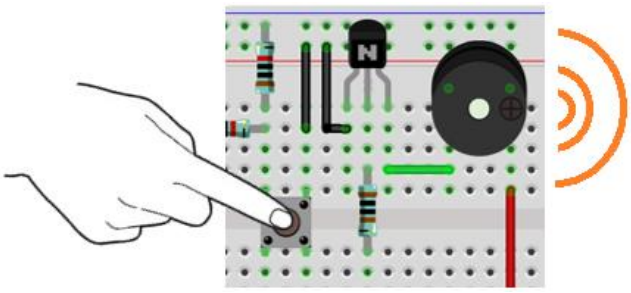
In this project, the buzzer alarm is controlled by the button. Press the button, then buzzer sounds. If you release the button, the buzzer will stop sounding. In the logic, it is the same as using button to control LED. In the control method, passive buzzer requires PWM of certain frequency to sound.

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "06.2_Alertor", and double click "Alertor.py".

06.2_Alertor



Click "Run current script", press the button, then alarm sounds. And when the button is release, the alarm will stop sounding.



The following is the program code:

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

PI=3.14
button=Pin(4,Pin.IN,Pin.PULL_UP)
passiveBuzzer=PWM(Pin(13),2000)

def alert():
    for x in range(0,36):
        sinVal=math.sin(x*10*PI/180)
        toneVal=2000+int(sinVal*500)
        passiveBuzzer.freq(toneVal)
        time.sleep_ms(10)
try:
    while True:
        if not button.value():
            passiveBuzzer.init()
            alert()
        else:
            passiveBuzzer.deinit()
except:
    passiveBuzzer.deinit()
```

Import PWM, Pin, math and time modules.

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

Define the pins of the button and passive buzzer.

```
PI=3.14
button=Pin(4,Pin.IN,Pin.PULL_UP)
passiveBuzzer=PWM(Pin(13),2000,512)
```

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

```
def alert():
    for x in range(0,36):
        sinVal=math.sin(x*10*PI/180)
        toneVal=2000+int(sinVal*500)
        passiveBuzzer.freq(toneVal)
        time.sleep_ms(10)
```

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

```
passiveBuzzer.deinit()
```

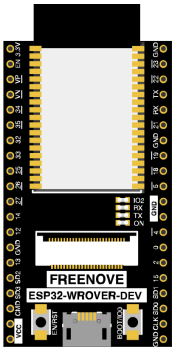
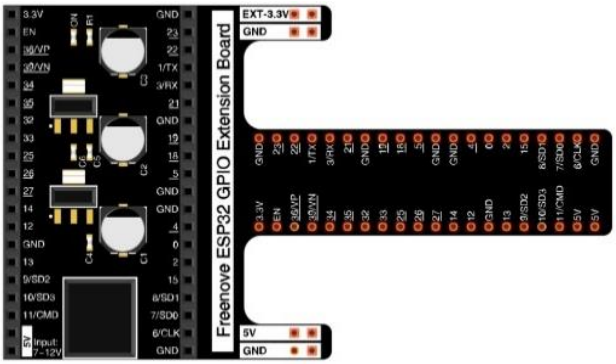
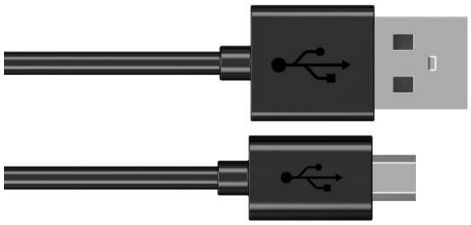
Chapter 7 Serial Communication

Serial Communication is a means of Communication between different devices/devices. This section describes ESP32's Serial Communication.

Project 7.1 Serial Print

This project uses ESP32's serial communicator to send data to the computer and print it on the serial monitor.

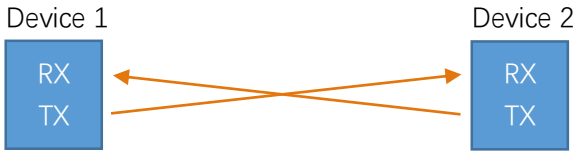
Component List

ESP32-WROVER x1 	GPIO Extension Board x1 
Micro USB Wire x1 	

Related knowledge

Serial communication

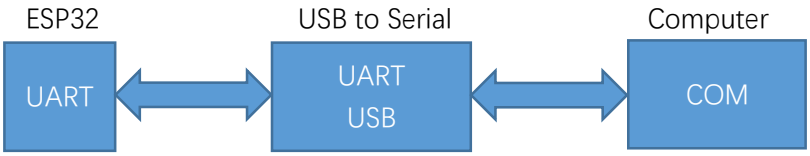
Serial communication generally refers to the Universal Asynchronous Receiver/Transmitter (UART), which is commonly used in electronic circuit communication. It has two communication lines, one is responsible for sending data (TX line) and the other for receiving data (RX line). The serial communication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



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

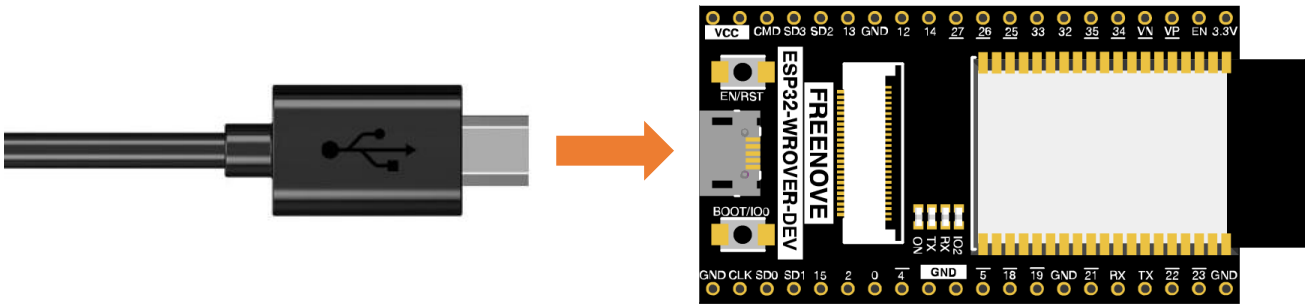
Serial port on ESP32

Freenove ESP32 has integrated USB to serial transfer, so it could communicate with computer connecting to USB cable.



Circuit

Connect Freenove ESP32 to the computer with USB cable

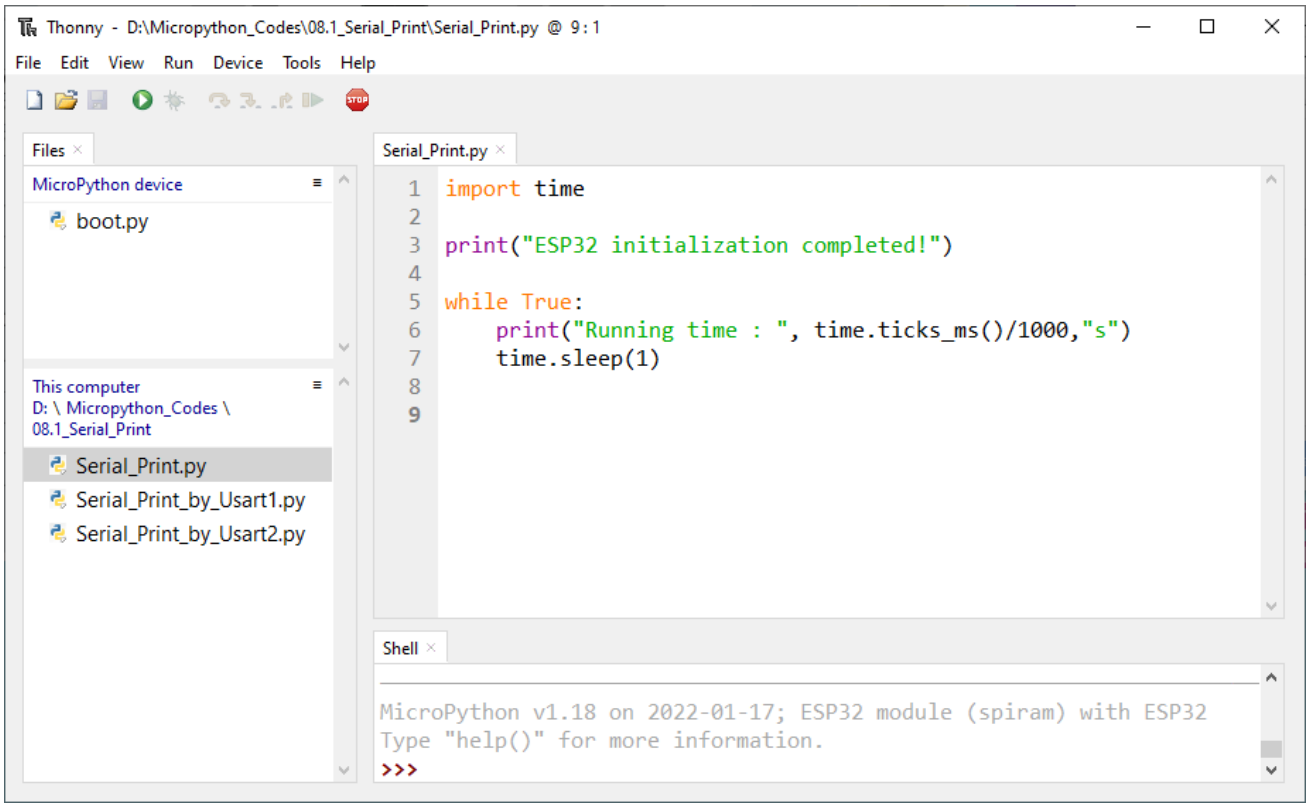


Code

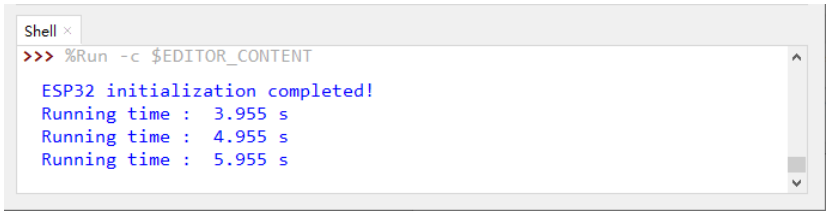
Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

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

07.1_Serial_Print



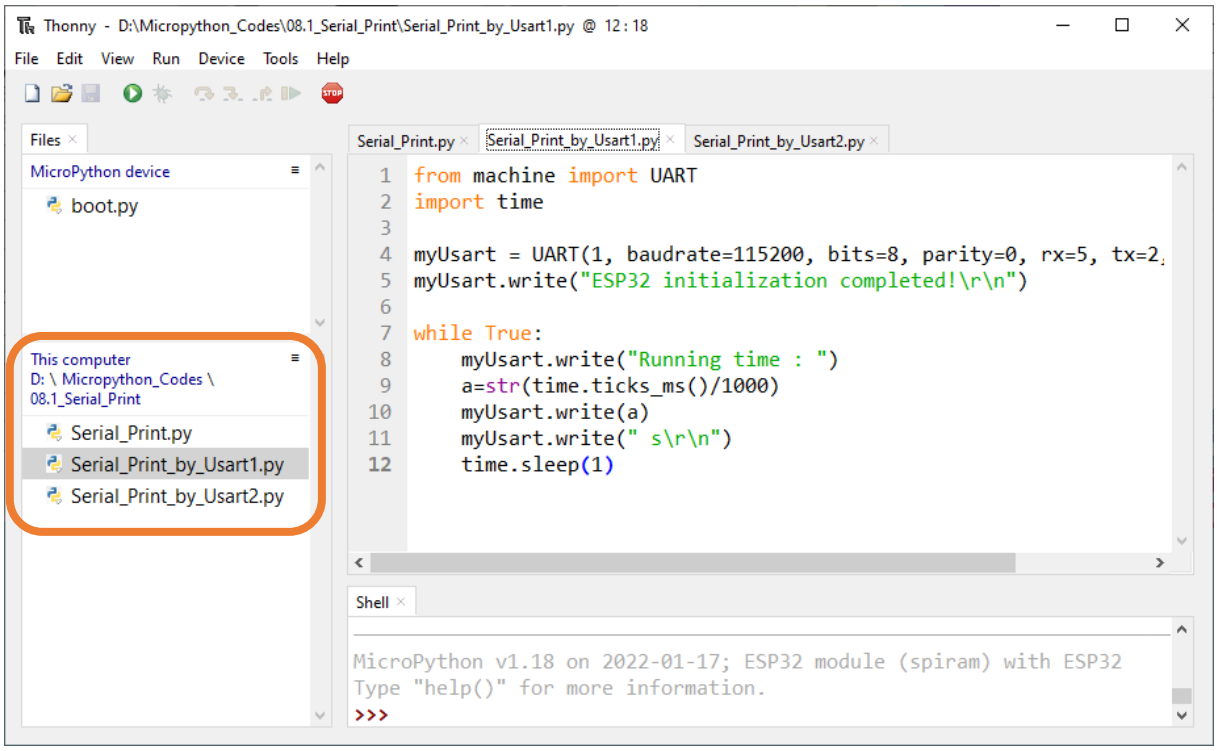
Click "Run current script" and observe the changes of "Shell", which will display the time when ESP32 is powered on once per second.



The following is the program code:

```
1  import time
2
3  print("ESP32 initialization completed!")
4
5  while True:
6      print("Running time : ", time.ticks_ms()/1000, "s")
7      time.sleep(1)
```

ESP32-WROVER has 3 serial ports, one of which is used as REPL, that is, Pin(1) and Pin(3) are occupied, and generally it is not recommended to be used as tx, rx. The other two serial ports can be configured simply by calling the UART module.



Reference

Class UART

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

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

id: Serial Number. The available serial port number is 1 or 2

baudrate: Baud rate

bits: The number of each character.

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

rx, tx: UAPT's reading and writing pins

Pin(0)、Pin(2)、Pin(4)、Pin(5)、Pin(9)、Pin(10)、Pin(12~19)、Pin(21~23)、Pin(25)、Pin(26)、Pin(34~36)、Pin(39)

Note: Pin(1) and Pin(3) are occupied and not recommend to be used as tx,rx.

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

timeout: timeout period (Unit: millisecond)

0 < timeout ≤ 0x7FFF FFFF (decimal: 0 < timeout ≤ 2147483647)

UART.init(baudrate, bits, parity, stop, tx, rx, rts, cts)): Initialize serial ports

tx: writing pins of uart

rx: reading pins of uart

rts: rts pins of uart

cts: cts pins of uart

UART.read(nbytes): Read nbytes bytes
UART.read(): Read data
UART.write(buf): Write byte buffer to UART bus
UART.readline(): Read a line of data, ending with a newline character.
UART.readinto(buf): Read and write data into buffer.
UART.readinto(buf, nbytes): Read and write data into buffer.
UART.any(): Determine whether there is data in serial ports. If yes, return the number of bytes; Otherwise, return 0.

Project 7.2 Serial Read and Write

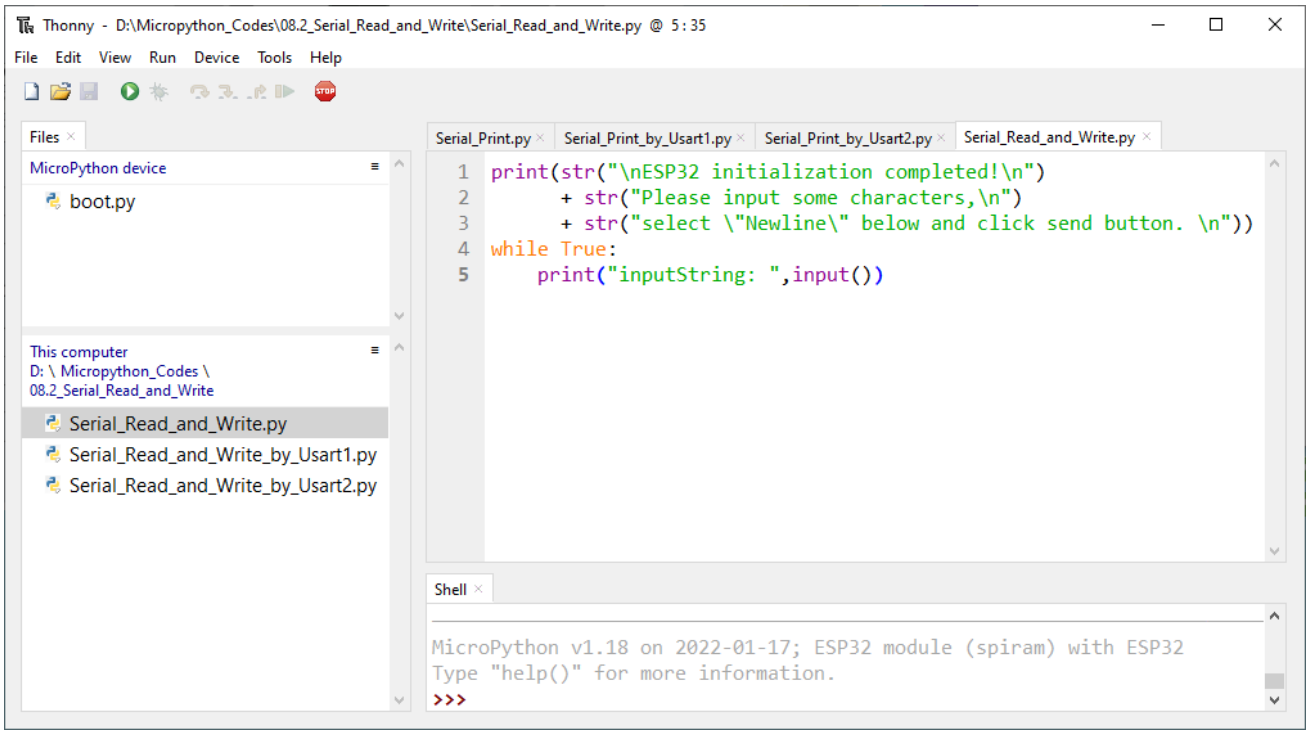
From last section, we use Serial port on Freenove ESP32 to send data to a computer, now we will use that to receive data from computer.

Component and Circuit are the same as in the previous project.

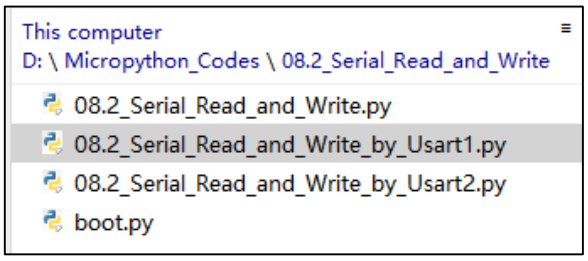
Code

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "07.2_Serial_Read_and_Write" and double click "Serial_Read_and_Write.py".

07.2_Serial_Read_and_Write



Click "Run current script" and ESP32 will print out data at "Shell" and wait for users to enter any messages. Press Enter to end the input, and "Shell" will print out data that the user entered. If you want to use other serial ports, you can use other python files in the same directory.



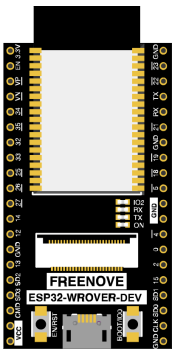
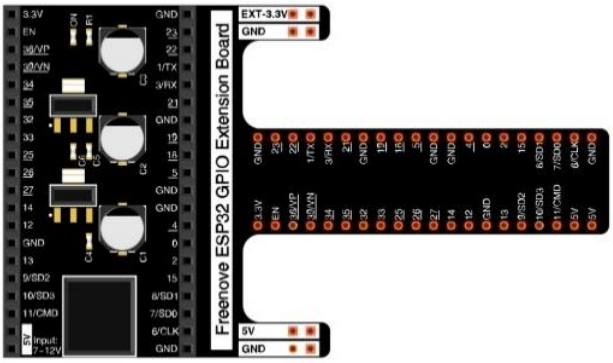
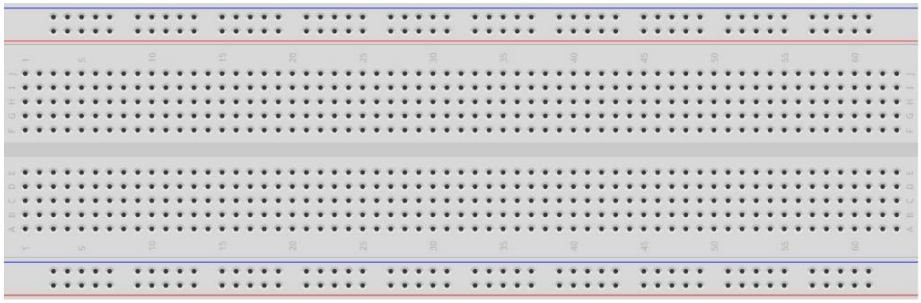
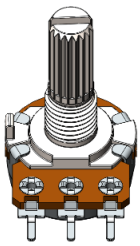
Chapter 8 AD/DA Converter

We have learned how to control the brightness of LED through PWM and understood that PWM is not the real analog before. In this chapter, we will learn how to read analog, convert it into digital and convert the digital into analog output. That is, ADC and DAC.

Project 8.1 Read the Voltage of Potentiometer

In this project, we will use the ADC function of ESP32 to read the voltage value of potentiometer. And then output the voltage value through the DAC to control the brightness of LED.

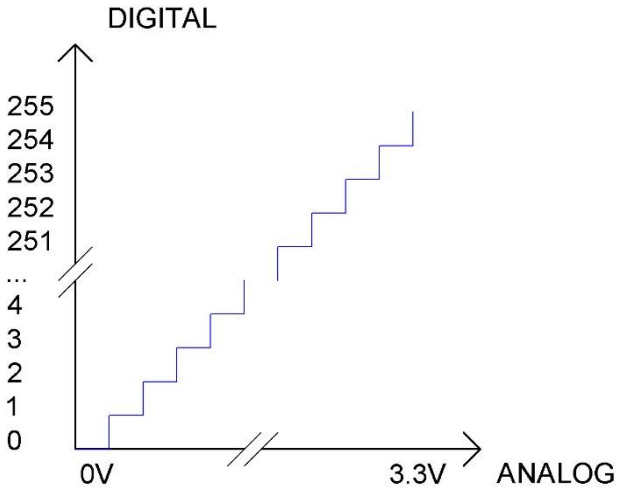
Component List

<table>
<tr><td colspan="2">ESP32-WROVER x1
  </td><td colspan="2">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>Rotary potentiometer x1
  </td><td>Resistor 220Ω x1
 </td><td>LED x1
 </td><td>Jumper M/M x5
 </td></tr>
</table>

Related knowledge

ADC

An ADC is an electronic integrated circuit used to convert analog signals such as voltages to digital or binary form consisting of 1s and 0s. The range of our ADC on ESP32 is 12 bits, that means the resolution is 2^12=4096, and it represents a range (at 3.3V) will be divided equally to 4096 parts. The rage of analog values corresponds to ADC values. So the more bits the ADC has, the denser the partition of analog will be and the greater the precision of the resulting conversion.



Subsection 1: the analog in rang of 0V---3.3/4095 V corresponds to digital 0;
Subsection 2: the analog in rang of 3.3/4095 V---2*3.3 /4095V corresponds to digital 1;
...
The following analog will be divided accordingly.
The conversion formula is as follows:

$$ADCValue = \frac{Analog\ \ Voltage}{3.3} * 4095$$

DAC

The reversing of this process requires a DAC, Digital-to-Analog Converter. The digital I/O port can output high level and low level (0 or 1), but cannot output an intermediate voltage value. This is where a DAC is useful. ESP32 has two DAC output pins with 8-bit accuracy, GPIO25 and GPIO26, which can divide VDD (here is 3.3V) into 2*8=256 parts. For example, when the digital quantity is 1, the output voltage value is 3.3/256 *1 V, and when the digital quantity is 128, the output voltage value is 3.3/256 *128=1.65V, the higher the accuracy of DAC, the higher the accuracy of output voltage value will be.
The conversion formula is as follows:

$$Analog\ \ Voltage = \frac{DAC\ \ Value}{255} * 3.3\ \ (V)$$

ADC on ESP32

ESP32 has 6 pins can be used to measure analog signals. GPIO pin sequence number and analog pin definition are shown in the following table.

ADC number in ESP32	ESP32 GPIO number
ADC1	GPIO 36
ADC2	GPIO 39
ADC3	GPIO 34
ADC4	GPIO 35
ADC5	GPIO 32
ADC6	GPIO 33

DAC on ESP32

ESP32 has two 8-bit digital analog converters to be connected to GPIO25 and GPIO26 pins, respectively, and it is immutable. As shown in the following table,

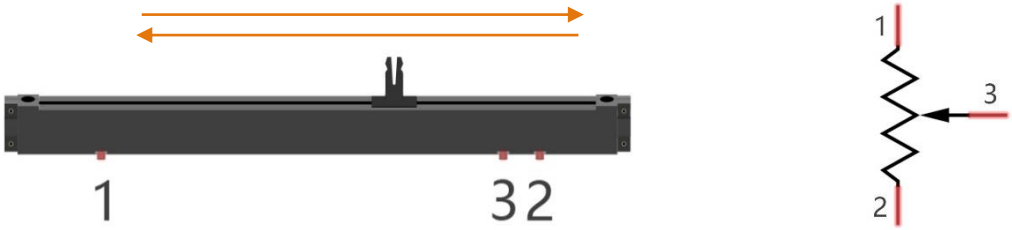
Simulate pin number	GPIO number
DAC1	25
DAC2	26

Note: In this ESP32, GPIO26 is connected to 3.3V through a resistor.Therefore, DAC2 cannot be used.

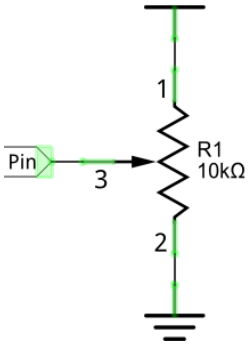
Component knowledge

Potentiometer

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

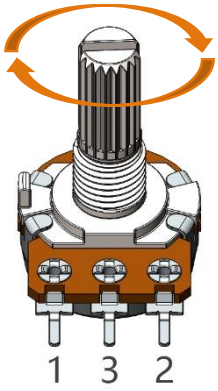


What between potentiometer pin 1 and pin 2 is the resistor body, and pins 3 is connected to brush. When brush moves from pin 1 to pin 2, the resistance between pin 1 and pin 3 will increase up to body resistance linearly, and the resistance between pin 2 and pin 3 will decrease down to 0 linearly.
In the circuit. The both sides of resistance body are often connected to the positive and negative electrode of the power. When you slide the brush pin 3, you can get a certain voltage in the range of the power supply.



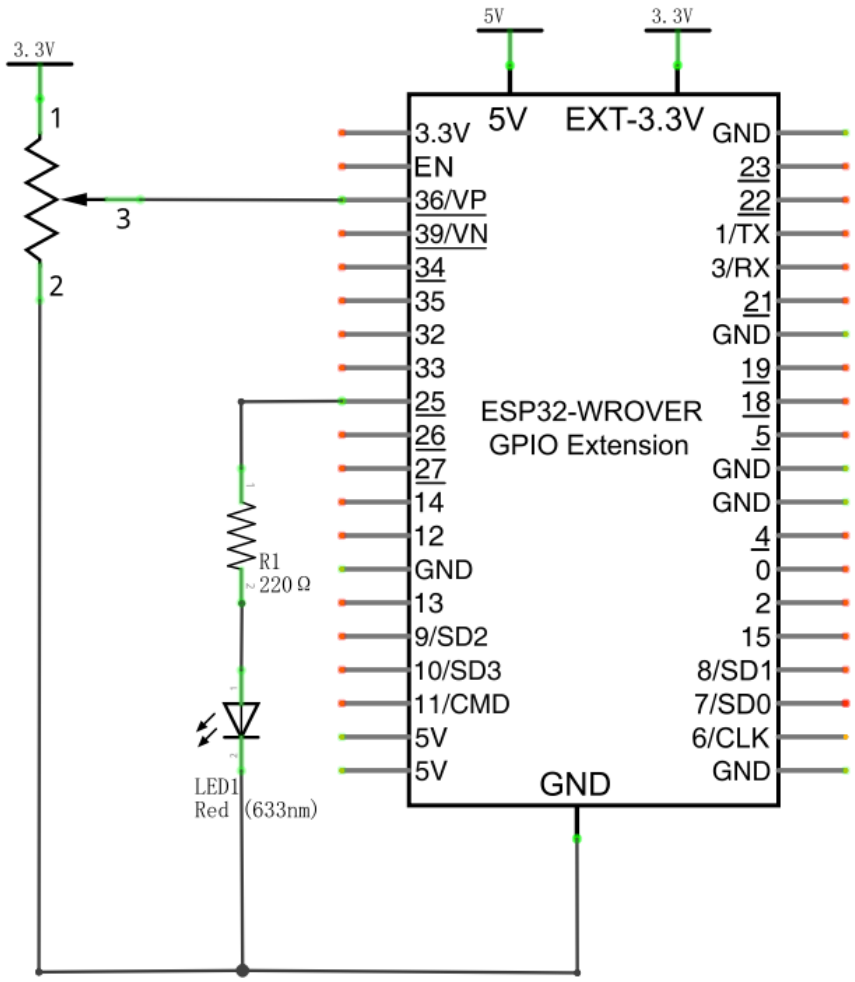
Rotary potentiometer

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

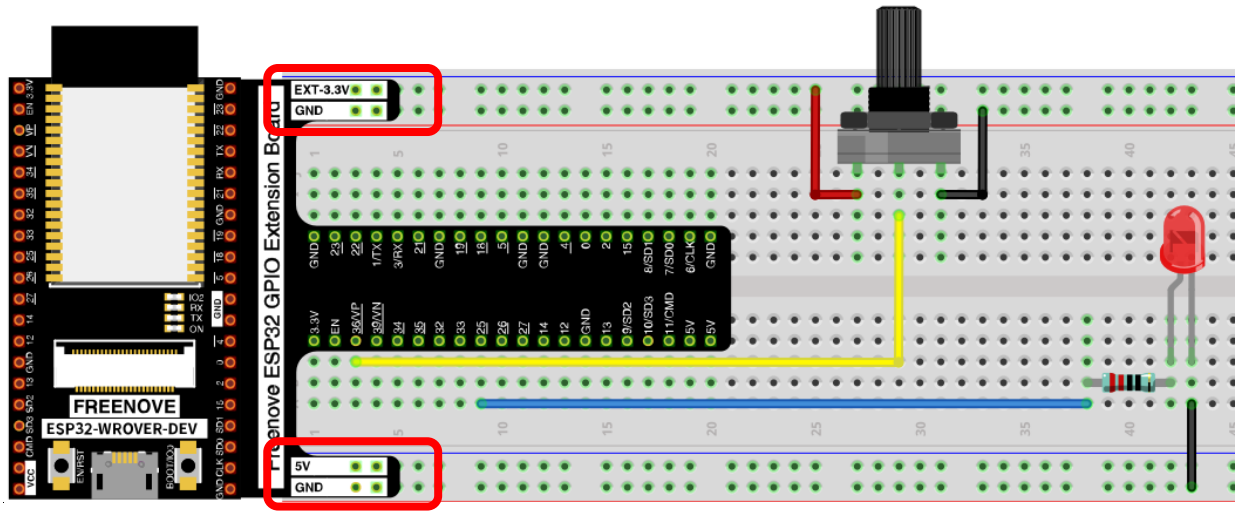


Circuit

Schematic diagram



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

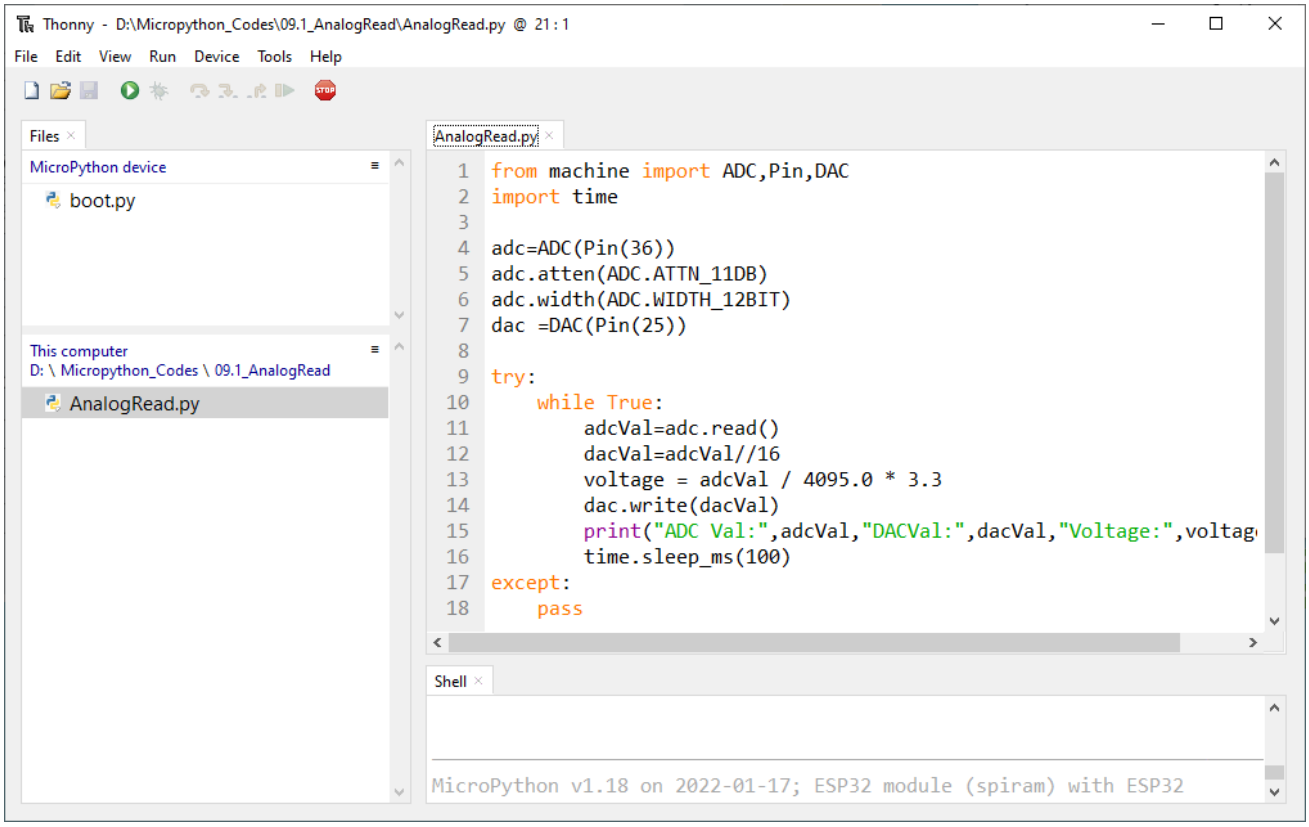


Code

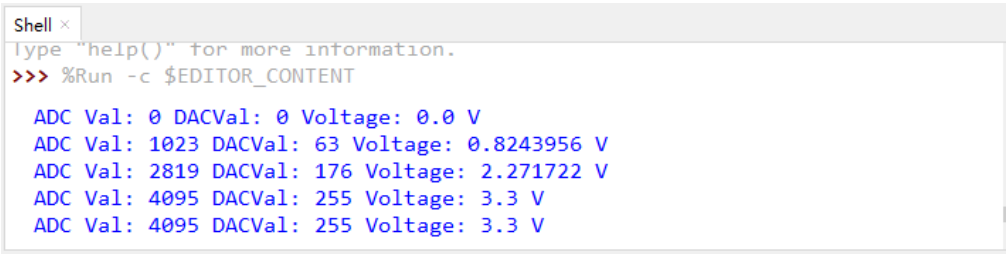
Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

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

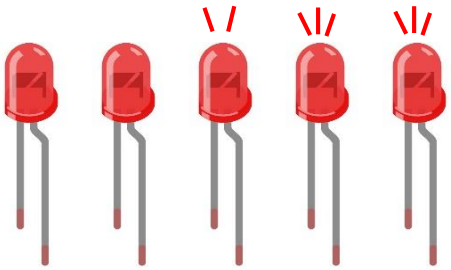
08.1_AnalogRead



Click "Run current script" and observe the message printed in "Shell".



LEDs display as below:



"Shell" prints ADC value, DAC value, the output voltage of potentiometer and other information. In the code, we make the output voltage of the DAC pin equal to the input voltage of the ADC pin. Rotate the handle of the potentiometer, the printed information will change. When the voltage is greater than 1.6V (turn-on voltage of red LED), the LED starts to emit light. If you continue to increase the output voltage, the LED will gradually become brighter. And when the voltage is less than 1.6V, the LED will not light up, because this does not reach the turn-on voltage of the LED, which indirectly proves the difference between DAC and PWM. (If you have an oscilloscope, you can view the waveform output by the DAC through the oscilloscope)

The following is the code:

```
1  from machine import ADC,Pin,DAC
2  import time
3
4  adc=ADC(Pin(36))
5  adc.atten(ADC.ATTN_11DB)
6  adc.width(ADC.WIDTH_12BIT)
7  dac =DAC(Pin(25))
8
9  try:
10     while True:
11         adcVal=adc.read()
12         dacVal=adcVal//16
13         voltage = adcVal / 4095.0 * 3.3
14         dac.write(dacVal)
15         print("ADC Val:",adcVal,"DACVal:",dacVal,"Voltage:",voltage,"V")
16         time.sleep_ms(100)
17 except:
18     pass
```

Import Pin, ADC and DAC modules.

```
1  from machine import ADC,Pin,DAC
2  import time
```

Turn on and configure the ADC with the range of 0-3.3V and the data width of 12-bit data width, and turn on the DAC pin.

```
4  adc=ADC(Pin(36))
5  adc.atten(ADC.ATTN_11DB)
6  adc.width(ADC.WIDTH_12BIT)
7  dac =DAC(Pin(25))
```

Read ADC value once every 100 millisecods, convert ADC value to DAC value and output it, control the brightness of LED and print these data to "Shell".

```
10      while True:
11          adcVal=adc.read()
12          dacVal=adcVal//16
13          voltage = adcVal / 4095.0 * 3.3
14          dac.write(dacVal)
15          print("ADC Val:",adcVal,"DACVal:",dacVal,"Voltage:",voltage,"V")
16          time.sleep_ms(100)
```

Reference

Class ADC

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

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

pin: Available pins are: Pin(36)、Pin(39)、Pin(34)、Pin(35)、Pin(32)、Pin(33)。

ADC.read(): Read ADC and return the value.

ADC.atten(db): Set attenuation ration (that is, the full range voltage, such as the voltage of 11db full range is 3.3V)

db: attenuation ratio

ADC.ATTIN_0DB —full range of 1.2V

ADC.ATTN_2_5_DB —full range of 1.5V

ADC.ATTN_6DB —full range of 2.0 V

ADC.ATTN_11DB —full range of 3.3V

ADC.width(bit): Set data width.

bit: data bit

ADC.WIDTH_9BIT —9 data width

ADC.WIDTH_10BIT — 10 data width

ADC.WIDTH_11BIT — 11 data width

ADC.WIDTH_12BIT — 12 data width

Class DAC

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

machine.DAC(pin): Create a DAC object associated with the given pin.

pin: Available pins are: Pin(25)、Pin(26)

DAC.write(value): Output voltage

value: The range of data value: 0-255, corresponding output voltage of 0-3.3V

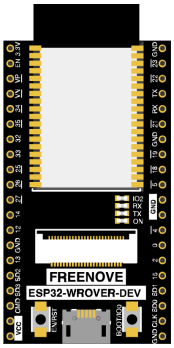
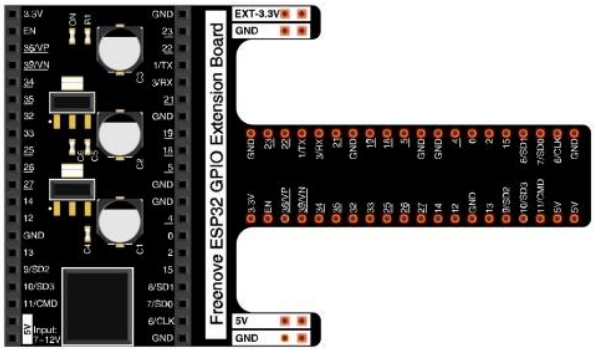
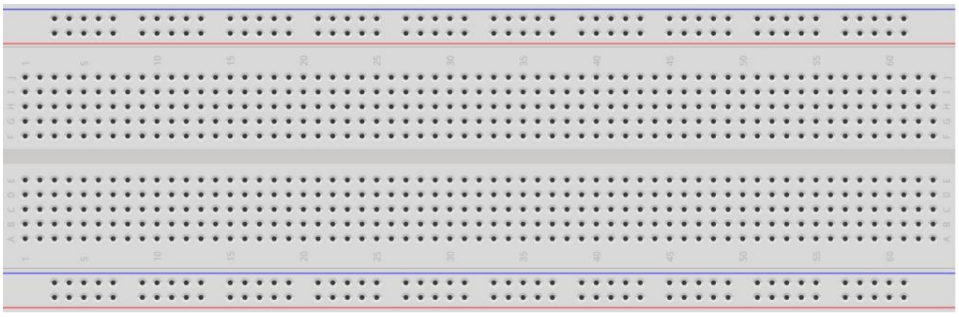
Chapter 9 TouchSensor

ESP32 offers up to 10 capacitive touch GPIO, and as you can see from the previous section, mechanical switches are prone to jitter that must be eliminated when used, which is not the case with ESP32's built-in touch sensor. In addition, on the service life, the touch switch also has advantages that mechanical switch is completely incomparable.

Project 9.1 Read Touch Sensor

This project reads the value of the touch sensor and prints it out.

Component List

ESP32-WROVER x1 	GPIO Extension Board x1 
Breadboard x1 	
Jumper M/M x1	

Related knowledge

Touch sensor

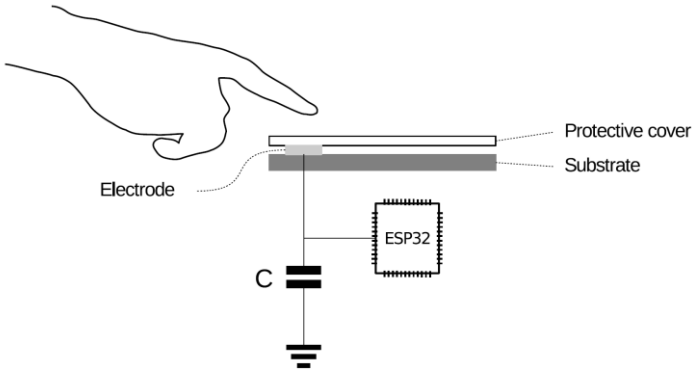
ESP32's touch sensor supports up to 7 GPIO channels as capacitive touch pins. Each pin can be used separately as an independent touch switch or be combined to produce multiple touch points. The following table is a list of available touch pins on ESP32.

Functions of pins	ESP32 GPIO number
GPIO4	GPIO4
MTDO	GPIO15
MTCK	GPIO13
MTDI	GPIO12
MTMS	GPIO14
32K_XN	GPIO33
32K_XP	GPIO32

The pin numbers are shown in the figure above. When you need to use the TouchPad, you only need to call the TouchPad function to initialize the corresponding pins.

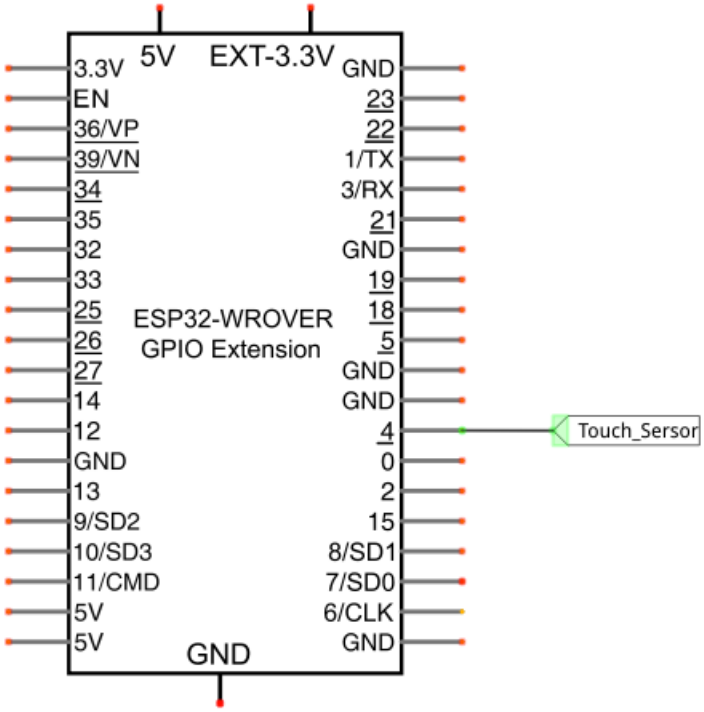
The electronic signals generated with the touch are analog, which are converted by the internal ADC. You may have found that all touch pins have featured with ADC.

The way to connect the hardware is as follows:

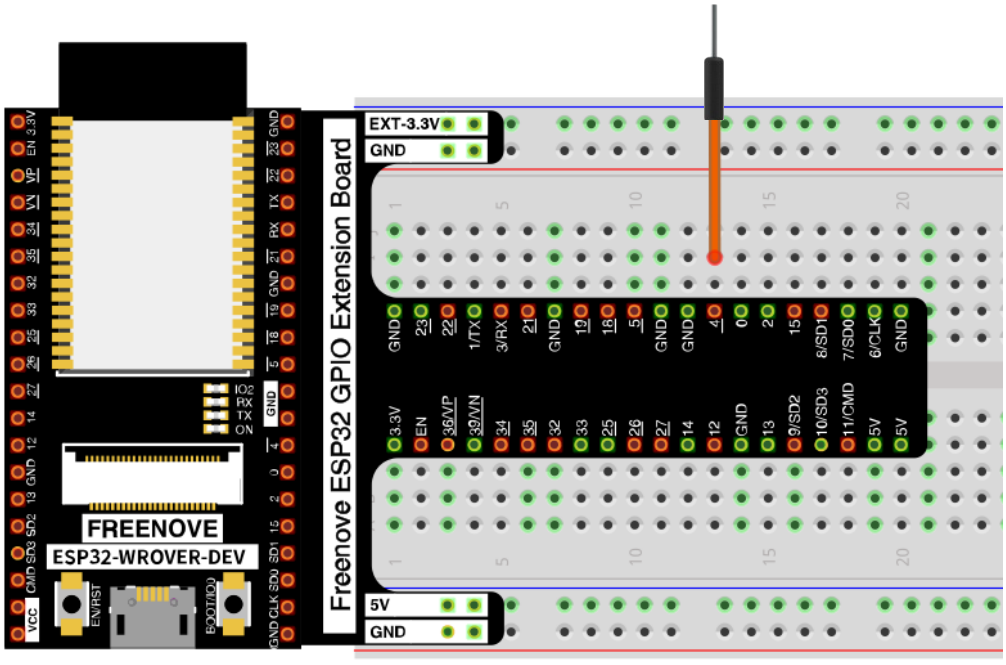


Circuit

Schematic diagram



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

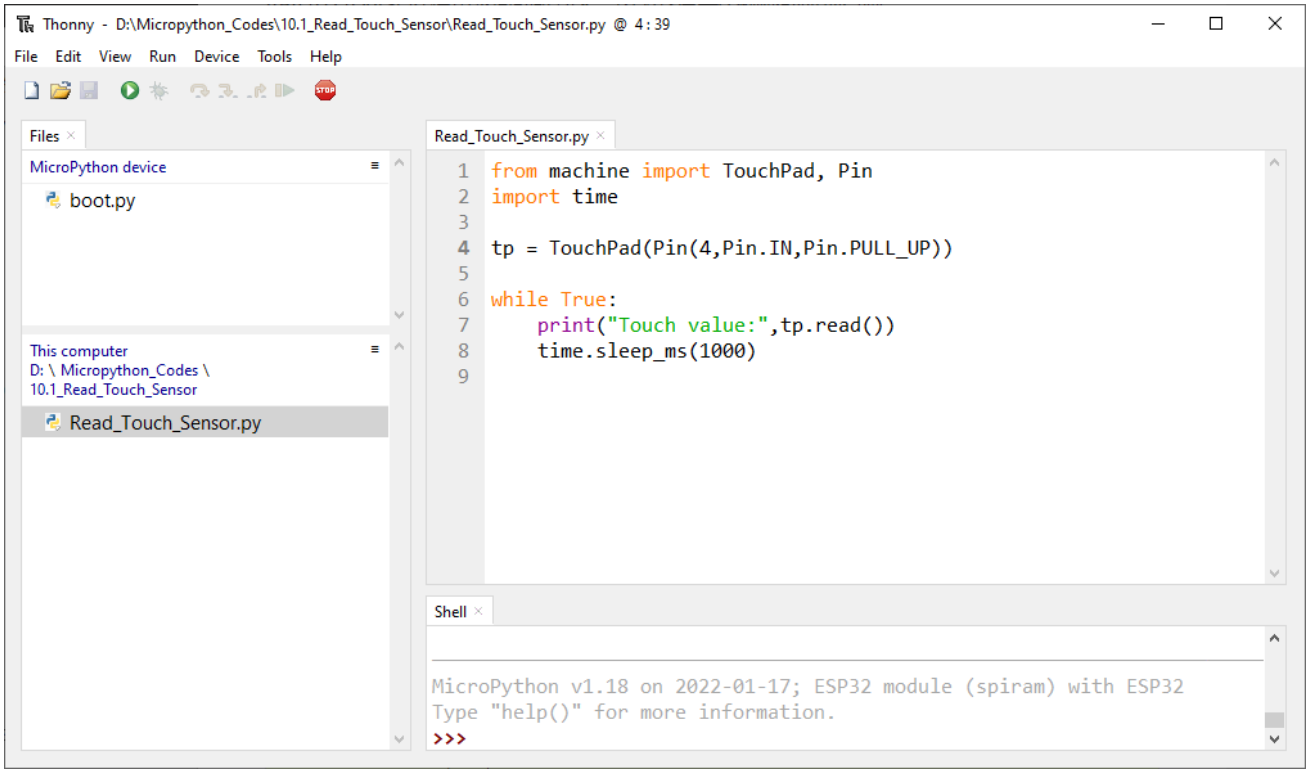


Code

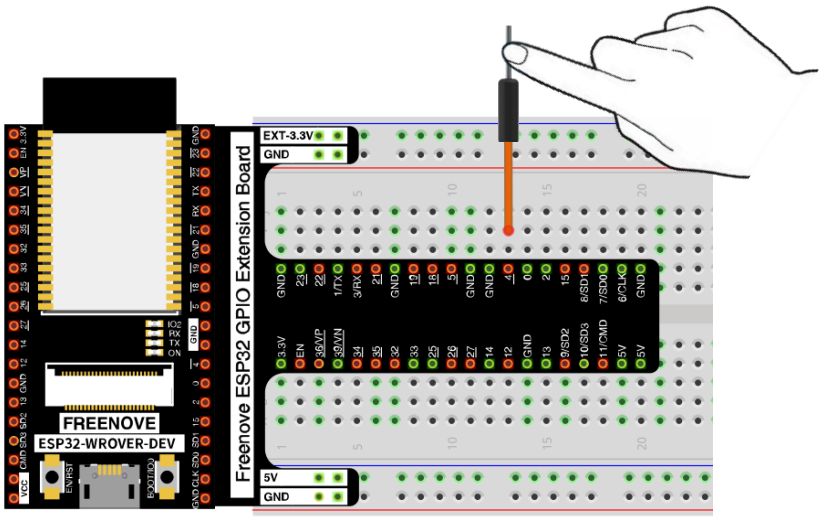
Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "09.1_Read_Touch_Sensor" and double click "Read_Touch_Sensor.py".

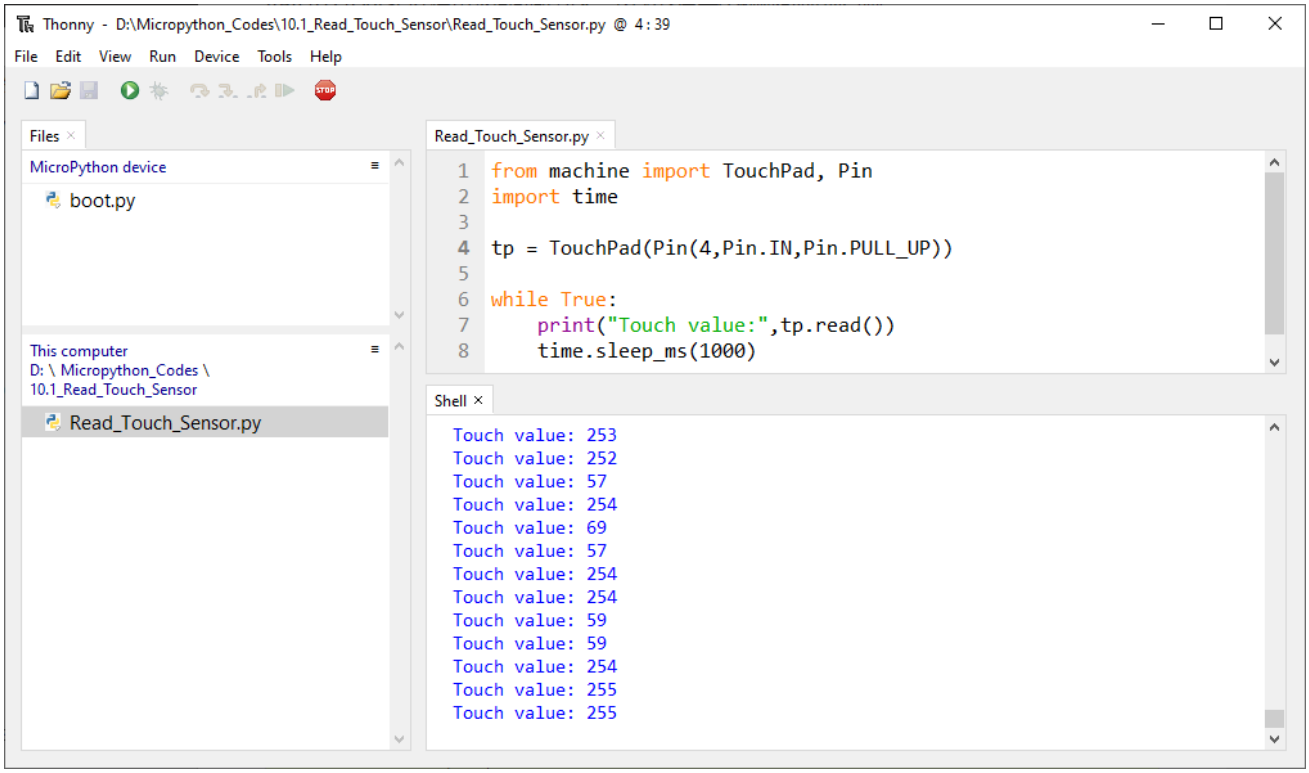
09.1_Read_Touch_Sensor



Click "Run current script", touch the jumper wire with your finger and observe the messages printed in "Shell".



Messages printed at "Shell":



Reference

Class TouchPad

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

TouchPad(pin): Initialize the TouchPad object and associate it with ESP32 pins.

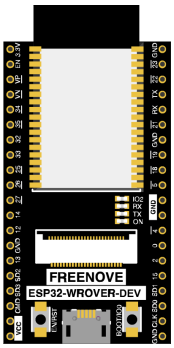
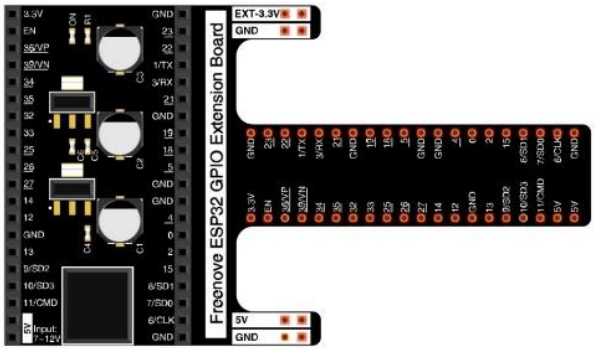
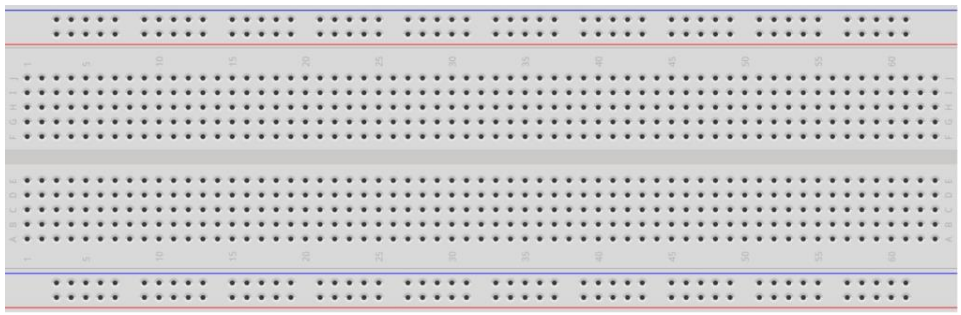
pin: Pin(4)、Pin(15)、Pin(13)、Pin(12)、Pin(14)、Pin(32)、Pin(33)

TouchPad.read(): Read the capacitance of touchpad. If your fingers touch TouchPad pins, the capacitance decreases; Otherwise, it will not change.

Project 9.2 TouchLamp

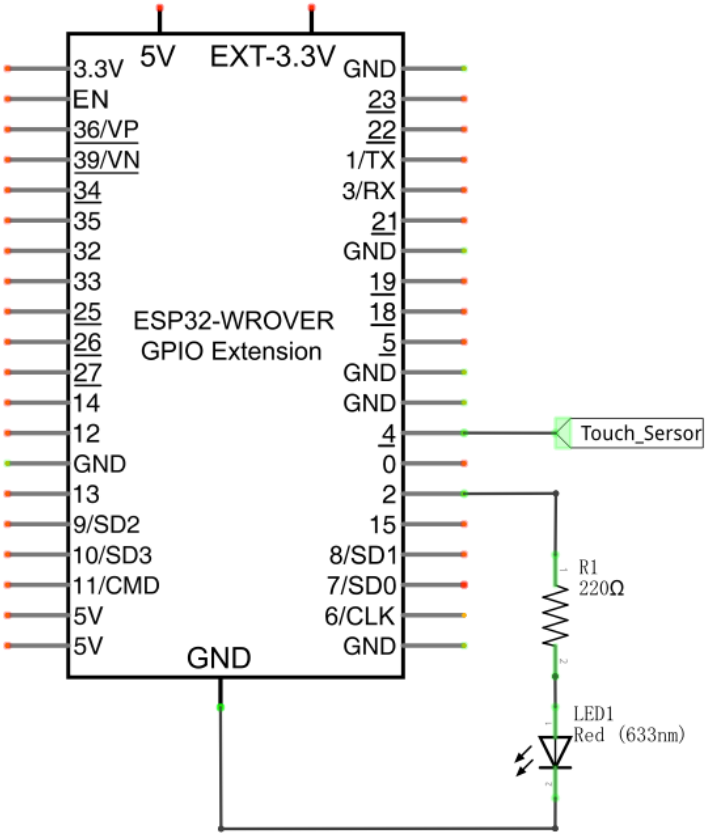
In this project, we will use ESP32's touch sensor to create a touch switch lamp.

Component List

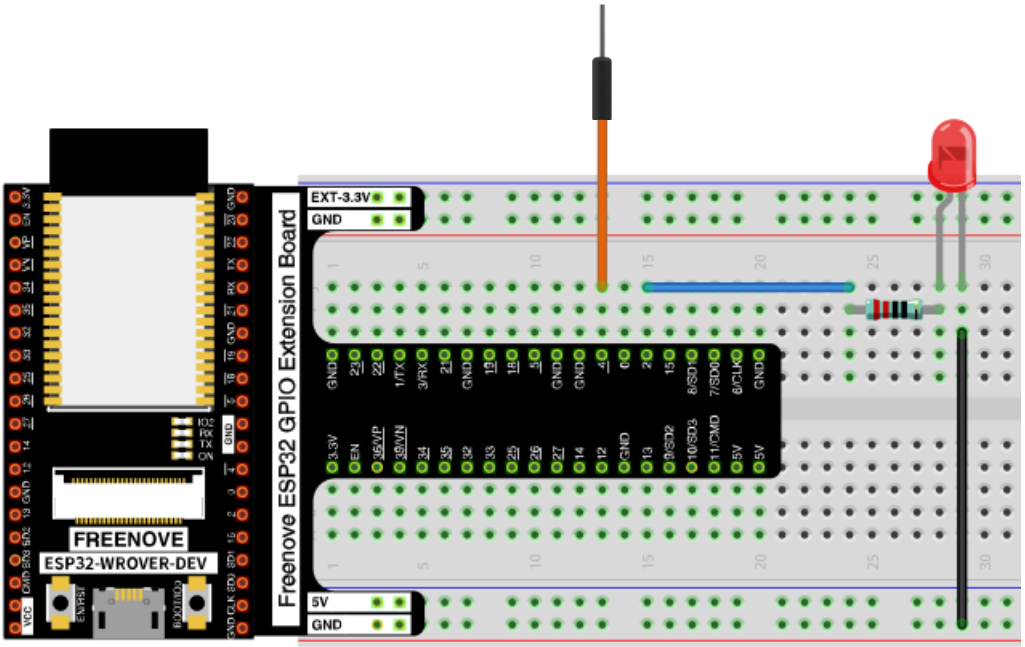
ESP32-WROVER x1 	GPIO Extension Board x1 	
Breadboard x1 		
Jumper M/M x3	LED x1	Resistor 220Ω x1

Circuit

Schematic diagram



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

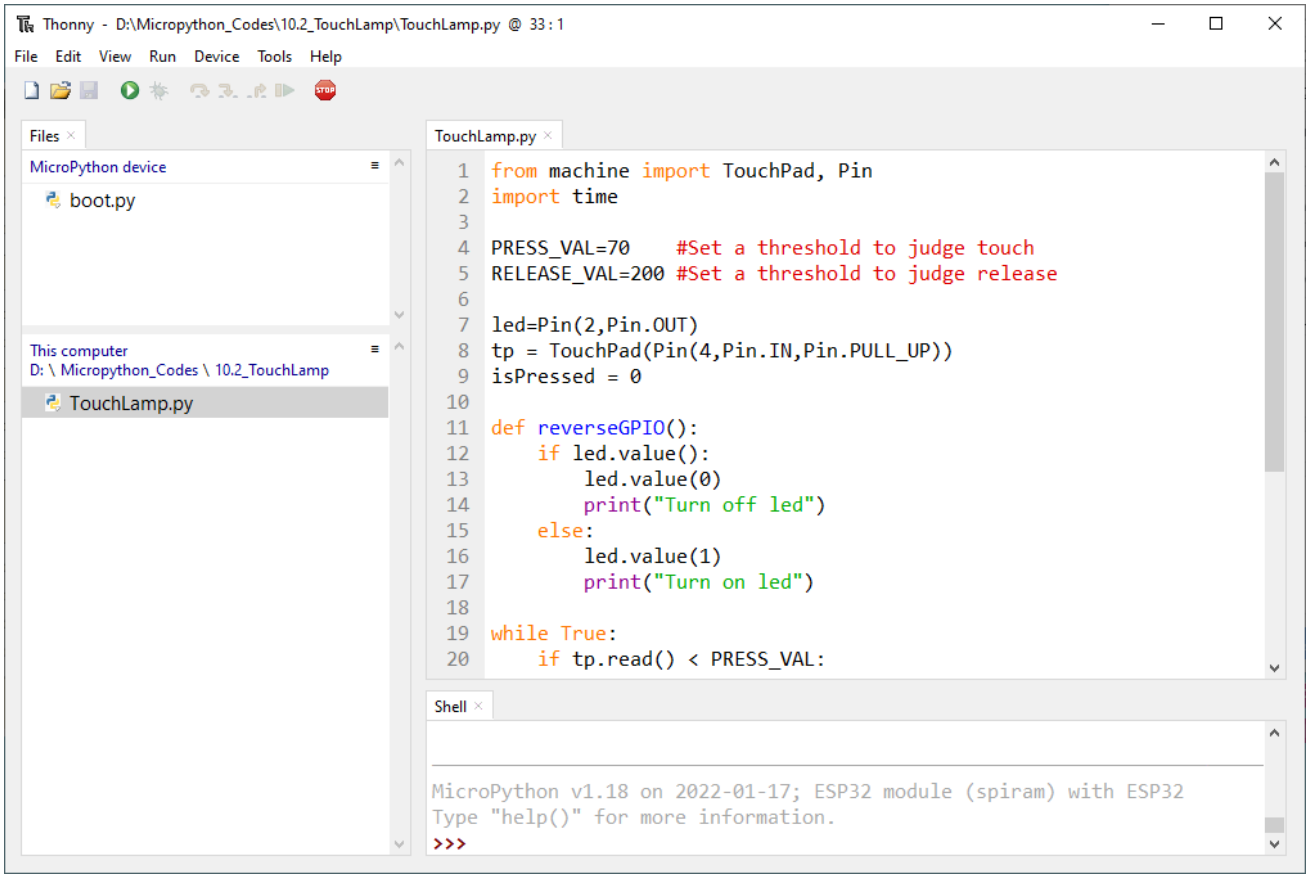


Code

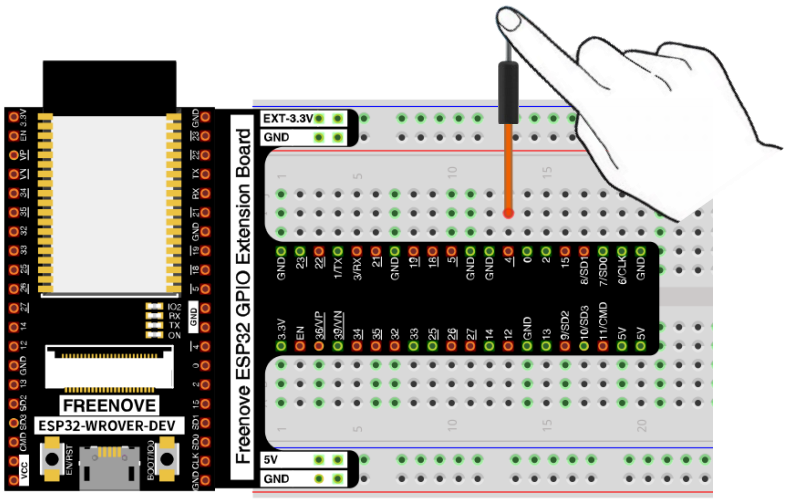
Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny, click "This computer" → "D:" → "Micropython_Codes" → "09.2_TouchLamp"and double click "TouchLamp.py".

09.2_TouchLamp



Click "Run current script" and then touch the jumper wire with your finger. The state of LED will change with each touch and the detection state of the touch sensor will be printed in the "Shell"



```
Shell
MicroPython v1.18 on 2022-01-17; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
  Turn on led
  Touch detected!
  Touch released!
  Turn off led
  Touch detected!
  Touch released!
  Turn on led
  Touch detected!
  Touch released!
  Turn off led
  Touch detected!
  Touch released!
  Turn on led
  Touch detected!
  Touch released!
  Turn off led
  Touch detected!
  Touch released!
```

LED displays as follows:

The following is the program code:

```
from machine import TouchPad, Pin
import time

PRESS_VAL=70    #Set a threshold to judge touch
RELEASE_VAL=200 #Set a threshold to judge release

led=Pin(2,Pin.OUT)
tp = TouchPad(Pin(4,Pin.IN,Pin.PULL_UP))
isPressed = 0

def reverseGPIO():
    if led.value():
        led.value(0)
        print("Turn off led")
    else:
        led.value(1)
        print("Turn on led")
while True:
    if tp.read() < PRESS_VAL:
```

```
20          if not isPressed:
21              isPressed = 1
22              reverseGPIO()
23              print("Touch detected!")
24              time.sleep_ms(100)
25      if tp.read() > RELEASE_VAL:
26          if isPressed:
27              isPressed = 0
28              print("Touch released!")
29              time.sleep_ms(100)
```

Import Pin and TouchPad modules.

```
1   from machine import TouchPad, Pin
2   import time
```

The closer the return value of the function read() is to 0, the more obviously the touch action is detected. As this is not a fixed value, a threshold value needs to be defined. When the value of the sensor is less than the threshold, it is considered a valid touch action. Similarly, define a threshold value for the released state, and the value between the sensor value and the threshold is regarded as an invalid interference value.

```
4   PRESS_VAL=70    #Set a threshold to judge touch
5   RELEASE_VAL=200 #Set a threshold to judge release
```

First, decide whether the touch is detected. If yes, print some messages, reverse the state of LED and set the flag bit isProcessed to 1 to avoid repeatedly executing the program after a touch is detected.

```
19      if tp.read() < PRESS_VAL:
20          if not isPressed:
21              isPressed = 1
22              reverseGPIO()
23              print("Touch detected!")
24              time.sleep_ms(100)
```

And then decide whether the touch key is released. If yes, print some messages, and set isProcessed to 0 to avoid repeatedly executing the program after a touch is released and to prepare for the next touch detector.

```
25      if tp.read() > RELEASE_VAL:
26          if isPressed:
27              isPressed = 0
28              print("Touch released!")
29              time.sleep_ms(100)
```

Customize a function that reverses the output level of the LED each time it is called.

```
11  def reverseGPIO():
12      if led.value():
13          led.value(0)
14          print("Turn off led")
15      else:
16          led.value(1)
17          print("Turn on led")
```

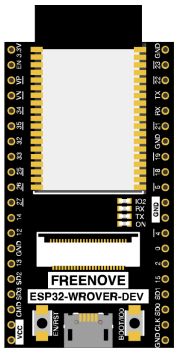
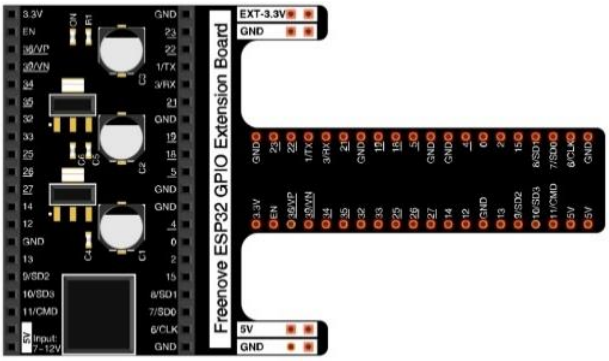
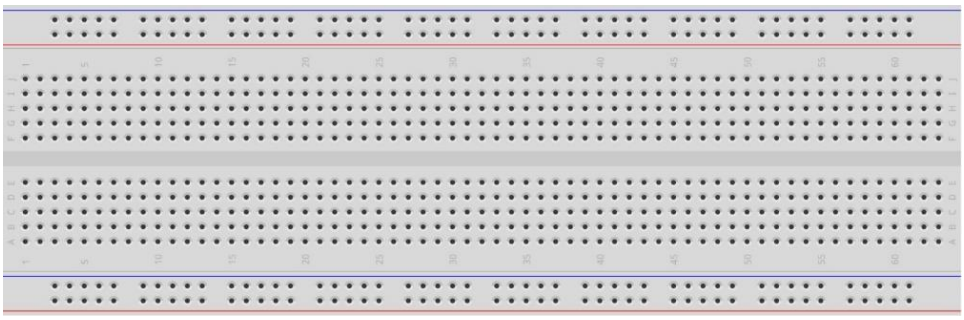
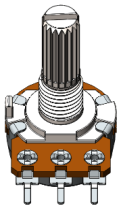
Chapter 10 Potentiometer & LED

We have learned how to use ADC and DAC before. When using DAC output analog to drive LED, we found that, when the output voltage is less than led turn-on voltage, the LED does not light; when the output analog voltage is greater than the LED voltage, the LED lights. This leads to a certain degree of waste of resources. Therefore, in the control of LED brightness, we should choose a more reasonable way of PWM control. In this chapter, we learn to control the brightness of LED through a potentiometer.

Project 10.1 Soft Light

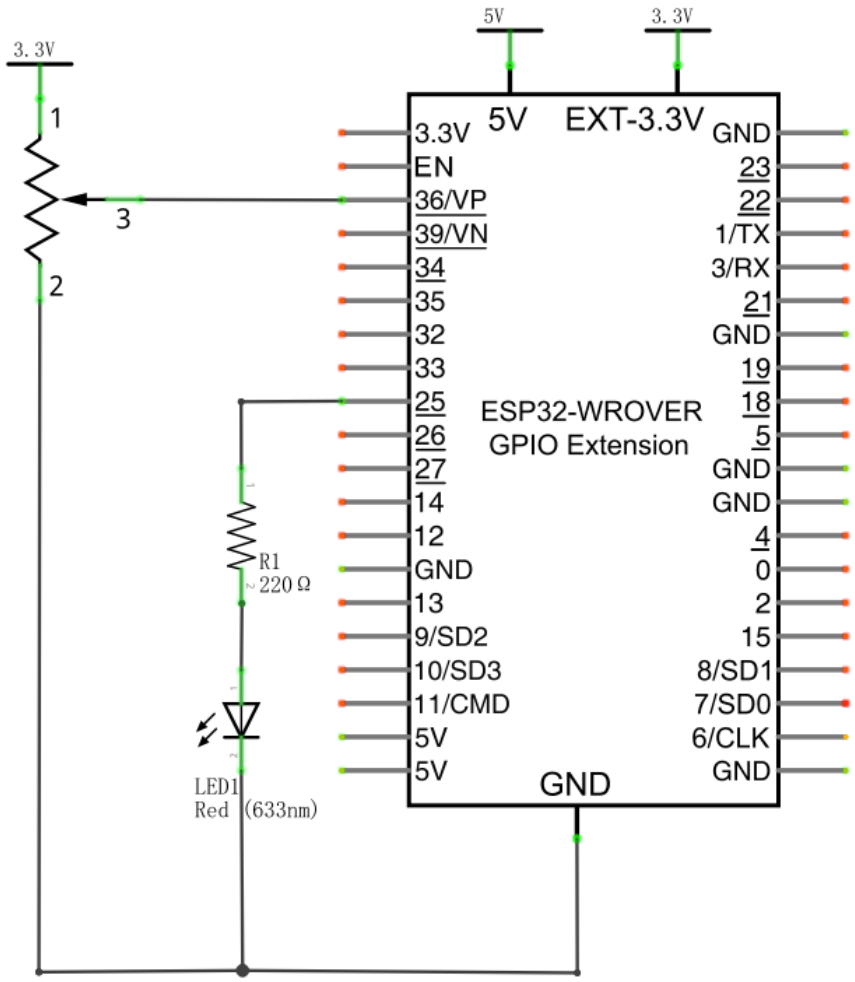
In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle of the PWM used to control the brightness of an LED. Then you can change the brightness of an LED by adjusting the potentiometer.

Component List

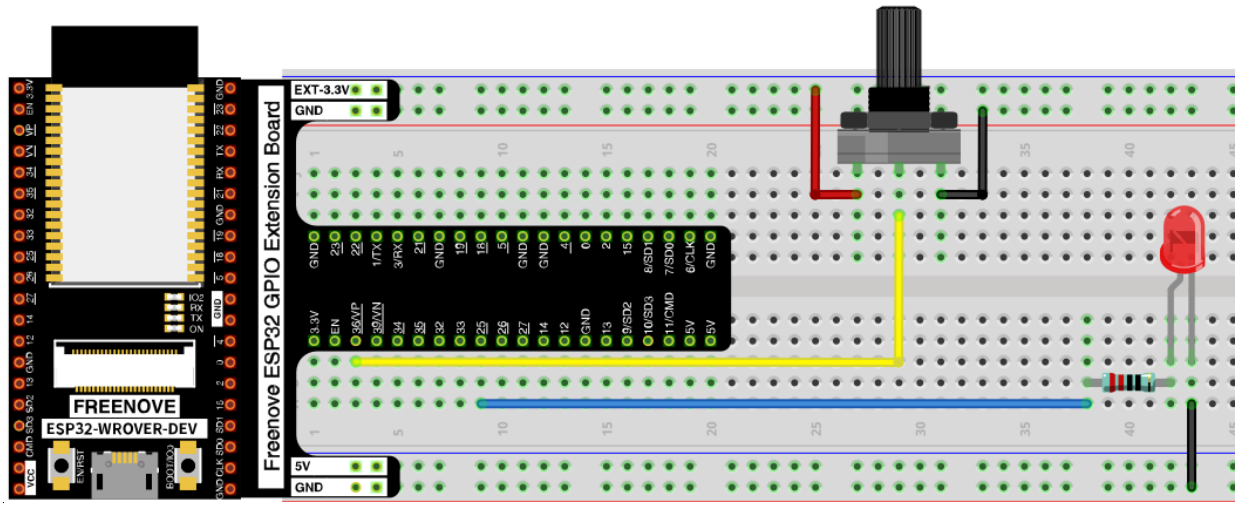
<table>
<tr><td colspan="2">ESP32-WROVER x1
  </td><td colspan="2">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>Rotary potentiometer x1
  </td><td>Resistor 220Ω x1
 </td><td>LED x1
 </td><td>Jumper M/M x5
</td></tr>
</table>

Circuit

Schematic diagram



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

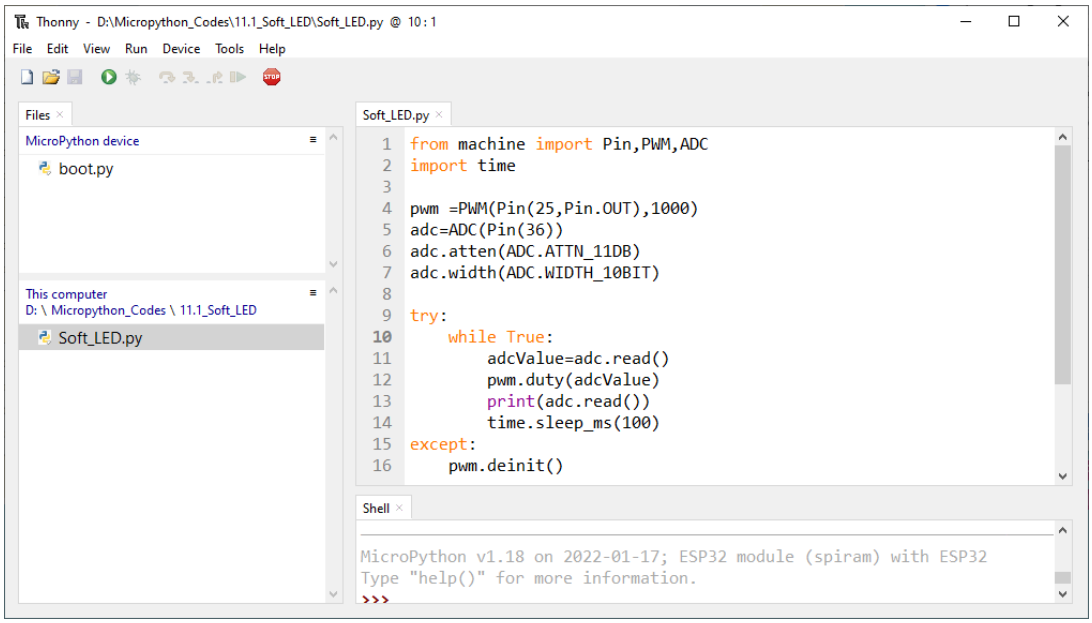


Code

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

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

10.1_Soft_LED



Click "Run current script". Rotate the handle of potentiometer and the brightness of LED will change correspondingly.

The following is the code:

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

pwm = PWM(Pin(25,Pin.OUT),1000)
adc=ADC(Pin(36))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_10BIT)

try:
    while True:
        adcValue=adc.read()
        pwm.duty(adcValue)
        print(adc.read())
        time.sleep_ms(100)
except:
    pwm.deinit()
```

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

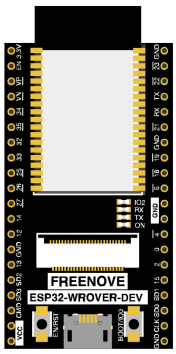
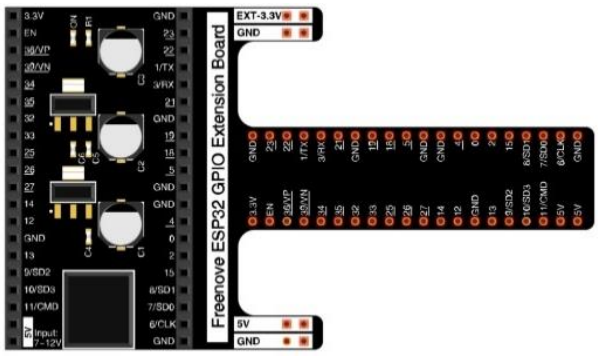
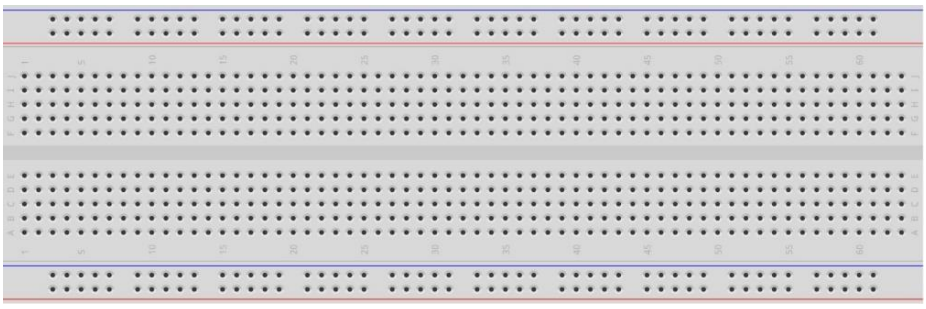
Chapter 11 Photoresistor & LED

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

Project 11.1 NightLamp

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

Component List

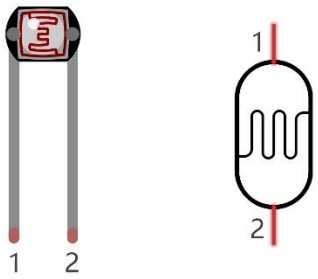
<table>
<tr><td colspan="2">ESP32-WROVER x1
  </td><td colspan="2">GPIO Extension Board x1
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>Photoresistor x1
 </td><td colspan="2">Resistor
220Ω x1

10KΩ x1
 </td><td>LED x1
 </td><td>Jumper M/M x4
 </td></tr>
</table>

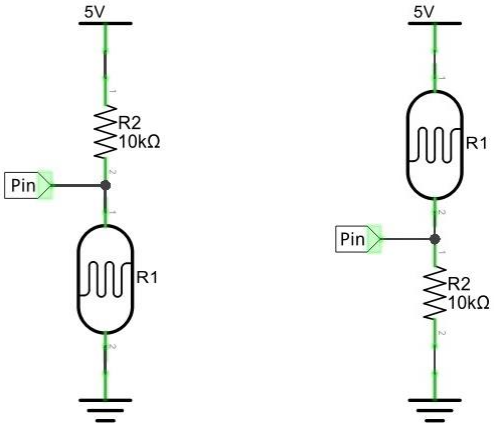
Component knowledge

Photoresistor

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



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

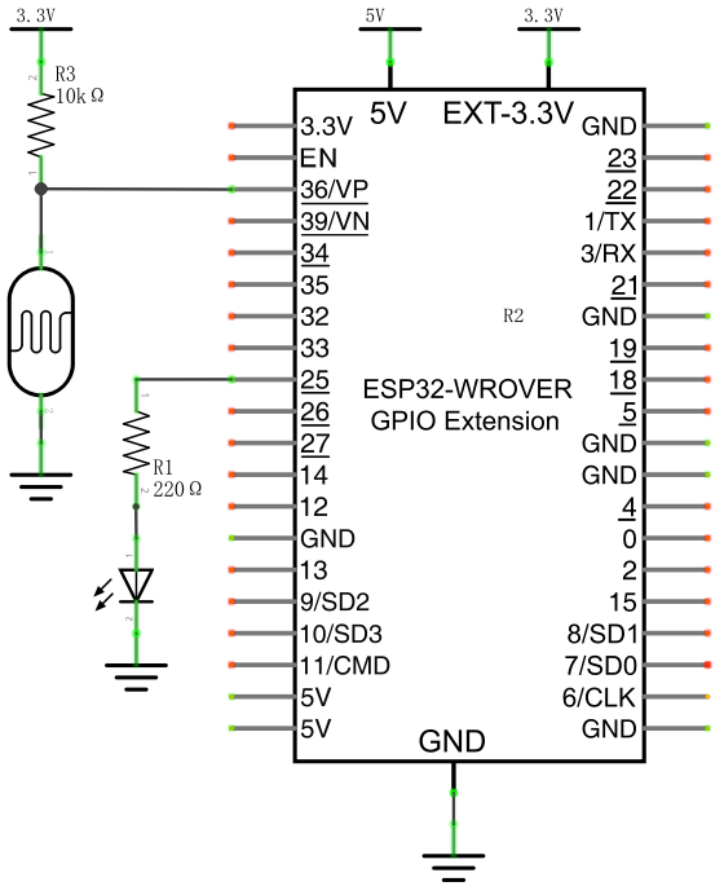


In the above circuit, when a Photoresistor's resistance vale changes due to a change in light intensity, the voltage between the Photoresistor and Resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

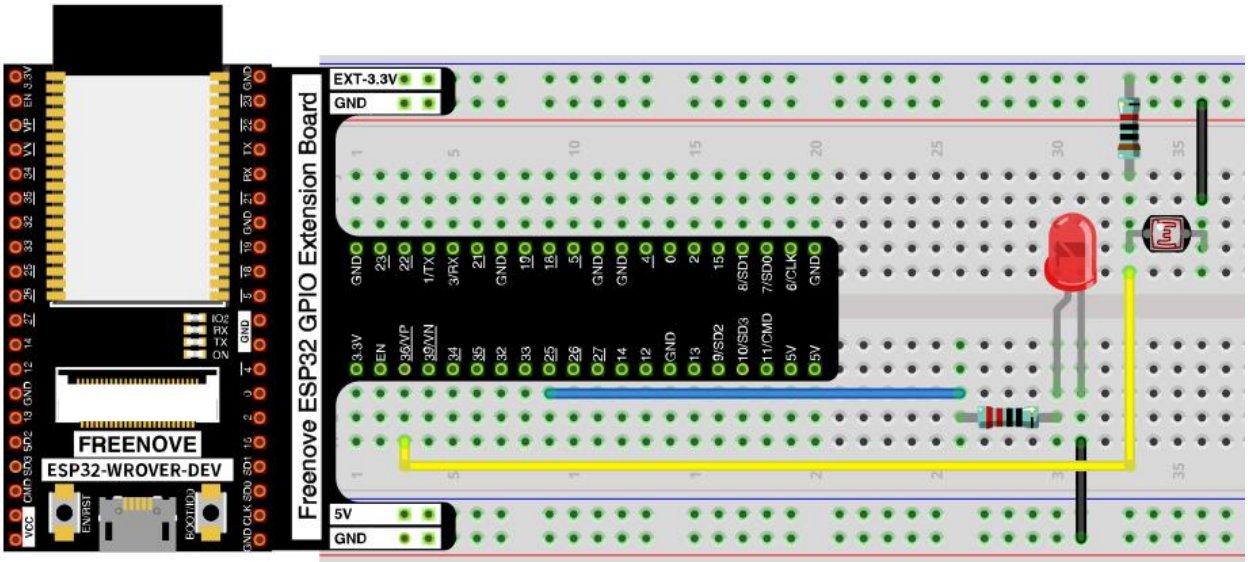
Circuit

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

Schematic diagram



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

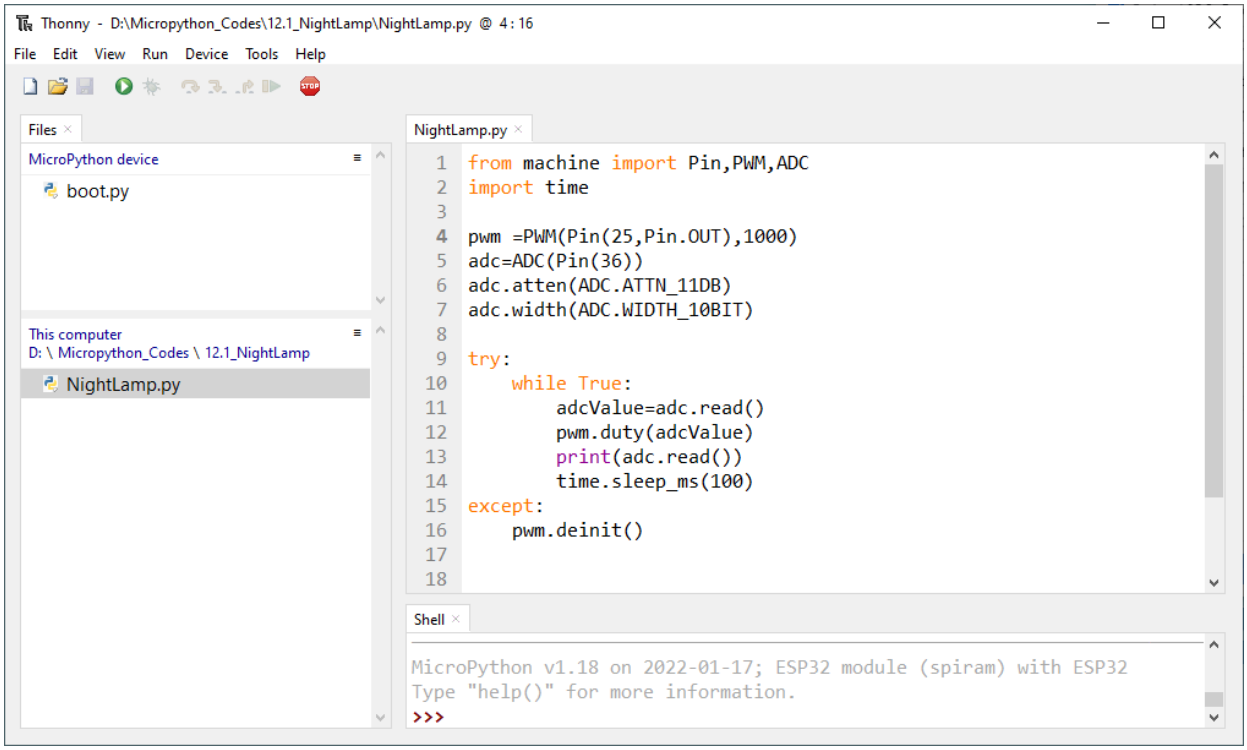


Code

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Codes of this project is logically the same as the project Soft Light.

11.1_Nightlamp



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

The following is the program code:

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

pwm =PWM(Pin(25,Pin.OUT),1000)
adc=ADC(Pin(36))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_10BIT)

try:
    while True:
        adcValue=adc.read()
        pwm.duty(adcValue)
        print(adc.read())
        time.sleep_ms(100)
except:
    pwm.deinit()
```

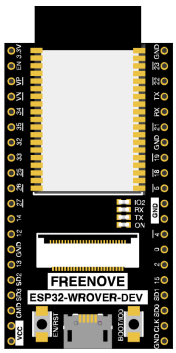
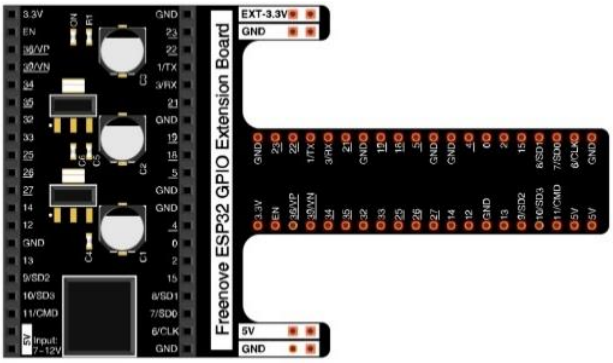
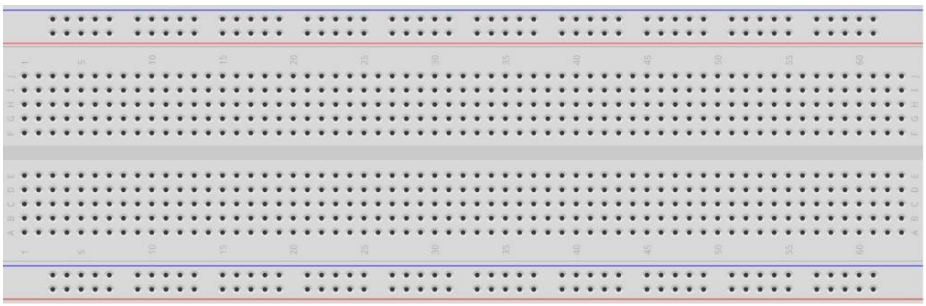
Chapter 12 Thermistor

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

Project 12.1 Thermometer

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

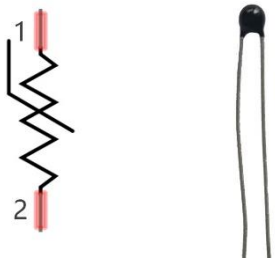
Component List

ESP32-WROVER x1 	GPIO Extension Board x1 	
Breadboard x1 		
Thermistor x1	Resistor 1kΩ x1	Jumper M/M x3

Component knowledge

Thermistor

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



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

$$Rt = R * EXP[\ B * (\frac{1}{T2} - \frac{1}{T1})\ ]$$

Where:

Rt is the thermistor resistance under T2 temperature;

R is the nominal resistance of thermistor under T1 temperature;

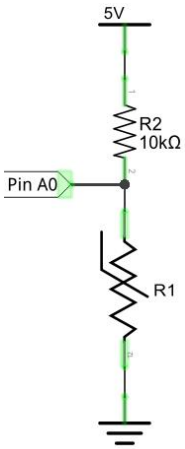
EXP[n] is nth power of e;

B is for thermal index;

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

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

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



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

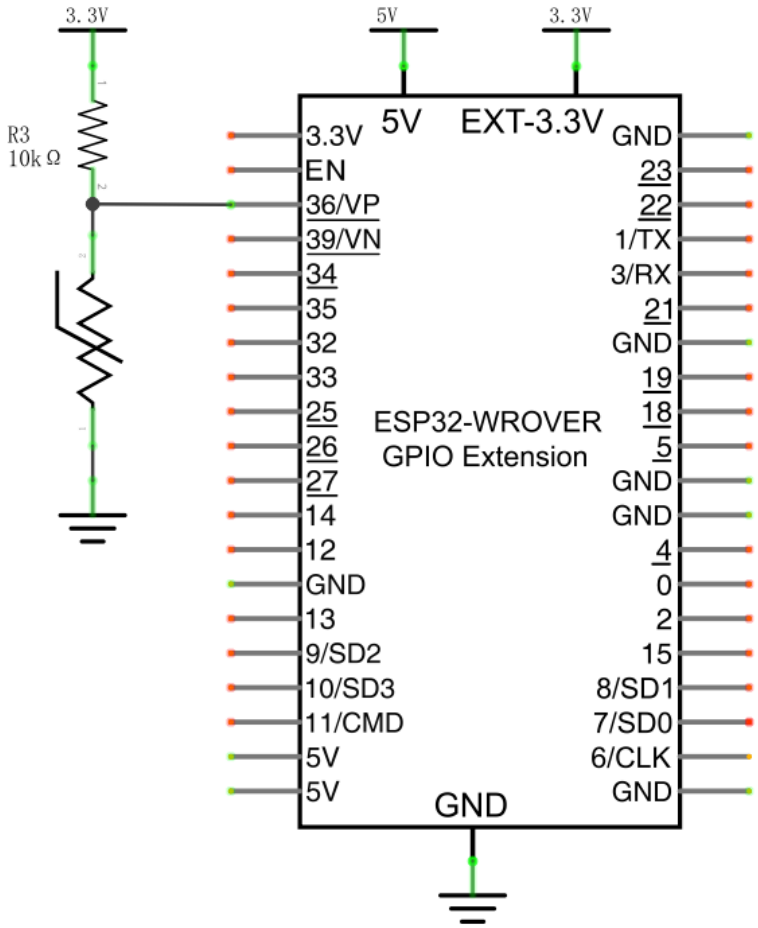
Therefore, the temperature formula can be derived as:

$$T2 = 1/(\frac{1}{T1} + \ln(\frac{Rt}{R})/B)$$

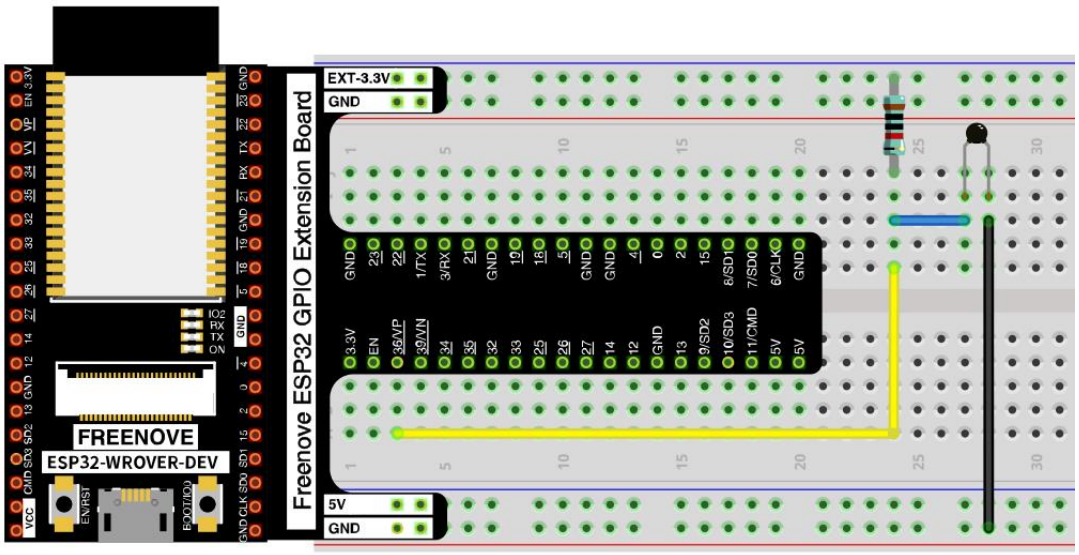
Circuit

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

Schematic diagram



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

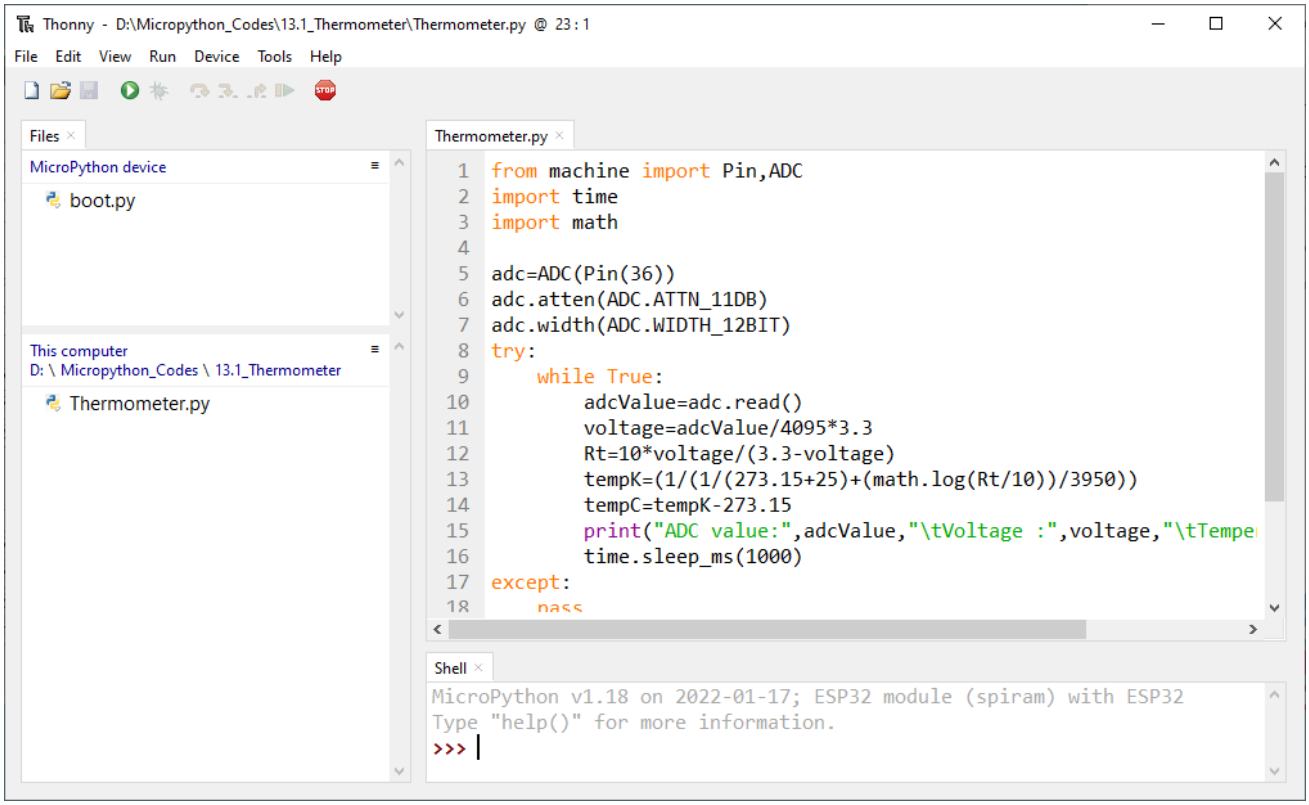


Code

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

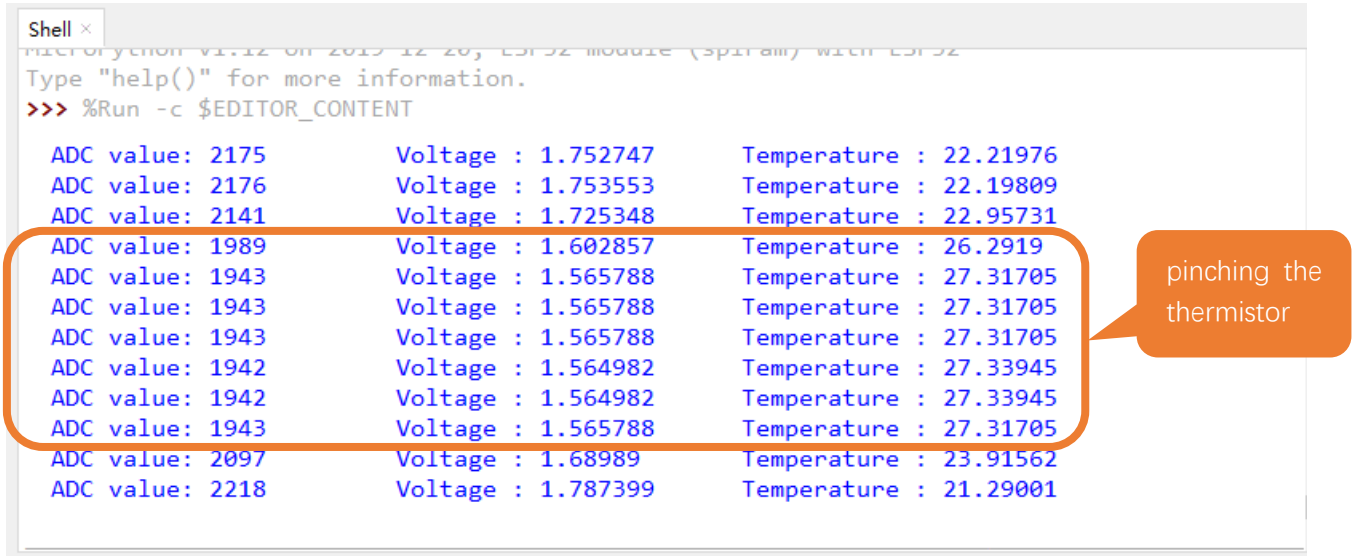
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "12.1_Thermometer" and double click "Thermometer.py".

12.1_Thermometer



Click "Run current script" and "Shell" will constantly display the current ADC value, voltage value and temperature value. Try to "pinch" the thermistor (without touching the leads) with your index finger and thumb for a brief time, you should see that the temperature value increases.

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


Shell
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
ADC value: 2175 Voltage : 1.752747 Temperature : 22.21976
ADC value: 2176 Voltage : 1.753553 Temperature : 22.19809
ADC value: 2141 Voltage : 1.725348 Temperature : 22.95731
ADC value: 1989 Voltage : 1.602857 Temperature : 26.2919
ADC value: 1943 Voltage : 1.565788 Temperature : 27.31705
ADC value: 1943 Voltage : 1.565788 Temperature : 27.31705
ADC value: 1943 Voltage : 1.565788 Temperature : 27.31705
ADC value: 1942 Voltage : 1.564982 Temperature : 27.33945
ADC value: 1942 Voltage : 1.564982 Temperature : 27.33945
ADC value: 1943 Voltage : 1.565788 Temperature : 27.31705
ADC value: 2097 Voltage : 1.68989 Temperature : 23.91562
ADC value: 2218 Voltage : 1.787399 Temperature : 21.29001
pinching the thermistor

The following is the code:

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

adc=ADC(Pin(36))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

try:
    while True:
        adcValue=adc.read()
        voltage=adcValue/4095*3.3
        Rt=10*voltage/(3.3-voltage)
        tempK=(1/(1/(273.15+25)+(math.log(Rt/10))/3950))
        tempC=tempK-273.15
        print("ADC value:",adcValue,"\tVoltage :",voltage,"\tTemperature :",tempC);
        time.sleep_ms(1000)
except:
    pass
```

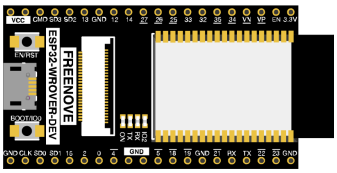
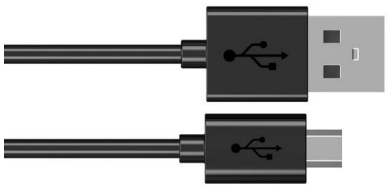
In the code, the ADC value of ADC module A0 port is read, and then it calculates the voltage and the resistance of Thermistor according to Ohms Law. Finally, it calculates the temperature sensed by the Thermistor, according to the formula.

Chapter 13 Bluetooth

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

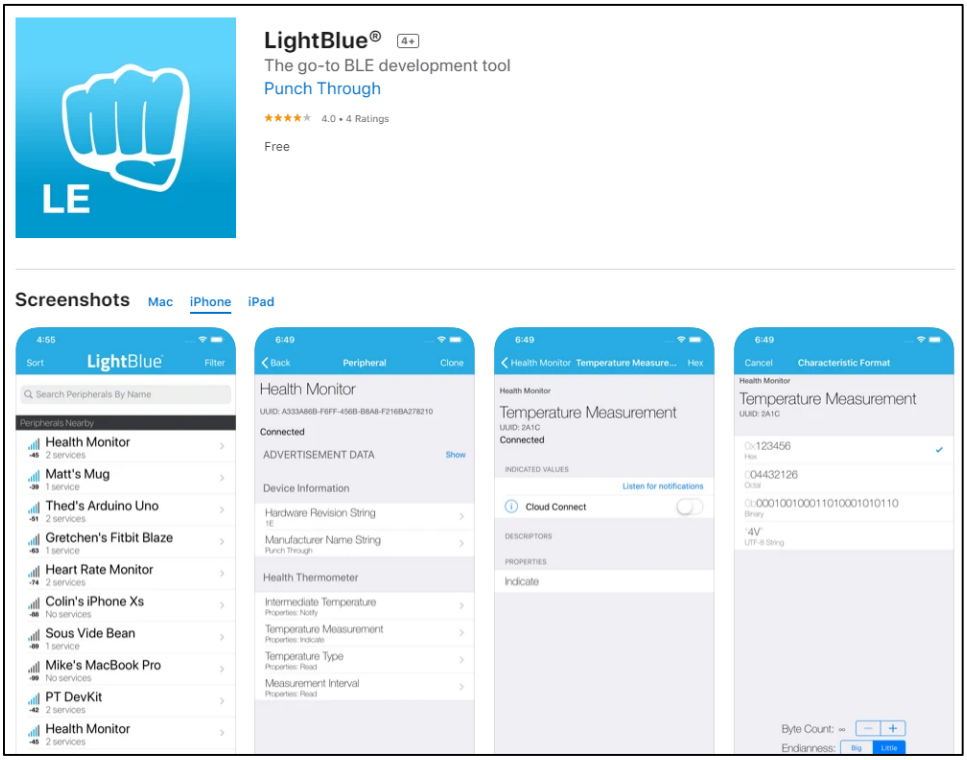
Project 13.1 Bluetooth Low Energy Data Passthrough

Component List

ESP32-WROVER x1 	Micro USB Wire x1 

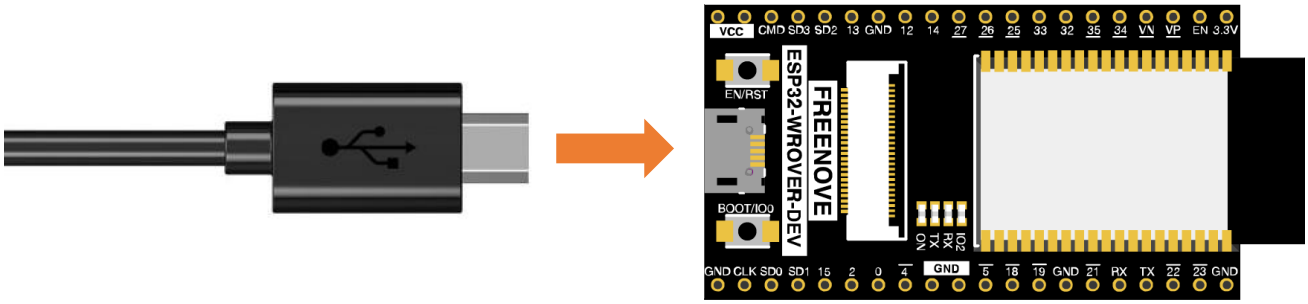
Lightblue

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



Circuit

Connect Freenove ESP32 to the computer using the USB cable.

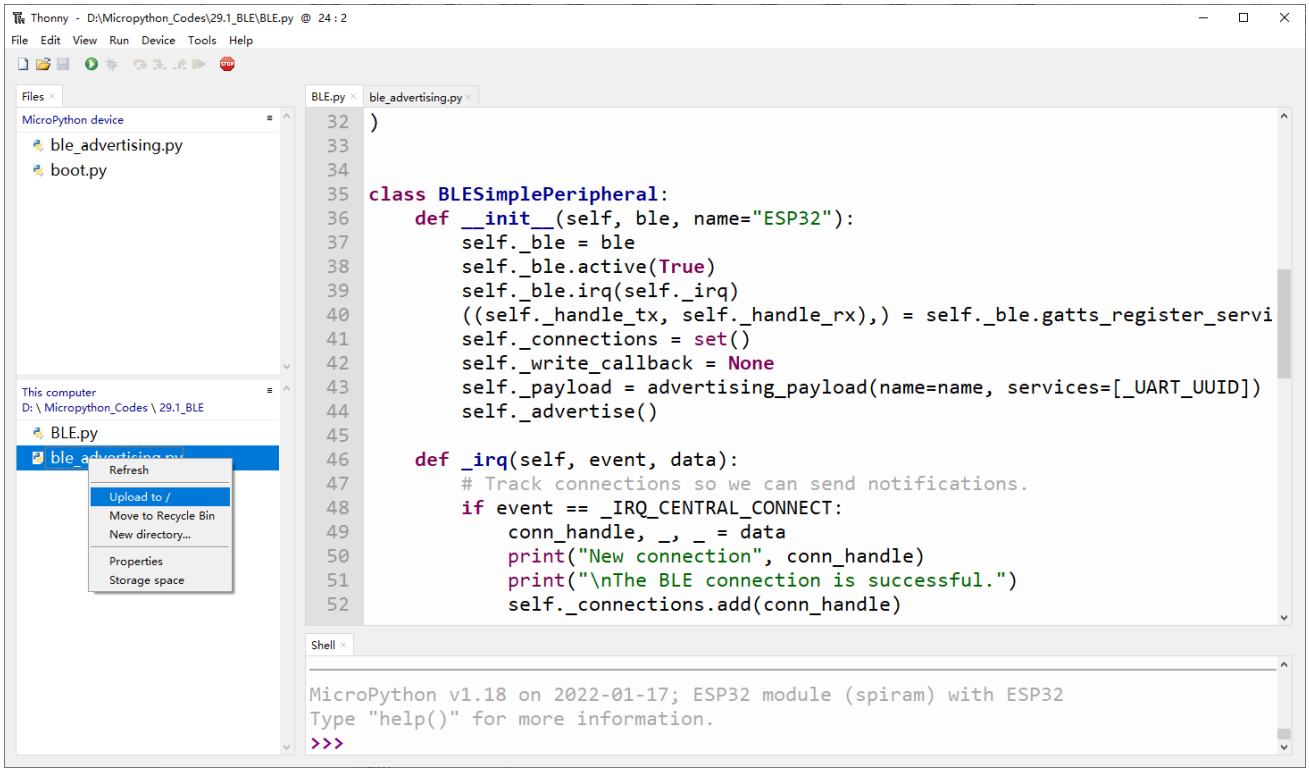


Code

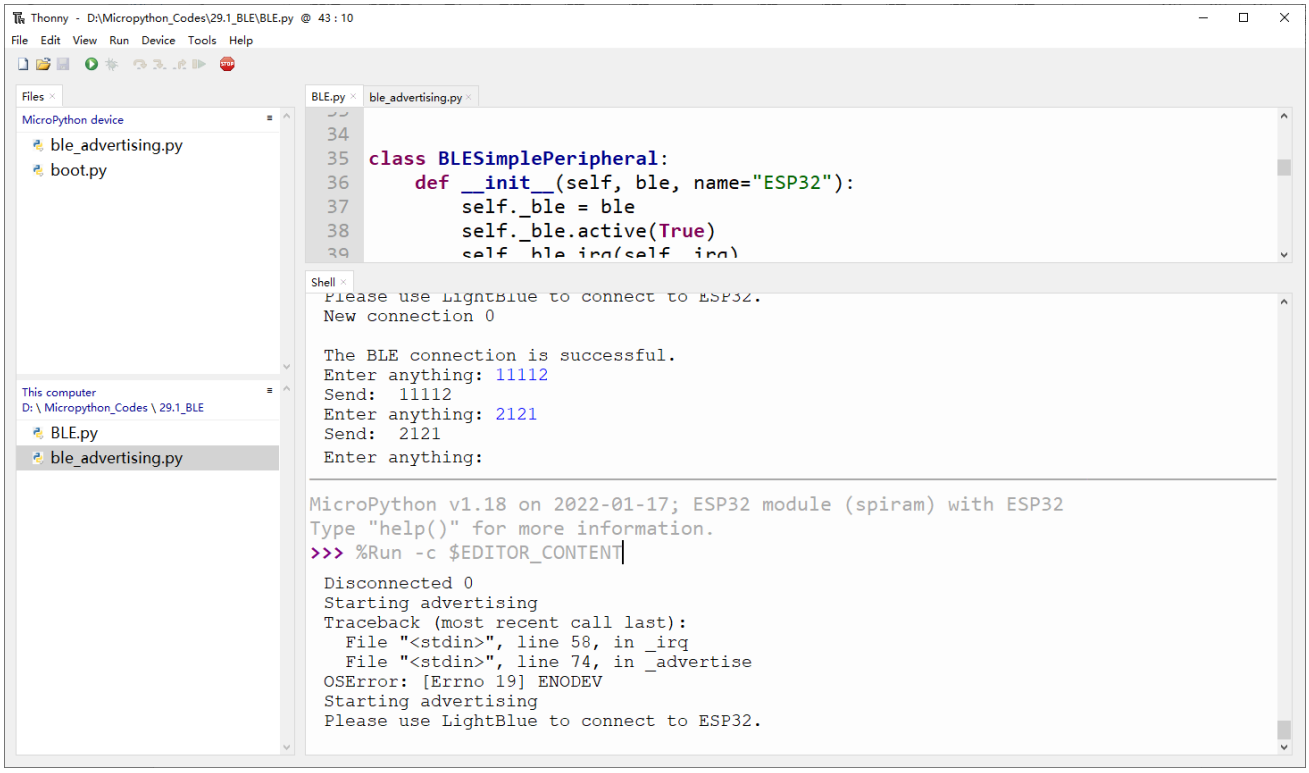
Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "13.1_BLE". Select "ble_advertising.py", right click your mouse to select "Upload to /", wait for "ble_advertising.py" to be uploaded to ESP32-WROVER and then double click "BLE.py".

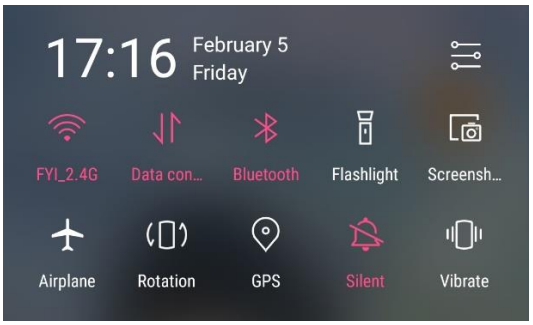
13.1_BLE



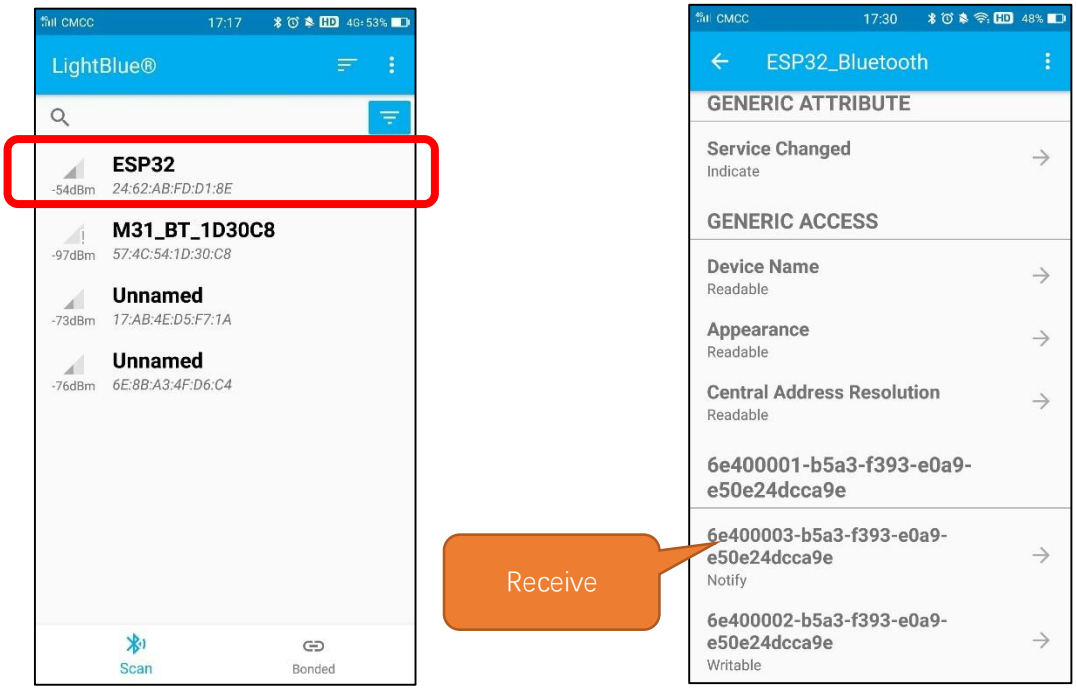
Click run for BLE.py.



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



In the Scan page, swipe down to refresh the name of Bluetooth that the phone searches for. Click ESP32.

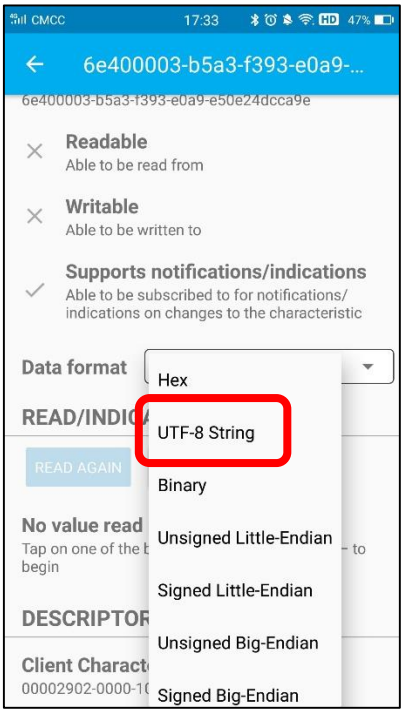


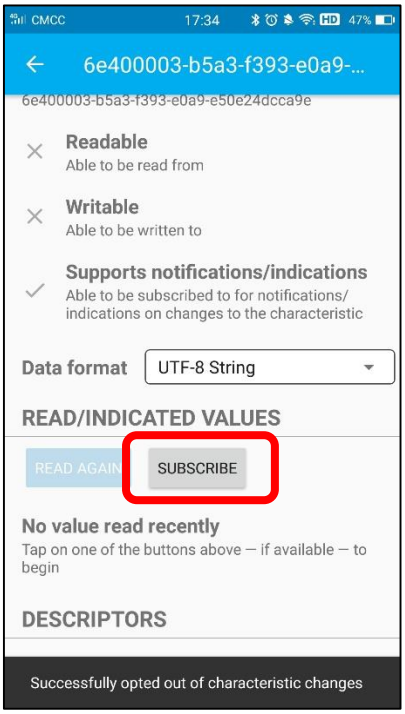
After Bluetooth is connect successfully, Shell will printer the information.

```
MicroPython v1.18 on 2022-01-17; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
 Disconnected 0
 Starting advertising
 Traceback (most recent call last):
   File "<stdin>", line 58, in _irq
   File "<stdin>", line 74, in _advertise
 OSError: [Errno 19] ENODEV
 Starting advertising
 Please use LightBlue to connect to ESP32.
 New connection 0

 The BLE connection is successful.
 Enter anything:
```

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.



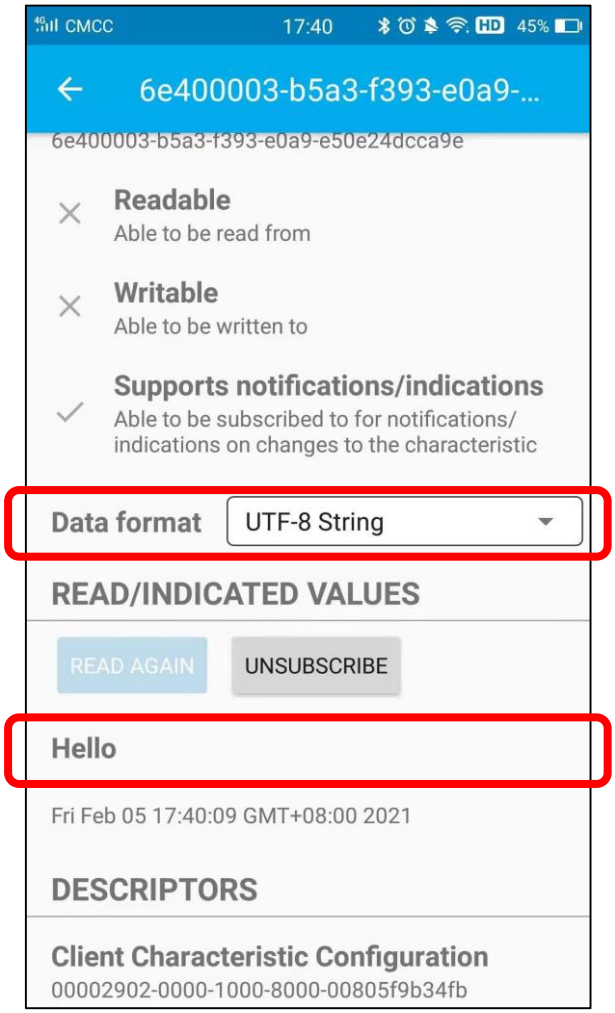


You can type "Hello" in Shell and press "Enter" to send.

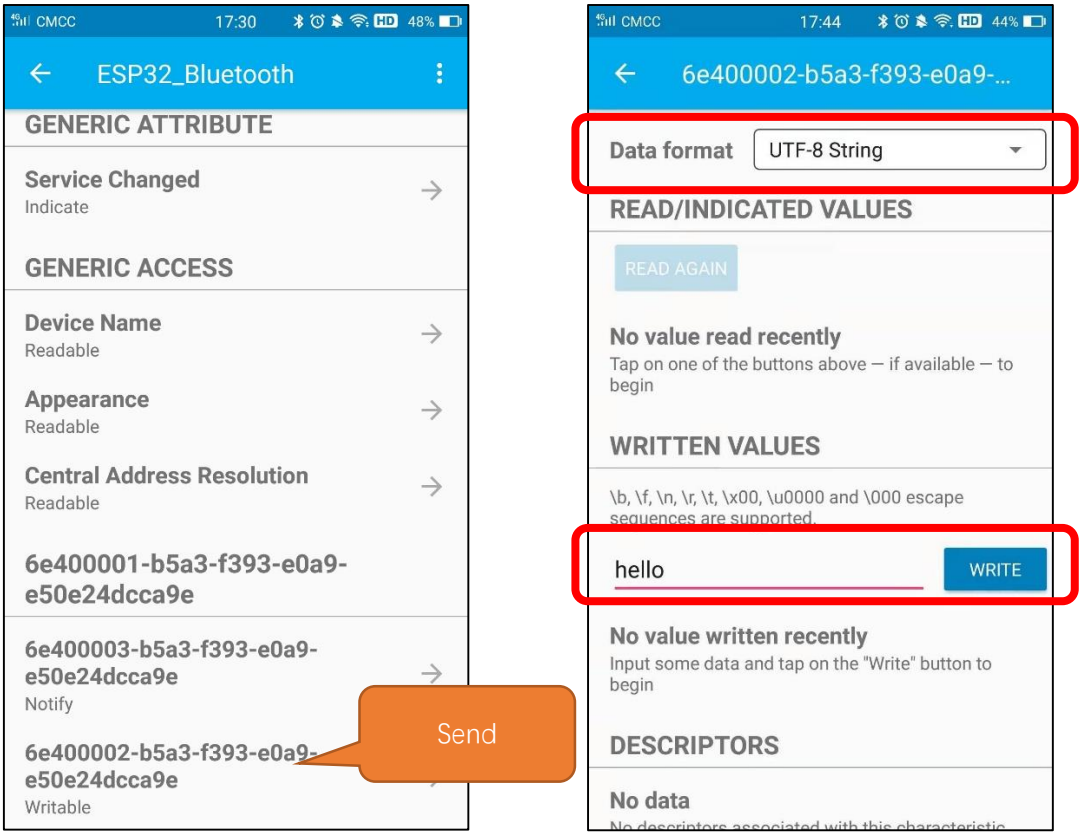
```
MicroPython v1.18 on 2022-01-17; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
 Disconnected 0
 Starting advertising
 Traceback (most recent call last):
   File "<stdin>", line 58, in _irq
   File "<stdin>", line 74, in _advertise
 OSError: [Errno 19] ENODEV
 Starting advertising
 Please use LightBlue to connect to ESP32.
 New connection 0

 The BLE connection is successful.
 Enter anything: Hello
 Send:  Hello
 Enter anything:
```

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".

```
MicroPython v1.18 on 2022-01-17; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
 Disconnected 0
 Starting advertising
 Traceback (most recent call last):
   File "<stdin>", line 58, in _irq
   File "<stdin>", line 74, in _advertise
 OSError: [Errno 19] ENODEV
 Starting advertising
 Please use LightBlue to connect to ESP32.
 New connection 0

 The BLE connection is successful.
 Enter anything: Hello
 Send:  Hello
 Enter anything: RX b'hello'
```

And now data can be transferred between your mobile phone and computer via ESP32-WROVER.
The following is the program code:

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

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

_UART_UUID = bluetooth.UUID("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")
_UART_TX = (
    bluetooth.UUID("6E400003-B5A3-F393-E0A9-E50E24DCCA9E"),
    _FLAG_READ | _FLAG_NOTIFY,
)
_UART_RX = (
    bluetooth.UUID("6E400002-B5A3-F393-E0A9-E50E24DCCA9E"),
    _FLAG_WRITE | _FLAG_WRITE_NO_RESPONSE,
)
_UART_SERVICE = (
    _UART_UUID,
    (_UART_TX, _UART_RX),
)
class BLESimplePeripheral:
    def __init__(self, ble, name="ESP32"):
        self._ble = ble
        self._ble.active(True)
        self._ble.irq(self._irq)
        ((self._handle_tx, self._handle_rx),) =
self._ble.gatts_register_services((_UART_SERVICE,))
        self._connections = set()
        self._write_callback = None
        self._payload = advertising_payload(name=name, services=[_UART_UUID])
        self._advertise()
    def _irq(self, event, data):
        # Track connections so we can send notifications.
        if event == _IRQ_CENTRAL_CONNECT:
```

```
            conn_handle, _, _ = data
            print("New connection", conn_handle)
            print("\nThe BLE connection is successful.")
            self._connections.add(conn_handle)
        elif event == _IRQ_CENTRAL_DISCONNECT:
            conn_handle, _, _ = data
            print("Disconnected", conn_handle)
            self._connections.remove(conn_handle)
            # Start advertising again to allow a new connection.
            self._advertise()
        elif event == _IRQ_GATTS_WRITE:
            conn_handle, value_handle = data
            value = self._ble.gatts_read(value_handle)
            if value_handle == self._handle_rx and self._write_callback:
                self._write_callback(value)
    def send(self, data):
        for conn_handle in self._connections:
            self._ble.gatts_notify(conn_handle, self._handle_tx, data)
    def is_connected(self):
        return len(self._connections) > 0
    def _advertise(self, interval_us=500000):
        print("Starting advertising")
        self._ble.gap_advertise(interval_us, adv_data=self._payload)
    def on_write(self, callback):
        self._write_callback = callback
def demo():
    ble = bluetooth.BLE()
    p = BLESimplePeripheral(ble)
    def on_rx(rx_data):
        print("RX", rx_data)
    p.on_write(on_rx)
    print("Please use LightBlue to connect to ESP32.")
    while True:
        if p.is_connected():
            # Short burst of queued notifications.
            tx_data = input("Enter anything: ")
            print("Send: ", tx_data)
            p.send(tx_data)
if __name__ == "__main__":
    demo()
```

Define the specified UUID number for BLE vendor.

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

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

Write an _irq function to manage BLE interrupt events.

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

Initialize the BLE function and name it.

```
def __init__(self, ble, name="ESP32"):
```

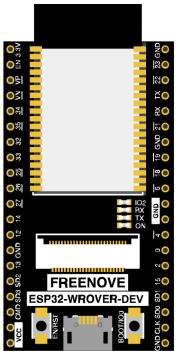
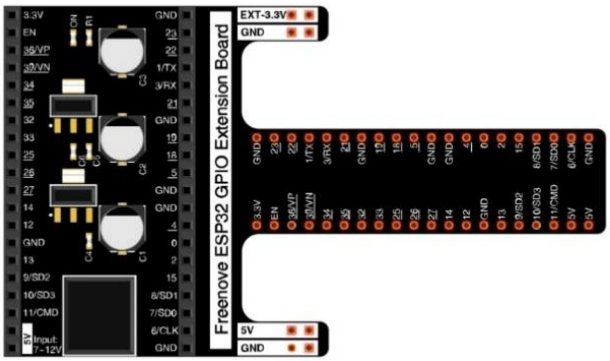
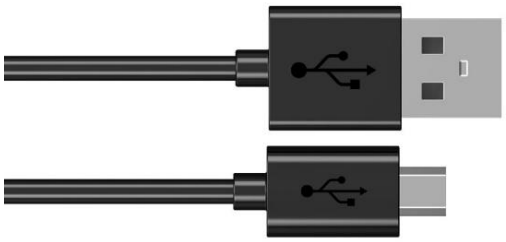
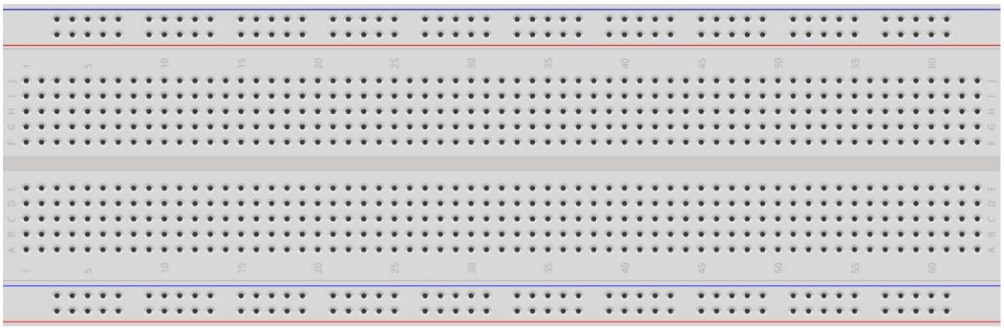
When the mobile phone send data to ESP32 via BLE Bluetooth, it will print them out with serial port; When the serial port of ESP32 receive data, it will send them to mobile via BLE Bluetooth.

```
  def demo():
    ble = bluetooth.BLE()
    p = BLESimplePeripheral(ble)
    def on_rx(rx_data):
        print("RX", rx_data)
    p.on_write(on_rx)
    print("Please use LightBlue to connect to ESP32.")
    while True:
        if p.is_connected():
            # Short burst of queued notifications.
            tx_data = input("Enter anything: ")
            print("Send: ", tx_data)
            p.send(tx_data)
    lastMsg = now;
  }
```

Project 13.2 Bluetooth Control LED

In this section, we will control the LED with Bluetooth.

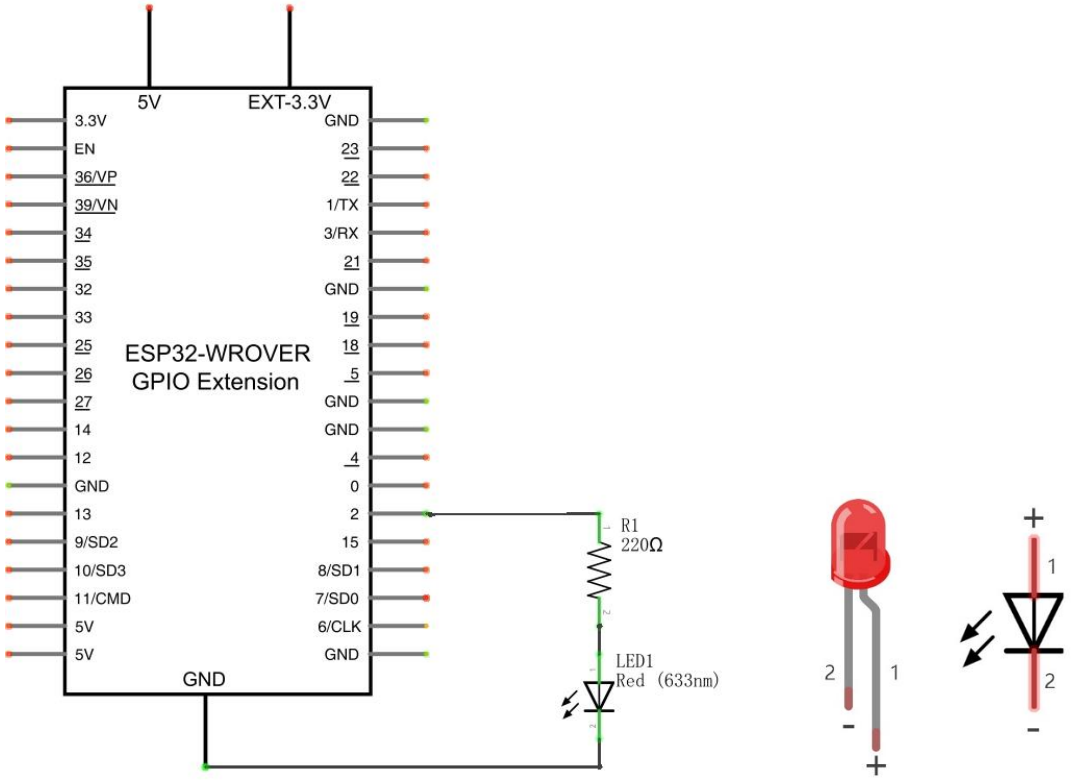
Component List

ESP32-WROVER x1 	GPIO Extension Board x1 		
Micro USB Wire x1 	LED x1	Resistor 220Ω x1	Jumper M/M x2
Breadboard x1 			

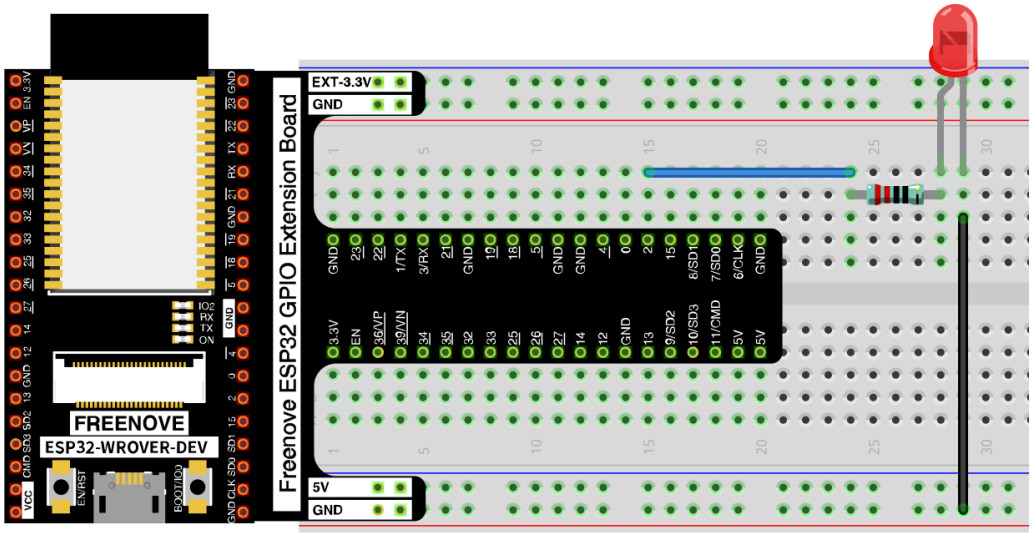
Circuit

Connect Freenove ESP32 to the computer using a USB cable.

Schematic diagram



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

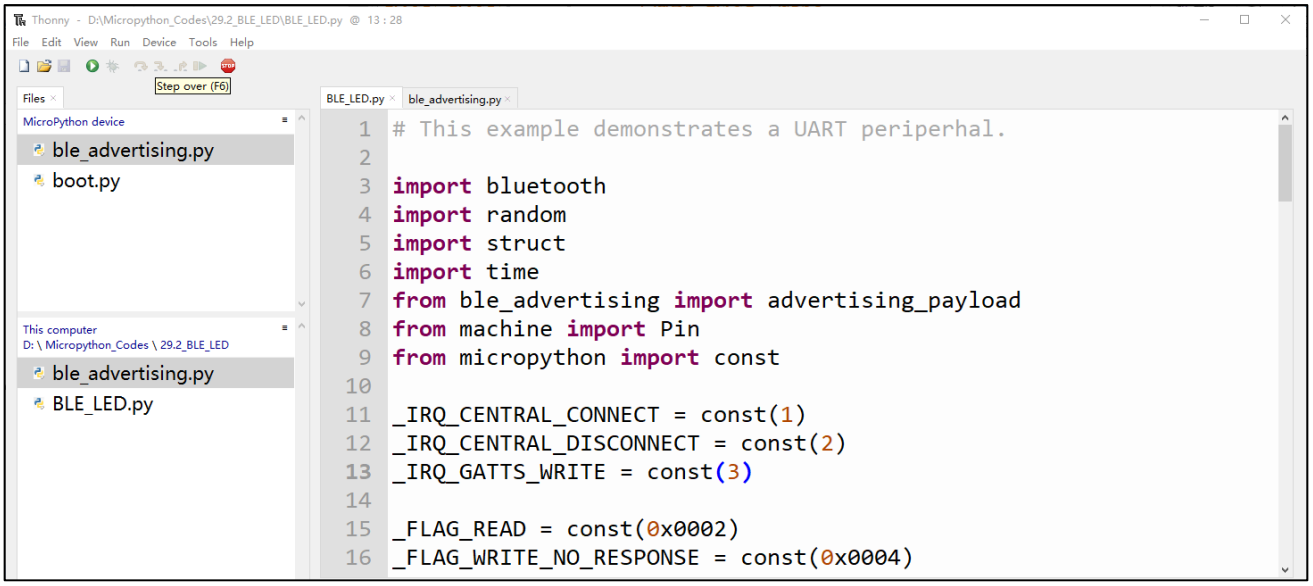


Code

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

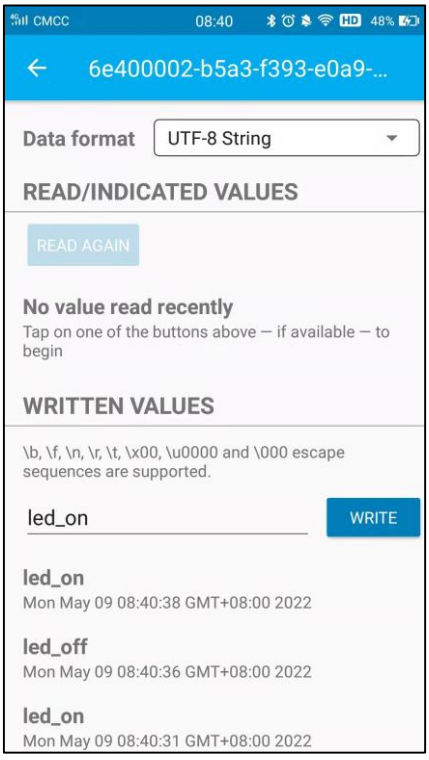
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "13.2_ BLE_LED". Select "ble_advertising.py", right click your mouse to select "Upload to /", wait for "ble_advertising.py" to be uploaded to ESP32-WROVER and then double click "BLE_LED.py".

13.2_BLE_LED



Compile and upload code to ESP32. The operation of the APP is the same as 13.1, you only need to change the sending content to "led_on" and "led_off" to operate LEDs on the ESP32-WROVER.

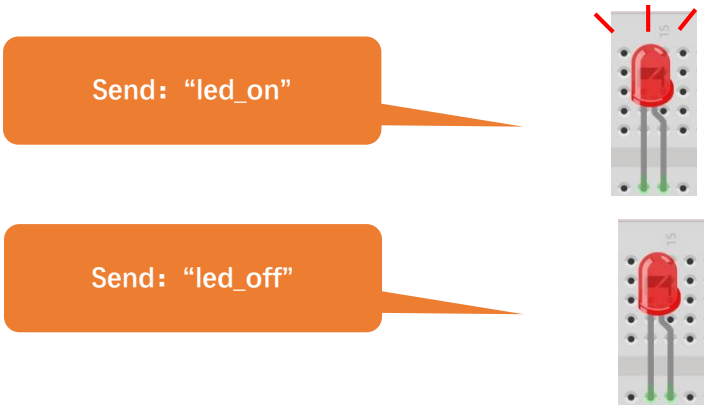
Data sent from mobile APP:



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

```
Shell
The BLE connection is successful.
Received:  b'led_on'
Received:  b'led_off'
Received:  b'led_on'
```

The phenomenon of LED



Attention: If the sending content isn't "led_on' or "led_off", then the state of LED will not change. If the LED is on, when receiving irrelevant content, it keeps on; Correspondingly, if the LED is off, when receiving irrelevant content, it keeps off.

The following is the program code:

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

```
)
_UART_RX = (
    bluetooth.UUID("6E400002-B5A3-F393-E0A9-E50E24DCCA9E"),
    _FLAG_WRITE | _FLAG_WRITE_NO_RESPONSE,
)
_UART_SERVICE = (
    _UART_UUID,
    (_UART_TX, _UART_RX),
)
class BLESimplePeripheral:
    def __init__(self, ble, name="ESP32"):
        self._ble = ble
        self._ble.active(True)
        self._ble.irq(self._irq)
        ((self._handle_tx, self._handle_rx),) =
self._ble.gatts_register_services((_UART_SERVICE,))
        self._connections = set()
        self._write_callback = None
        self._payload = advertising_payload(name=name, services=[_UART_UUID])
        self._advertise()
    def _irq(self, event, data):
        # Track connections so we can send notifications.
        if event == _IRQ_CENTRAL_CONNECT:
            conn_handle, _, _ = data
            print("New connection", conn_handle)
            print("\nThe BLE connection is successful.")
            self._connections.add(conn_handle)
        elif event == _IRQ_CENTRAL_DISCONNECT:
            conn_handle, _, _ = data
            print("Disconnected", conn_handle)
            self._connections.remove(conn_handle)
            # Start advertising again to allow a new connection.
            self._advertise()
        elif event == _IRQ_GATTS_WRITE:
            conn_handle, value_handle = data
            value = self._ble.gatts_read(value_handle)
            if value_handle == self._handle_rx and self._write_callback:
                self._write_callback(value)
    def send(self, data):
        for conn_handle in self._connections:
            self._ble.gatts_notify(conn_handle, self._handle_tx, data)
    def is_connected(self):
        return len(self._connections) > 0
    def _advertise(self, interval_us=500000):
```

```
66          print("Starting advertising")
67          self._ble.gap_advertise(interval_us, adv_data=self._payload)
68      def on_write(self, callback):
69          self._write_callback = callback
70  def demo():
71      ble = bluetooth.BLE()
72      p = BLESimplePeripheral(ble)
73      led=Pin(2,Pin.OUT)
74      def on_rx(rx_data):
75          print("Received: ", rx_data)
76          if rx_data == b'led_on':
77              led.value(1)
78          elif rx_data == b'led_off':
79              led.value(0)
80          else:
81              pass
82      p.on_write(on_rx)
83      print("Please use LightBlue to connect to ESP32.")
84  if __name__ == "__main__":
85      demo()
```

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

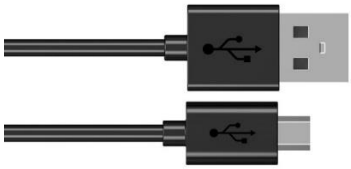
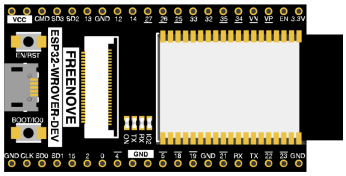
```
76          if rx_data == b'led_on':
77              led.value(1)
78          elif rx_data == b'led_off':
79              led.value(0)
```

Chapter 14 WiFi Working Modes

In this chapter, we'll focus on the WiFi infrastructure for ESP32-WROVER.
ESP32-WROVER 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 14.1 Station mode

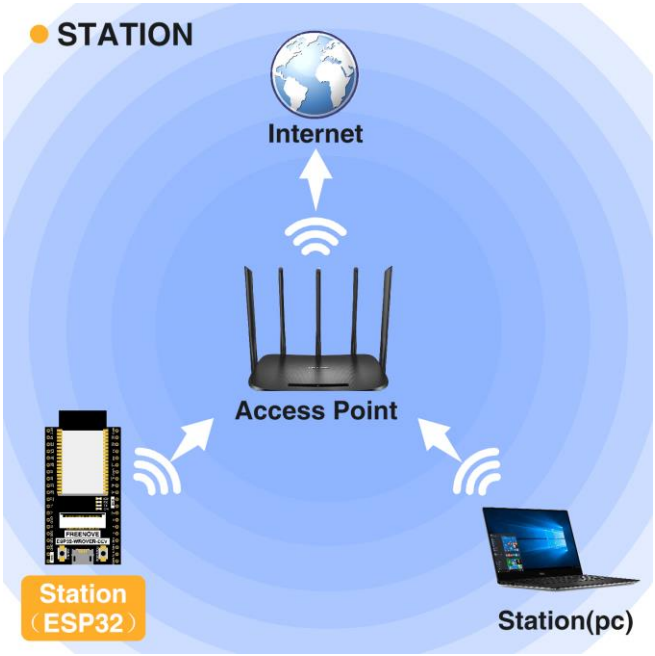
Component List

Micro USB Wire x1 	ESP32-WROVER x1 

Component knowledge

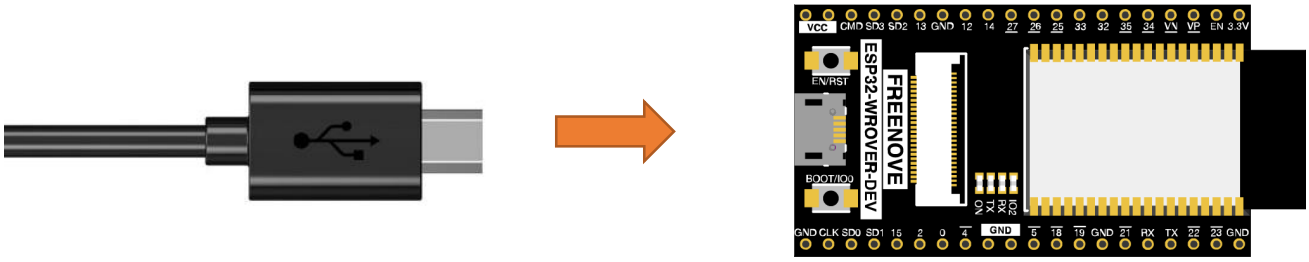
Station mode

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



Circuit

Connect Freenove ESP32 to the computer using the USB cable.

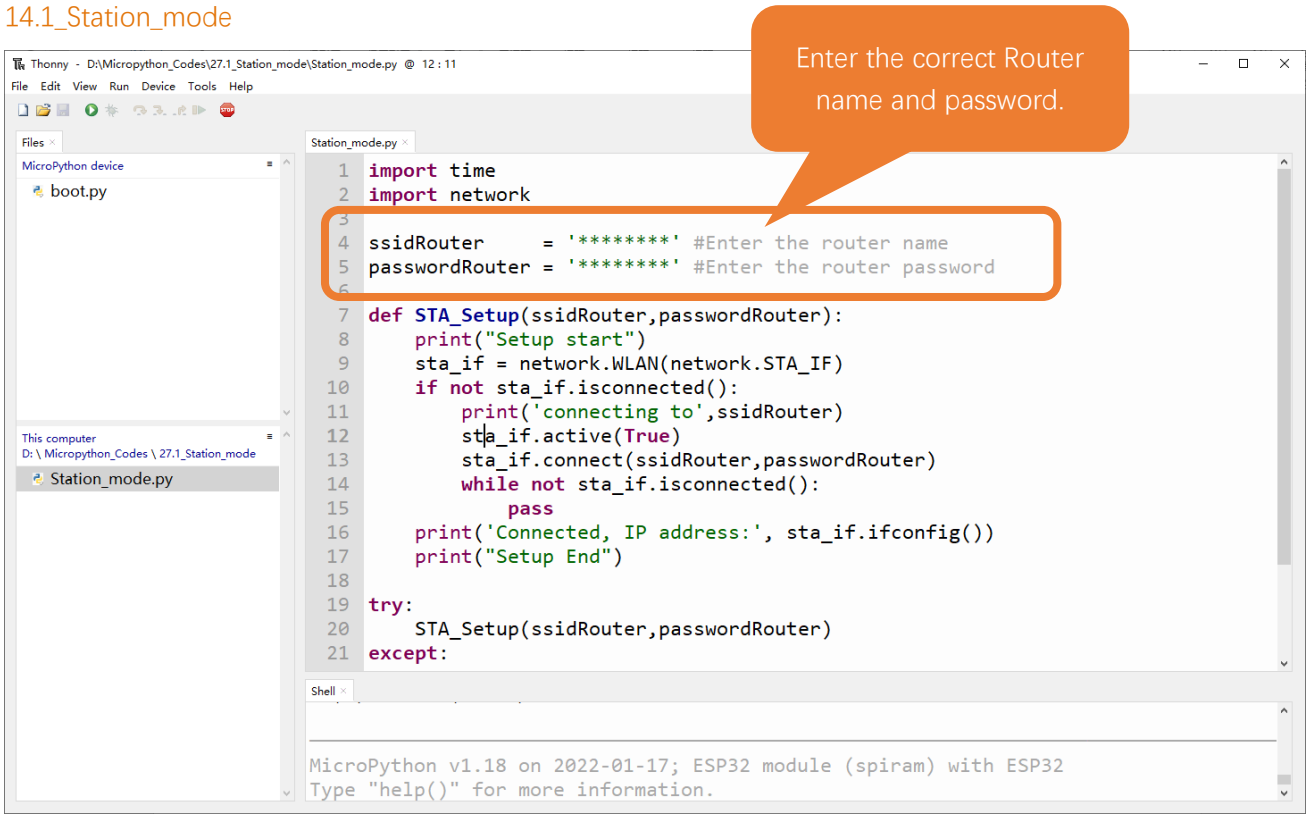


Code

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "14.1_Station_mode" and double click "Station_mode.py".

14.1_Station_mode



Because the names and passwords of routers in various places are different, before the Code runs, users need to enter the correct router's name and password in the box as shown in the illustration above.
After making sure the router name and password are entered correctly, compile and upload codes to ESP32-WROVER, wait for ESP32 to connect to your router and print the IP address assigned by the router to ESP32 in "Shell".

```
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

  Setup start
  connecting to FYI_2.4G
  I (4155528) phy: phy_version: 4102, 2fa7a43, Jul 15 2019, 13:06:06,
  0, 2
  Connected, IP address: ('192.168.1.102', '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 in Station mode.

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

Activate ESP32's Station mode, initiate a connection request to the router and enter the password to connect.

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

Wait for ESP32 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-WROVER 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 is connected to AP in Station mode. In STA mode, it returns True if it is connected to a WiFi access point and has a valid IP address; Otherwise it returns False.

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

ssid: WiFiname

password: WiFipassword

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

Project 14.2 AP mode

Component List & Circuit

Component List & Circuit are the same as in Section 14.1.

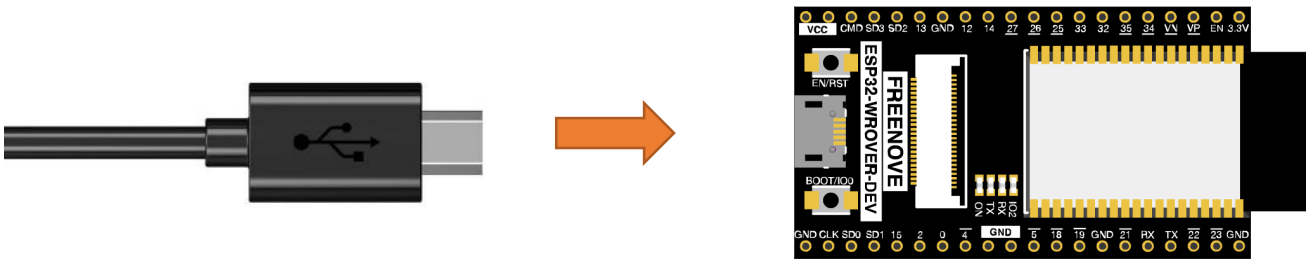
Component knowledge

AP mode

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

Circuit

Connect Freenove ESP32 to the computer using the USB cable.



Code

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "14.2_AP_mode". and double click "AP_mode.py".

14.2_AP_mode

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

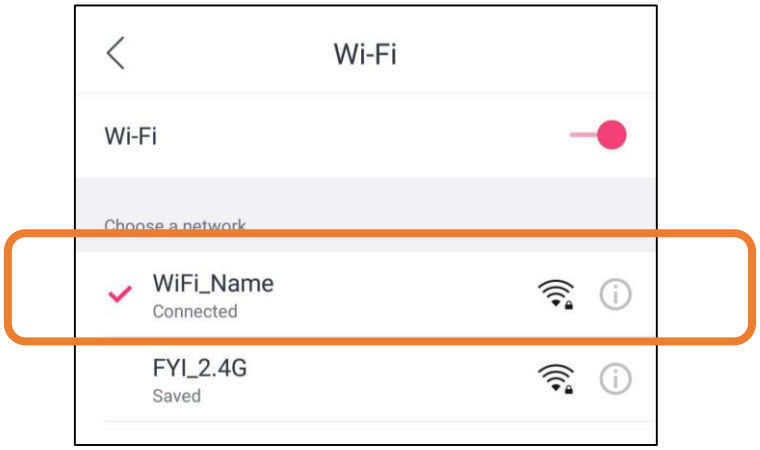
Click "Run current script", open the AP function of ESP32 and print the access point information.

```
Shell
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> %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


>>>
```

Turn on the WiFi scanning function of your phone, and you can see the ssid_AP on ESP32, which is called "WiFi_Name" in this Code. You can enter the password "12345678" to connect it or change its AP name and password by modifying Code.



The following is the program code:

```
import network

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

local_IP        = '192.168.1.10'
gateway         = '192.168.1.1'
subnet          = '255.255.255.0'
dns             = '8.8.8.8'

ap_if = network.WLAN(network.AP_IF)

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

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

Import network module.

```
import network
```

Enter correct AP name and password.

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

Set ESP32 in AP mode.

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

Configure IP address, gateway and subnet mask for ESP32.

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

Turn on an AP in ESP32, whose name is set by ssid_AP and password is set by password_AP.

```
16     ap_if.config(essid=ssidAP,authmode=network.AUTH_WPA_WPA2_PSK, password=passwordAP)
17     ap_if.active(True)
```

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

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

Reference

Class network

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

WLAN(interface_id): Set to WiFi mode.

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

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

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

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

connect(ssid, password): Connecting to wireless network

ssid: WiFiname

password: WiFipassword

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

ssid: WiFi account name

channel: WiFichannel

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

ip: IPaddress

subnet_mask: subnet mask

gateway: gateway

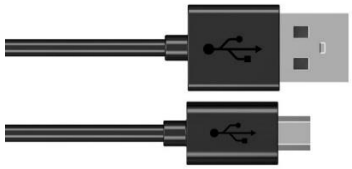
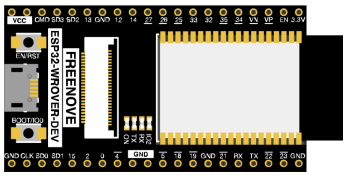
DNS_server: DNSserver

disconnect(): Disconnect from the currently connected wireless network

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

Project 14.3 AP+Station mode

Component List

Micro USB Wire x1	ESP32-WROVER x1
	

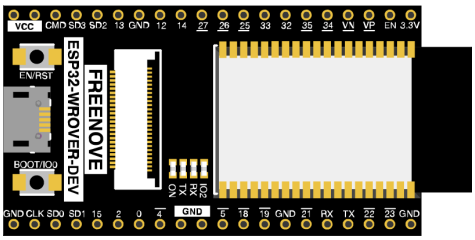
Component knowledge

AP+Station mode

In addition to AP mode and Station mode, ESP32 can also use AP mode and Station mode at the same time. This mode contains the functions of the previous two modes. Turn on ESP32'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 ESP32.

Circuit

Connect Freenove ESP32 to the computer using the USB cable.

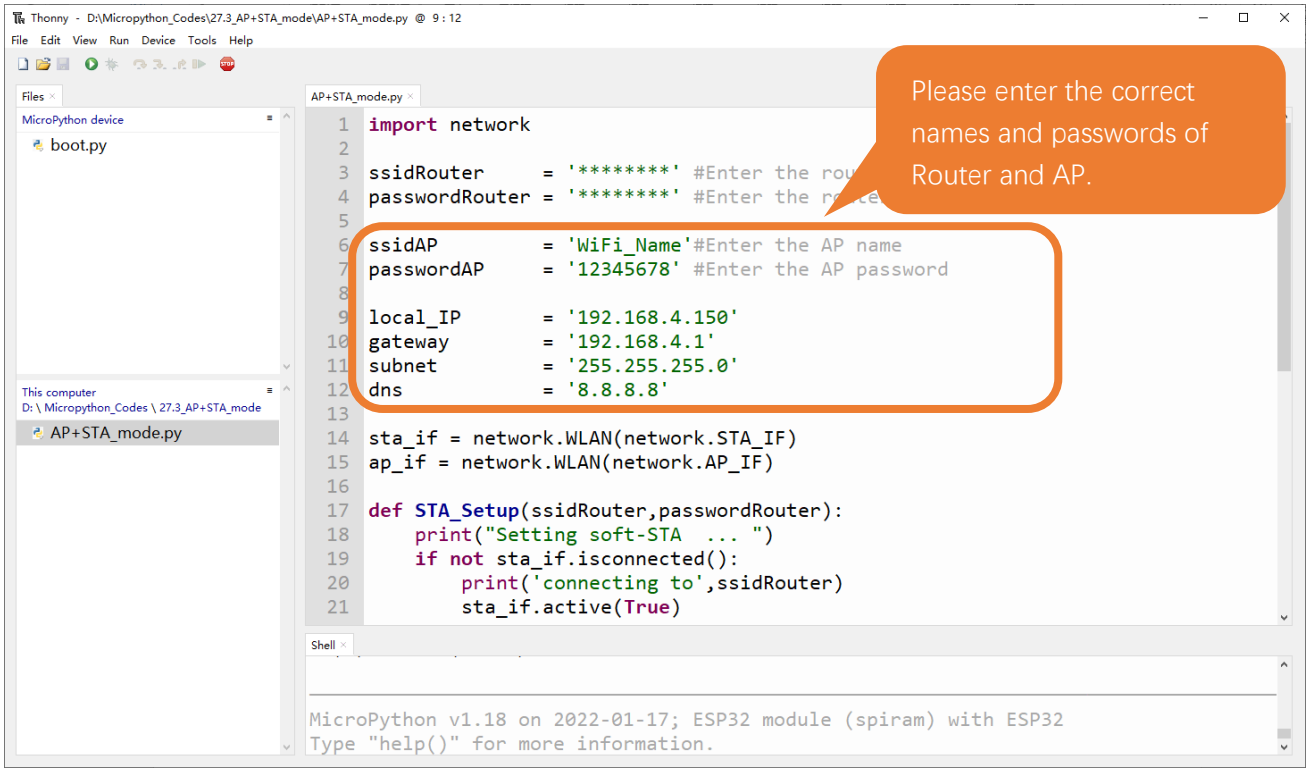


Code

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "14.3_AP+STA_mode"and double click "AP+STA_mode.py".

14.3_AP+STA_mode



It is analogous to Project 14.1 and Project 14.2. Before running the Code, you need to modify ssidRouter, passwordRouter, ssidAP and passwordAP shown in the box of the illustration above.

After making sure that the code is modified correctly, click "Run current script" and the "Shell" will display as follows:

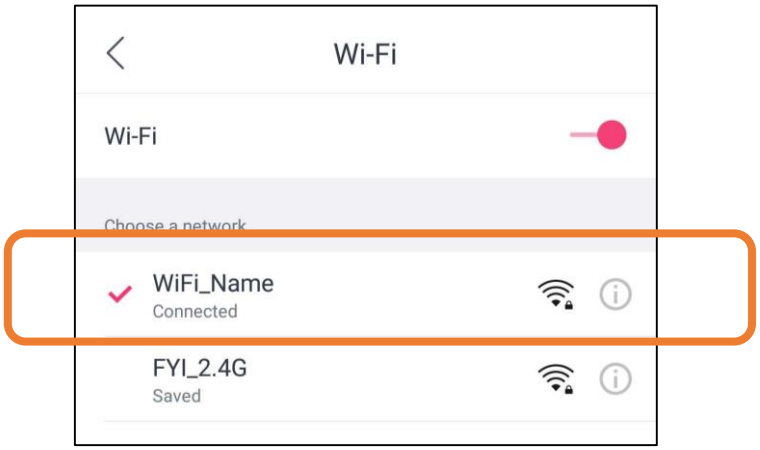
```
Shell
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

  Setting soft-AP  ...
  Success, IP address: ('192.168.4.150', '192.168.4.1', '255.255.255.0', '8.8.8.8')
  Setup End

  Setting soft-STA  ...
  Connected, IP address: ('192.168.1.102', '255.255.255.0', '192.168.1.1', '192.168.1.1')
  Setup End

>>>
```

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



The following is the program code:

```
import network

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

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

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

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

def STA_Setup(ssidRouter,passwordRouter):
    print("Setting soft-STA  ... ")
    if not sta_if.isconnected():
        print('connecting to',ssidRouter)
        sta_if.active(True)
        sta_if.connect(ssidRouter,passwordRouter)
        while not sta_if.isconnected():
            pass
    print('Connected, IP address:', sta_if.ifconfig())
    print("Setup End")

def AP_Setup(ssidAP,passwordAP):
    ap_if.ifconfig([local_IP,gateway,subnet,dns])
    print("Setting soft-AP  ... ")
```

```
31      ap_if.config(essid=ssidAP,authmode=network.AUTH_WPA_WPA2_PSK, password=passwordAP)
32      ap_if.active(True)
33      print('Success, IP address:', ap_if.ifconfig())
34      print("Setup End\n")
35
36  try:
37      AP_Setup(ssidAP,passwordAP)
38      STA_Setup(ssidRouter,passwordRouter)
39  except:
40      sta_if.disconnect()
41      ap_if.idsconnect()
```

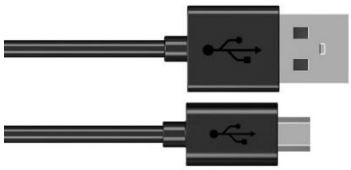
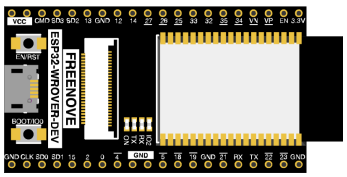
Chapter 15 TCP/IP

In this chapter, we wil introduce how ESP32 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 chaper.

Project 15.1 As Client

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

Component List

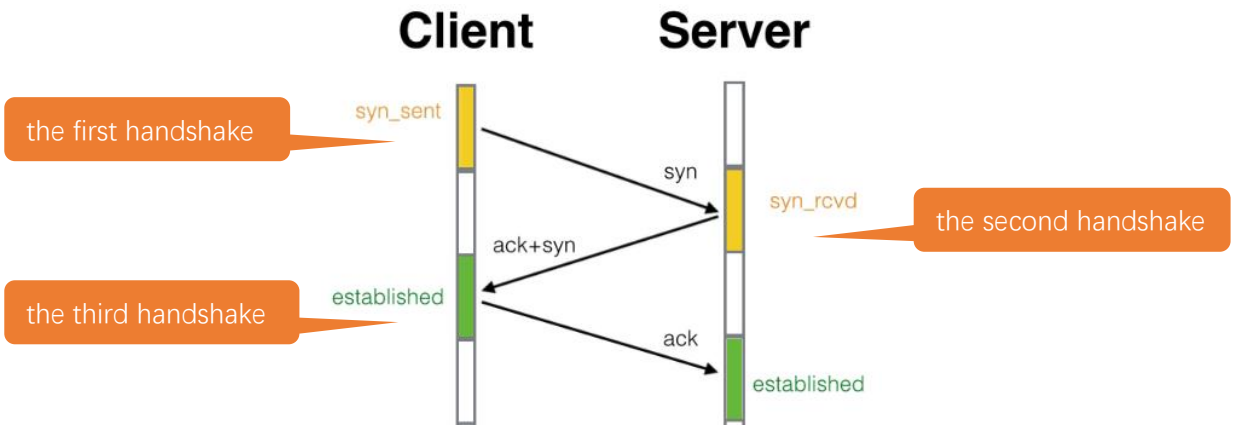
Micro USB Wire x1 	ESP32-WROVER x1 

Component knowledge

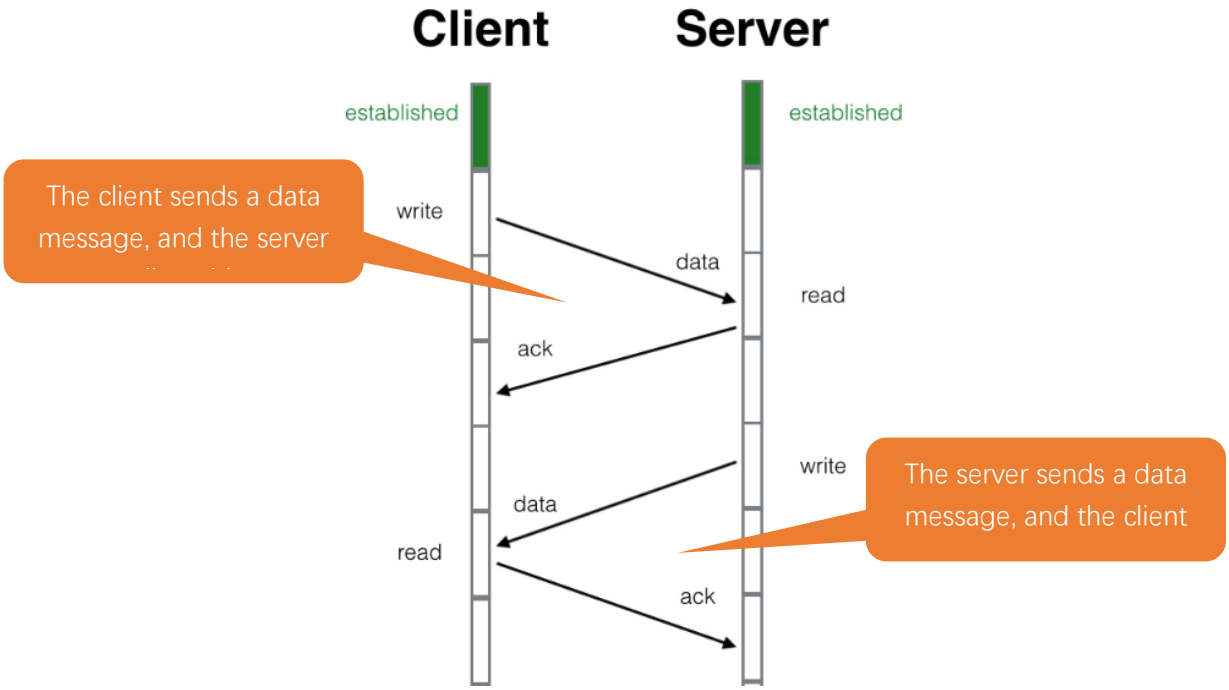
TCP connection

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

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



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



Install Processing

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

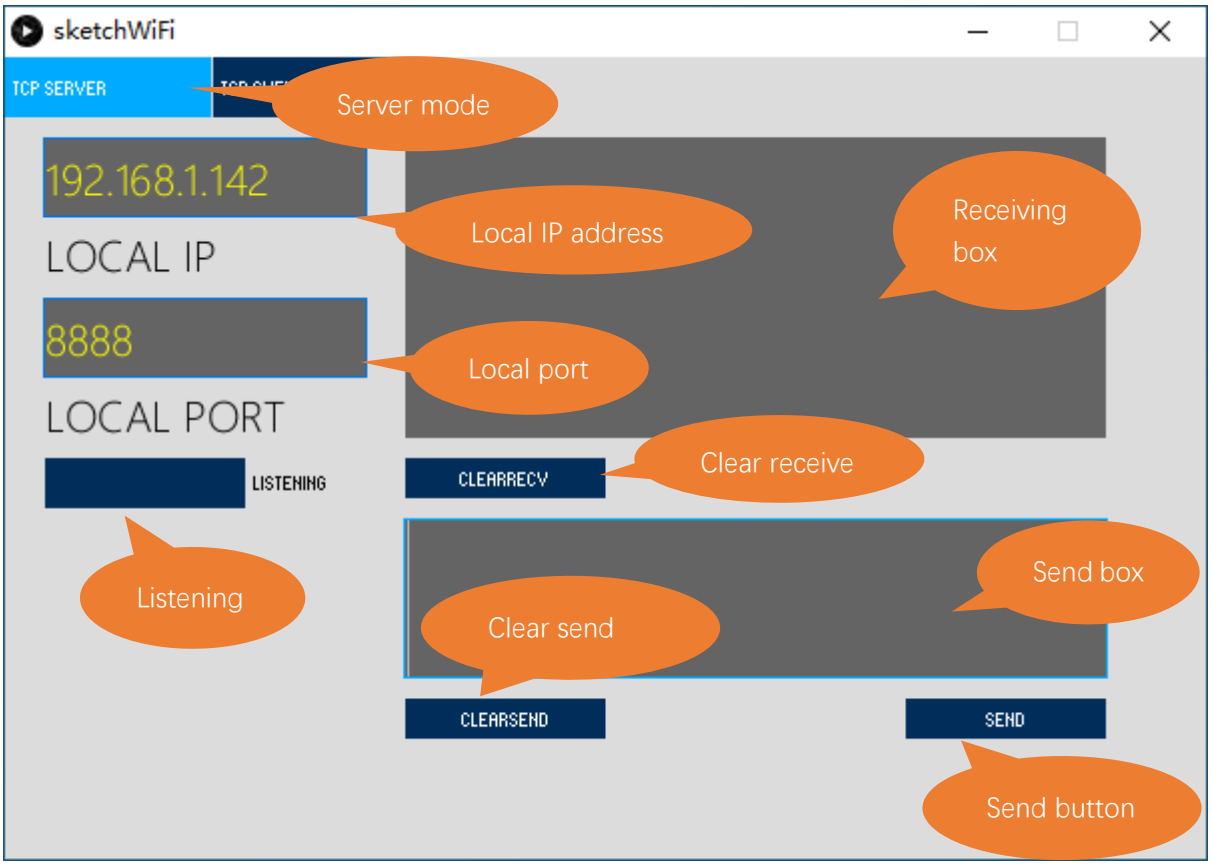
If you've not installed Processing, you can download it by clicking https://processing.org/download/. You can choose an appropriate version to download according to your PC system.

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

Use Server mode for communication

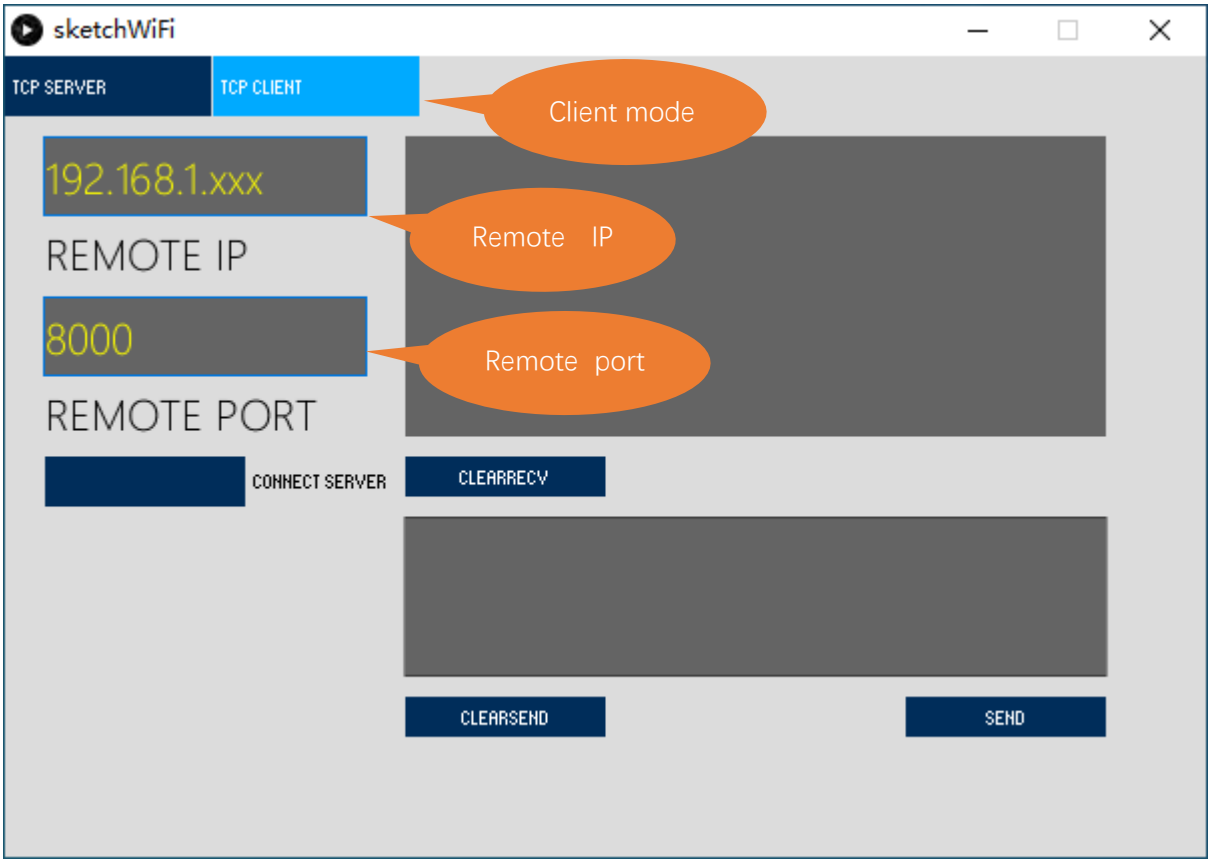
Open the "**Freenove_Basic_Starter_Kit_for_ESP32/Codes/Micropython_Codes/15.1_TCP_as_Client/sketchWiFi/sketchWiFi.pde**". Click "Run".

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



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

If ESP32 serves as Server, select TCP CLIENT mode for sketchWiFi.



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

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

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

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

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

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

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

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

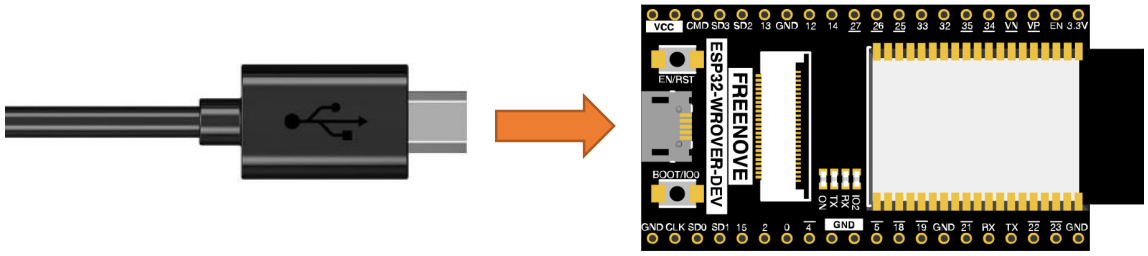
clear receive: clear out the content in the receiving text box

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

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

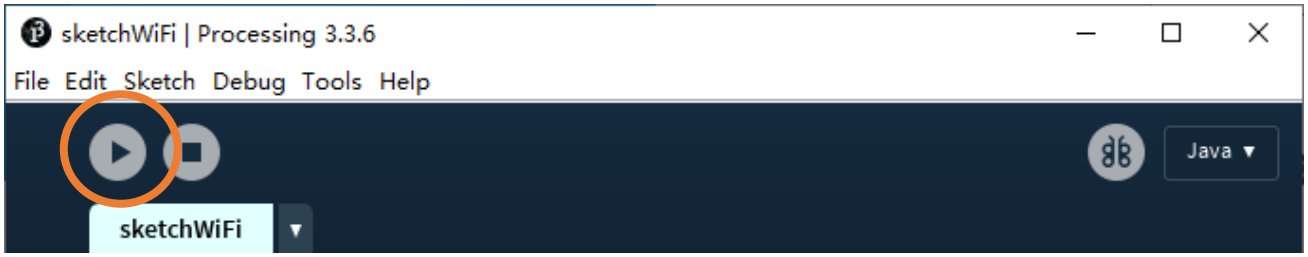
Circuit

Connect Freenove ESP32 to the computer using USB cable.

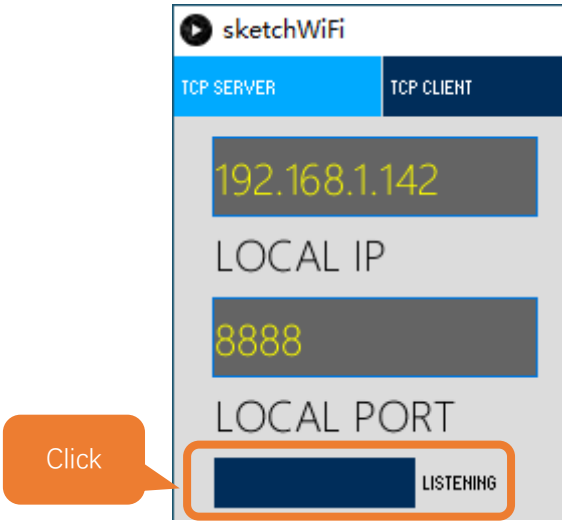


Code

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"。

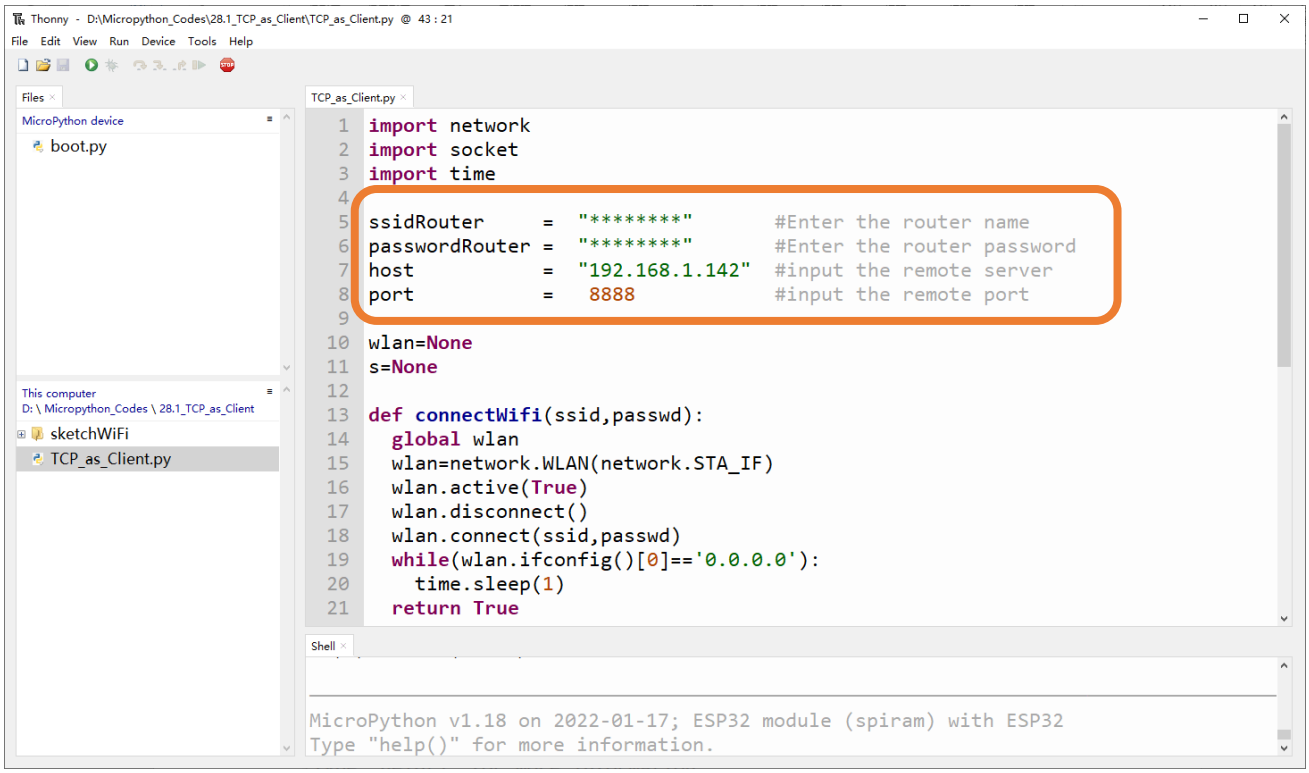


Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "15.1_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:

15.1_TCP_as_Client

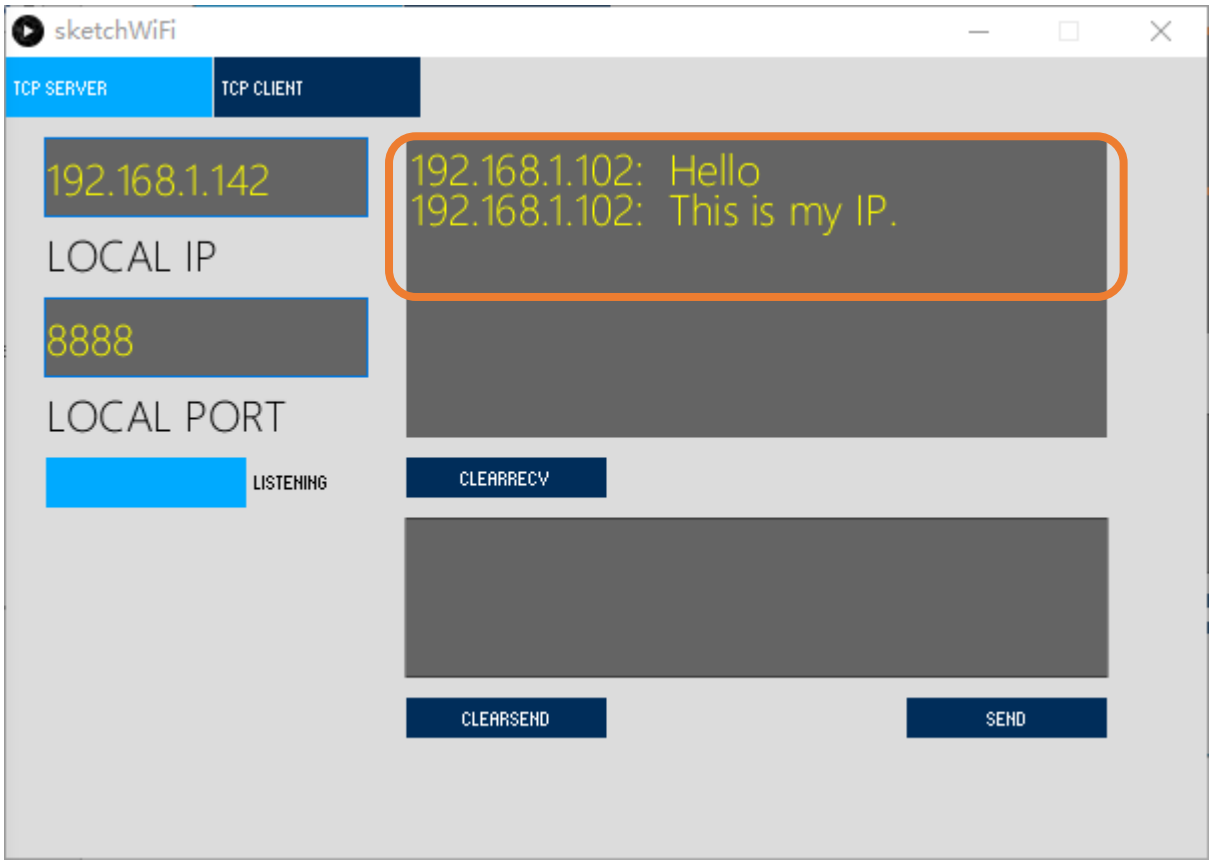


Click "Run current script" and in "Shell", you can see ESP32-WROVER automatically connects to sketchWiFi.

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

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

ESP32 connects with TCP SERVER, and TCP SERVER receives messages from ESP32, as shown in the figure below.



The following is the program code:

```
1  import network
2  import socket
3  import time
4
5  ssidRouter     = "********"    #Enter the router name
6  passwordRouter = "********"    #Enter the router password
7  host           = "********"    #input the remote server
8  port           =  8888         #input the remote port
9
10 wlan=None
11 s=None
```

```
12
13  def connectWifi(ssid,passwd):
14    global wlan
15    wlan= network.WLAN(network.STA_IF)
16    wlan.active(True)
17    wlan.disconnect()
18    wlan.connect(ssid,passwd)
19    while(wlan.ifconfig()[0]=='0.0.0.0'):
20      time.sleep(1)
21    return True
22  try:
23    connectWifi(ssidRouter,passwordRouter)
24    s = socket.socket()
25    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
26    s.connect((host,port))
27    print("TCP Connected to:", host, ":", port)
28    s.send('Hello')
29    s.send('This is my IP.')
30    while True:
31      data = s.recv(1024)
32      if(len(data) == 0):
33        print("Close socket")
34        s.close()
35        break
36      print(data)
37      ret=s.send(data)
38  except:
39    print("TCP close, please reset!")
40    if (s):
41      s.close()
42    wlan.disconnect()
43    wlan.active(False)
```

Import network、socket、time modules.

```
1  import network
2  import socket
3  import time
```

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

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

Connect specified Router until it is successful.

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

Connect router and then connect it to remote server.

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

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

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

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

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

Reference

Class socket

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

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

af: address

socket.AF_INET: IPv4

socket.AF_INET6: IPv6

type: type

socket.SOCK_STREAM : TCP stream

socket.SOCK_DGRAM : UDP datagram

socket.SOCK_RAW : Original socket

socket.SO_REUSEADDR : socket reusable

proto: protocol number

socket.IPPROTO_TCP: TCPmode

socket.IPPROTO_UDP: UDPmode

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

Level: Level of socket option

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

optname: Options of socket

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

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

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

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

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

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

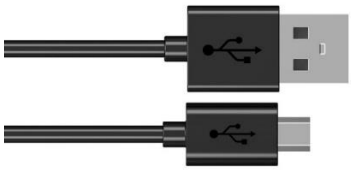
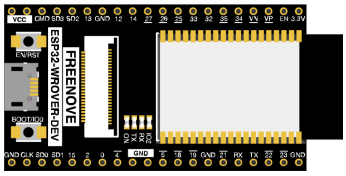
close(): Close socket.

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

Project 15.2 As Server

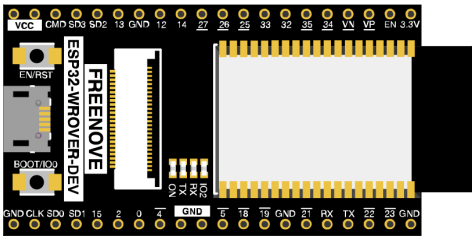
In this section, ESP32 is used as a Server to wait for the connection and communication with Client on the same LAN.

Component List

Micro USB Wire x1 	ESP32-WROVER x1 

Circuit

Connect Freenove ESP32 to the computer using the USB cable.



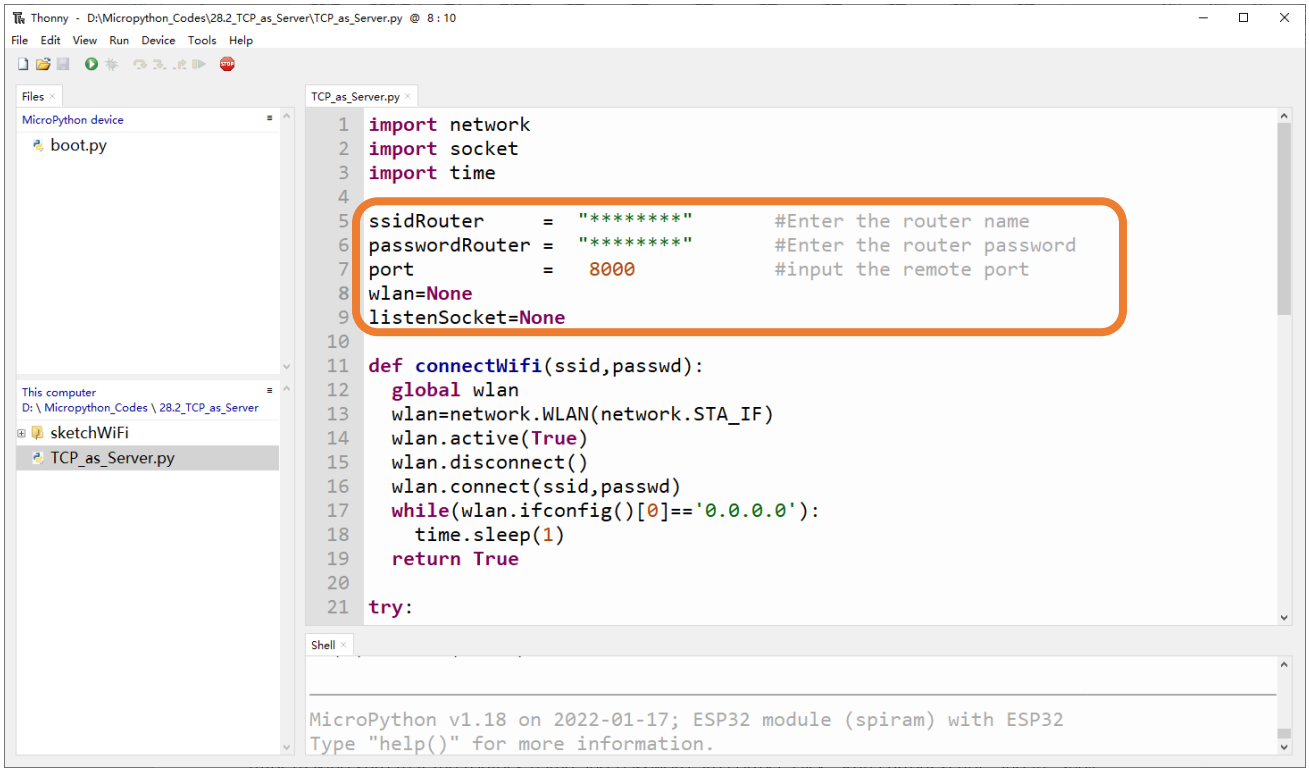
Code

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

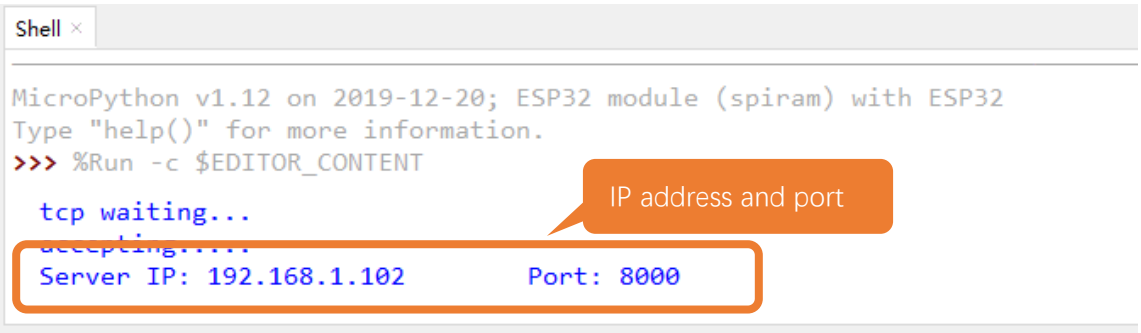
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "15.2_TCP_as_Server" and double click "TCP_as_Server.py".

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

15.2_TCP_as_Server



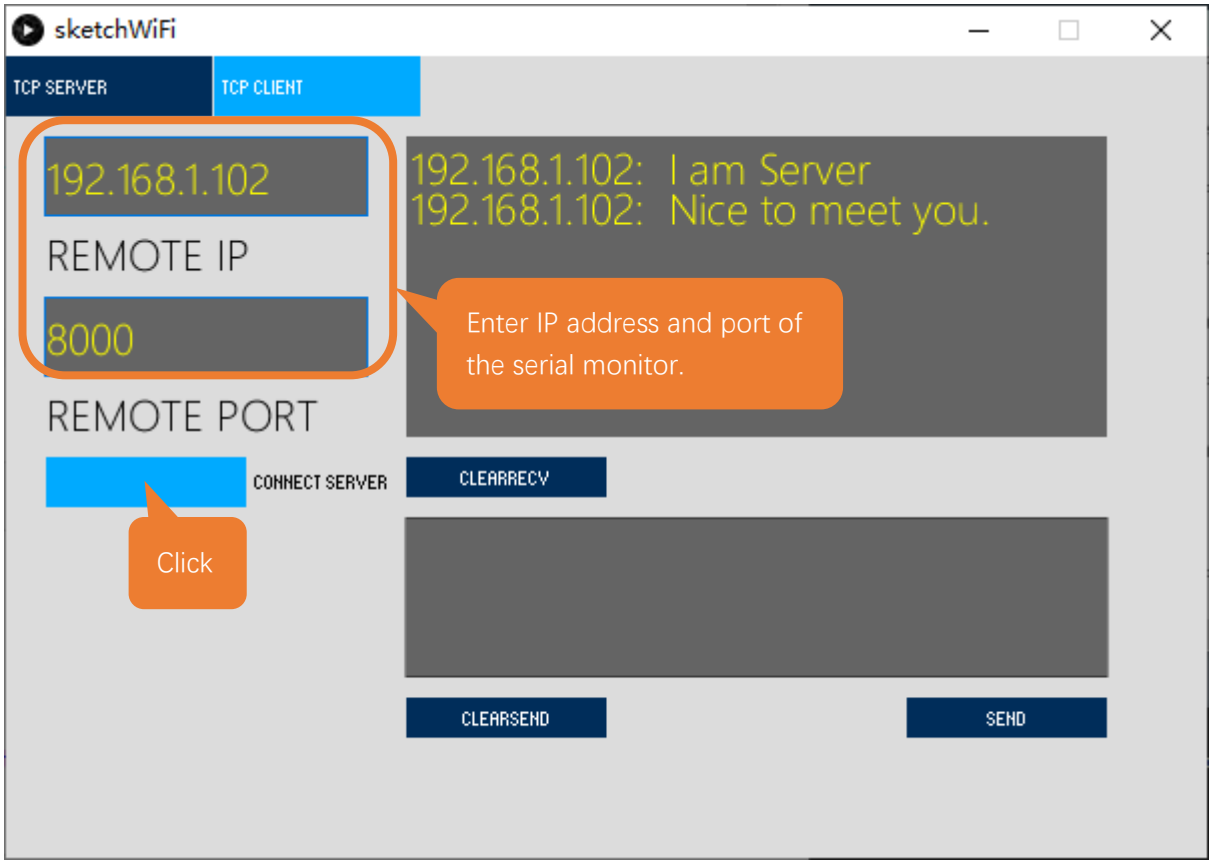
After making sure that the router's name and password are correct, click "Run current script" and in "Shell", you can see a server opened by the ESP32- WROVER waiting to connecting to other network devices.



Processing:

Open the "**Freenove_Basic_Starter_Kit_for_ESP32/Codes/MicroPython_Codes/15.2_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 to communicate.



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

```
Shell
MicroPython v1.12 on 2019-12-20; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

  tcp waiting...
  accepting.....
  Server IP: 192.168.1.102        Port: 8000
  ('192.168.1.142', 51326) connected
  b'Nice to meet you.'
```

The following is the program code:

```
import network
import socket
import time

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

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

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

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

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

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

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

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

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

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

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

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

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

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

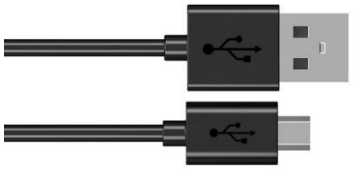
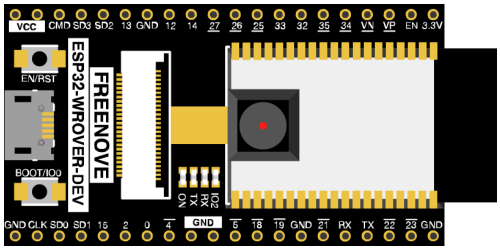
Chapter 16 Camera Web Server

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

Project 16.1 Camera Web Server

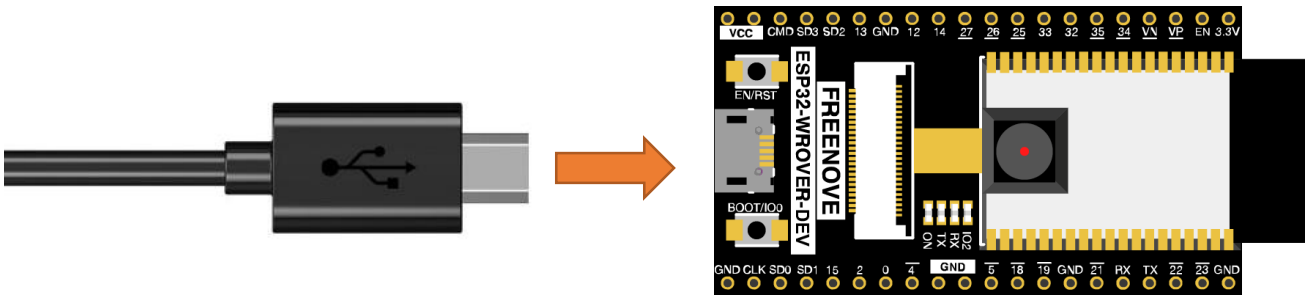
Connect ESP32 using USB and check its IP address through serial monitor. Use web page to access IP address to obtain video and image data.

Component List

Micro USB Wire x1 	ESP32-WROVER x1 

Circuit

Connect Freenove ESP32 to the computer using USB cable.

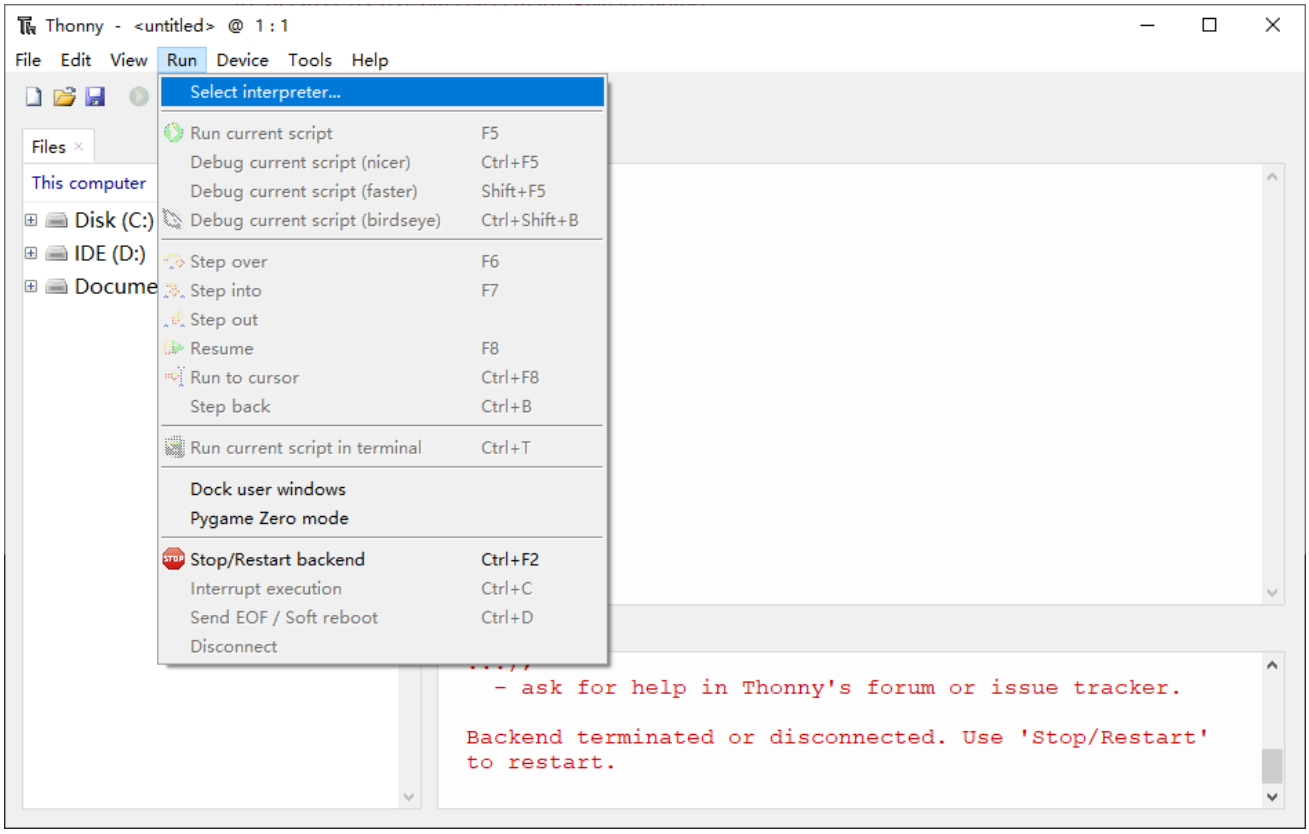


Code

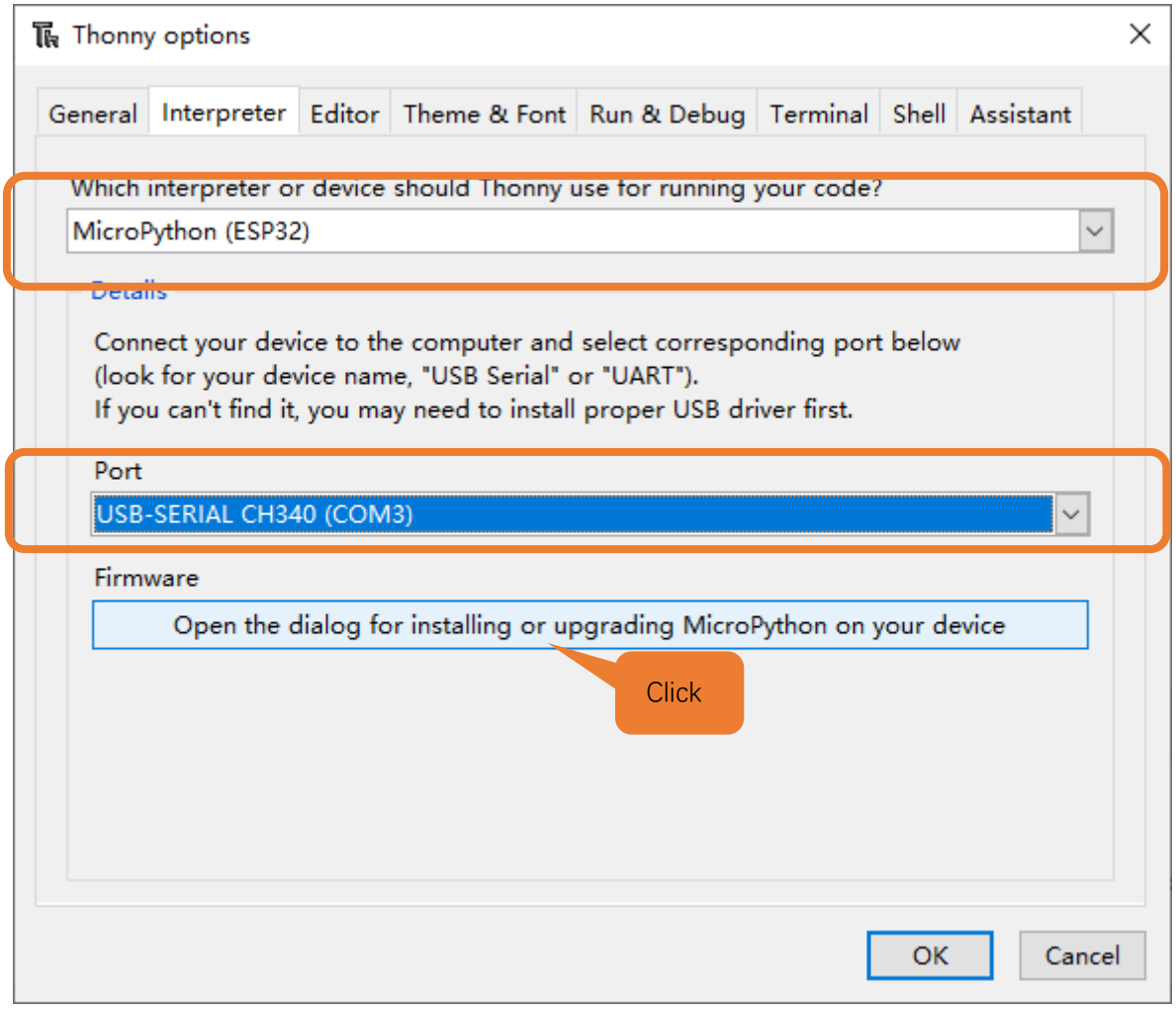
Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP32/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

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

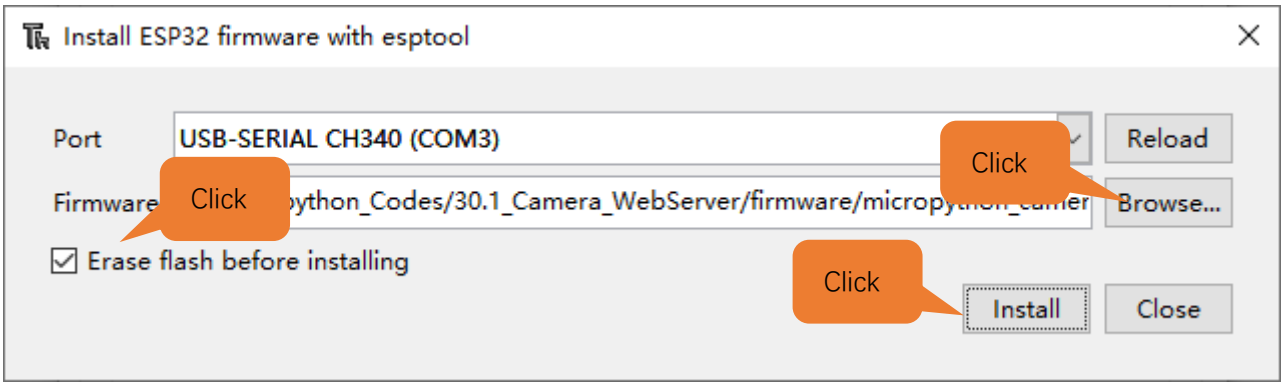
Before starting the project, we need to re-upload the firmware with the camera module via steps below.
Open Thonny, click "run" and select "Select interpreter..."".



Select "Micropython (ESP32)", select "USB-SERIAL CH340 (COM4)", and then click the long button under "Firmware".



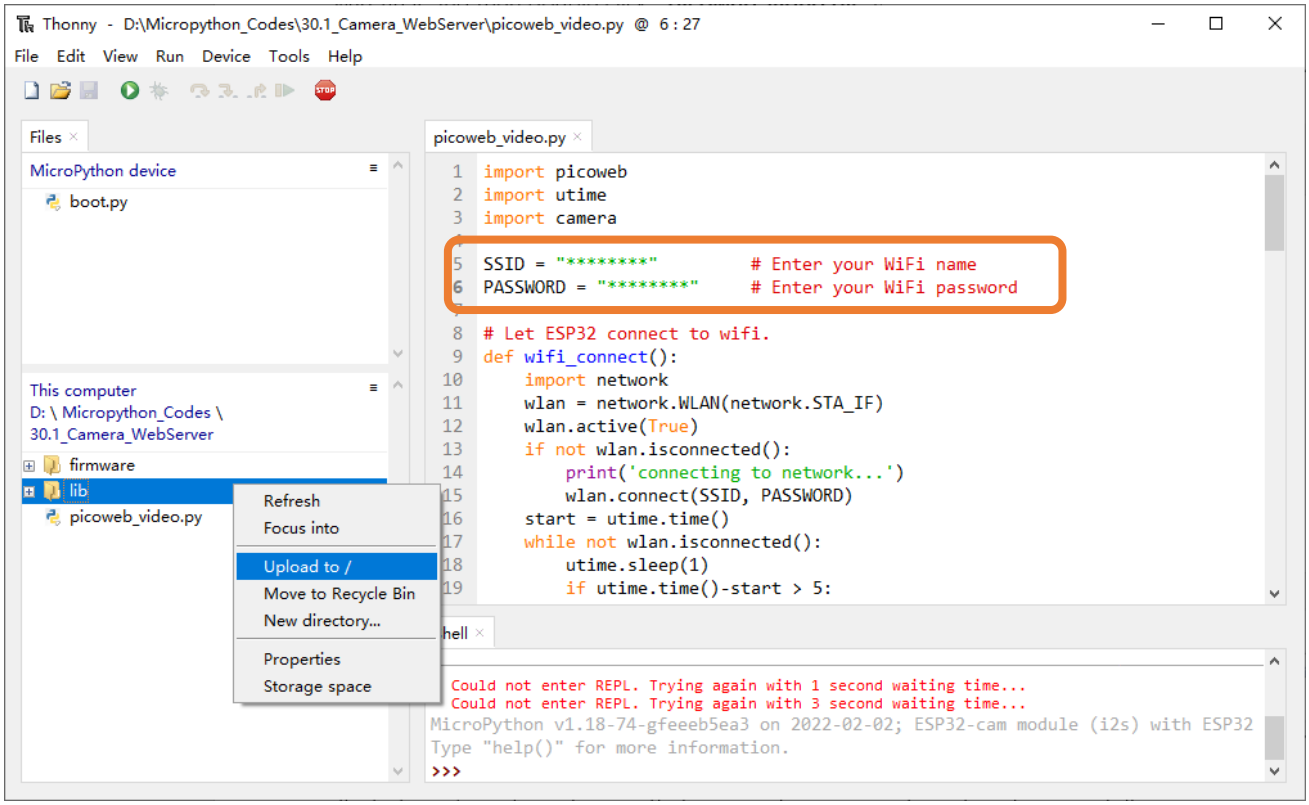
In the new popup window, select corresponding "USB-SERIAL CH340 (COM3)" for port. Click "Browse", select "**16.1_Camera_WebServer\firmware\micropython_camera_feeeb5ea3_esp32_idf4_4.bin**".
Select "Erase…" and click "Install"。



Wait for completion.

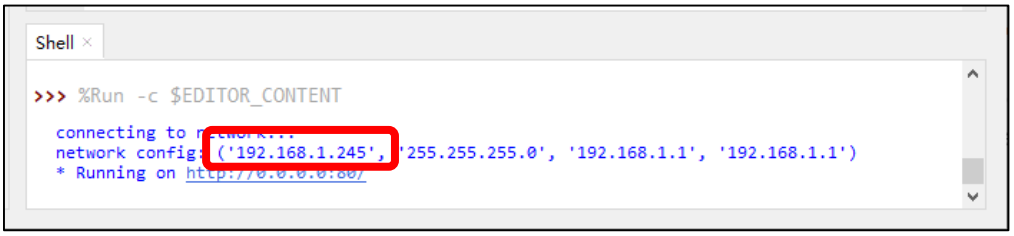
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "16.1_Camera_WebServer". Select folder "**lib**", right click your mouse to select "Upload to /", wait for "**lib**" to be uploaded to ESP32-WROVER and then double click "**picoweb_video.py**".

16.1_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.

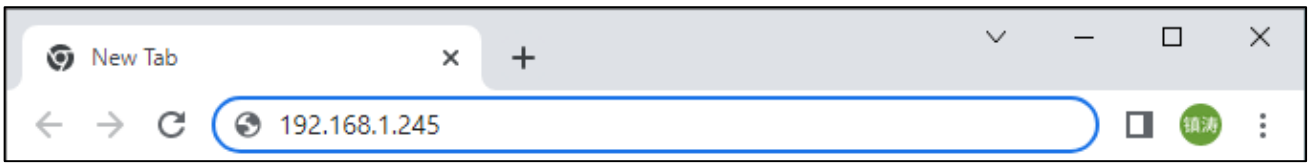
Click "run" to run the code "picoweb_video.py", then you can see the following content in the shell area.



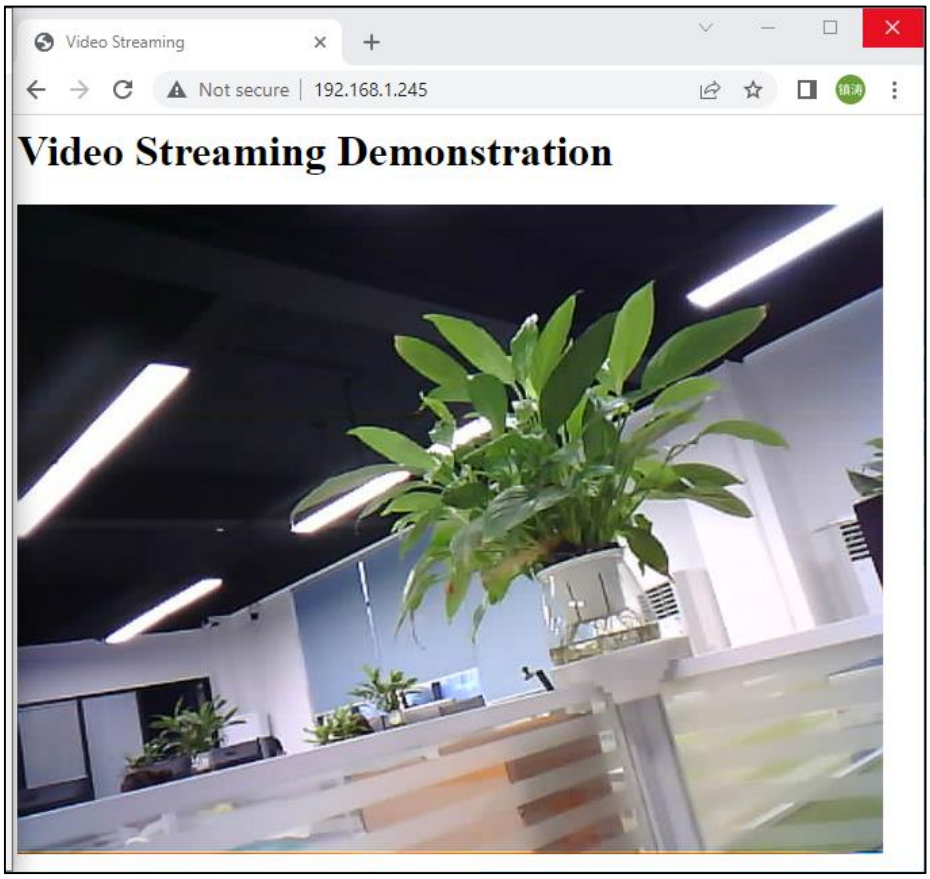
If your ESP32 has been in the process of connecting to router, but the information above has not been printed out, please re-check whether the router name and password have been entered correctly and press the reset key on ESP32-WROVER to wait for a successful connection prompt.

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 ESP32'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 esp32, wait for the system reset to complete, and then re-run the code.

The following is the program code.

```
1  import picoweb
2  import utime
3  import camera
4  import gc
5
6  SSID = "********"          # Enter your WiFi name
7  PASSWORD = "********"      # Enter your WiFi password
8
9  # Let ESP32 connect to wifi.
10 def wifi_connect():
11     import network
12     wlan = network.WLAN(network.STA_IF)
13     wlan.active(True)
14     if not wlan.isconnected():
15         print('connecting to network...')
16         wlan.connect(SSID, PASSWORD)
17     start = utime.time()
18     while not wlan.isconnected():
```

```
        utime.sleep(1)
        if utime.time()-start > 5:
            print("connect timeout!")
            break
    if wlan.isconnected():
        print('network config:', wlan.ifconfig())

# Initializing the Camera
def camera_init():
    # Disable camera initialization
    camera.deinit()
    # Enable camera initialization
    camera.init(0, d0=4, d1=5, d2=18, d3=19, d4=36, d5=39, d6=34, d7=35,
                format=camera.JPEG, framesize=camera.FRAME_VGA,
                xclk_freq=camera.XCLK_20MHz,
                href=23, vsync=25, reset=-1, pwdn=-1,
                sioc=27, siod=26, xclk=21, pclk=22, fb_location=camera.PSRAM)

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

    camera.flip(1)                      # Flip up and down window: 0-1
    camera.mirror(1)                    # Flip window left and right: 0-1
    camera.saturation(0)                # saturation: -2,2 (default 0). -2 grayscale
    camera.brightness(0)                # brightness: -2,2 (default 0). 2 brightness
    camera.contrast(0)                  # contrast: -2,2 (default 0). 2 highcontrast
    camera.quality(10)                  # quality: # 10-63 lower number means higher quality
    # Note: The smaller the number, the sharper the image. The larger the number, the more
blurry the image

    camera.speffect(camera.EFFECT_NONE)  # special effects:
    # EFFECT_NONE (default) EFFECT_NEG EFFECT_BW EFFECT_RED EFFECT_GREEN EFFECT_BLUE
EFFECT_RETRO
    camera.whitebalance(camera.WB_NONE)  # white balance
    # WB_NONE (default) WB_SUNNY WB_CLOUDY WB_OFFICE WB_HOME

# HTTP Response Content
index_web="""
HTTP/1.0 200 OK\r\n
```

```
<html>
  <head>
    <title>Video Streaming</title>
  </head>
  <body>
    <h1>Video Streaming Demonstration</h1>
    <img src="/video" margin-top:100px; style="transform:rotate(180deg); "/>
  </body>
</html>
"""

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

# Send camera pictures
def send_frame():
    buf = camera.capture()
    yield (b'--frame\r\n'
           b'Content-Type: image/jpeg\r\n\r\n'
           + buf + b'\r\n')
    del buf
    gc.collect()

# Video transmission
def video(req, resp):
    yield from picoweb.start_response(resp, content_type="multipart/x-mixed-replace;
boundary=frame")
    while True:
        yield from resp.awrite(next(send_frame()))
        gc.collect()

ROUTES = [
    # You can specify exact URI string matches...
    ("/", index),
    ("/video", video),
]

if __name__ == '__main__':

    import ulogging as logging
    logging.basicConfig(level=logging.INFO)
    camera_init()
```

```
    wifi_connect()

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

    app.run(debug=1, port=80, host="0.0.0.0")
    # debug values:
    # -1 disable all logging
    # 0 (False) normal logging: requests and errors
    # 1 (True) debug logging
    # 2 extra debug logging
```

Import picoweb、utime、camera、gc modules.

```
import picoweb
import utime
import camera
import gc
```

Before running the code, please modify the WiFi name and password in the code to ensure that the ESP32 can connect to the network.

```
SSID = "********"          # Enter your WiFi name
PASSWORD = "********"      # Enter your WiFi password
```

Define the WiFi connection function, set the ESP32 to STA mode, and let the ESP32 connect to the nearby WiFi. If the connection is successful, the WiFi configuration information of the ESP32 will be printed; if the connection fails, the connection timeout will be printed.

```
def wifi_connect():
    import network
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        print('connecting to network...')
        wlan.connect(SSID, PASSWORD)
    start = utime.time()
    while not wlan.isconnected():
        utime.sleep(1)
        if utime.time()-start > 5:
            print("connect timeout!")
            break
    if wlan.isconnected():
        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.

```
    camera.deinit()
    # Enable camera initialization
    camera.init(0, d0=4, d1=5, d2=18, d3=19, d4=36, d5=39, d6=34, d7=35,
                format=camera.JPEG, framesize=camera.FRAME_VGA,
                xclk_freq=camera.XCLK_20MHz,
                href=23, vsync=25, reset=-1, pwdn=-1,
                sioc=27, siod=26, xclk=21, pclk=22, 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.

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

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

```
    camera.flip(1)                      # Flip up and down window: 0-1
    camera.mirror(1)                    # Flip window left and right: 0-1
    camera.saturation(0)                # saturation: -2,2 (default 0). -2 grayscale
    camera.brightness(0)                # brightness: -2,2 (default 0). 2 brightness
    camera.contrast(0)                  # contrast: -2,2 (default 0). 2 highcontrast
    camera.quality(10)                  # quality: # 10-63 lower number means higher quality
    # Note: The smaller the number, the sharper the image. The larger the number, the more blurry the image

    camera.speffect(camera.EFFECT_NONE)  # special effects:
    # EFFECT_NONE (default) EFFECT_NEG EFFECT_BW EFFECT_RED EFFECT_GREEN EFFECT_BLUE EFFECT_RETRO
    camera.whitebalance(camera.WB_NONE)  # white balance
    # 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.

```
index_web="""
HTTP/1.0 200 OK\r\n
<html>
  <head>
    <title>Video Streaming</title>
  </head>
  <body>
    <h1>Video Streaming Demonstration</h1>
    <img src="/video" margin-top:100px; style="transform:rotate(180deg); "/>
  </body>
```

```
</html>
"""
```

Web page response function. When a user visits the webpage "/" built by ESP32, ESP32 calls this function, allowing the user to observe a display interface in the browser.

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

send_frame() can send the image obtained by ESP32 in web page format. When someone visits the webpage "/video" built by the ESP32, the video(req, resp) function is used to continuously fetch images and send them to the browser.

```
# Send camera pictures
def send_frame():
    buf = camera.capture()
    yield (b'--frame\r\n'
           b'Content-Type: image/jpeg\r\n\r\n'
           + buf + b'\r\n')
    del buf
    gc.collect()

# Video transmission
def video(req, resp):
    yield from picoweb.start_response(resp, content_type="multipart/x-mixed-replace;
boundary=frame")
    while True:
        yield from resp.awrite(next(send_frame()))
        gc.collect()
```

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

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

This is the main part of the program. First initialize the ESP32 camera, and then configure WiFi to connect the ESP32 to the network. Call the picoweb library, build a webserver, and run it.

```
    import ulogging as logging
    logging.basicConfig(level=logging.INFO)
    camera_init()
    wifi_connect()

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

    app.run(debug=1, port=80, host="0.0.0.0")
```

Reference

Image resolution	Sharpness	Image resolution	Sharpness
FRAME_96X96	96x96	FRAME_HVGA	480x320
FRAME_QQVGA	160x120	FRAME_VGA	640x480
FRAME_QCIF	176x144	FRAME_SVGA	800x600
FRAME_HQVGA	240x176	FRAME_XGA	1024x768
FRAME_240X240	240x240	FRAME_HD	1280x720
FRAME_QVGA	320x240	FRAME_SXGA	1280x1024
FRAME_CIF	400x296	FRAME_UXGA	1600x1200

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 ESP32, you view our ultimate tutorial:
https://github.com/Freenove/Freenove_Ultimate_Starter_Kit_for_ESP32/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.