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®, ESP8266®, 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 ESP8266® are trademarks of ESPRESSIF Systems (Shanghai) Co., Ltd (https://www.espressif.com/).

Any concerns? ✉ support@freenove.com

Contents

Prepare

ESP8266 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.

ESP8266 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 ESP8266 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.

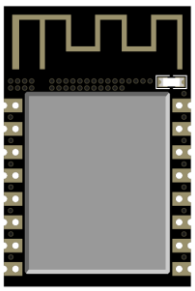
We divide each project into four parts, namely Component List, Component Knowledge, Circuit and Code. Component List helps you to prepare material for the experiment more quickly. Component Knowledge allows you to quickly understand new electronic modules or components, while Circuit helps you understand the operating principle of the circuit. And Code allows you to easily master the use of ESP8266 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

ESP8266

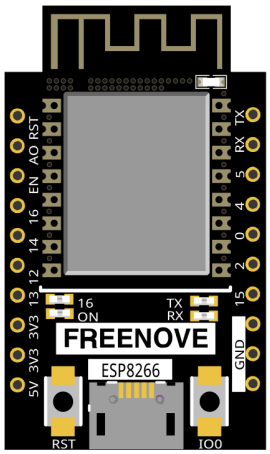
ESP8266 has PCB on-board antenna. The PCB on-board antenna is an integrated antenna in the chip module itself, so it is convenient to carry and design.

PCB on-board antenna

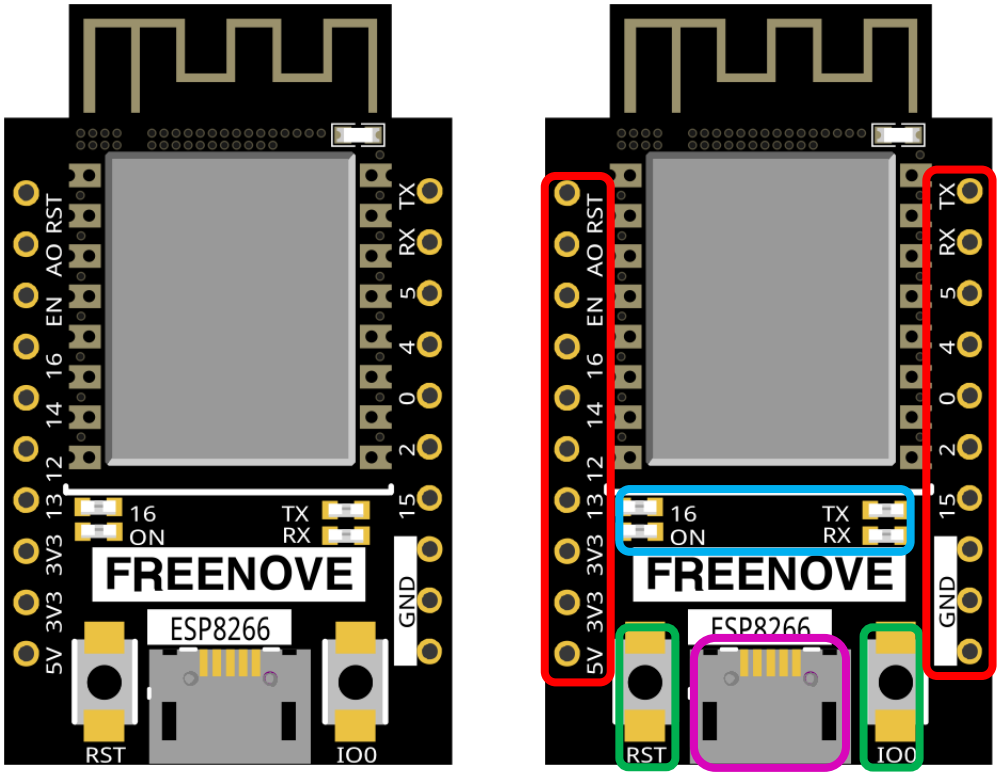


In this tutorial, the ESP8266 development board is designed based on the PCB on-board antenna-packaged ESP8266 module. The following tutorials will be based on the ESP8266 development board.

ESP8266 development board



The hardware interfaces of ESP8266 are distributed as follows:



Compare the left and right images. We've boxed off the resources on the ESP8266 in different colors to facilitate your understanding of the ESP8266 development board.

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

NO.	Pin Name	Functional Description
1	RST	Reset Pin, Active Low
2	ADC	AD conversion, Input voltage range 0~1V, the value range is 0~1024.
3	EN	Chip Enabled Pin, Active High
4	IO16	Connect with RST pin to wake up Deep Slee
5	IO14	GPIO14; HSPI_CLK
6	IO12	GPIO12; HSPI_MISO
7	IO13	GPIO13; HSPI_MOSI; UART0_CTS
8	VCC	Module power supply pin, Voltage 3.0V ~ 3.6V
9	GND	GND
10	IO15	GPIO15; MTDO; HSPICS; UART0
11	IO2	GPIO2; UART1_TXD
12	IO0	GPIO2; UART1_TXD
13	IO4	GPIO4
14	IO5	GPIO5;IR_R
15	RXD	UART0_RXD; GPIO3
16	TXD	UART0_TXD; GPIO1

Description of the ESP8266 series module boot mode:

Mode	CH_PD(EN)	RST	GPIO15	GPIO0	GPIO2	TXD0
Download mode	high	high	low	low	high	high
Running mode	high	high	low	high	high	high

Notes: Some of the pins inside the module have been pulled or pulled down.

If you want to learn more about this, you can read the following files:

“Freenove_Basic_Starter_Kit_for_ESP8266/Datasheet/esp-12s_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 ESP6266 during the whole process.
Thonny supports various operating system, including Windows、Mac OS、Linux.

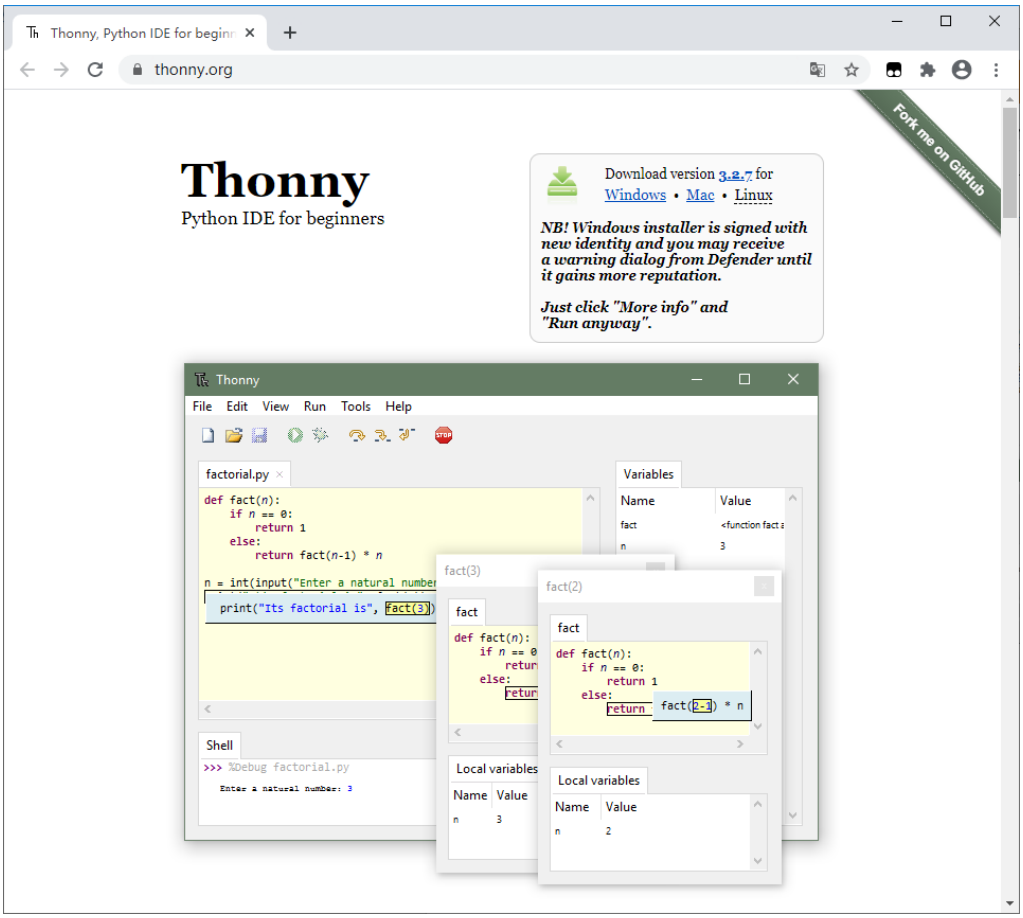
Downloading Thonny

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

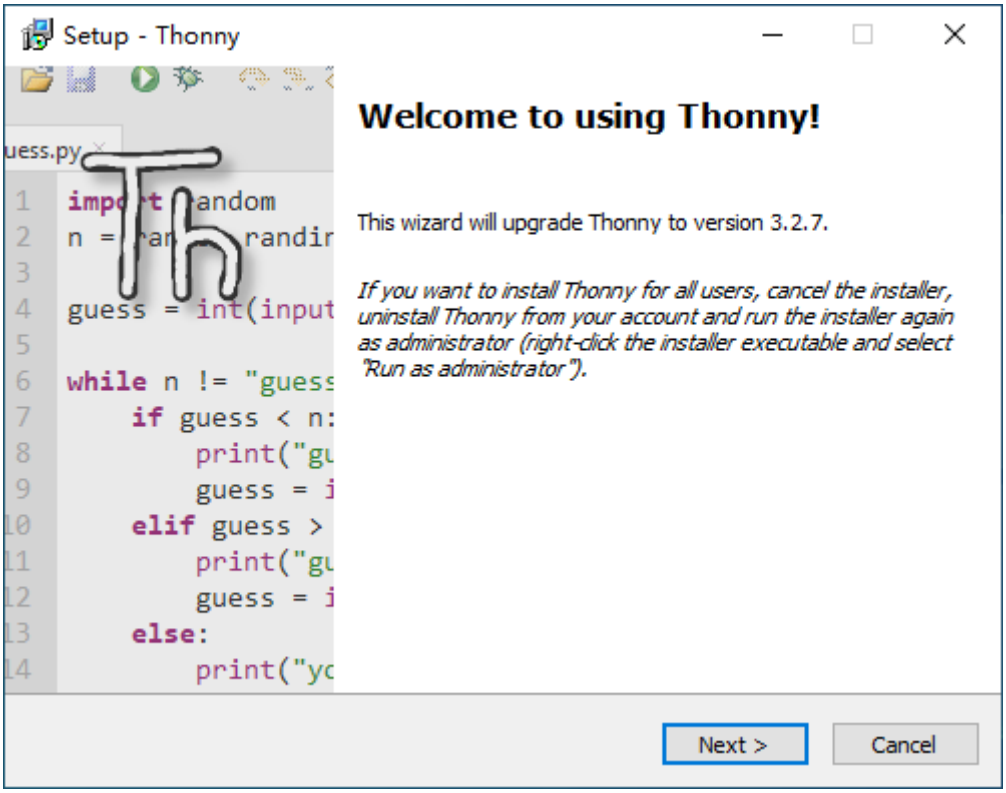
You can also open "**Freenove_Basic_Starter_Kit_for_ESP8266/Python/Python_Software**", we have prepared it in advance.



Installing on Windows

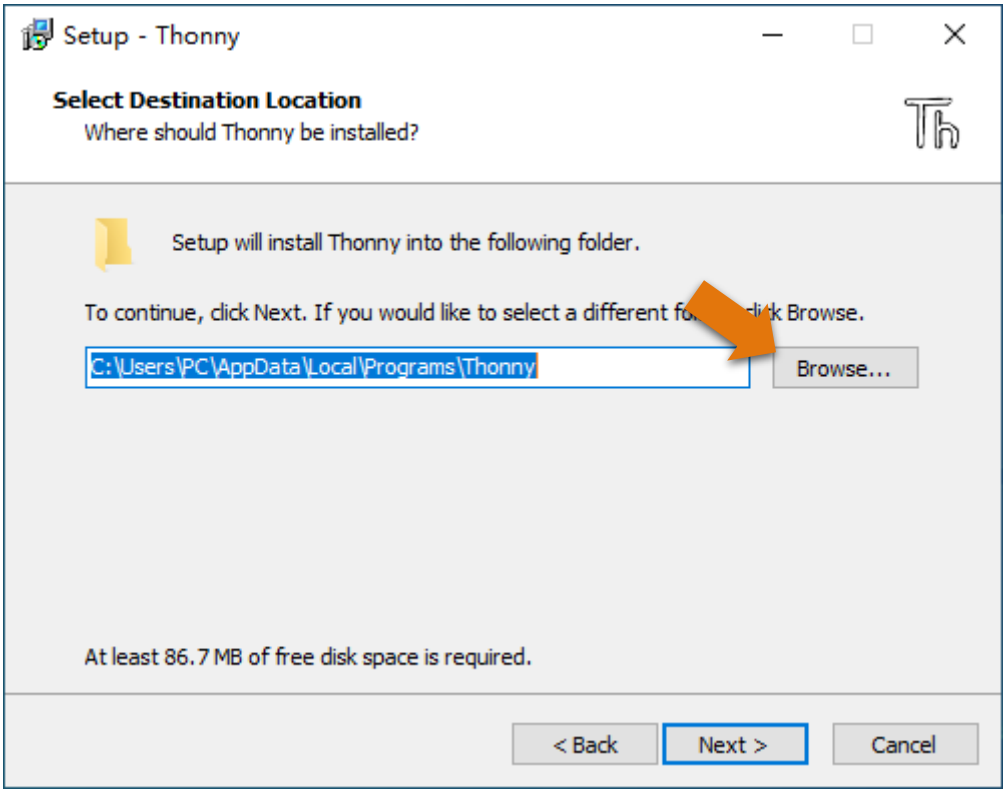
The icon of Thonny after downloading is as below. Double click “thonny-3.2.7.exe”.

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

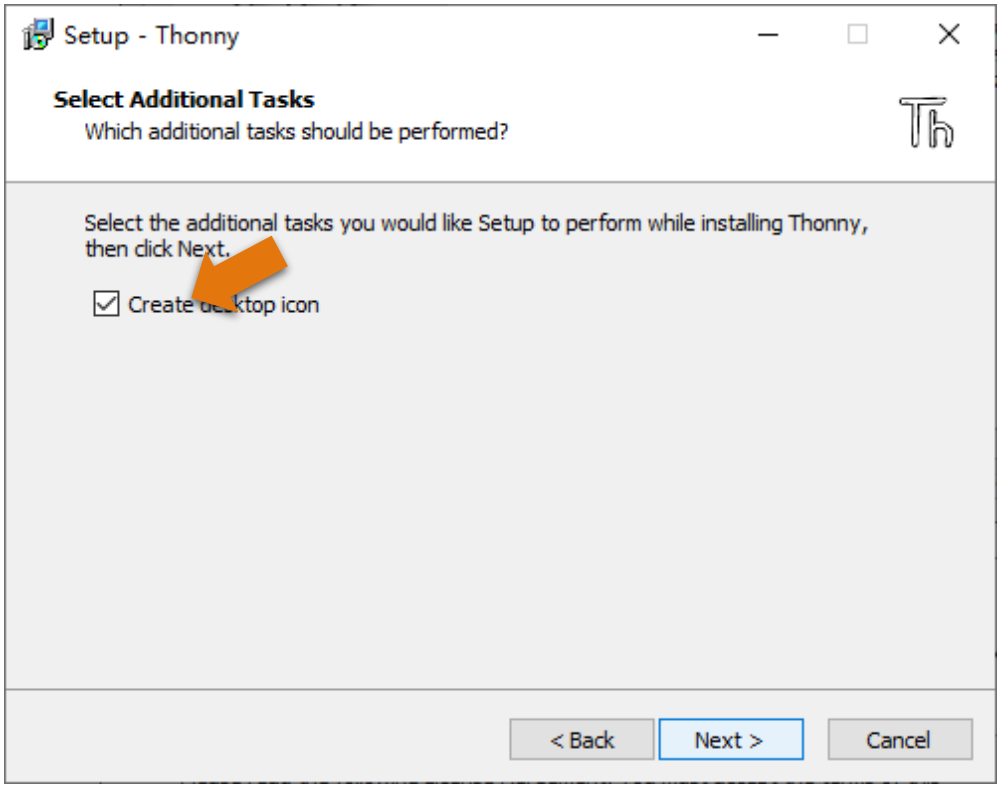


If you want to change Thonny's installation path, you can click "Browse" to modify it. After selecting installation path, click "OK".

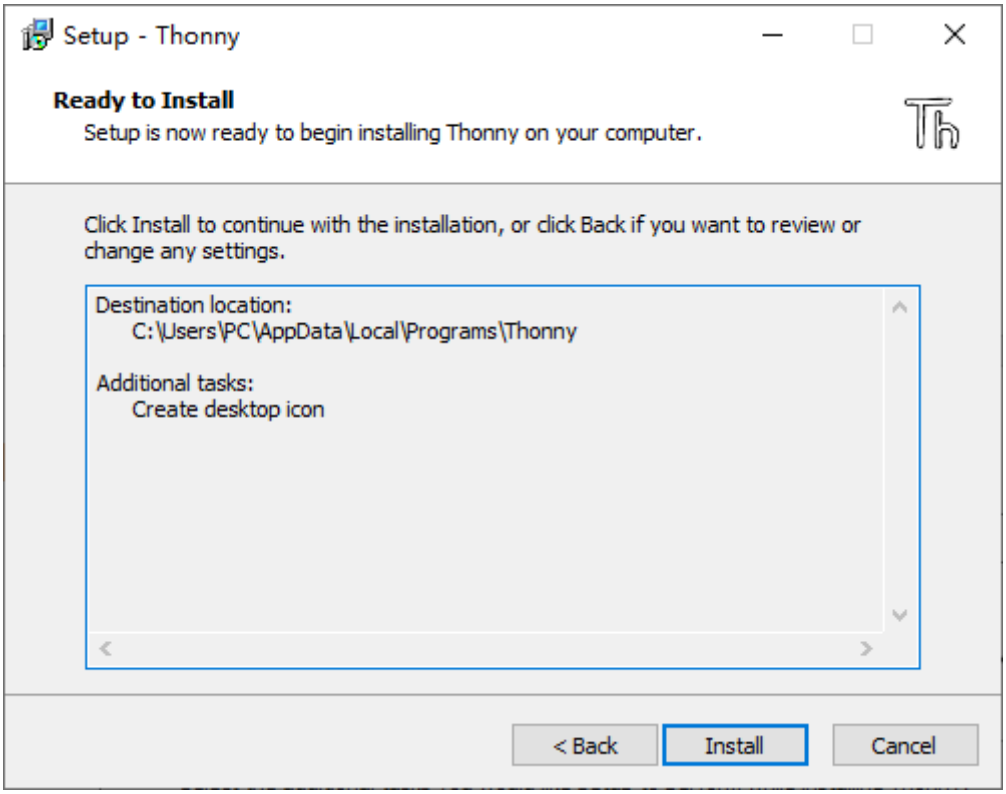
If you do not want to change it, just click "Next".



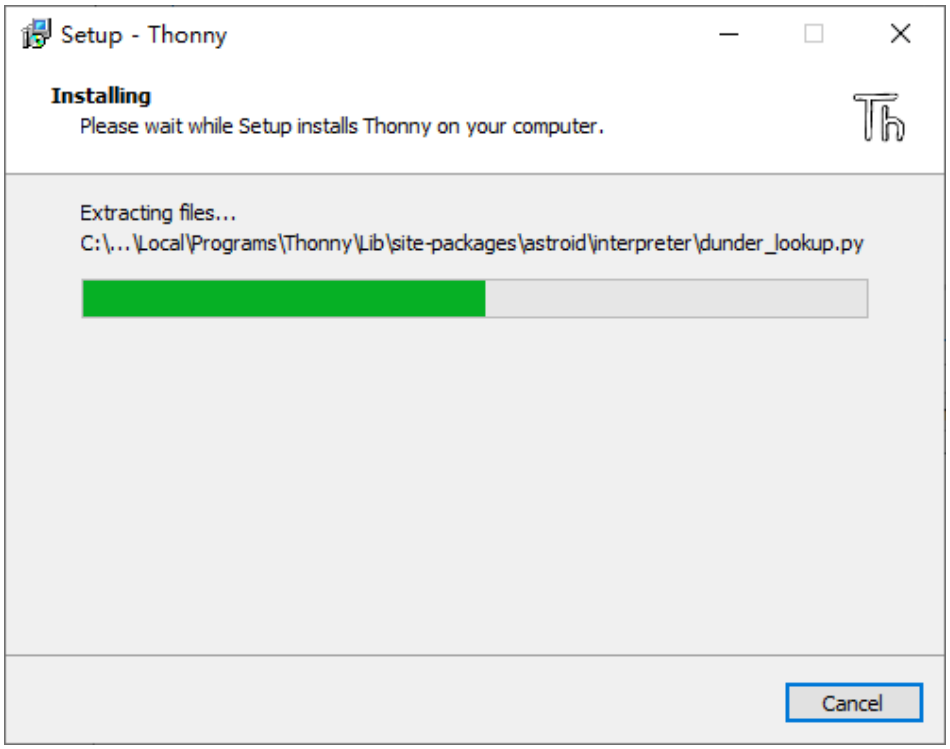
Check "Create desktop icon" and then it will generate a shortcut on your desktop to facilitate you to open Thonny later.



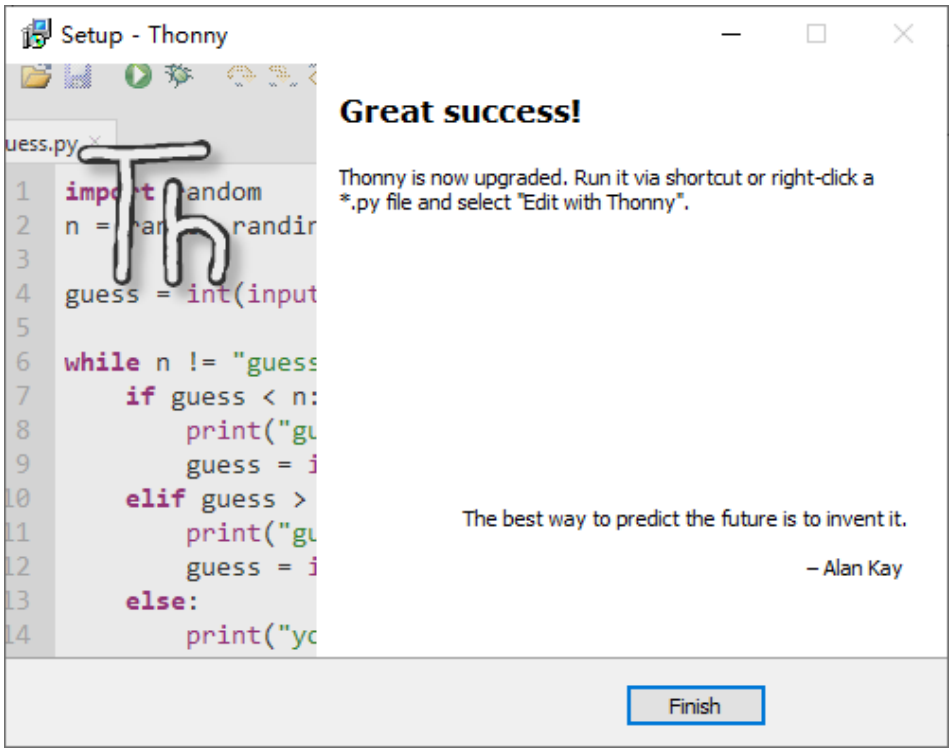
Click "install" to install the software.



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.



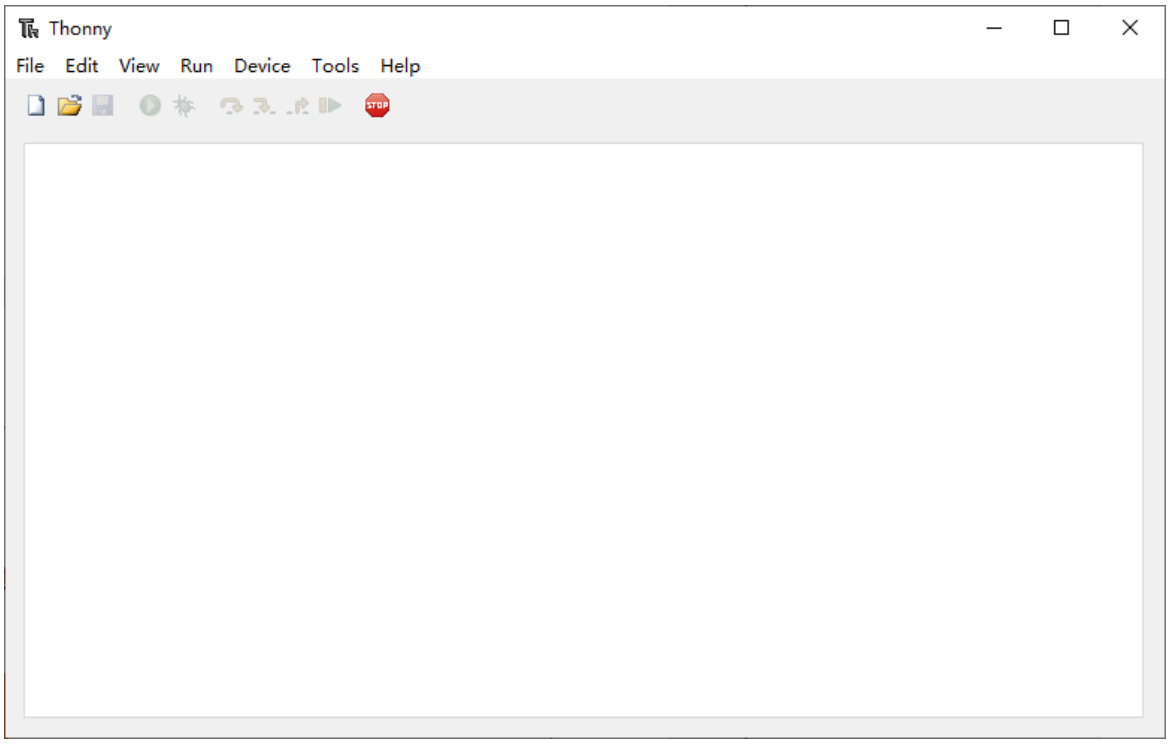
Once you see the interface as below, Thonny has been installed successfully.



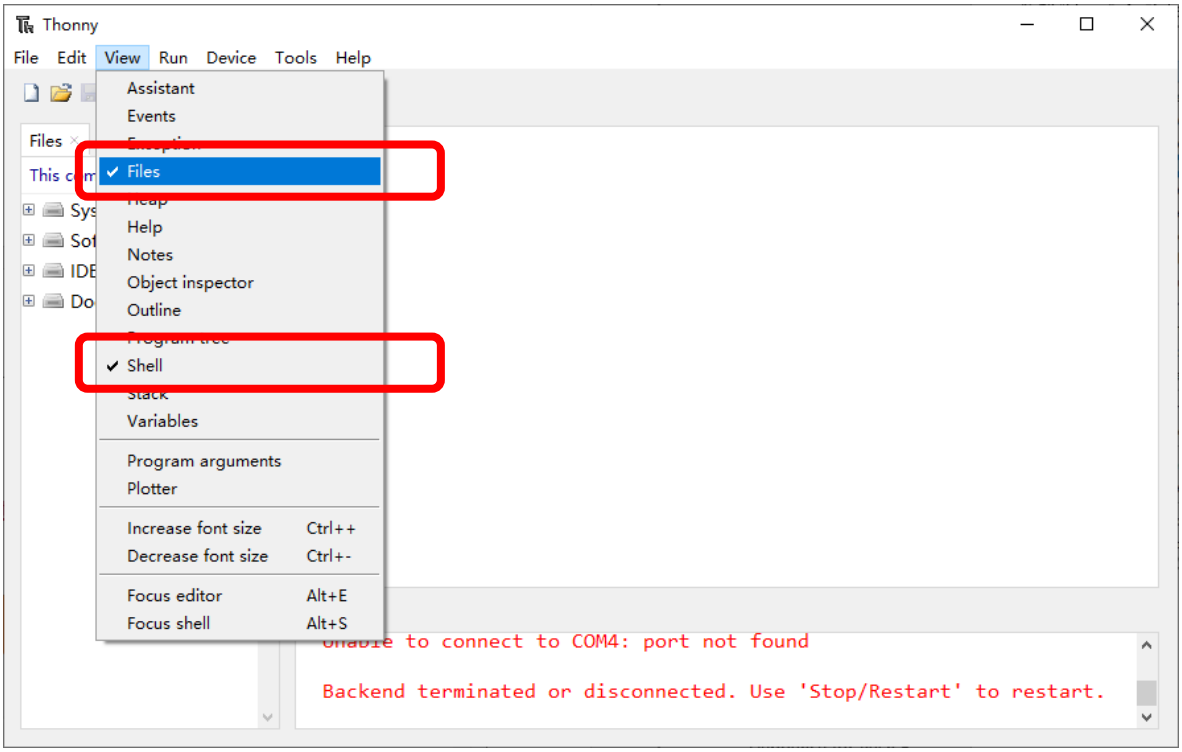
If you've check "Create desktop icon" during the installation process, you can see the below icon on your desktop.

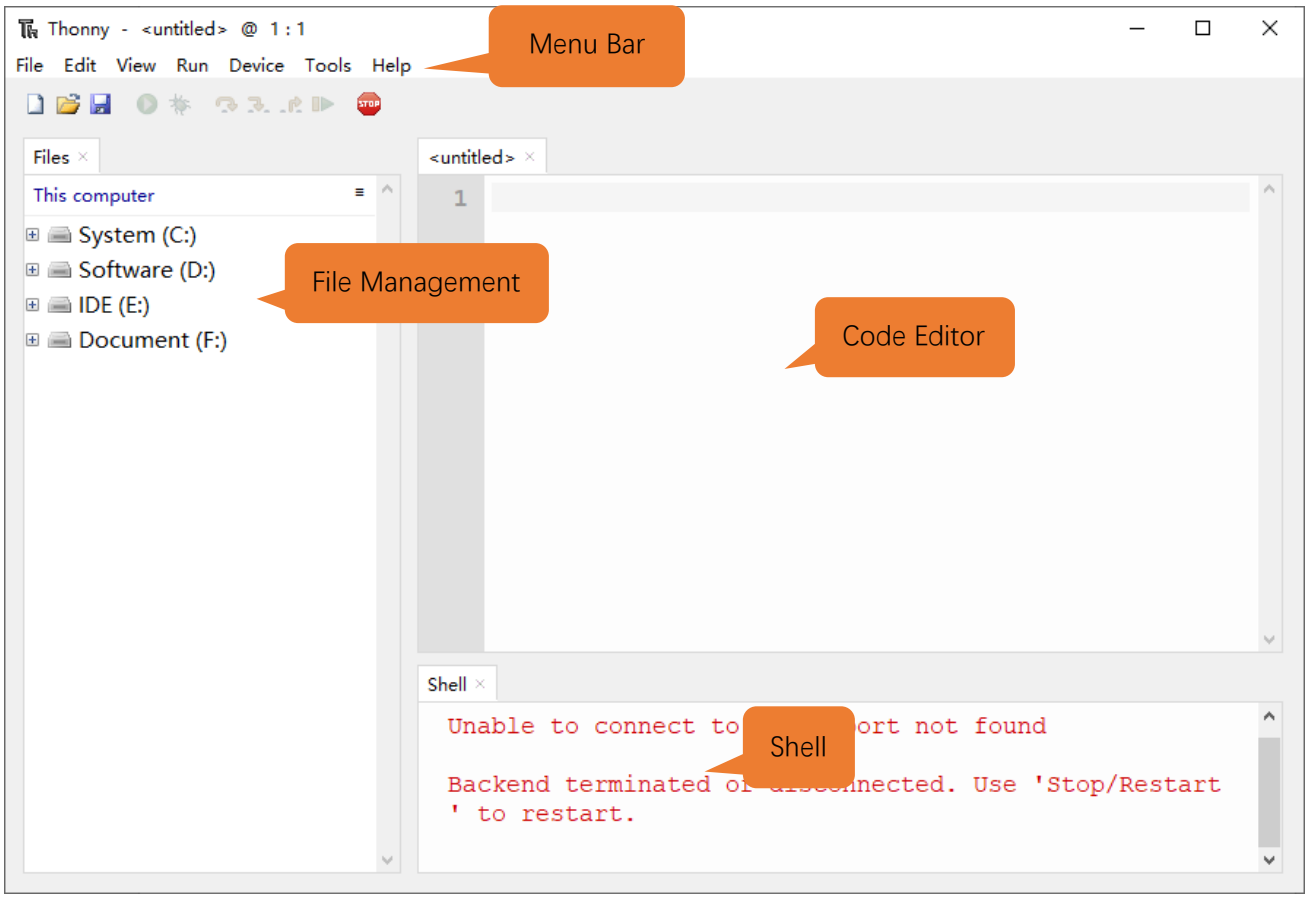
0.2 Basic Configuration of Thonny

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

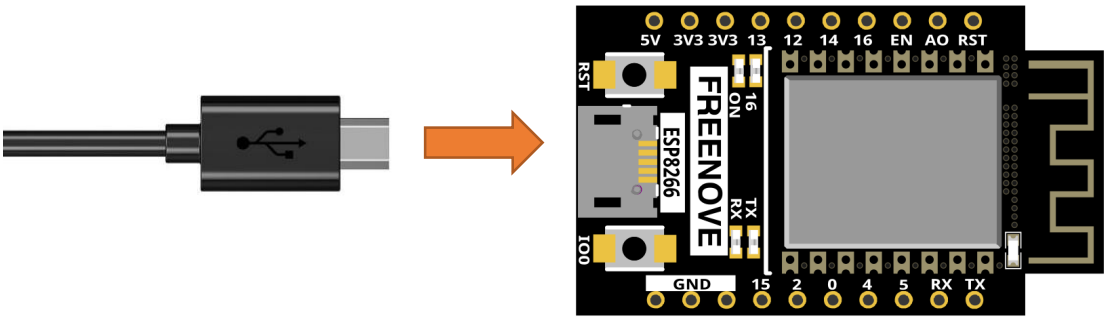
0.3 Installing CH340 (Important)

ESP8266 uses CH340 to download codes. So before using it, we need to install CH340 driver in our computers.

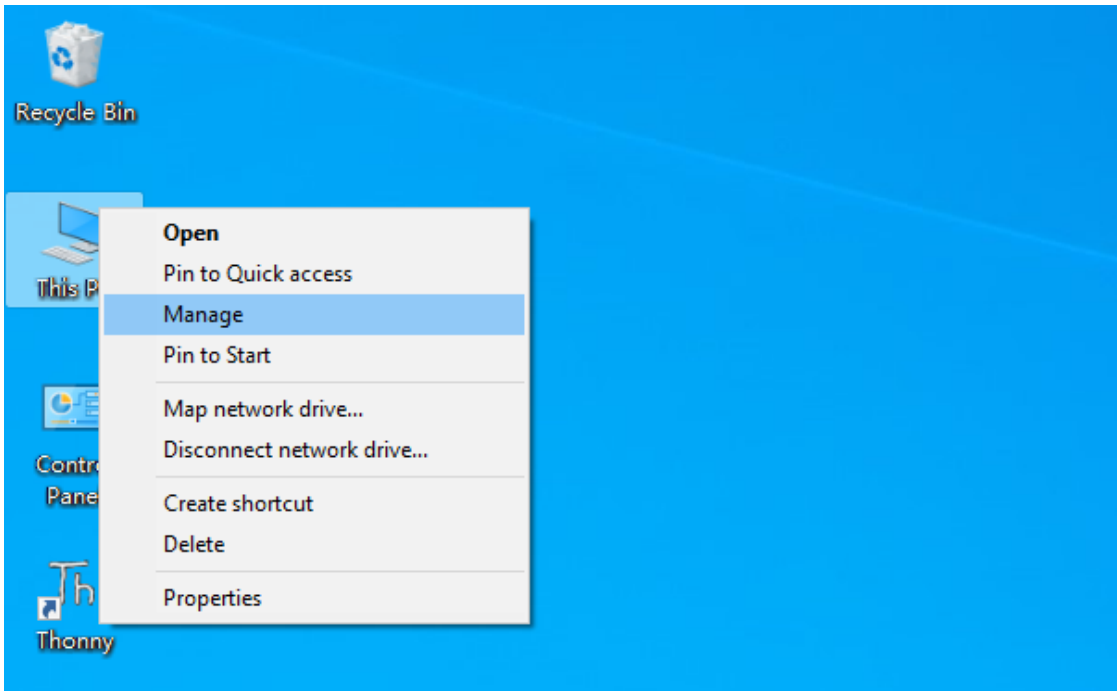
Windows

Check whether CH340 has been installed

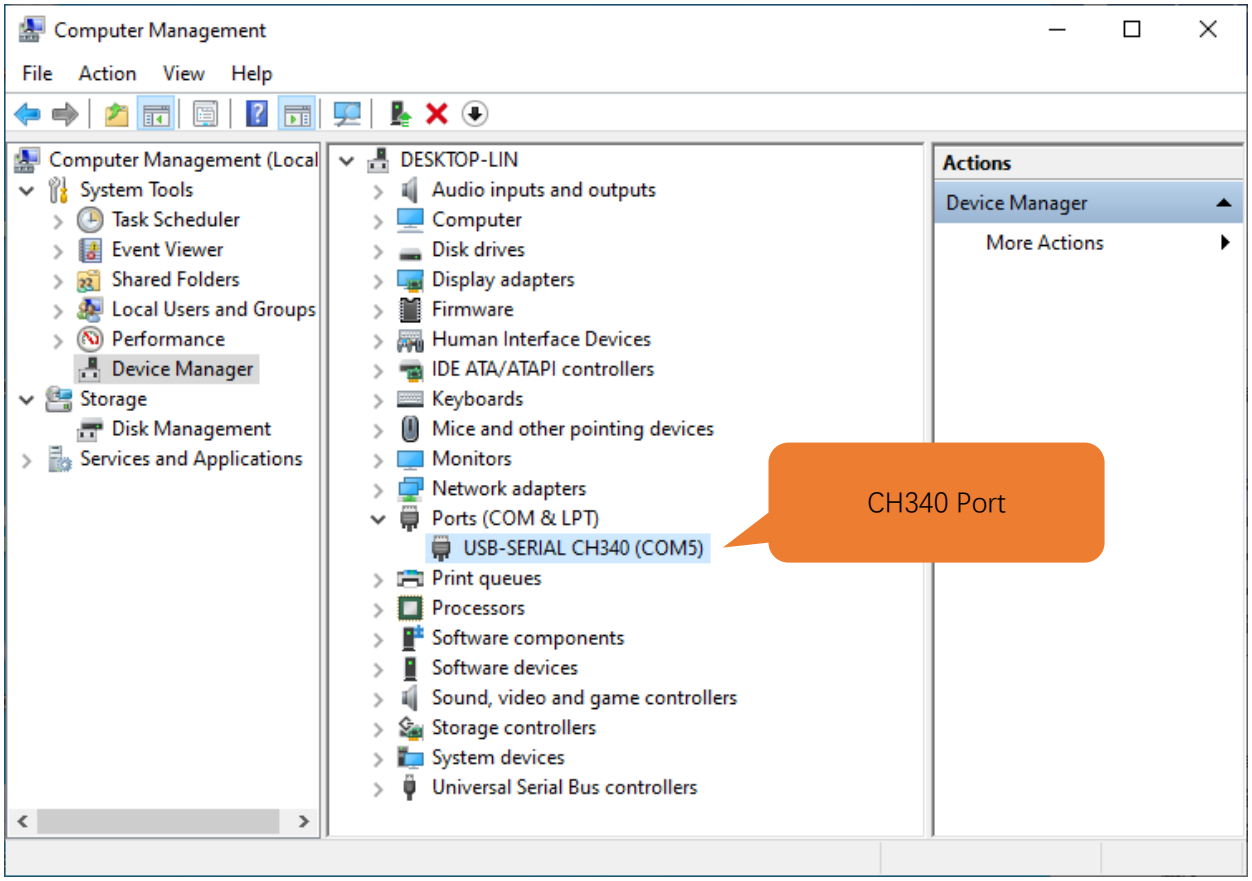
1. Connect your computer and ESP8266 with a USB cable.



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

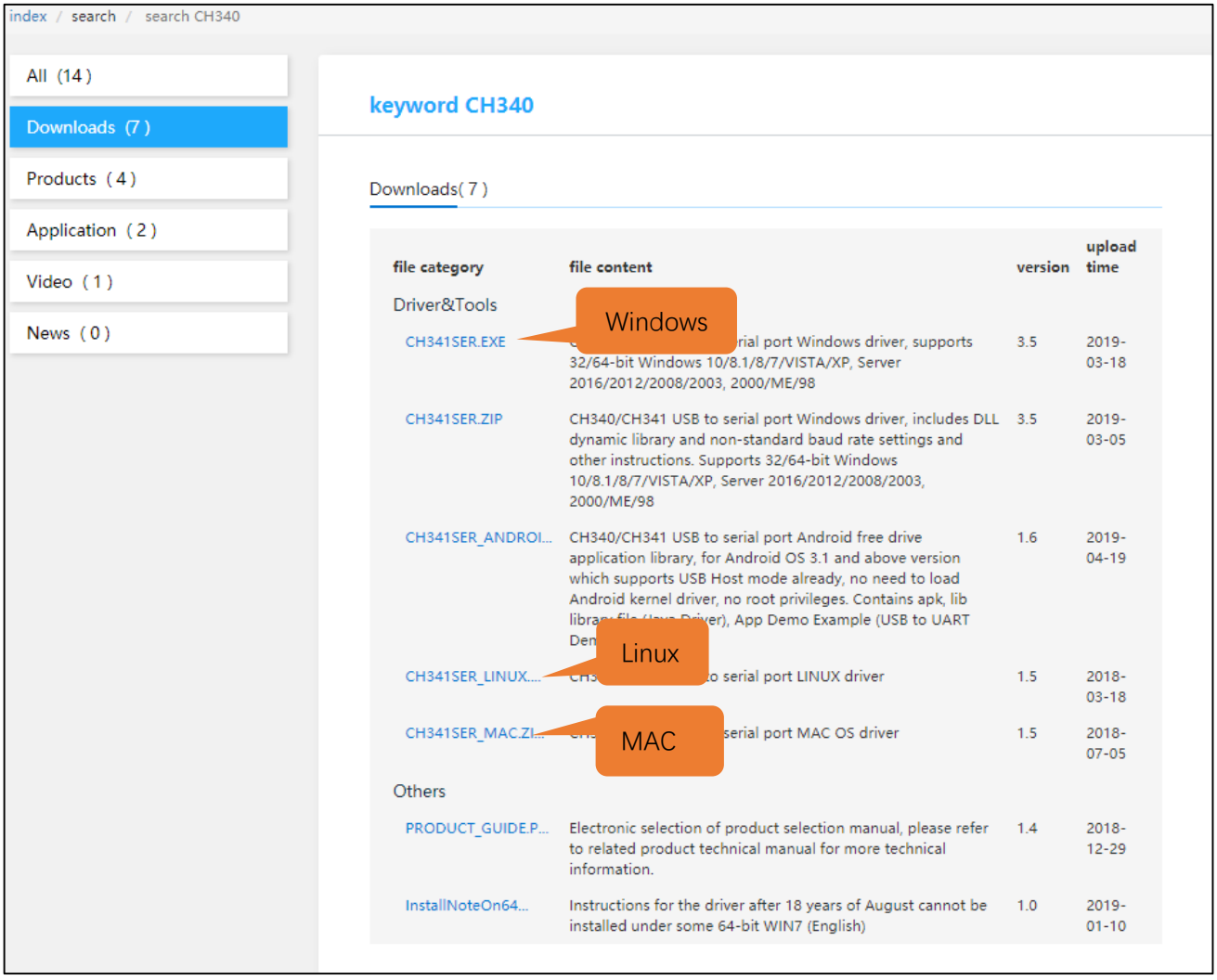


3. Click "Device Manager". If your computer has installed CH340, you can see"USB-SERIAL CH340 (COMx)". And you can click here to move to the next step.

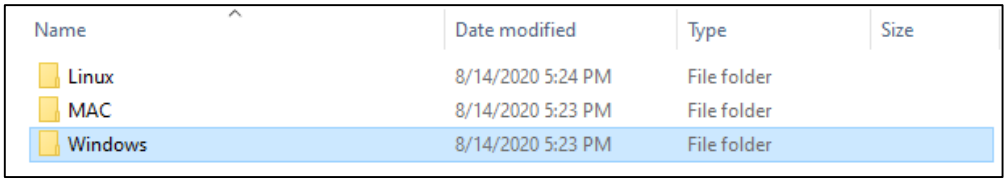


Installing CH340

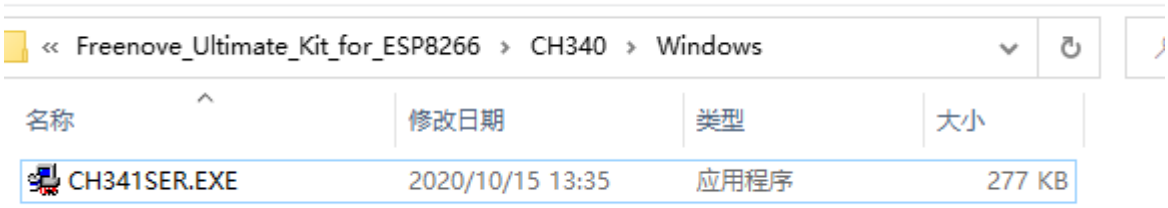
1. First, download CH340 driver, click http://www.wch-ic.com/search?q=CH340&t=downloads to download the appropriate one based on your operating system.



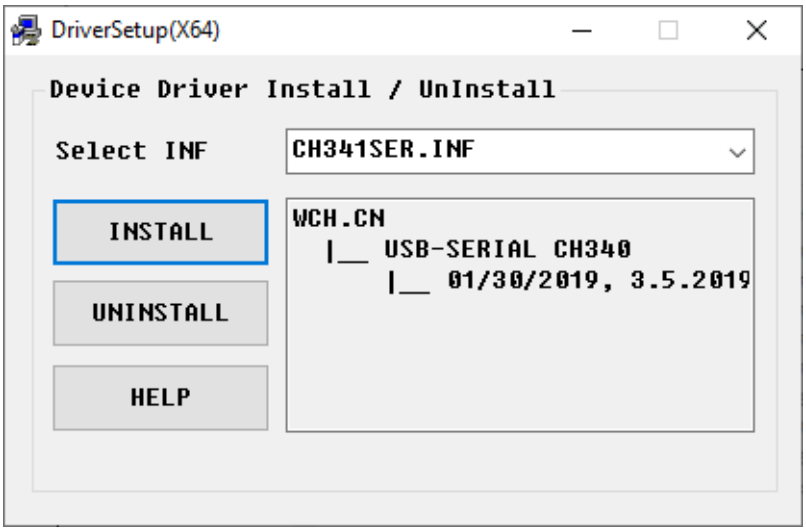
You can also open **"Freenove_Basic_Starter_Kit_for_ESP8266/CH340"**, we have prepared the installation package.



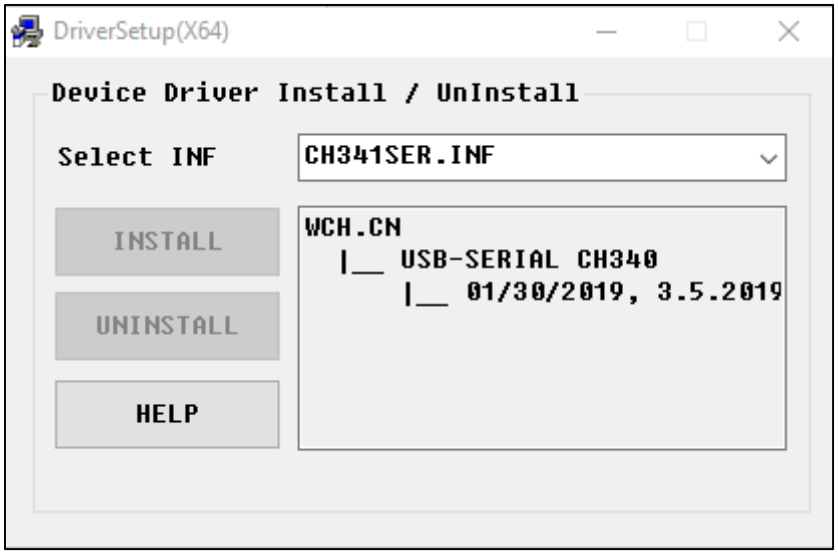
2. Open the folder "**Freenove_Basic_Starter_Kit_for_ESP8266/CH340/Windows/ch341ser**"



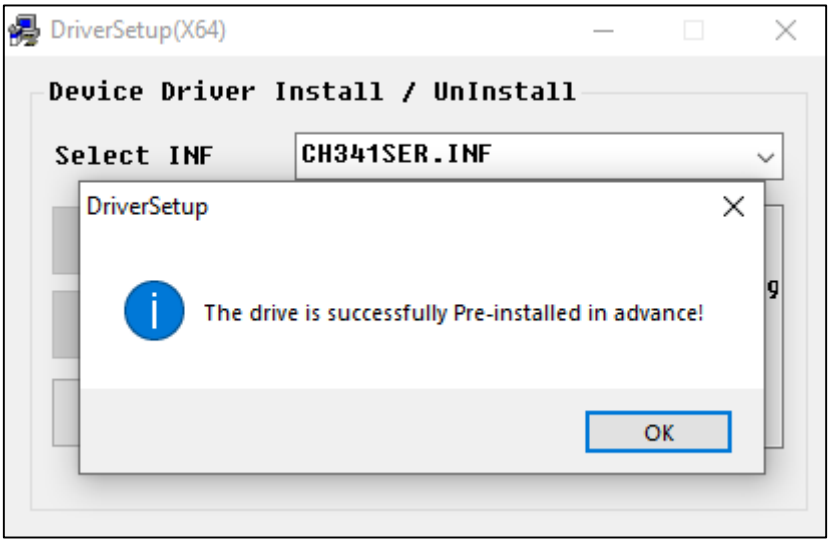
3. Double click "CH341SER.EXE".



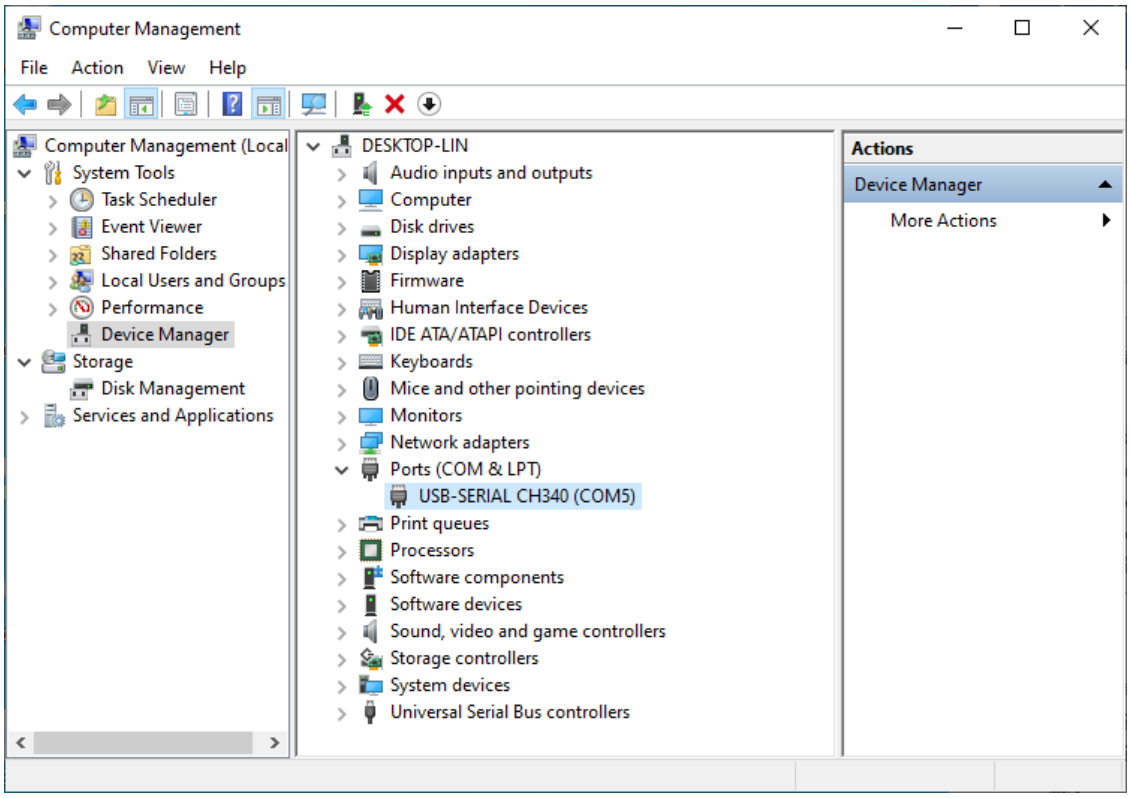
4. Click “INSTALL” and wait for the installation to complete.



5. Install successfully. Close all interfaces.



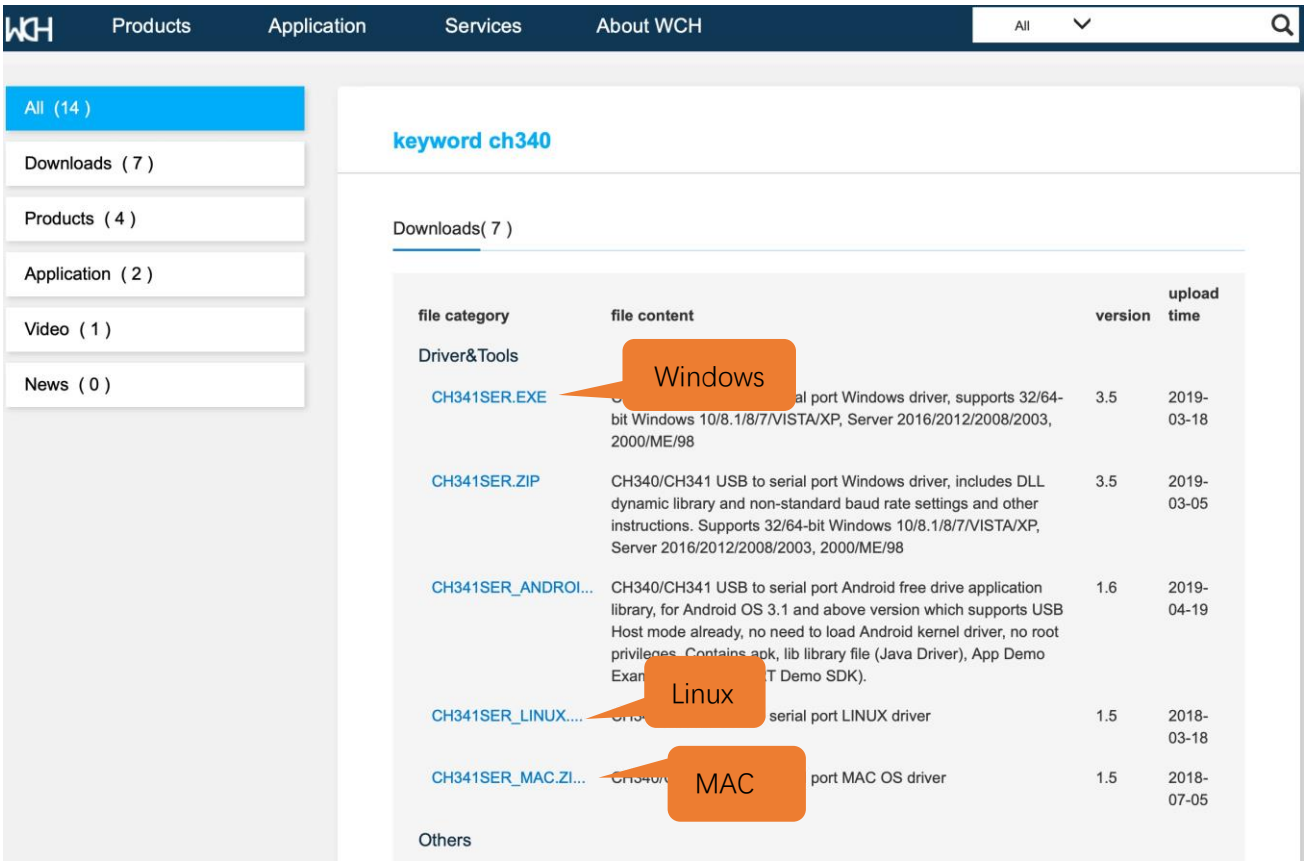
6. When ESP8266 is connected to computer, select "This PC", right-click to select "Manage" and click "Device Manager" in the newly pop-up dialog box, and you can see the following interface.



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

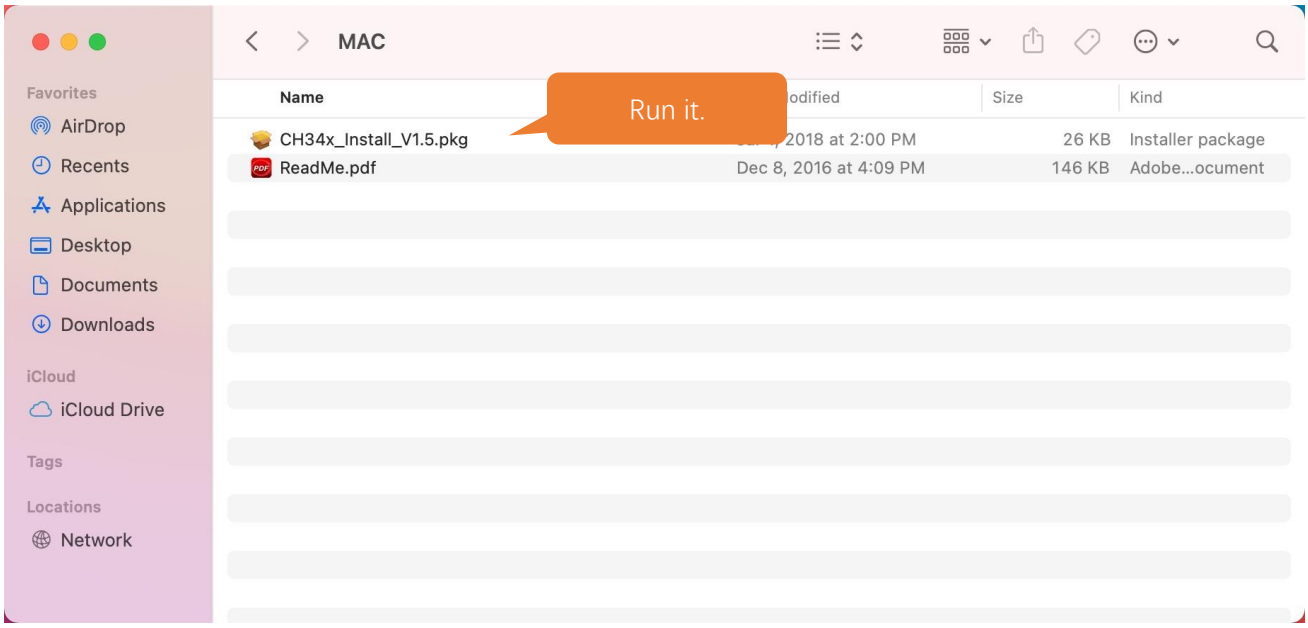
MAC

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.

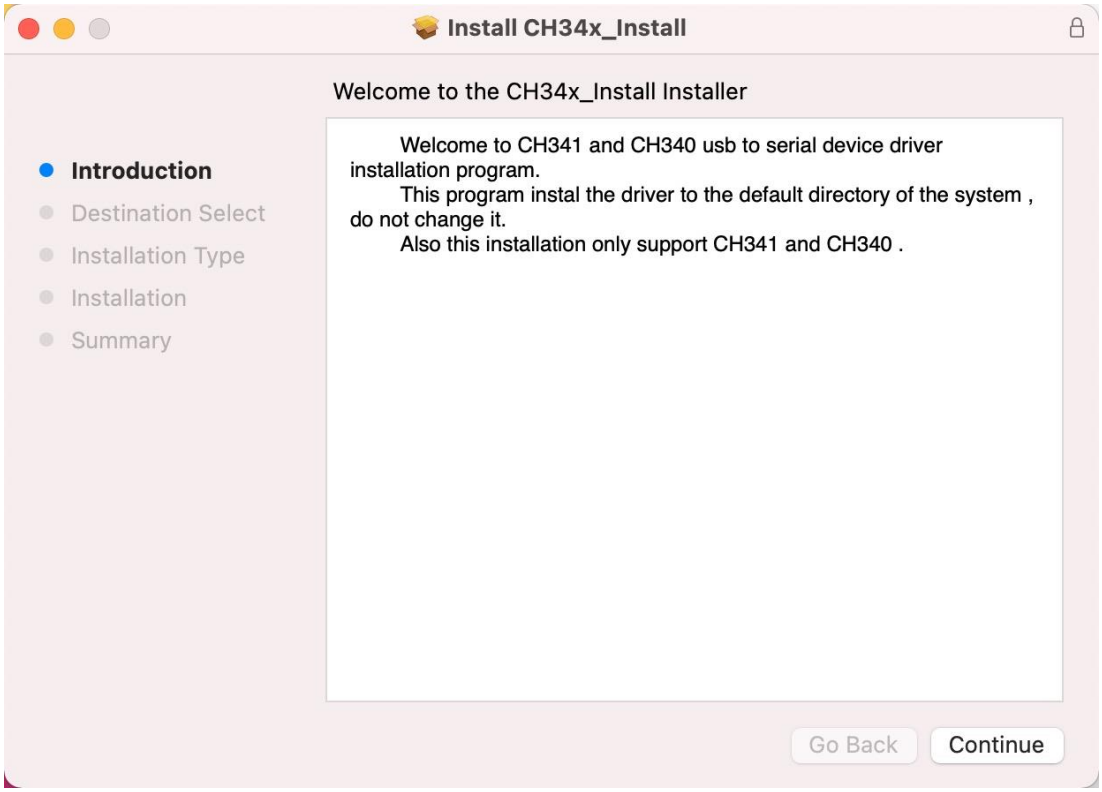


If you would not like to download the installation package, you can open **"Freenove_Basic_Starter_Kit_for_ESP8266/CH340"**, we have prepared the installation package.

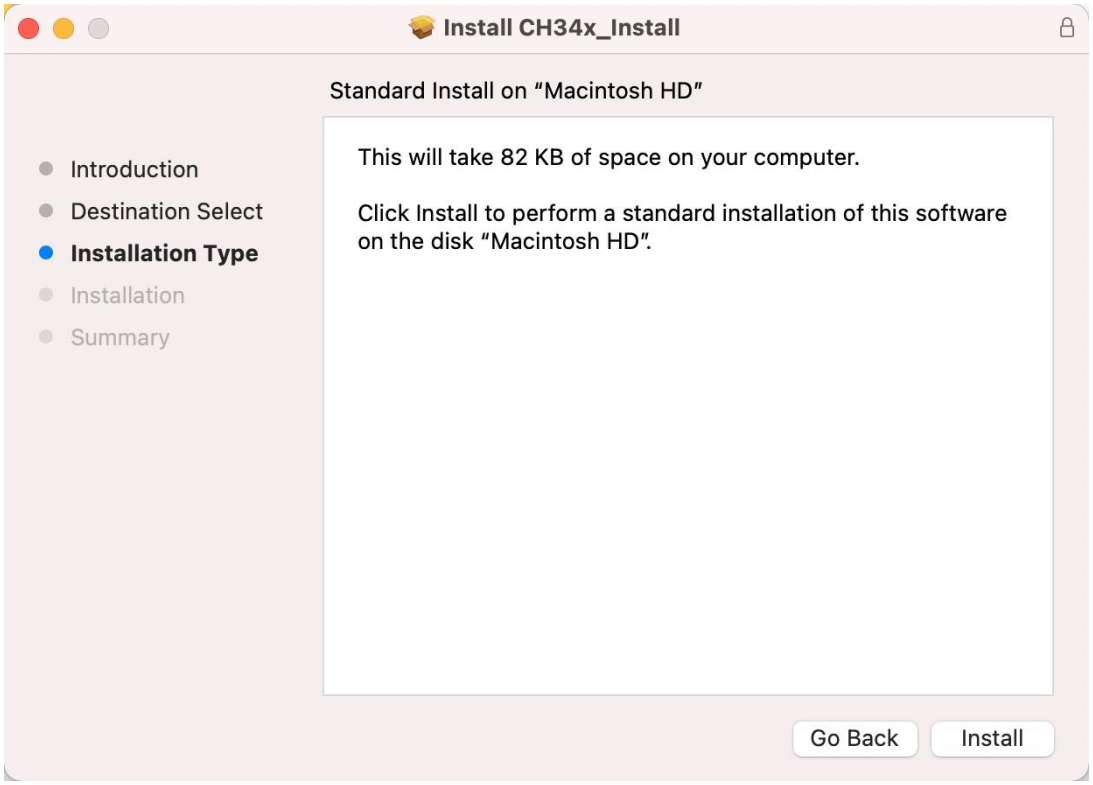
Second, open the folder "**Freenove_Basic_Starter_Kit_for_ESP8266/CH340/MAC/**"



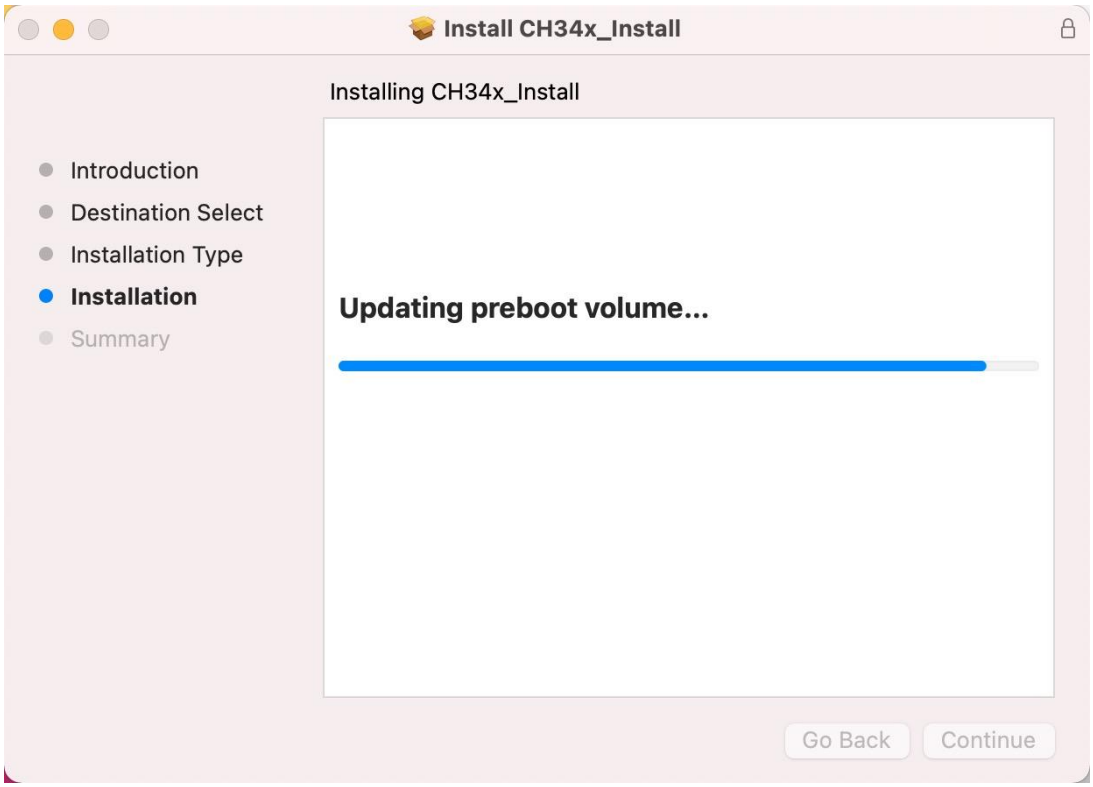
Third, click Continue.



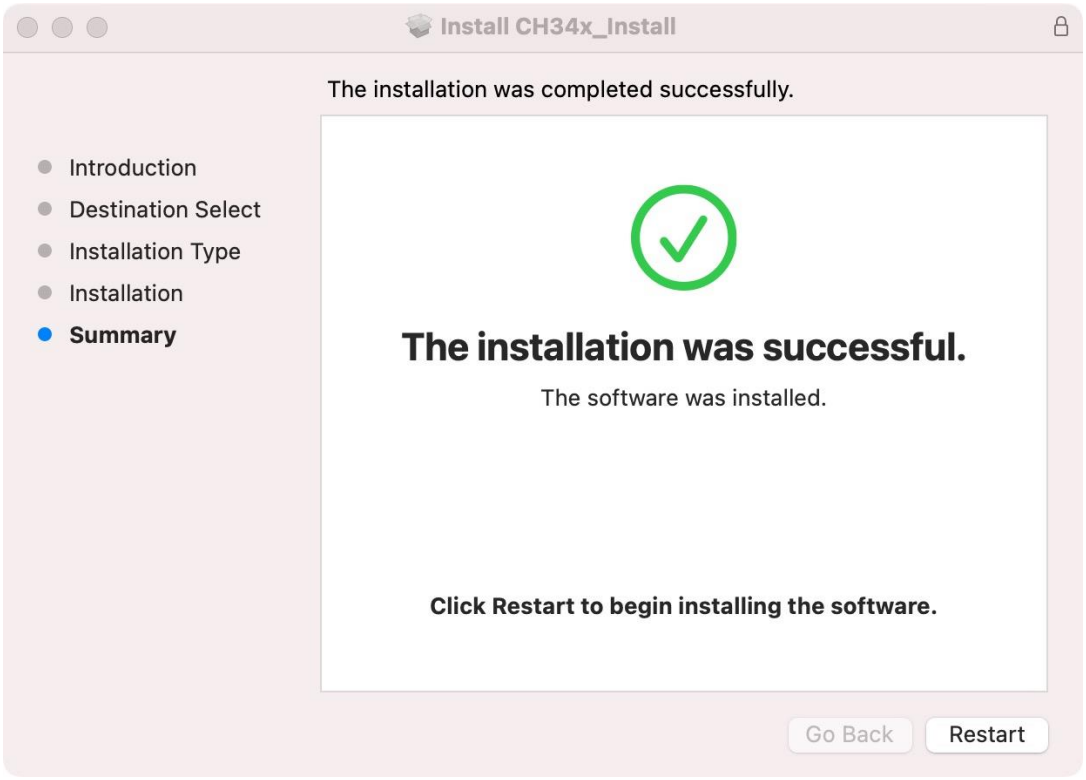
Fourth, click Install.


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

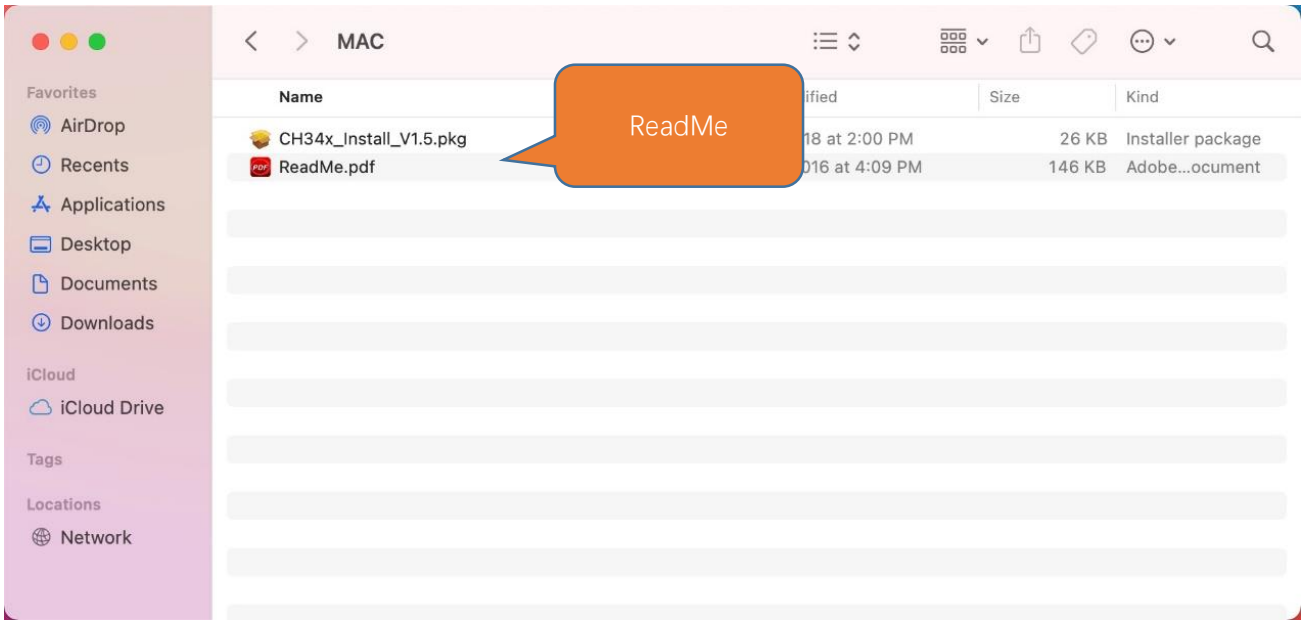
Then, waiting Finsh.



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.

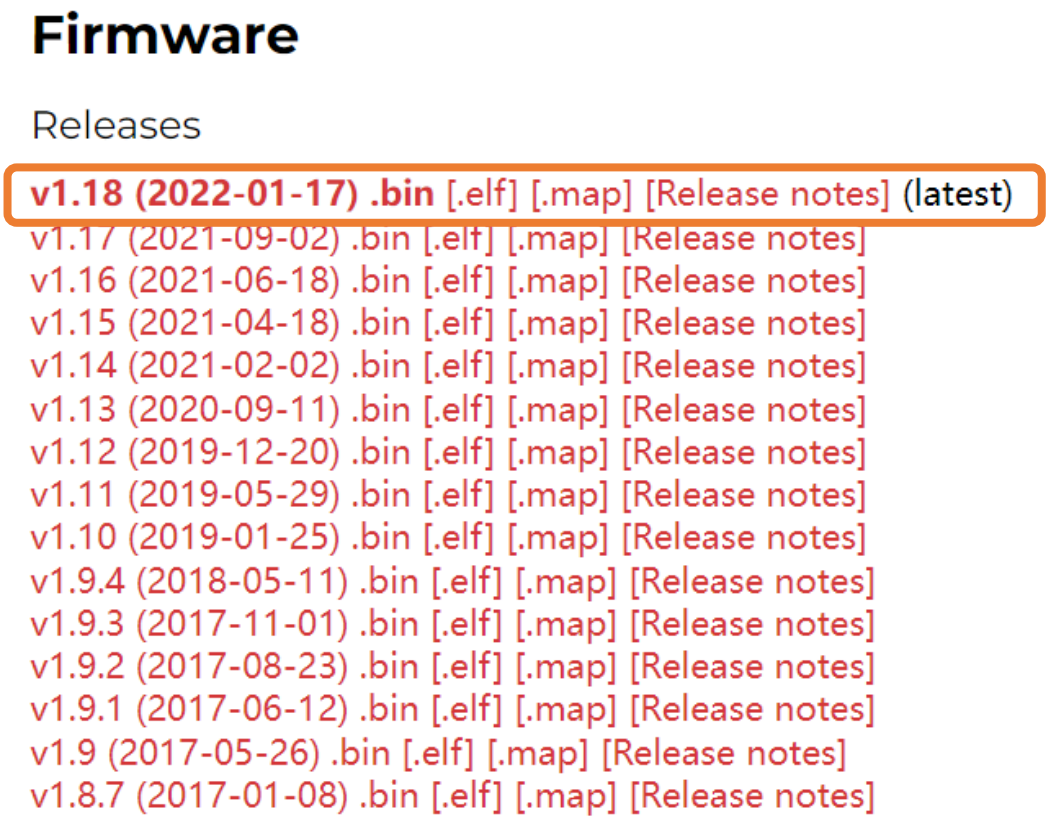


0.4 Burning Micropython Firmware (Important)

To run Python programs on ESP8266, we need to burn a firmware to ESP8266 first.

Downloading Micropython Firmware

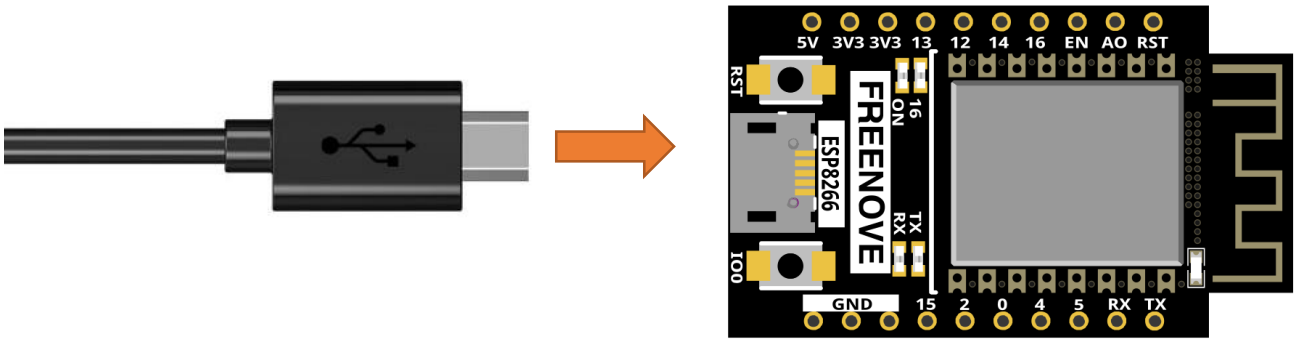
Official website of microPython: http://micropython.org/
Webpage listing firmware of microPython for ESP8266: https://micropython.org/download/esp8266/



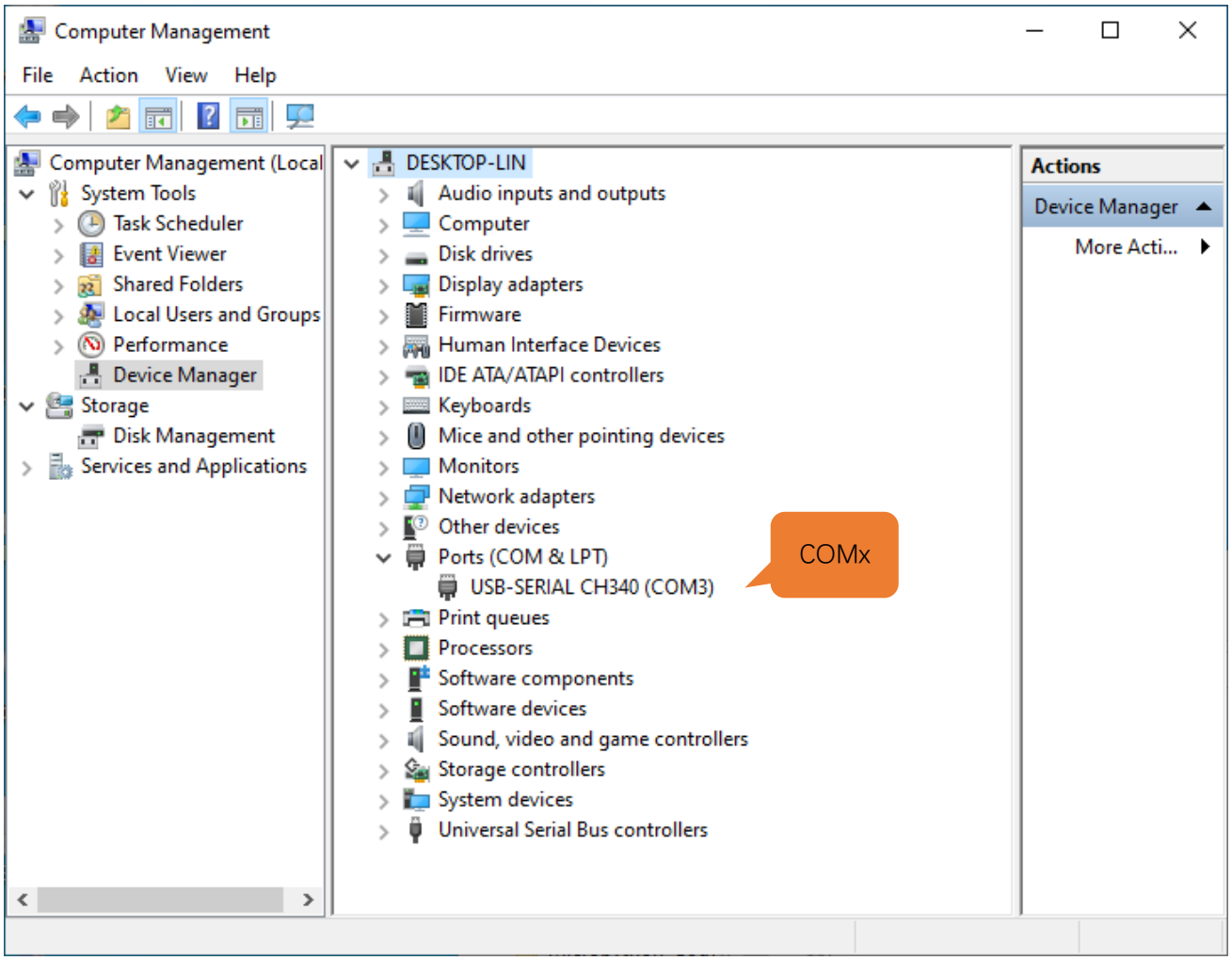
Firmware used in this tutorial is **esp8266-20220117-v1.18.bin**
Click the following link to download directly:
https://micropython.org/resources/firmware/esp8266-20220117-v1.18.bin
This file is also provided in our data folder "**Freenove_Basic_Starter_Kit_for_ESP8266 /Python/Python_Firmware**".

Burning a Micropython Firmware

Connect your computer and ESP8266 with a USB cable.

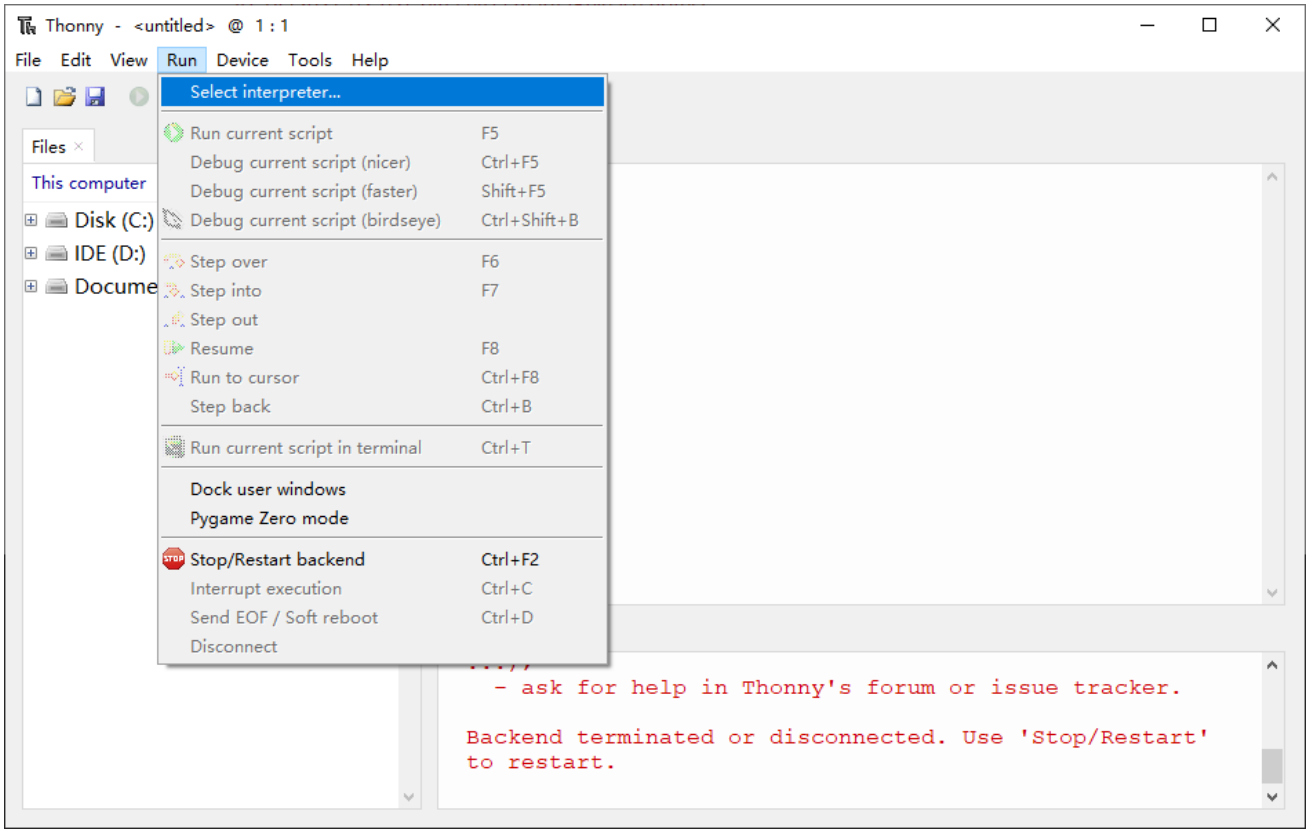


Make sure that the driver has been installed successfully and that it can recognize COM port correctly. Open device manager and expand "Ports".

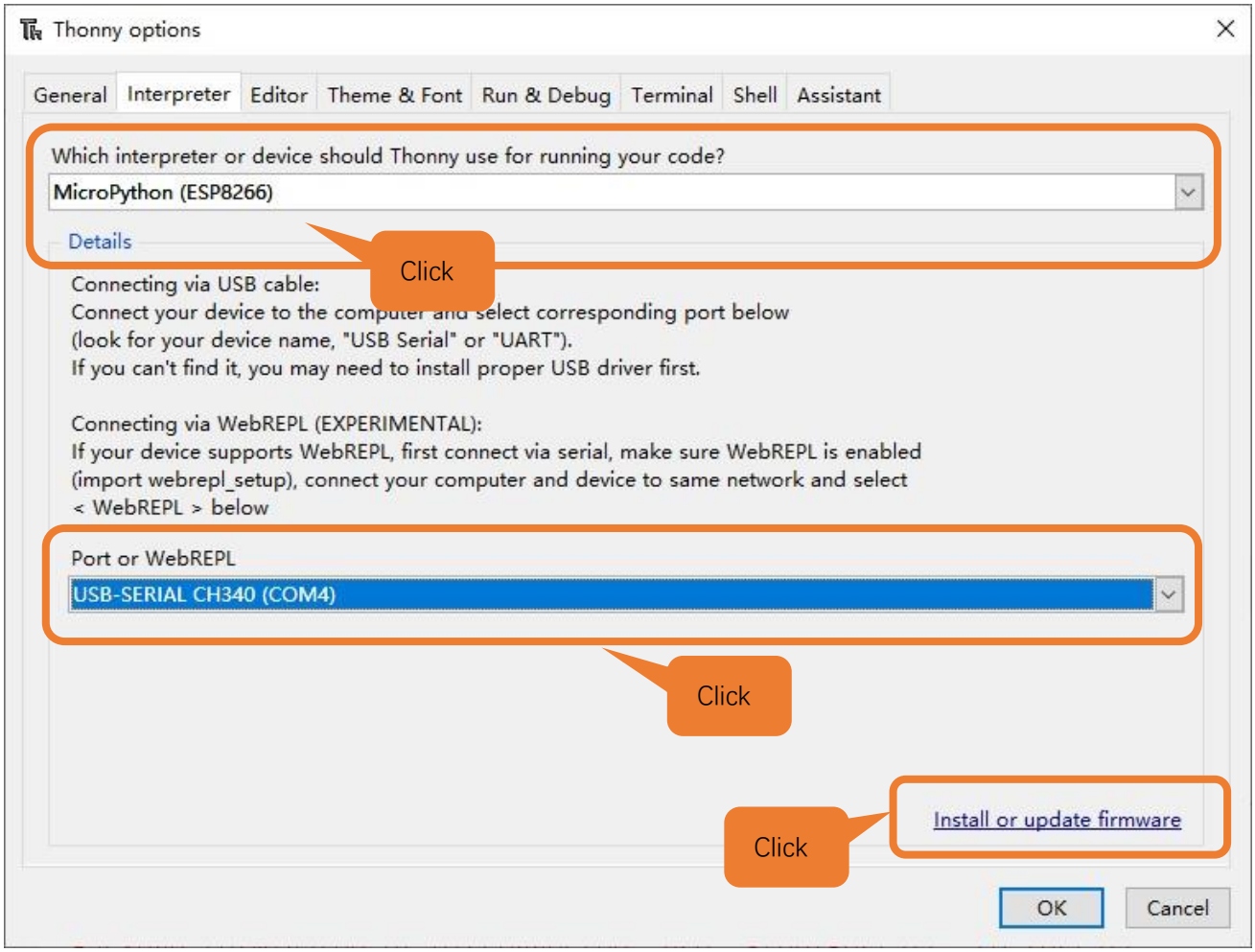


Note: the port of different people may be different, which is a normal situation.

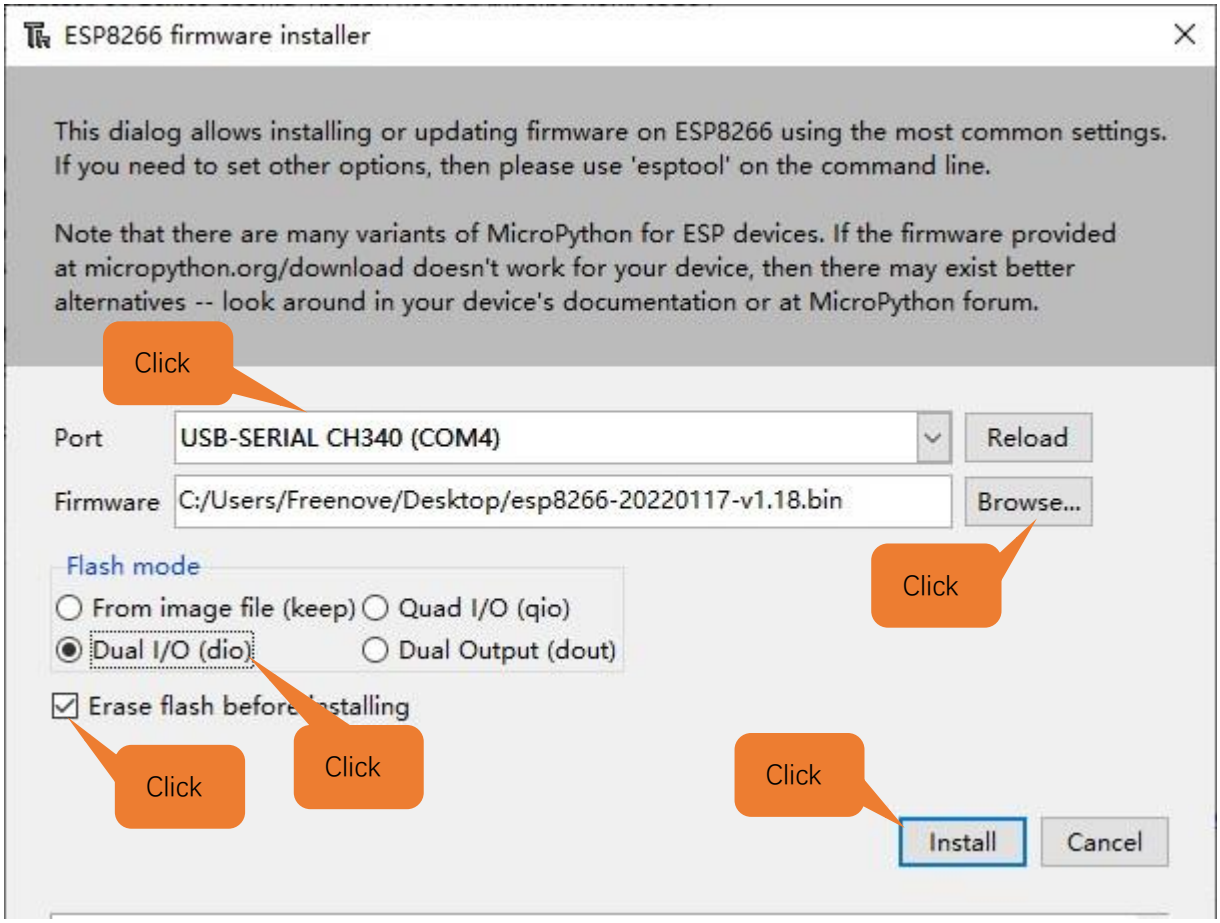
1. Open Thonny, click “run” and select “Select interpreter...””



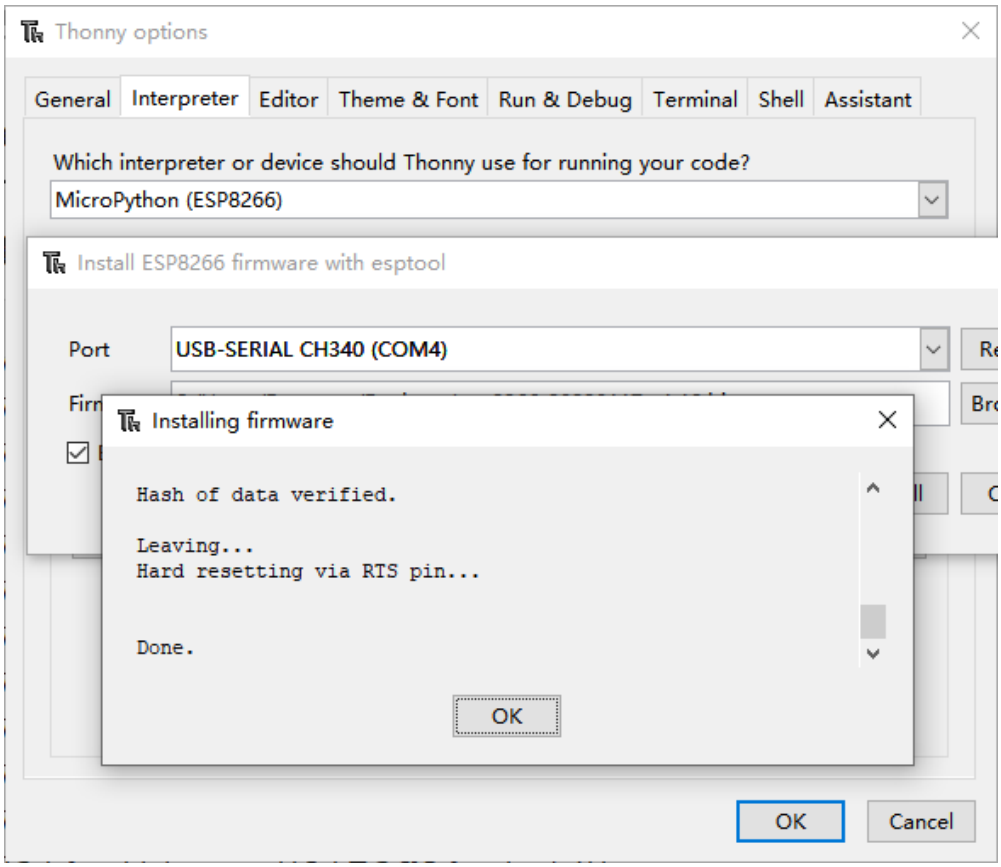
2. Select "Micropython (ESP8266)", select "USB-SERIAL CH340 (COM4)", and then click the long button under "Firmware".

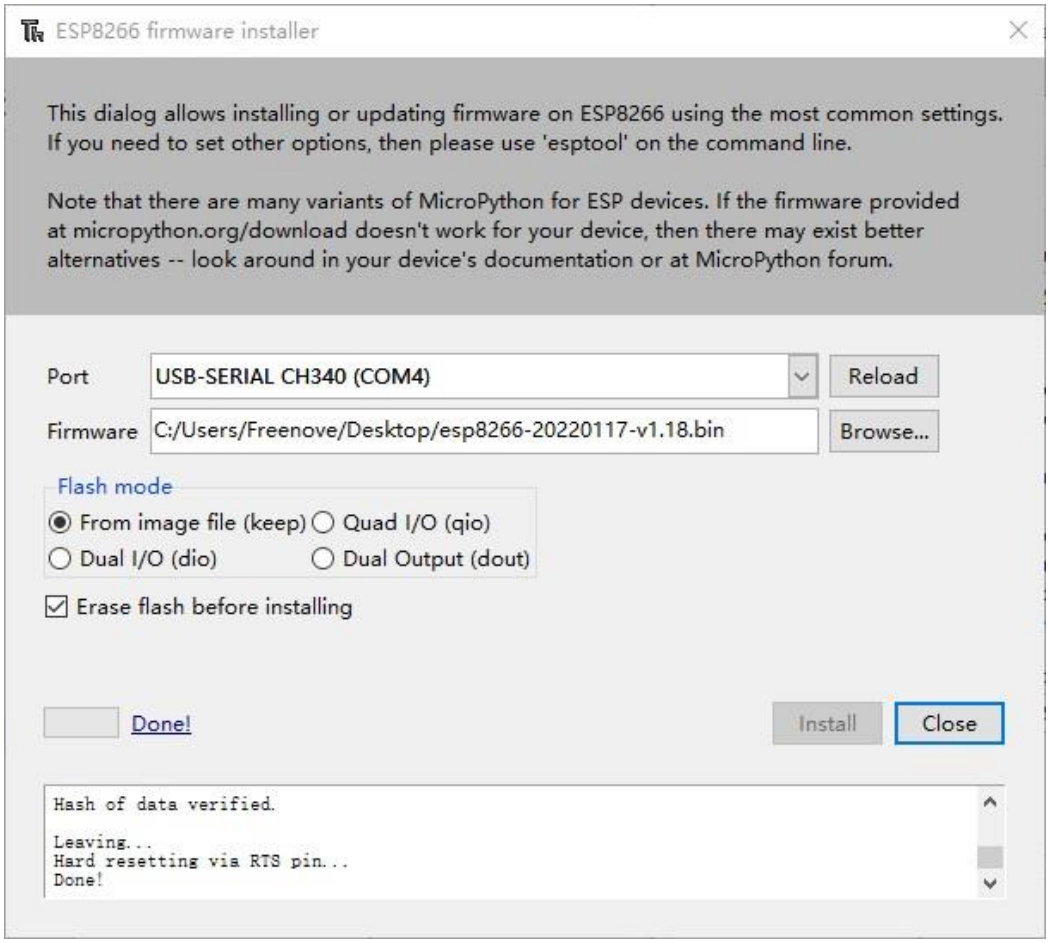


3. The following dialog box pops up. Select "USB-SERIAL CH340 (COM4)" for "Port" and then click "Browse...". Select the previous prepared microPython firmware "**esp8266-20220117-v1.18.bin**". Check "Erase flash before installing" and click "install" to wait for the prompt of finishing installation.
   Here we need to select Flash mode.On our ESP8266 development board, choose "DIO" mode or "DOUT" mode for better compatibility.If the ESP8266 module is abnormal, check whether the ESP8266 module works in the two modes.
   Flash works in DOUT, DIO, QOUT, and QIO modes.
   1.DOUT: Address is input in 1-line mode and data is output in 2-line mode.
   2.DIO: Address is input in 2-line mode and data is output in 2-line mode.
   3.QOUT: Address is input in 1-line mode and data is output in 4-line mode.
   4.QIO: Address is input in 4-line mode and data is output in 4-line mode.
   If you need to use the QIO mode, ensure that the Flash supports the QIO mode.

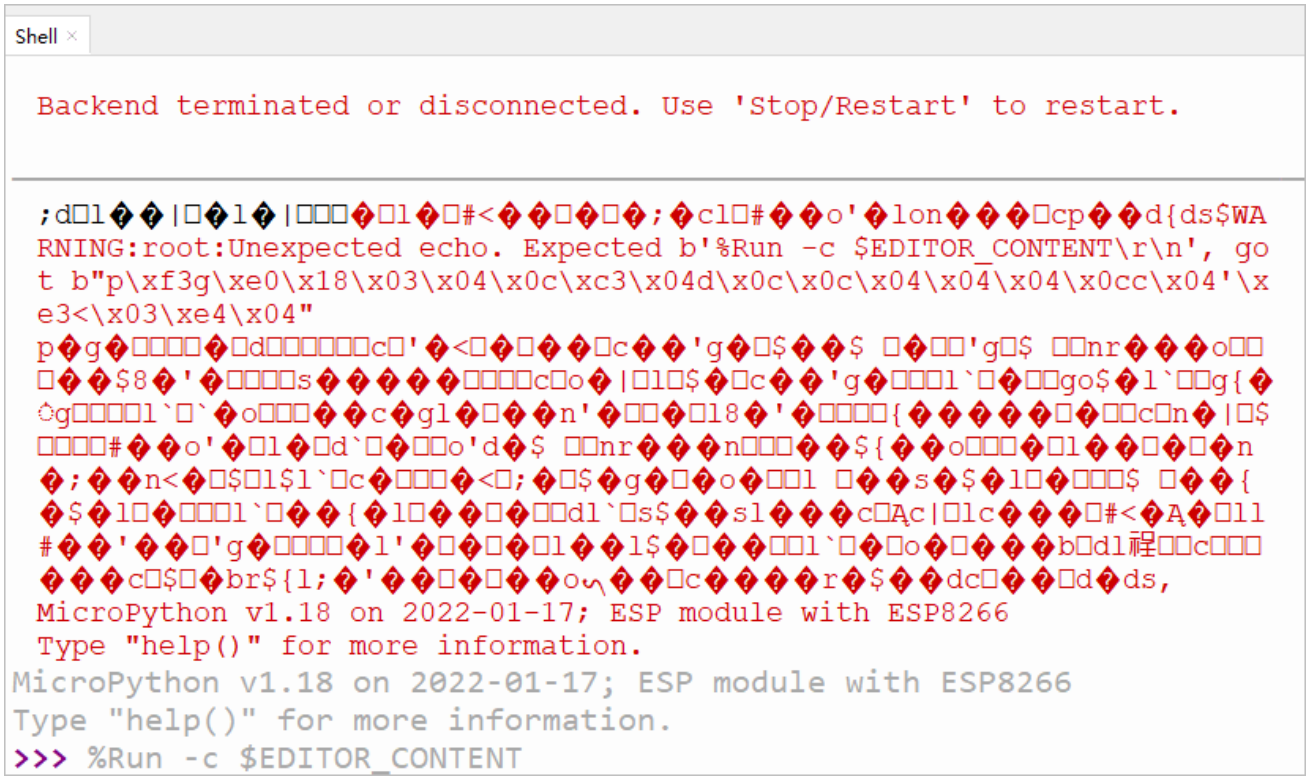


4. Wait for the installation to be done.

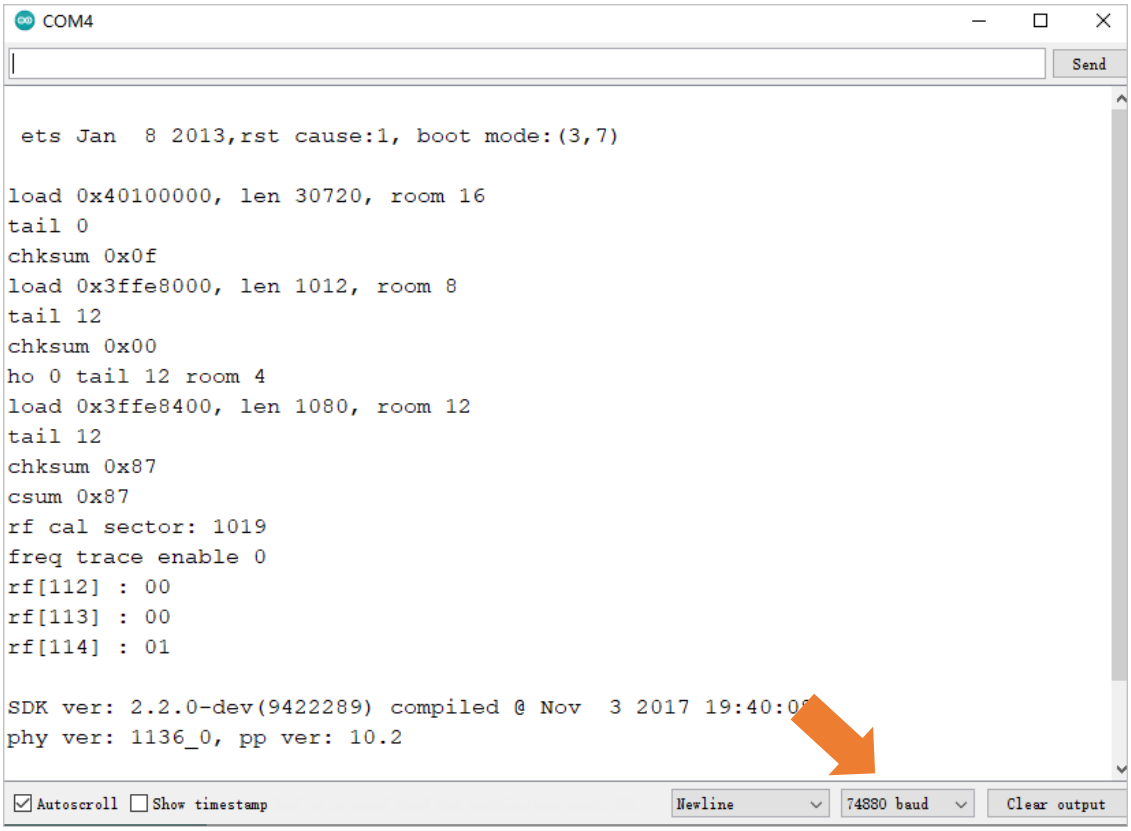




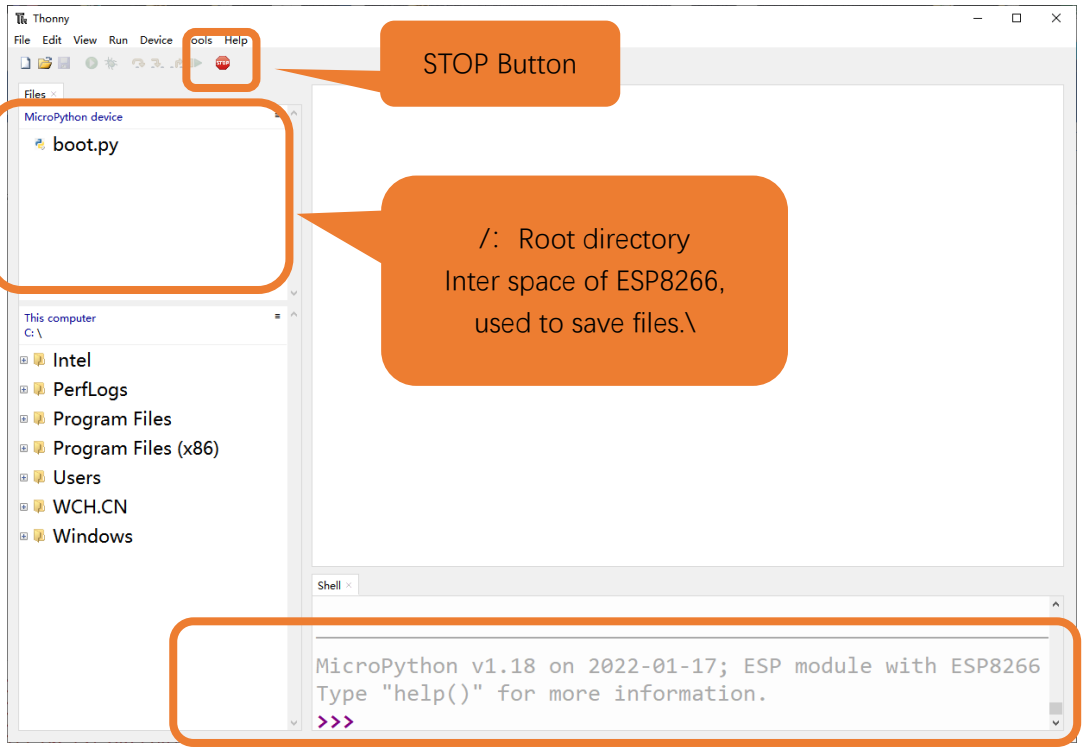
After burning the Micropython firmware, "shell" will display some garbled characters, please do not worry, the garbled characters are displayed as follows:



When the ESP8266 is powered on, the default baud rate is 74880. The default communication and serial port in the ESP8266 firmware is 115200. So if you set the serial port to 74880, this time can be displayed normally. Here, we use The Arduino IDE serial port tool for output and display. The details are as follows:



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

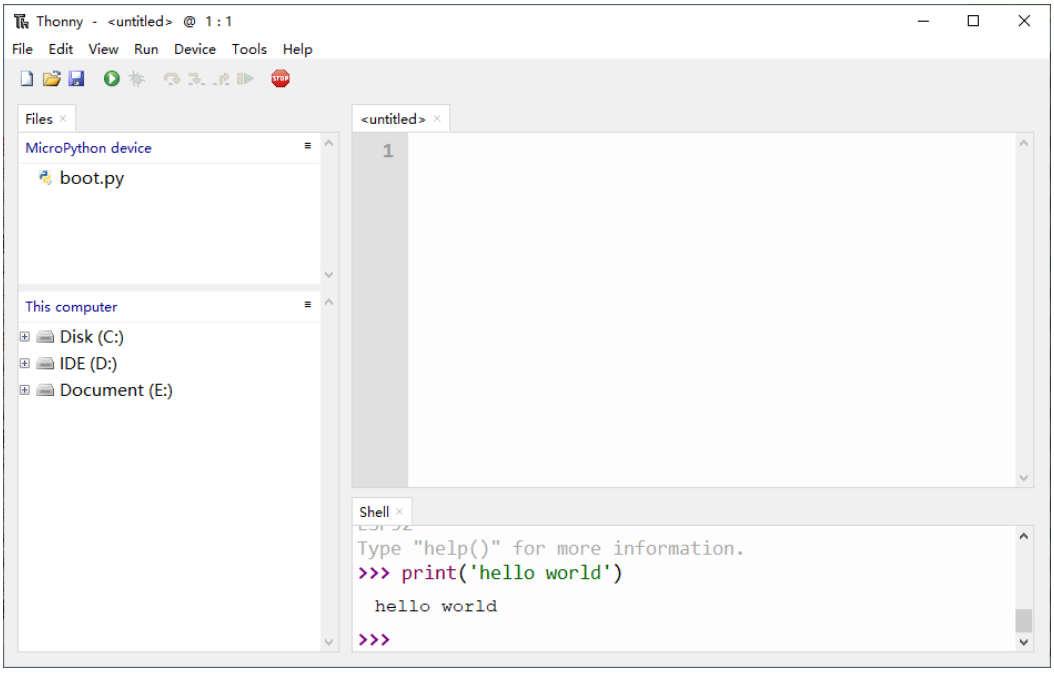


6. So far, all the preparations have been made.

0.5 Testing codes (Important)

Testing Shell Command

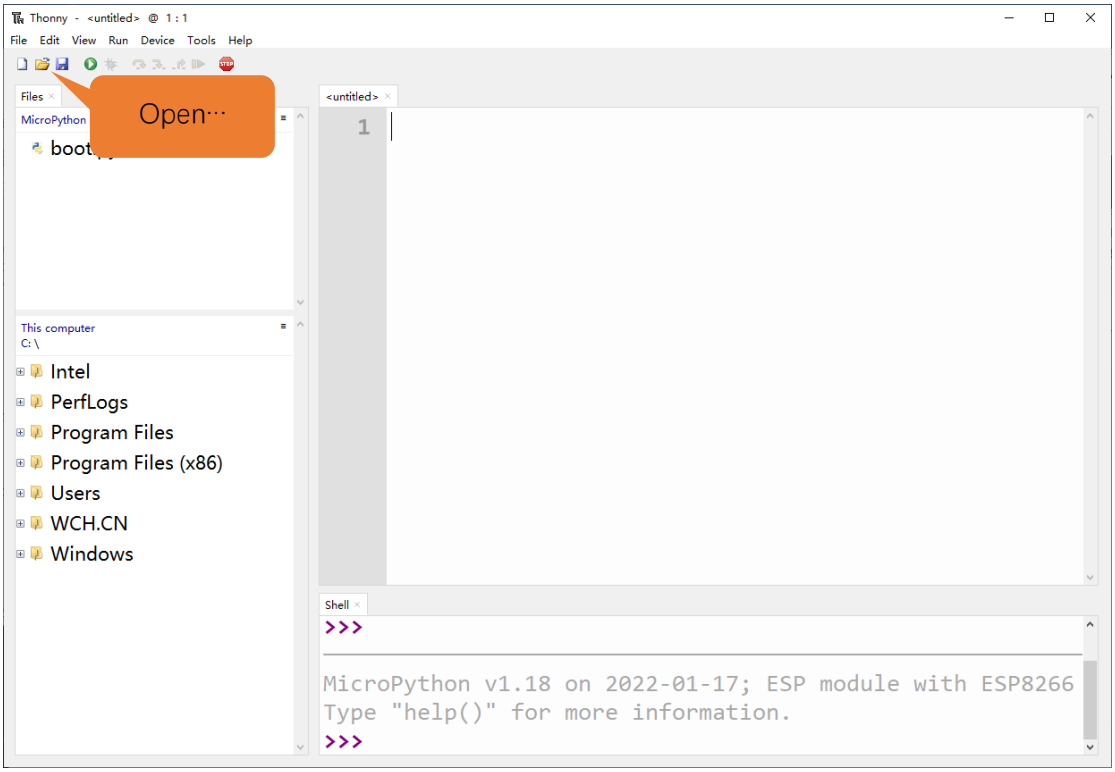
Enter "print('hello world')" in "Shell" and press Enter.



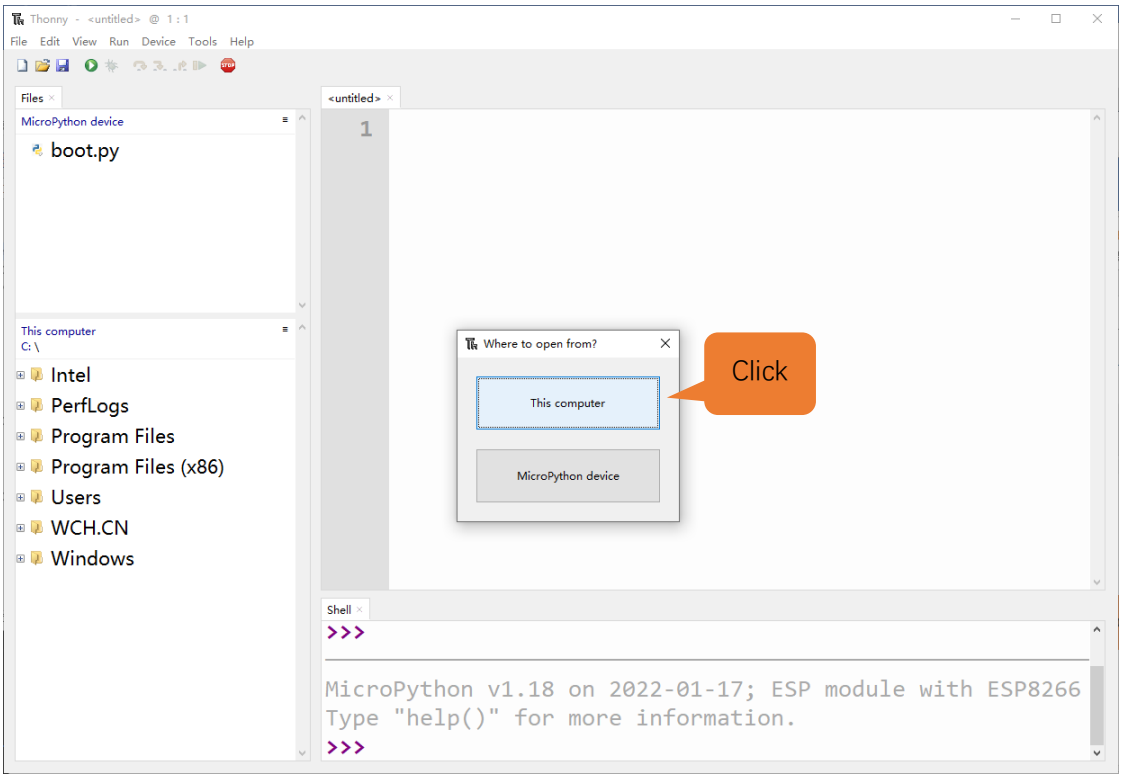
Running Online

ESP8266 needs to be connected to a computer when it is run online. Users can use Thonny to writer and debug programs.

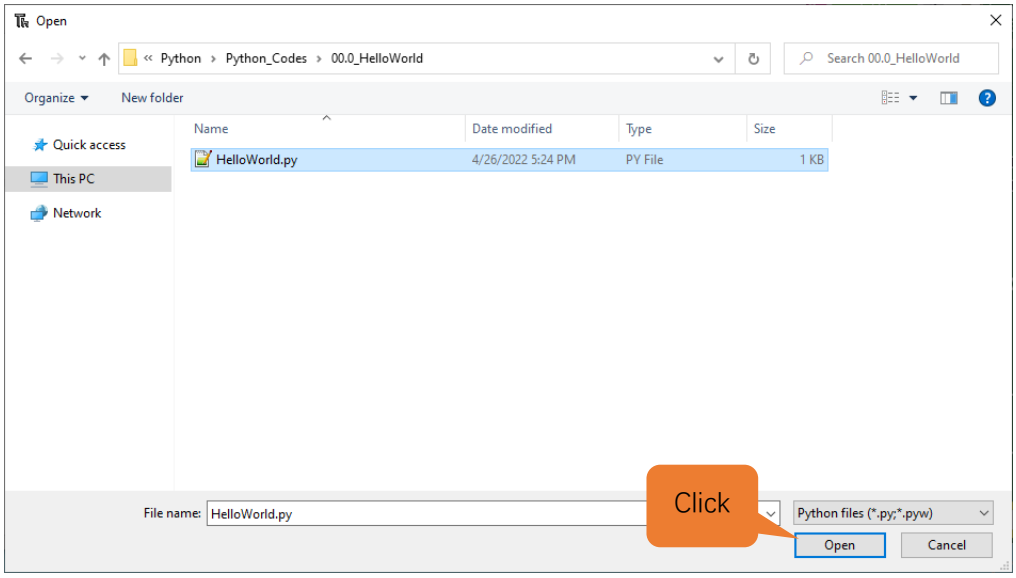
1. Open Thonny and click “Open···”.



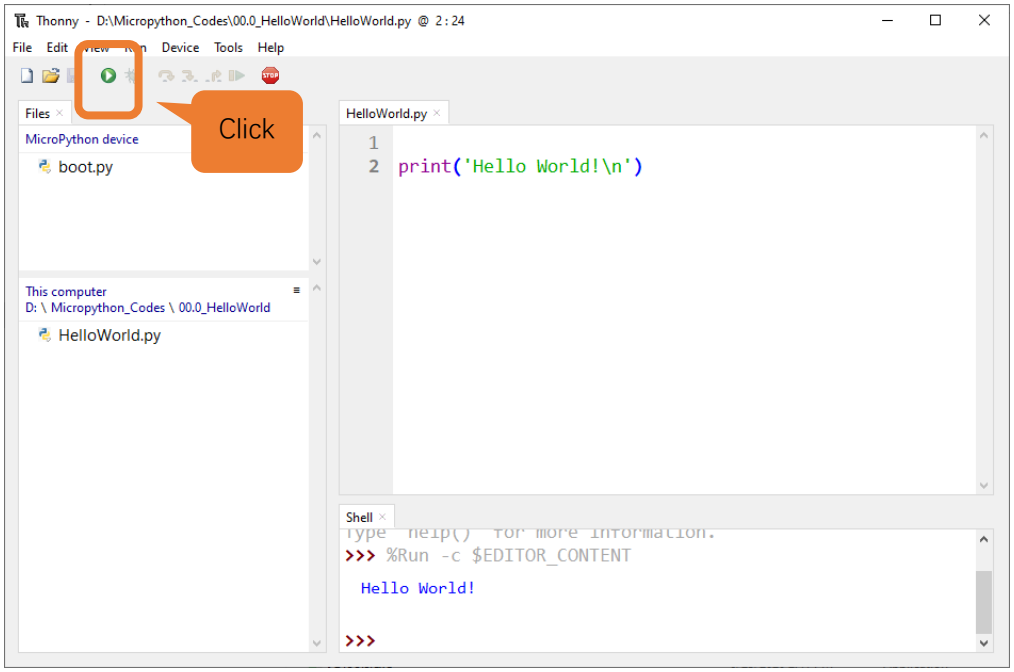
2. On the newly pop-up window, click “This computer”.



In the new dialog box, select "**HelloWorld.py**" in "**Freenove_Basic_Starter_Kit_for_ESP8266/Python/Python_Codes/00.0_HelloWorld**" folder.



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

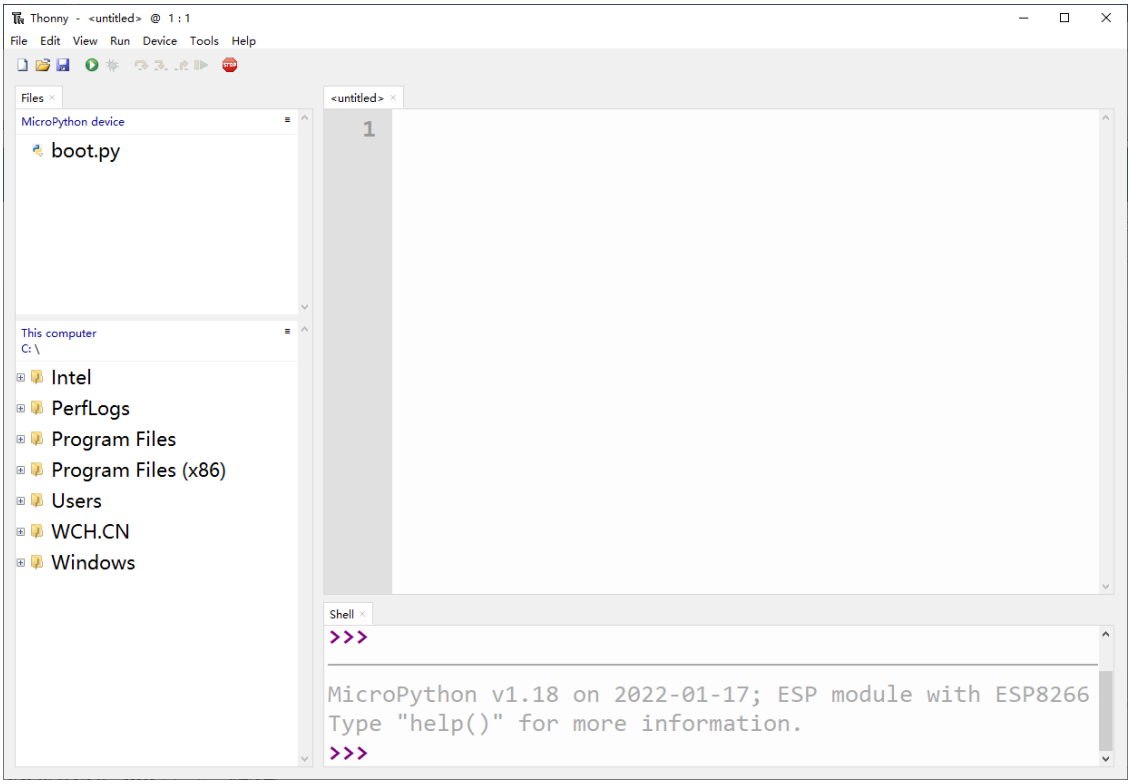


Note: When running online, if you press the reset key of ESP8266, user's code will not be executed again. If you wish to run the code automatically after resetting the code, please refer to the following Running Offline.

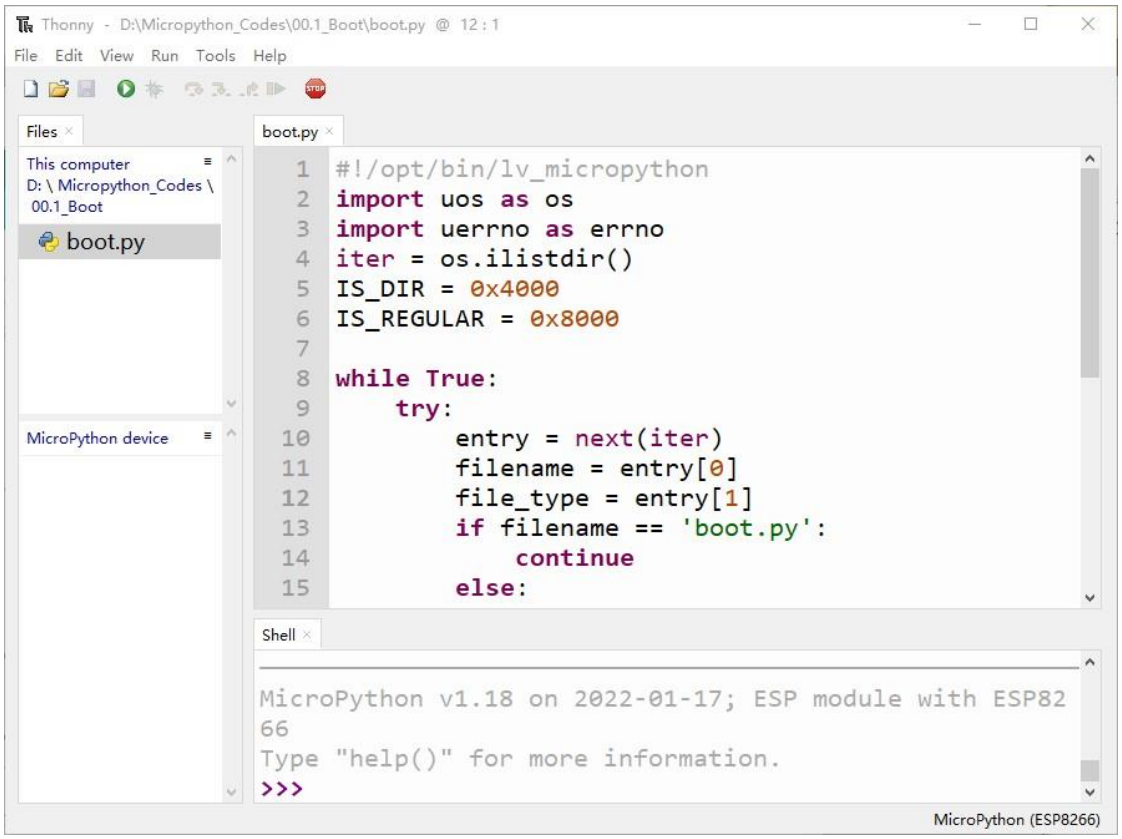
Running Offline（Important）

After ESP8266 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 ESP8266 execute user's programs after resetting, we need to add a guiding program in boot.py to execute user's code.

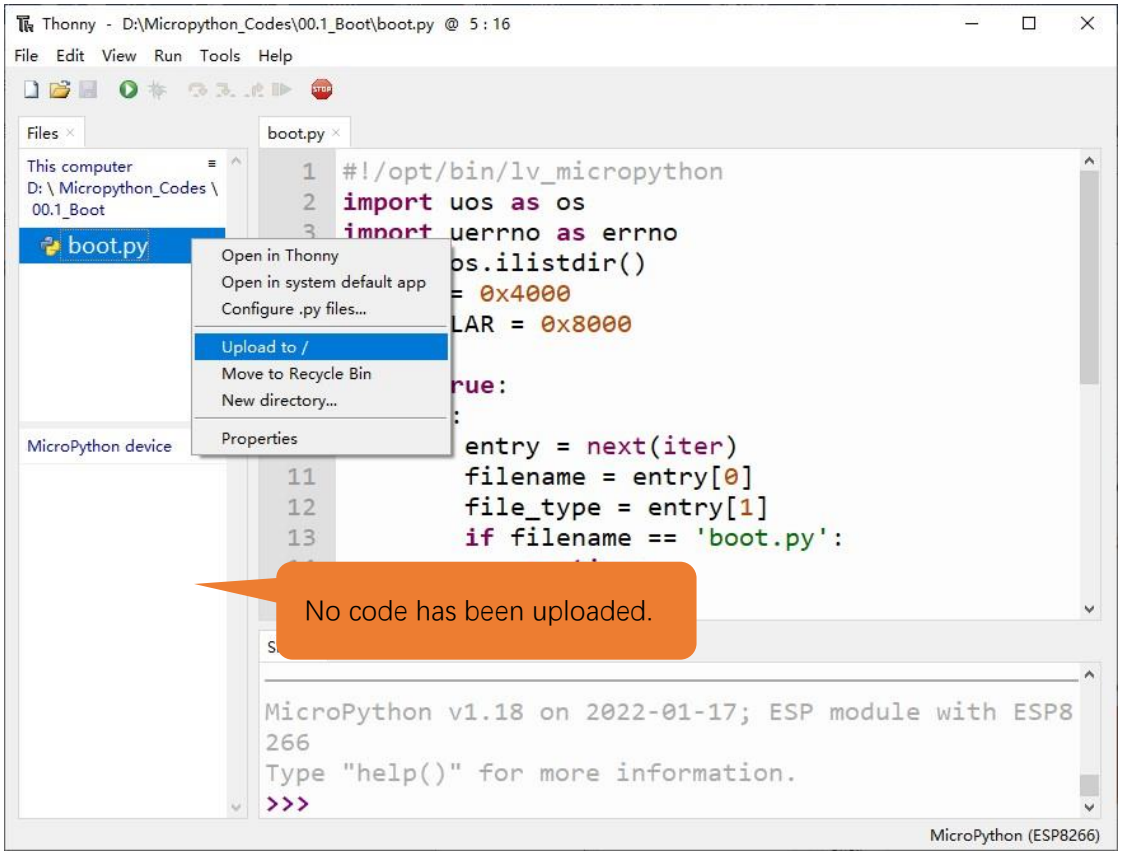
1. Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**". Open "Thonny".



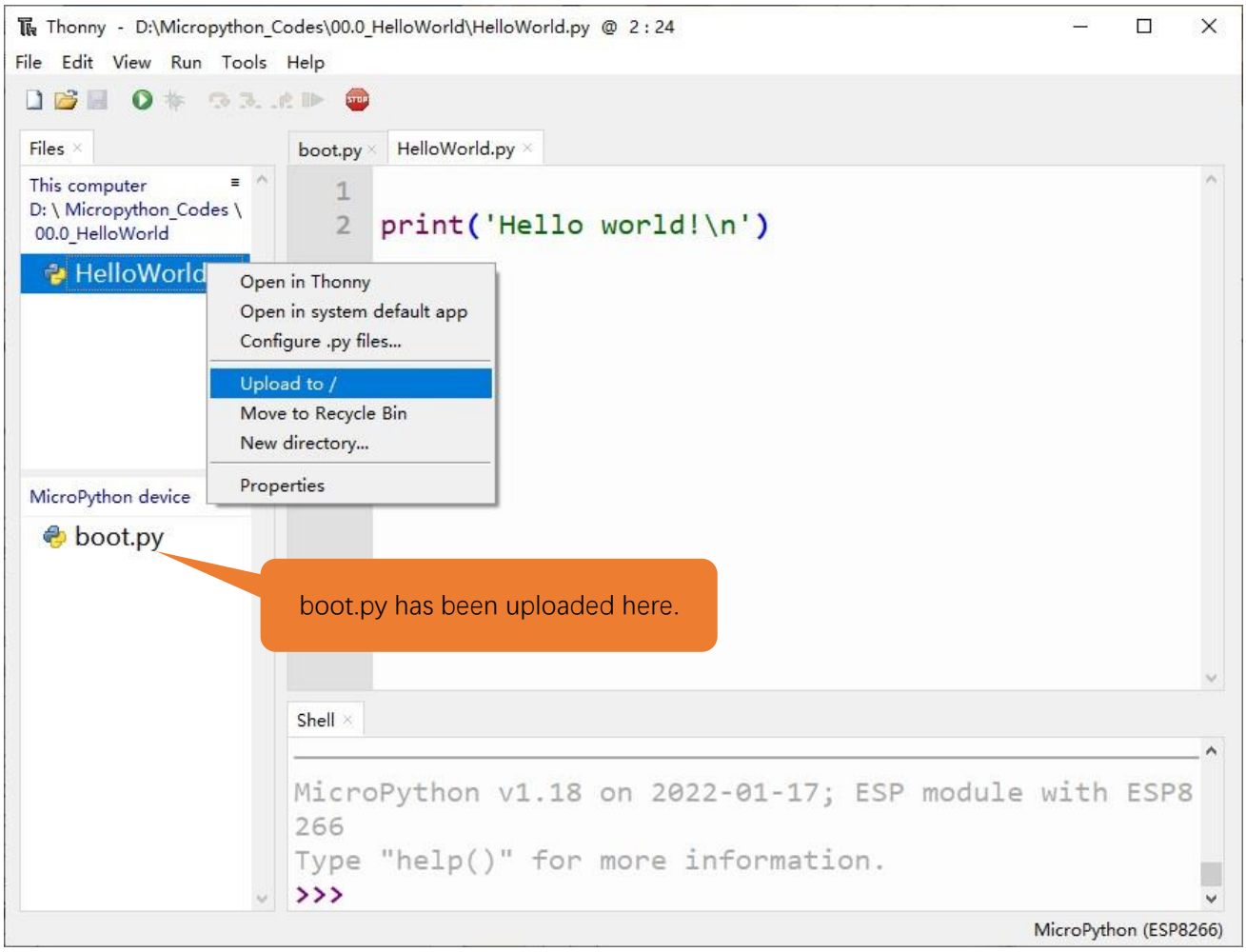
2. Expand "00.1_Boot" in the "Micropython_Codes" in the directory of disk(D), and double-click boot.py, which is provided by us to enable programs in "MicroPython device" to run offline.



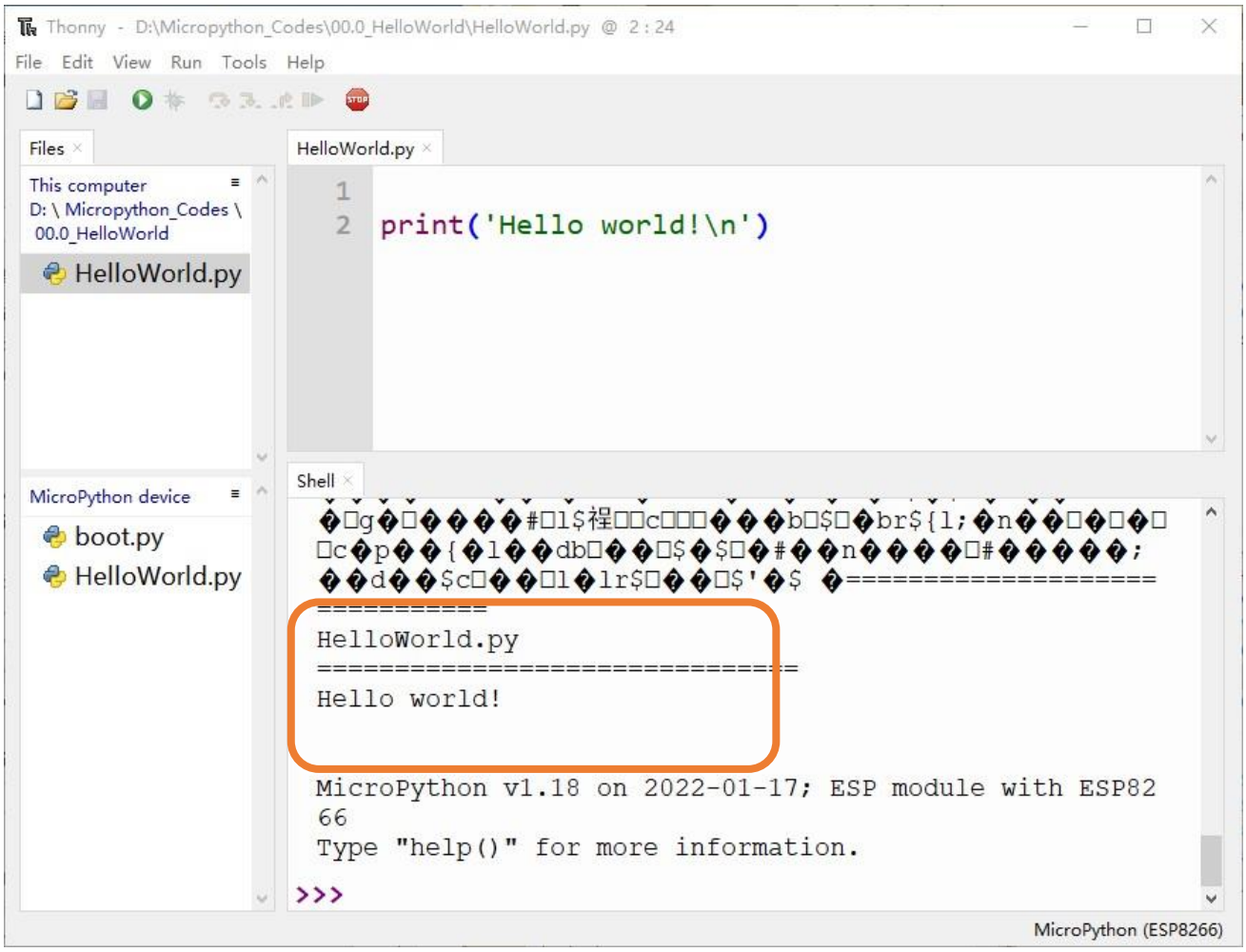
If you want your written programs to run offline, you need to upload boot.py we provided and all your codes to "MicroPython device" and press ESP8266's reset key. Here we use programs 00.0 and 00.1 as examples. Select "boot.py", right-click to select "Upload to /".



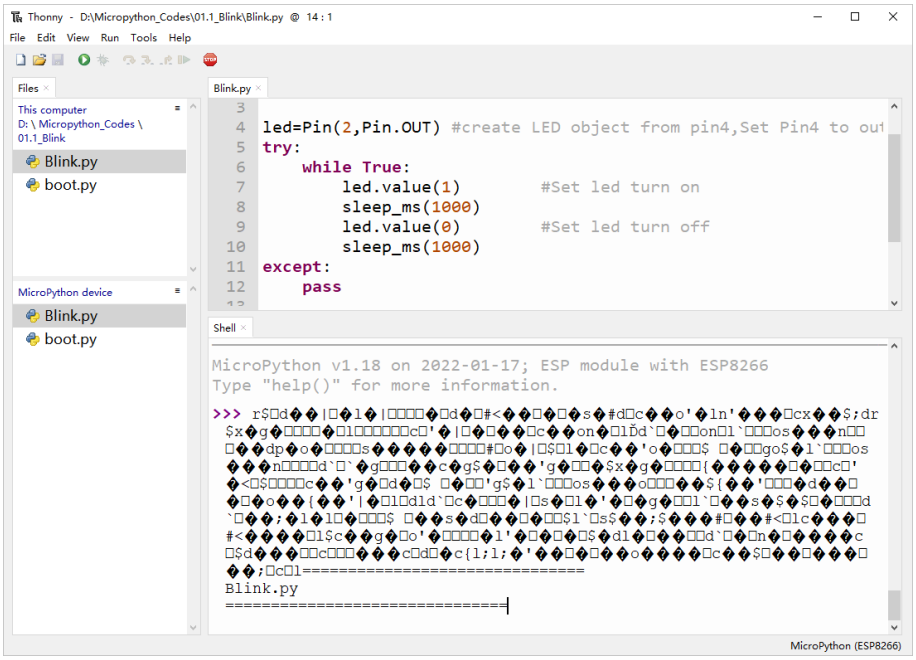
Similarly, upload “HelloWorld.py” to “MicroPython device”.



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

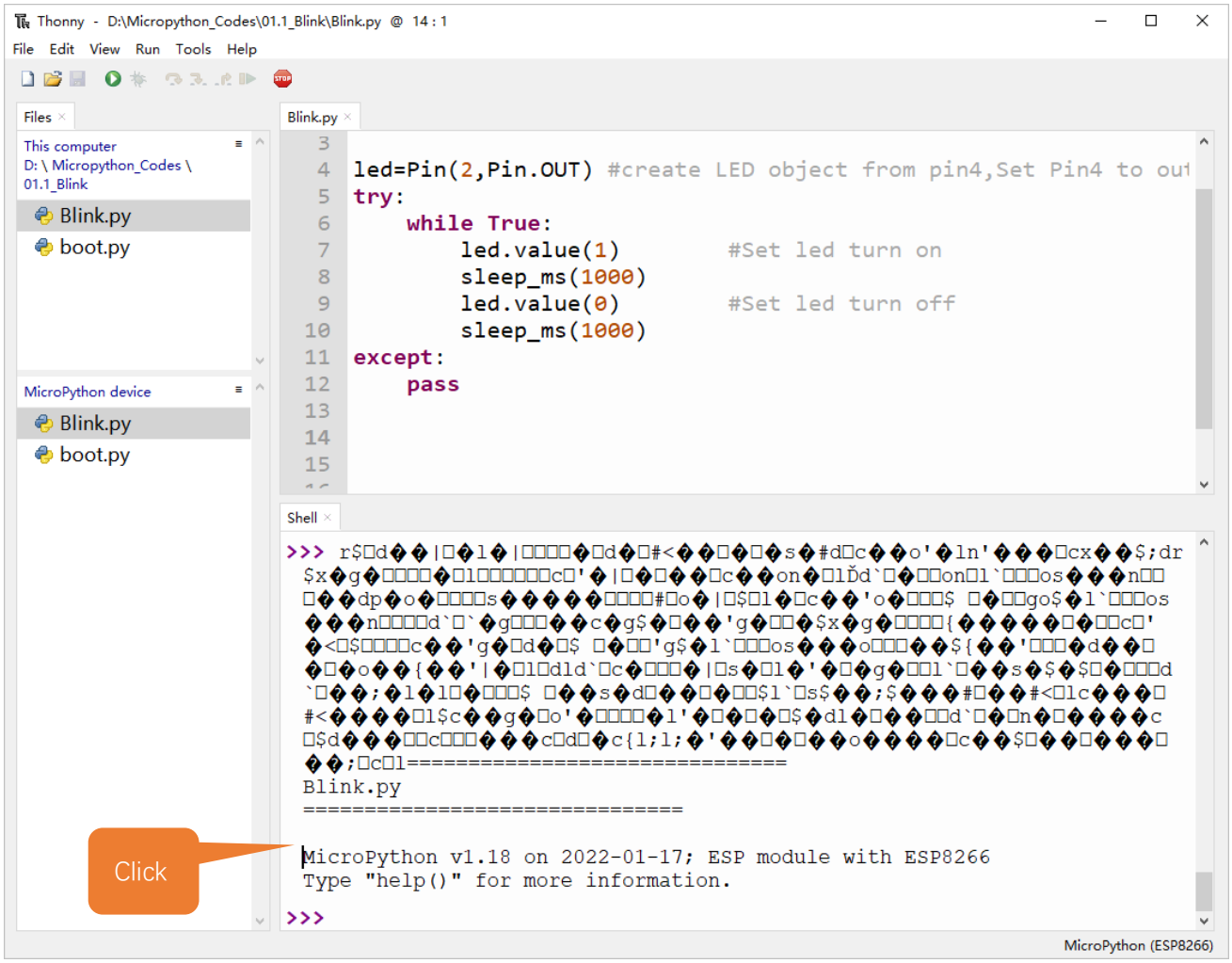


When you press the Reset key to run the offline code, the program will continue to execute while the ESP8266 is powered on.

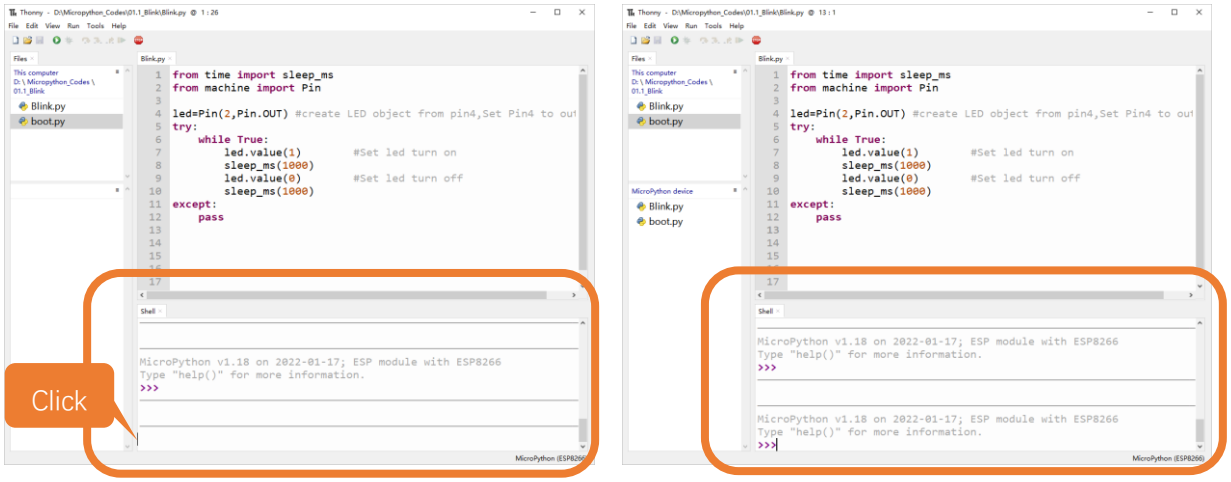


When you run offline code, you can exit the running program by pressing "CTRL" and "C" at the same time.

Before pressing the keyboard, click "Shell" with the mouse, and then press the keyboard key.



When your "Shell" is unresponsive or abnormal, you can exit the running program by pressing "CTRL" and "C" simultaneously.

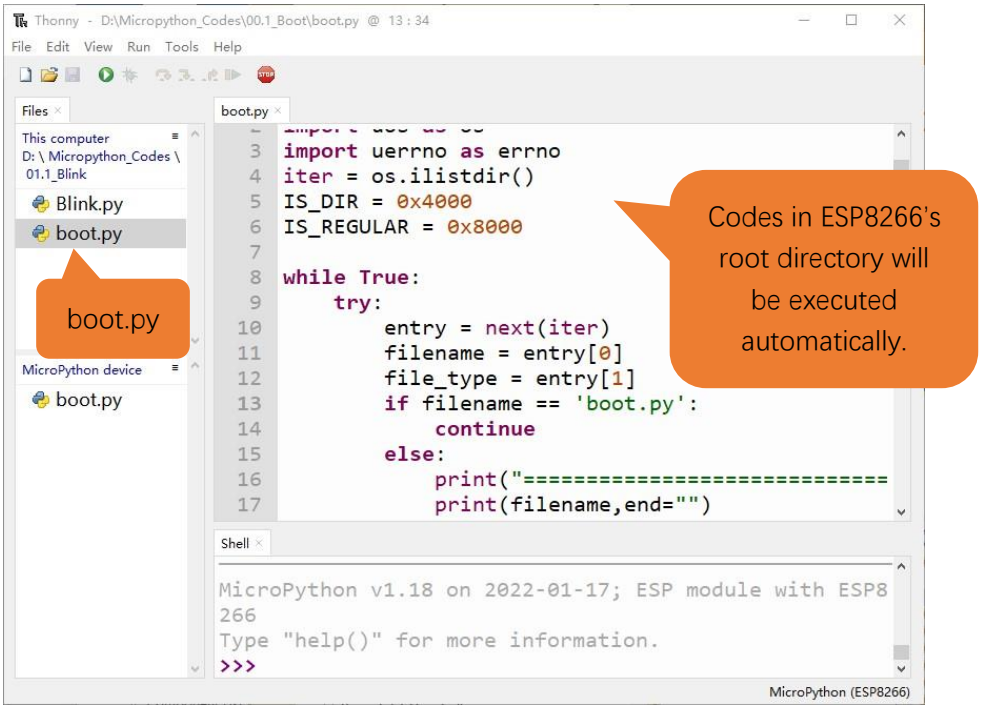


If the ESP8266 does not work properly, you can press CTRL and C at the same time to observe whether the Shell responds. If the ESP8266 still does not work properly, you can also rewrite the Micropython firmware and perform related operations again.

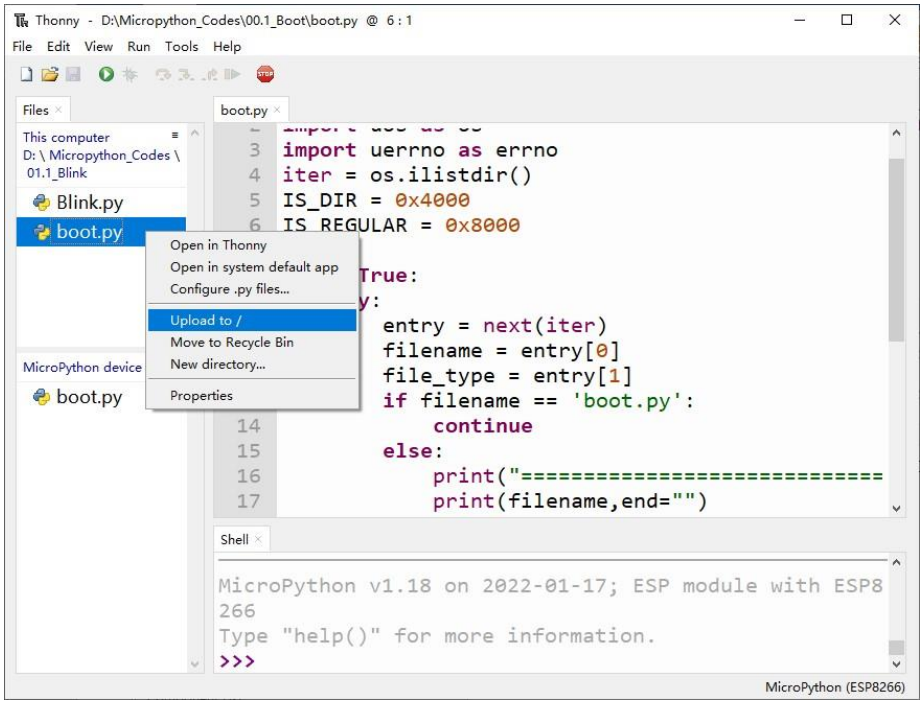
0.6 Thonny Common Operation

Uploading Code to ESP8266

For convenience, we take the opertation on “boot.py” as an example here. We have added “boot.py” to every code directory. Each time when ESP8266 restarts, if there is a “boot.py” in the root directory, it will execute this code first.

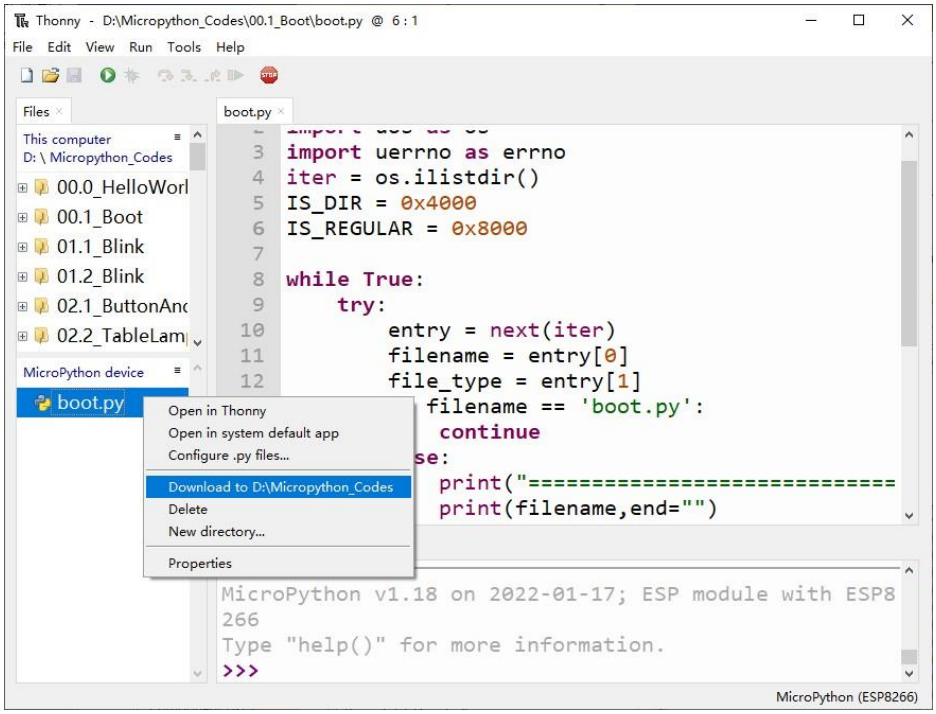


Select “Blink.py” in “01.1_Blink”, right-click your mouse and select “Upload to /” to upload code to ESP8266’s root directory.



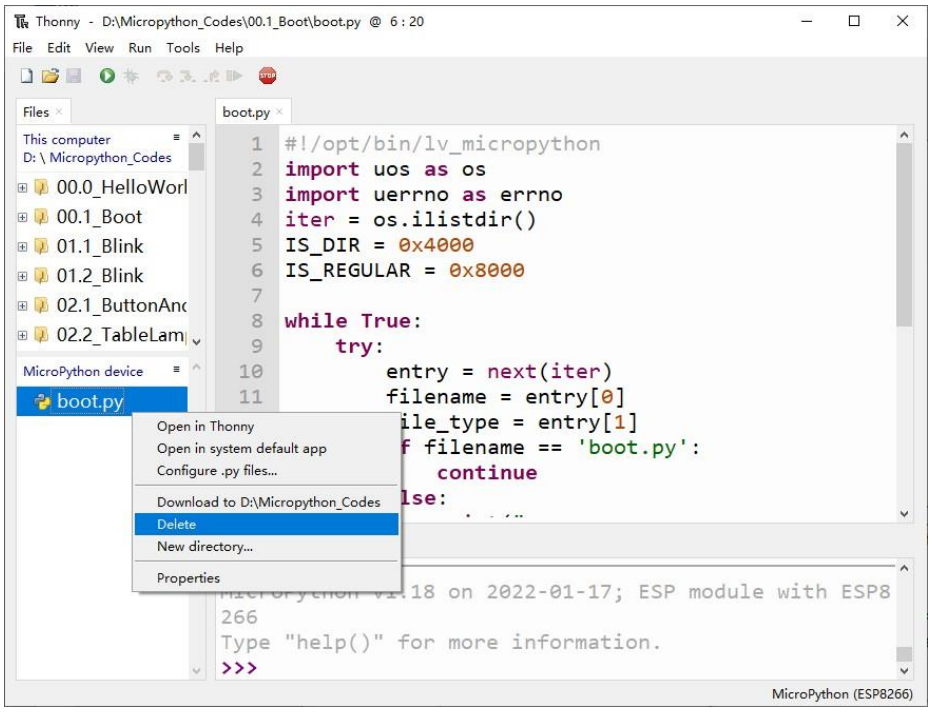
Downloading Code to Computer

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



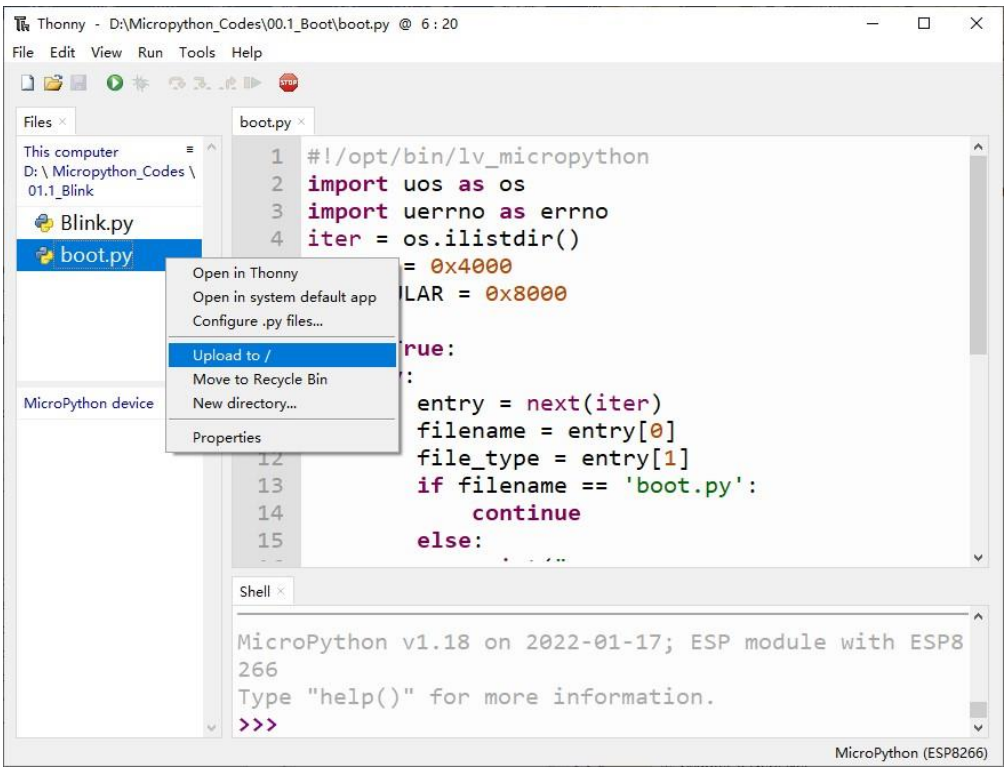
Deleting Files from ESP8266’s Root Directory

Select “boot.py” in “MicroPython device”, right-click it and select “Delete” to delete “boot.py” from ESP8266’s root directory.



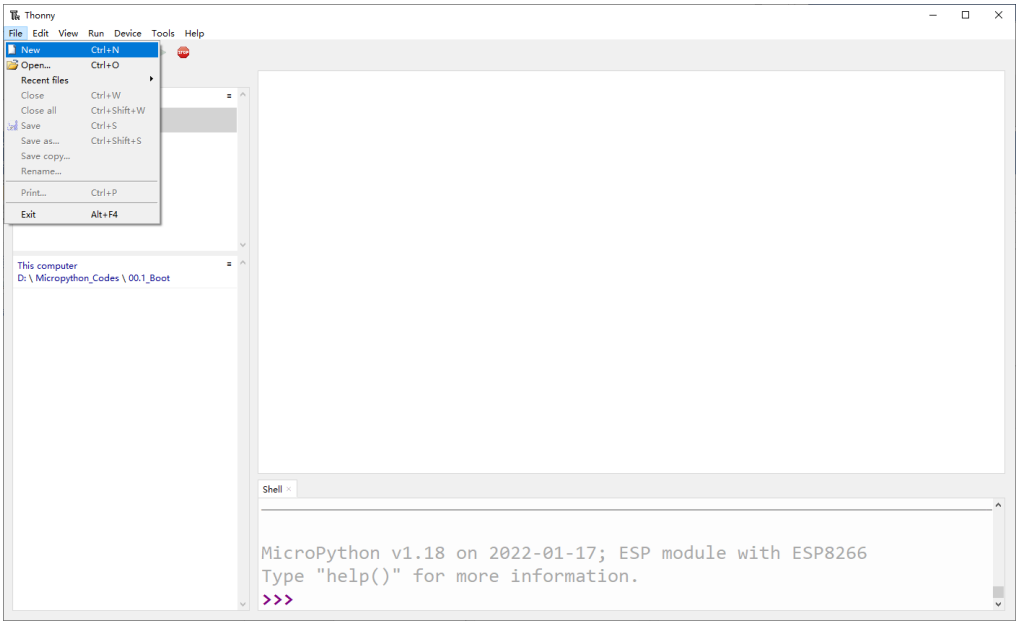
Deleting Files from your Computer Directory

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

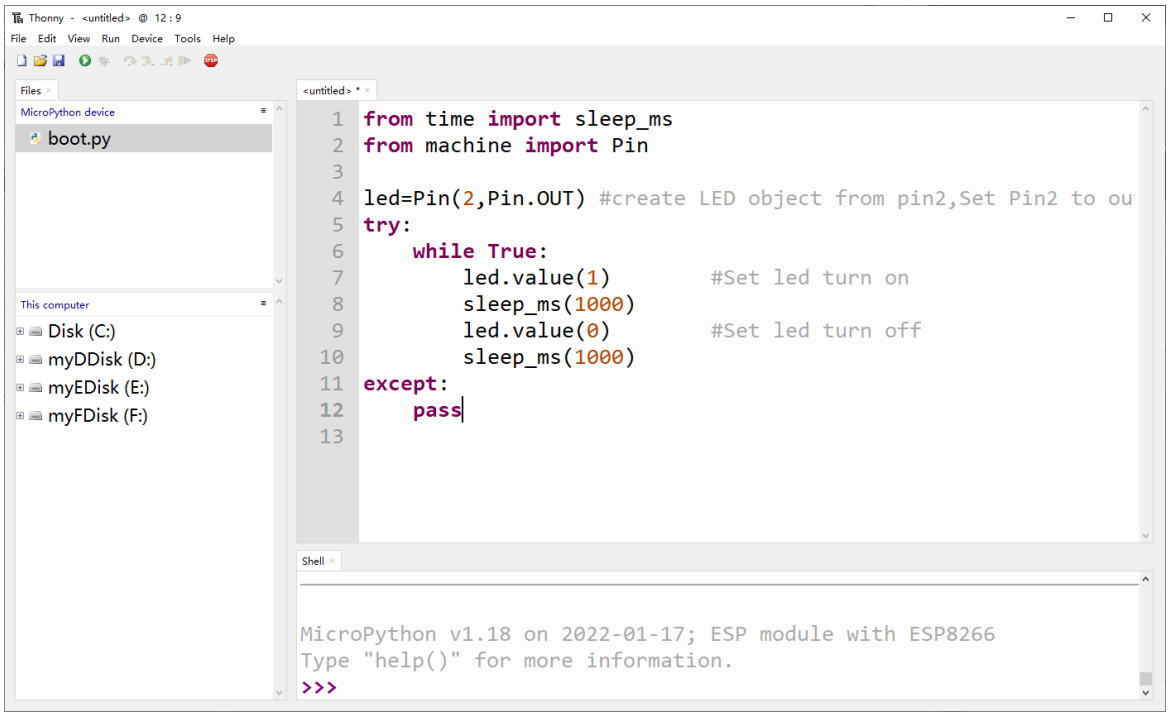


Creating and Saving the code

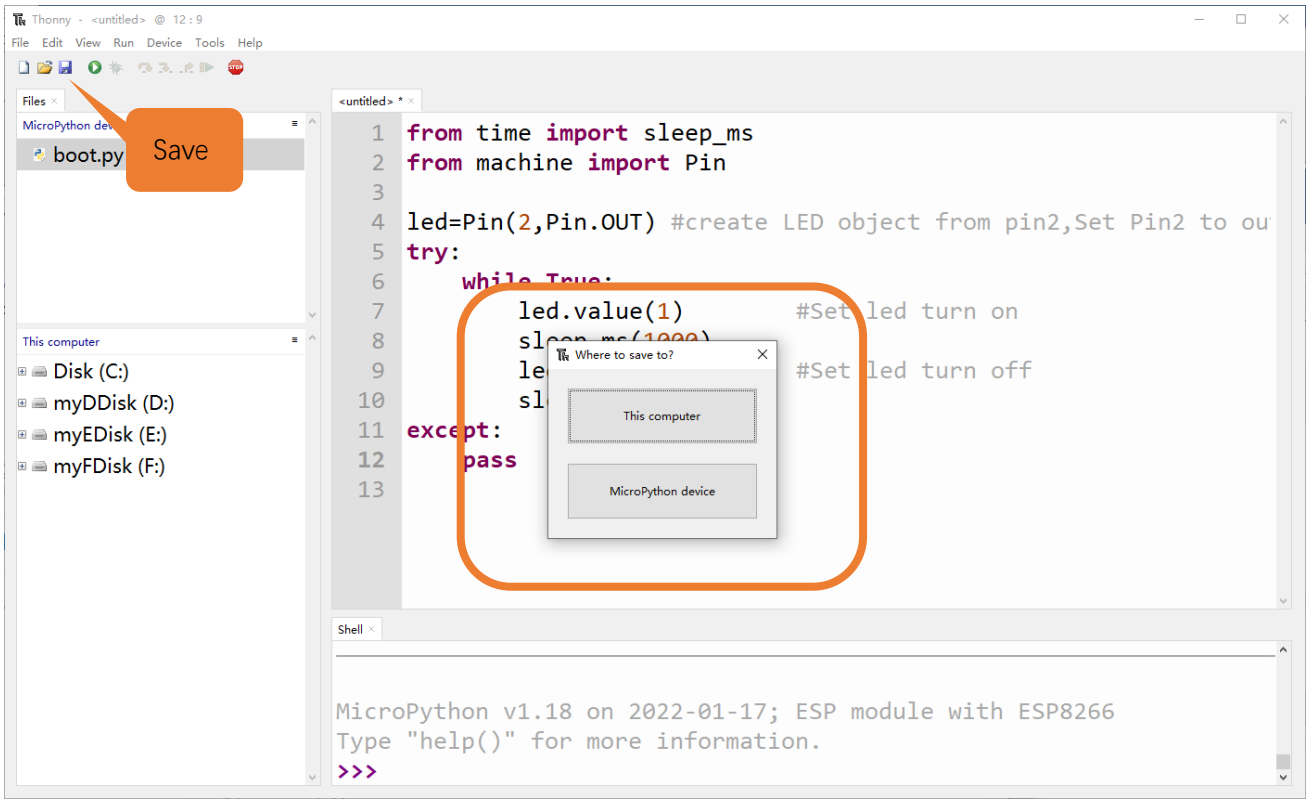
Click "File"→"New" to create and write codes.



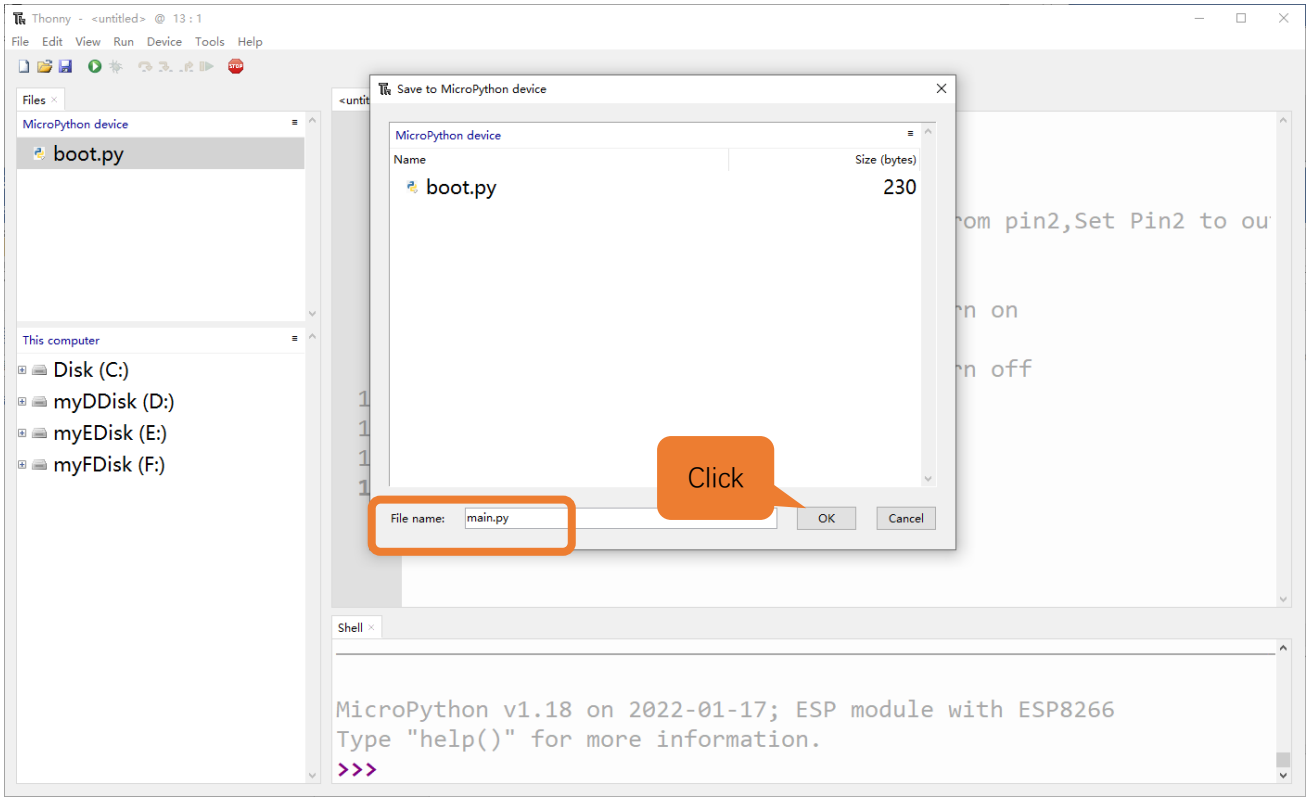
Enter codes in the newly opened file. Here we use codes of "01.1_Blink.py" as an example.



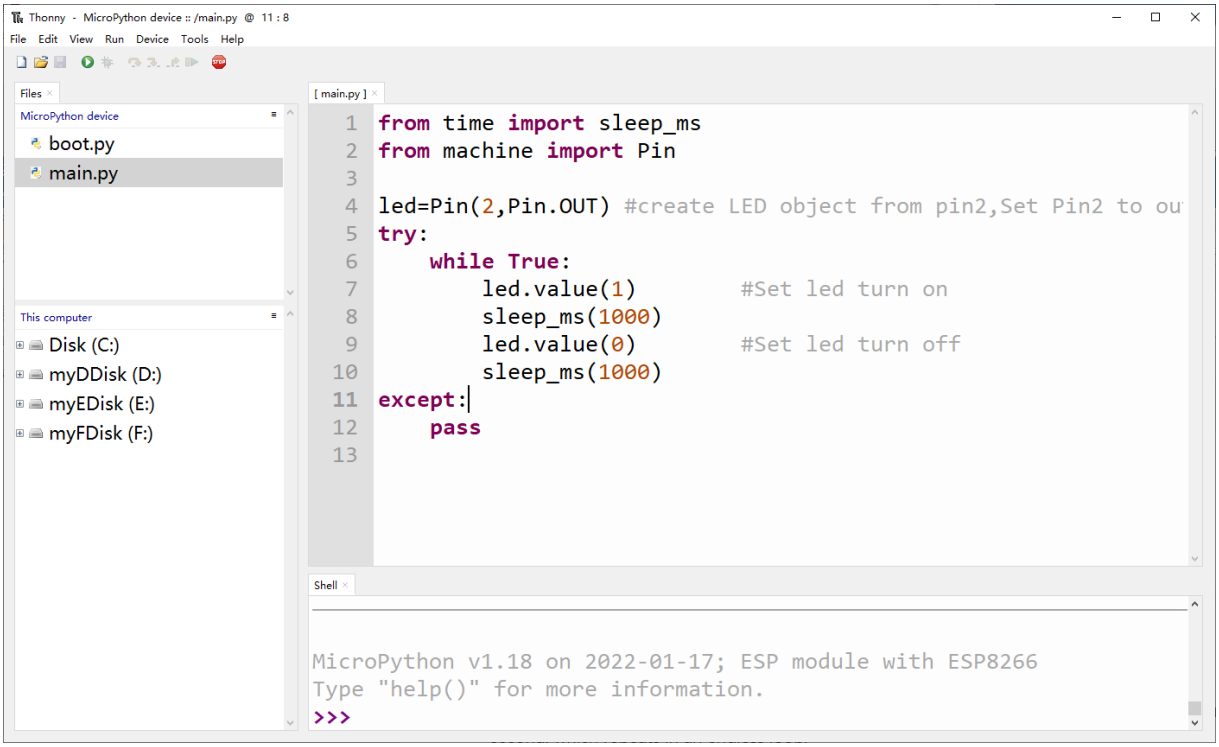
Click "Save" on the menu bar. You can save the codes either to your computer or to ESP8266.

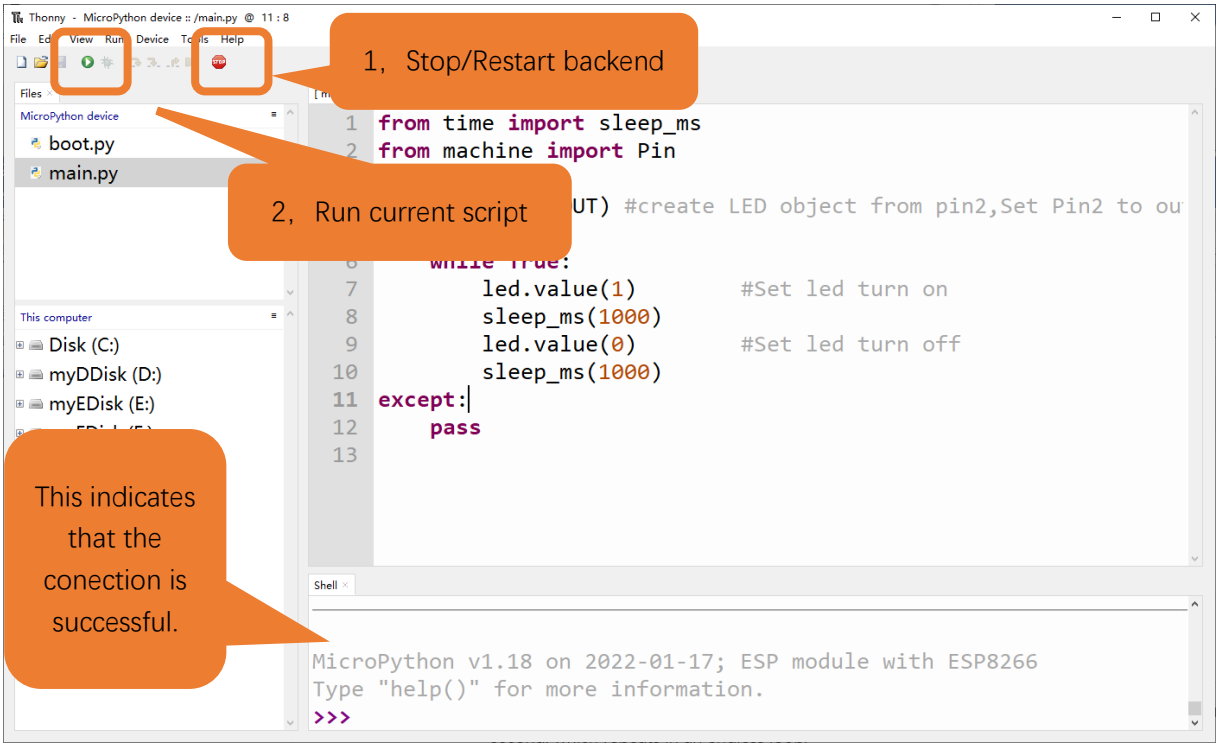


Select "MicroPython device", enter "main.py" in the newly pop-up window and click "OK".

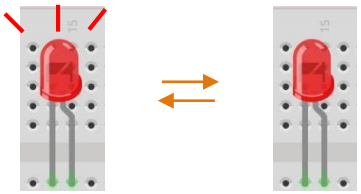


You can see that codes have been uploaded to ESP8266.





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.



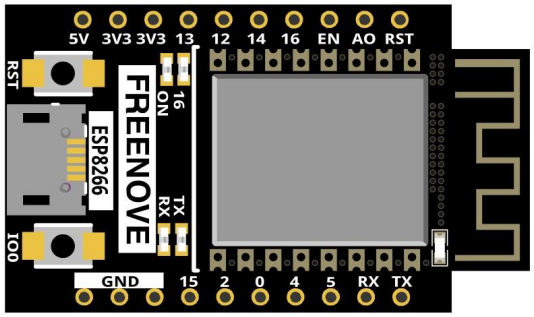
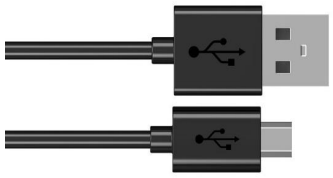
Chapter 1 LED (Important)

This chapter is the Start Point in the journey to build and explore ESP8266 electronic projects. We will start with simple “Blink” project.

Project 1.1 Blink

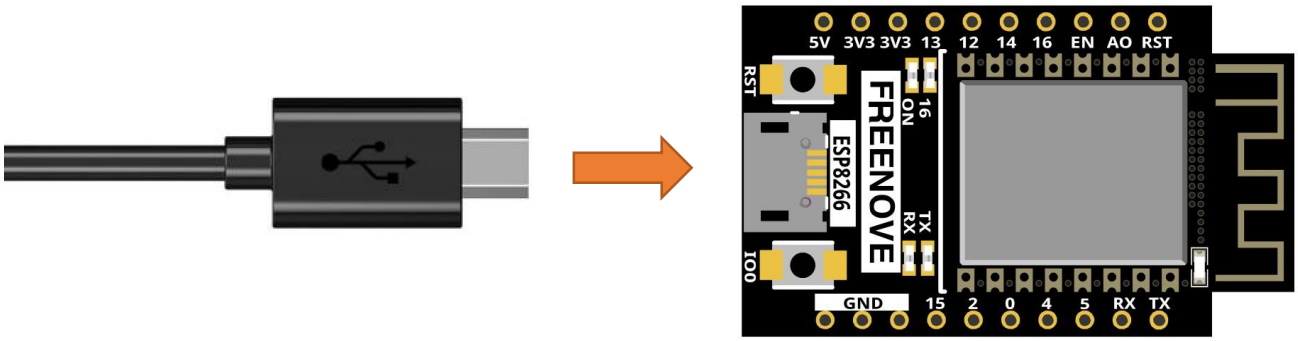
In this project, we will use ESP8266 to control blinking a common LED.
If you have not yet installed Thonny, click here.
If you have not yet downloaded Micropython Firmware, click here.
If you have not yet loaded Micropython Firmware, click here.

Component List

ESP8266 x1 	USB cable 

Power

ESP8266 needs 5v power supply. In this tutorial, we need connect ESP8266 development board to computer via USB cable to power it and program it. We can also use other 5v power source to power it.



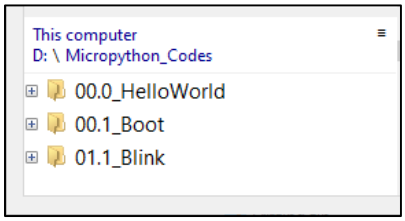
In the following projects, we only use USB cable to power ESP8266 development board by default.

Code

Codes used in this tutorial are saved in "**Freenove_Basic_Starter_Kit_for_ESP8266/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**".

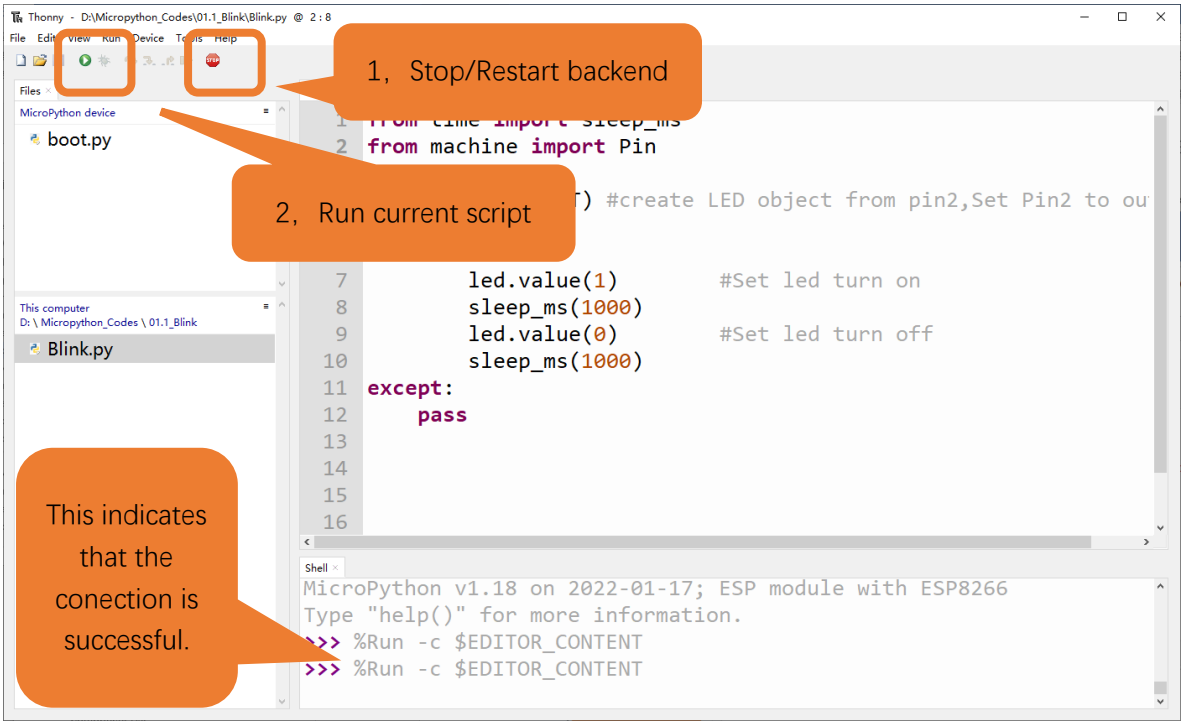
01.1_Blink

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

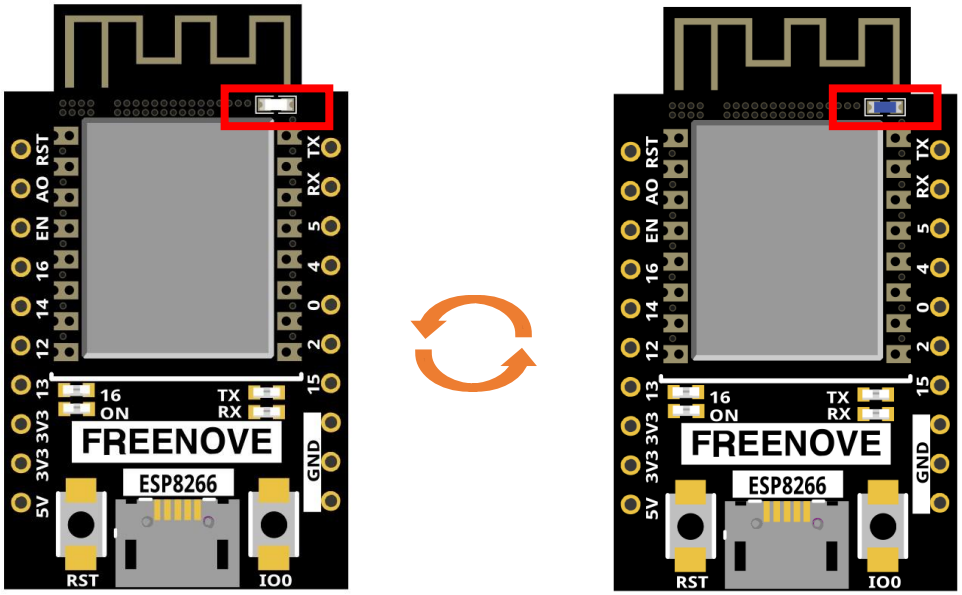


Expand folder "01.1_Blink" and double click "Blink.py" to open it. As shown in the illustration below.

Make sure ESP8266 is properly connected to your computer. Click "Stop/Restart backend" or press the reset button, and then wait to see what interface will show up.

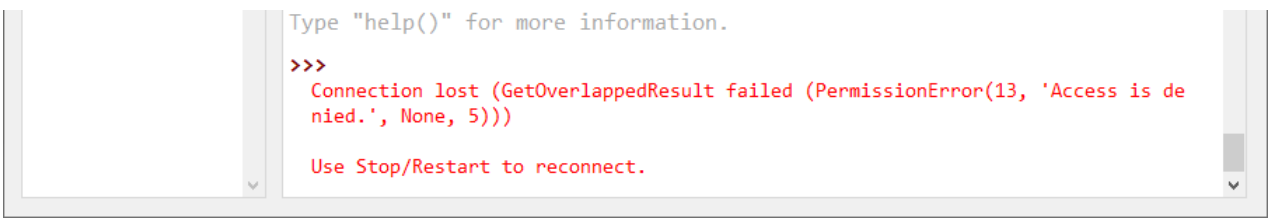


Click "Run current script" shown in the box above, the code starts to be executed and the LED in the circuit starts to blink.



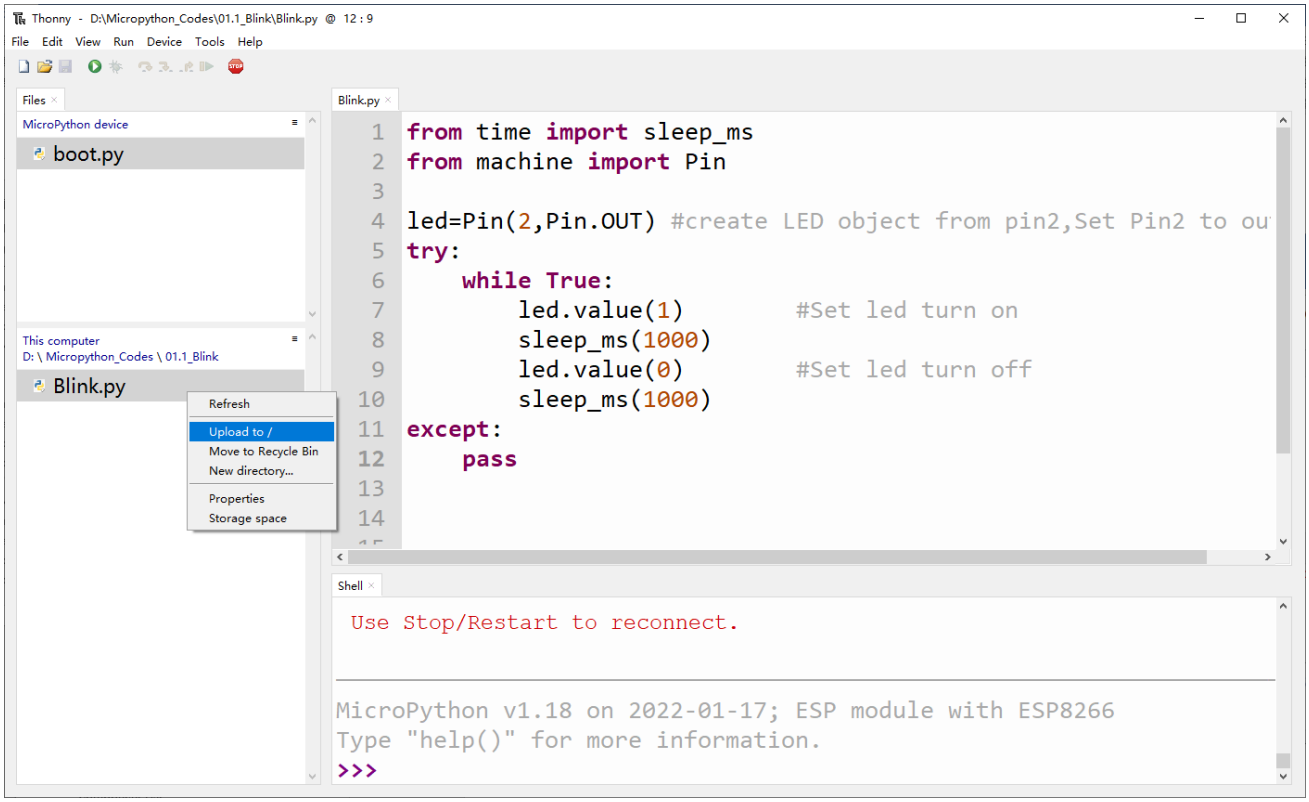
Note:

This is the code running online. If you disconnect USB cable and repower ESP8266 or press its reset key, LED stops blinking and the following messages will be displayed in Thonny.

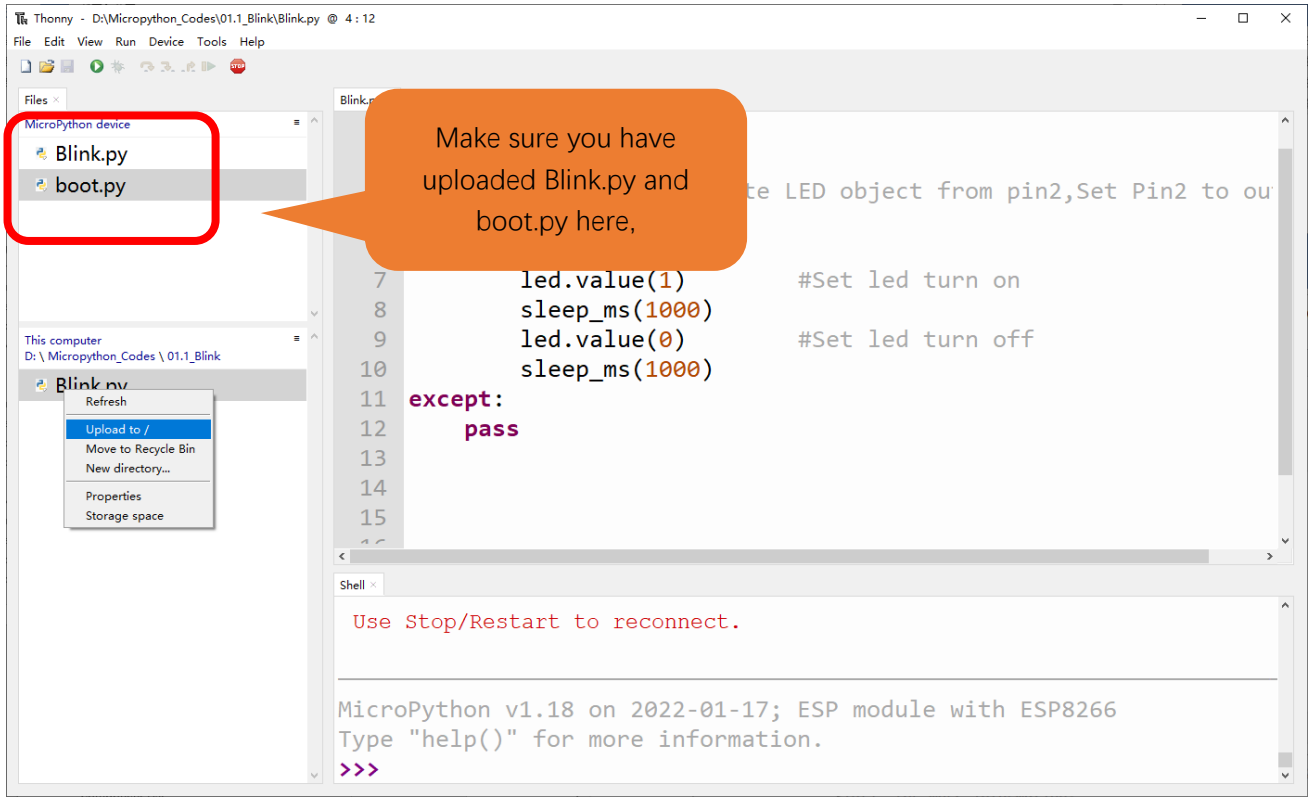


Uploading code to ESP8266

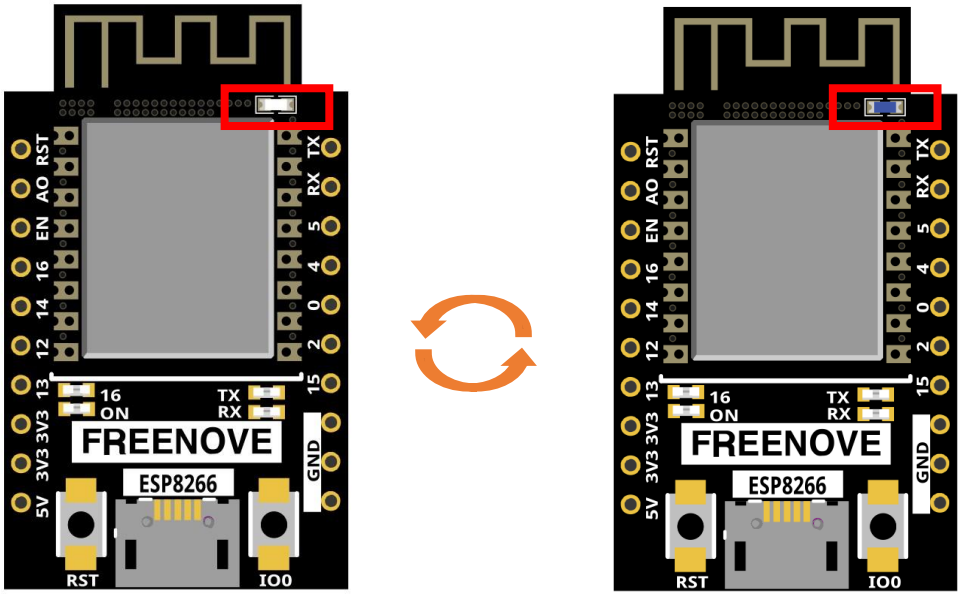
As shown in the following illustration, right-click the file Blink.py and select "Upload to /" to upload code to ESP8266.



Upload boot.py in the same way.

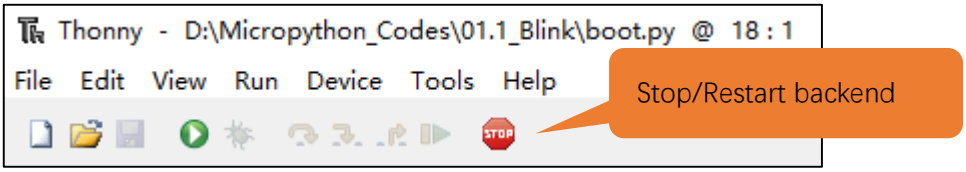


Press the reset key of ESP8266 and you can see LED is ON for one second and then OFF for one second, which repeats in an endless loop.



Note:

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



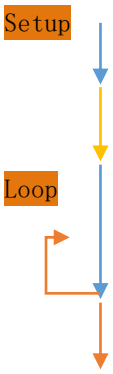
If you have any concerns, please contact us via: support@freenove.com

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 ESP8266, 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 ESP8266 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 ESP8266 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().

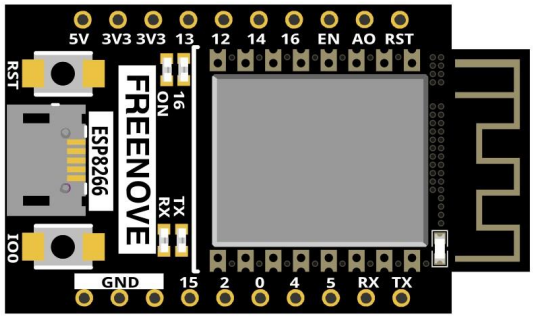
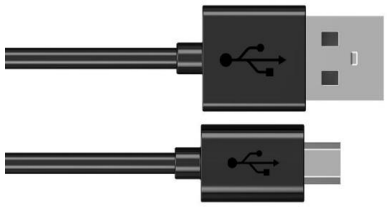
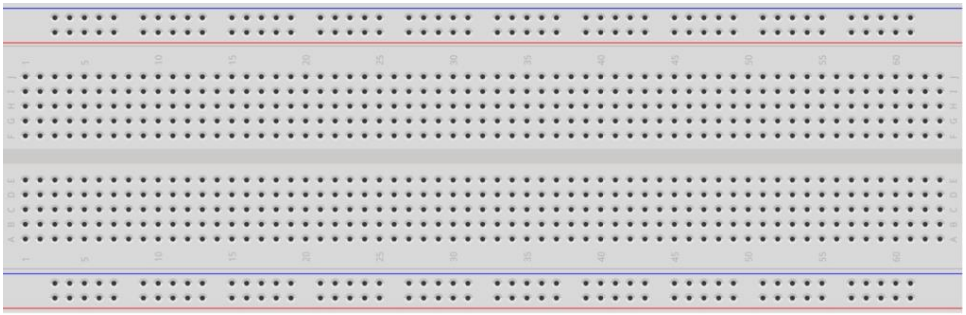
old_t: Starting time

new_t: Ending time

Project 1.2 Blink

In this project, we will use ESP8266 to control blinking a common LED.

Component List

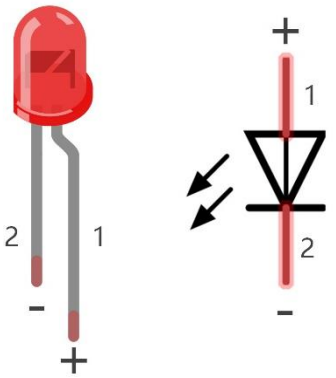
ESP8266 x1 	USB cable 	
Breadboard x1 		
LED x1 	Resistor 220Ω x1 	Jumper wire M/M x3 

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.



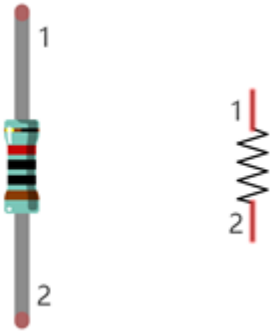
LED	Voltage	Maximum current	Recommended current
Red	1.9-2.2V	20mA	10mA
Green	2.9-3.4V	10mA	5mA
Blue	2.9-3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

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.

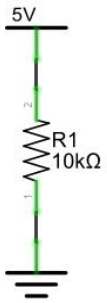
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



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.

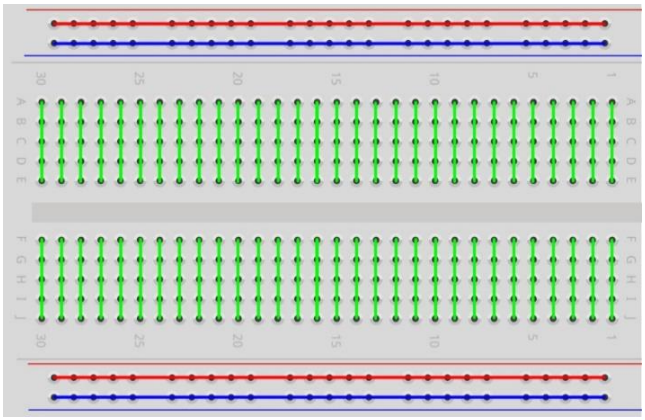


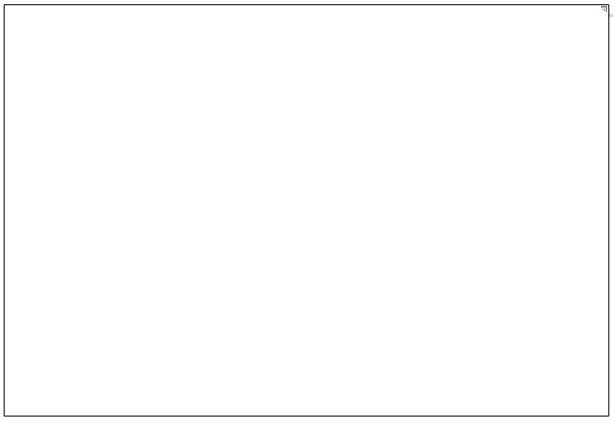
WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

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

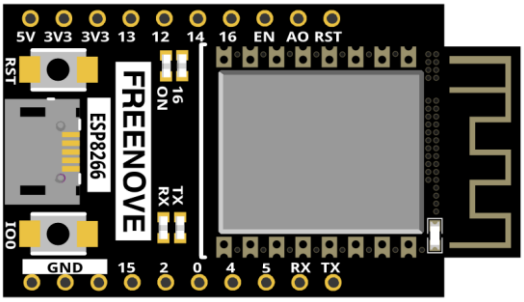
Here we have a small breadboard as an example of how the rows of holes (sockets) are electrically attached. The left picture shows the way to connect pins. The right picture shows the practical internal structure.





Power

ESP8266 needs 5v power supply. In this tutorial, we need connect ESP8266 to computer via USB cable to power it and program it. We can also use other 5v power source to power it.



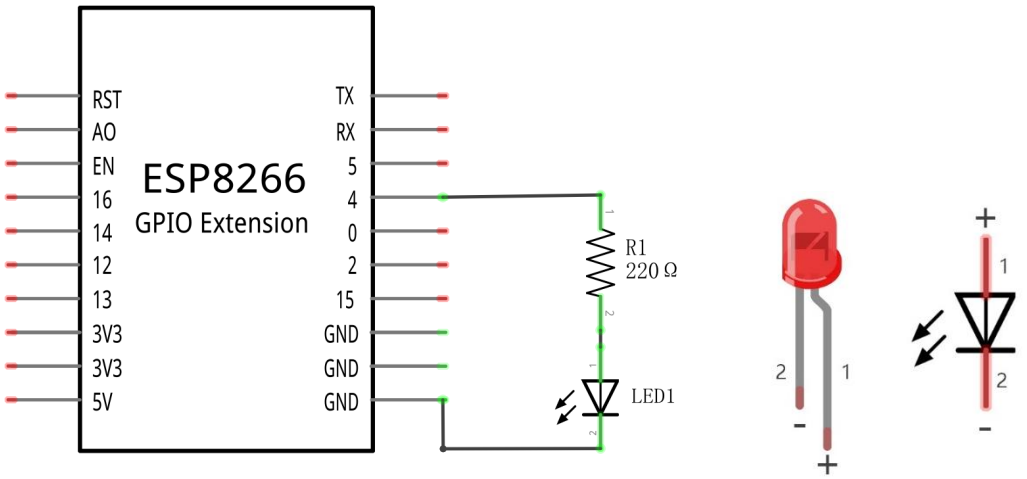
Later, we only use USB cable to power ESP8266 in default.

Circuit

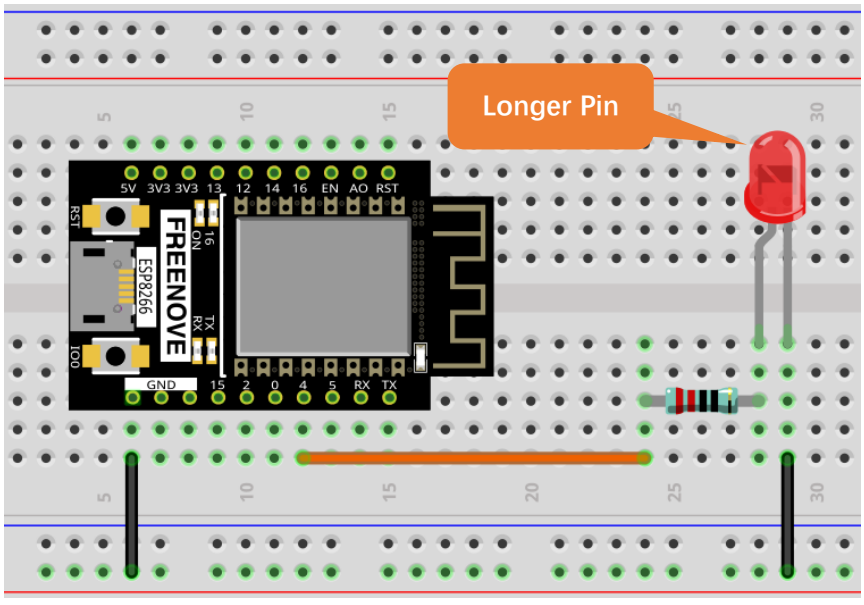
First, disconnect all power from the ES8266. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the PC to ESP8266.

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**



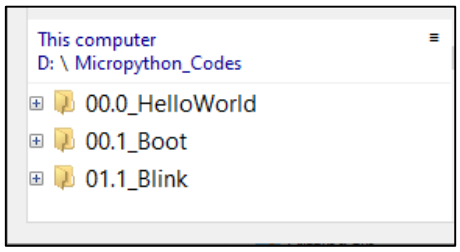
Code

Codes used in this tutorial are saved in "**Freenove_Basic_Starter_Kit_for_ESP8266/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**".

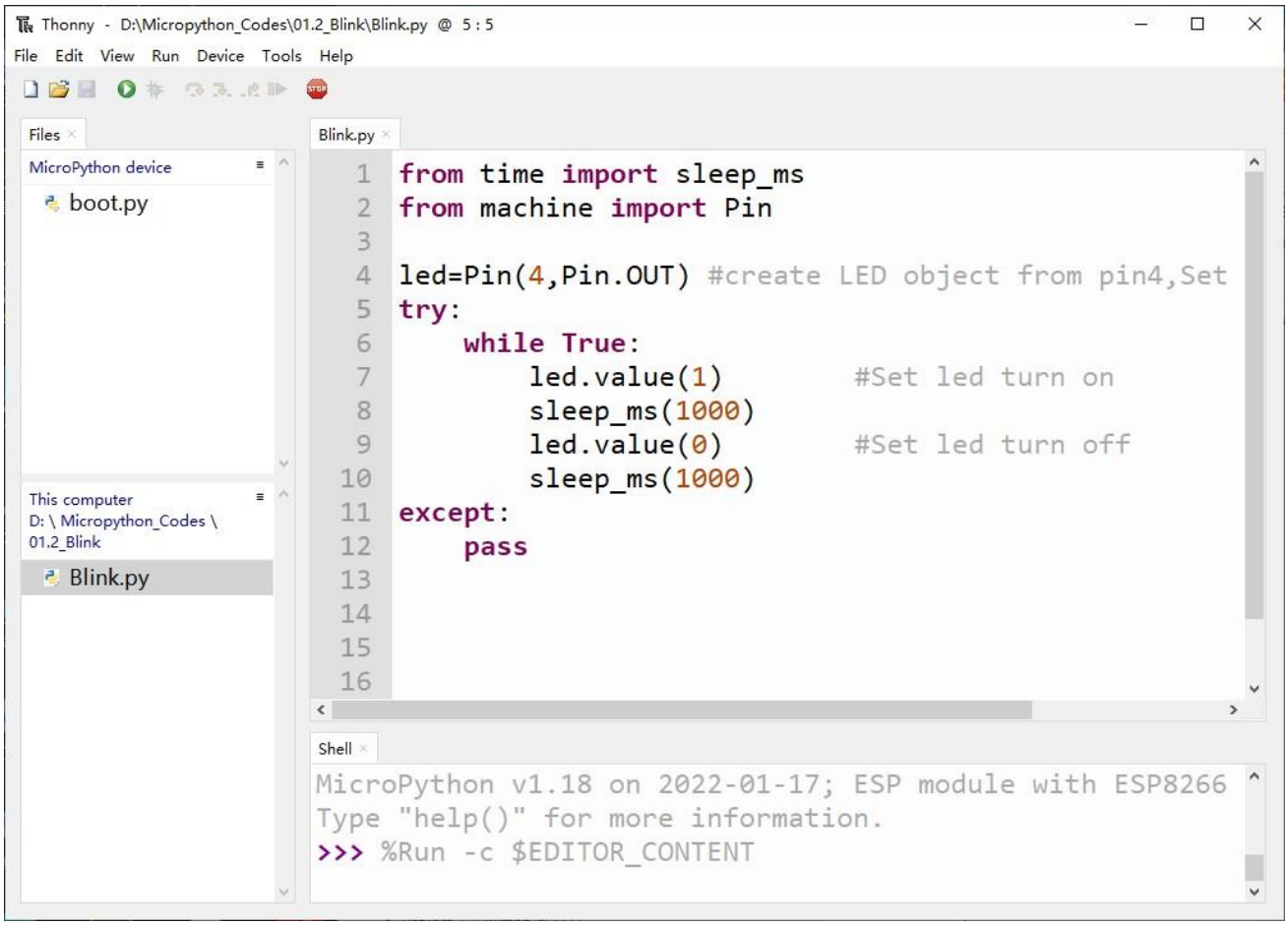
Any concerns? ✉ support@freenove.com

01.2_Blink

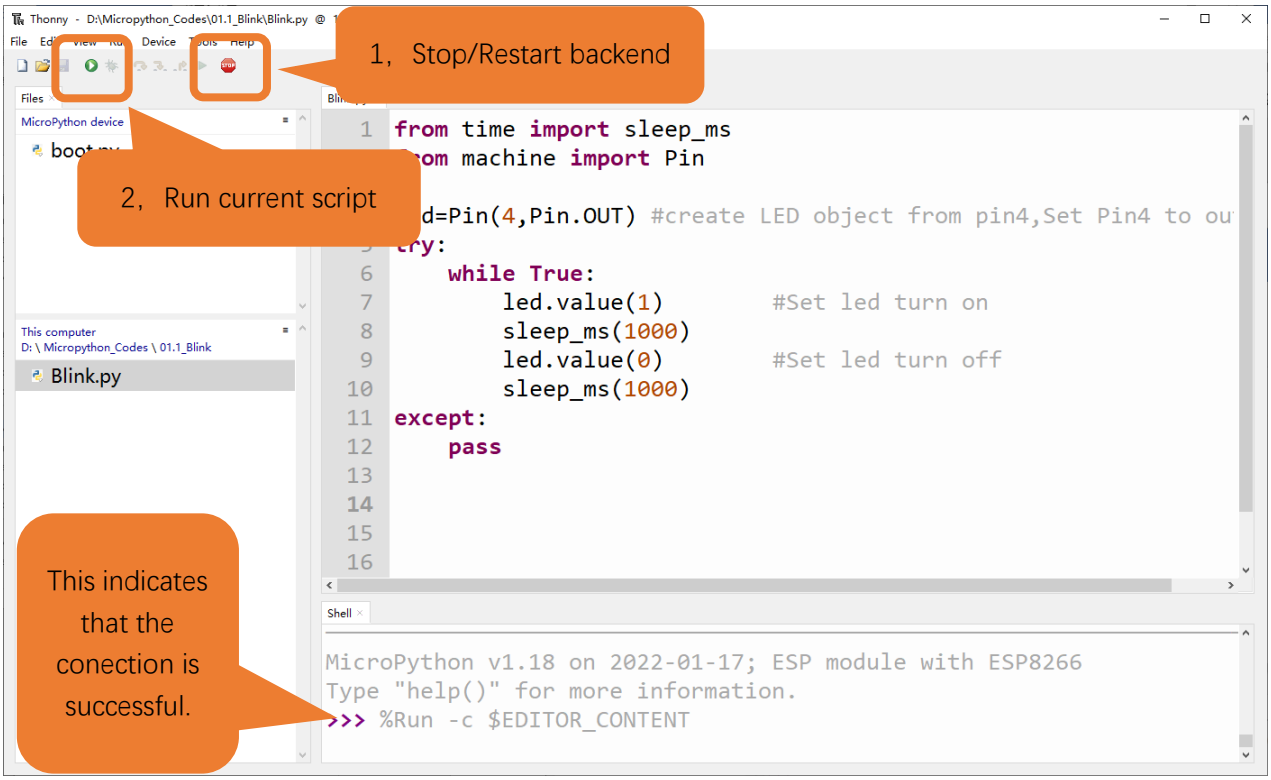
Open "Thonny", click "This computer"→"D:"→"Micropython_Codes".



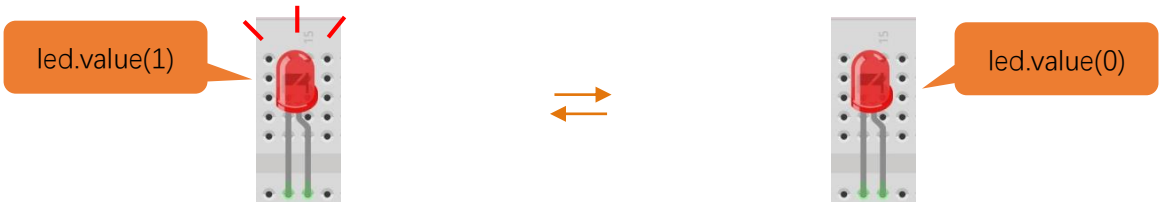
Expand folder "01.2_Blink" and double click "Blink.py" to open it. As shown in the illustration below.



Make sure ESP8266 has been connected with the computer with ESP8266 correctly. Click "Stop/Restart backend" or press the reset button, and then wait to see what interface will show up.

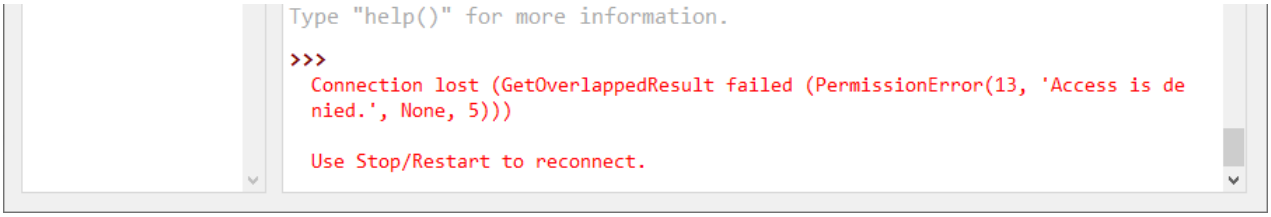


Click "Run current script" shown in the box above, the code starts to be executed and the LED in the circuit starts to blink.



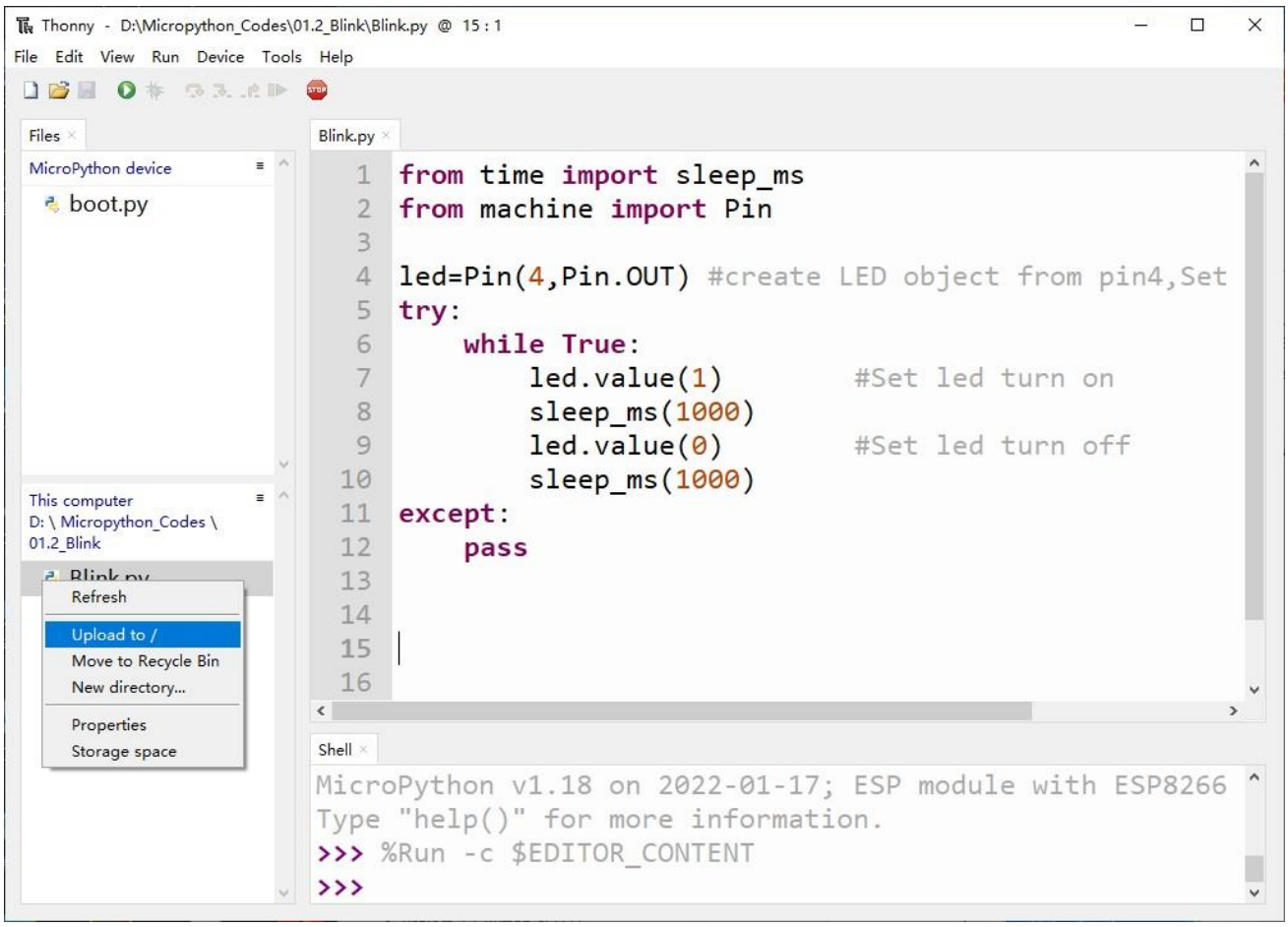
Note:

This is the code running online. If you disconnect USB cable and repower ESP8266 or press its reset key, LED stops blinking and the following messages will be displayed in Thonny.

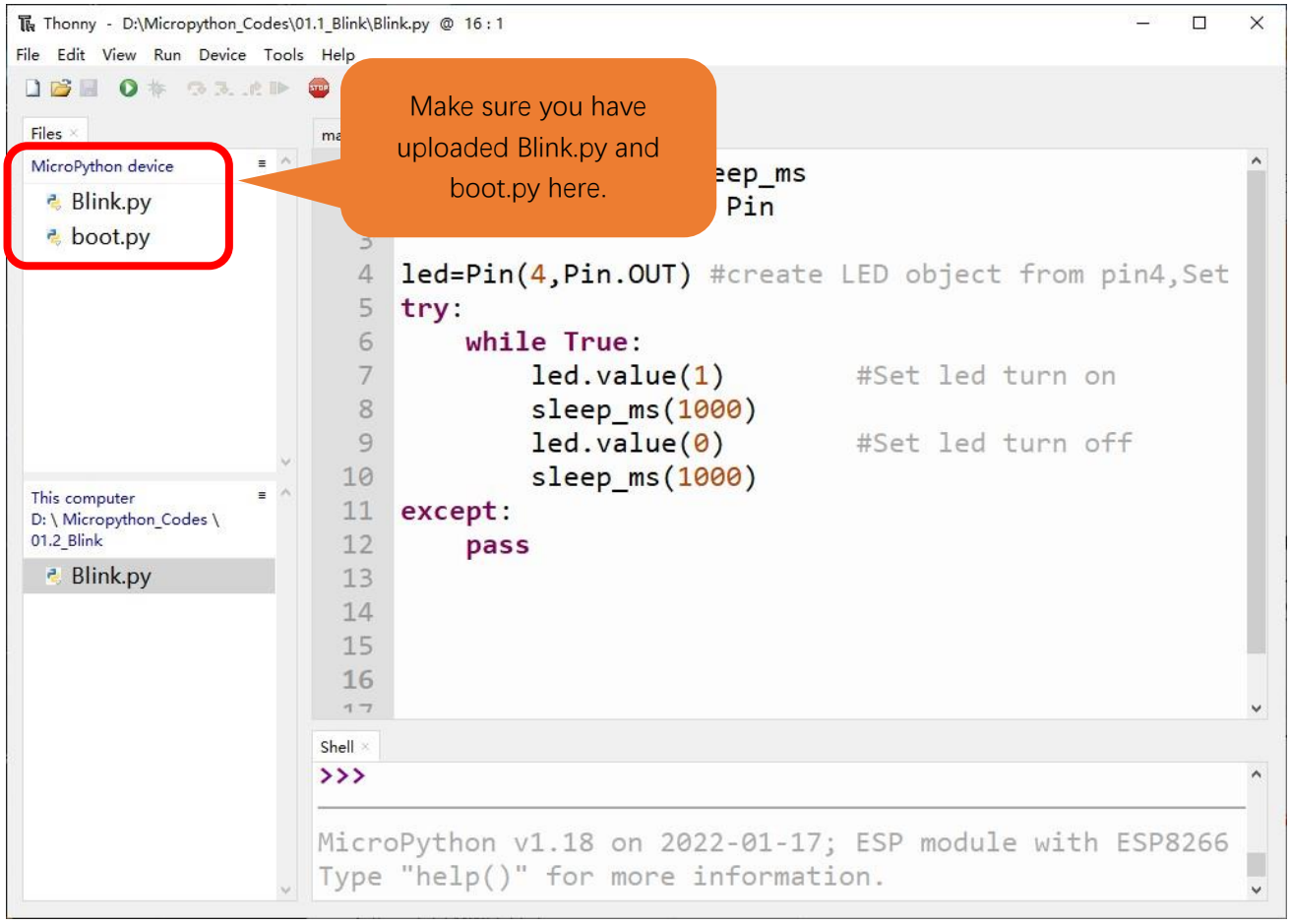


Uploading code to ESP8266

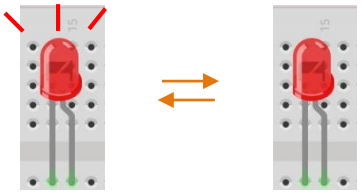
As shown in the following illustration, right-click the file Blink.py and select "Upload to /" to upload code to ESP8266.


Thonny - D:\Micropython_Codes\01.2_Blink\Blink.py @ 15 : 1
File Edit View Run Device Tools Help
Files
MicroPython device
boot.py
This computer
D: \ Micropython_Codes \
01.2_Blink
Refresh
Upload to /
Move to Recycle Bin
New directory...
Properties
Storage space
Blink.py
1 from time import sleep_ms
2 from machine import Pin
3
4 led=Pin(4,Pin.OUT) #create LED object from pin4,Set
5 try:
6 while True:
7 led.value(1) #Set led turn on
8 sleep_ms(1000)
9 led.value(0) #Set led turn off
10 sleep_ms(1000)
11 except:
12 pass
13
14
15
16
Shell
MicroPython v1.18 on 2022-01-17; ESP module with ESP8266
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
>>>

Upload boot.py in the same way.

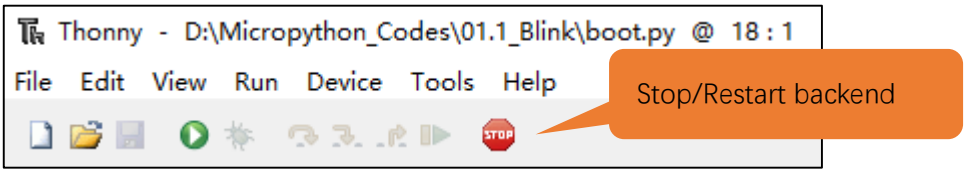


Press the reset key of ESP8266 and you can see LED is ON for one second and then OFF for one second, which repeats in an endless loop.



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

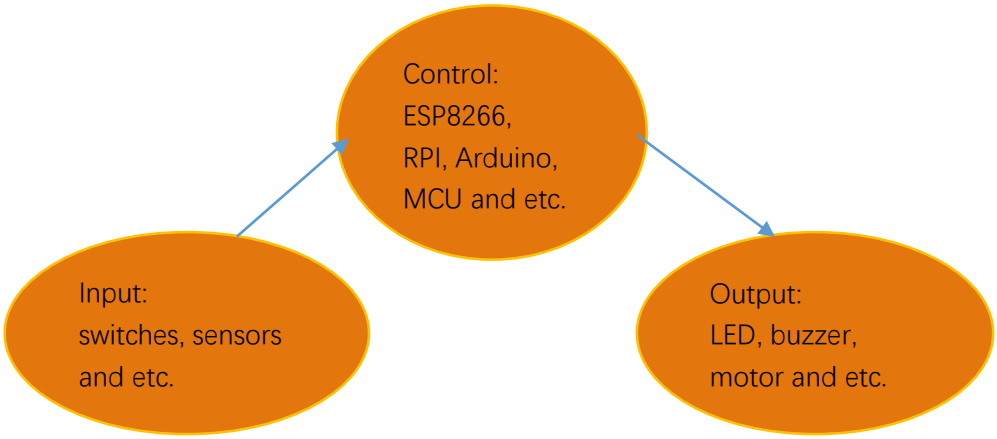
The following is the program code:

```
from time import sleep_ms
from machine import Pin

led=Pin(4,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
```

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 ESP8266 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.



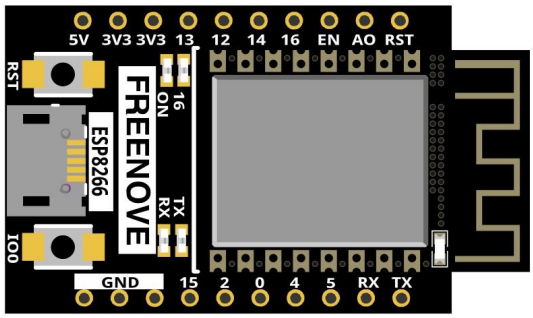
Next, we will build a simple control system to control an LED through a push button switch.

Project 2.1 Button & LED

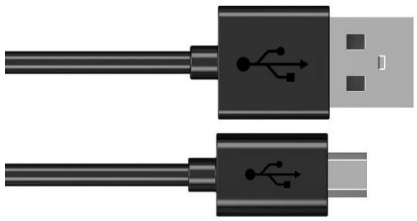
In the project, we will control the LED state through a Push Button Switch. When the button is pressed, our LED will turn ON, and when it is released, the LED will turn OFF. This describes a Momentary Switch.

Component List

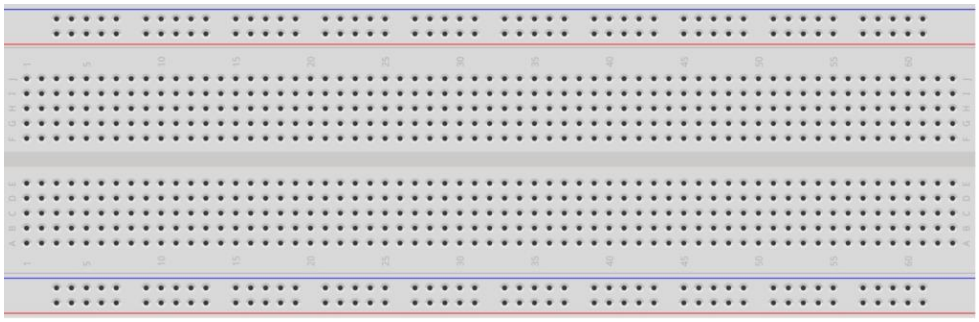
ESP8266 x1



USB cable



Breadboard x1



Jumper wire M/M x6

LED x1

Resistor 220Ω x1

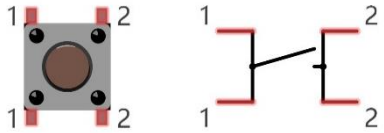
Resistor 10kΩ x2

Push button x1

Component knowledge

Push button

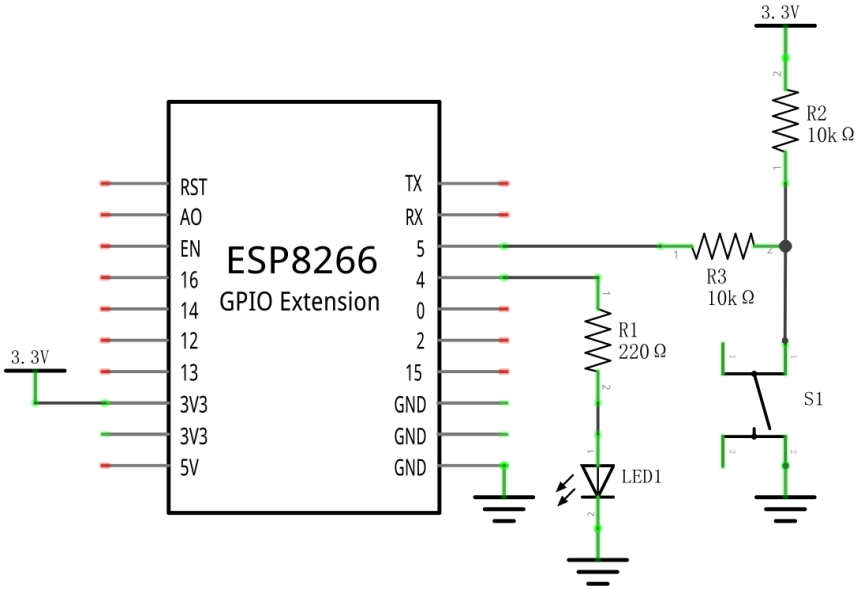
This type of Push Button Switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



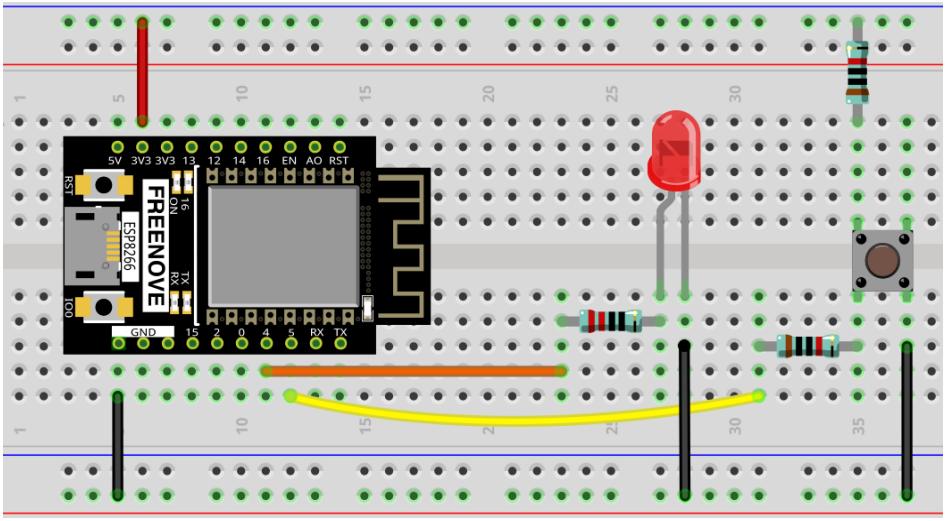
When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

Circuit

Schematic diagram

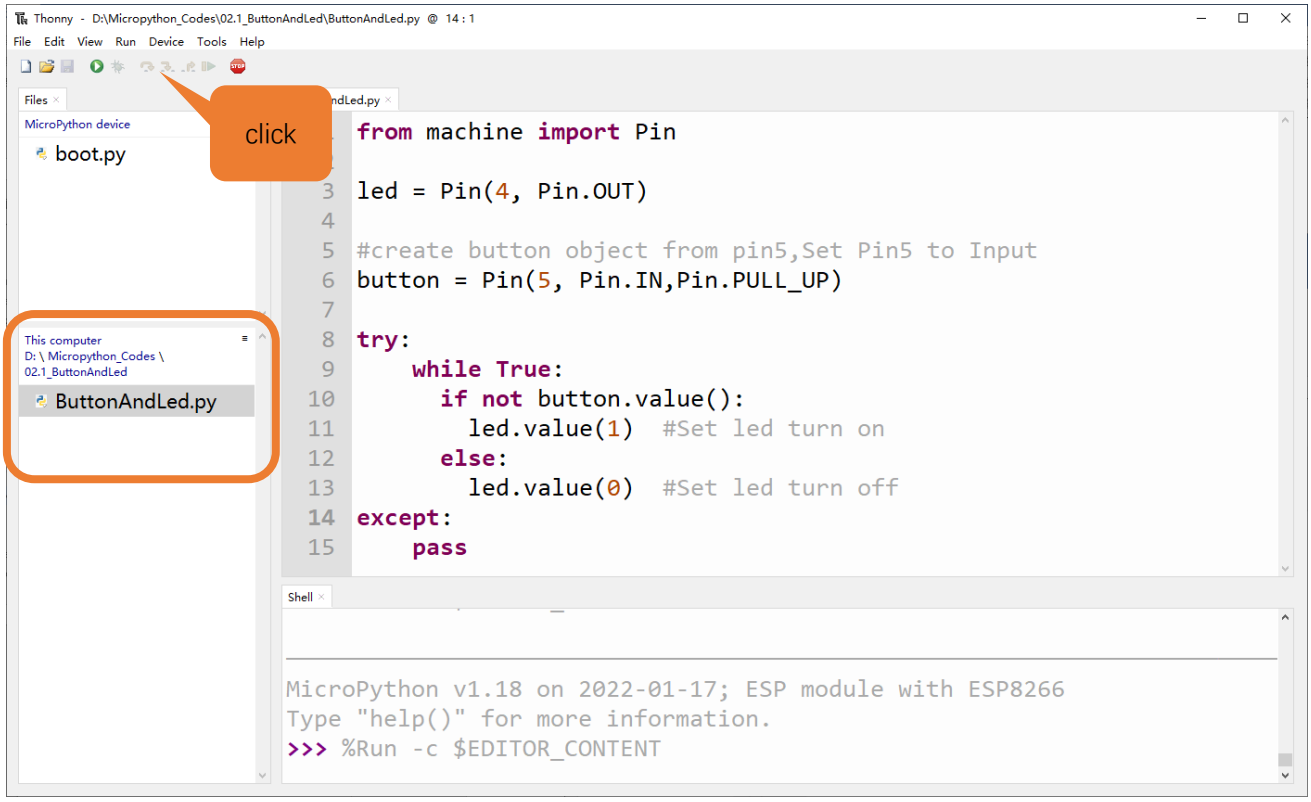


Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

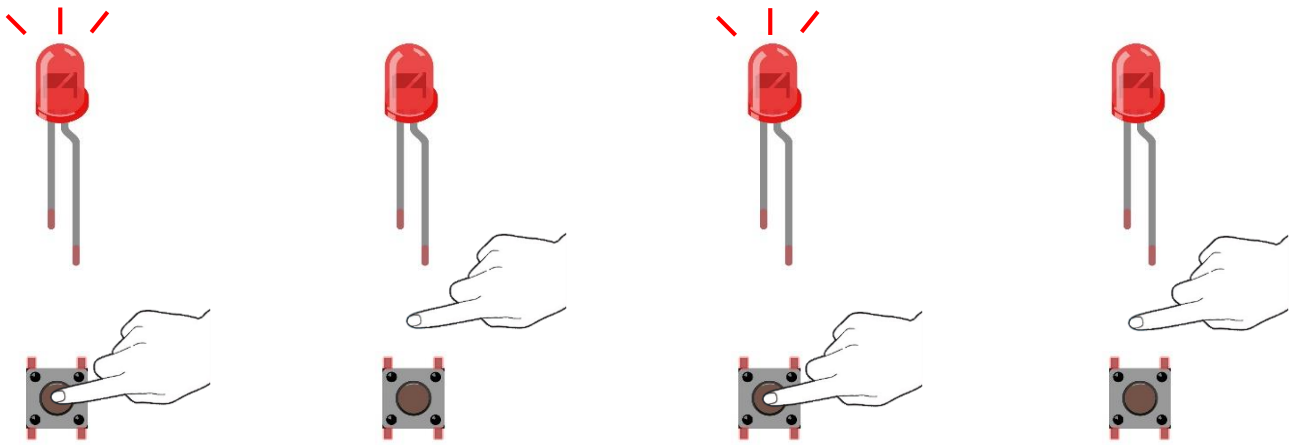


Code

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_ESP8266/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

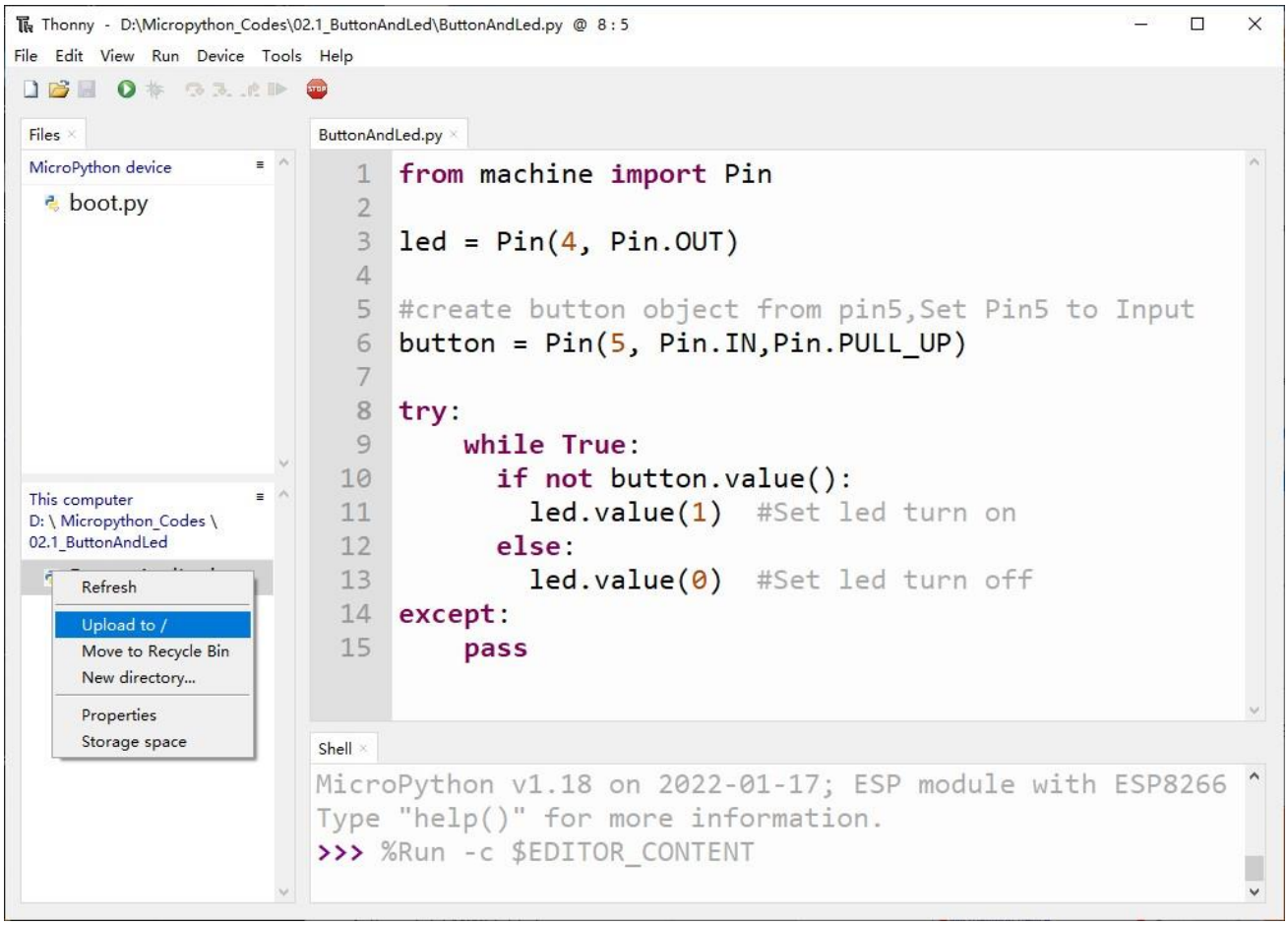


Click "Run current script" shown in the box of the above illustration, press the push button switch, LED turns ON; release the switch, LED turns OFF.

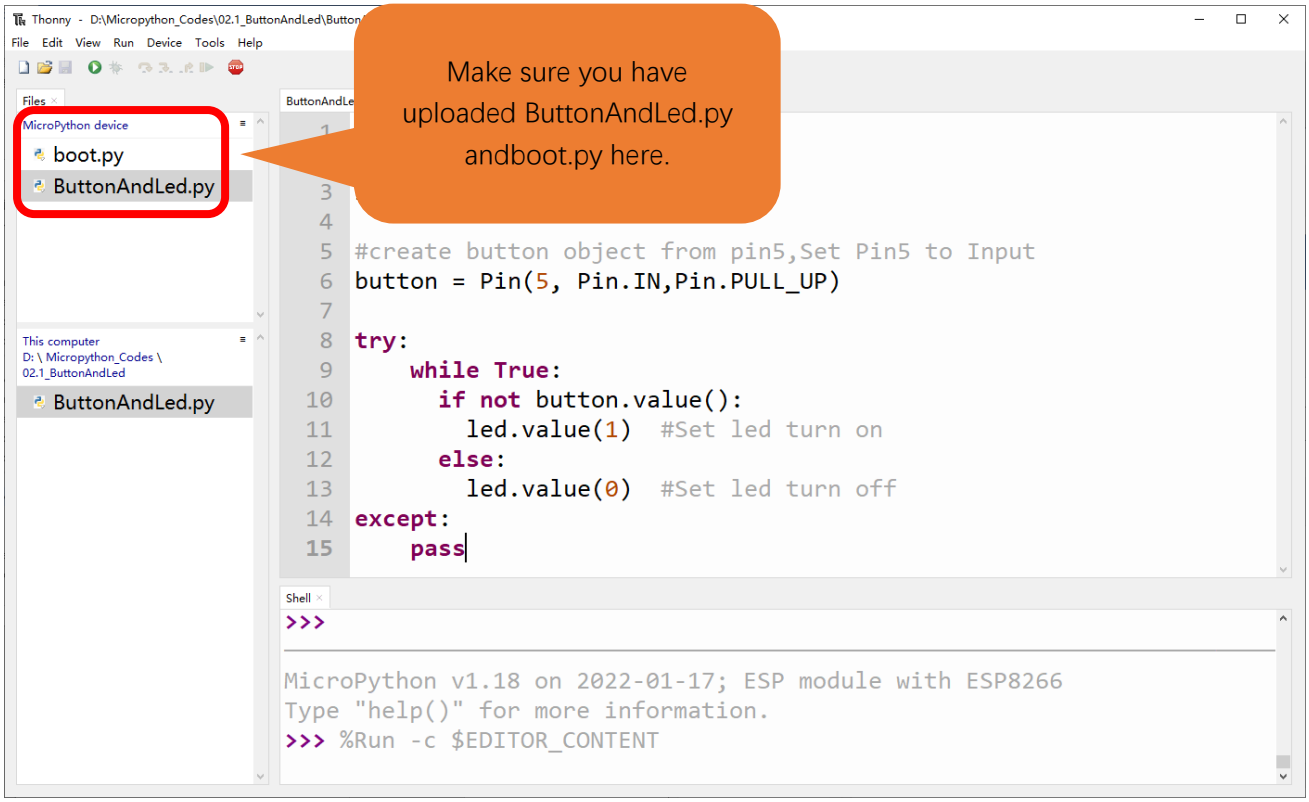


Upload Code to ESP8266

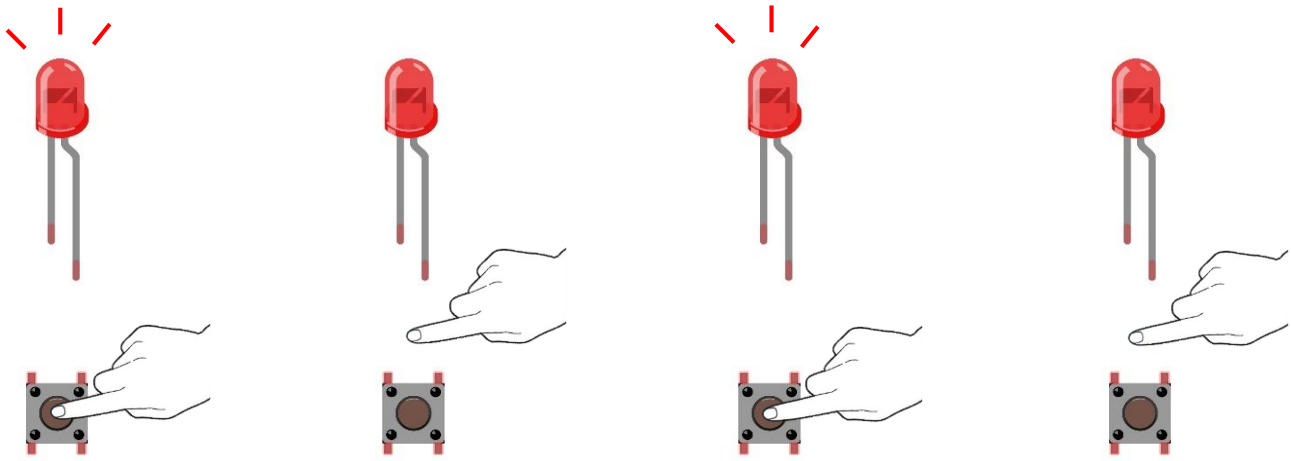
As shown in the following illustration, right-click file 02.1_ButtonAndLed and select "Upload to /" to upload code to ESP8266.



Upload boot.py in the same way.


Thonny - D:\Micropython_Codes\02.1_ButtonAndLed\Button
File Edit View Run Device Tools Help
Files
MicroPython device
boot.py
ButtonAndLed.py
This computer
D: \ Micropython_Codes \
02.1_ButtonAndLed
ButtonAndLed.py
Make sure you have uploaded ButtonAndLed.py andboot.py here.
#create button object from pin5,Set Pin5 to Input
button = Pin(5, Pin.IN,Pin.PULL_UP)
try:
while True:
if not button.value():
led.value(1) #Set led turn on
else:
led.value(0) #Set led turn off
except:
pass
Shell
>>>
MicroPython v1.18 on 2022-01-17; ESP module with ESP8266
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

Press ESP8266'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:

```
from machine import Pin

led = Pin(4, Pin.OUT)

#create button object from pin5,Set Pin5 to Input
button = Pin(5, Pin.IN,Pin.PULL_UP)

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

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

```
from machine import Pin
```

In the circuit connection, LED and Button are connected with GPIO4 and GPIO5 respectively, so define led and button as 4 and 5 respectively.

```
led = Pin(4, Pin.OUT)

#create button object from pin5,Set Pin5 to Input
button = Pin(5, 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.

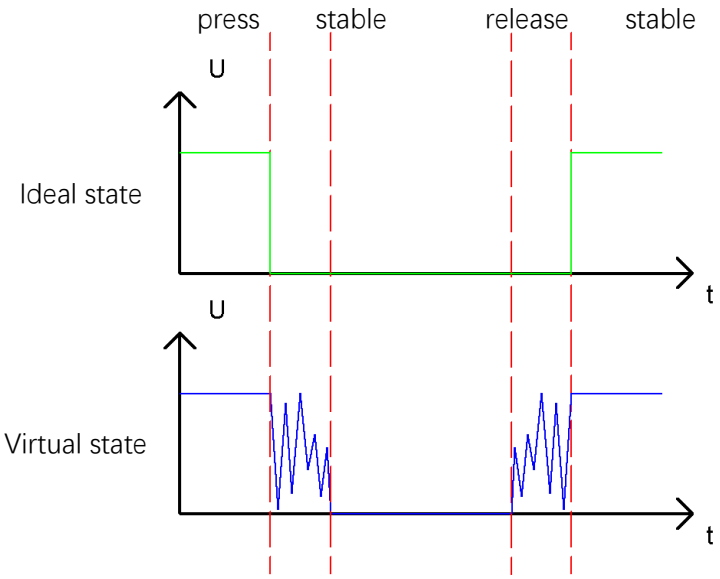
```
10  if not button.value():
11      …
12  else:
13      …
```

Project 2.2 MINI table lamp

We will also use a Push Button Switch, LED and ESP8266 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”.



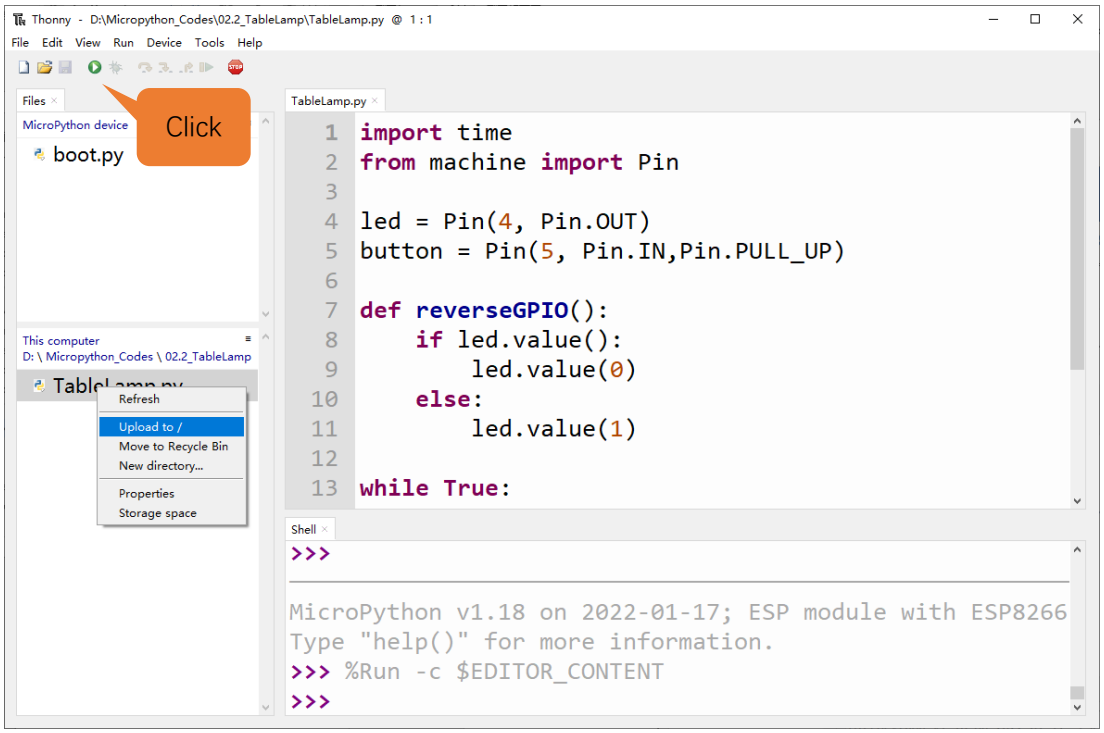
Therefore, if we can directly detect the state of the Push Button Switch, there are multiple pressing and releasing actions in one pressing cycle. This buffeting will mislead the high-speed operation of the microcontroller to cause many false decisions. Therefore, we need to eliminate the impact of buffeting. 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

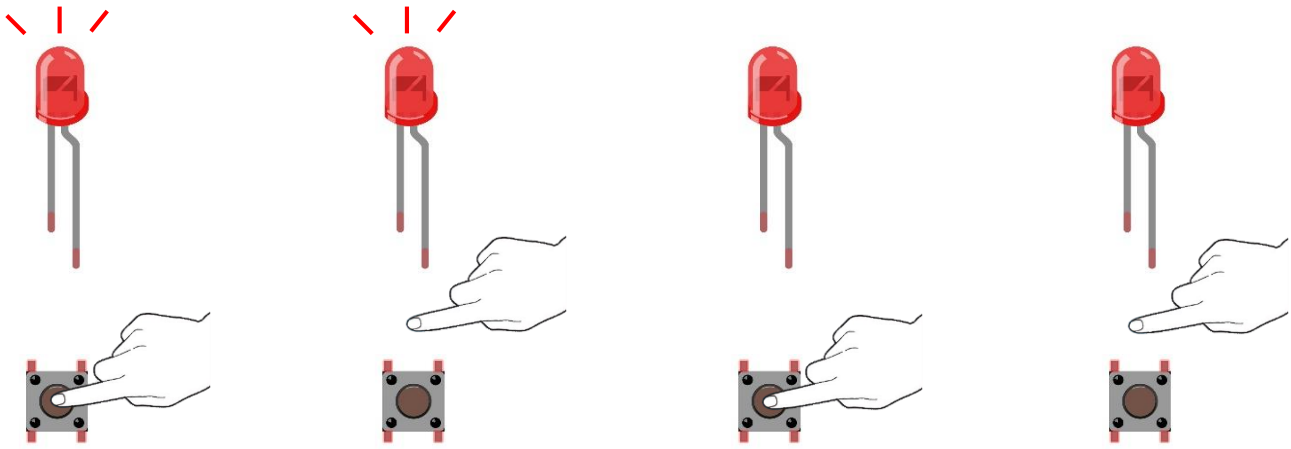
02.2_Tablelamp

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP8266/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".

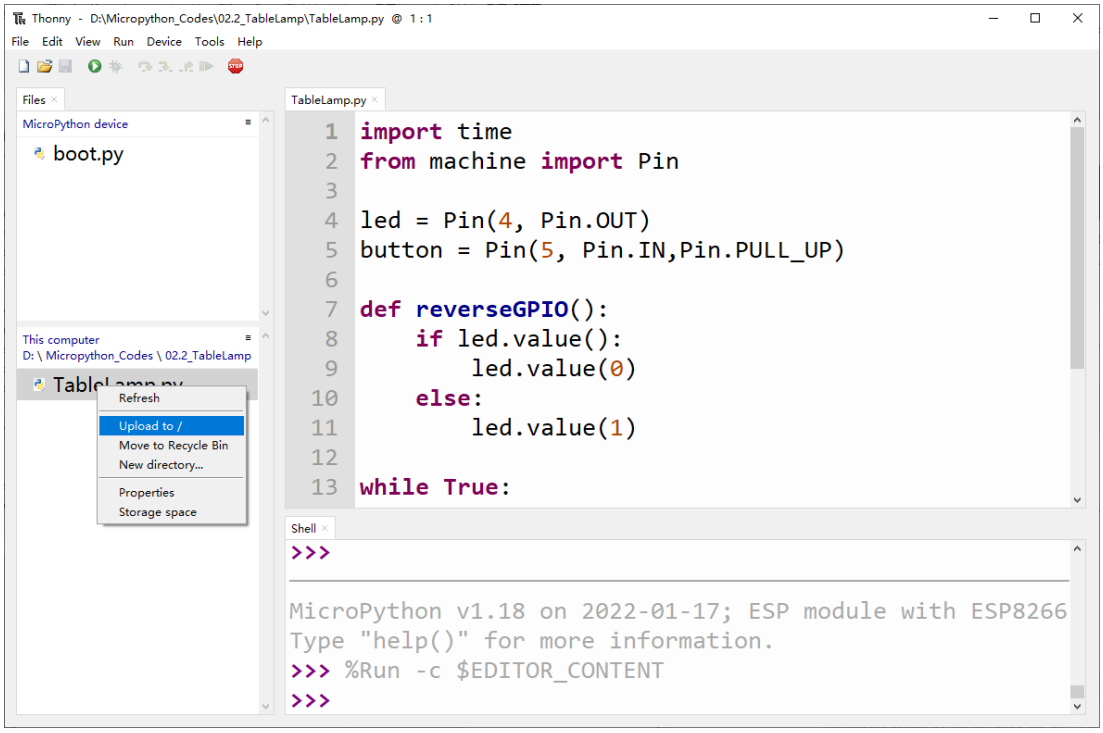


Click "Run current script" shown in the box of the above illustration, press the push button switch, LED turns ON; press it again, LED turns OFF.

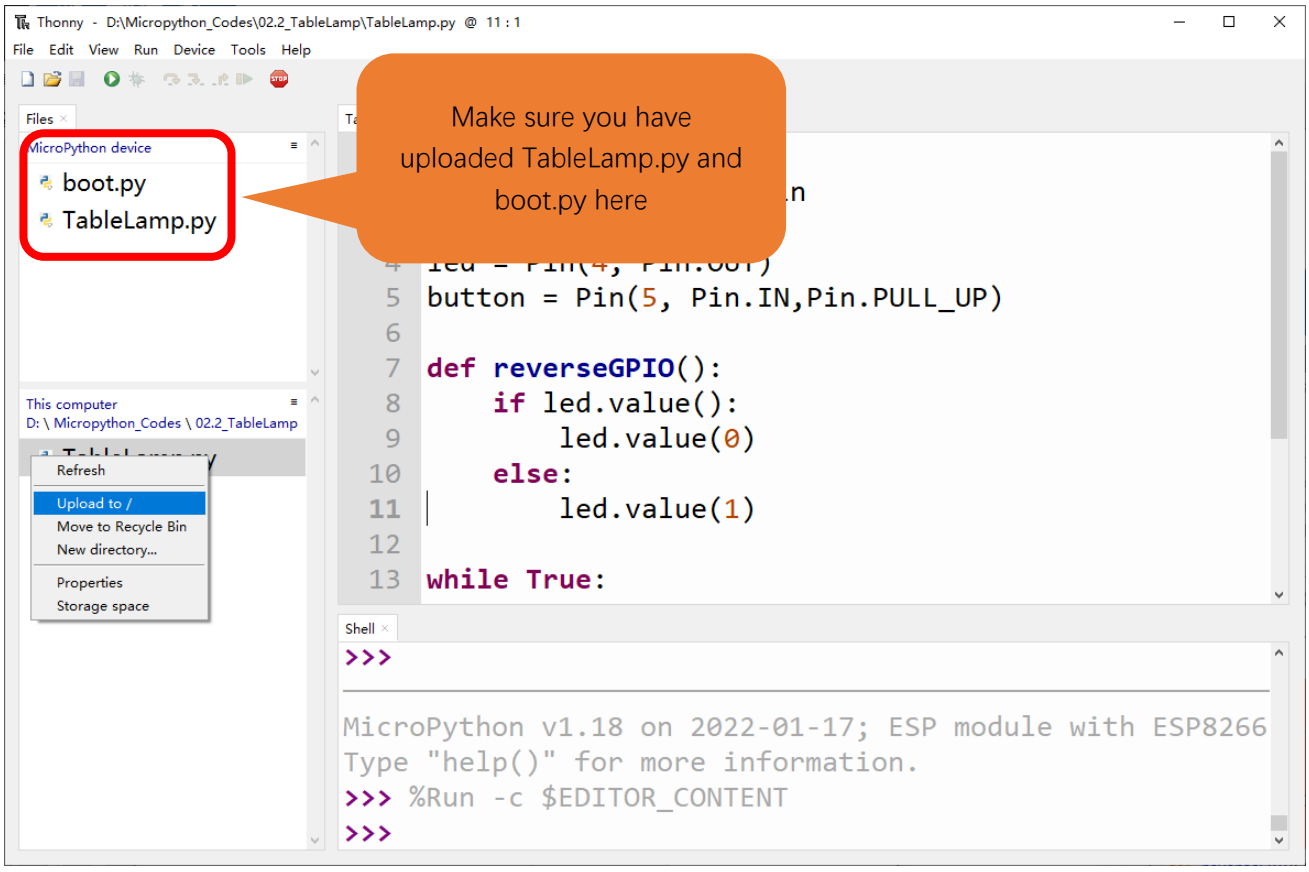


Upload code to ESP8266

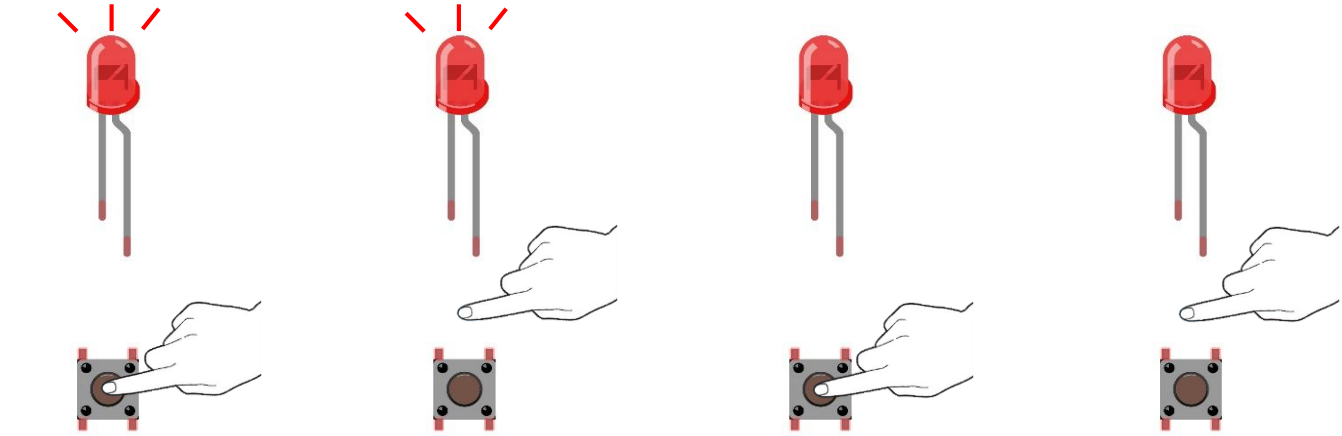
As shown in the following illustration, right-click file 02.2_TableLamp and select "Upload to /" to upload code to ESP8266.



Upload boot.py in the same way.



Press ESP8266'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(4, Pin.OUT)
button = Pin(5, 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)
```

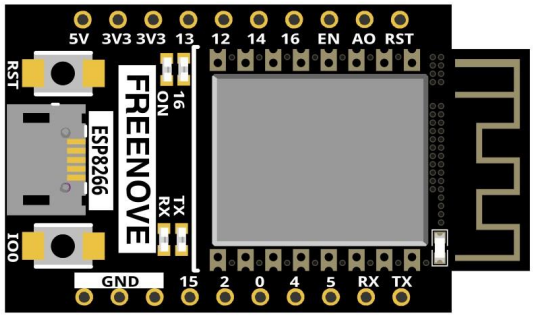
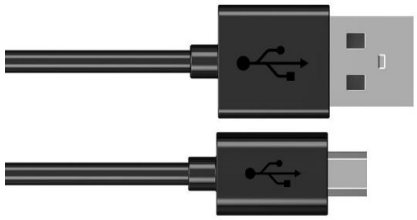
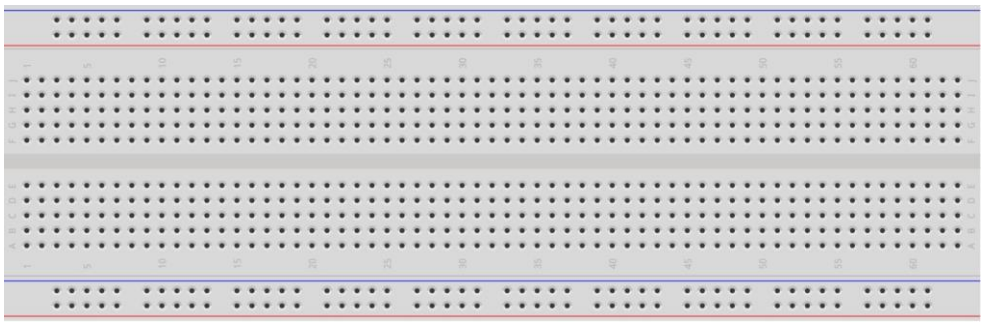
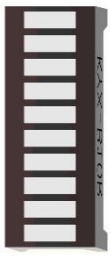
Chapter 3 LED Bar

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

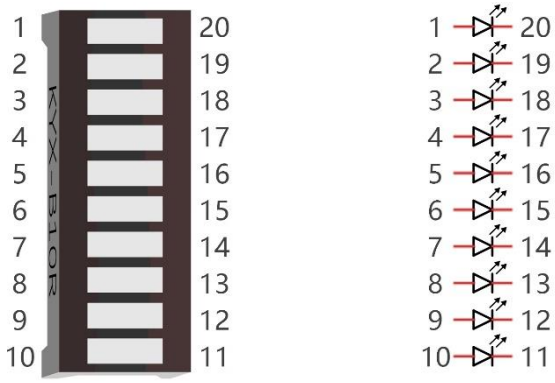
ESP8266 x1 	USB cable 	
Breadboard x1 		
Jumper wire M/M x10 	LED bar graph x1 	Resistor 220Ω x9 

Component knowledge

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

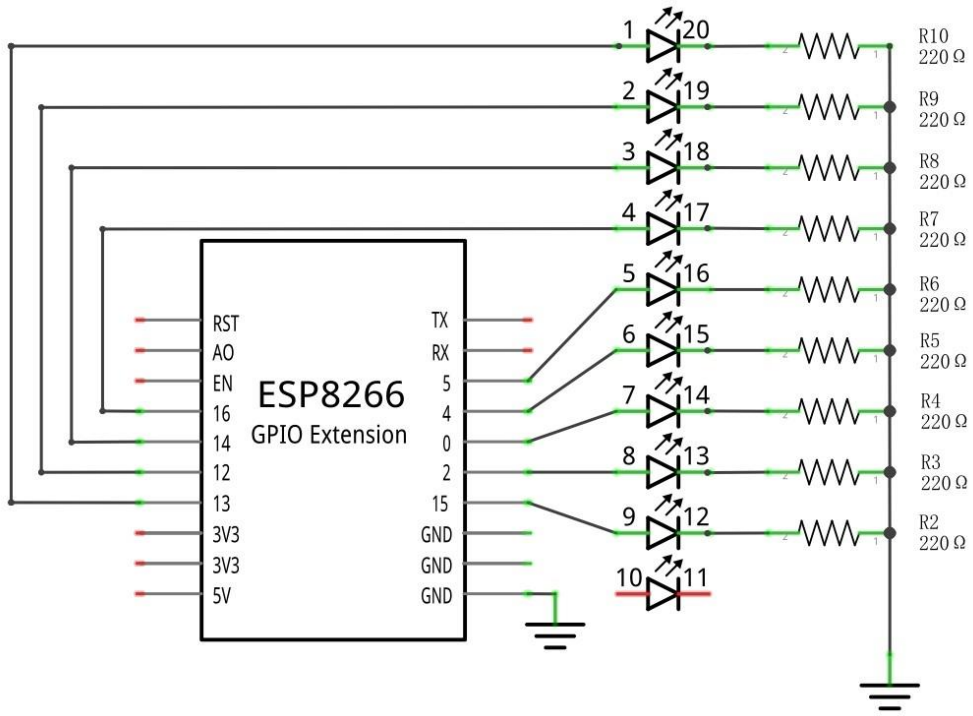
LED bar

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.

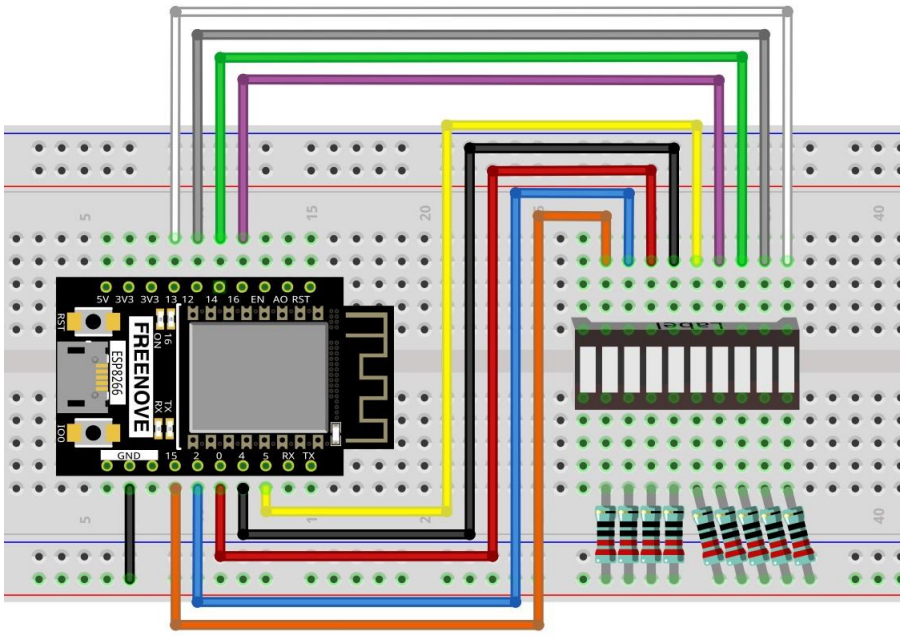


Circuit

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.

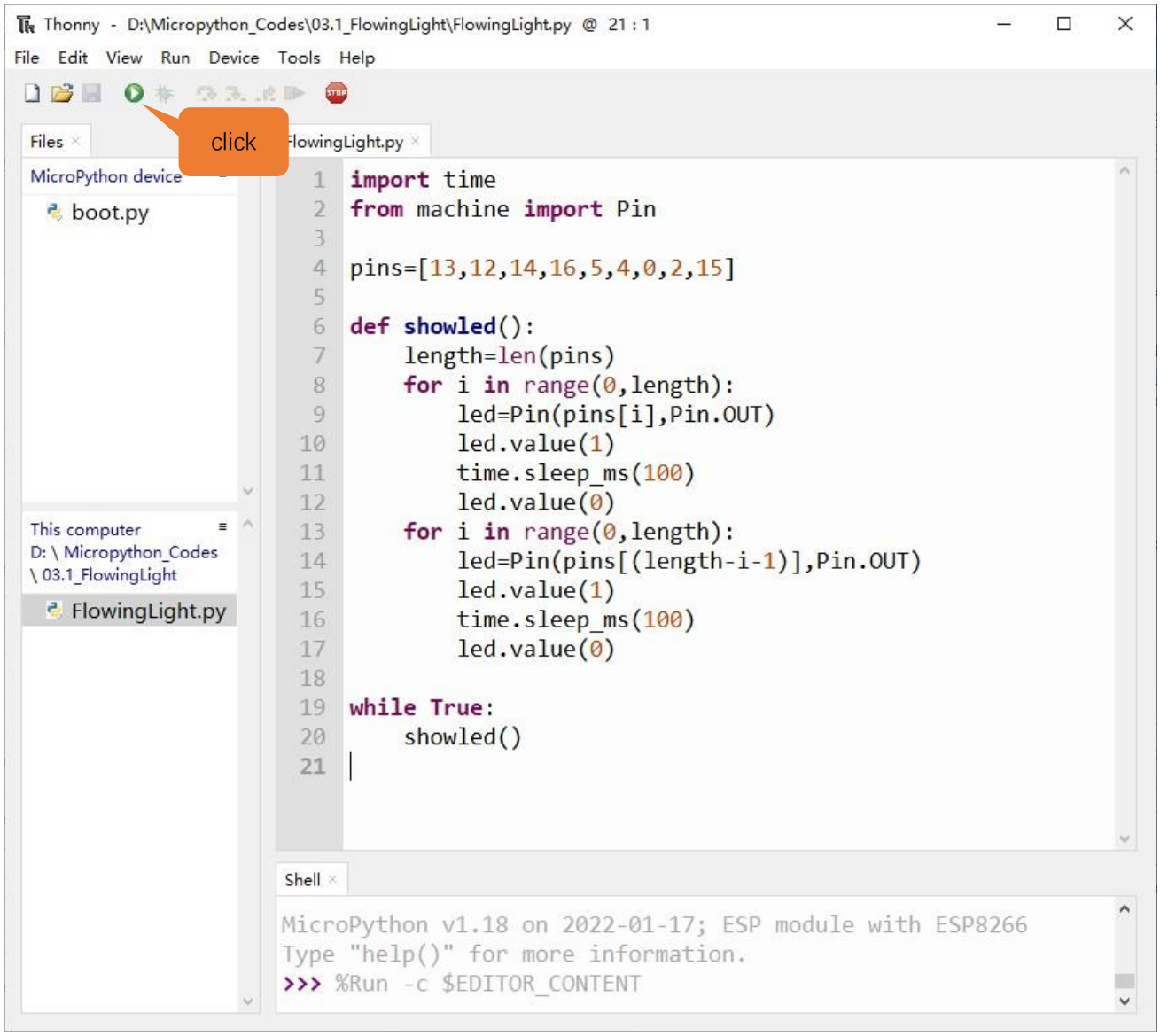
Code

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.

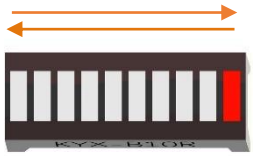
03.1_FlowingLight

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP8266/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".

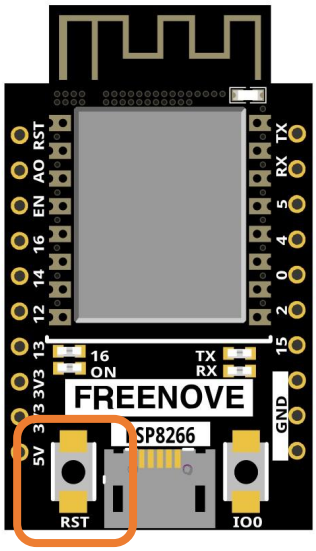


Click “Run current script” shown in the box above, LED Bar Graph will light up from left to right and then back from right to left.



Press the "RST" button on the ESP8266 development board and exit the program. You can also click “Run current script” again.

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



The following is the program code:

```
import time
from machine import Pin

pins=[13,12,14,16,5,4,0,2,15]

def showled():
    length=len(pins)
    for i in range(0,length):
        led=Pin(pins[i],Pin.OUT)
        led.value(1)
        time.sleep_ms(100)
        led.value(0)
    for i in range(0,length):
        led=Pin(pins[(length-i-1)],Pin.OUT)
        led.value(1)
        time.sleep_ms(100)
        led.value(0)

```

```
while True:
    showled()
```

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

```
pins=[13, 12, 14, 16, 5, 4, 0, 2, 15]
```

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

```
    length=len(pins)
    for i in range(0,length):
        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.

```
    for i in range(0,length):
        led=Pin(pins[i],Pin.OUT)
        led.value(1)
        time.sleep_ms(100)
        led.value(0)
    for i in range(0,length):
        led=Pin(pins[(length-i-1)],Pin.OUT)
        led.value(1)
        time.sleep_ms(100)
        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

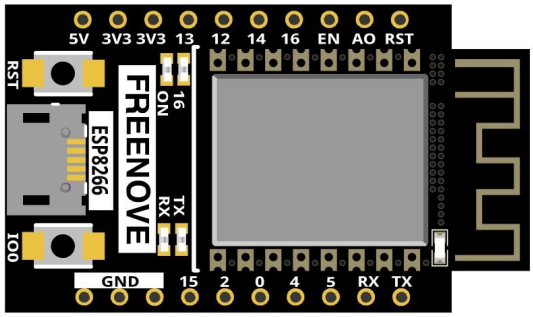
In previous study, we have known that one button has two states: pressed and released, and LED has light-on/off state, then how to enter a middle state? How to output an intermediate state to let LED "semi bright"? That's what we're going to learn.
First, let's learn how to control the brightness of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

Project 4.1 Breathing LED

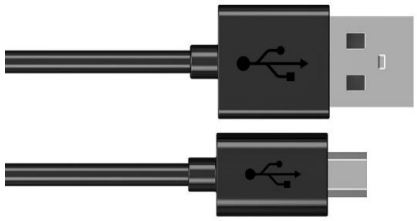
Breathing light, that is, LED is turned from off to on gradually, and gradually from on to off, just like "breathing". So, how to control the brightness of a LED? We will use PWM to achieve this target.

Component List

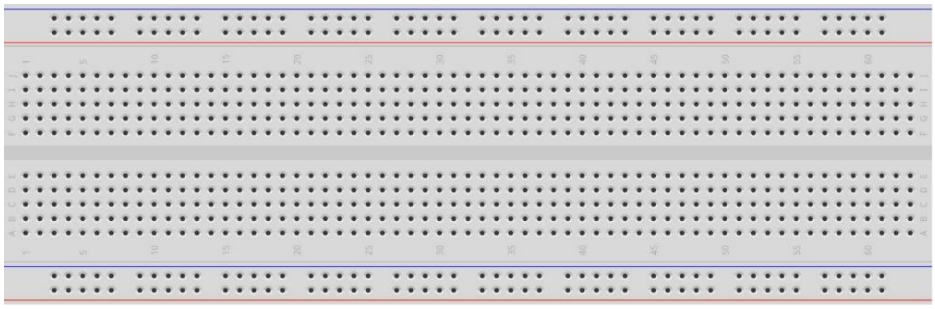
ESP8266 x1



USB cable



Breadboard x1



LED x1

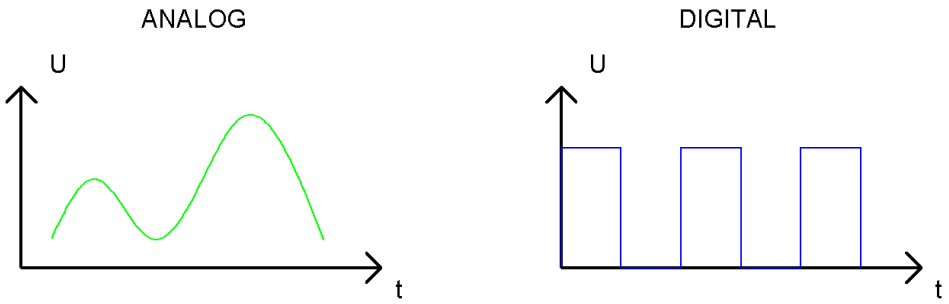
Resistor 220Ω x1

Jumper wire M/M x3

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.



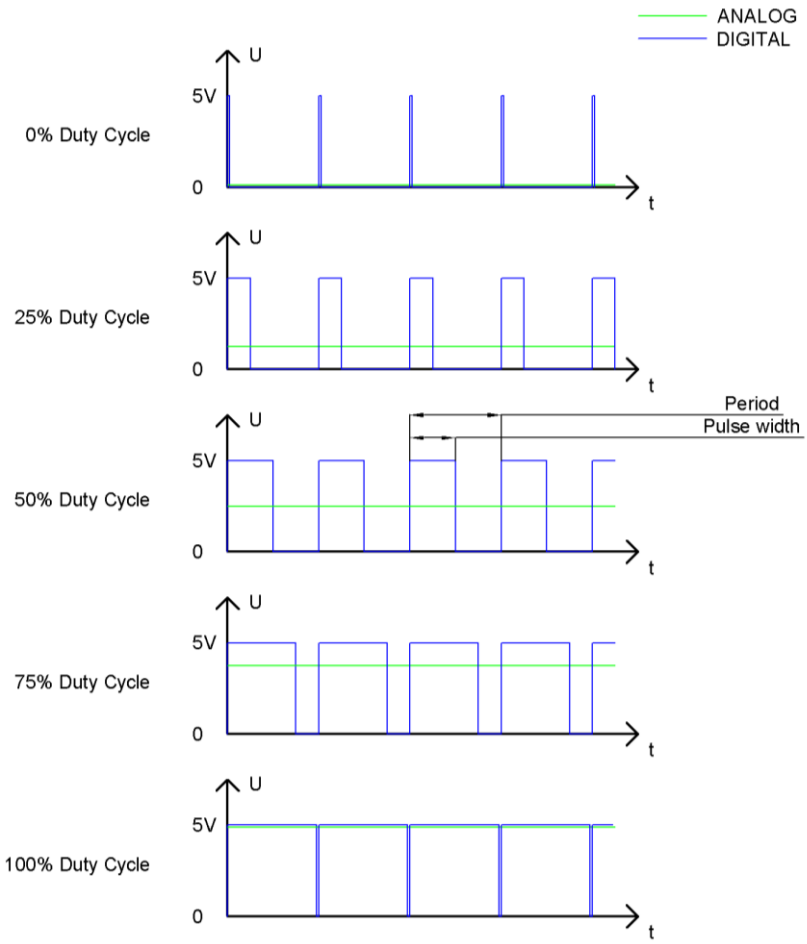
In practical application, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1), it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

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%:

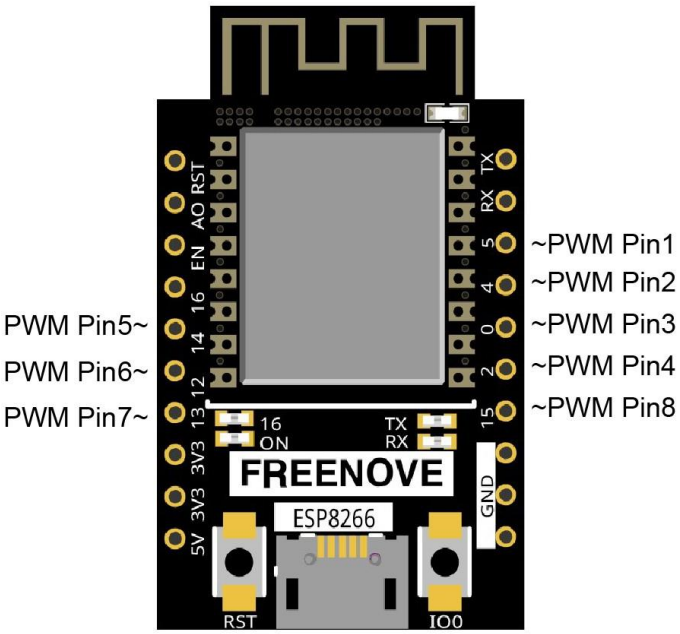


The longer the PWM duty cycle is, the higher the output power will be. Now that we understand this relationship, we can use PWM to control the brightness of an LED or the speed of DC motor and so on.
It is evident from the above that PWM is not real analog, and the effective value of the voltage is equivalent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

ESP8266 and PWM

The ESP8266 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 ESP8266 are configurable and they can be configured to PWM.

The ESP8266 supports PWM pins as follows:

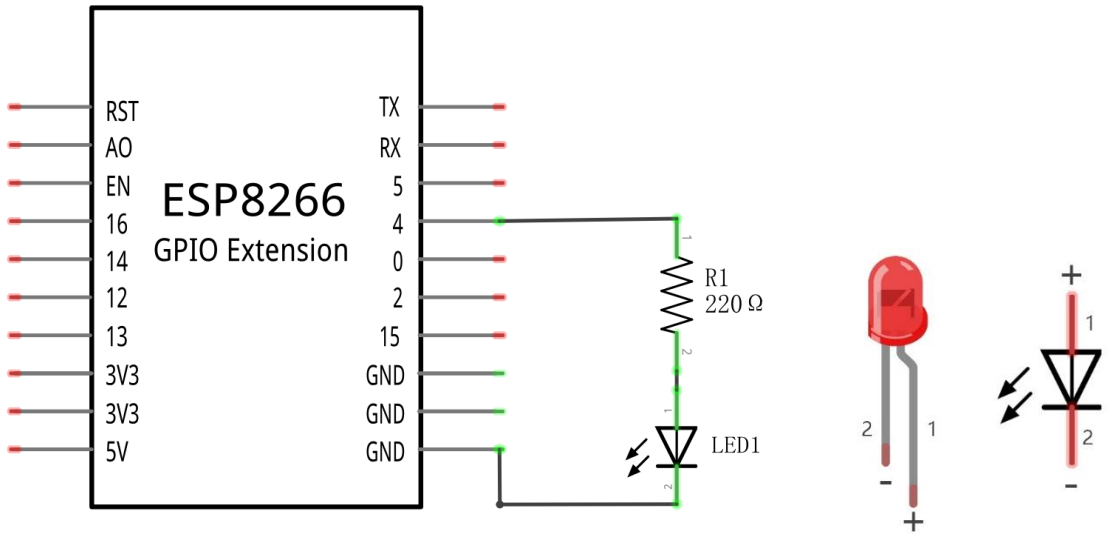


The longer the PWM duty cycle is, the higher the output power will be. Now that we understand this relationship, we can use PWM to control the brightness of an LED or the speed of DC motor and so on.

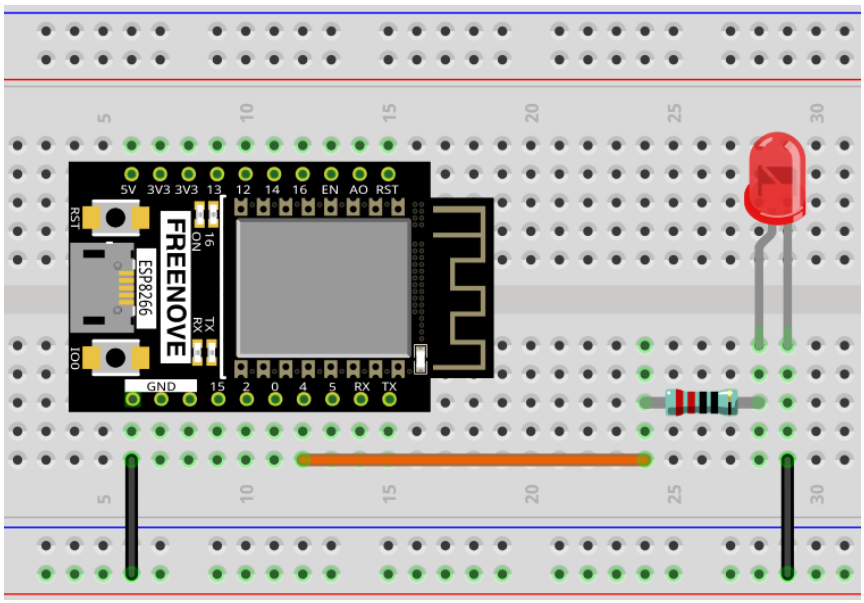
Circuit

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



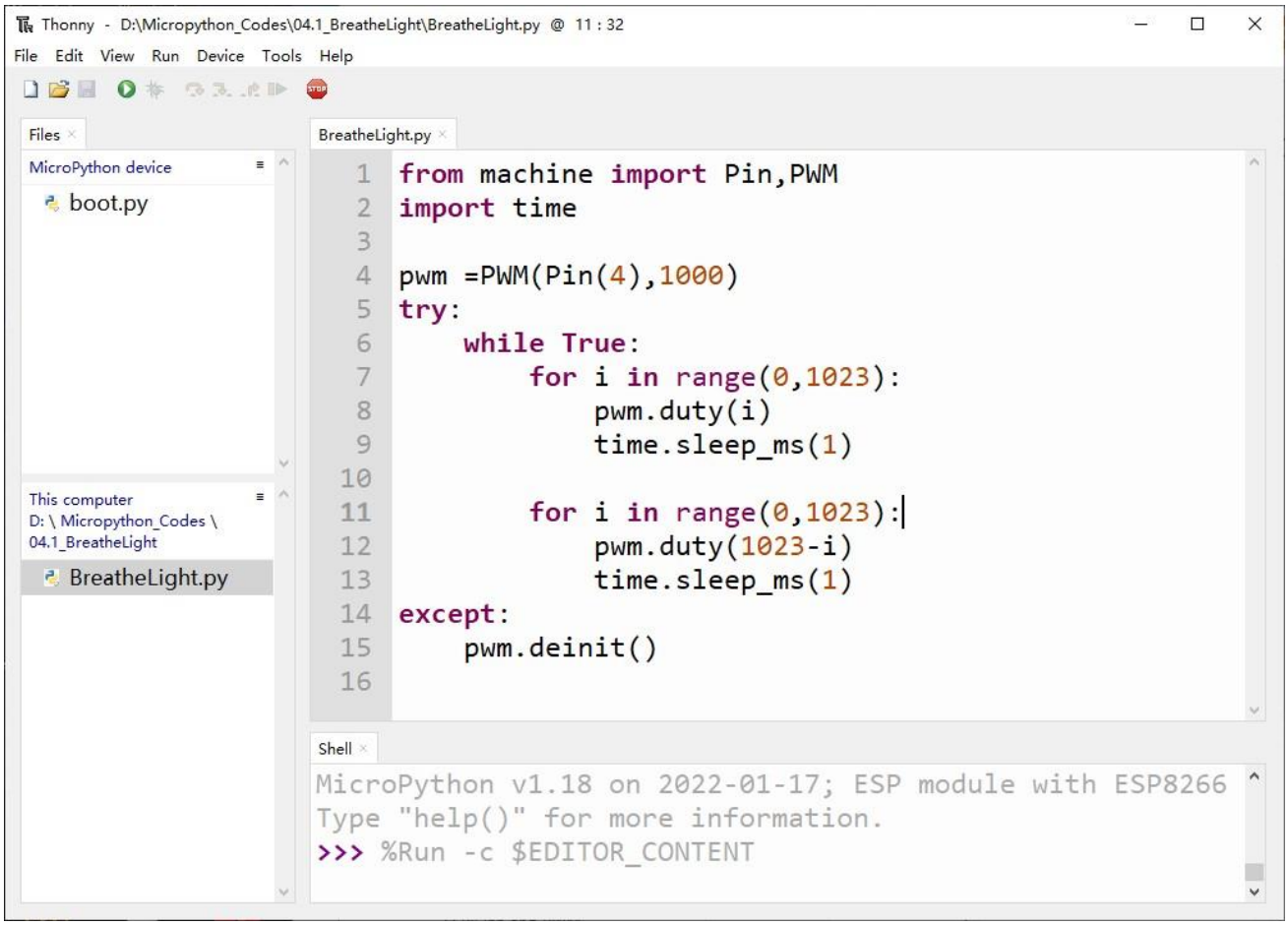
Code

This project is designed to make PWM output GPIO4 with pulse width increasing from 0% to 100%, and then reducing from 100% to 0% gradually.

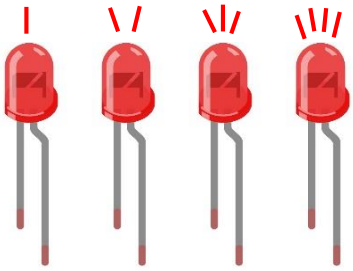
Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

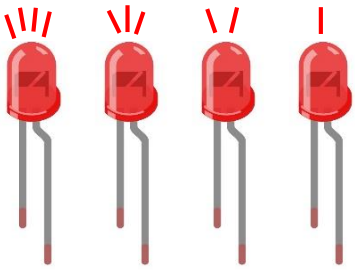
Open "Thonny", click"This computer" → "D:" → "Micropython_Codes" → "04.1_BreatheLight" and double click "BreatheLight.py".

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:

```
from machine import Pin,PWM
import time

pwm =PWM(Pin(4),1000)
try:
    while True:
        for i in range(0,1023):
            pwm.duty(i)
            time.sleep_ms(1)

        for i in range(0,1023):
            pwm.duty(1023-i)
            time.sleep_ms(1)
except:
    pwm.deinit()
```

The way that the ESP8266 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 GPIO4's output frequency as 1000Hz, and assign them to PWM.

```
pwm =PWM(Pin(4),1000)
```

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%.

```
        for i in range(0,1023):
            pwm.duty(i)
            time.sleep_ms(1)

        for i in range(0,1023):
            pwm.duty(1023-i)
            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.

```
pwm.deinit()
```

Note: PWM can be enabled on all pins except pin (16). All channels have a frequency that ranges from 1 to 1000 (measured in Hz). Duty cycle is between 0 and 1023 inclusive.

Reference

Class PWM(pin, freq)

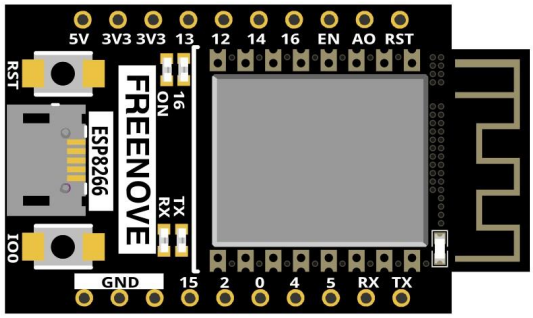
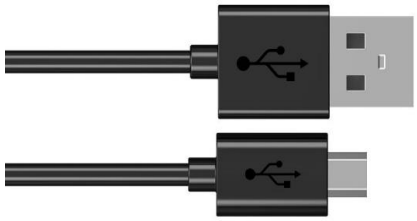
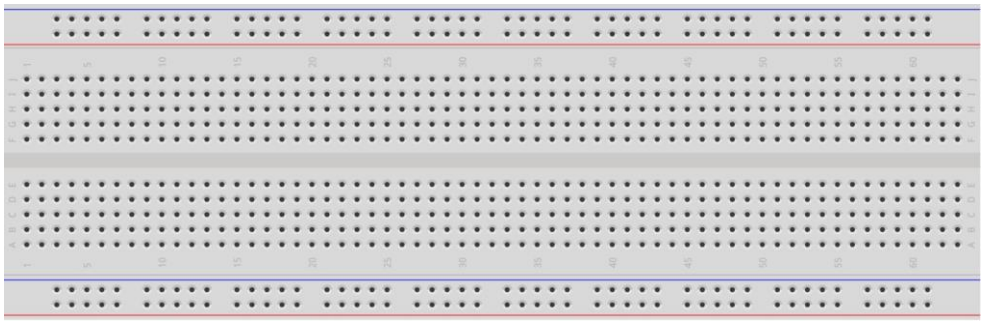
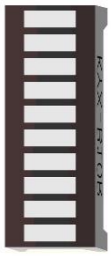
Before each use of PWM module, please add the statement "**from machine import PWM**" to the top of the python file.

pin: PWM can be enabled on all pins except pin (16), such as Pin(0)、Pin(2)….
freq: Output frequency, with the range of 0-1000 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

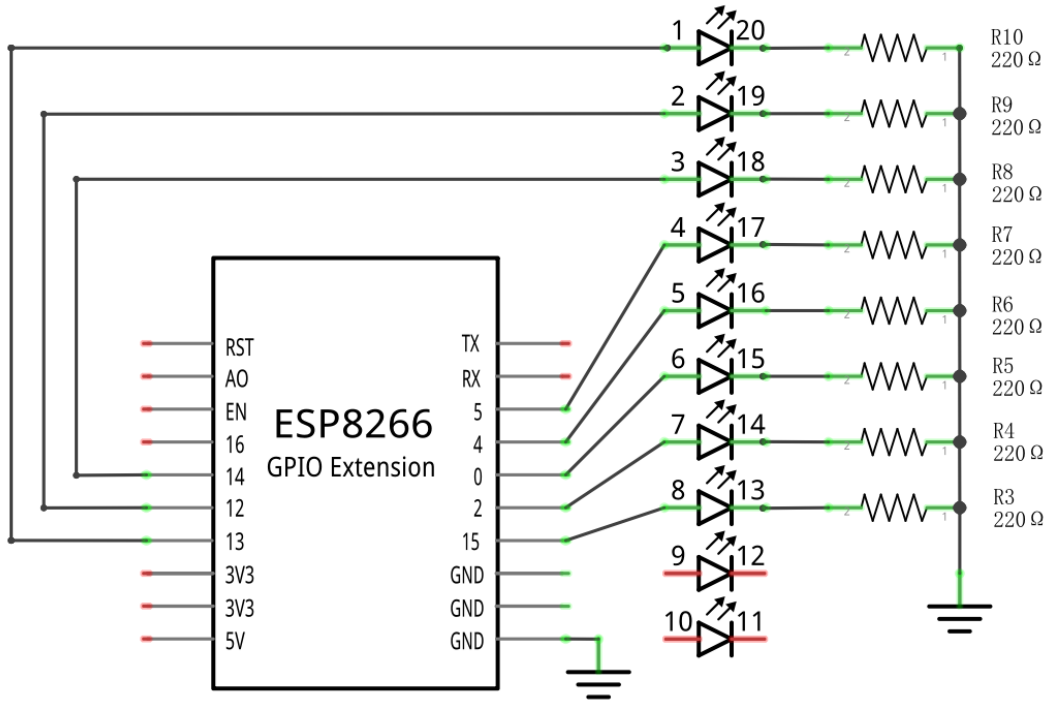
After learning about PWM, we can use it to control LED Bar Graph and realize a cooler Flowing Light.

Component List

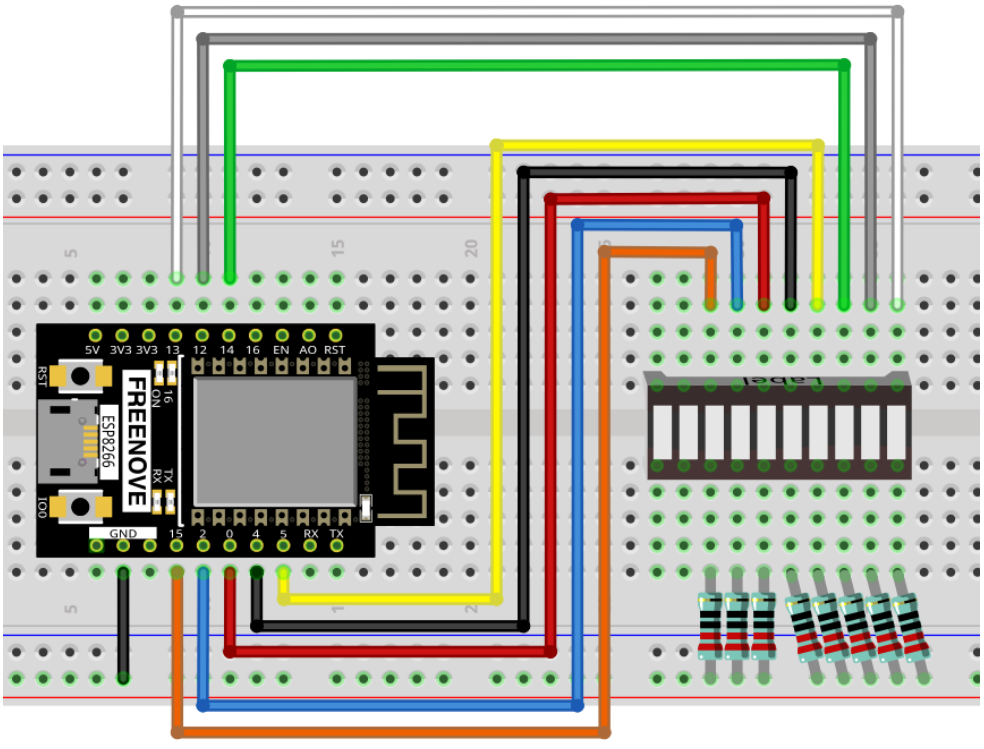
ESP8266 x1 	USB cable 	
Breadboard x1 		
Jumper wire M/M x9 	LED bar graph x1 	Resistor 220Ω x8 

Circuit

Schematic diagram



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



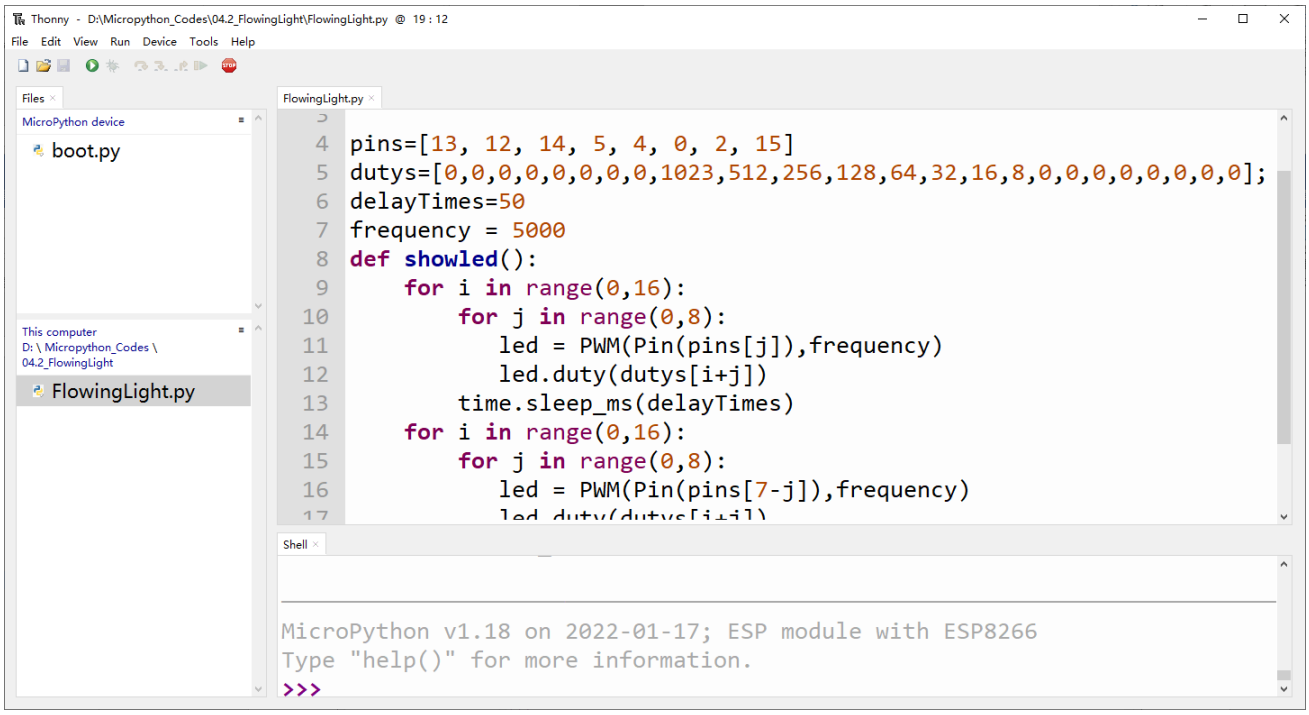
If LEDbar doesn't work, try to rotate LEDbar for 180°. The label is random.

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 ESP8266 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:

```
from machine import Pin,PWM
import time
pins=[13, 12, 14, 5, 4, 0, 2, 15]
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];
delayTimes=50
frequency = 1000
def showled():
    for i in range(0,16):
        for j in range(0,8):
            led = PWM(Pin(pins[j]),frequency)
            led.duty(dutys[i+j])
        time.sleep_ms(delayTimes)
    for i in range(0,16):
        for j in range(0,8):
            led = PWM(Pin(pins[7-j]),frequency)
            led.duty(dutys[i+j])
        time.sleep_ms(delayTimes)
while True:
    showled()
```

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

```
from machine import Pin,PWM

pins=[13, 12, 14, 5, 4, 0, 2, 15]
```

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

```
pins=[13, 12, 14, 5, 4, 0, 2, 15]
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];
```

Set the PWM pin[j] and duty cycle [I +j].

```
            led = PWM(Pin(pins[j]),frequency)
            led.duty(dutys[i+j])
```

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

```
8       for i in range(0,16):
9           for j in range(0,8):
10              led = PWM(Pin(pins[j]),frequency)
11              led.duty(dutys[i+j])
12          time.sleep_ms(delayTimes)
13      for i in range(0,16):
14          for j in range(0,8):
15              led = PWM(Pin(pins[7-j]),frequency)
16              led.duty(dutys[i+j])
17          time.sleep_ms(delayTimes)
```

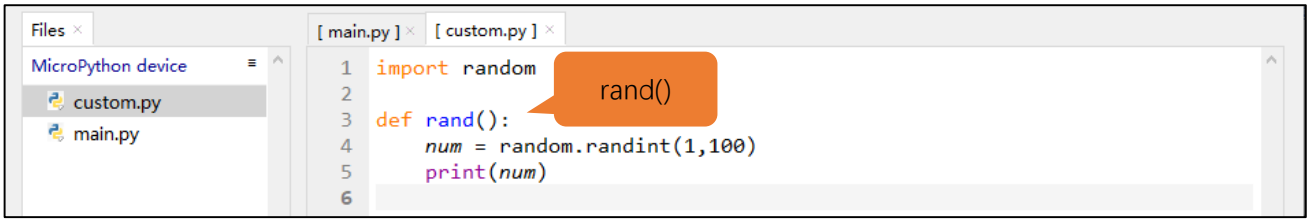
In the main function, a nested for loop is used to control the pulse width of the PWM. Every time i in the first for loop increases by 1, the LED Bar Graph will move one grid, and gradually change according to the value in the array dutys. As shown in the following table, the value in the second row 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																								

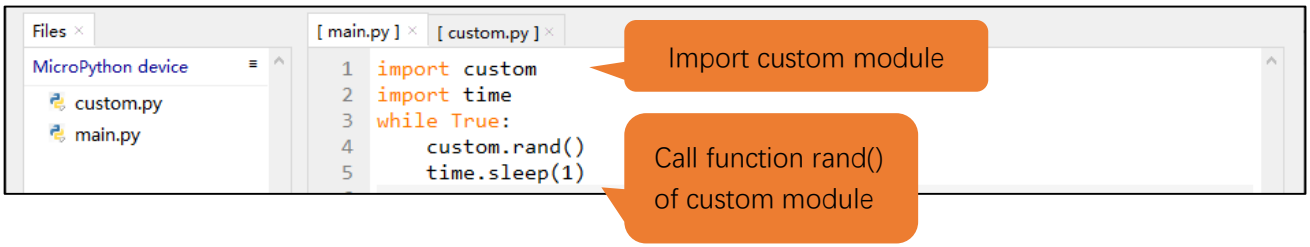
How to import a custom python module

Each Python file, as long as it's stored on the file system of ESP8266, 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 ESP8266.



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



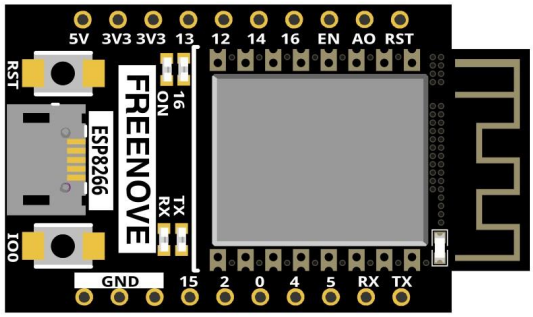
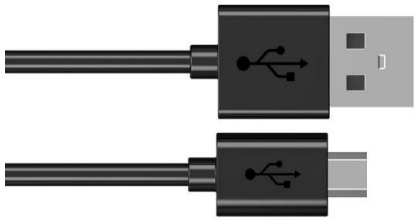
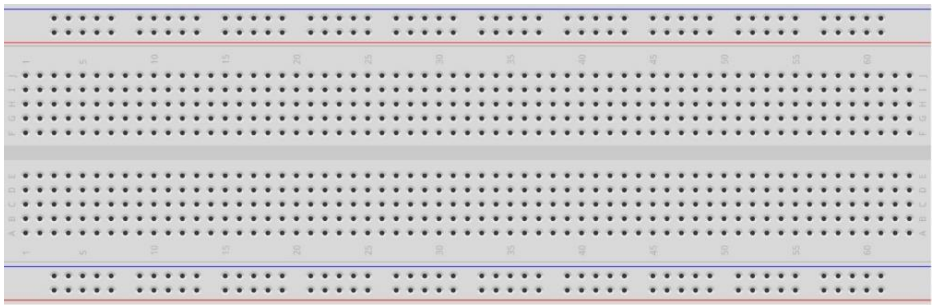
Chapter 5 RGBLED

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

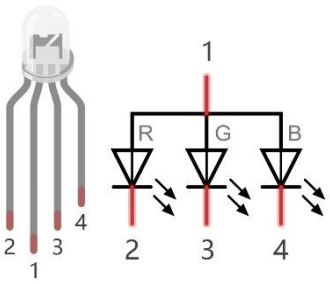
In this project, we will make a multicolored LED. And we can control RGBLED to switch different colors automatically.

Component List

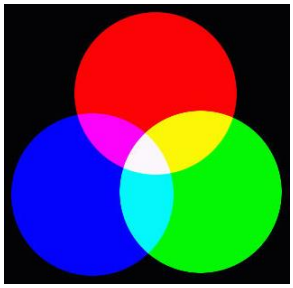
ESP8266 x1 	USB cable 	
Breadboard x1 		
RGBLED x1 	Resistor 220Ω x3 	Jumper wire M/M x5 

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.

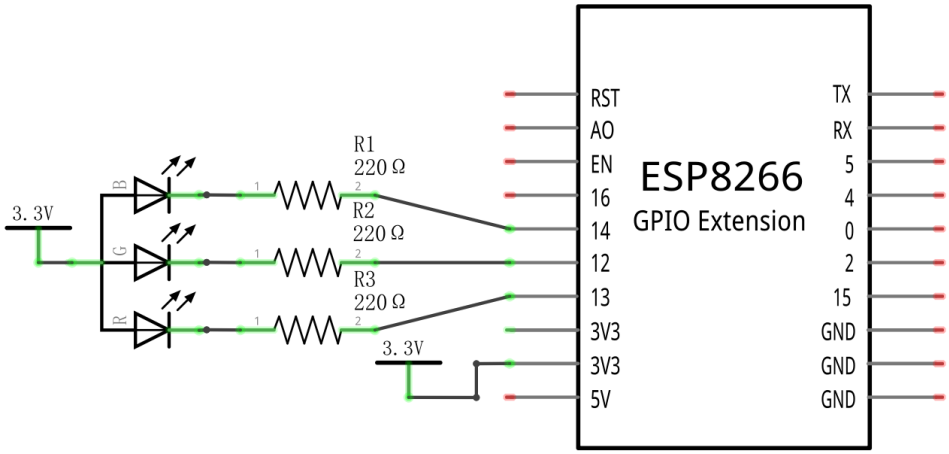


RGB

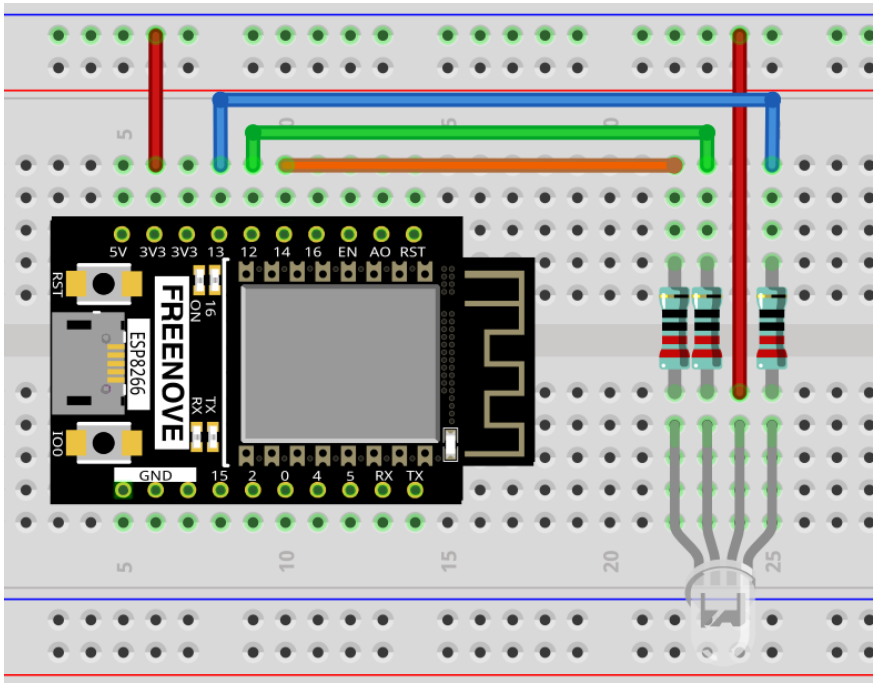
If we use three 10-bit PWM to control the RGBLED, in theory, we can create $2^{10}*2^{10}*2^{10}$= 1,073,741,824(1 billion) colors through different combinations.

Circuit

Schematic diagram



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



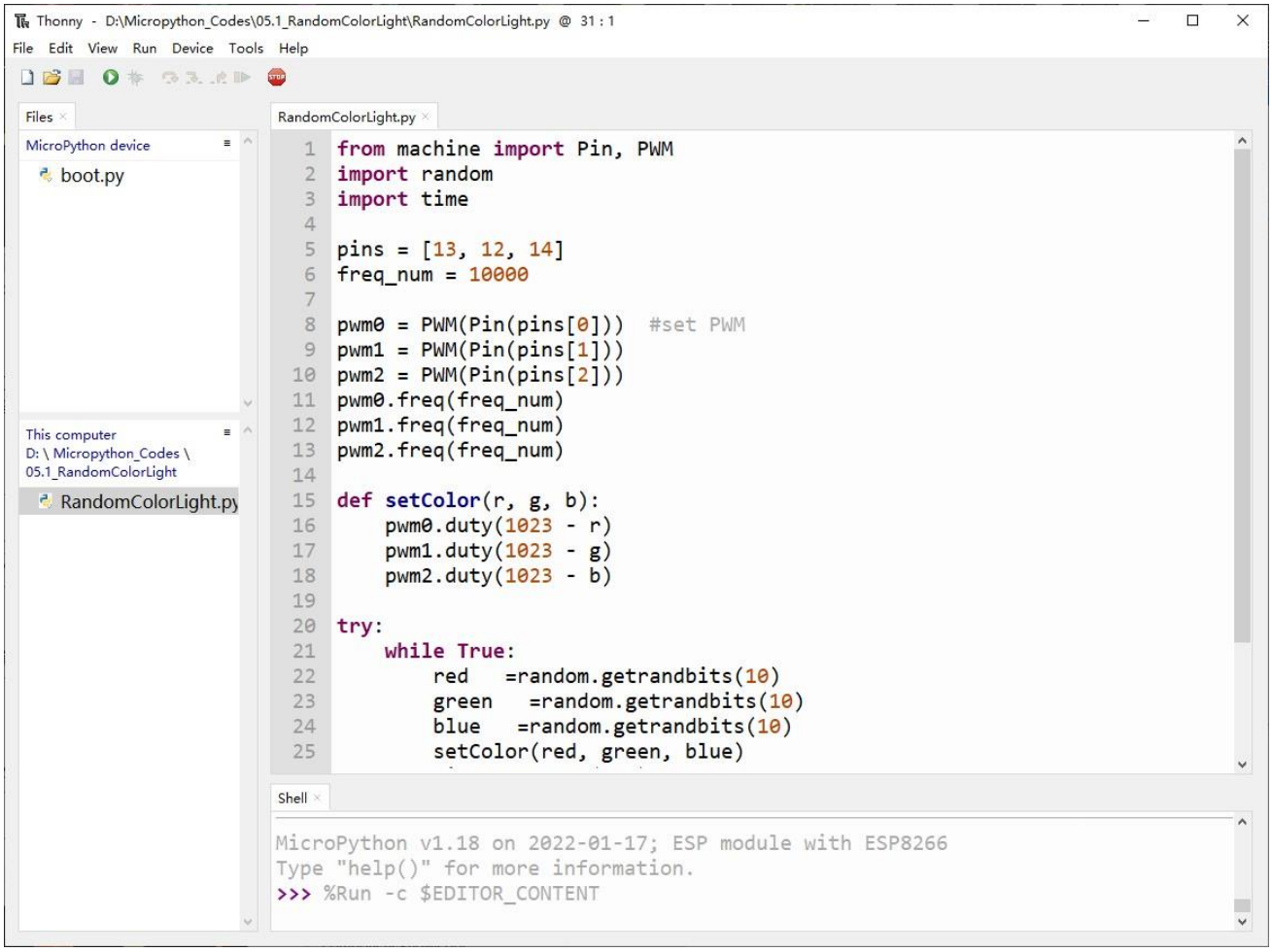
Code

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_ESP8266/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
import random
import time

pins=[13, 12, 14]
freq_num = 1000

pwm0 = PWM(Pin(pins[0]))  #set PWM
pwm1 = PWM(Pin(pins[1]))
pwm2 = PWM(Pin(pins[2]))
pwm0.freq(freq_num)
pwm1.freq(freq_num)
pwm2.freq(freq_num)

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

try:
    while True:
        red   =random.getrandbits(10)
        green =random.getrandbits(10)
        blue  =random.getrandbits(10)
        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
import random
import time
```

Configure ouput mode of GPIO13, GPIO12 and GPIO14 as PWM output and PWM frequency as 1000Hz

```
pins=[13, 12, 14]
freq_num = 1000
pwm0 = PWM(Pin(pins[0]))  #set PWM
pwm1 = PWM(Pin(pins[1]))
pwm2 = PWM(Pin(pins[2]))
pwm0.freq(freq_num)
pwm1.freq(freq_num)
pwm2.freq(freq_num)
```

Define a function to set the color of RGBLED.

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

Call random function getrandbits(size) to generates an integer with 10 random bits and assign the value to red. size = 10, it generates an integer in the range of 0 to 0b1111111111

```
22          red   =random.getrandbits(10)
```

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

```
17      while True:
18          red   =random.getrandbits(10)
19          green =random.getrandbits(10)
20          blue  =random.getrandbits(10)
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:

```
1  from machine import Pin,PWM
2  import time
3
4  pins=[14,12,13];
5
6  pwm0=PWM(Pin(pins[0]),1000)
7  pwm1=PWM(Pin(pins[1]),1000)
8  pwm2=PWM(Pin(pins[2]),1000)
9
10 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

    elif WheelPos>=341 and WheelPos<682:
        WheelPos -= 341;
        red=0
        green=1023-WheelPos*3
        blue=WheelPos*3
    else :
        WheelPos -= 682;
        red=WheelPos*3
        green=0
        blue=1023-WheelPos*3

try:
    while True:
        for i in range(0,1023):
            wheel(i)
            setColor()
            time.sleep_ms(15)
except:
    pwm0.deinit()
    pwm1.deinit()
    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.

```
def wheel(pos):
    global red,green,blue
    WheelPos=pos%1023
    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
```

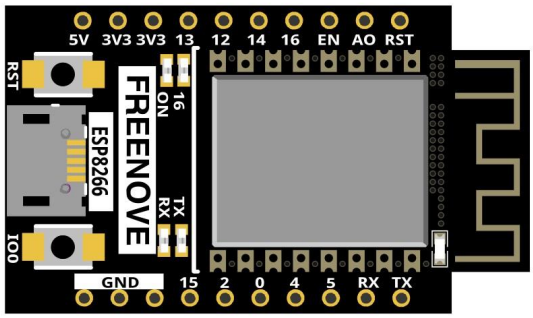
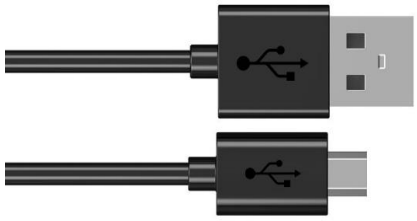
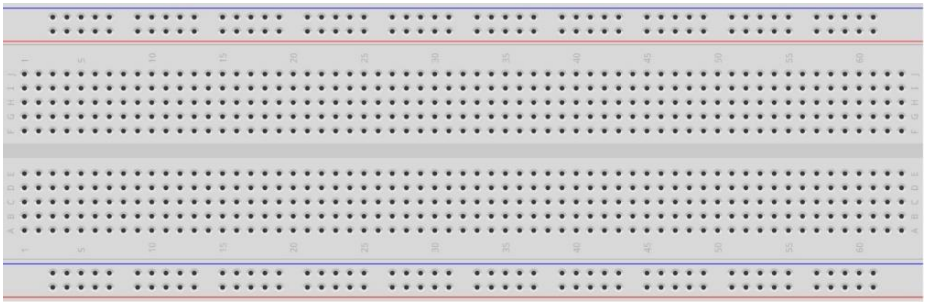
Chapter 6 Buzzer

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

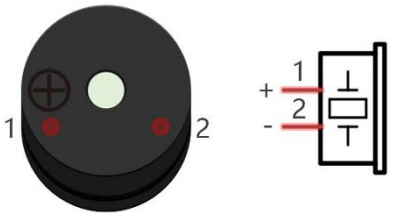
<table>
<tr><td colspan="3">ESP8266 x1
  </td><td colspan="2">USB cable
  </td></tr>
<tr><td colspan="5">Breadboard x1
  </td></tr>
<tr><td colspan="5">Jumper wire M/M x9
  </td></tr>
<tr><td>NPN transistorx1
(S8050)
  </td><td>Active buzzer x1
  </td><td>Push button x1
  </td><td>Resistor 1kΩ x1
  </td><td>Resistor 10kΩ x2
  </td></tr>
</table>

Component knowledge

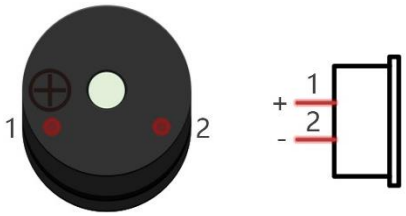
Buzzer

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

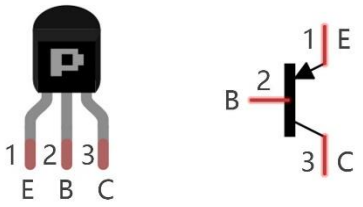
Passive buzzer

Transistor

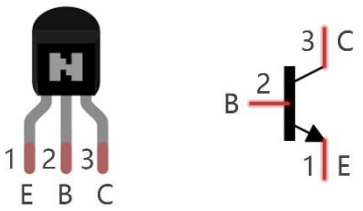
Because the buzzer requires such large current that GPIO of ESP8266 output capability cannot meet the requirement, a transistor of NPN type is needed here to amplify the current.

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

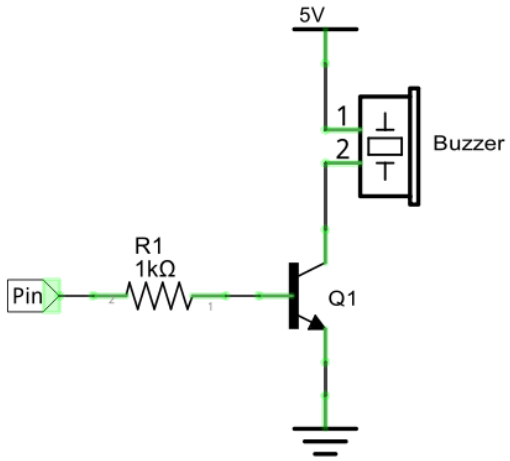
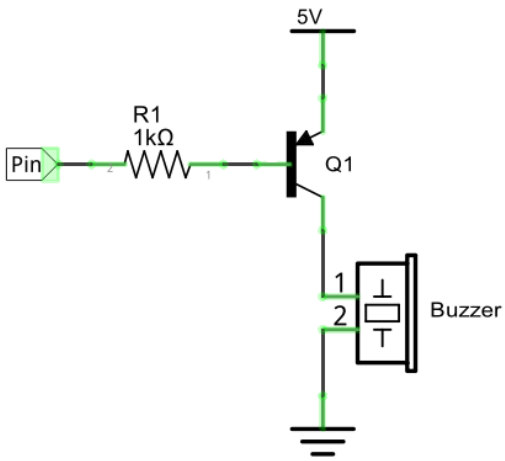


In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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.

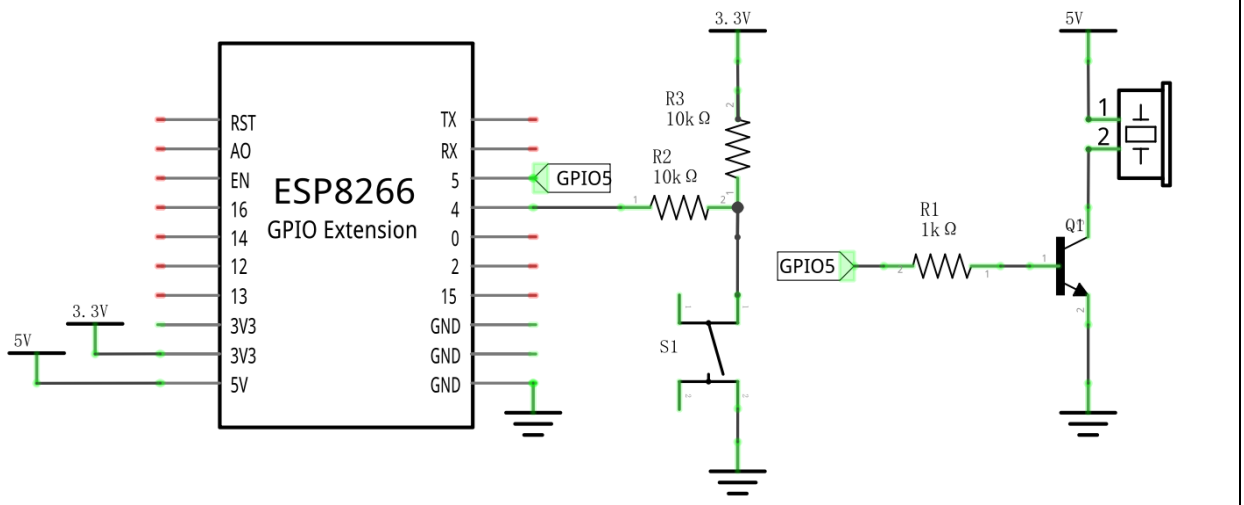
When using NPN transistor to drive buzzer, we often adopt the following method. If GPIO outputs high level, current will flow through R1, the transistor will get conducted, and the buzzer will sound. If GPIO outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer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

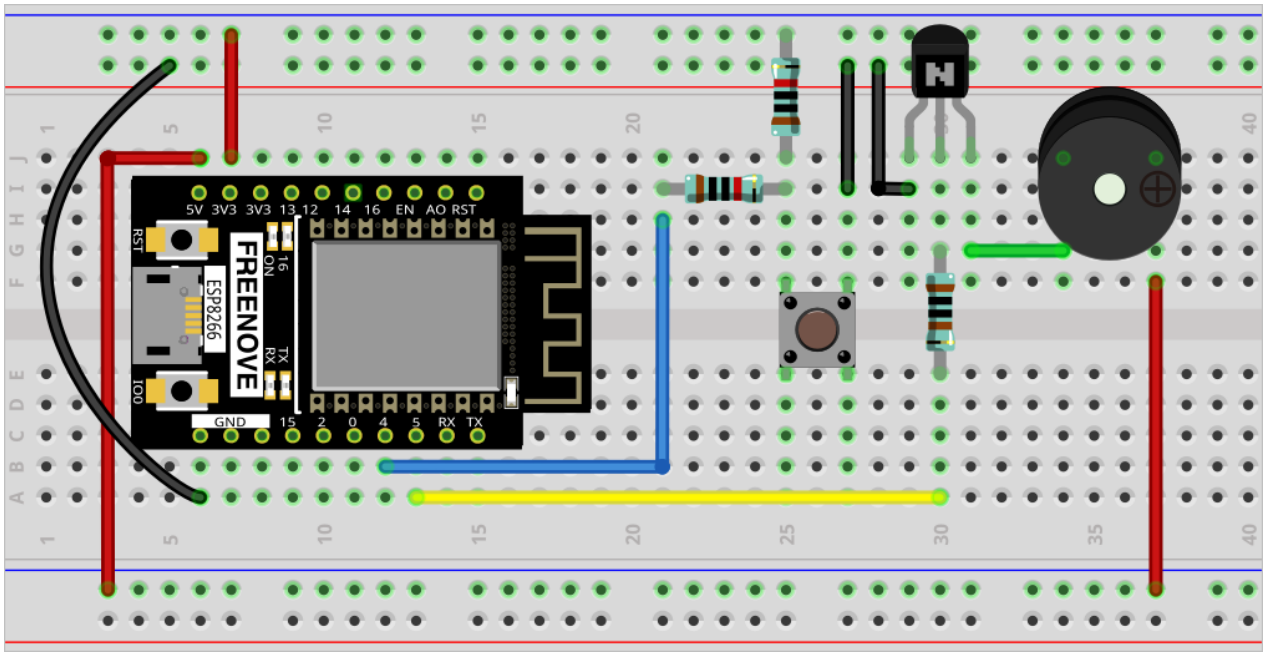
NPN transistor to drive buzzer	PNP transistor to drive buzzer
	

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.

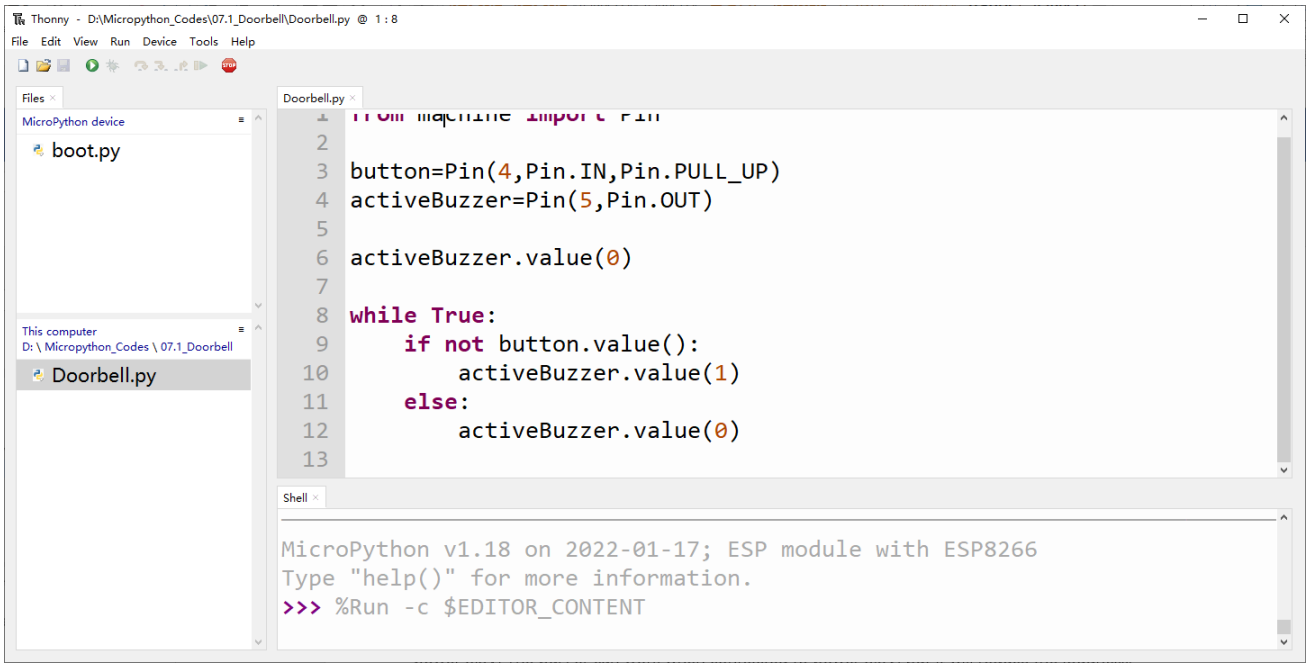
Code

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.

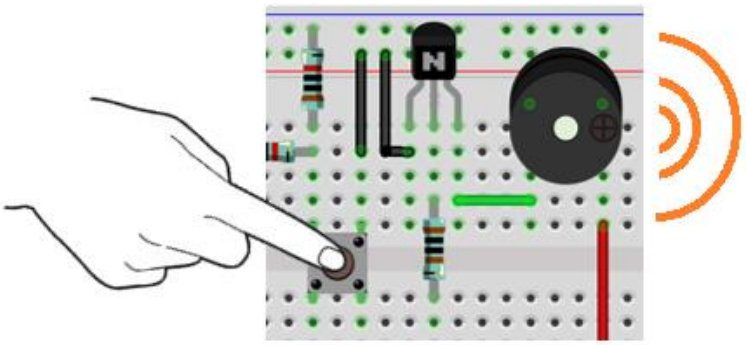
Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

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

07.1_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(5,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.

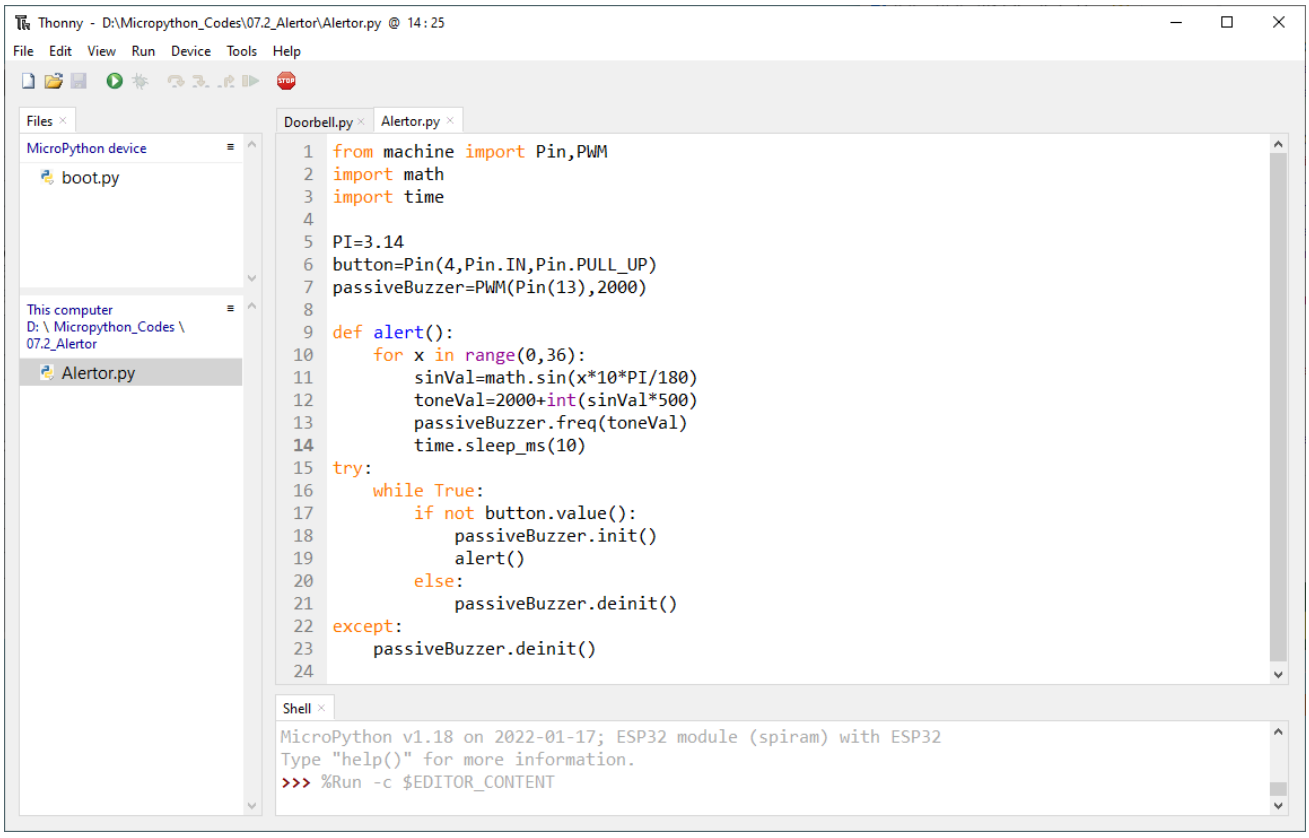
Component list and the circuit part is similar to last section. In the Doorbell circuit only the **active buzzer** needs to be **replaced** with a **passive buzzer.**

Code

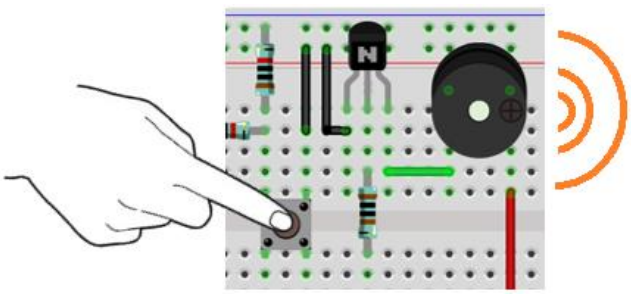
In this project, the buzzer alarm is controlled by the button. Press the button, then buzzer sounds. If you release the button, the buzzer will stop sounding. In the logic, it is the same as using button to control LED. In the control method, passive buzzer requires PWM of certain frequency to sound.

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

07.2_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:

```
1  from machine import Pin,PWM
2  import math
3  import time
4
5  PI=3.14
6  button=Pin(4,Pin.IN,Pin.PULL_UP)
7  passiveBuzzer=PWM(Pin(5),1000)
8
9  def alert():
10     for x in range(0,36):
11         sinVal=math.sin(x*10*PI/180)
12         toneVal=500+int(sinVal*500)
13         passiveBuzzer.duty(1000)
14         passiveBuzzer.freq(toneVal)
15         time.sleep_ms(10)
16 try:
17     while True:
18         if not button.value():
19             passiveBuzzer.init()
20             alert()
21         else:
22             passiveBuzzer.duty(0)
23             passiveBuzzer.deinit()
24 except:
25     passiveBuzzer.deinit()
```

Import PWM, Pin, math and time modules.

```
1  from machine import Pin,PWM
2  import math
3  import time
```

Define the pins of the button and passive buzzer.

```
5  PI=3.14
6  button=Pin(4,Pin.IN,Pin.PULL_UP)
7  passiveBuzzer=PWM(Pin(5),1000)
```

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

```
9  def alert():
10     for x in range(0,36):
11         sinVal=math.sin(x*10*PI/180)
12         toneVal=500+int(sinVal*500)
13         passiveBuzzer.duty(1000)
14         passiveBuzzer.freq(toneVal)
15         time.sleep_ms(10)
```

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

```
22             passiveBuzzer.duty(0)
```

```
23                  passiveBuzzer.deinit()
```

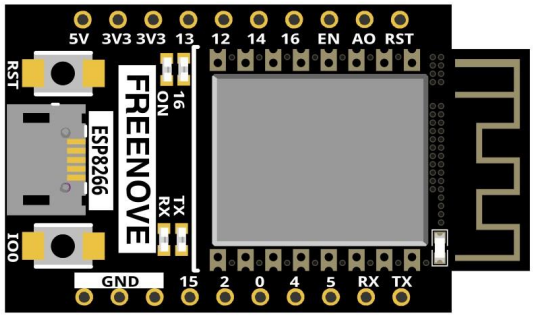
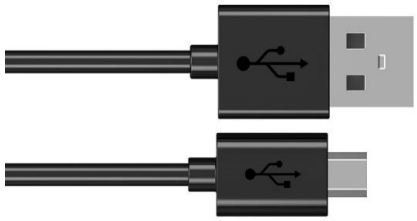
Chapter 7 Serial Communication

Serial Communication is a means of Communication between different devices/devices. This section describes ESP8266's Serial Communication.

Project 7.1 Serial Print

This project uses ESP8266's serial communicator to send data to the computer and print it on the serial monitor.

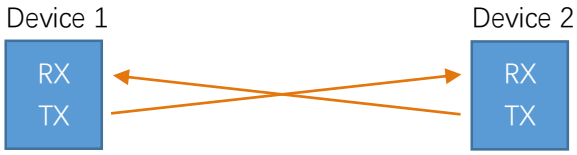
Component List

ESP8266 x1 	USB cable 

Related knowledge

Serial communication

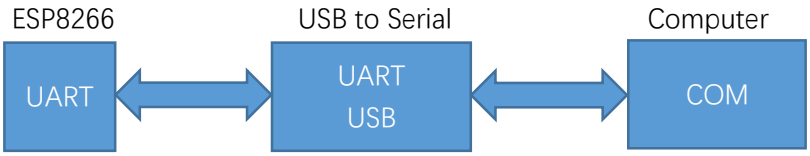
Serial communication generally refers to the Universal Asynchronous Receiver/Transmitter (UART), which is commonly used in electronic circuit communication. It has two communication lines, one is responsible for sending data (TX line) and the other for receiving data (RX line). The serial communication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

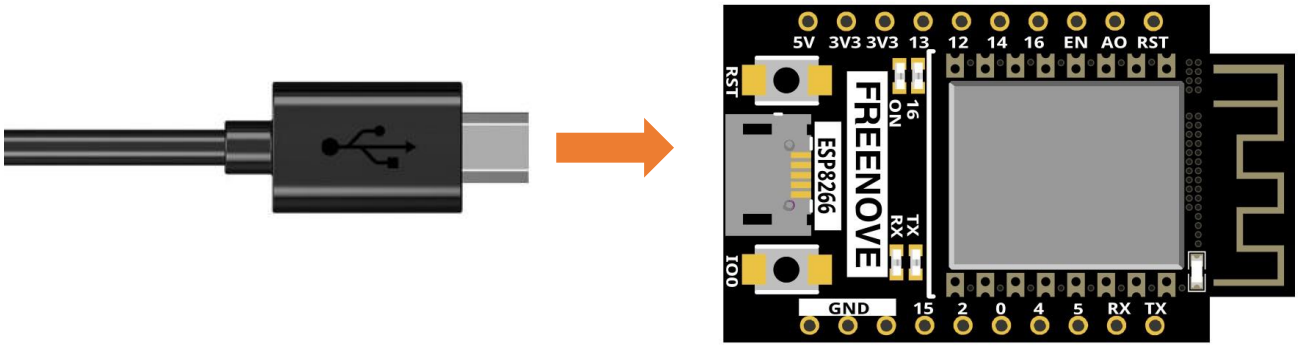
Serial port on ESP8266

Freenove ESP8266 has integrated USB to serial transfer, so it could communicate with computer connecting to USB cable.



Circuit

Connect Freenove ESP8266 to the computer with USB cable.



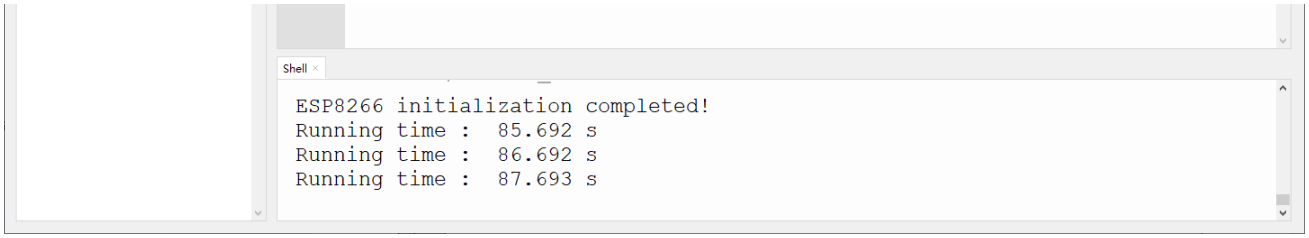
Code

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP8266/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_Serial_Print" and double "Serial_Print.py".

08.1_Serial_Print

Click "Run current script" and observe the changes of "Shell", which will display the time when ESP8266 is powered on once per second.



The following is the program code:

```
1  import time
2
3  print("ESP8266 initialization completed!")
4
5  while True:
6      print("Running time : ", time.ticks_ms()/1000, "s")
7      time.sleep(1)
```

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

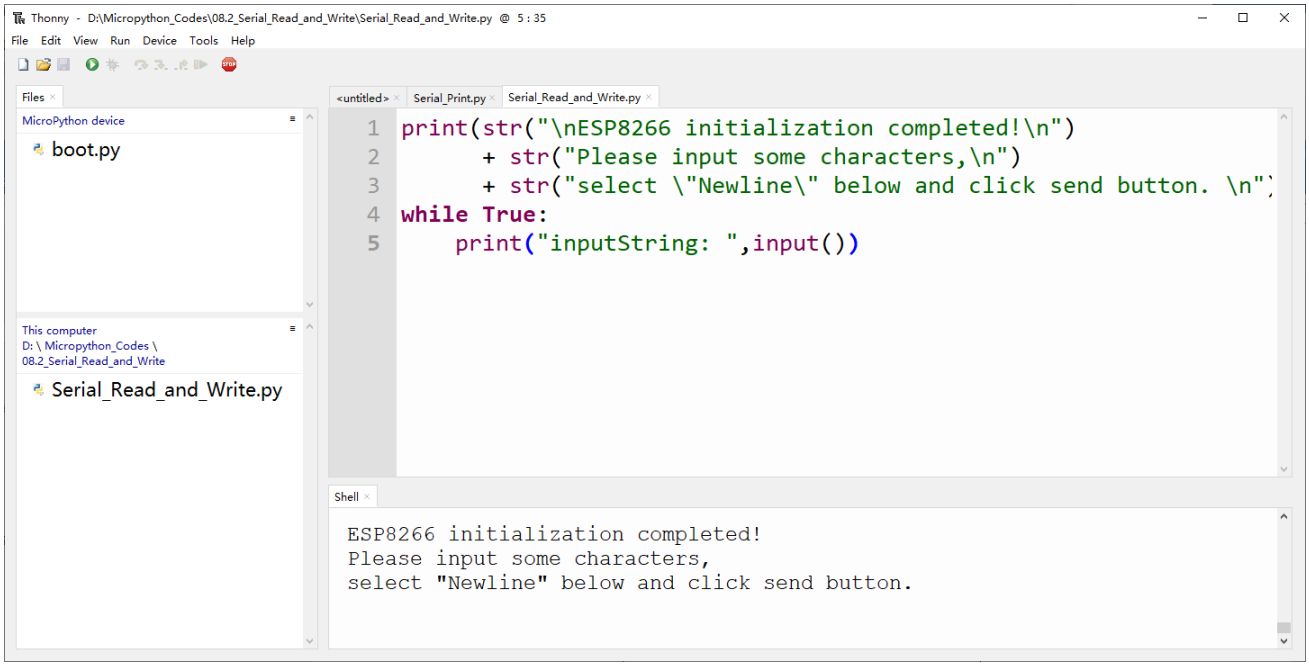
From last section, we use Serial port on Freenove ESP8266 to send data to a computer, now we will use that to receive data from computer.

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

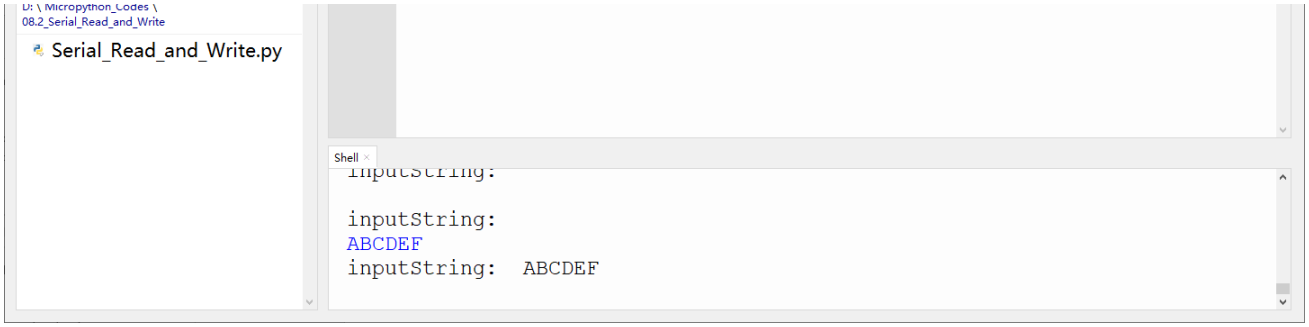
Code

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "08.2_Serial_Read_and_Write" and double click "Serial_Read_and_Write.py".

08.2_Serial_Read_and_Write



Click "Run current script" and ESP8266 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.



The following is the program code:

```
print(str("\nESP8266 initialization completed!\n")
      + str("Please input some characters,\n")
      + str("select \"Newline\" below and click send button. \n"))
while True:
    print("inputString: ",input())
```

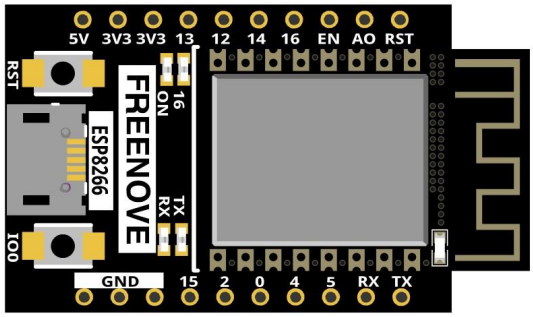
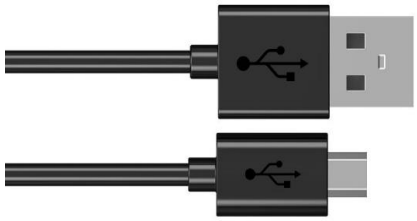
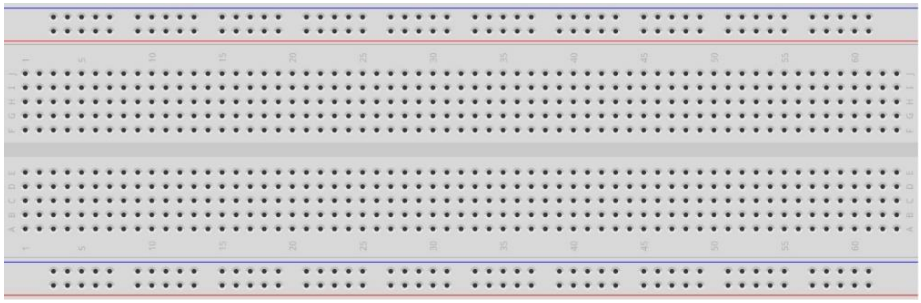
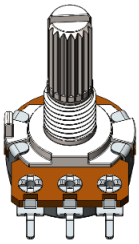
Chapter 8 ADC Converter

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. That is, ADC.

Project 8.1 Read the Voltage of Potentiometer

In this project, ADC is used to convert analog signals into digital signals. Control chip on the control board has integrated this function. Now let us try to use this function to convert analog signals into digital signals.

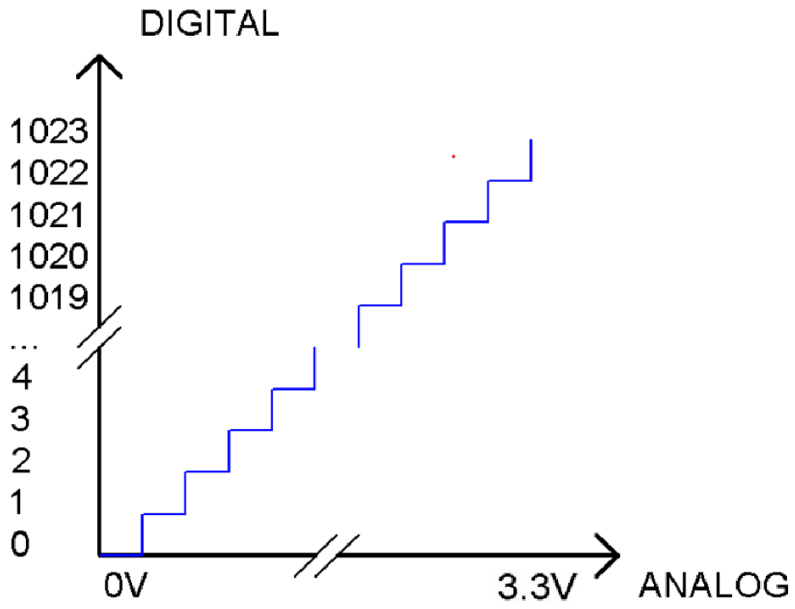
Component List

<table>
<tr><td>ESP8266 x1
  </td><td>USB cable
  </td></tr>
<tr><td colspan="2">Breadboard x1
  </td></tr>
<tr><td>Rotary potentiometer x1
  </td><td>Jumper wire M/M x3
  </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 ESP8266 is 10 bits, that means the resolution is 2^10=1024, and it represents a range (at 3.3V) will be divided equally to 1024 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---3.3/1023 V corresponds to digital 0;
Subsection 2: the analog in rang of 3.3/1023 V---2*3.3 /1023V corresponds to digital 1;
...
The following analog will be divided accordingly.
The conversion formula is as follows:

$$ADC\ \ Value = \frac{\text{Analog}\ \ \text{Voltage}}{3.3} * 1023$$

ADC on ESP8266

ESP8266 has one digital analog converters with successive approximations of 10-bit accuracy, and a total of 1 pins can be used to measure analog signals. GPIO analog pin definition are shown in the following table. Note that the input voltage on the ADC pins of the ESP8266 module must be between 0V and 1.0V. For the ESP8266 development board designed by us, its input voltage range has been sampled by resistors. The ADC input voltage of the development board is 0V to 3.3V. Do not exceed this voltage range when you use the ADC function.Exceeding this voltage range can cause permanent damage to your hardware!

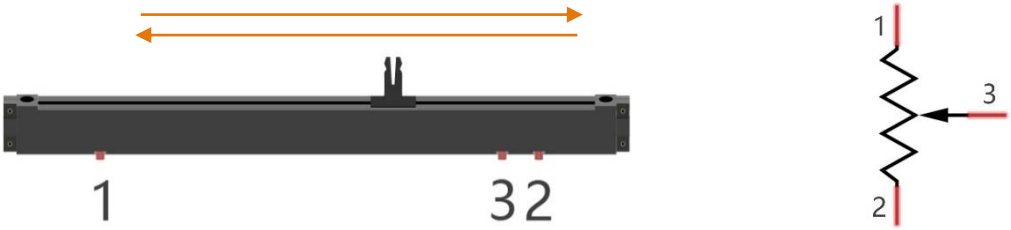
Pin number in ESP8266
A0

The analog pin number is also defined in ESP8266's code base. For example, you can use A0 in your code.

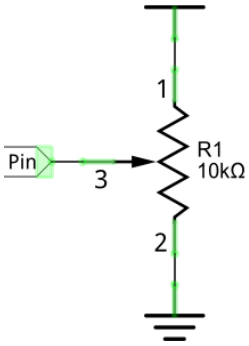
Component knowledge

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.

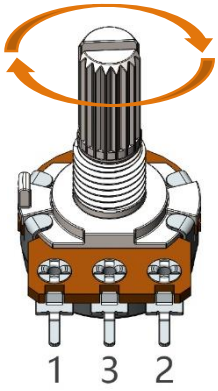


What between potentiometer pin 1 and pin 2 is the resistor body, and pins 3 is connected to brush. When brush moves from pin 1 to pin 2, the resistance between pin 1 and pin 3 will increase up to body resistance linearly, and the resistance between pin 2 and pin 3 will decrease down to 0 linearly.
In 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

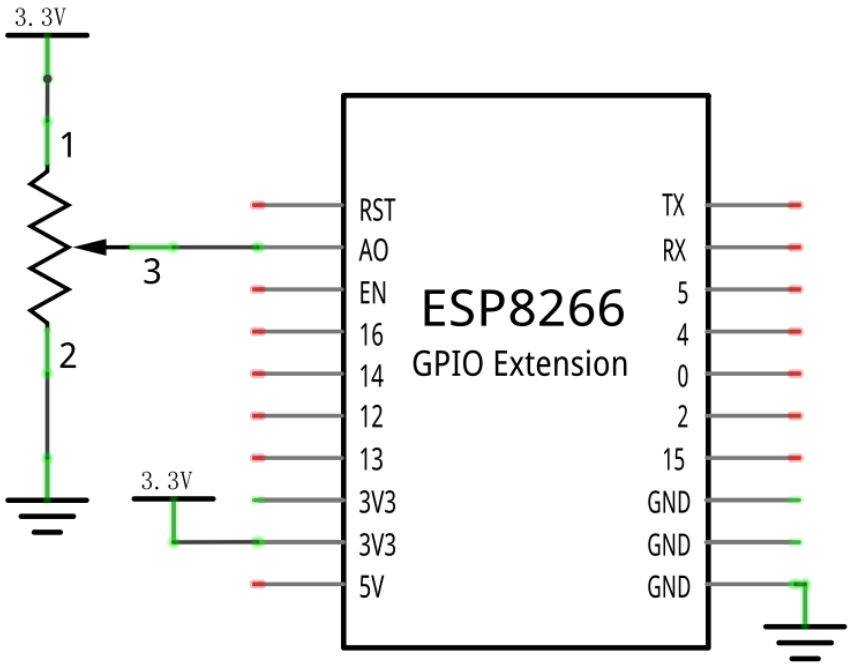
Rotary potentiometers and linear potentiometers have the same function; the only difference being the physical action being a rotational rather than a sliding movement.



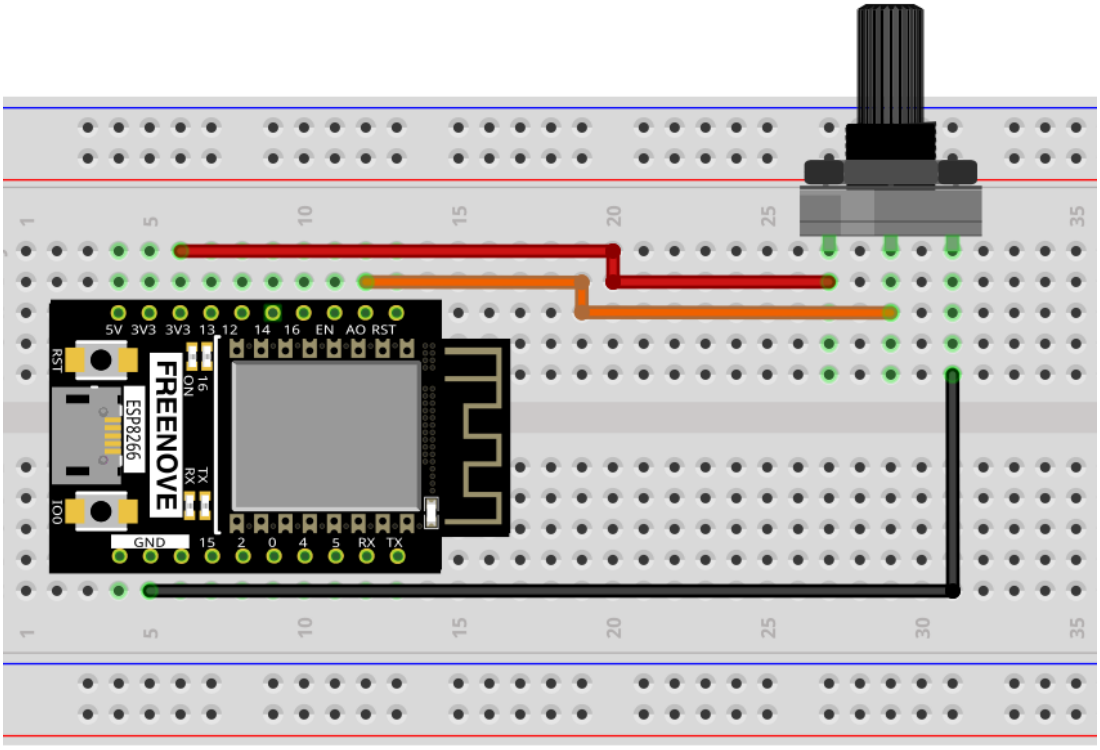
Circuit

Please note that the voltage range of the ADC is 0V to 3.3V.Exceeding this voltage range may cause permanent damage to your hardware!

Schematic diagram



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

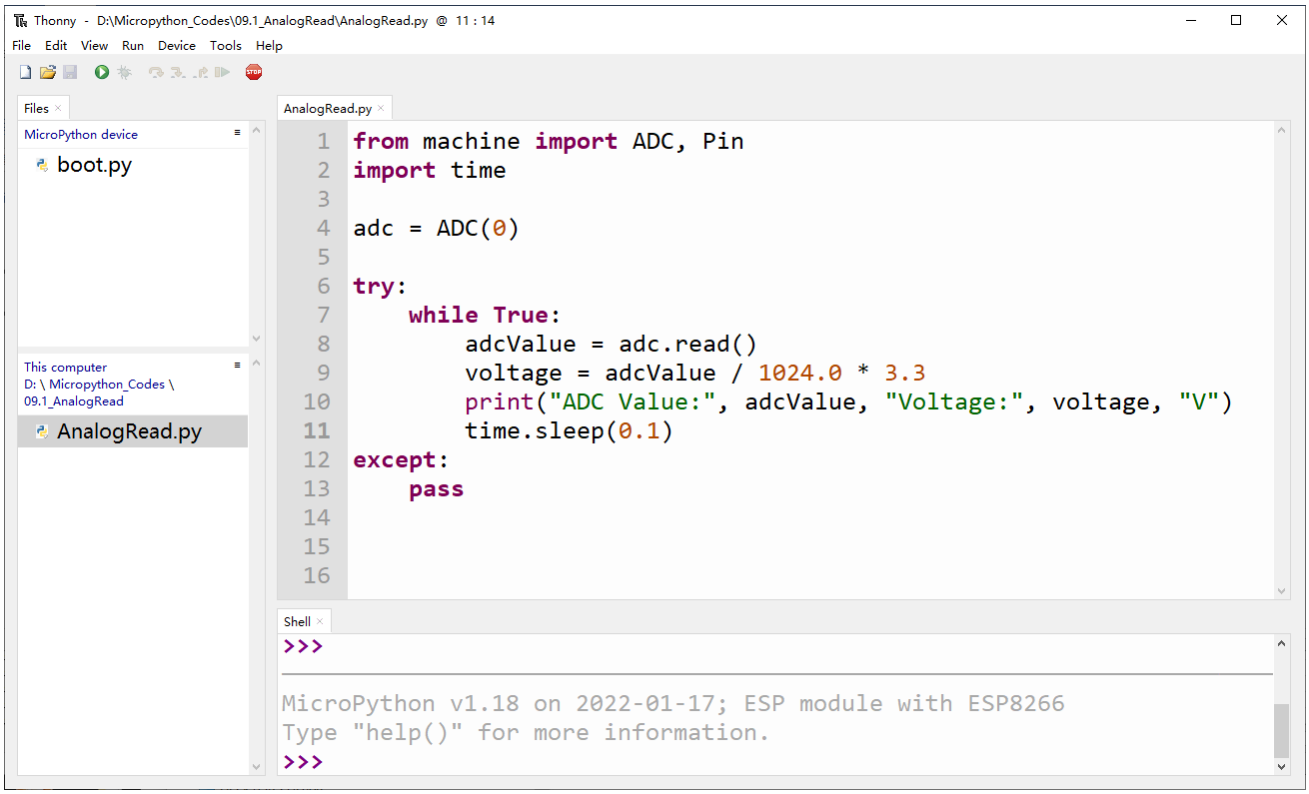


Code

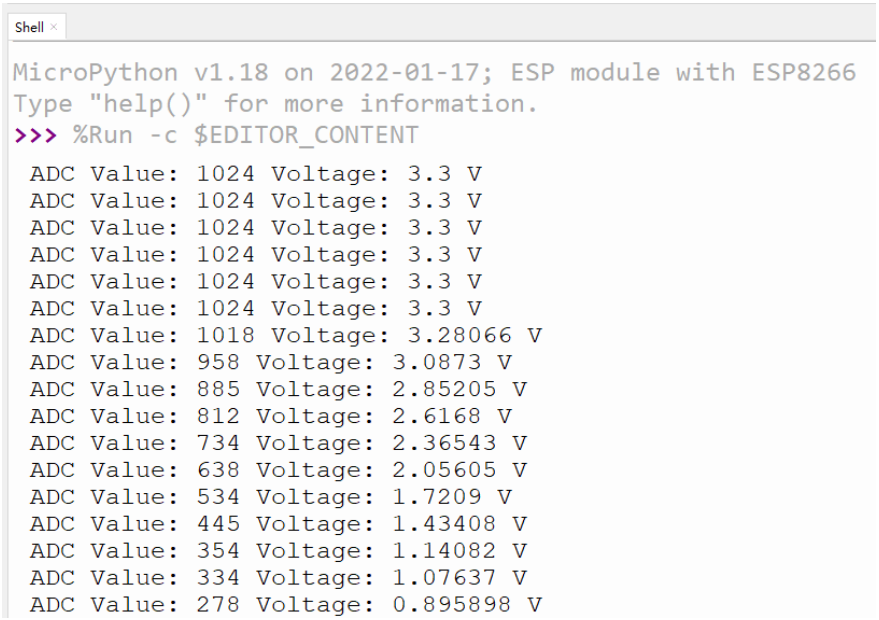
Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "09.1_AnalogRead and then click "AnalogRead.py".

09.1_AnalogRead



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



"Shell" prints ADC value and the output voltage of potentiometer and other information. From the code, we get the ADC value of pin A0, then convert it into voltage value.
Turn the rotary potentiometer shaft, and you can see the voltage change.
The following is the code:

```
1  from machine import ADC, Pin
2  import time
3
4  adc = ADC(0)
5  try:
6      while True:
7          adcValue = adc.read()
8          voltage = adcValue / 1024.0 * 3.3
9          print("ADC Value:", adcValue, "Voltage:", voltage, "V")
10         time.sleep(0.1)
11 except:
12     pass
```

Import Pin, ADC and DAC modules.

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

Read ADC value once every 100 millisecods, and "Shell" prints ADC value and the output voltage of potentiometer and other information.

```
7          adcValue = adc.read()
8          voltage = adcValue / 1024.0 * 3.3
9          print("ADC Value:", adcValue, "Voltage:", voltage, "V")
10         time.sleep(0.1)
```

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: ADC0.

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

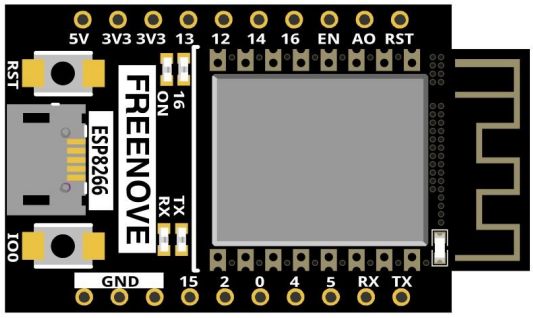
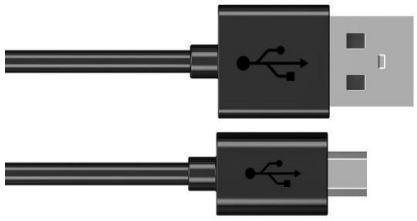
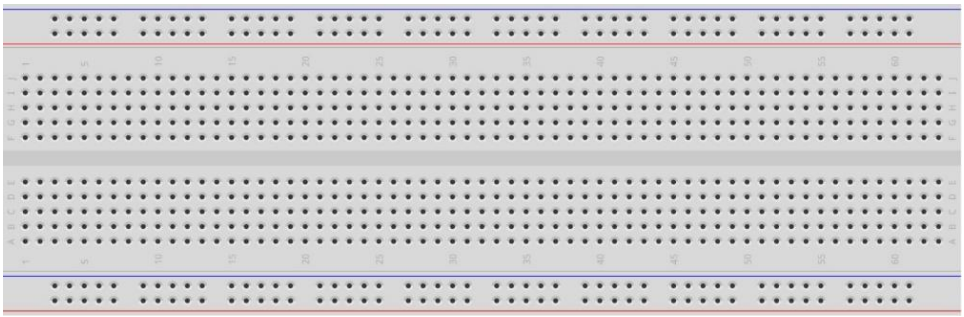
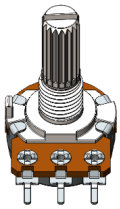
Chapter 9 Potentiometer & LED

In the previous section, we have finished reading ADC value and converting it into voltage. Now, we will try to use potentiometer to control the brightness of LED.

Project 9.1 Soft Light

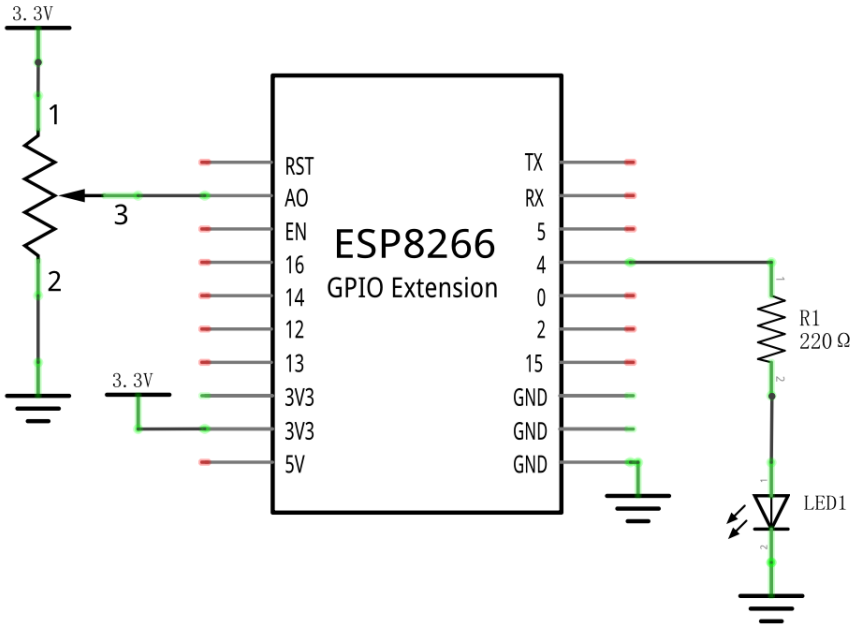
In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle of the PWM used to control the brightness of a LED. Then you can change the brightness of a LED by adjusting the potentiometer.

Component List

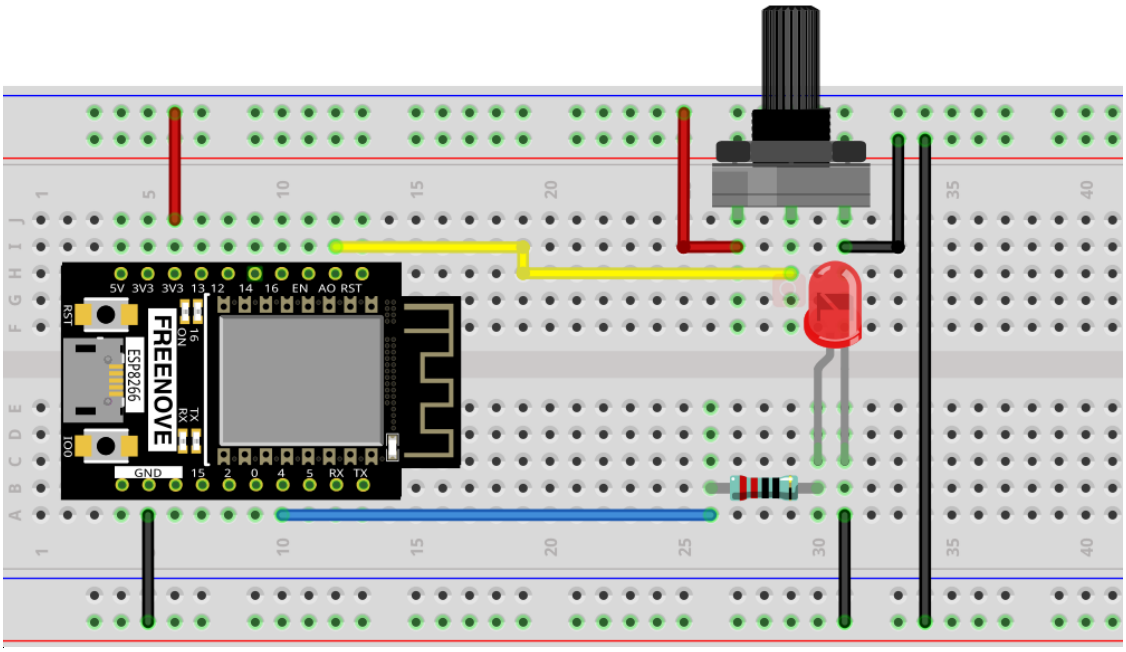
<table>
<tr><td colspan="2">ESP8266 x1
 
</td><td colspan="2">USB cable
  </td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>Rotary potentiometer x1
  </td><td>Resistor 220Ω x1
  </td><td>LED x1
  </td><td>Jumper wire M/M x8
</td></tr>
</table>

Circuit

Schematic diagram



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

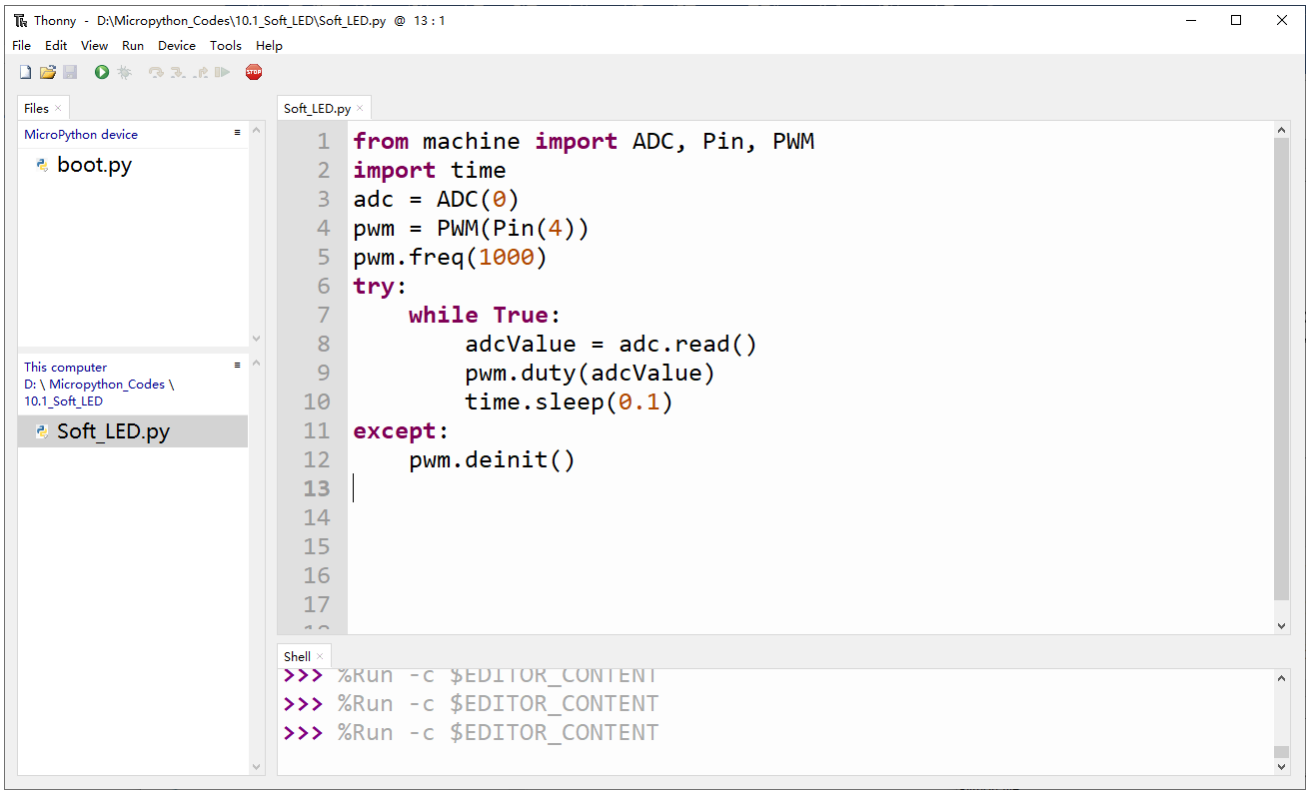


Code

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP8266/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:

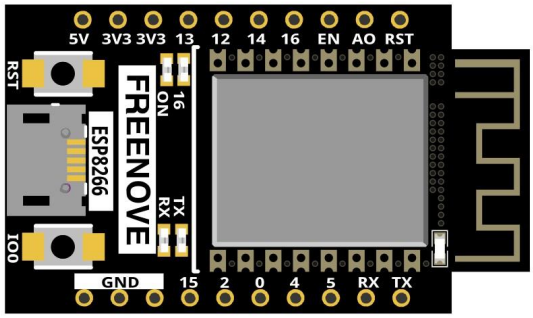
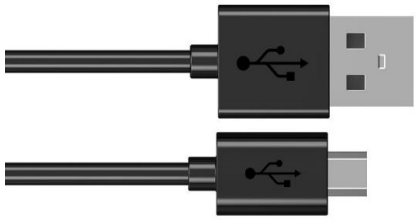
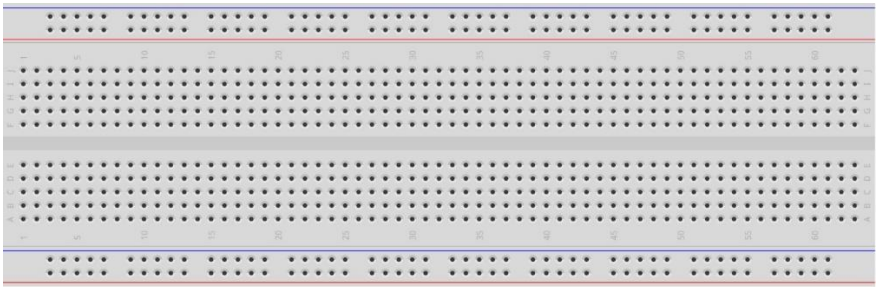
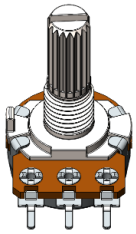
```
from machine import ADC, Pin, PWM
import time
adc = ADC(0)
pwm = PWM(Pin(4))
pwm.freq(1000)
try:
    while True:
        adcValue = adc.read()
        pwm.duty(adcValue)
        time.sleep(0.1)
except:
    pwm.deinit()
```

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

Project 9.2 Color Light

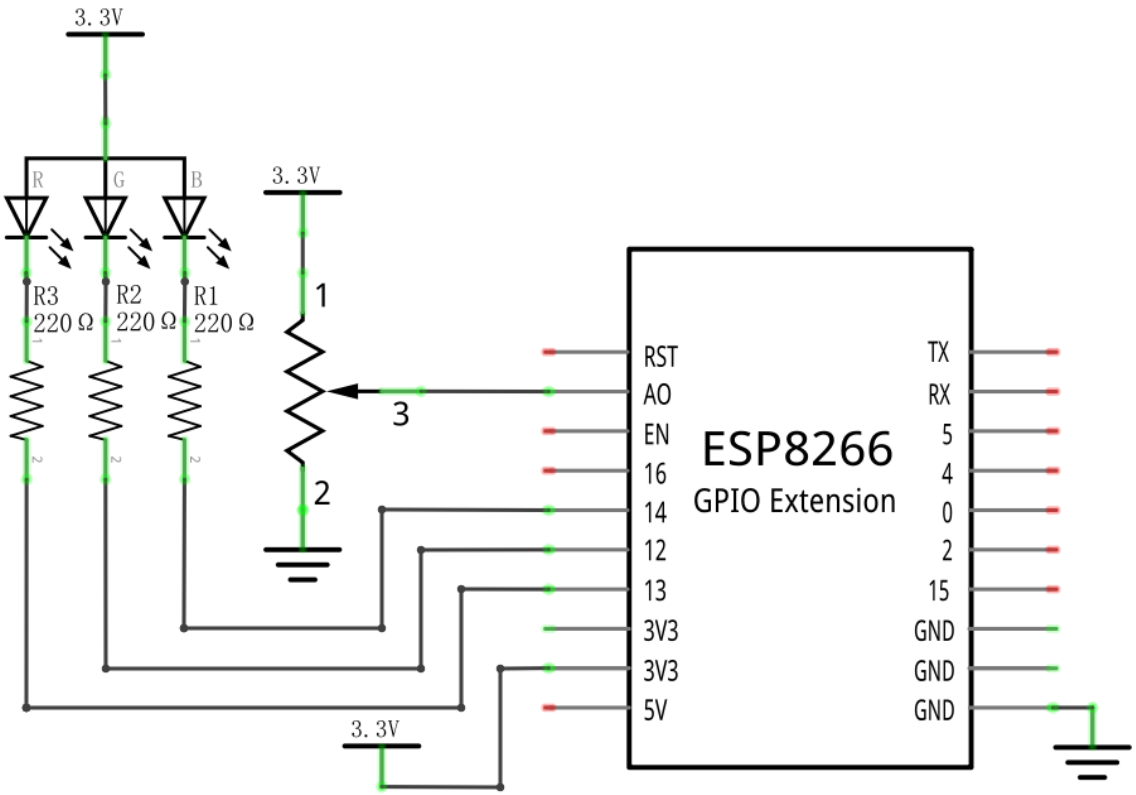
In this project, a potentiometer is used to control the RGB LED. The RGB LED is bright red when the potentiometer is near the midpoint, green when the potentiometer rotates to the "left" and blue when the potentiometer rotates to the "right".

Component List

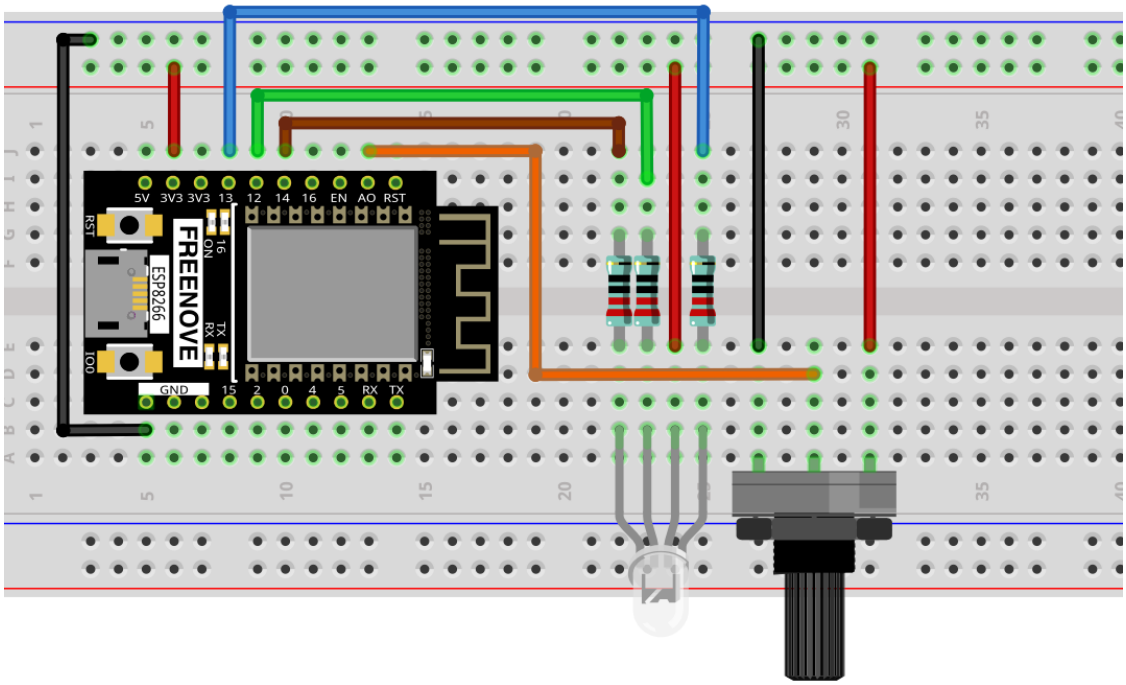
<table>
<tr><td>ESP8266 x1
  </td><td>USB cable
  </td></tr>
<tr><td colspan="2">Breadboard x1
  </td></tr>
<tr><td>Rotary potentiometer x1
  </td><td>Resistor 220Ω x3
  </td><td>RGBLED x1
  </td><td>Jumper wire M/M x9
  </td></tr>
</table>

Circuit

Schematic diagram



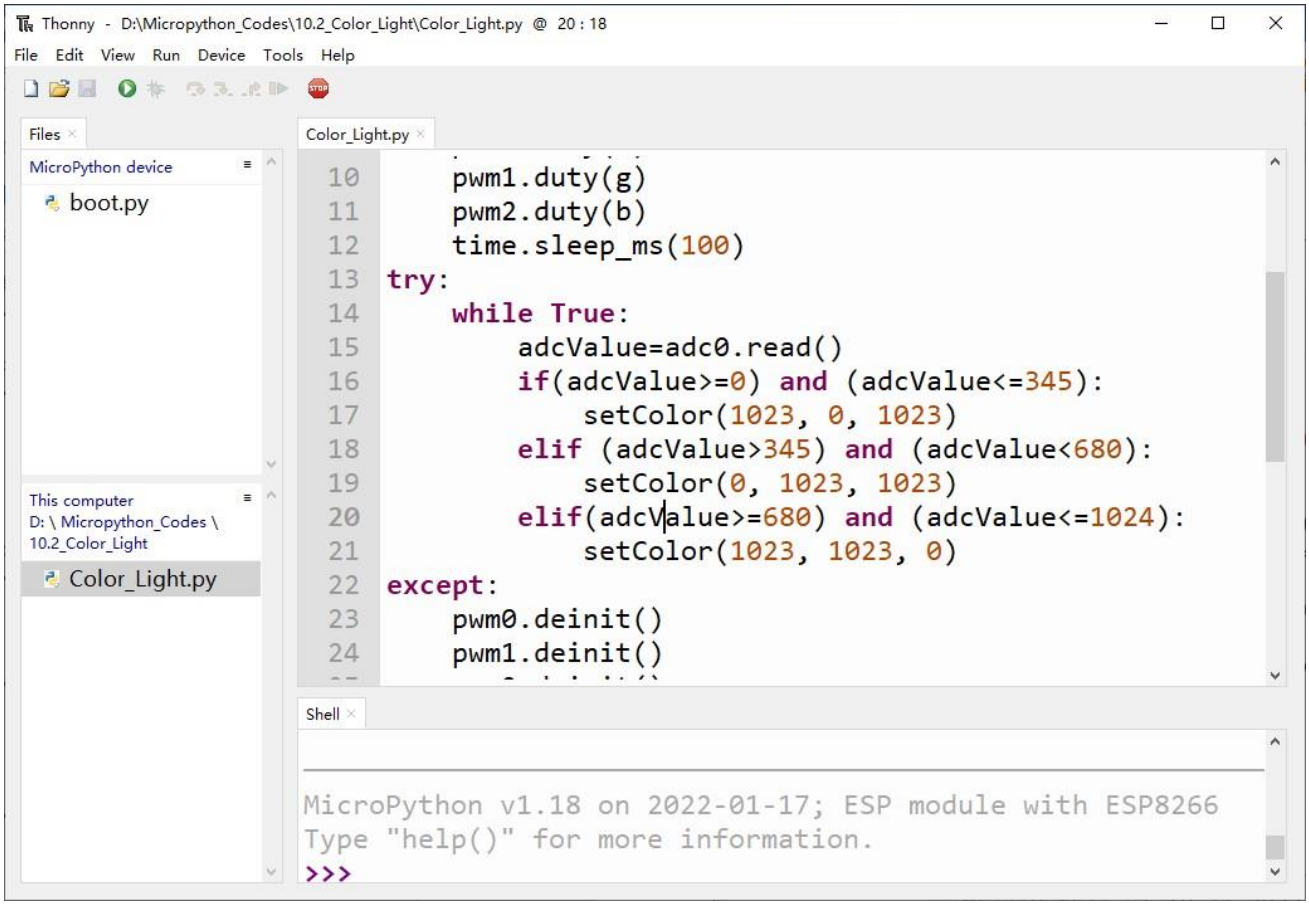
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "10.2_Color_Light" and double click "Color_Light.py".

10.2_Color_Light



Download the code to ESP8266, rotate the potentiometers, then the color of RGB LED will change.

The following is the program code:

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

pwm0=PWM(Pin(13,Pin.OUT),1000)
pwm1=PWM(Pin(14,Pin.OUT),1000)
pwm2=PWM(Pin(12,Pin.OUT),1000)
adc0=ADC(0)
def setColor(r, g, b):
    pwm0.duty(r)
    pwm1.duty(g)
    pwm2.duty(b)
    time.sleep_ms(100)
try:
```

```
14      while True:
15          adcValue=adc0.read()
16          if(adcValue>=0) and (adcValue<=345):
17              setColor(1023, 0, 1023)
18          elif (adcValue>345) and (adcValue<680):
19              setColor(0, 1023, 1023)
20          elif(adcValue>=680) and (adcValue<=1024):
21              setColor(1023, 1023, 0)
22  except:
23      pwm0.deinit()
24      pwm1.deinit()
25      pwm2.deinit()
```

In the code, you can read the potentiometer ADC value, judge the range of ADC value, to control the RGB LED color.

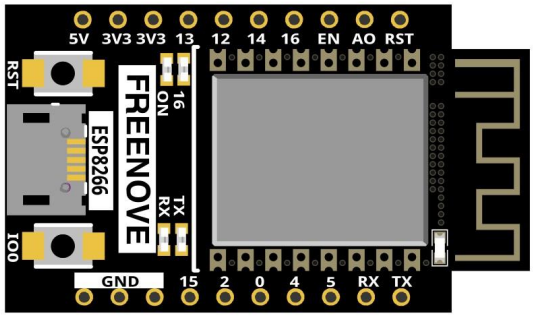
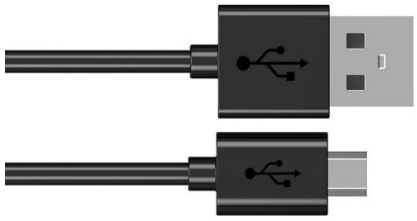
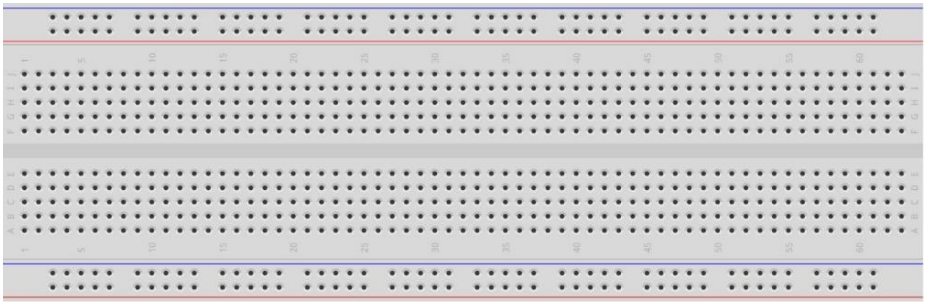
Chapter 10 Photoresistor & LED

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

Project 10.1 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.

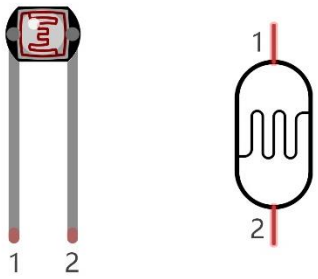
Component List

<table>
<tr><td colspan="2">ESP8266 x1
  </td><td colspan="2">USB cable
  </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 wire M/M x7
  </td></tr>
</table>

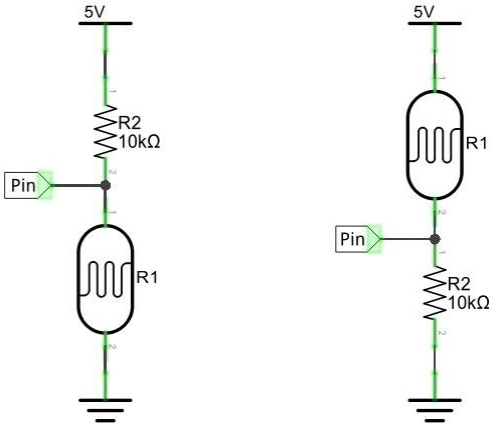
Component knowledge

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:

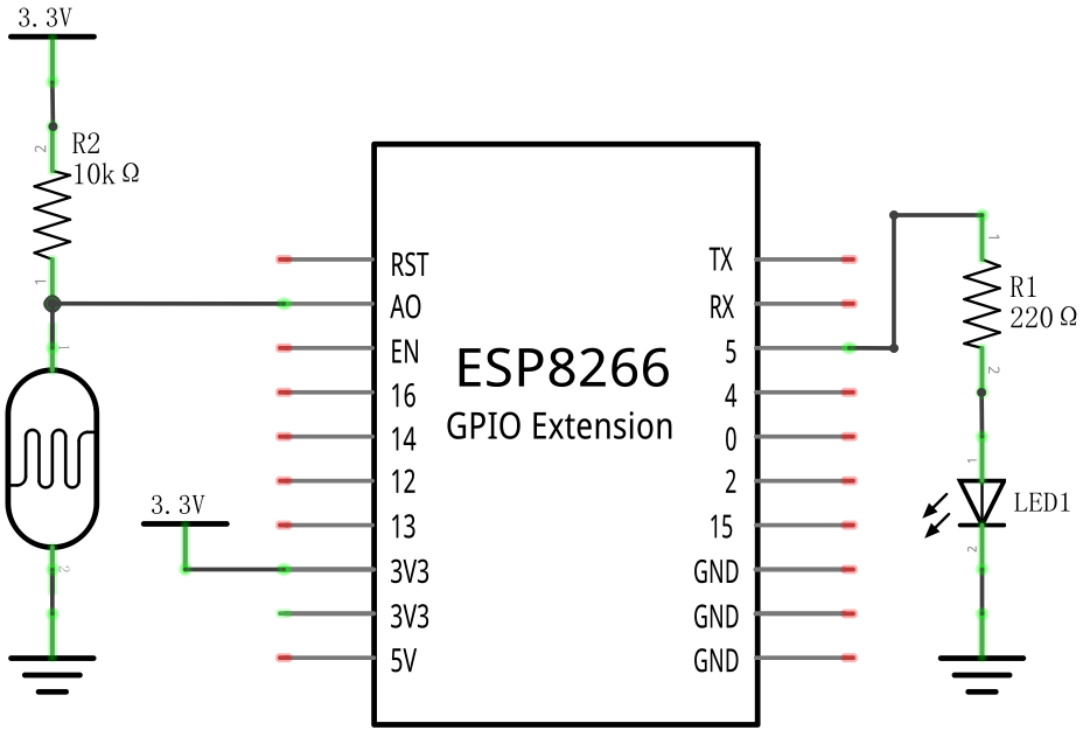


In the above circuit, when a Photoresistor's resistance vale changes due to a change in light intensity, the voltage between the Photoresistor and Resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

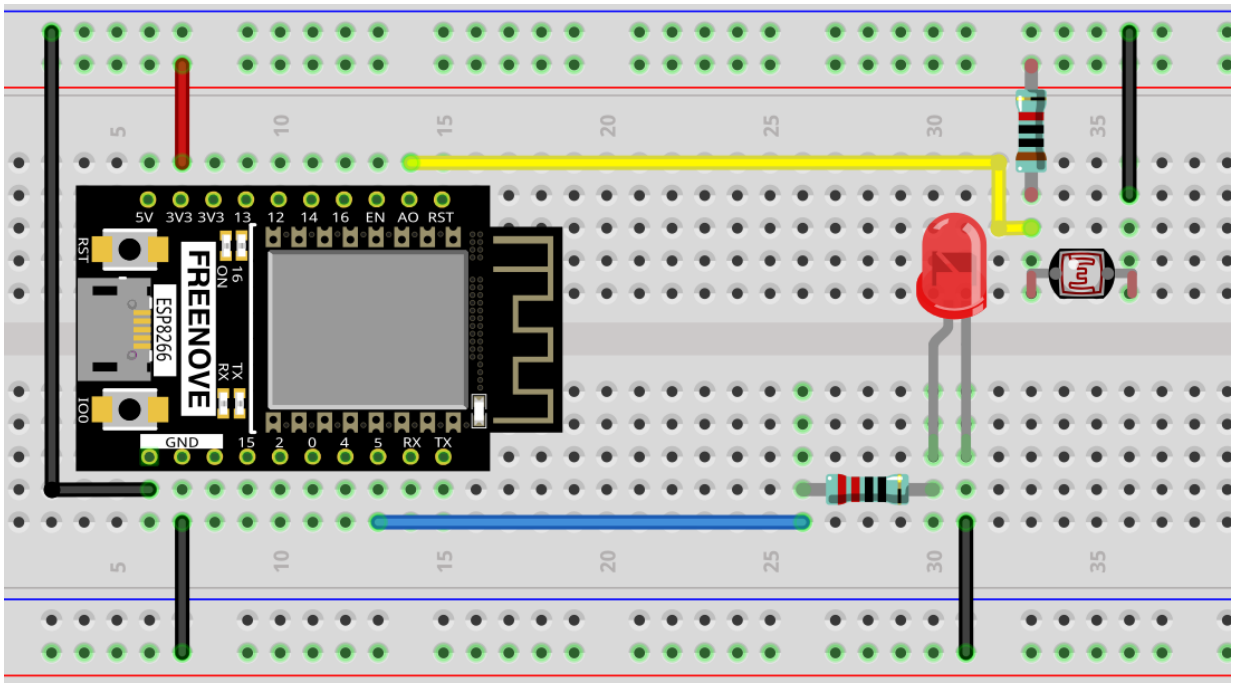
Circuit

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

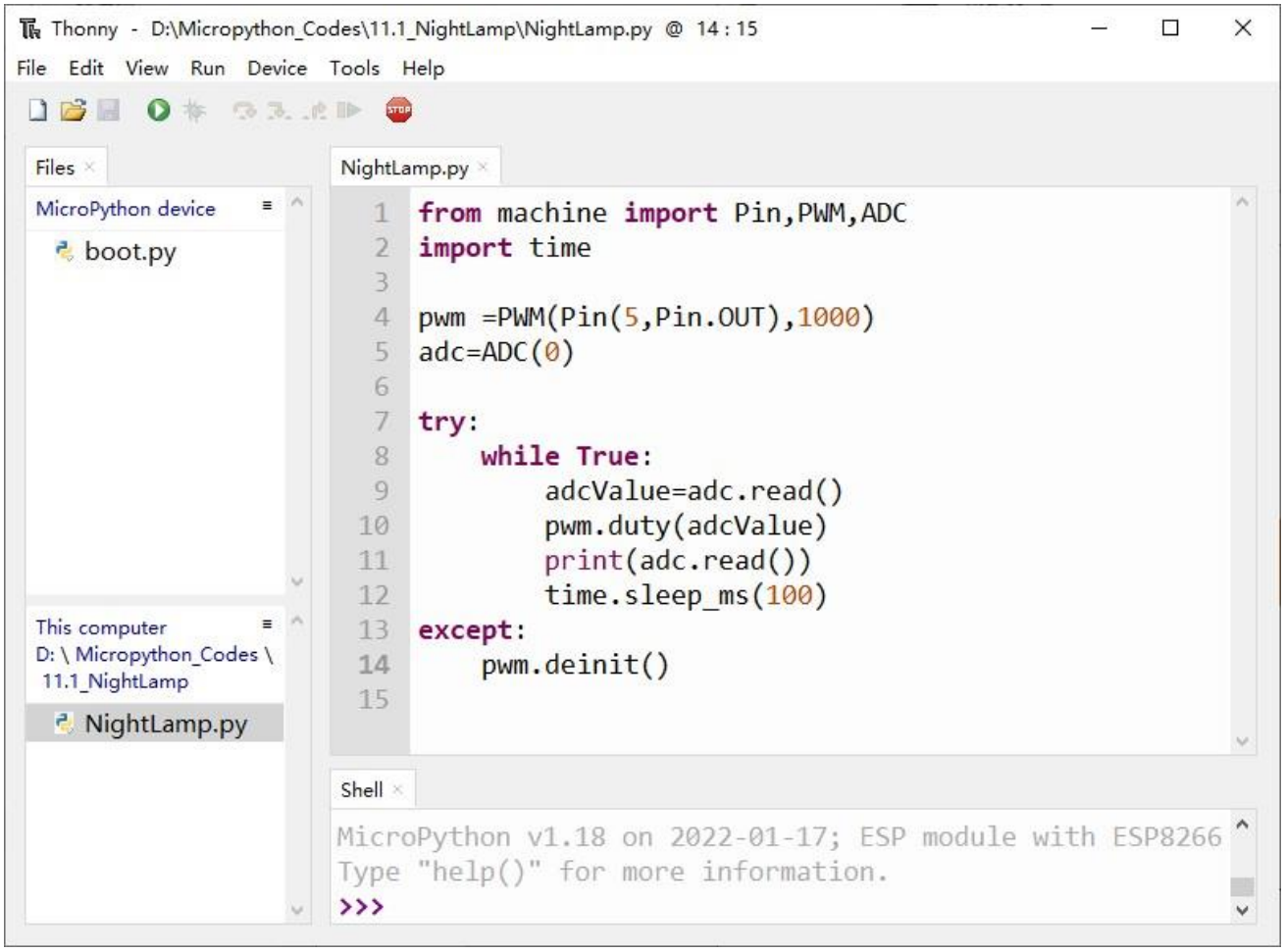


Code

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP8266/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(5,Pin.OUT),1000)
adc=ADC(0)
try:
    while True:
        adcValue=adc.read()
        pwm.duty(adcValue)
        print(adc.read())
```

```
11          time.sleep_ms(100)
12  except:
13      pwm.deinit()
```

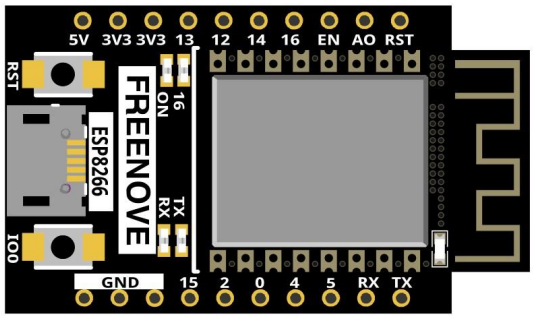
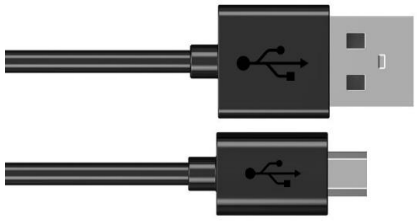
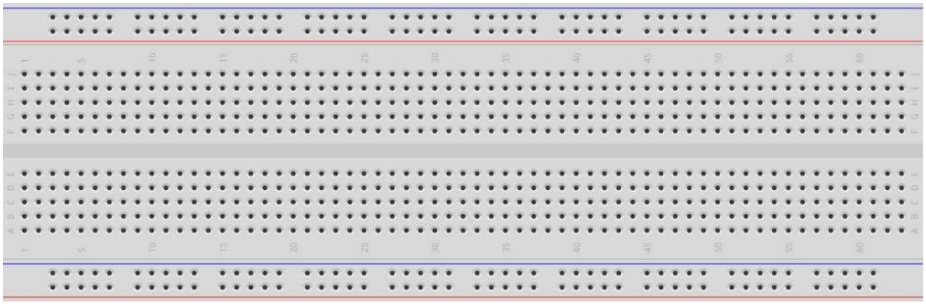
Chapter 11 Thermistor

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

Project 11.1 Thermometer

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

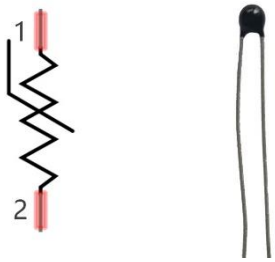
Component List

<table>
<tr><td>ESP8266 x1
  </td><td>USB cable
  </td></tr>
<tr><td colspan="2">Breadboard x1
  </td></tr>
<tr><td>Thermistor x1
  </td><td>Resistor 1kΩ x1
  </td><td>Jumper wire M/M x4
  </td></tr>
</table>

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;

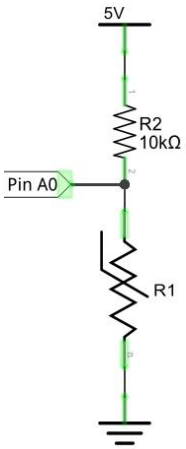
EXP[n] is nth power of e;

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.

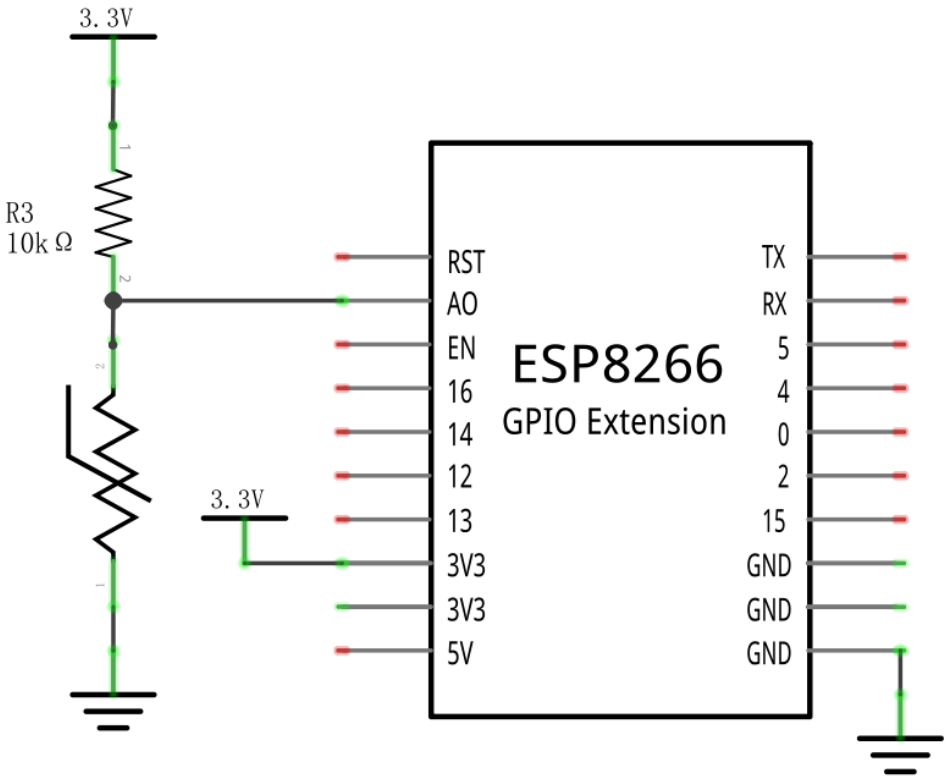
Therefore, the temperature formula can be derived as:

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

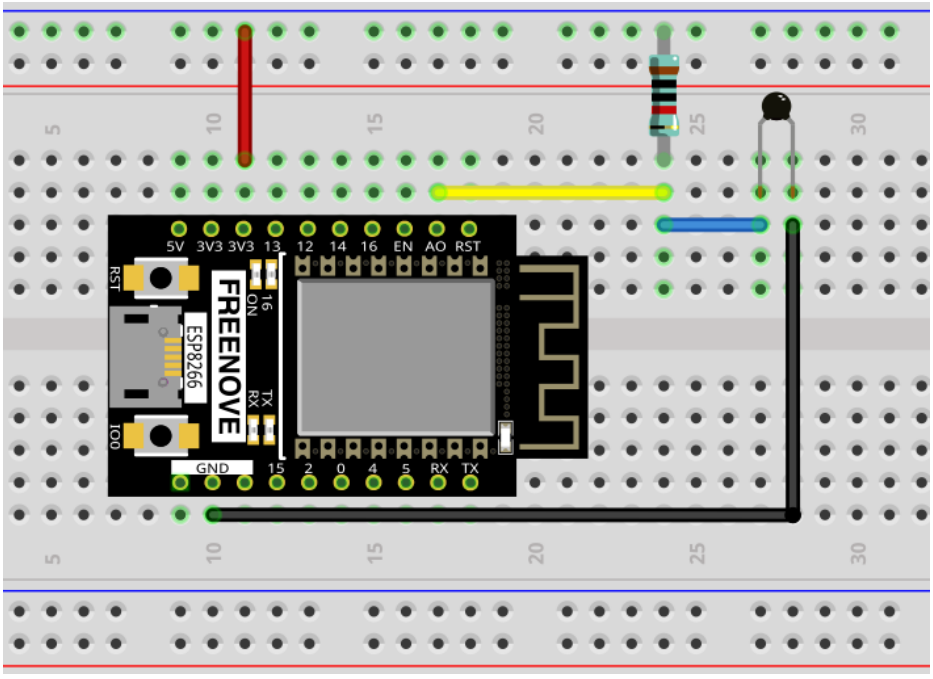
Circuit

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

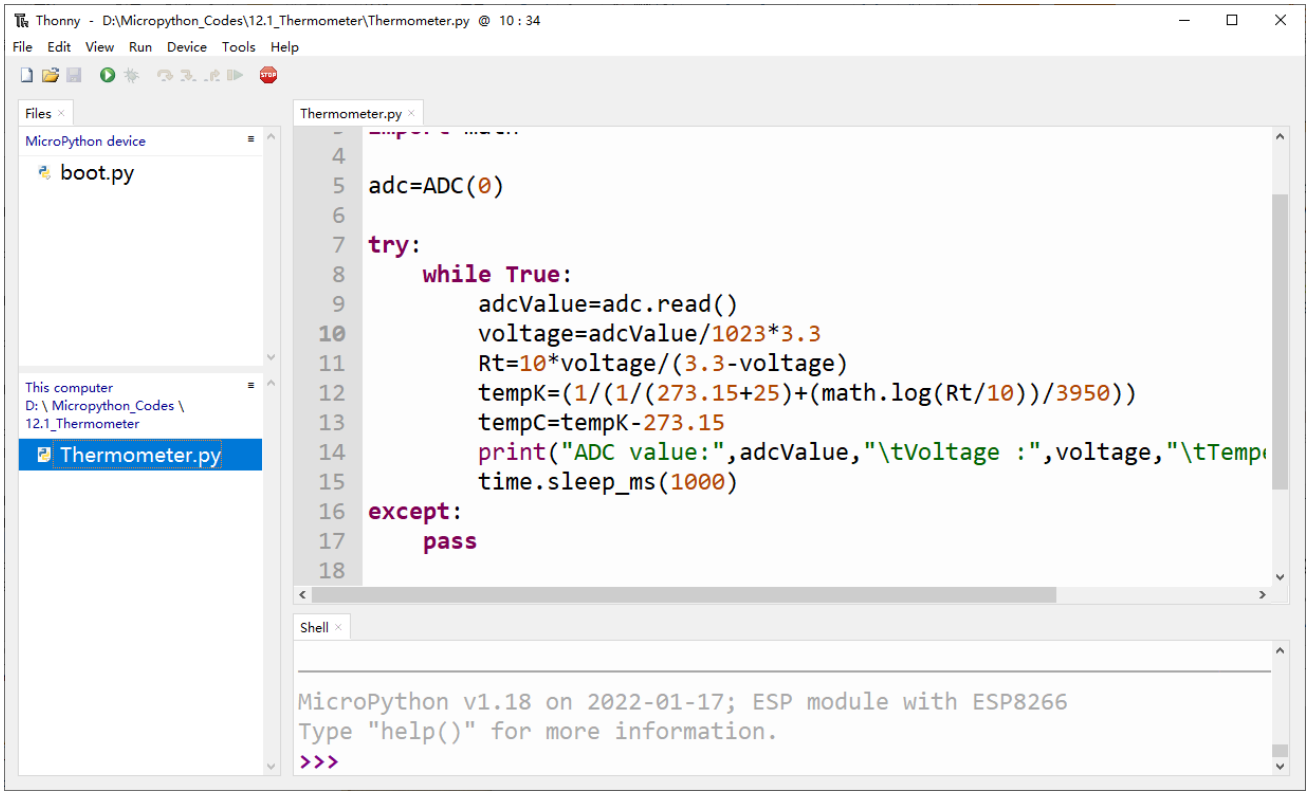


Code

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

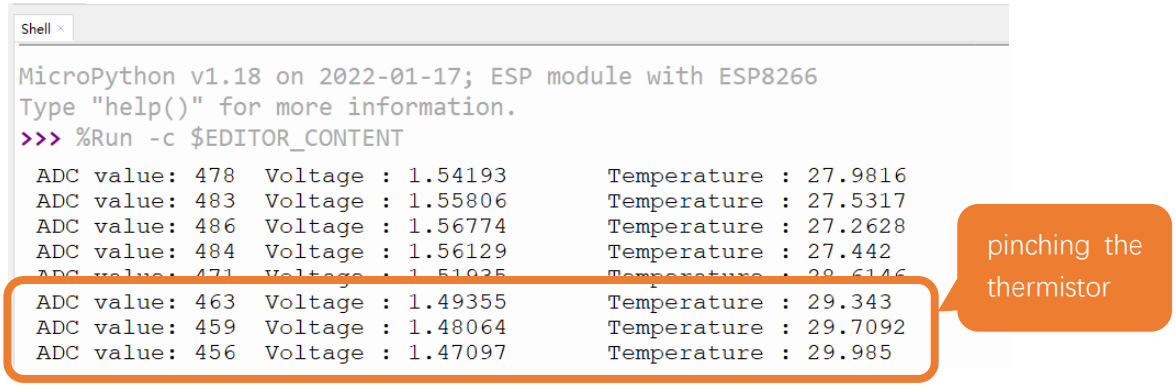
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "12.1_Thermometer" and double click "Thermometer.py".

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



The following is the code:

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

adc=ADC(0)

try:
    while True:
        adcValue=adc.read()
        voltage=adcValue/1023*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.

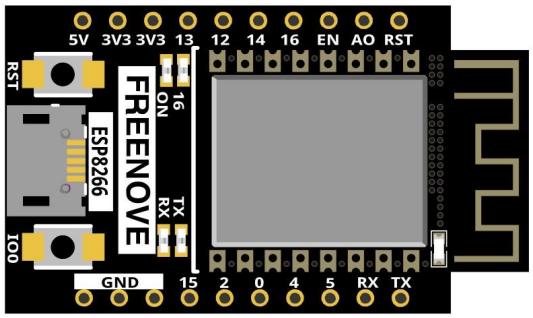
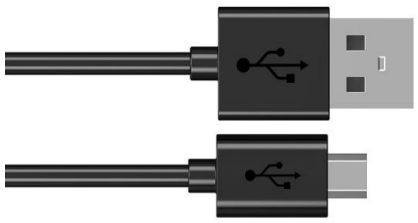
Chapter 12 WiFi Working Modes

In this chapter, we'll focus on the WiFi infrastructure for ESP8266.

ESP8266 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 12.1 Station mode

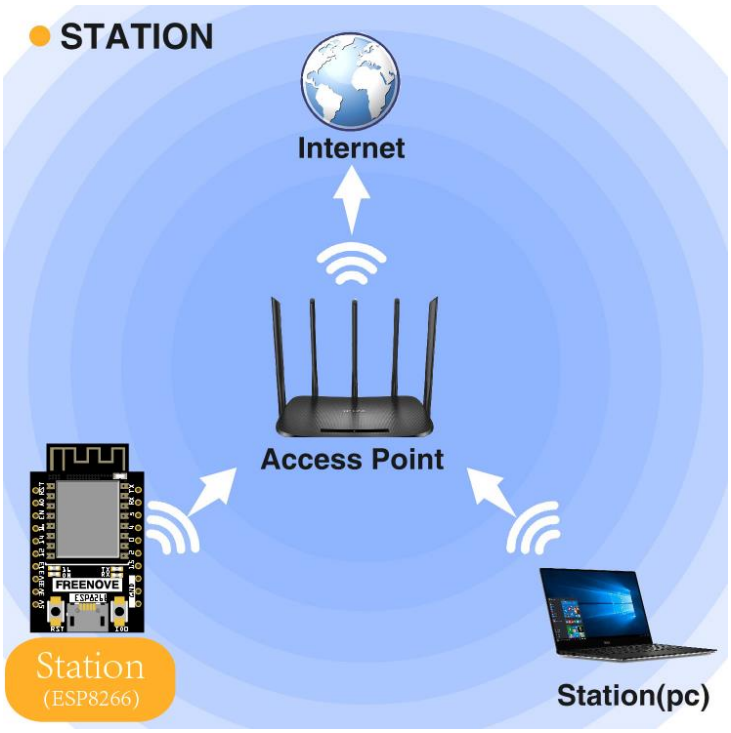
Component List

ESP8266 x1 	USB cable 

Component knowledge

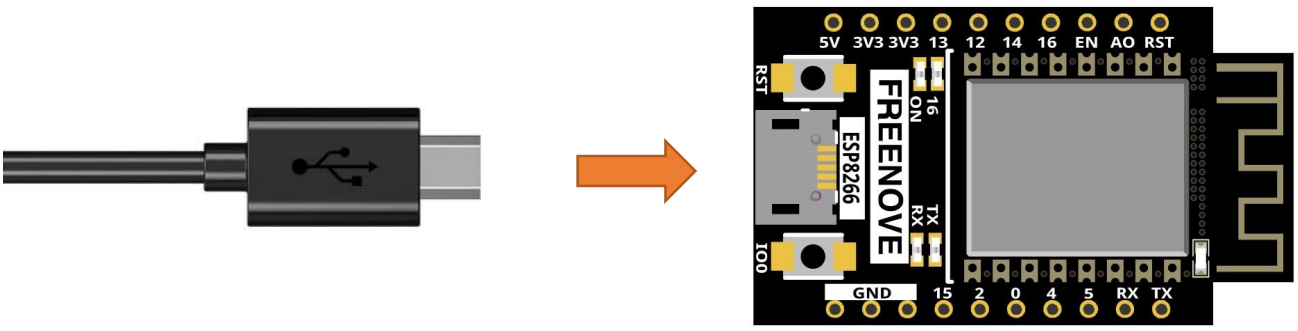
Station mode

When ESP8266 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 ESP8266 wants to communicate with the PC, it needs to be connected to the router.



Circuit

Connect Freenove ESP8266 to the computer using the USB cable.

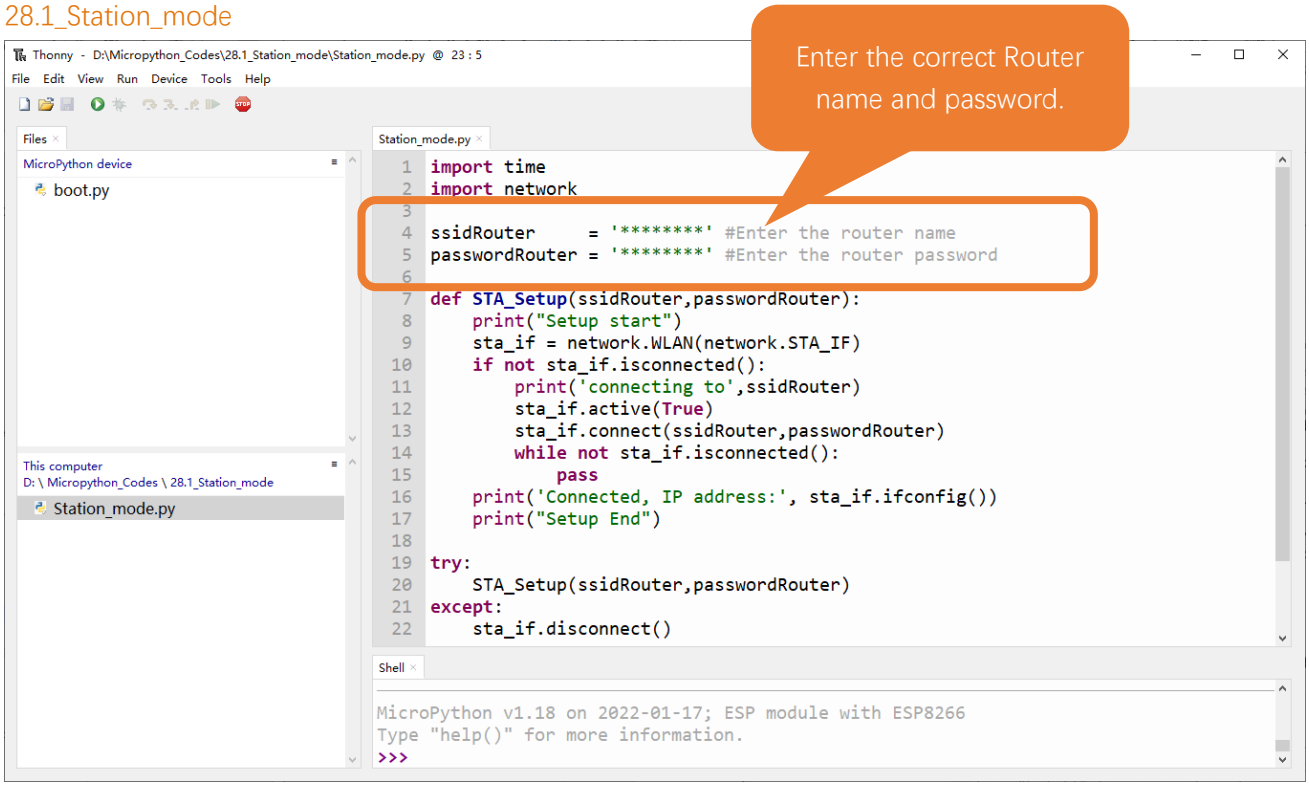


Code

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "28.1_Station_mode" and double click "Station_mode.py".

28.1_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 ESP8266, wait for ESP8266 to connect to your router and print the IP address assigned by the router to ESP8266 in "Shell".

```
Shell
MicroPython v1.18 on 2022-01-17; ESP module with ESP8266
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
  Setup start
  Connected, IP address: ('192.168.1.113', '255.255.255.0', '192.168.1.1', '8.8.8.8')
  Setup End

>>>
```

The following is the program code:

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

Import network module.

```
2  import network
```

Enter correct router name and password.

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

Set ESP8266 in Station mode.

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

Activate ESP8288 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 ESP8266 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 ESP8266 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 ESP8266 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 12.2 AP mode

Component List & Circuit

Component List & Circuit are the same as in Section 28.1.

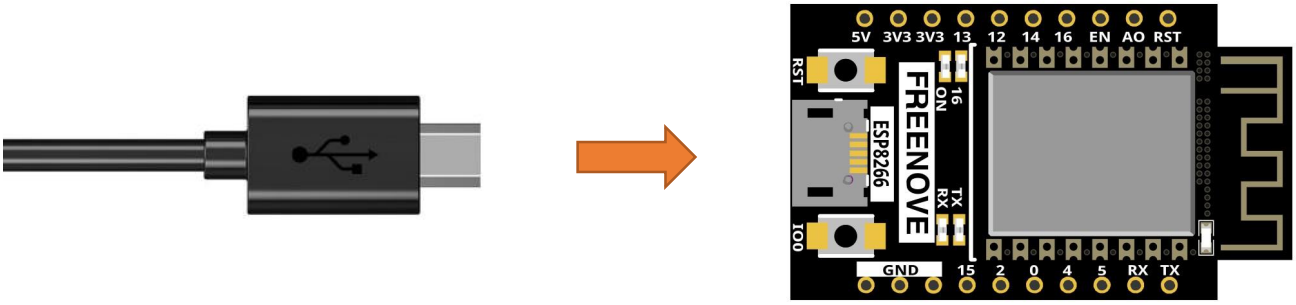
Component knowledge

AP mode

When ESP8266 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, ESP8266 is used as a hotspot. If a mobile phone or PC wants to communicate with ESP8266, it must be connected to the hotspot of ESP8266. Only after a connection is established with ESP8266 can they communicate.

Circuit

Connect Freenove ESP8266 to the computer using the USB cable.

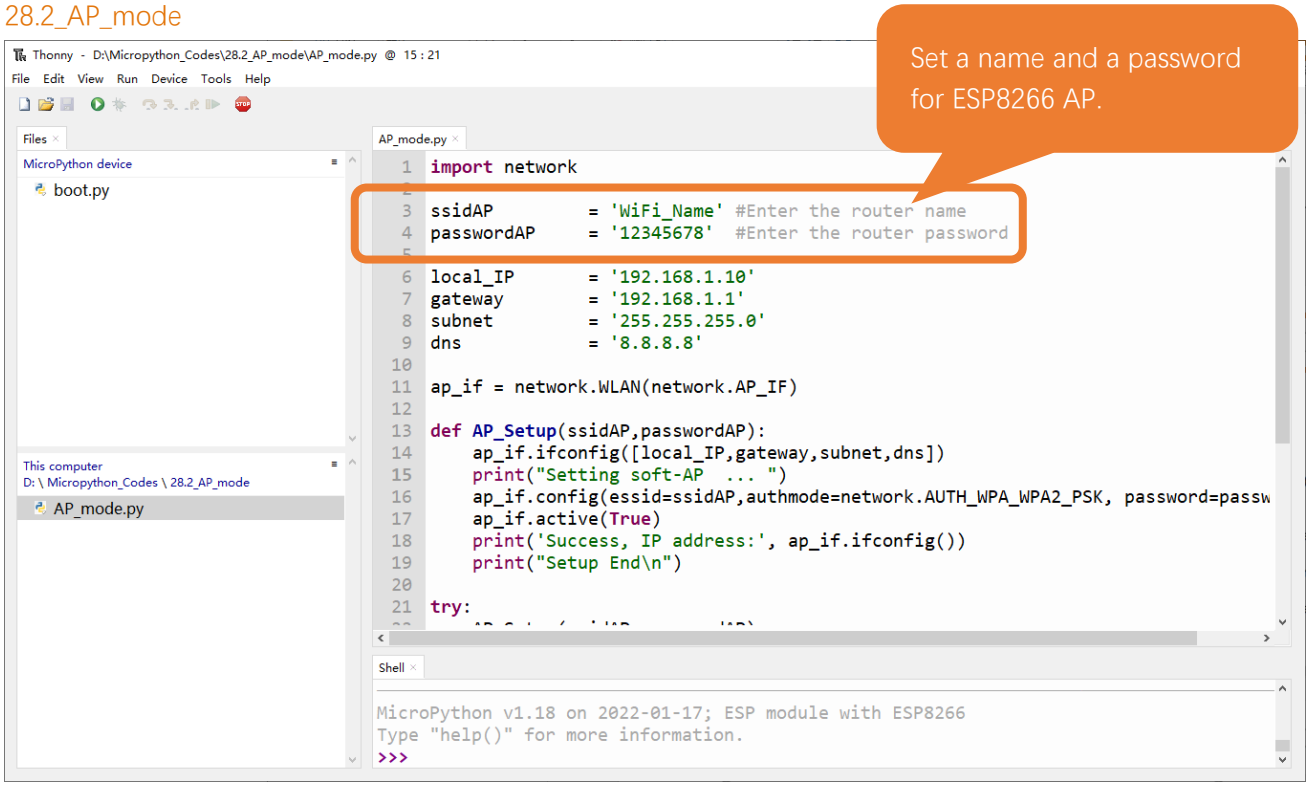


Code

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

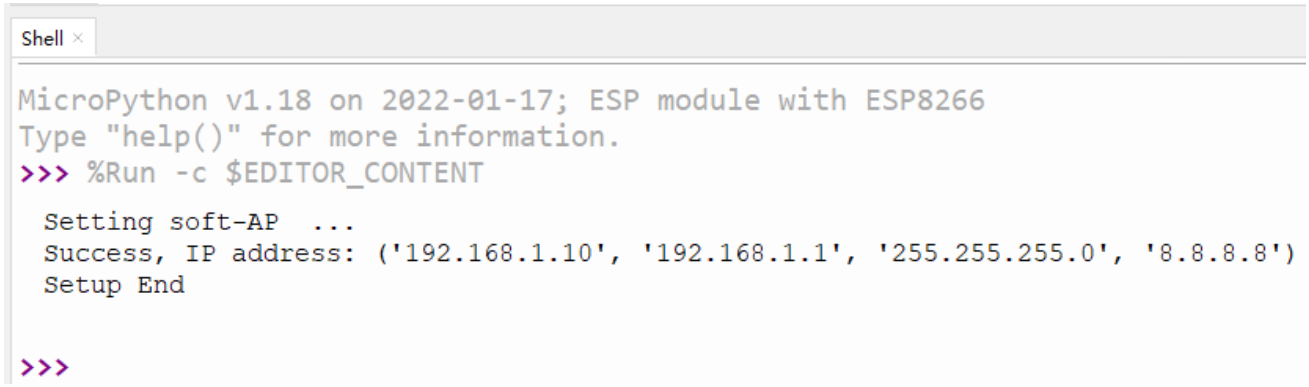
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "28.2_AP_mode". and double click "AP_mode.py".

28.2_AP_mode

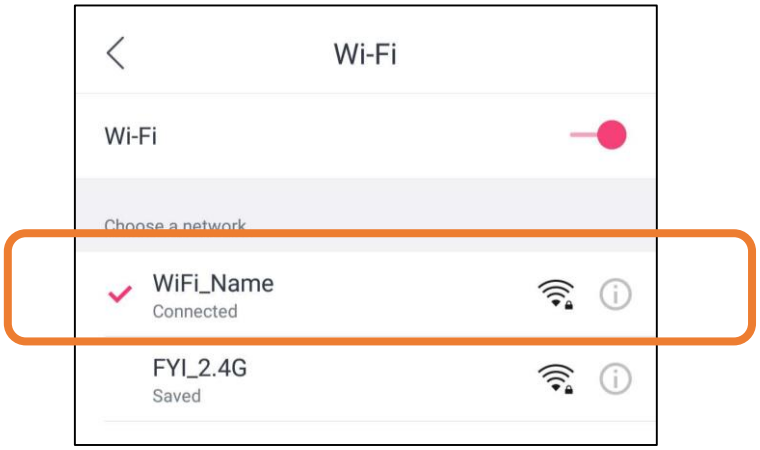


Before the Code runs, you can make any changes to the AP name and password for ESP8266 in the box as shown in the illustration above. Of course, you can leave it alone by default.

Click "Run current script", open the AP function of ESP8266 and print the access point information.



Turn on the WiFi scanning function of your phone, and you can see the ssid_AP on ESP8266, 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 ESP8266 in AP mode.

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

Configure IP address, gateway and subnet mask for ESP8266.

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

Turn on an AP in ESP8266, 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

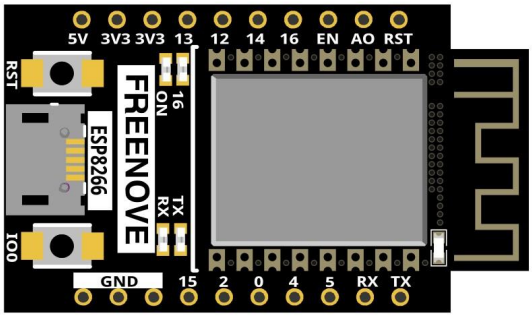
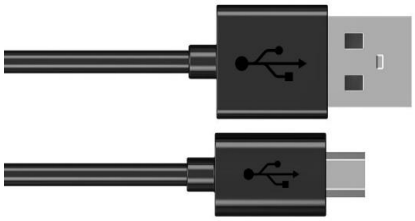
DNS_server: DNSserver

disconnect(): Disconnect from the currently connected wireless network

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

Project 12.3 AP+Station mode

Component List

ESP8266 x1 	Micro USB Wire x1 

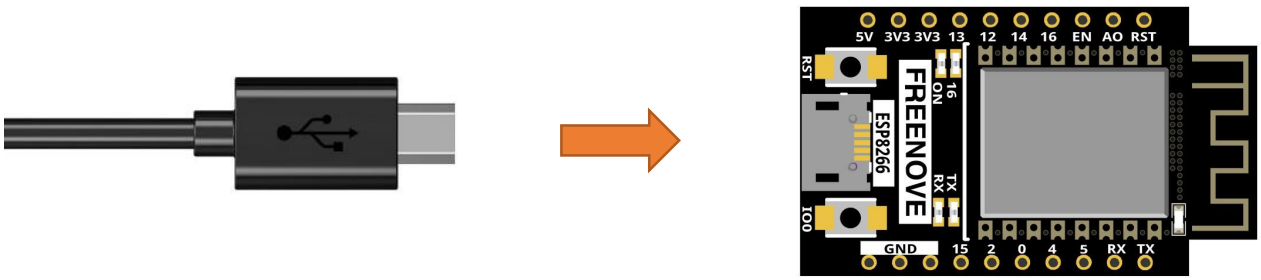
Component knowledge

AP+Station mode

In addition to AP mode and Station mode, ESP8266 can also use AP mode and Station mode at the same time. This mode contains the functions of the previous two modes. Turn on ESP8266'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 ESP8266.

Circuit

Connect Freenove ESP8266 to the computer using the USB cable.

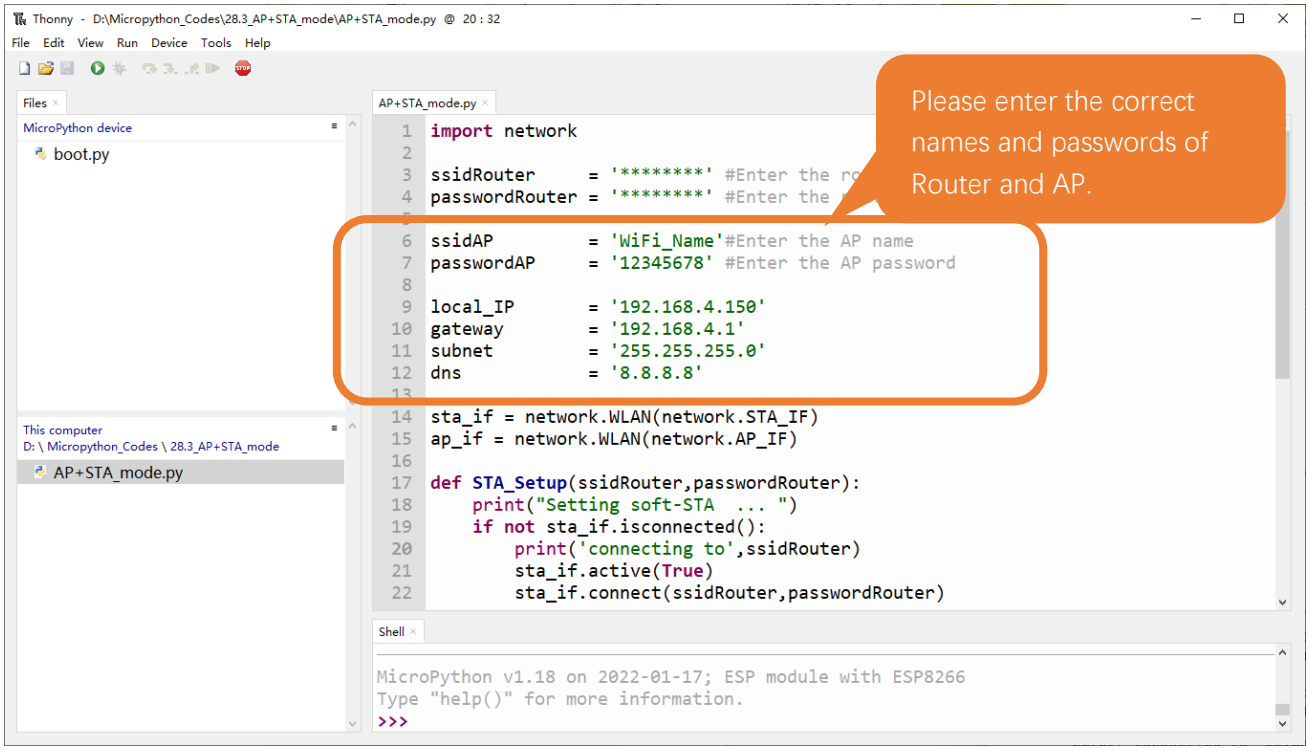


Code

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "28.3_AP+STA_mode"and double click "AP+STA_mode.py".

28.3_AP+STA_mode



It is analogous to Project 12.1 and Project 12.2. Before running the Code, you need to modify ssidRouter, passwordRouter, ssidAP and passwordAP shown in the box of the illustration above.

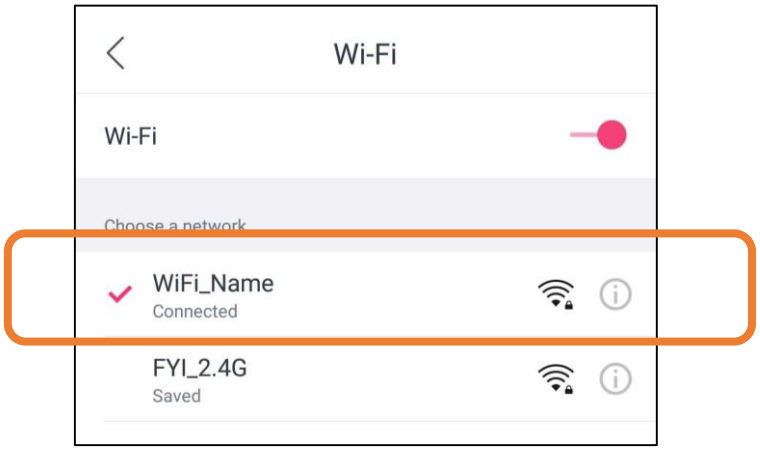
After making sure that the code is modified correctly, click "Run current script" and the "Shell" will display as follows:

```
Shell
MicroPython v1.18 on 2022-01-17; ESP module with ESP8266
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: ('192.168.1.113', '255.255.255.0', '192.168.1.1', '8.8.8.8')
  Setup End

>>>
```

Turn on the WiFi scanning function of your phone, and you can see the ssidAP on ESP8266.



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()
```

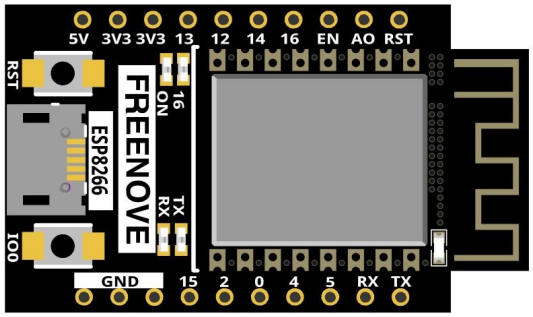
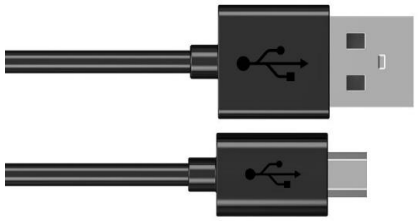
Chapter 13 TCP/IP

In this chapter, we wil introduce how ESP8266 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 13.1 As Client

In this section, ESP8266 is used as Client to connect Server on the same LAN and communicate with it.

Component List

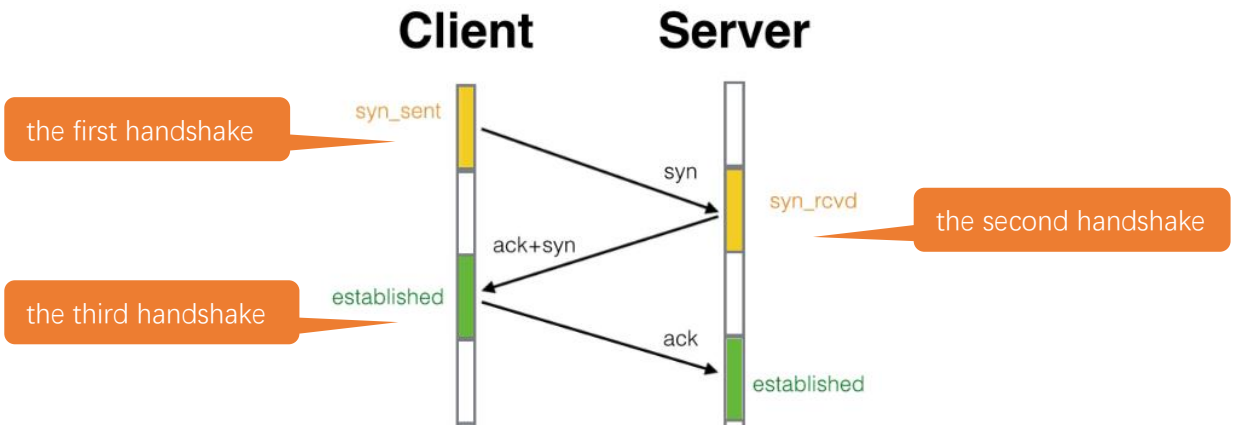
ESP8266 x1 	USB cable 

Component knowledge

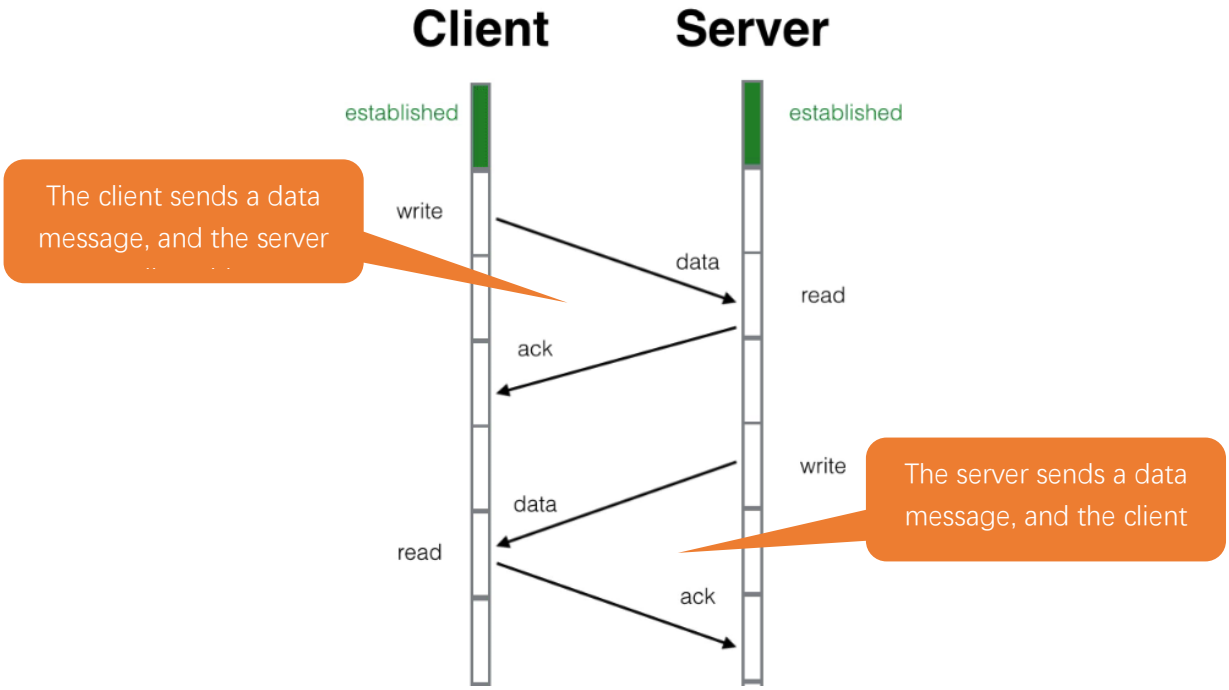
TCP connection

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.

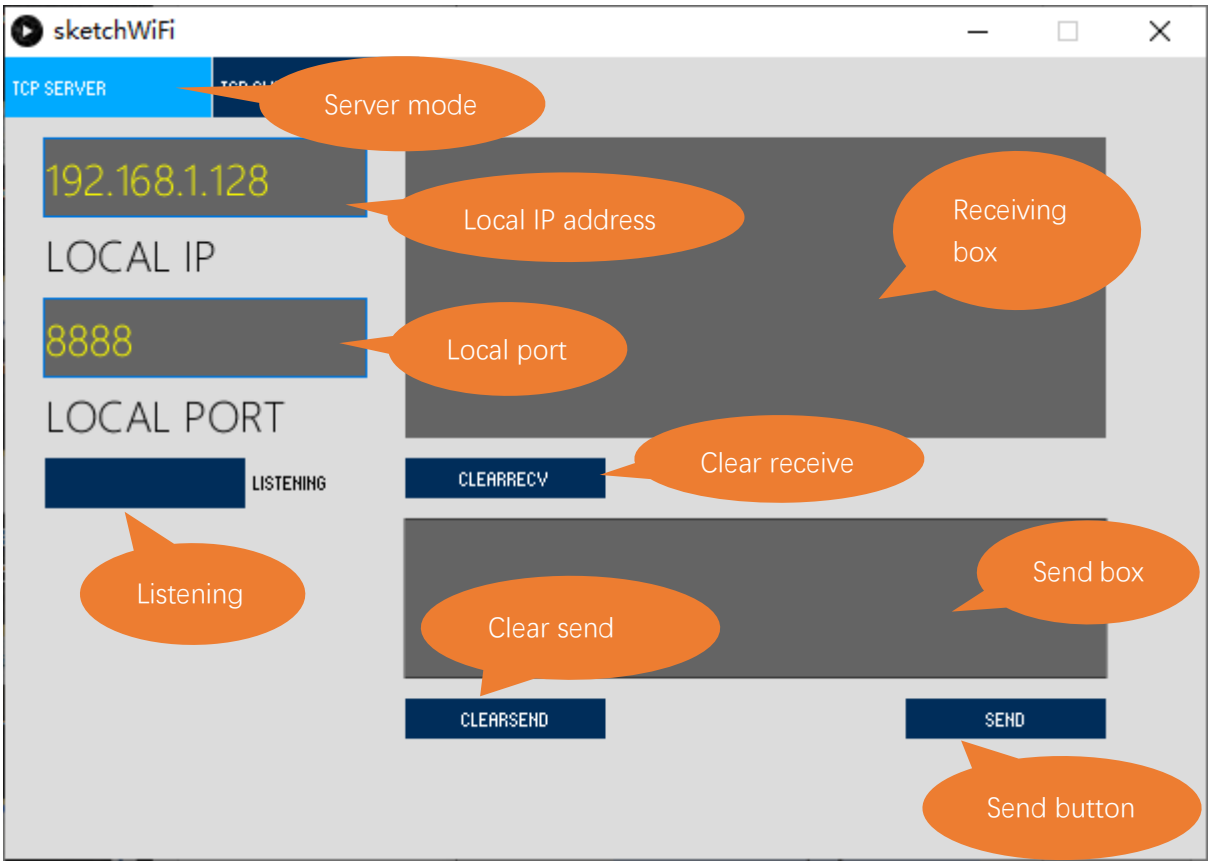
If you've not installed Processing, you can download it by clicking https://processing.org/download/. You can choose an appropriate version to download according to your PC system.

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

Use Server mode for communication

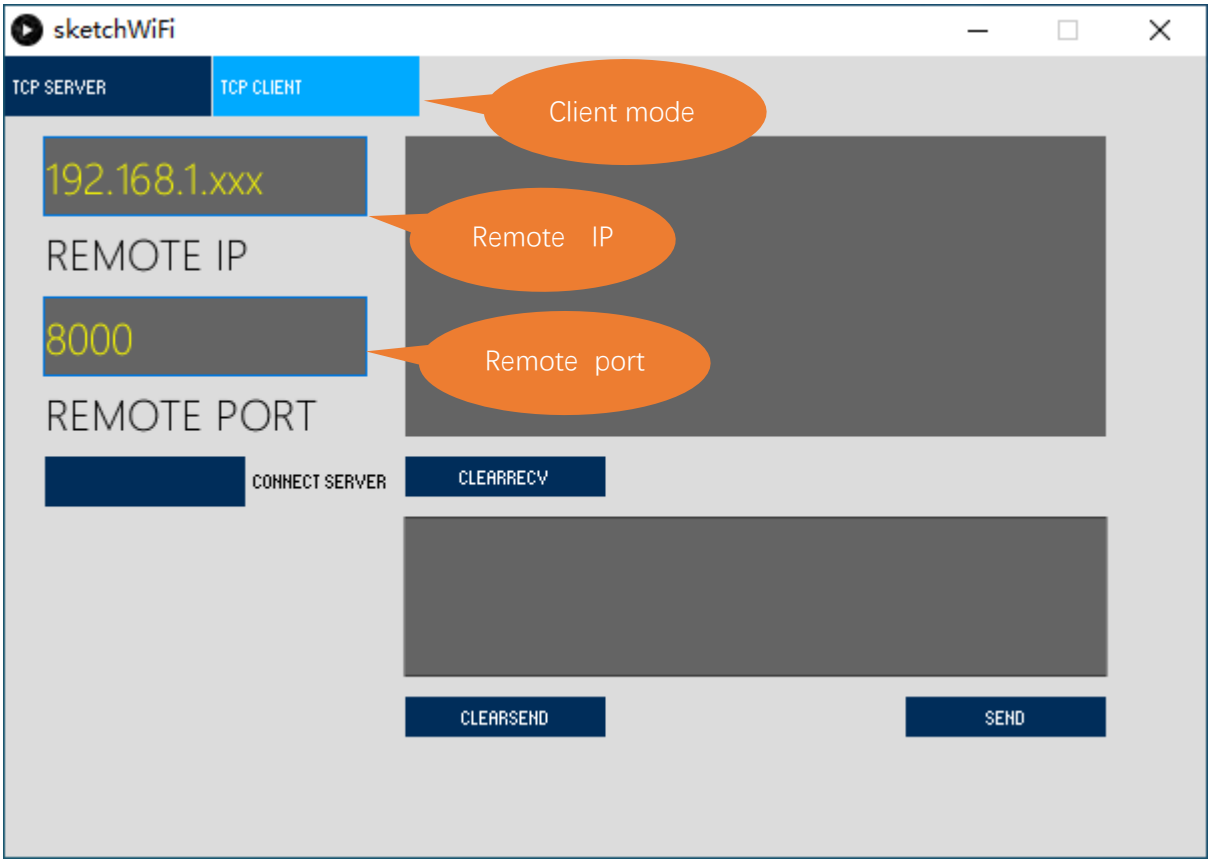
Open the "**Freenove_Basic_Starter_Kit_for_ESP8266/Codes/Micropython_Codes/29.1_TCP_as_Client/sketchWiFi/sketchWiFi.pde**". Click "Run".

The new pop-up interface is as follows. If ESP8266 is used as Client, select TCP SERVER mode for sketchWiFi.



When sketchWiFi selects TCP SERVER mode, ESP8266 Code needs to be changed according to sketchWiFi's displaying of LOCAL IP or LOCAL PORT.

If ESP8266 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.

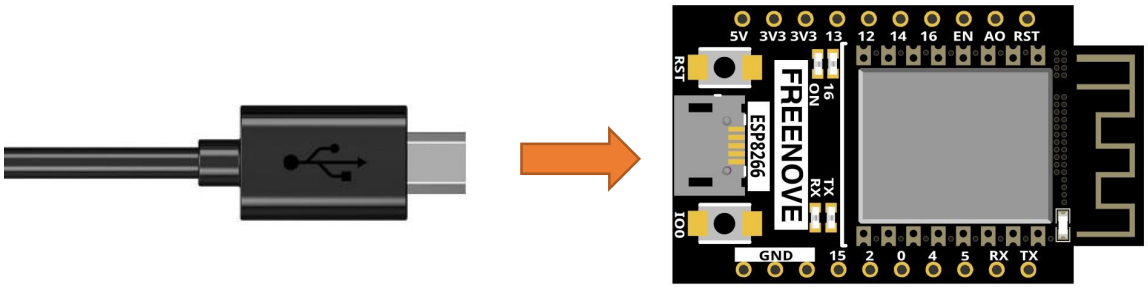
clear receive: clear out the content in the receiving text box

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

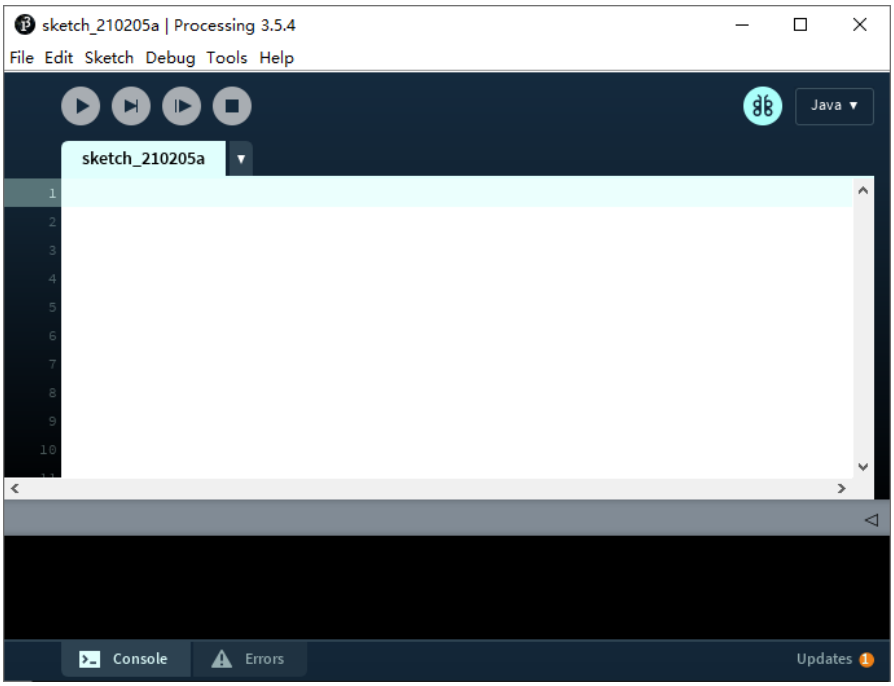
Connect Freenove ESP8266 to the computer using USB cable.



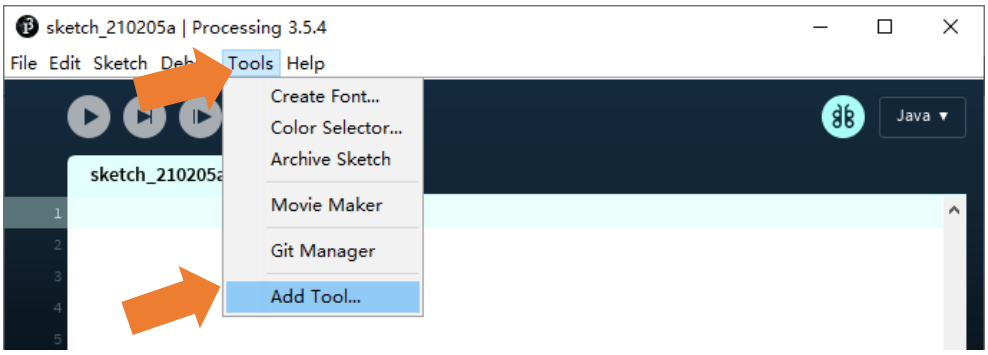
Code

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

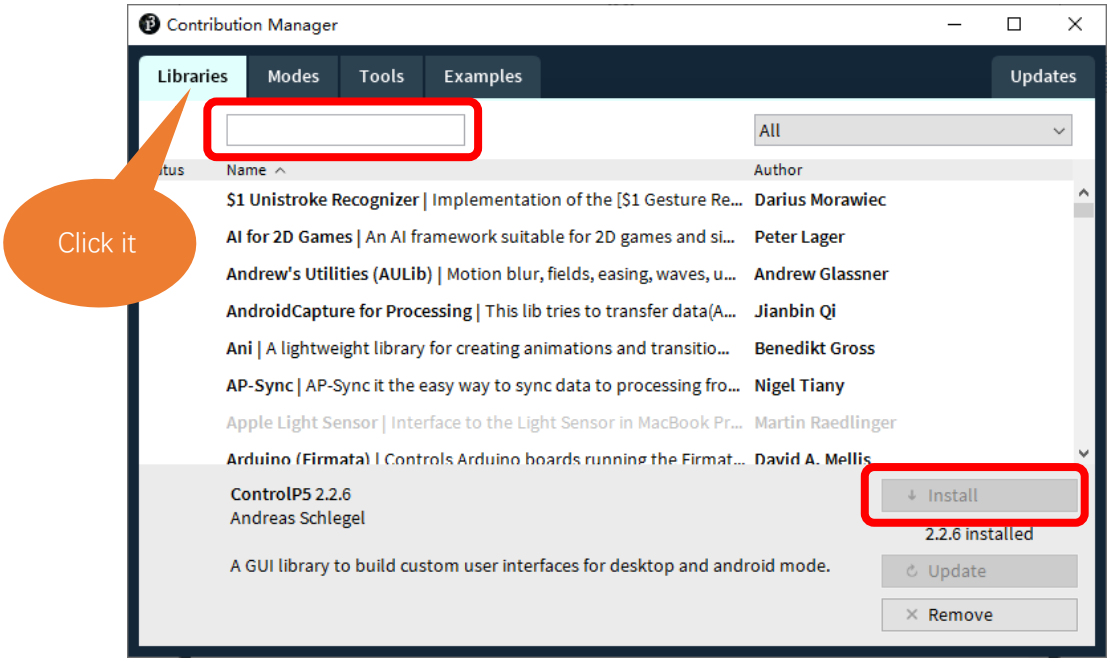
Open Processing.



Click Add Tool under Tools.

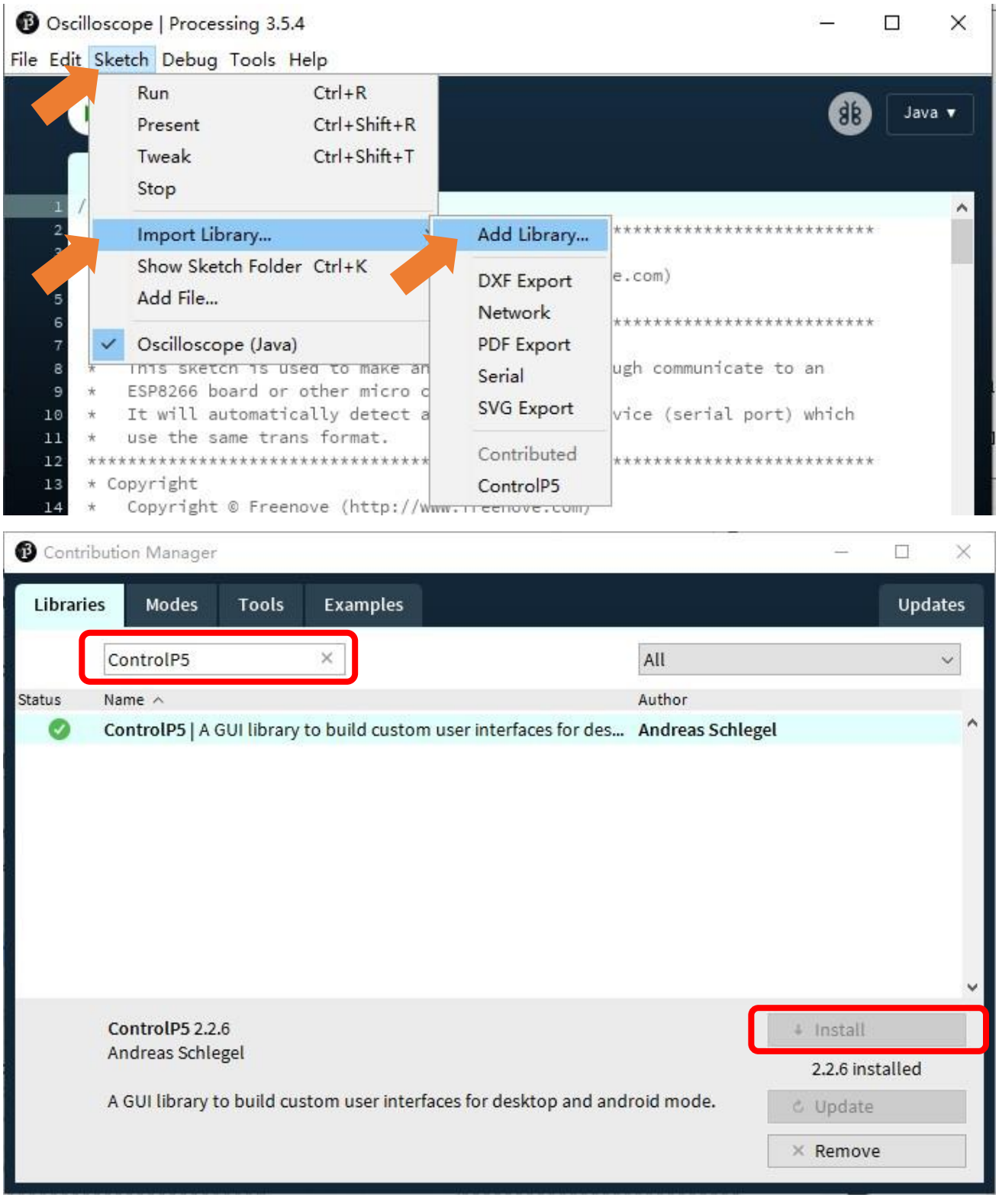


Select Libraries in the pop-up window.

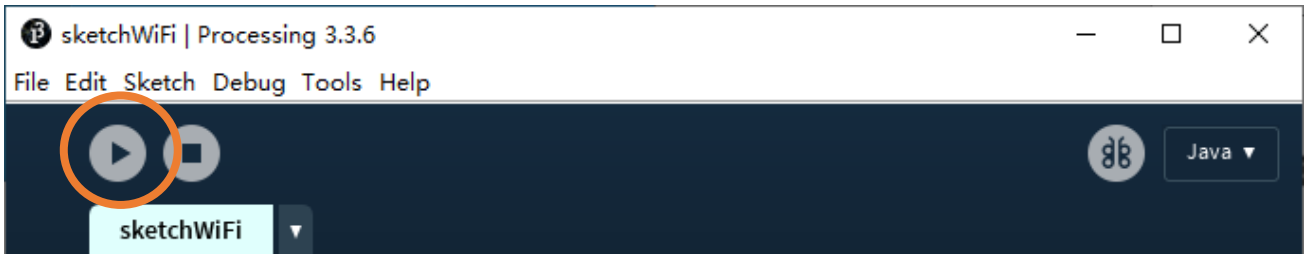


Input "ControlP5" in the searching box, and then select the option as below. Click "Install" and wait for the installation to finish.

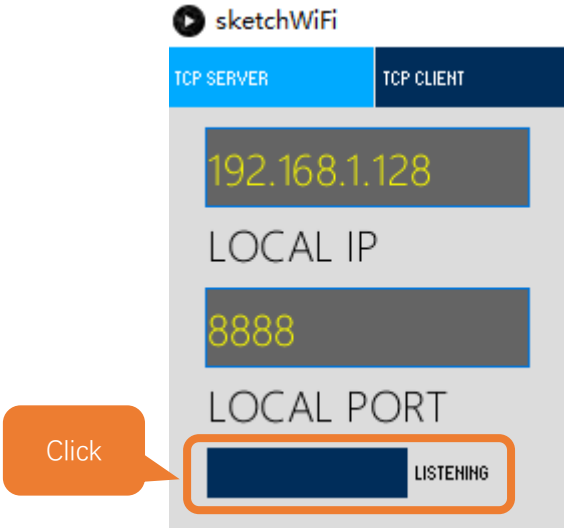
You can also click Add Library under 'Import Library' under 'Sketch'.


Oscilloscope | Processing 3.5.4
File Edit Sketch Debug Tools Help
Run Ctrl+R
Present Ctrl+Shift+R
Tweak Ctrl+Shift+T
Stop
Import Library...
Show Sketch Folder Ctrl+K
Add File...
Oscilloscope (Java)
Add Library...
DXF Export
Network
PDF Export
Serial
SVG Export
Contributed
ControlP5
Java
Contribution Manager
Libraries
Modes
Tools
Examples
Updates
ControlP5
All
Status
Name
Author
ControlP5 | A GUI library to build custom user interfaces for des...
Andreas Schlegel
ControlP5 2.2.6
Andreas Schlegel
A GUI library to build custom user interfaces for desktop and android mode.
Install
2.2.6 installed
Update
Remove

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"。

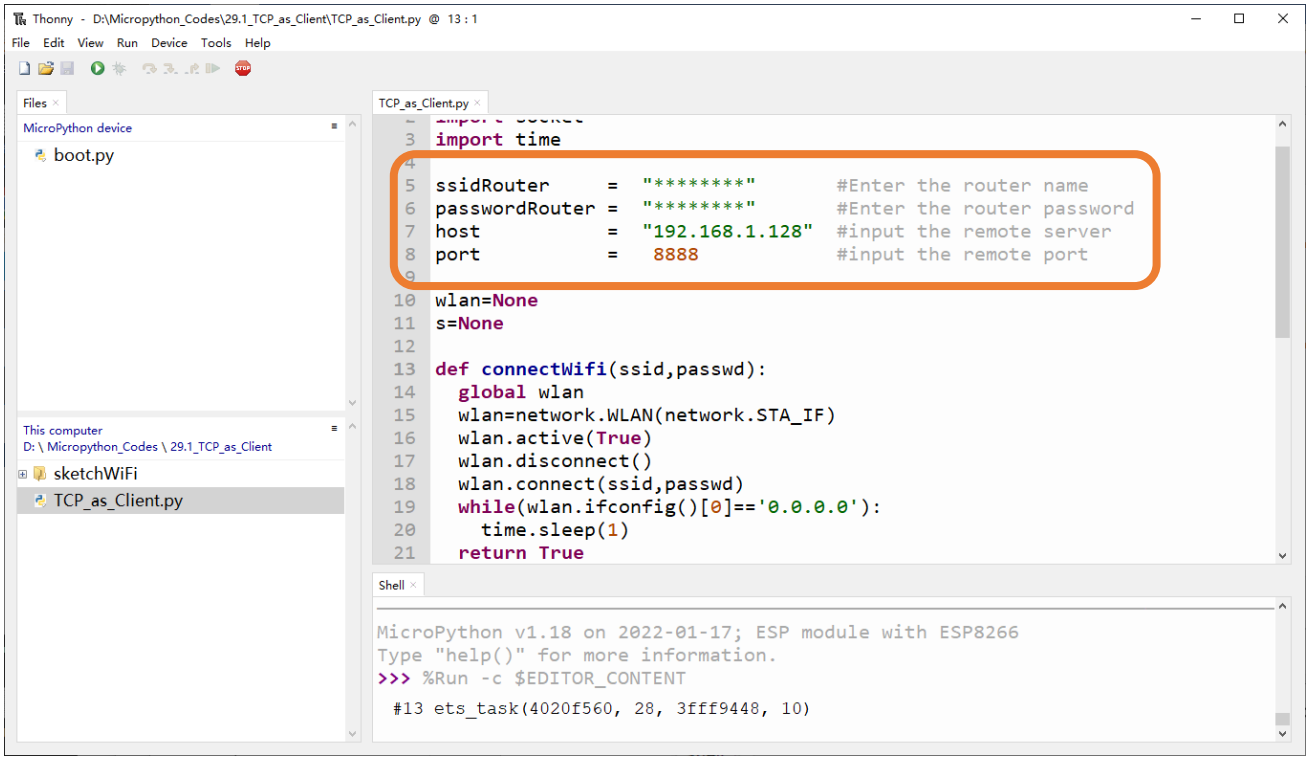


Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

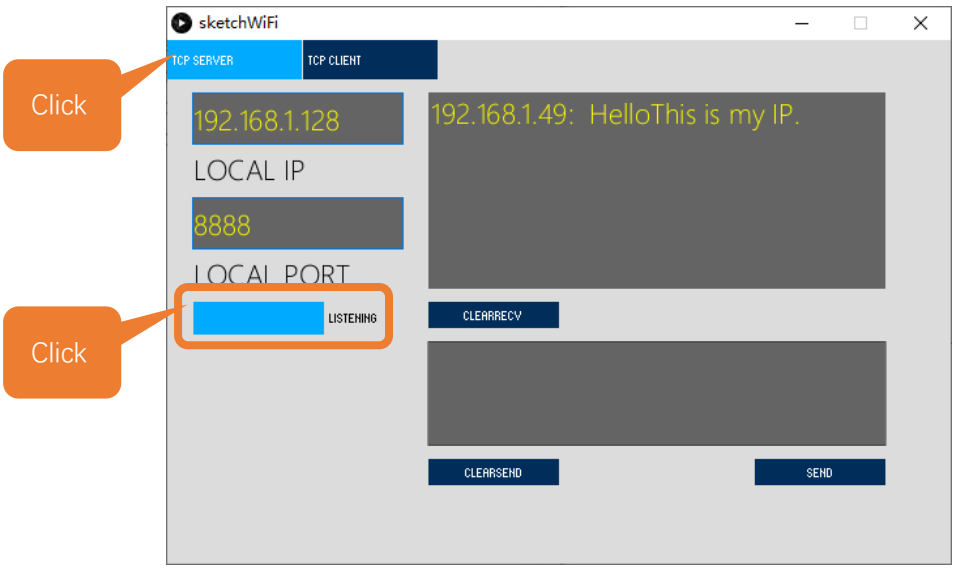
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "29.1_TCP_as_Client" and double click "TCP_as_Client.py".

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 in processing app** shown in the box below:

29.1_TCP_as_Client



Click "Run current script" and in "Shell", you can see ESP8266 automatically connects to sketchWiFi.

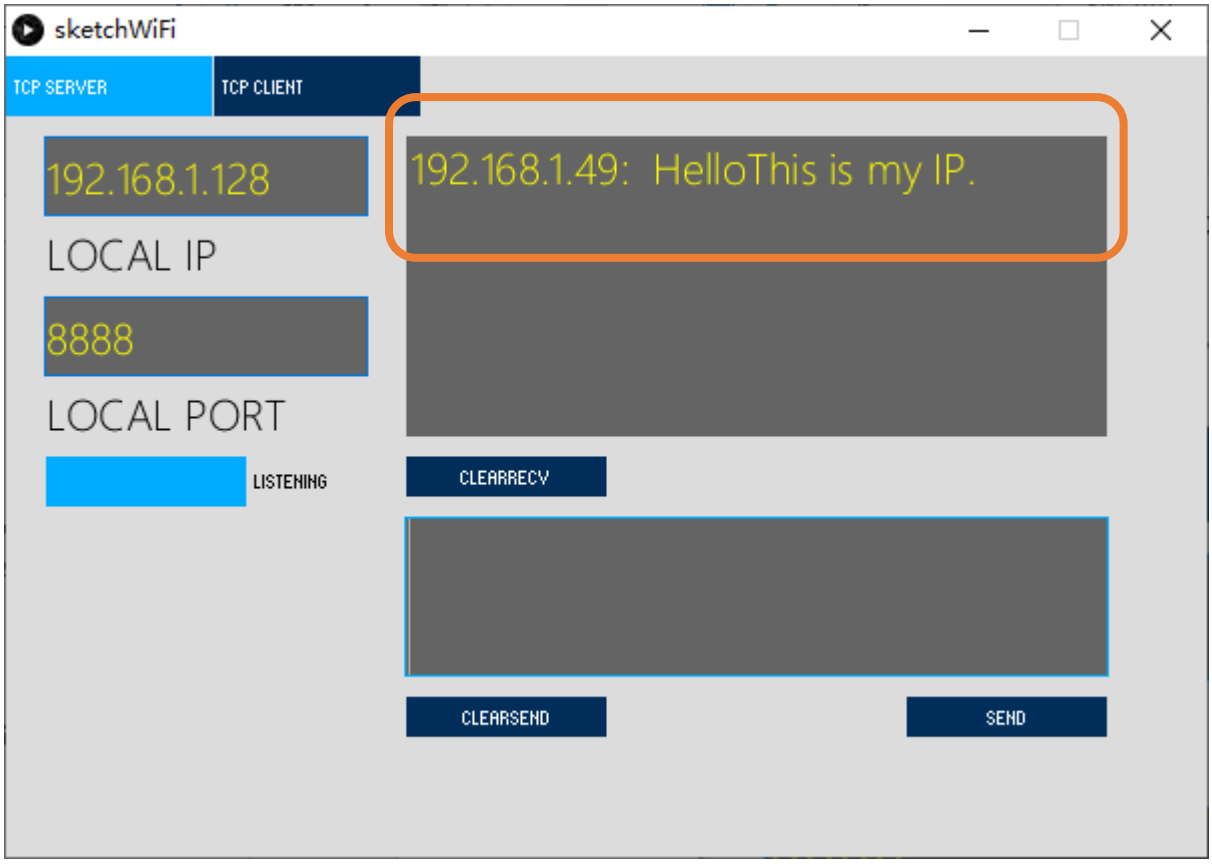


```
Shell
MicroPython v1.18 on 2022-01-17; ESP module with ESP8266
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
  #5 ets_task(4020f560, 28, 3fff9ef0, 10)
TCP Connected to: 192.168.1.128 : 8888
```

If you don't click "Listening" for sketchWiFi, ESP8266 will fail to connect and will print information as follows:

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

ESP8266 connects with TCP SERVER, and TCP SERVER receives messages from ESP8266, 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.

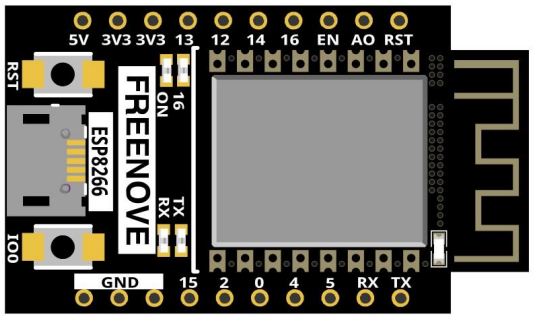
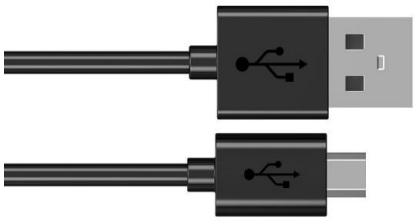
close(): Close socket.

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

Project 13.2 As Server

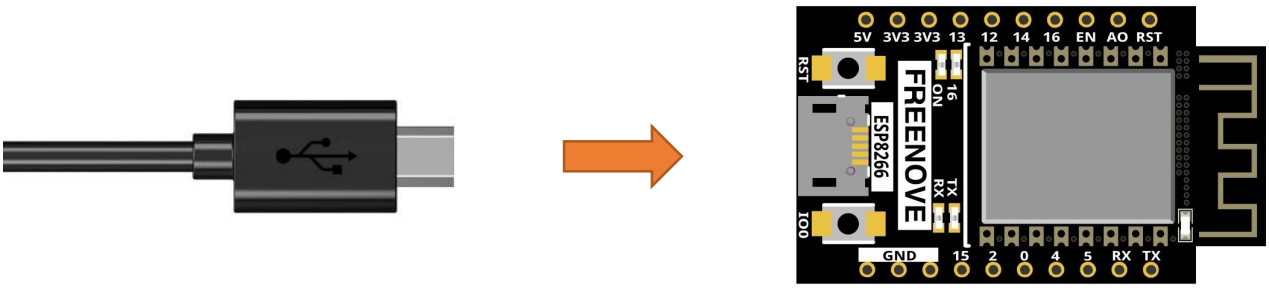
In this section, ESP8266 is used as a Server to wait for the connection and communication with Client on the same LAN.

Component List

ESP8266 x1 	USB cable 

Circuit

Connect Freenove ESP8266 to the computer using the USB cable.



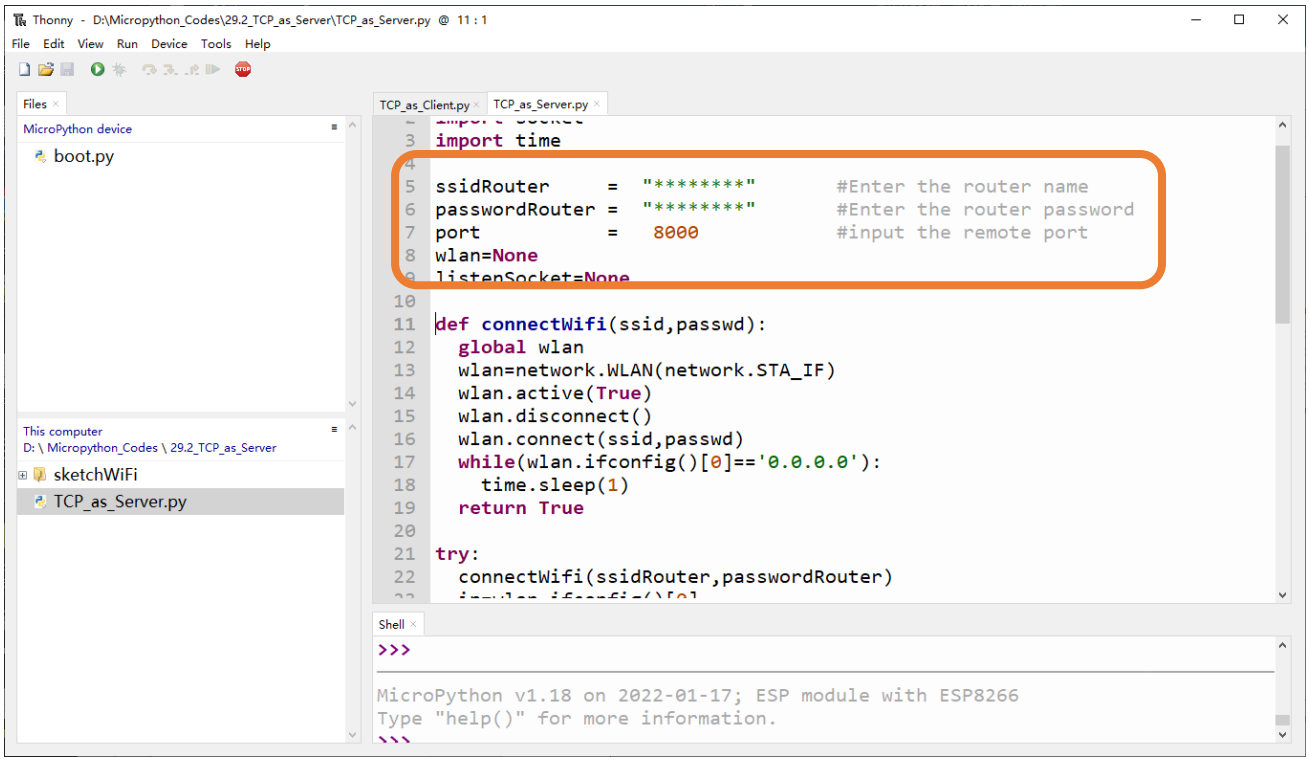
Code

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

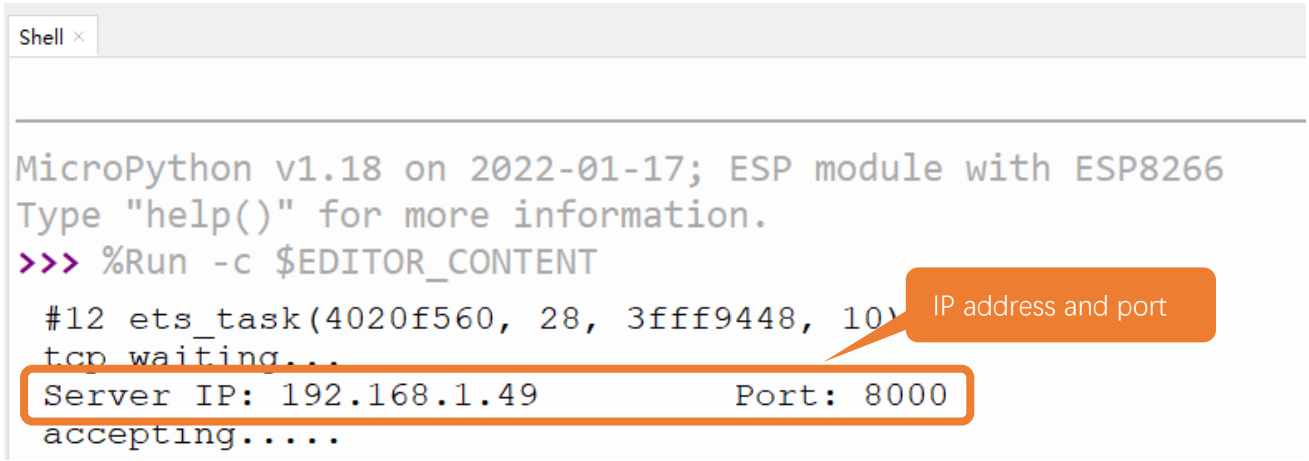
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "29.2_TCP_as_Server" and double click "TCP_as_Server.py".

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

29.2_TCP_as_Server



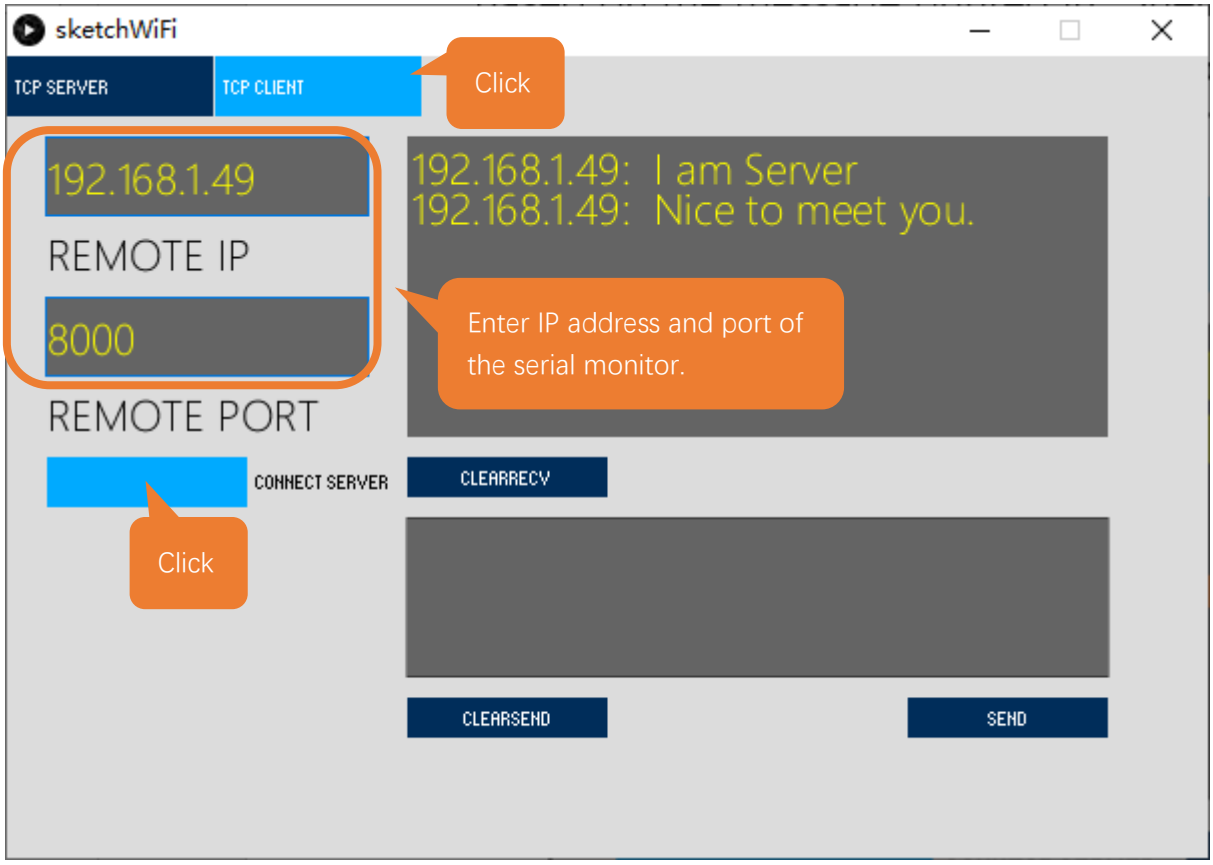
After making sure that the router's name and password are correct, click "Run current script" and in "Shell", you can see a server opened by the ESP8266 waiting to connecting to other network devices.



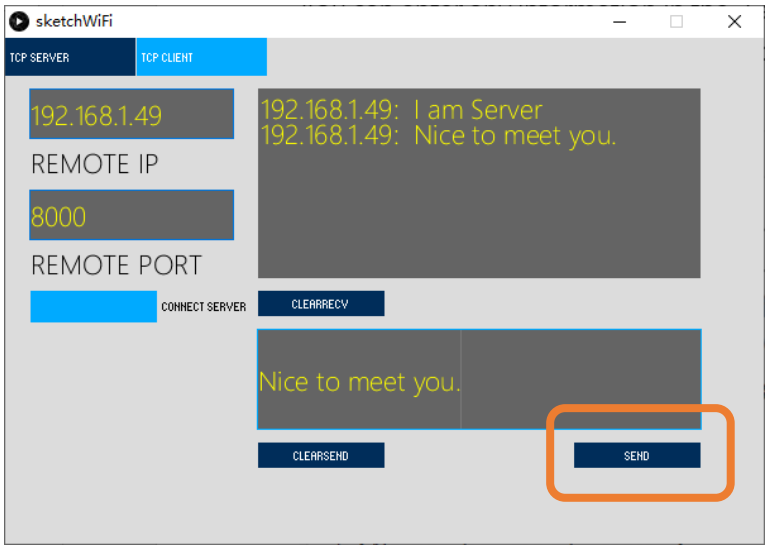
Processing:

Open the "**Freenove_Basic_Starter_Kit_for_ESP8266/Codes/MicroPython_Codes/29.2_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 ESP8266 to communicate.



You can enter any information in the "Send Box" of sketchWiFi. Click "Send" and ESP8266 will print the received messages to "Shell" and send them back to sketchWiFi.



```
Shell
MicroPython v1.18 on 2022-01-17; ESP module with ESP8266
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
 #12 ets_task(4020f560, 28, 3fff9448, 10)
 tcp waiting...
 Server IP: 192.168.1.49          Port: 8000
 accepting.....
 ('192.168.1.128', 50312) 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 ESP8266.

```
  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)
```

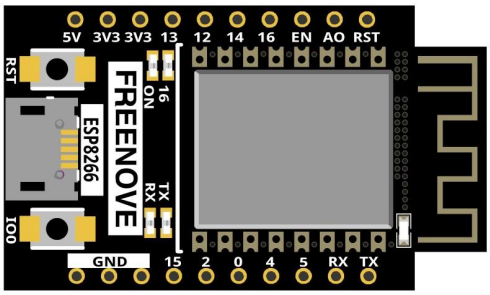
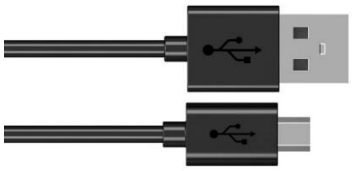
Chapter 14 Smart Home

In this chapter, we will use ESP8266 to make a simple smart home.We will learn how to control LED lights through web pages.

Project 14.1 Control_LED_through_Web

In this project, we need to build a Web Service and then use ESP8266 to control the LED through the Web browser of the PC.Through this example, you can remotely control the appliances in your home to achieve smart home.

Component List

ESP8266 x1 	Micro USB Wire x1 

Component knowledge

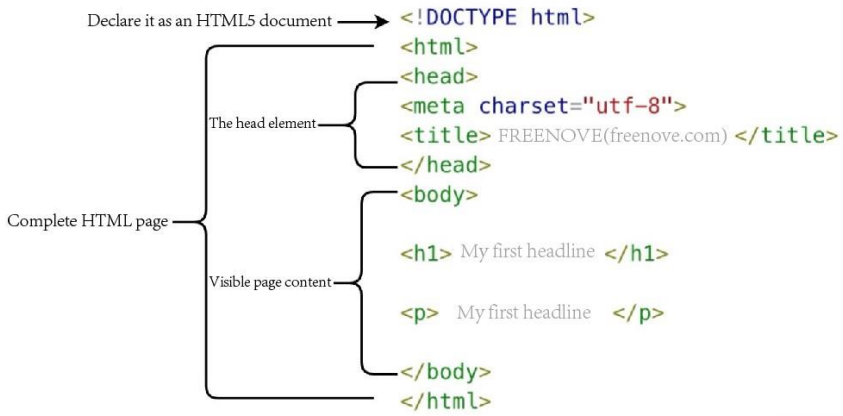
HTML

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

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

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

Example analysis is shown in the figure below:



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

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

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

<title>: Notesthe title of the document

<body>: Contains visible page content

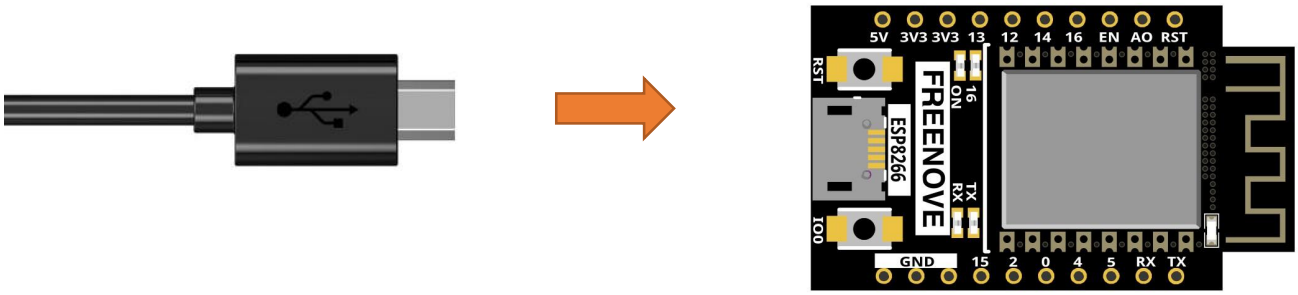
<h1>: Define a big heading

<p>: Define a paragraph

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

Circuit

Connect Freenove ESP8266 to the computer using a USB cable.

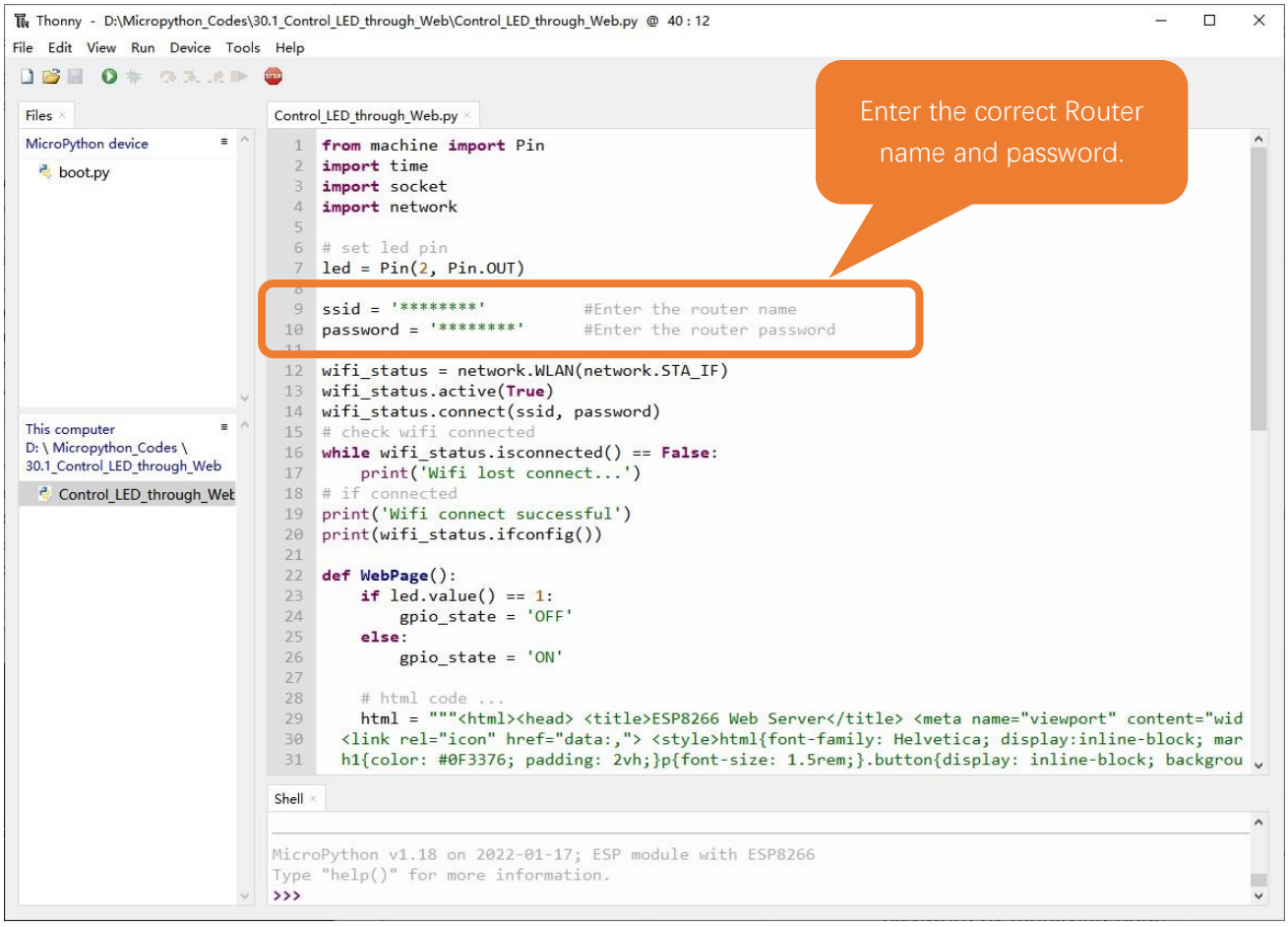


Code

Move the program folder "**Freenove_Basic_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "30.1_Control_LED_through_Web". and double click "Control_LED_through_Web".

30.1_Control_LED_through_Web



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 ESP8266, wait for ESP8266 to connect to your router and print the IP address assigned by the router to ESP8266 in "Shell".

```
Shell
Wifi lost connect...
Wifi lost connect...
Wifi lost connect...
Wifi lost connect...
Wifi lost connect...
Wifi lost connect...
Wifi lost connect...
Wifi lost connect...
Wifi lost connect...
Wifi connect successful
('192.168.1.45', '255.255.255.0', '192.168.1.1', '192.168.1.1')
```

When ESP8266 successfully connects to "ssid", "Shell" displays the IP address assigned to ESP8266 by the router. Access http://192.168.1.45 in a computer browser on the LAN. As shown in the following figure:

You can click the corresponding button to control the LED on and off.
The following is the program code:

```
1  from machine import Pin
2  import time
3  import socket
4  import network
5
6  # set led pin
7  led = Pin(2, Pin.OUT)
8
9  ssid = '********'          #Enter the router name
10 password = '********'      #Enter the router password
11
```

```
wifi_status = network.WLAN(network.STA_IF)
wifi_status.active(True)
wifi_status.connect(ssid, password)
# check wifi connected
while wifi_status.isconnected() == False:
    print('Wifi lost connect...')
# if connected
print('Wifi connect successful')
print(wifi_status.ifconfig())

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

    # html code ...
    html = """
    <html>
        <head>
            <title>ESP8266 Web Server</title>
            <meta name="viewport" content="width=device-width, initial-scale=1">
            <link rel="icon" href="data:,">
            <style>
                html{font-family: Helvetica; display:inline-block; margin: 0px auto; text-
align: center;}
                h1{color: #0F3376; padding: 2vh;}
                p{font-size: 1.5rem;}
                button{display: inline-block; background-color: #4286f4; border: none;border-
radius: 4px; color: white; padding: 16px 40px; text-decoration: none; font-size: 30px; margin:
2px; cursor: pointer;}
                button2{background-color: #4286f4;}
            </style>
        </head>
        <body> <h1>ESP8266 Web Server</h1>
          <p>GPIO state: <strong>""" + gpio_state + """</strong></p>
          <p><a href="/?led=on"><button class="button">ON</button></a></p>
          <p><a href="/?led=off"><button class="button button2">OFF</button></a></p>
        </body>
    </html>
    """
    return html

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
53  s.bind(('', 80))
54  s.listen(5)
55  try:
56      while True:
57          conn, addr = s.accept()
58          print('Connection: %s' % str(addr))
59          req = conn.recv(1024)
60          req = str(req)
61          print('Connect = %s' % req)
62          led_on = req.find('/?led=on')
63          led_off = req.find('/?led=off')
64          if led_on == 6:
65              print('LED ON')
66              led.value(0)
67          else:
68              print('LED OFF')
69              led.value(1)
70          if led.value() == 1:
71              gpio_state = 'OFF'
72          else:
73              gpio_state = 'ON'
74          response = WebPage()
75          conn.send('HTTP/1.1 200 OK\n')
76          conn.send('Content-Type: text/html\n')
77          conn.send('Connection: close\n\n')
78          conn.sendall(response)
79          conn.close()
80  except:
81      pass
```

Import socket module and Import network module.

```
3  import socket
4  import network
```

Enter correct AP name and password.

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

Set ESP8266 in Station mode and connect it to your router.

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

"Shell" displays the IP address assigned to ESP8266.

```
20  print(wifi_status.ifconfig())
```

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

```
55          if led_on == 6:
56              print('LED ON')
```

```
57                led.value(0)
58            else:
59                print('LED OFF')
60                led.value(1)
61            if led.value() == 1:
62                gpio_state = 'OFF'
63            else:
64                gpio_state = 'ON'
```

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 ESP8266, you view our ultimate tutorial:
https://github.com/Freenove/Freenove_Basic_Starter_Kit_for_ESP8266/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.