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.

TM

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.

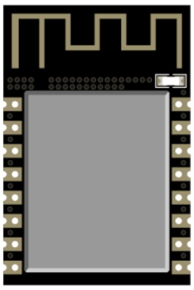
We divide each project into four parts, namely Component List, Component Knowledge, Circuit and Code. Component List helps you to prepare material for the experiment more quickly. Component Knowledge allows you to quickly understand new electronic modules or components, while Circuit helps you understand 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

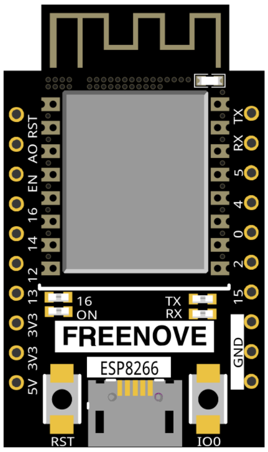
ESP8266 has PCB on-board antenna. The PCB on-board antenna is an integrated antenna in the chip module itself, so it is convenient to carry and design.

PCB on-board antenna

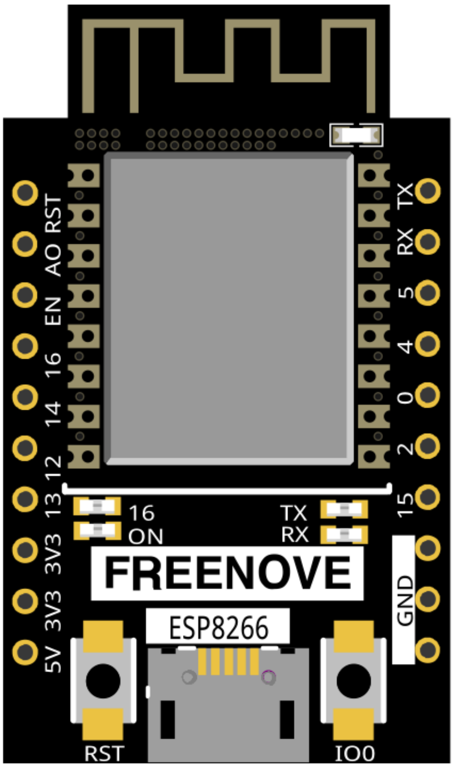


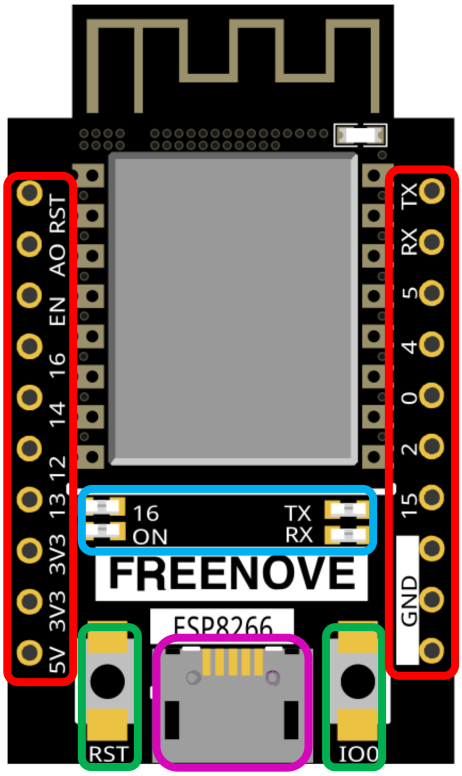
In this tutorial, the ESP8266 development board is designed based on the PCB on-board antenna-packaged ESP8266 module. The following tutorials will be based on the ESP8266 development board.

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.

For more information, please visit: https://docs.ai-thinker.com/_media/esp8266/docs/esp-12s_product_specification_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.

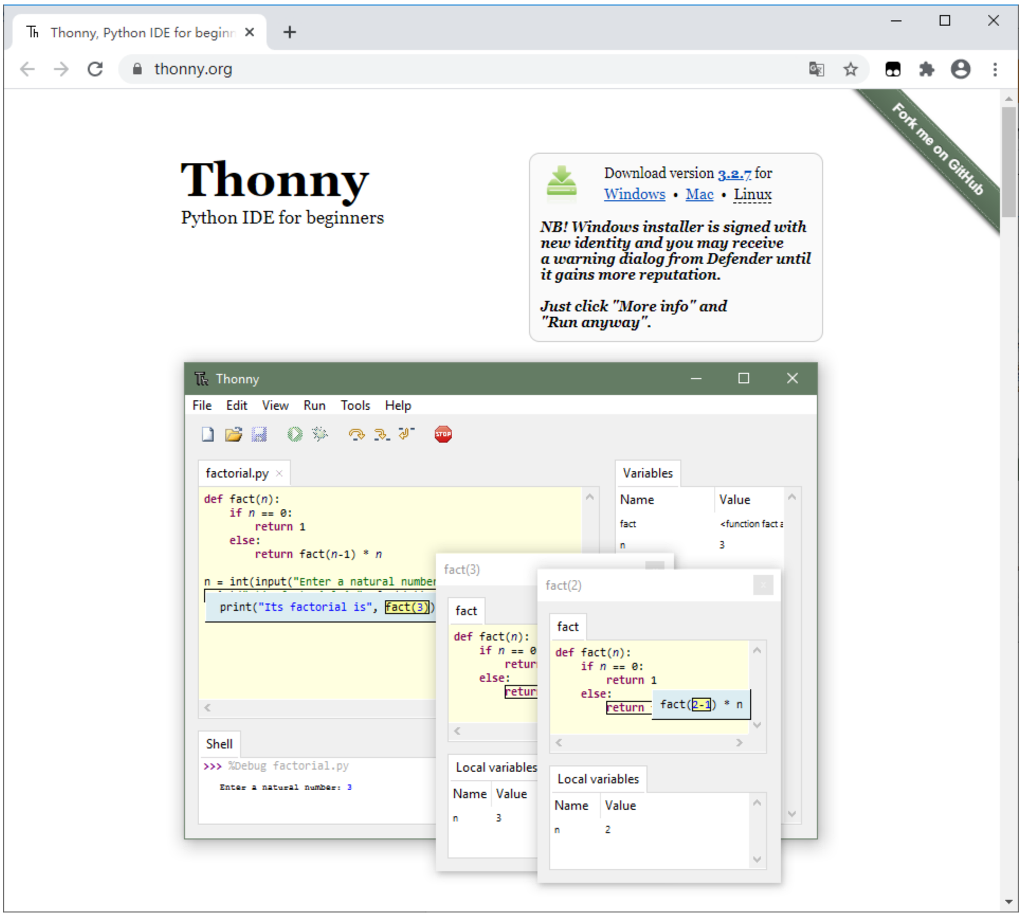
Downloading Thonny

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

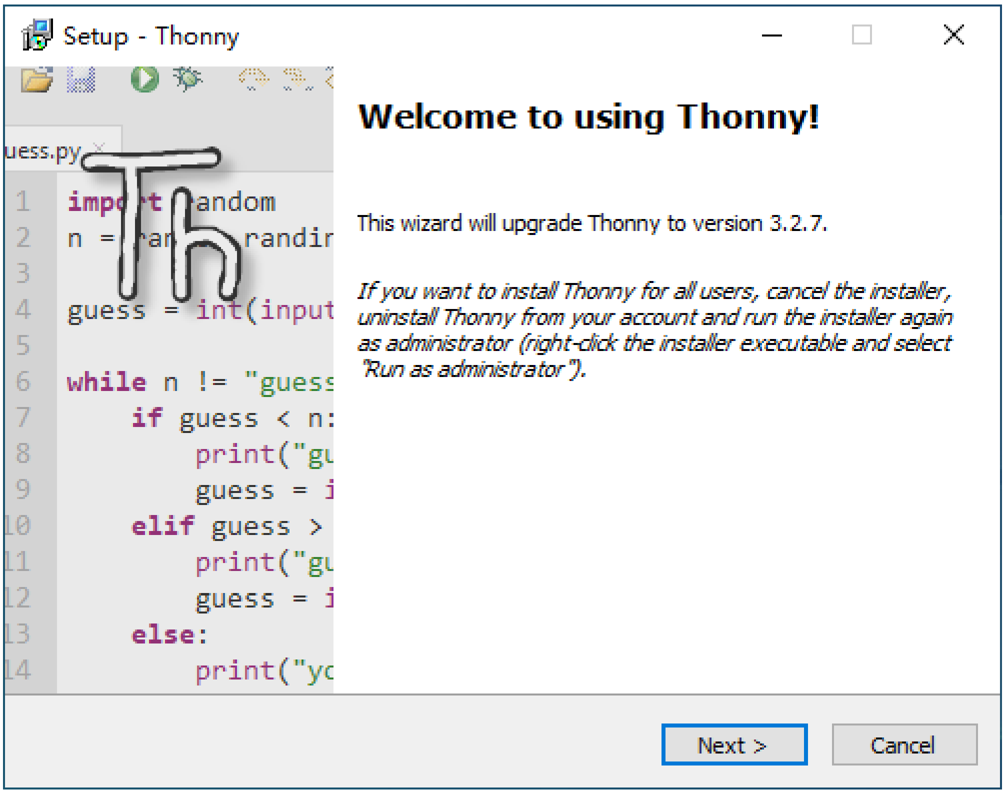
You can also open "**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Software**", we have prepared it in advance.



Installing on Windows

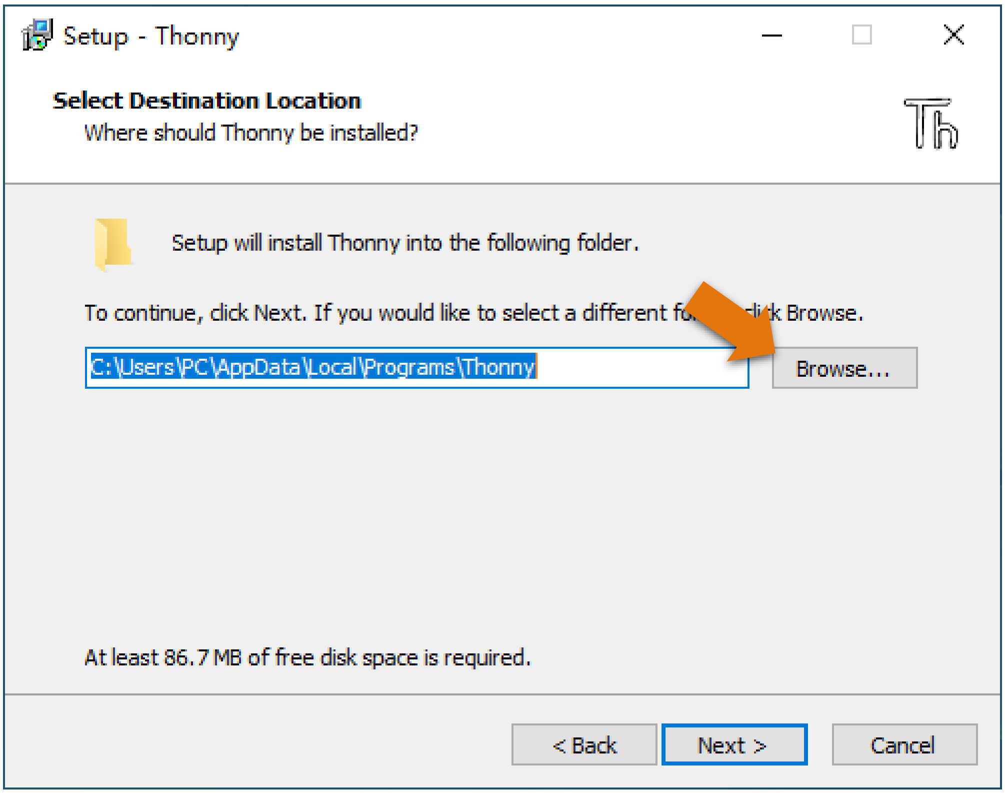
The icon of Thonny after downloading is as below. Double click “thonny-3.2.7.exe”.

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

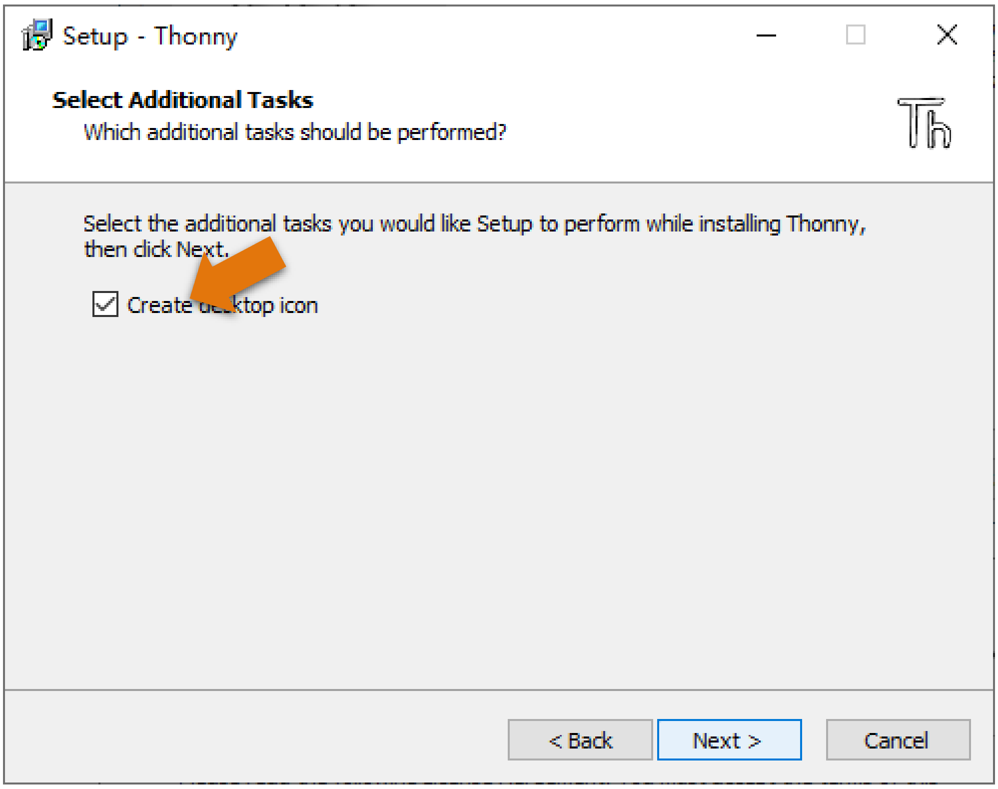


If you want to change Thonny's installation path, you can click "Browse" to modify it. After selecting installation path, click "OK".

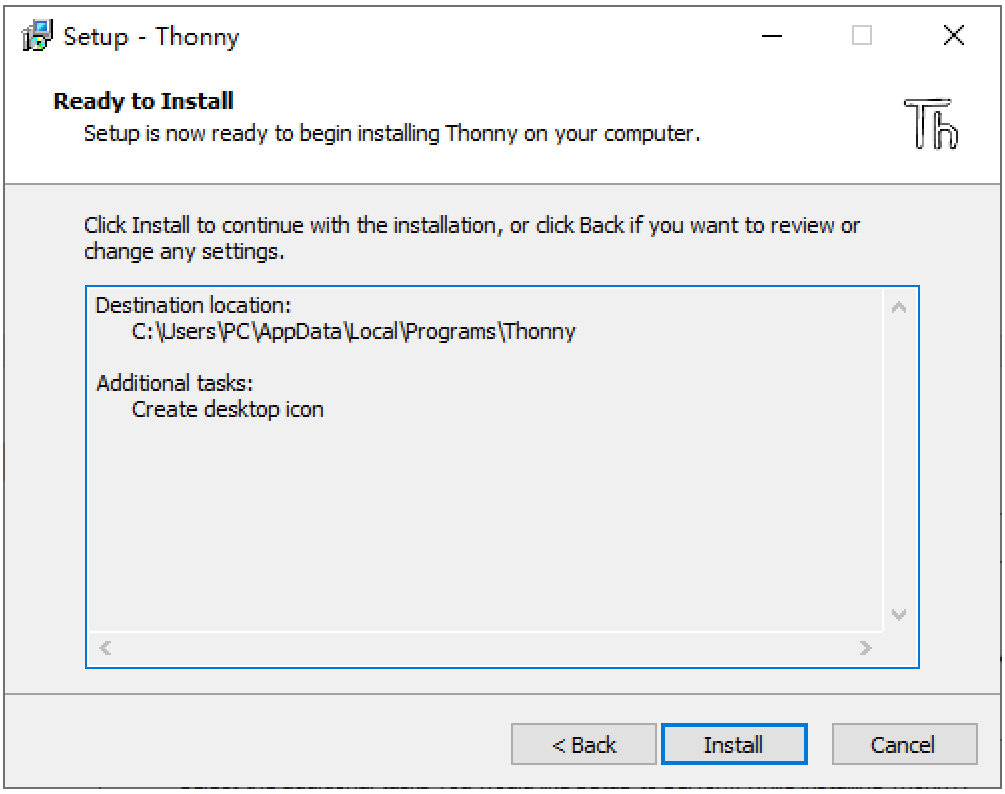
If you do not want to change it, just click "Next".



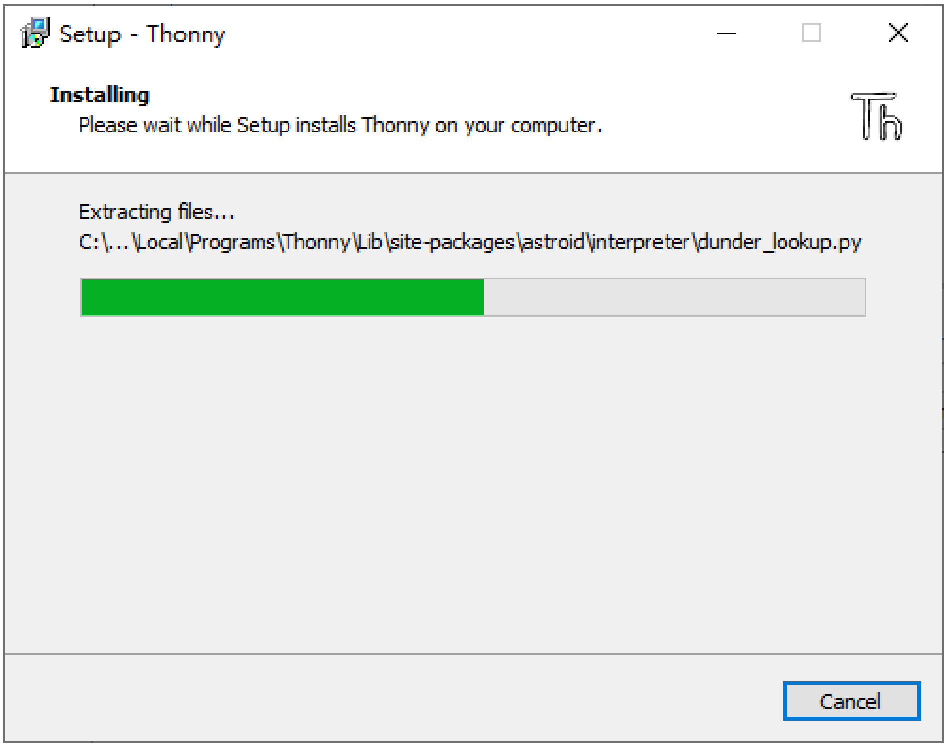
Check "Create desktop icon" and then it will generate a shortcut on your desktop to facilitate you to open Thonny later.



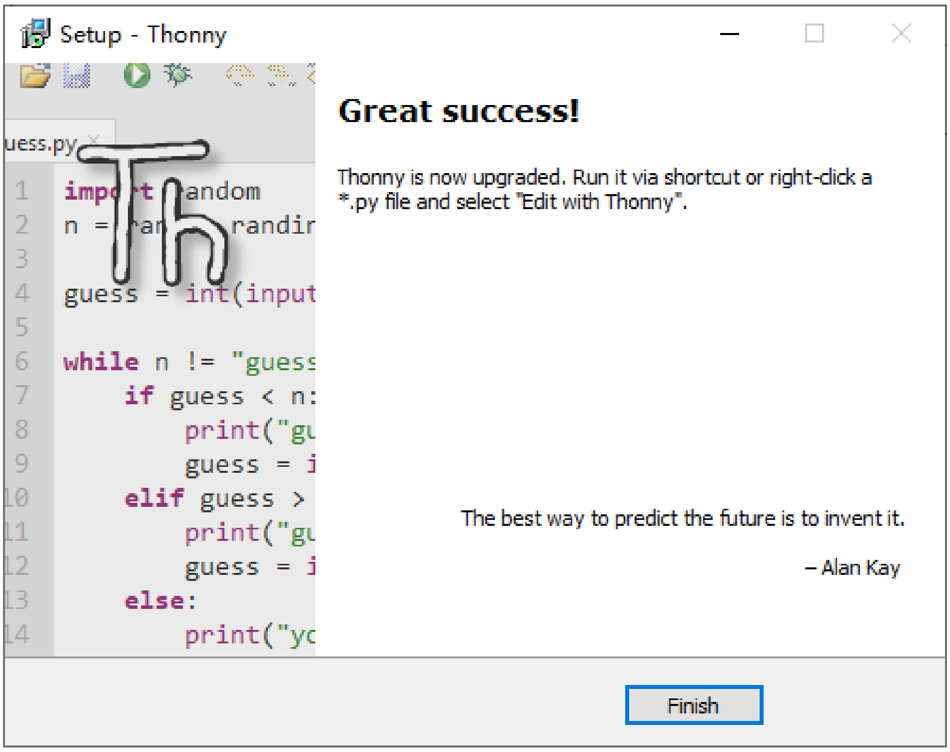
Click "install" to install the software.



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.



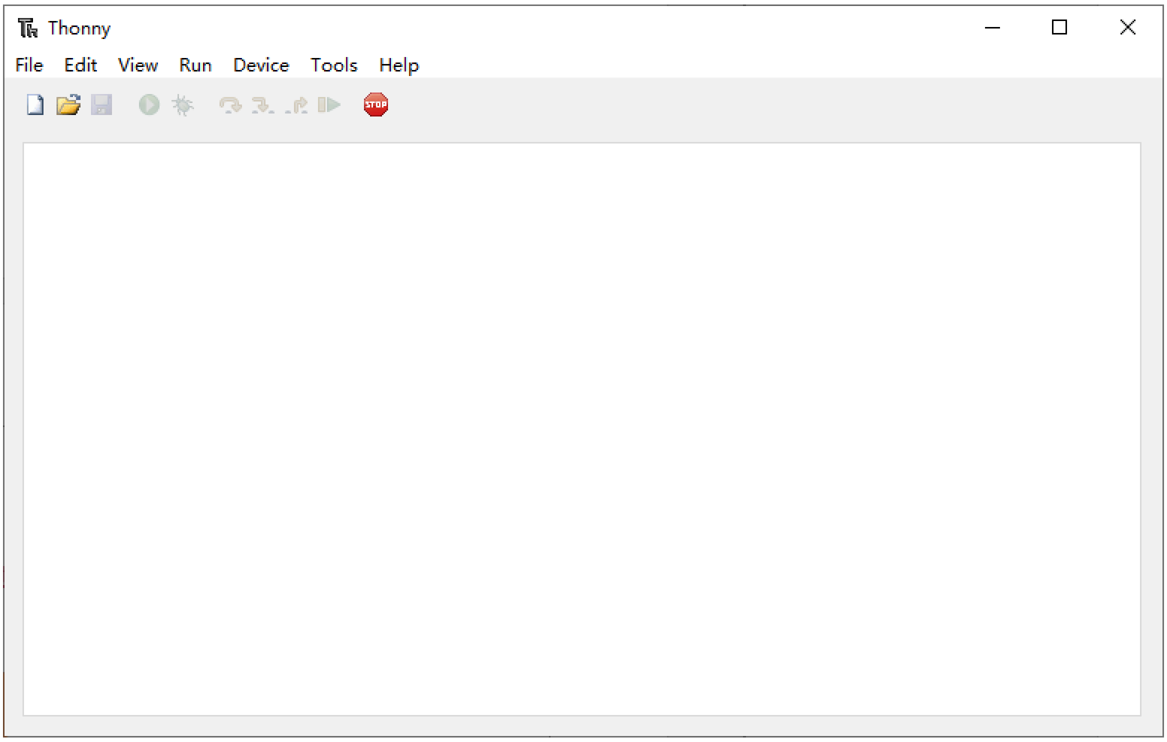
Once you see the interface as below, Thonny has been installed successfully.



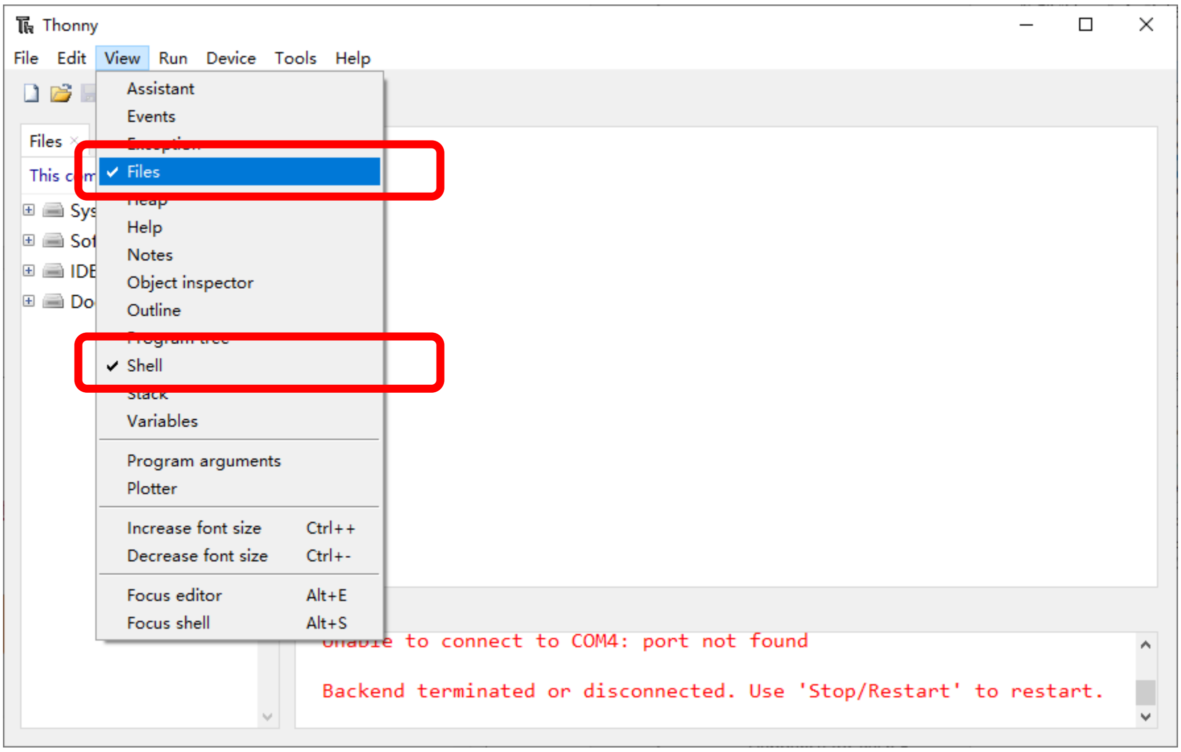
If you've check "Create desktop icon" during the installation process, you can see the below icon on your desktop.

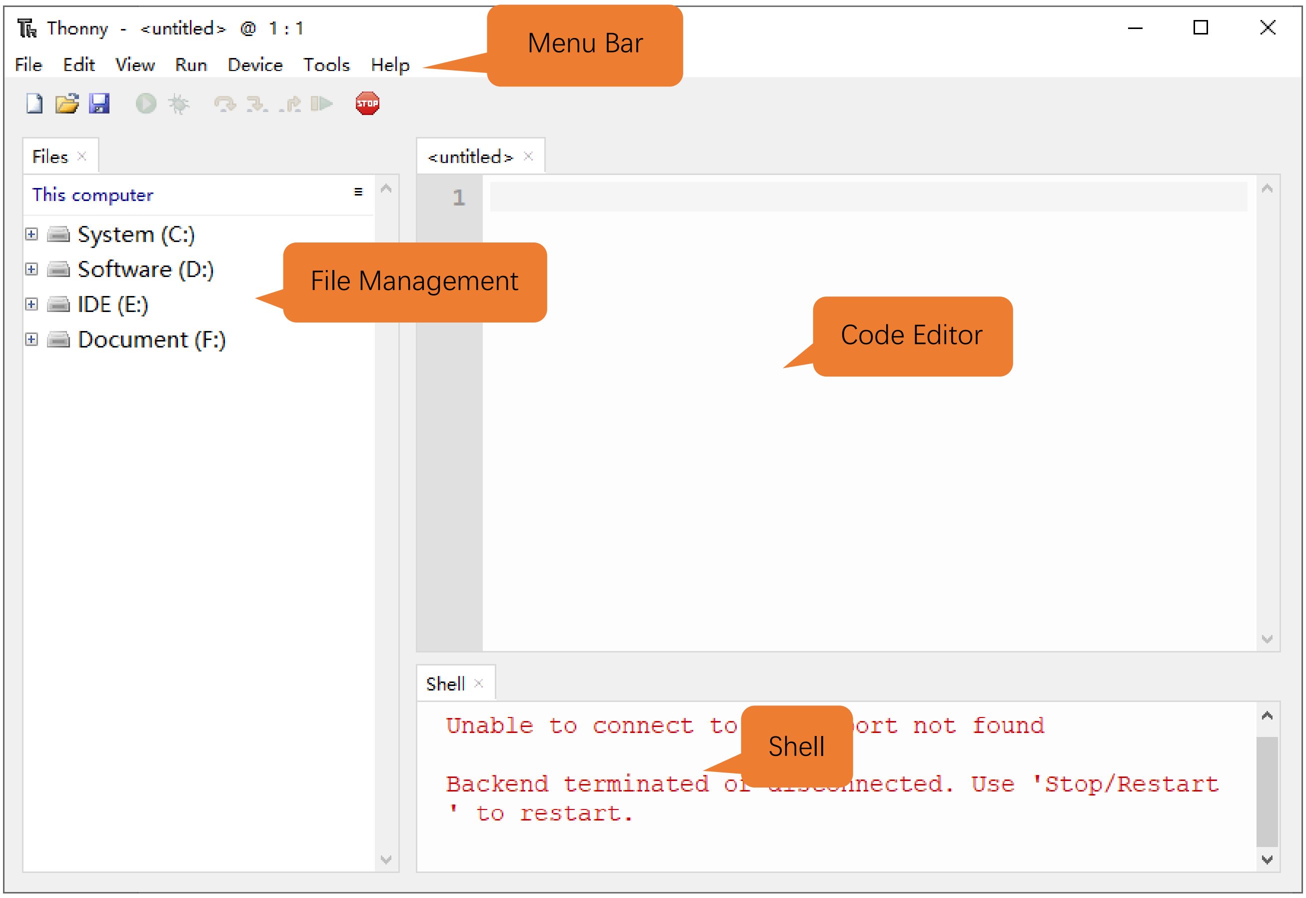
0.2 Basic Configuration of Thonny

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>
Code Editor
Shell
Shell
Backend terminated or disconnected. Use 'Stop/Restart' to restart.

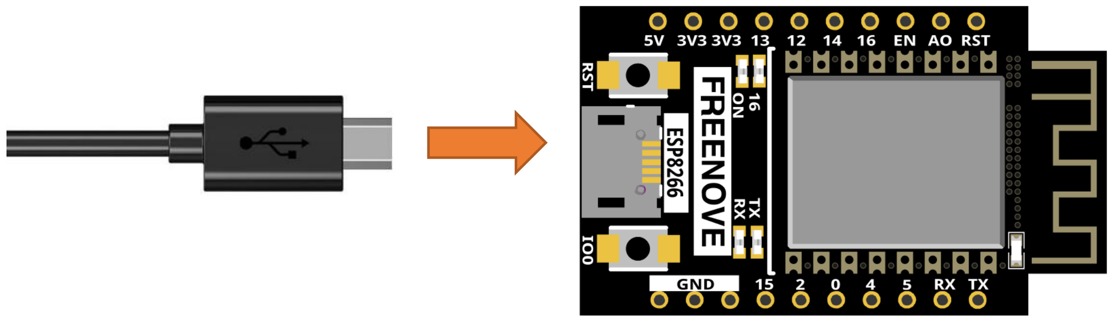
0.3 Installing CH340 (Important)

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

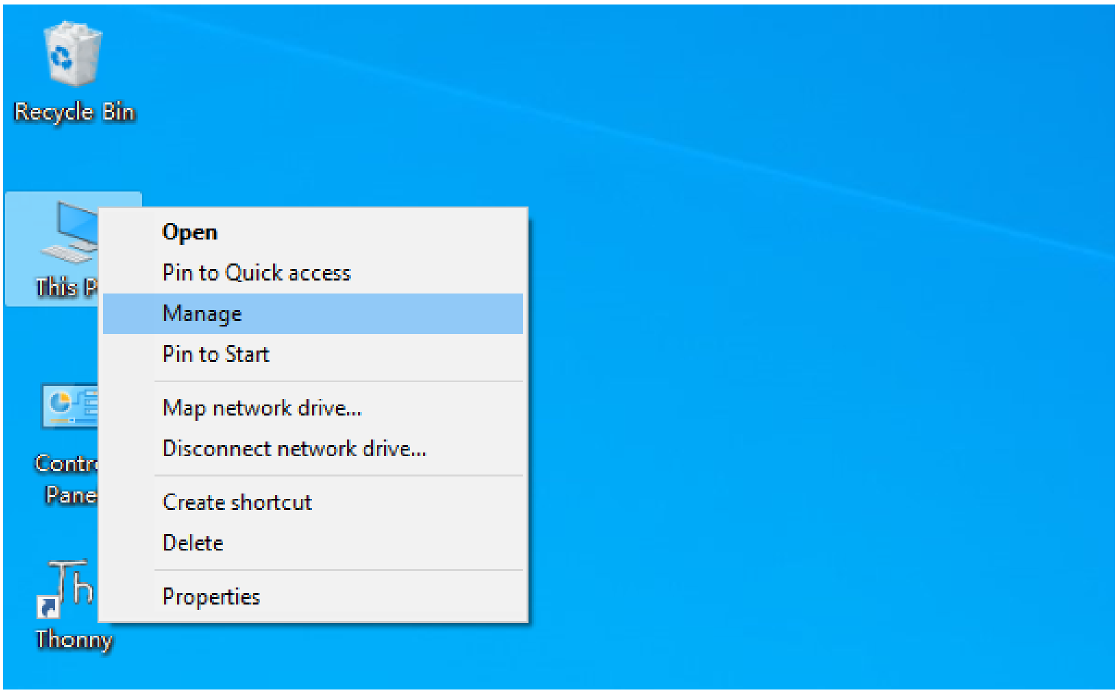
Windows

Check whether CH340 has been installed

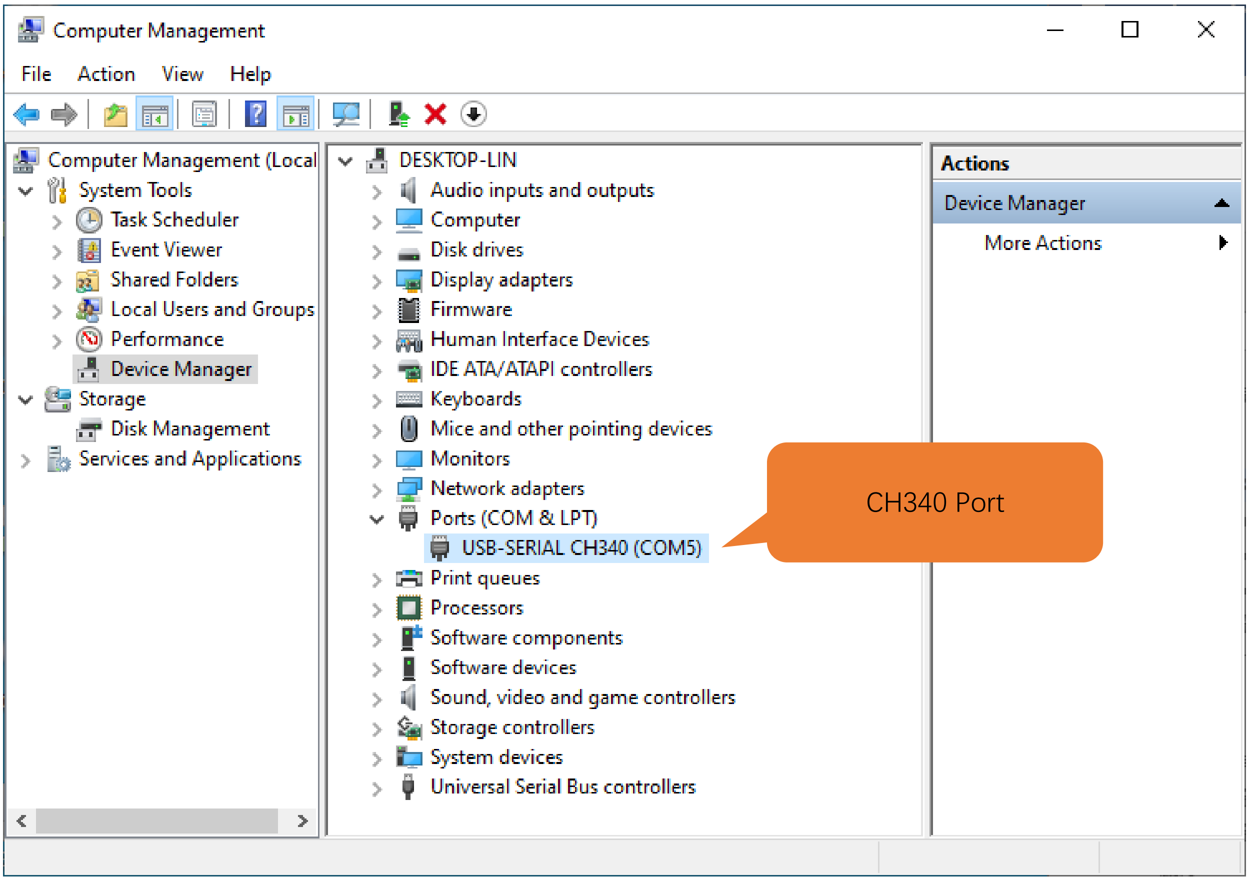
1. Connect your computer and ESP8266 with a USB cable.



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

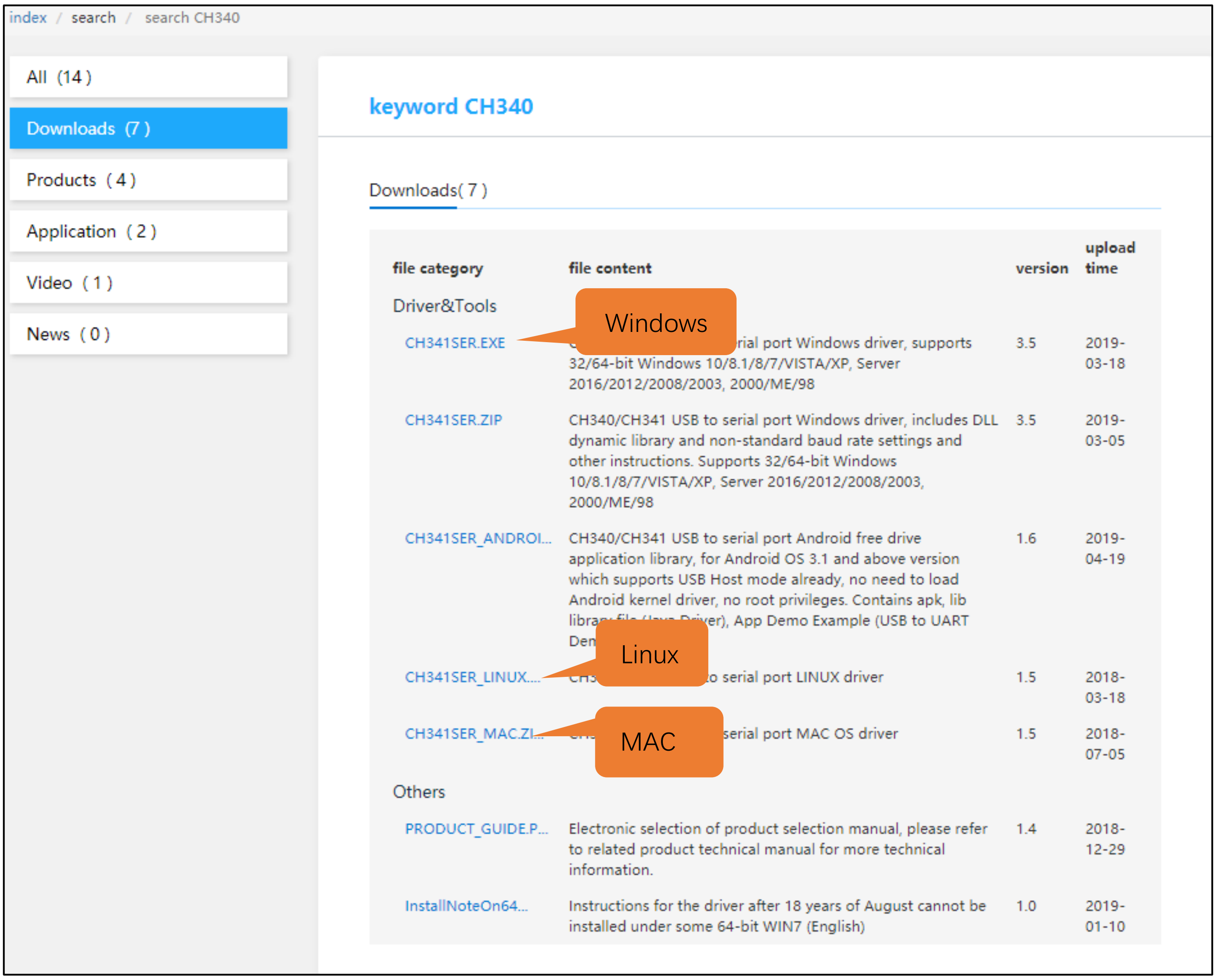


3. Click "Device Manager". If your computer has installed CH340, you can see"USB-SERIAL CH340 (COMx)". And you can click here to move to the next step.

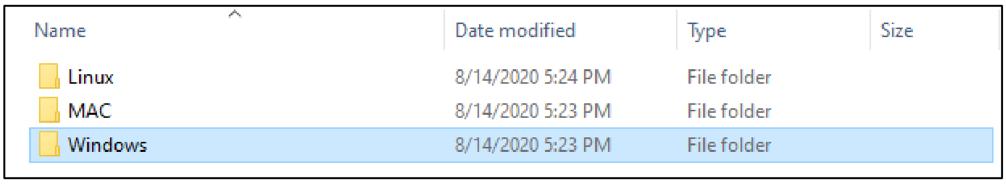


Installing CH340

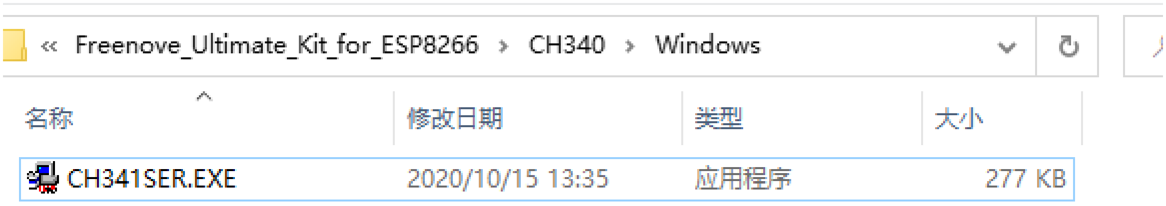
1. First, download CH340 driver, click http://www.wch-ic.com/search?q=CH340&t=downloads to download the appropriate one based on your operating system.



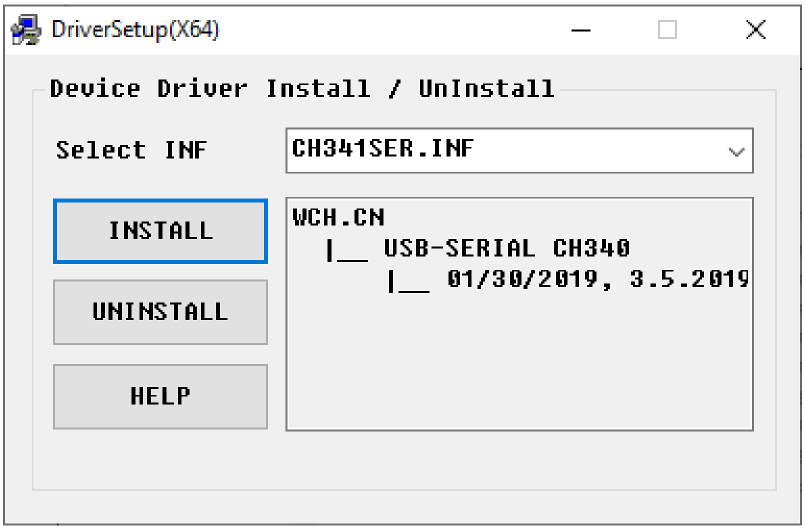
You can also open **"Freenove_Ultimate_Starter_Kit_for_ESP8266/CH340"**, we have prepared the installation package.



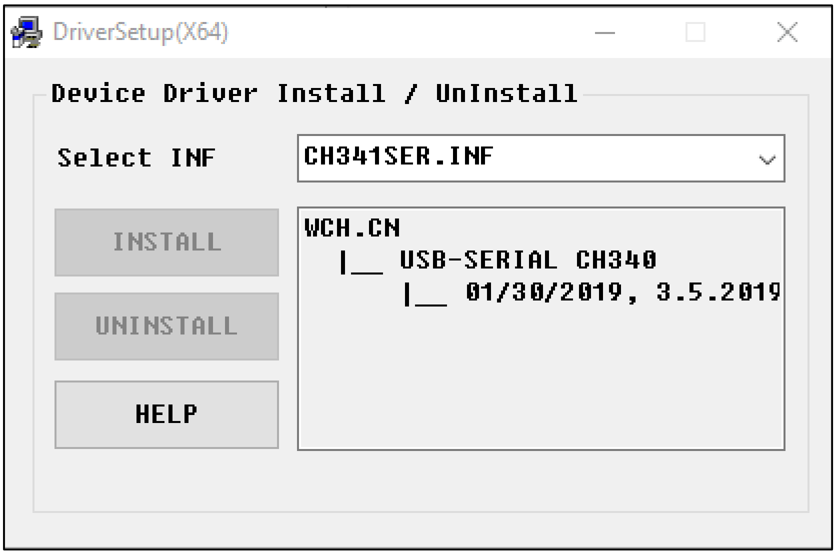
2. Open the folder "**Freenove_Ultimate_Starter_Kit_for_ESP8266/CH340/Windows/ch341ser**"



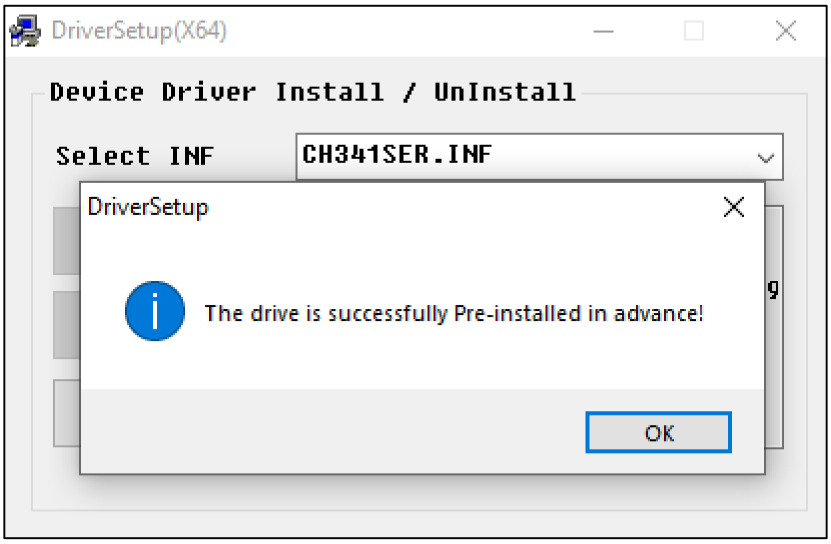
3. Double click "CH341SER.EXE".



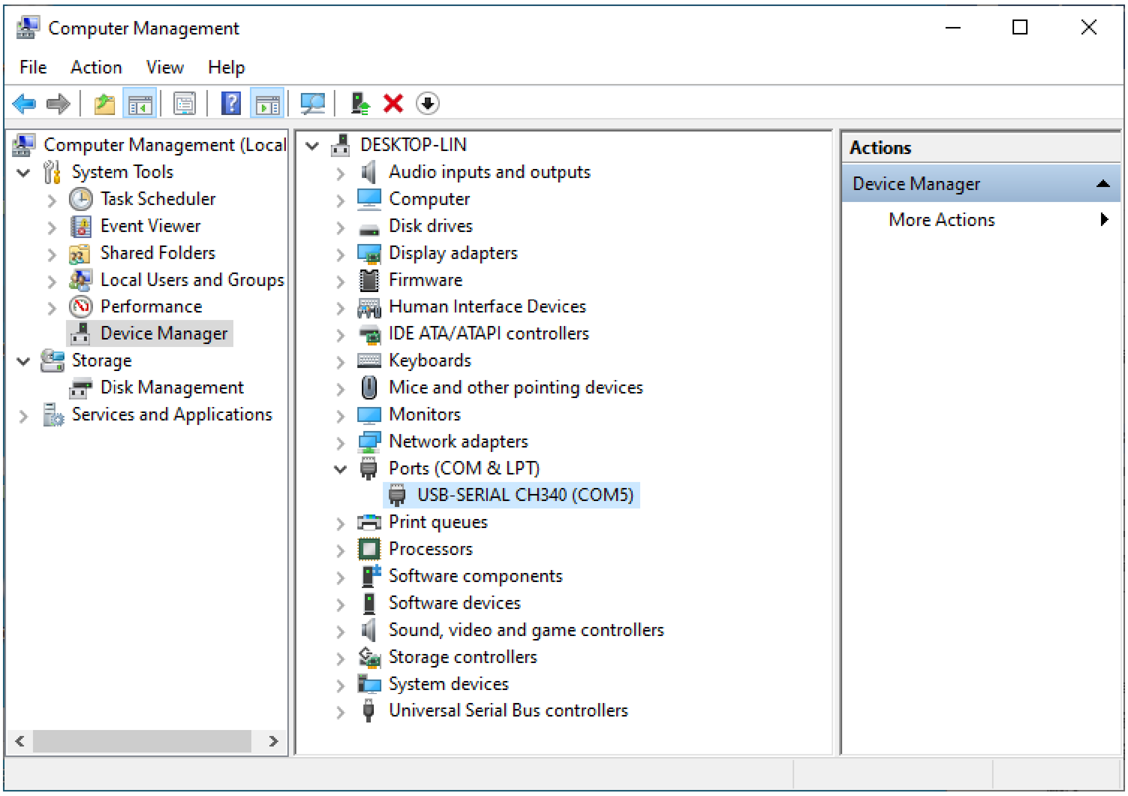
4. Click "INSTALL" and wait for the installation to complete.



5. Install successfully. Close all interfaces.



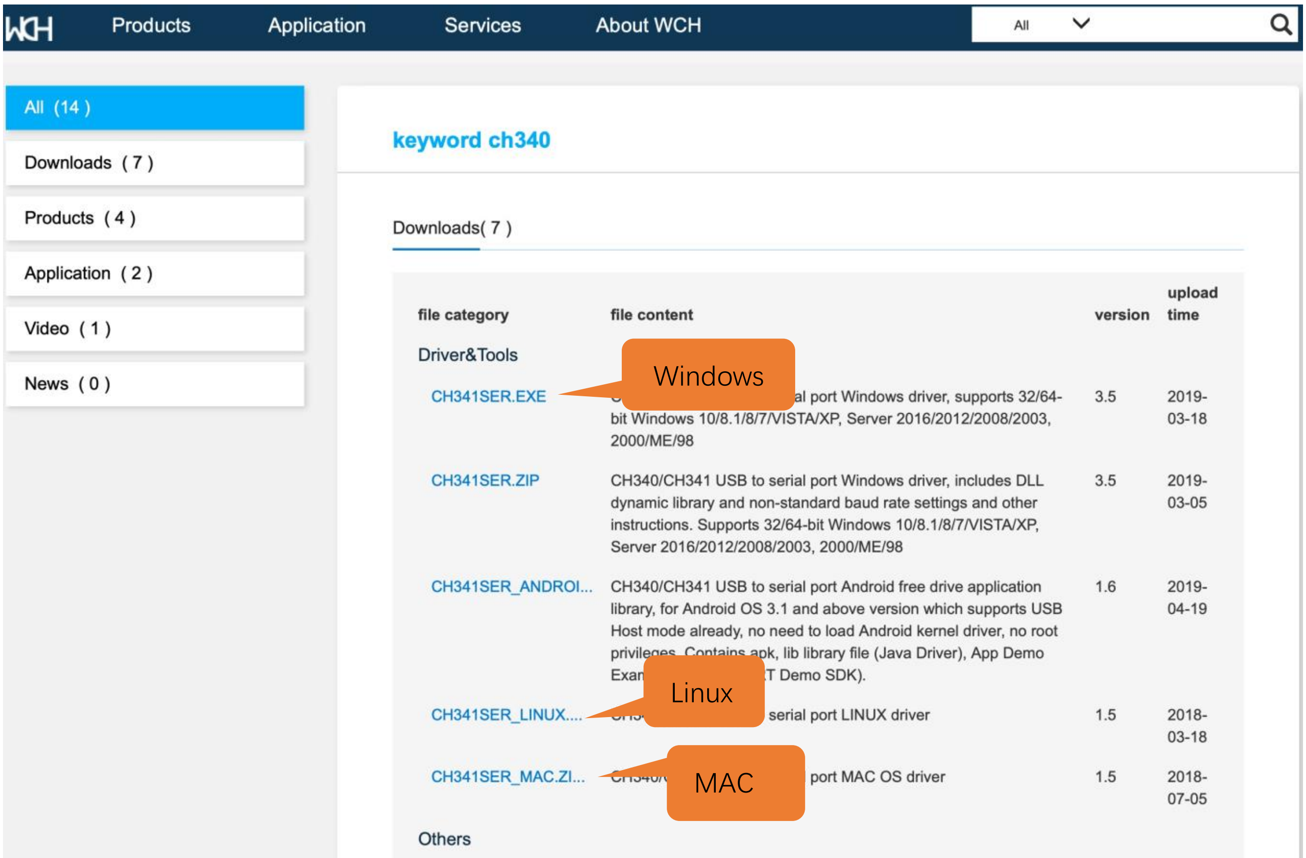
6. When ESP8266 is connected to computer, select "This PC", right-click to select "Manage" and click "Device Manager" in the newly pop-up dialog box, and you can see the following interface.



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

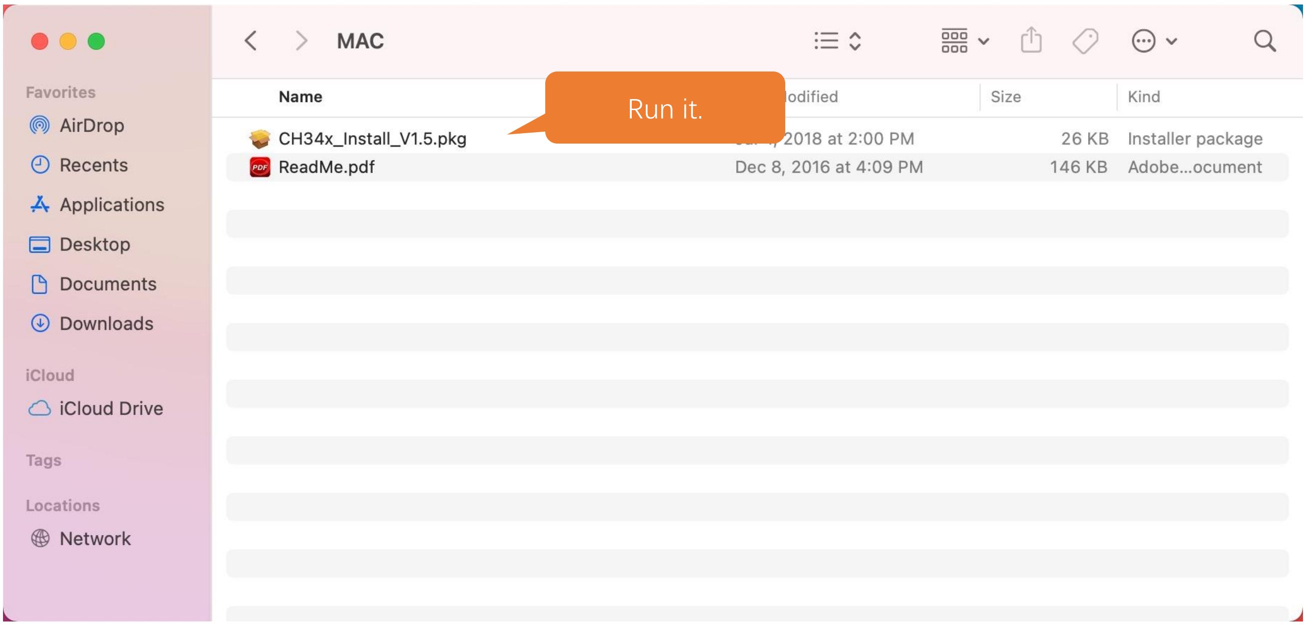
MAC

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.

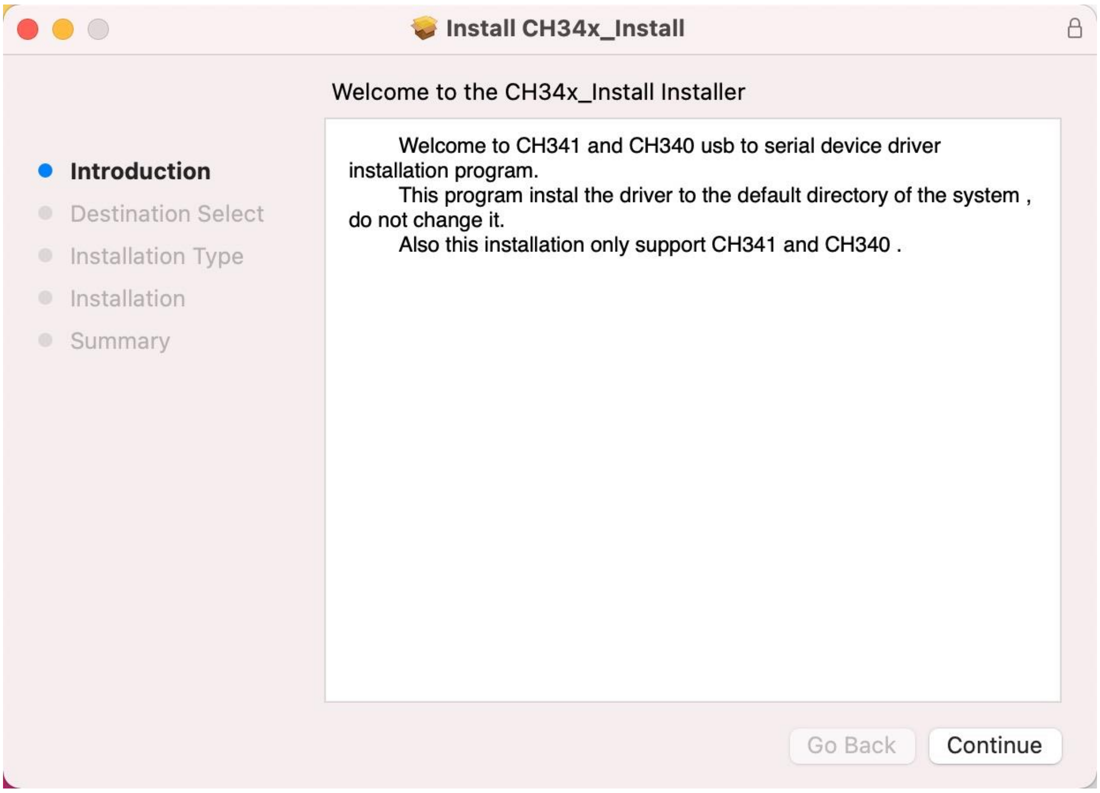


If you would not like to download the installation package, you can open **"Freenove_Ultimate_Starter_Kit_for_ESP8266/CH340"**, we have prepared the installation package.

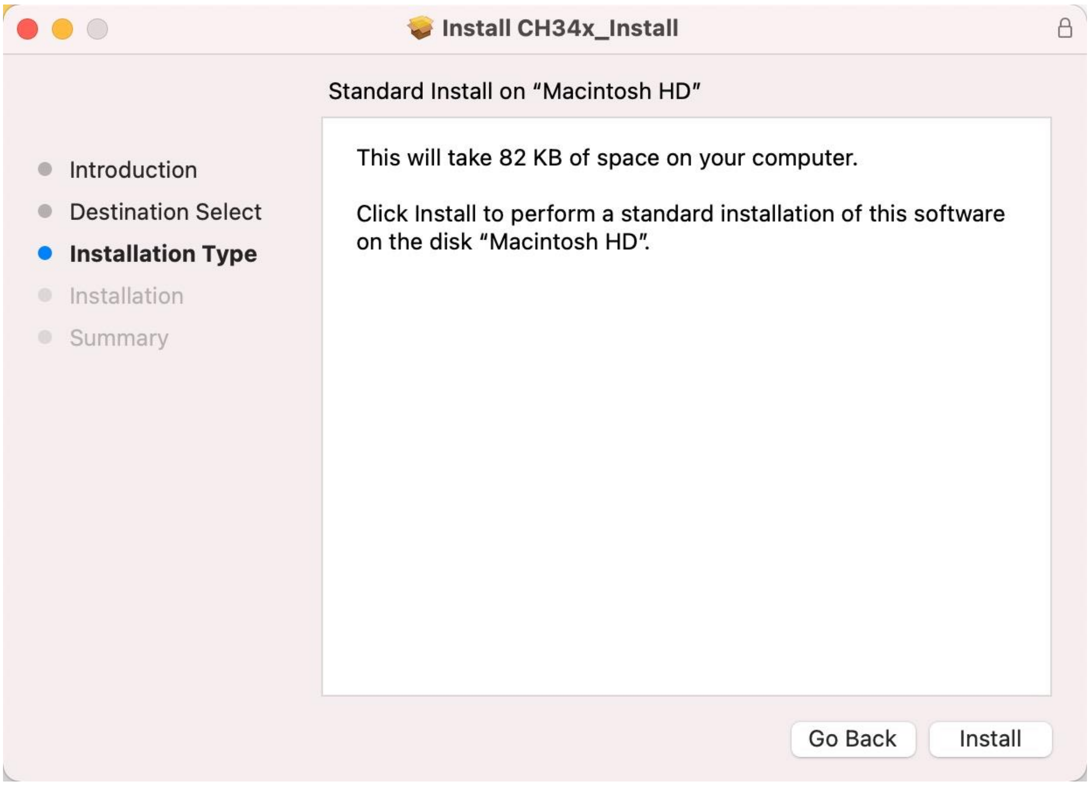
Second, open the folder "**Freenove_Ultimate_Starter_Kit_for_ESP8266/CH340/MAC/**"



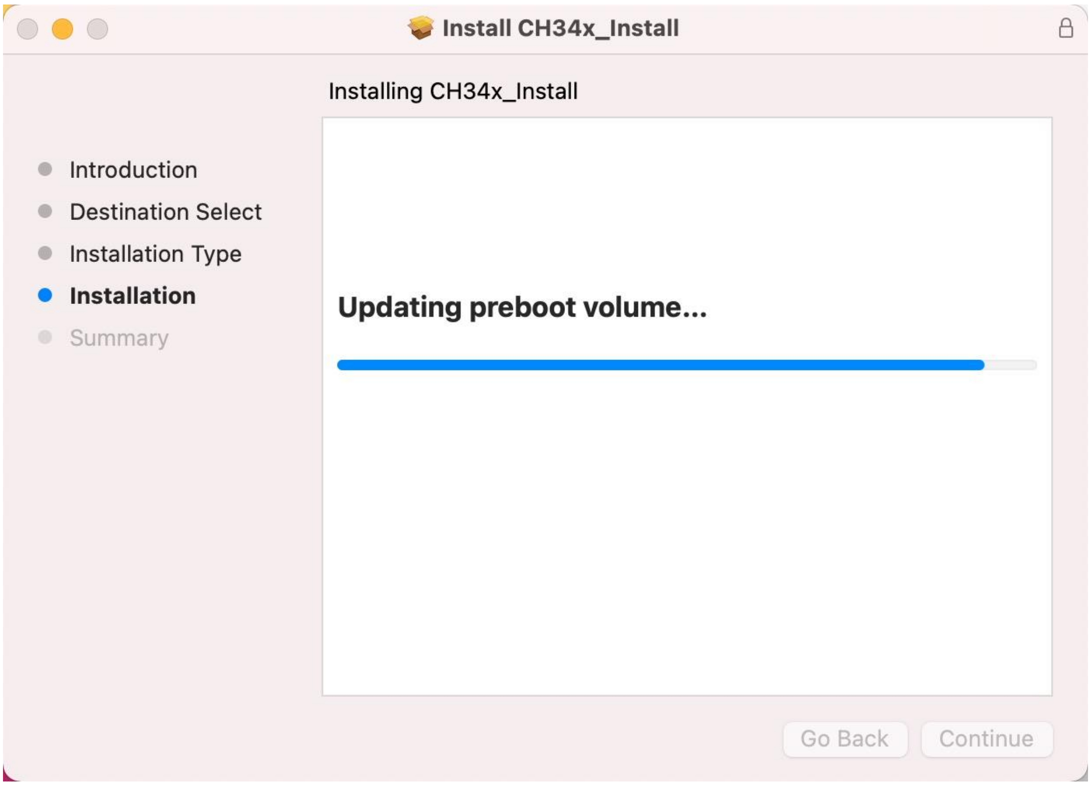
Third, click Continue.



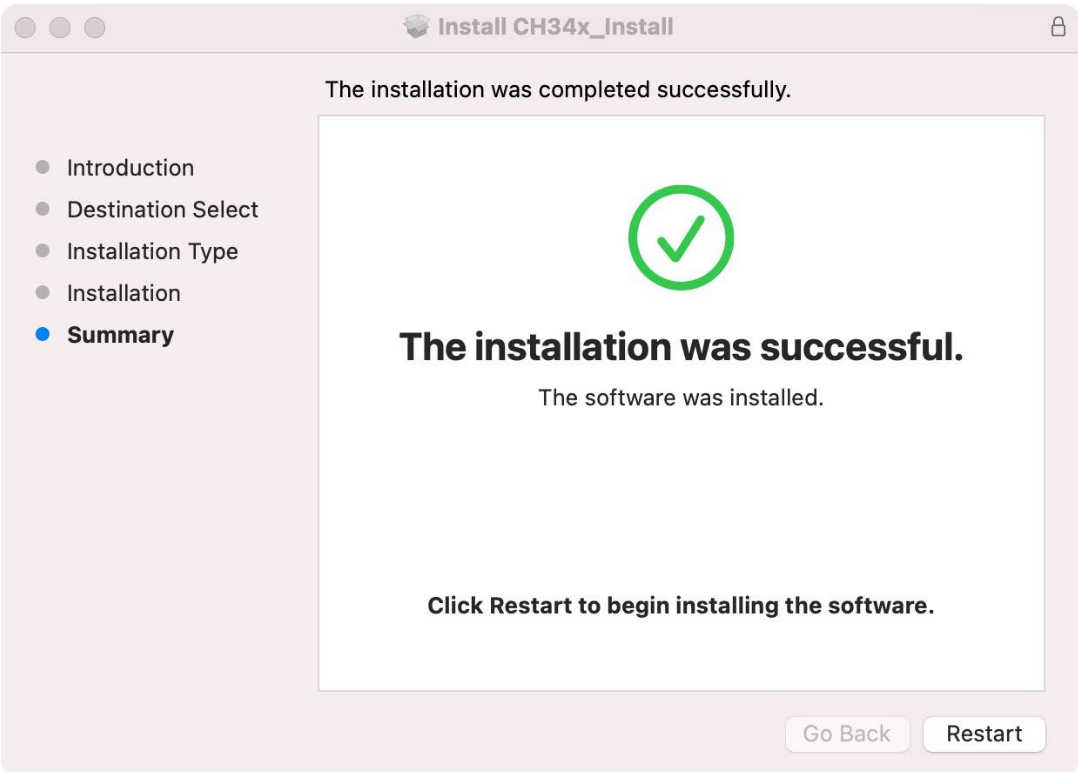
Fourth, click Install.


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

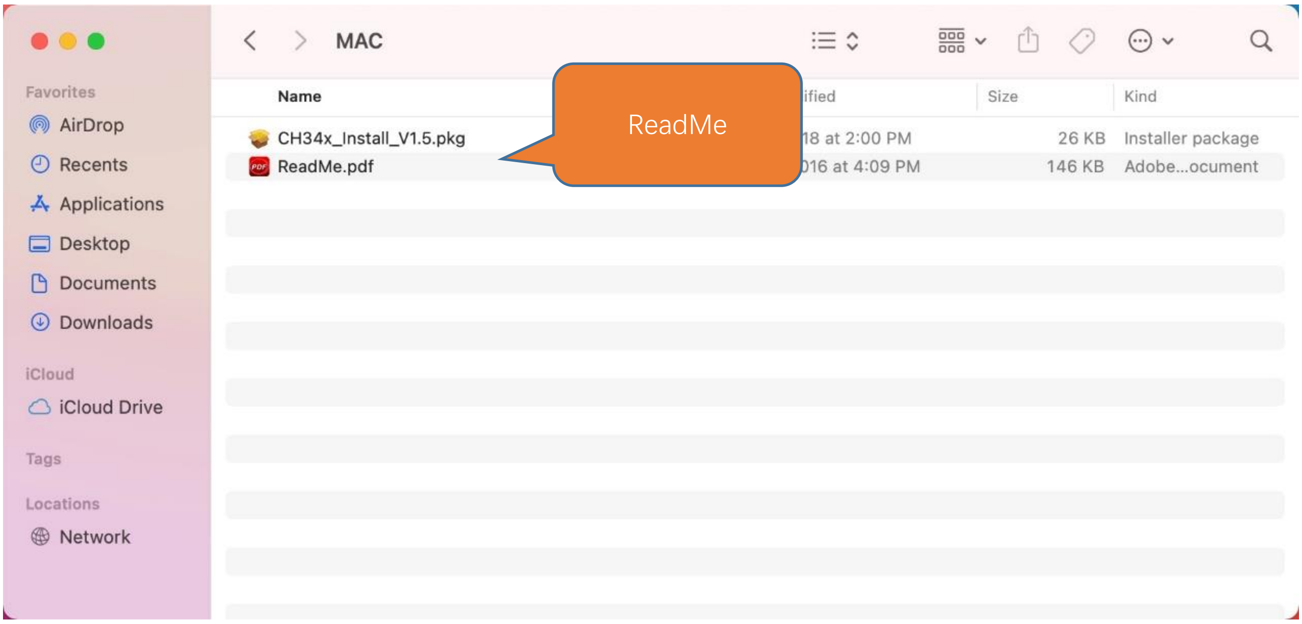
Then, waiting Finsh.



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

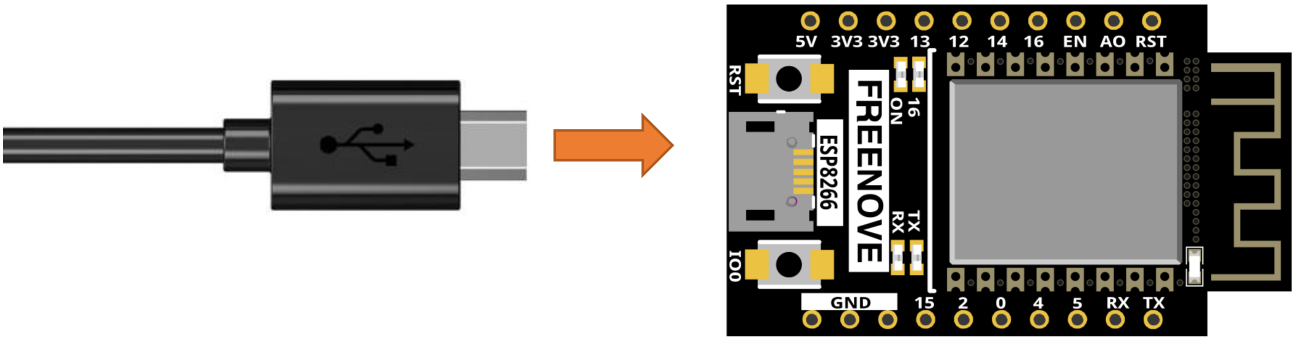
Releases

v1.18 (2022-01-17) .bin [.elf] [.map] [Release notes] (latest)
v1.17 (2021-09-02) .bin [.elf] [.map] [Release notes]
v1.16 (2021-06-18) .bin [.elf] [.map] [Release notes]
v1.15 (2021-04-18) .bin [.elf] [.map] [Release notes]
v1.14 (2021-02-02) .bin [.elf] [.map] [Release notes]
v1.13 (2020-09-11) .bin [.elf] [.map] [Release notes]
v1.12 (2019-12-20) .bin [.elf] [.map] [Release notes]
v1.11 (2019-05-29) .bin [.elf] [.map] [Release notes]
v1.10 (2019-01-25) .bin [.elf] [.map] [Release notes]
v1.9.4 (2018-05-11) .bin [.elf] [.map] [Release notes]
v1.9.3 (2017-11-01) .bin [.elf] [.map] [Release notes]
v1.9.2 (2017-08-23) .bin [.elf] [.map] [Release notes]
v1.9.1 (2017-06-12) .bin [.elf] [.map] [Release notes]
v1.9 (2017-05-26) .bin [.elf] [.map] [Release notes]
v1.8.7 (2017-01-08) .bin [.elf] [.map] [Release notes]

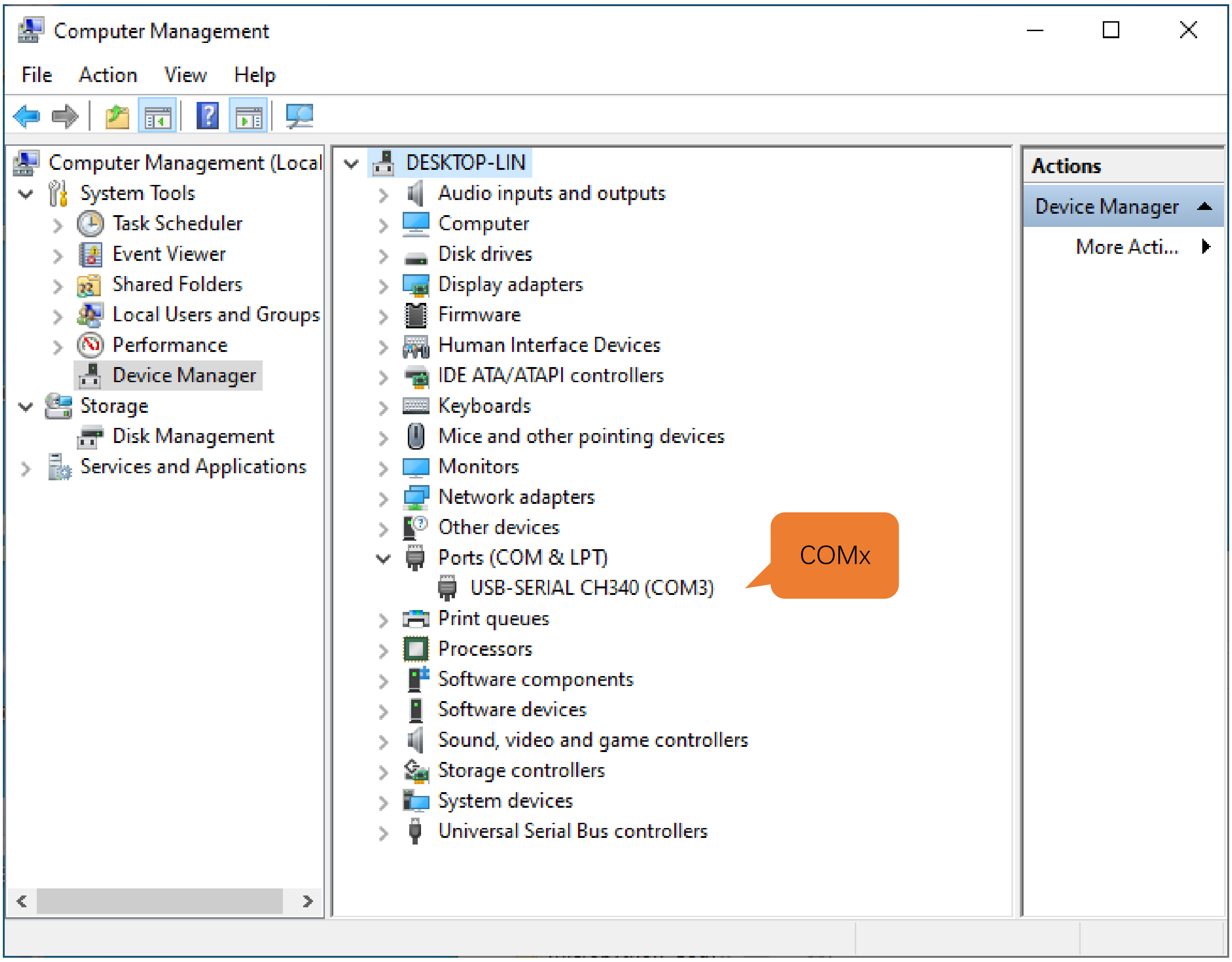
Firmware used in this tutorial is **esp8266-20220117-v1.18.bin**
Click the following link to download directly:
https://micropython.org/resources/firmware/esp8266-20220117-v1.18.bin
This file is also provided in our data folder "**Freenove_Ultimate_Starter_Kit_for_ESP8266 /Python/Python_Firmware**".

Burning a Micropython Firmware

Connect your computer and ESP8266 with a USB cable.

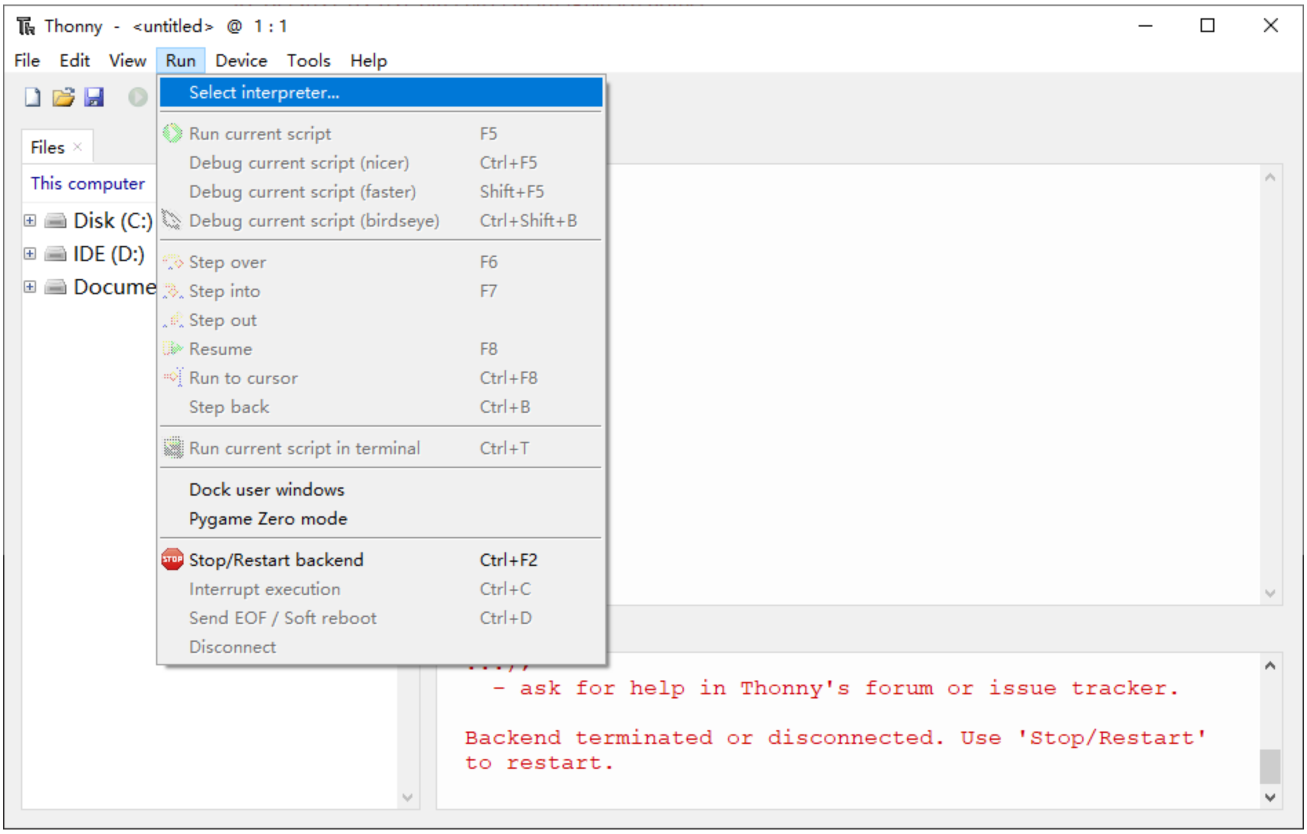


Make sure that the driver has been installed successfully and that it can recognize COM port correctly. Open device manager and expand “Ports”.

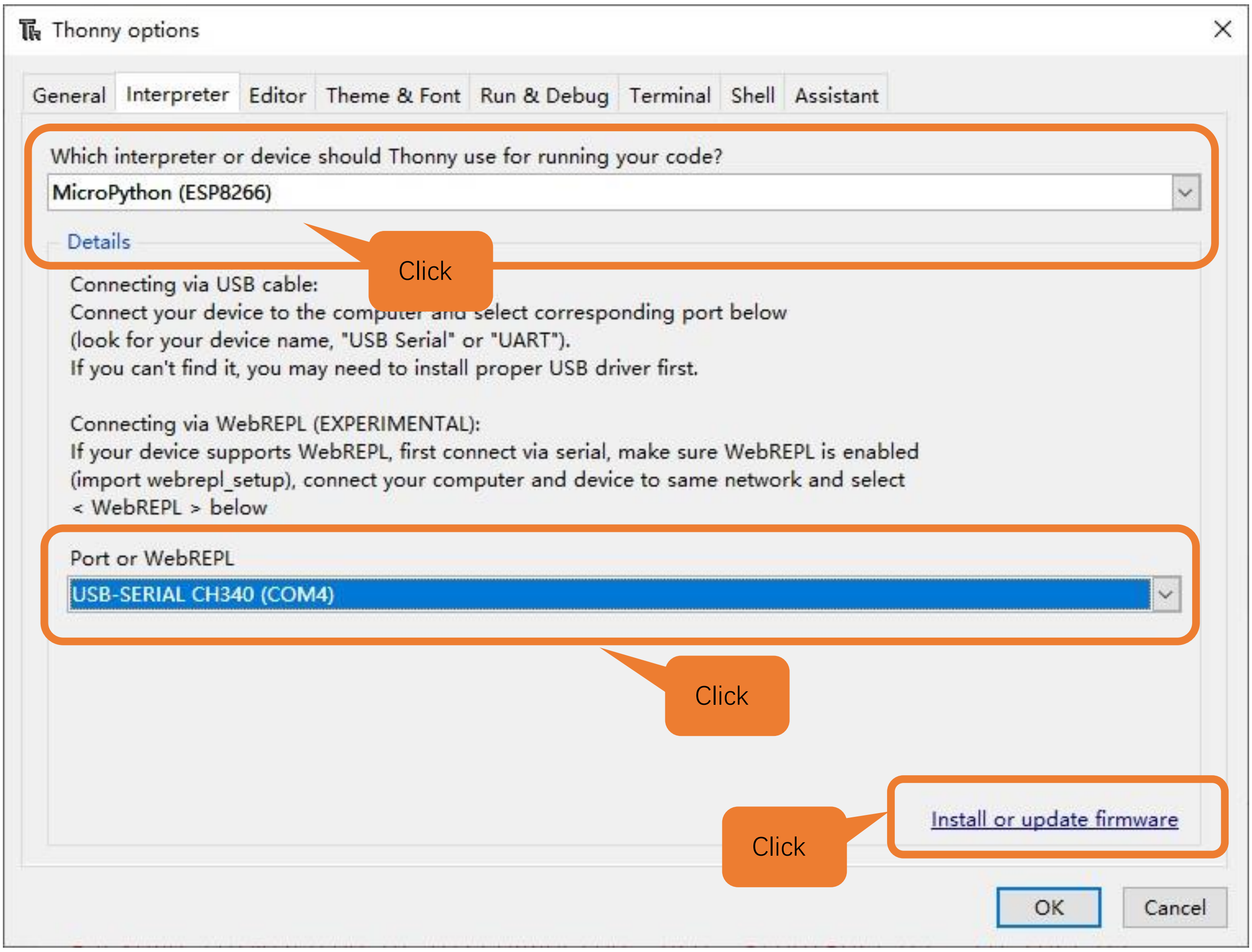


Note: the port of different people may be different, which is a normal situation.

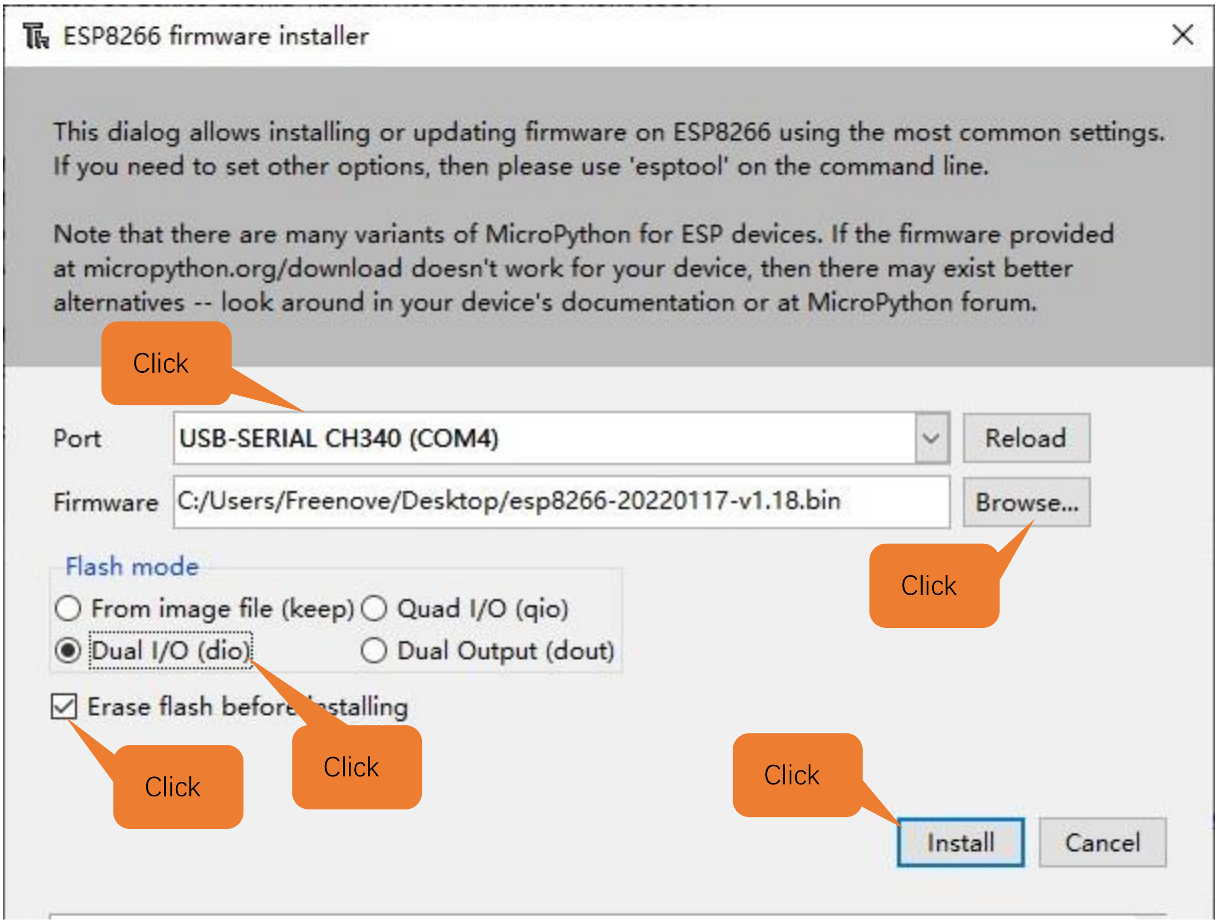
1. Open Thonny, click “run” and select “Select interpreter…””



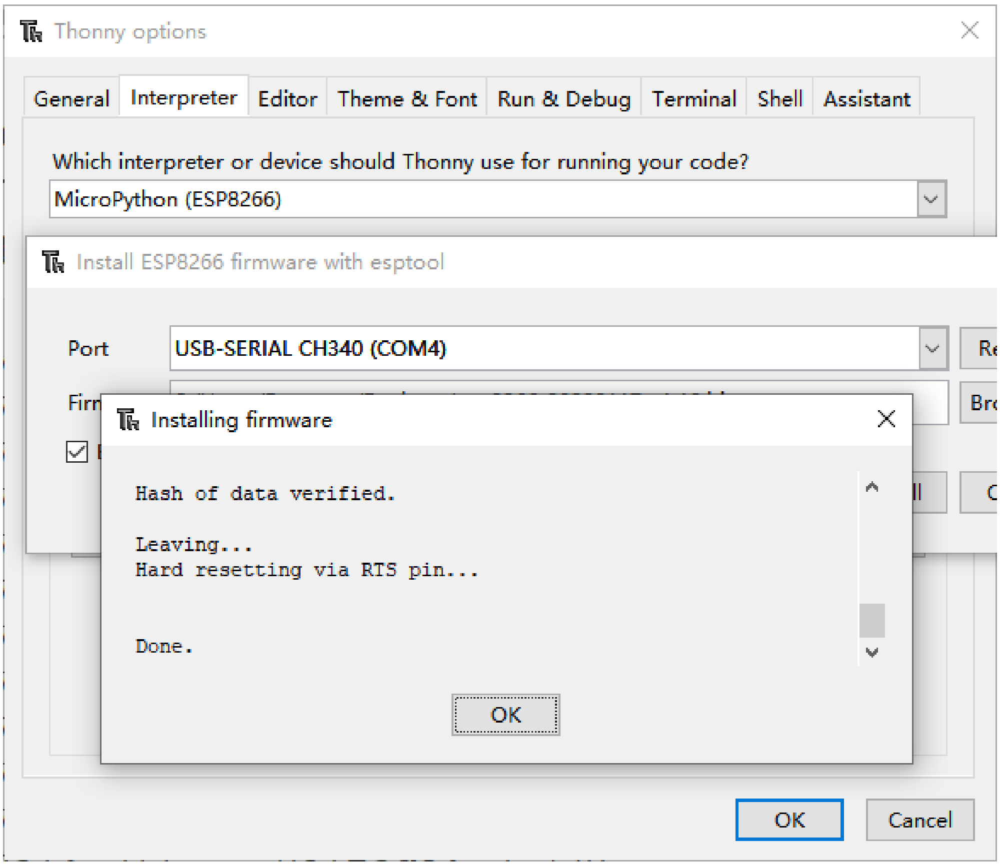
2. Select "Micropython (ESP8266)", select "USB-SERIAL CH340 (COM4)", and then click the long button under "Firmware".

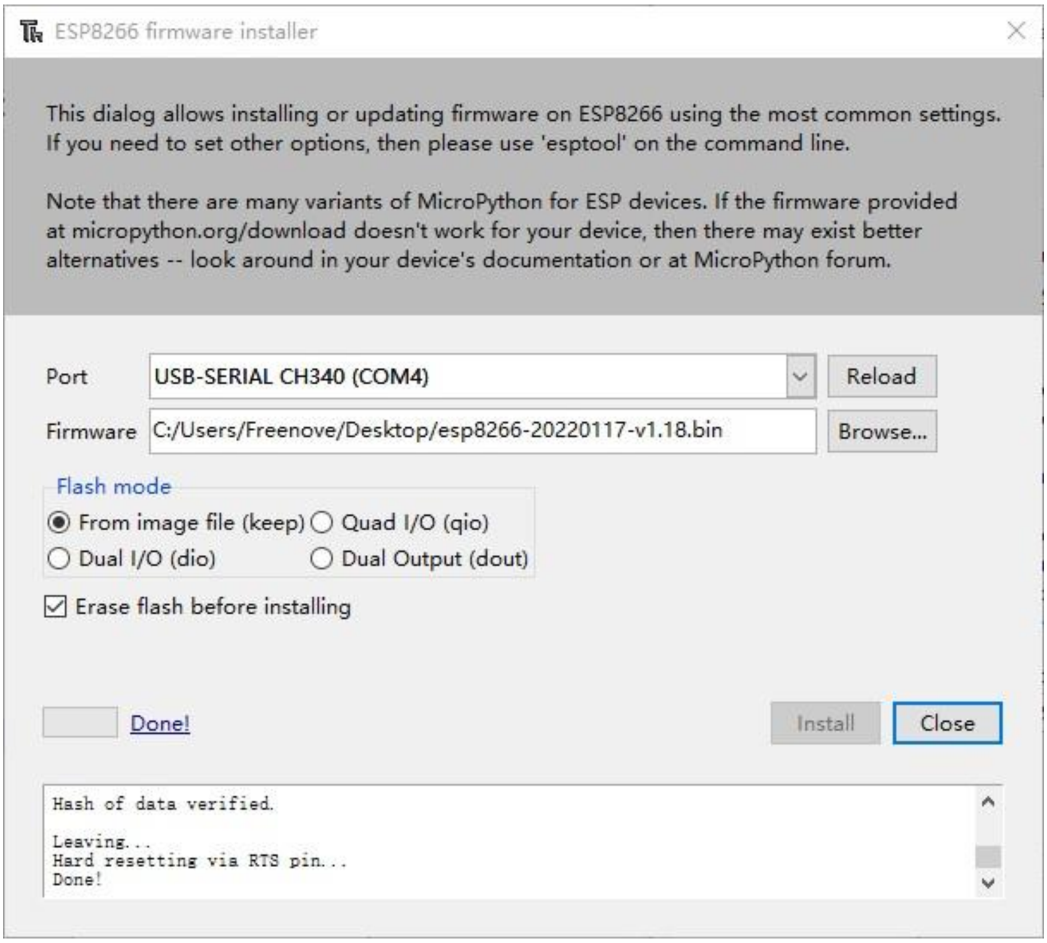


3. The following dialog box pops up. Select "USB-SERIAL CH340 (COM4)" for "Port" and then click "Browse...". Select the previous prepared microPython firmware "**esp8266-20220117-v1.18.bin**". Check "Erase flash before installing" and click "install" to wait for the prompt of finishing installation.
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.

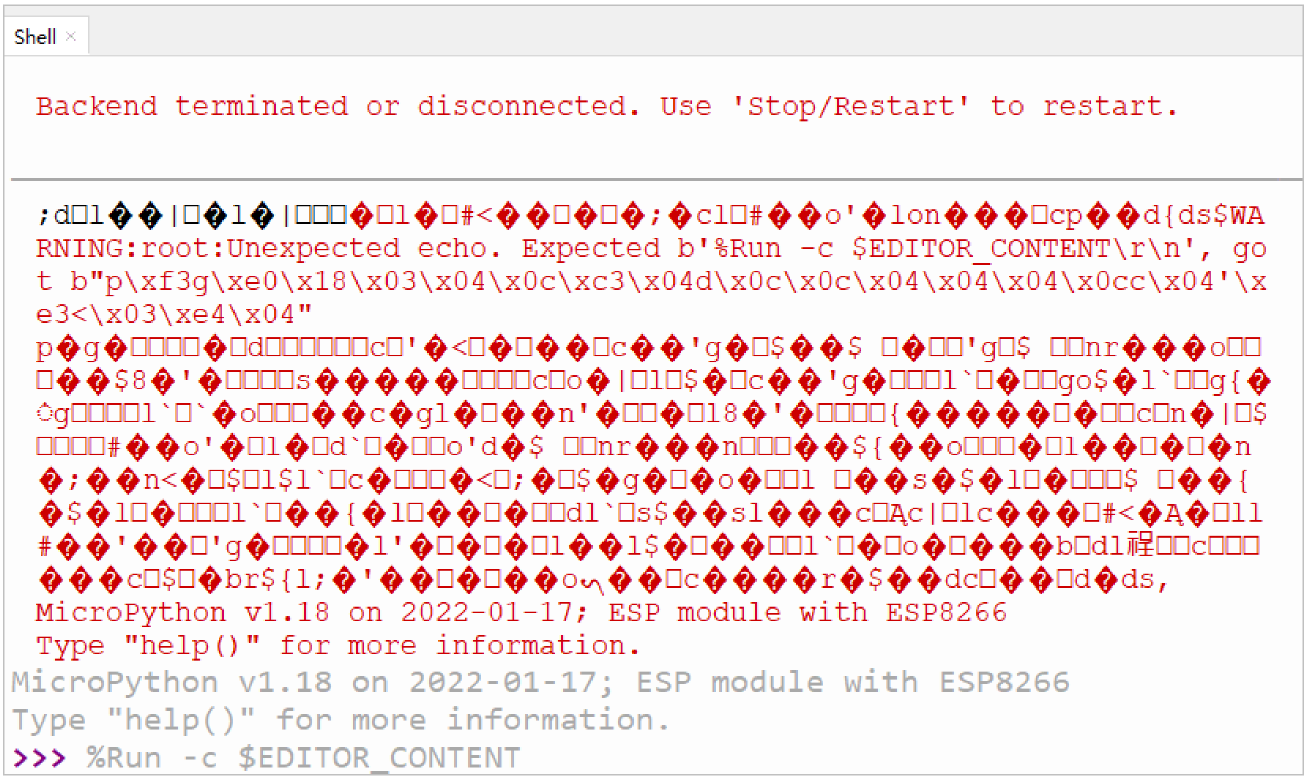


4. Wait for the installation to be done.

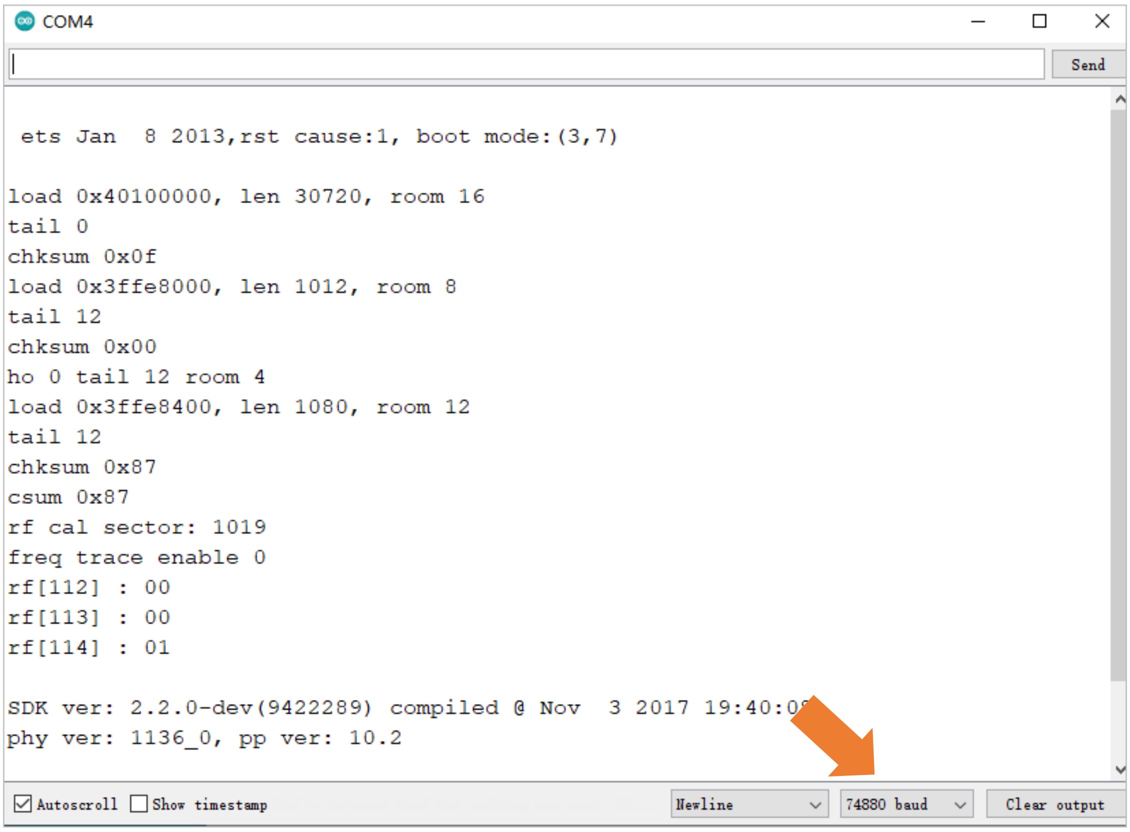




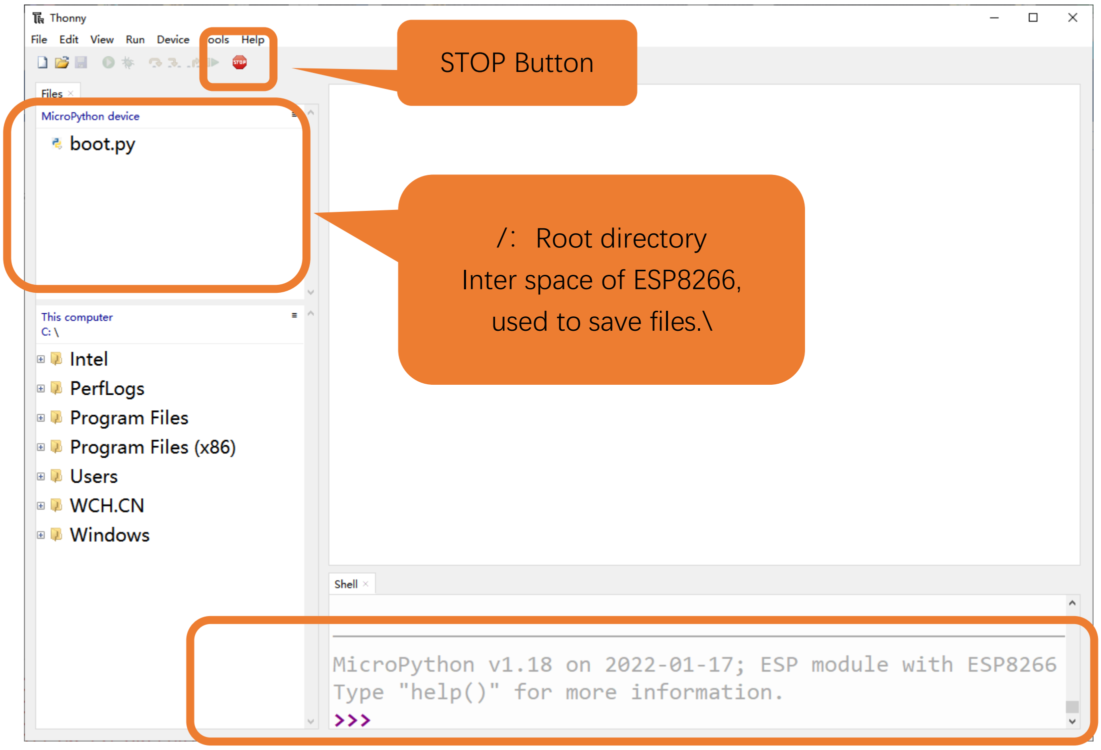
After burning the Micropython firmware, "shell" will display some garbled characters, please do not worry, the garbled characters are displayed as follows:



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.

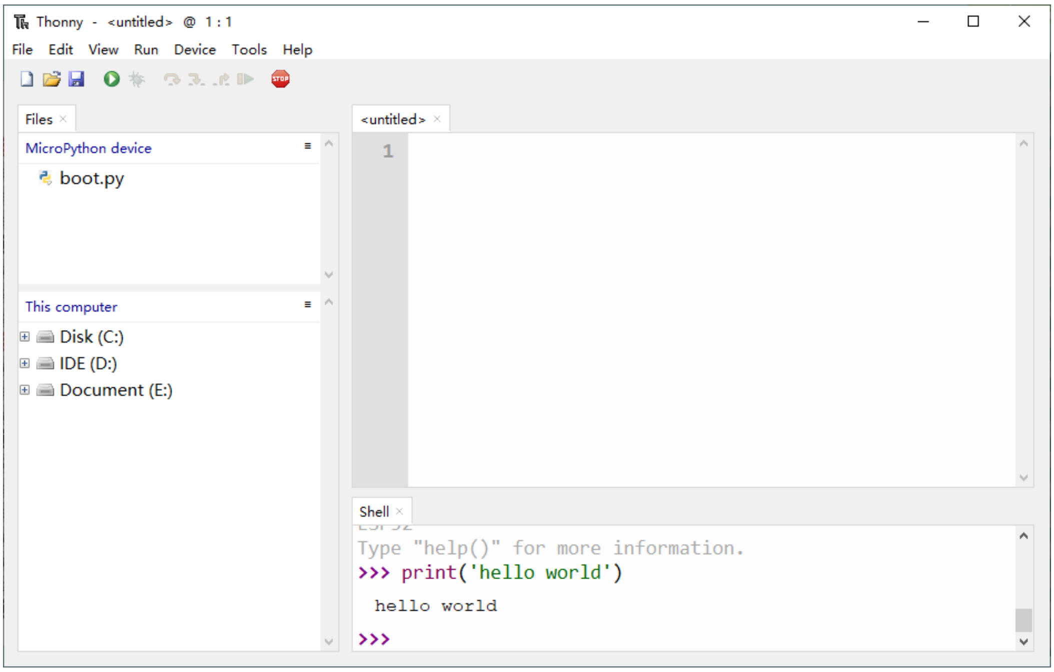


6. So far, all the preparations have been made.

0.5 Testing codes (Important)

Testing Shell Command

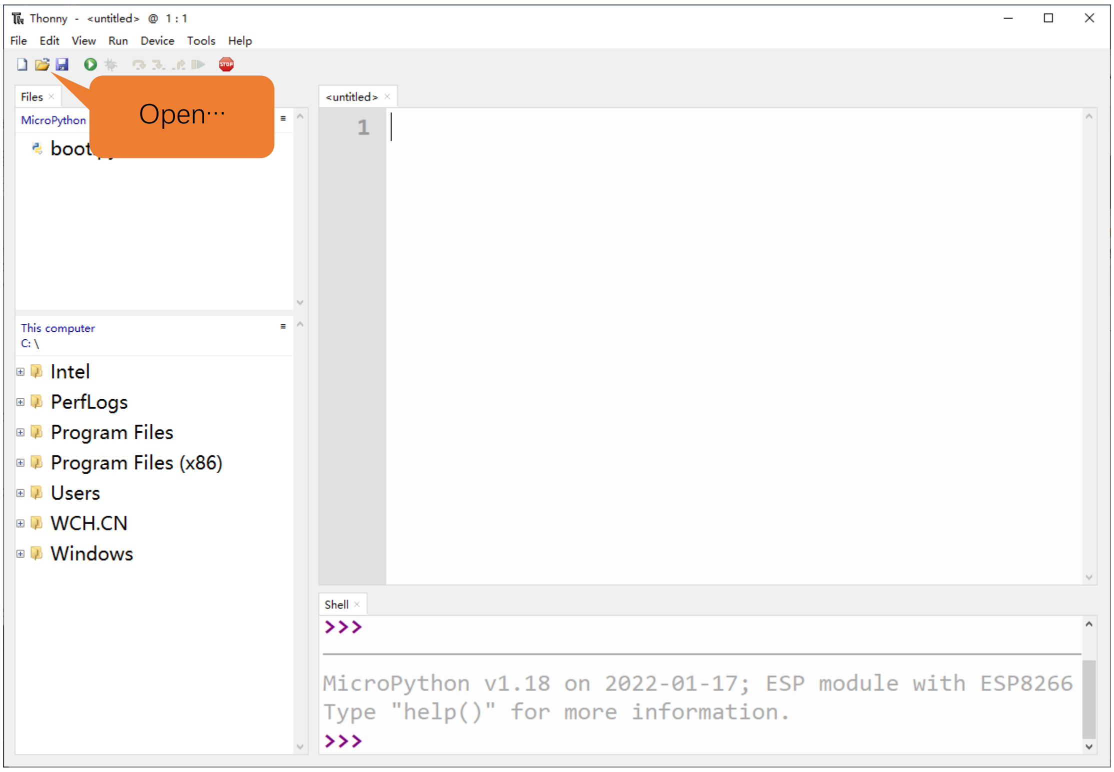
Enter "print('hello world')" in "Shell" and press Enter.



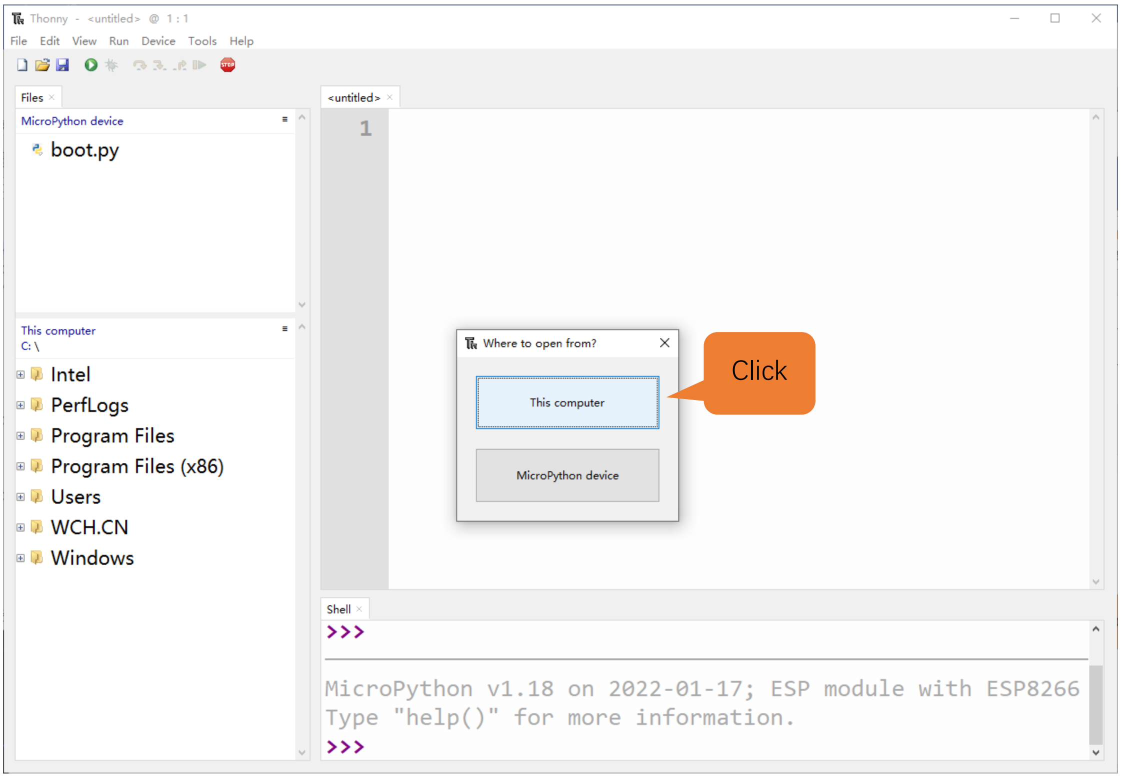
Running Online

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

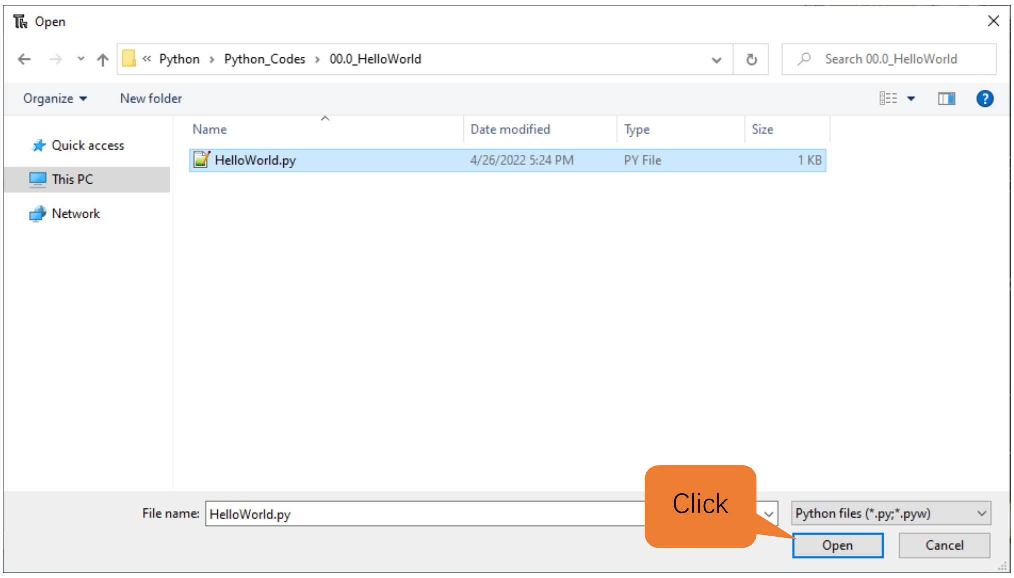
1. Open Thonny and click "Open…".



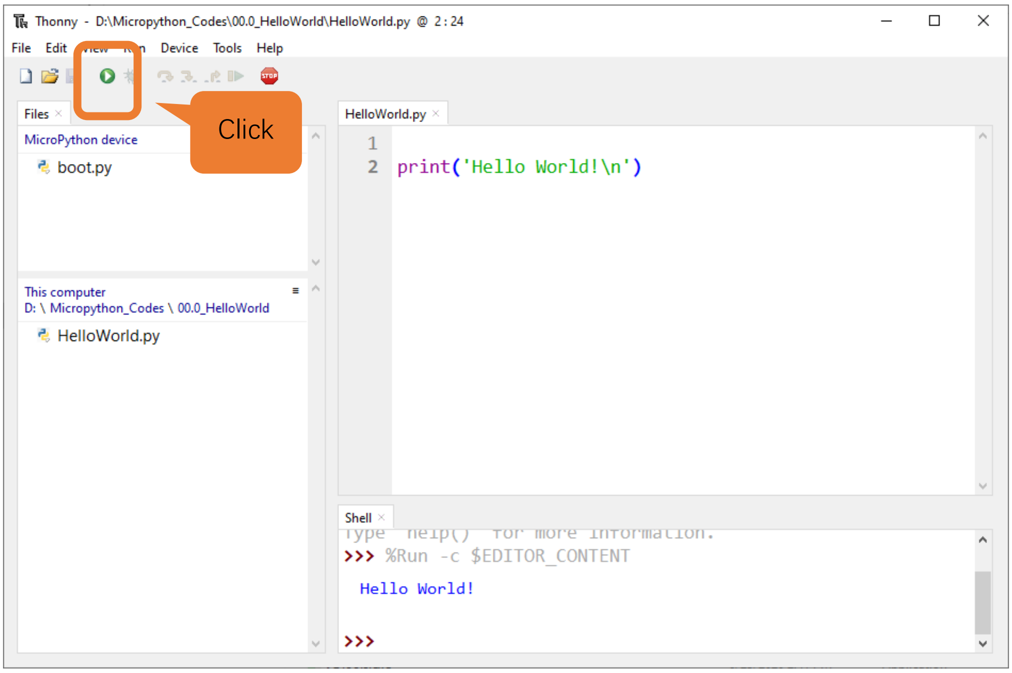
2. On the newly pop-up window, click "This computer".



In the new dialog box, select "**HelloWorld.py**" in "**Freenove_Ultimate_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".

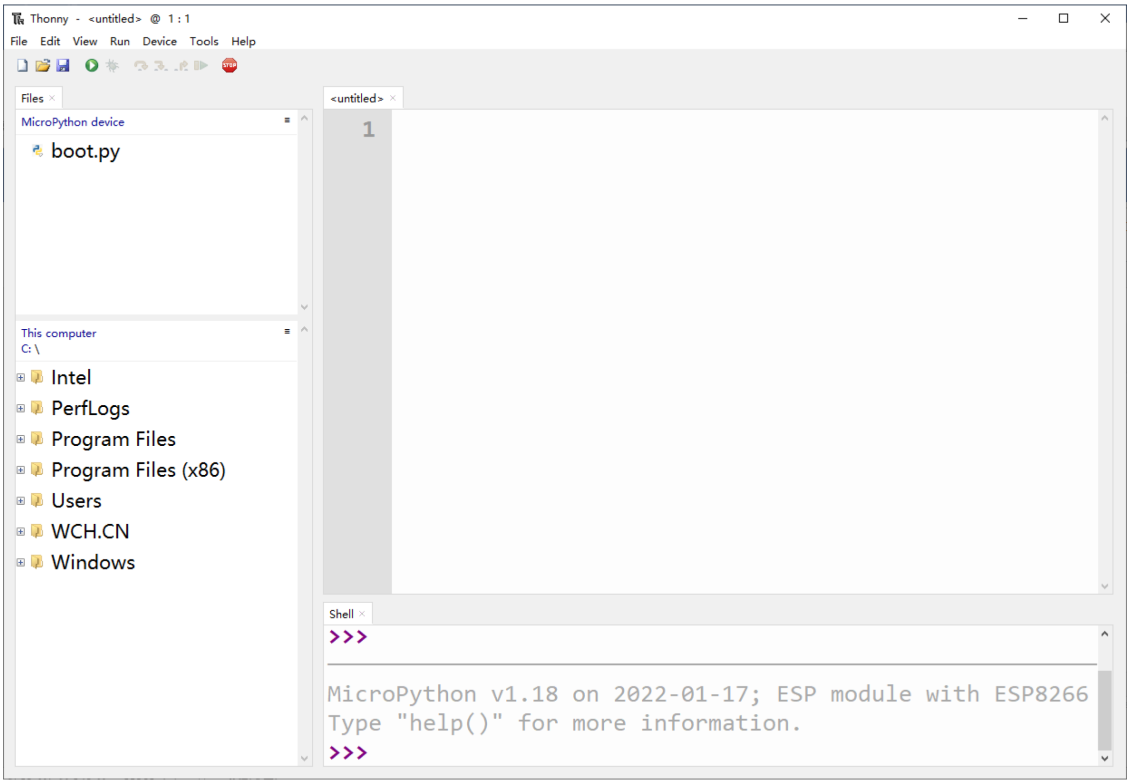


Note: When running online, if you press the reset key of ESP8266, user's code will not be executed again. If you wish to run the code automatically after resetting the code, please refer to the following Running Offline.

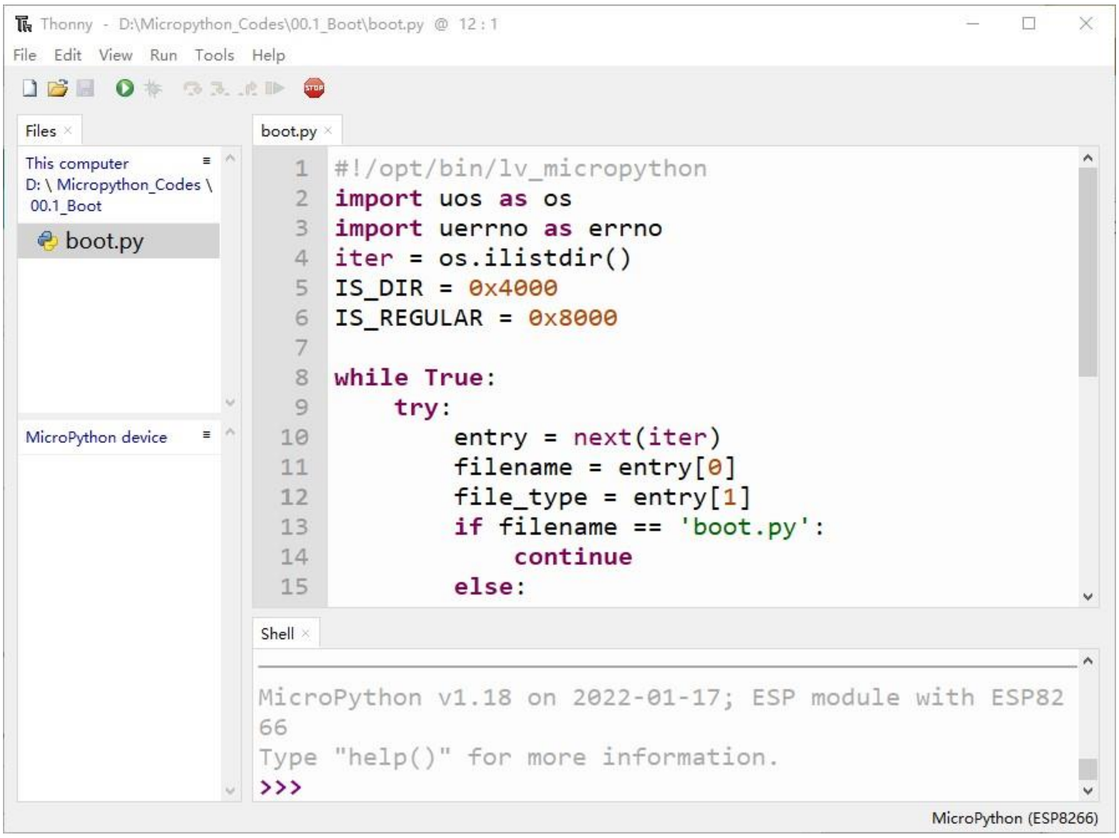
Running Offline (Important)

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.

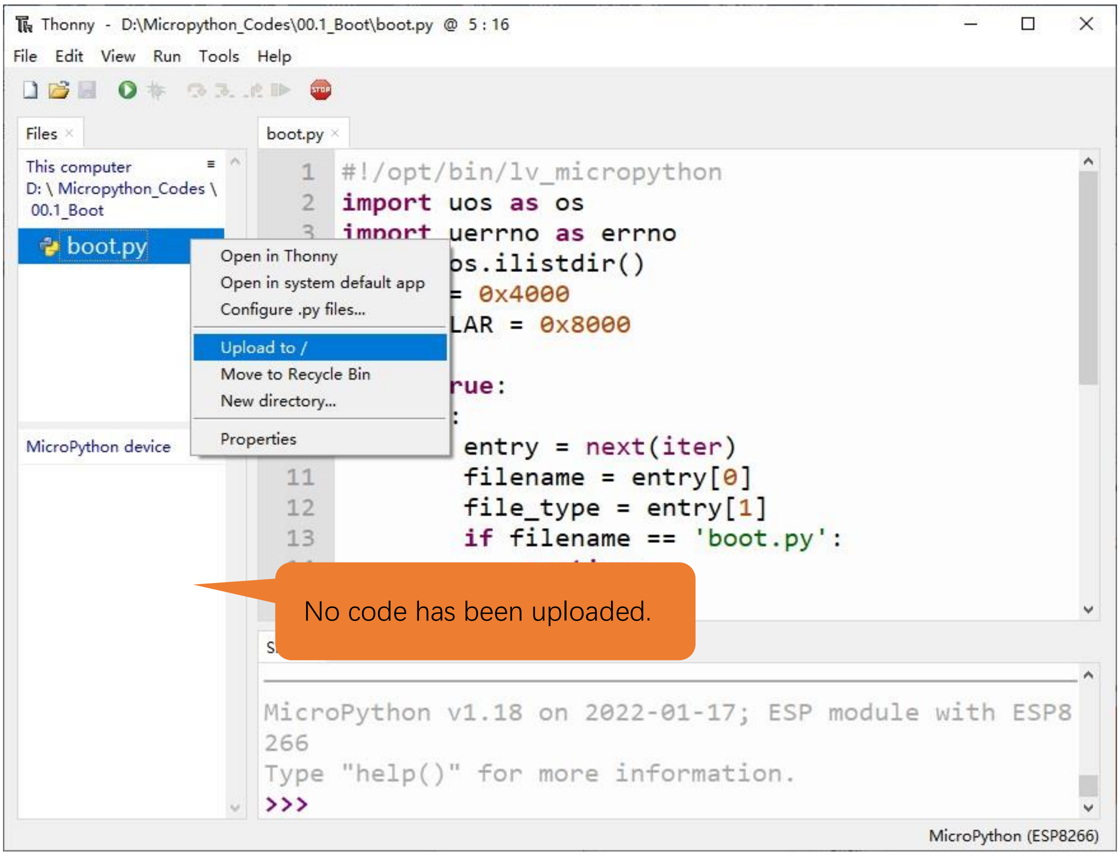
1. Move the program folder “**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**” to disk(D) in advance with the path of “**D:/Micropython_Codes**”. Open “Thonny”.



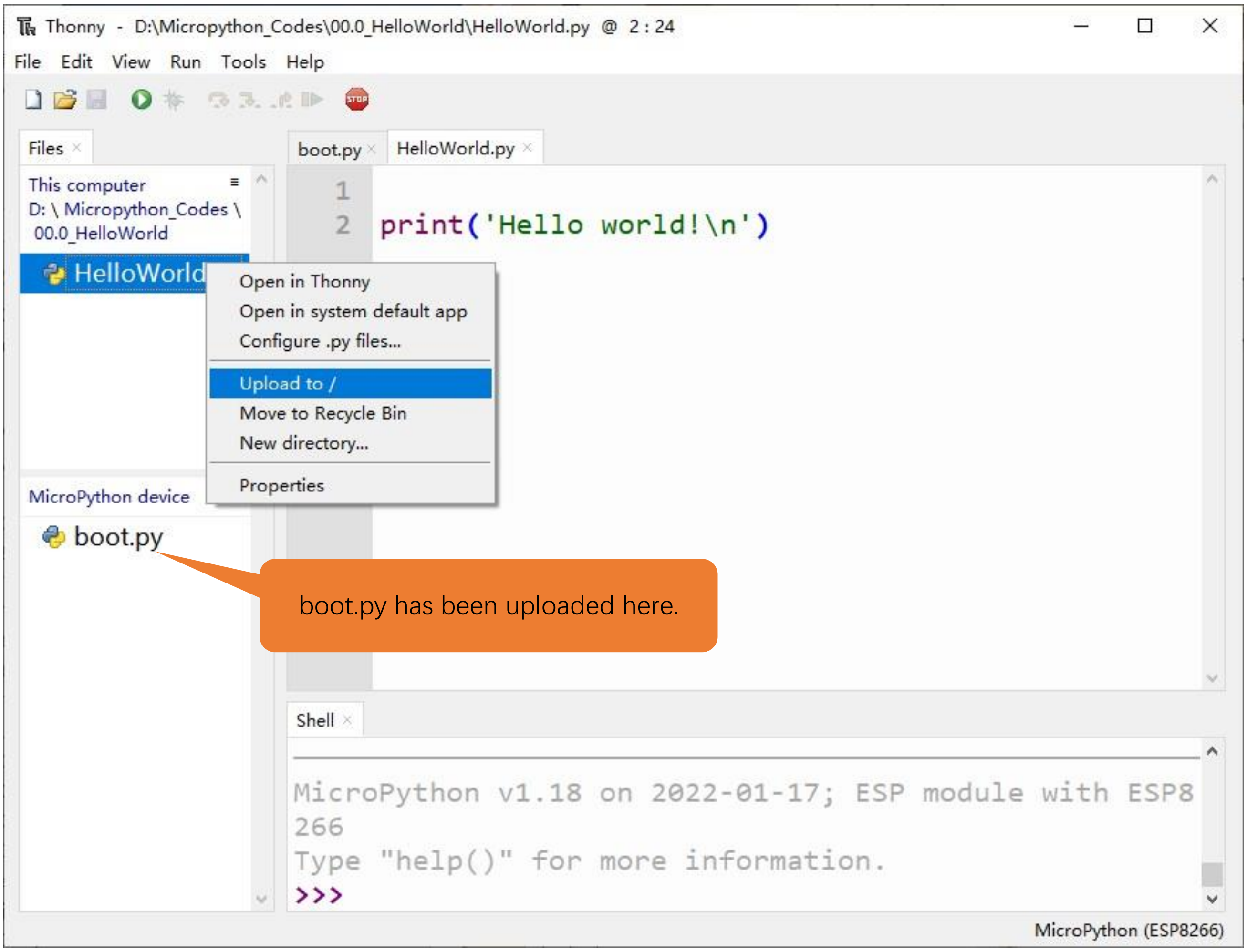
2. Expand “00.1_Boot” in the “Micropython_Codes” in the directory of disk(D), and double-click boot.py, which is provided by us to enable programs in “MicroPython device” to run offline.



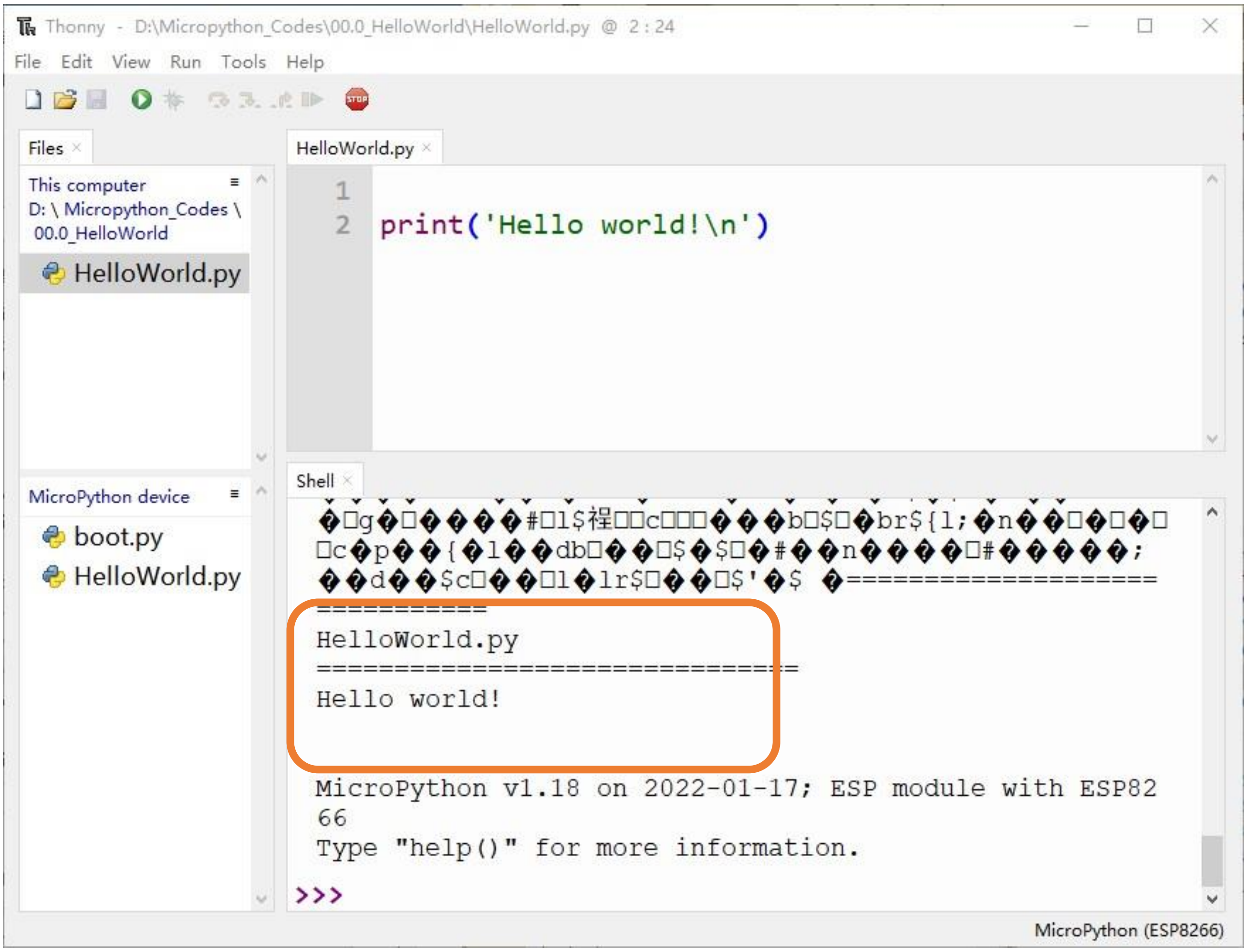
If you want your written programs to run offline, you need to upload boot.py we provided and all your codes to "MicroPython device" and press ESP8266's reset key. Here we use programs 00.0 and 00.1 as examples. Select "boot.py", right-click to select "Upload to /".



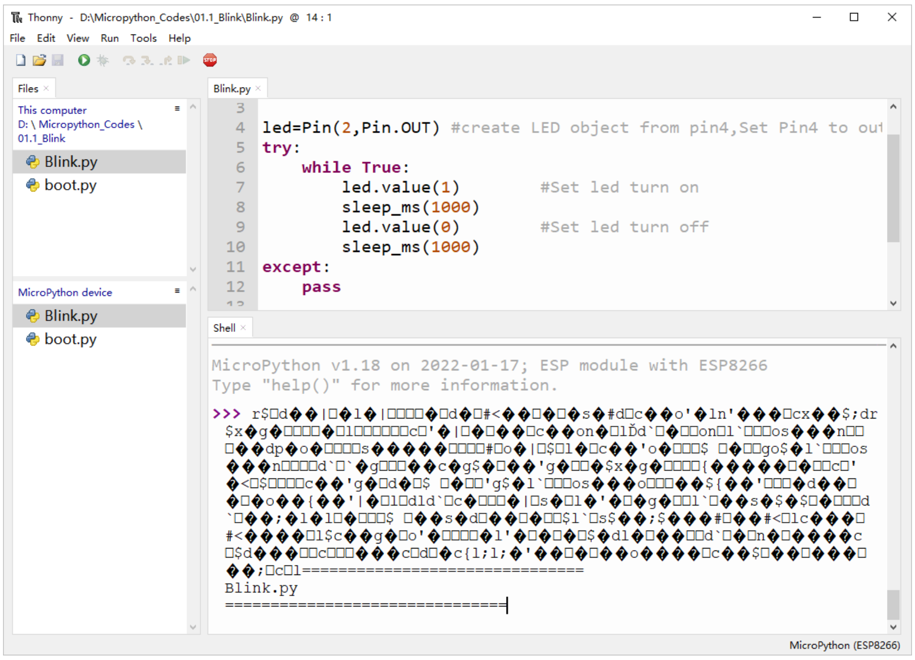
Similarly, upload “HelloWorld.py” to “MicroPython device”.



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

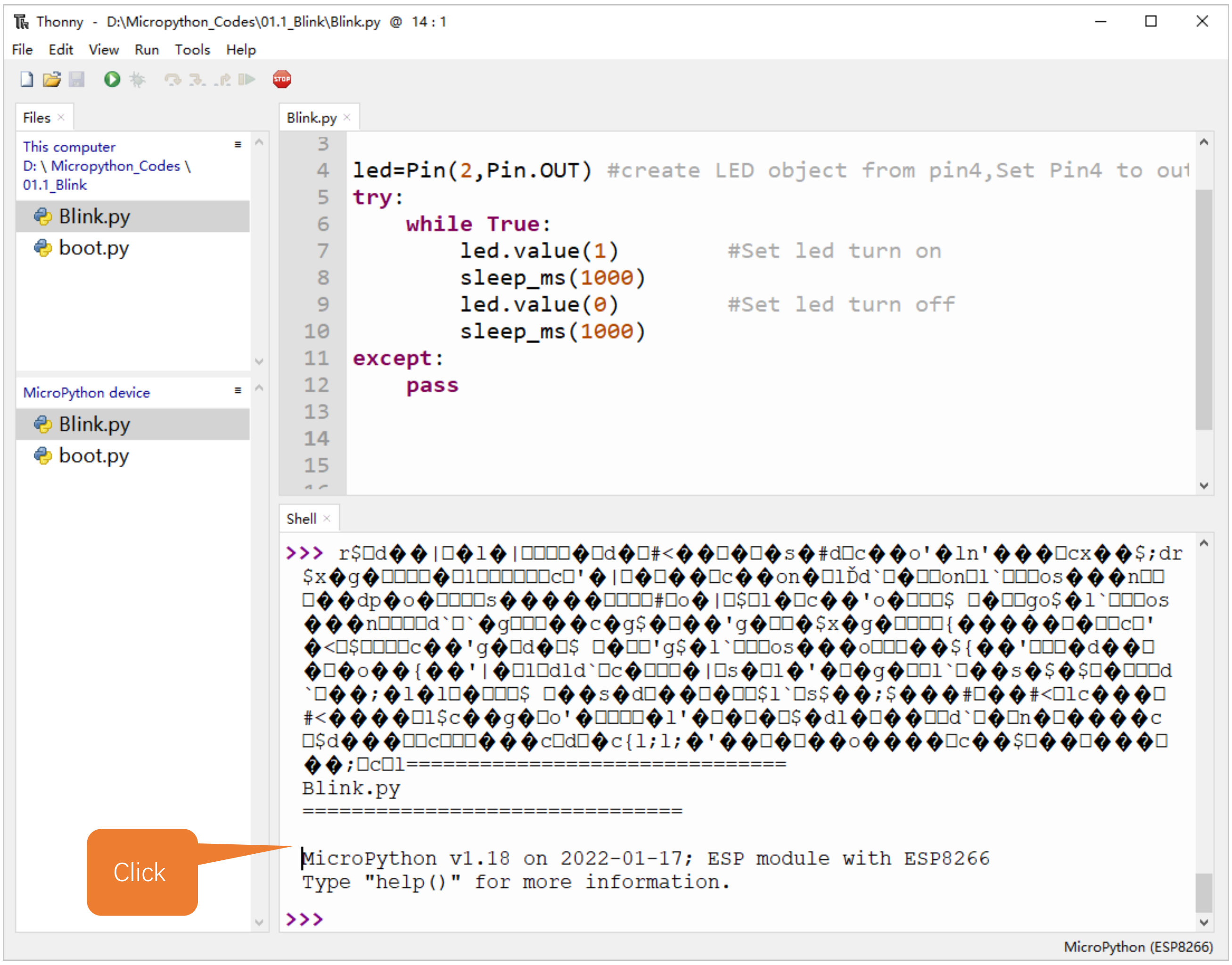


When you press the Reset key to run the offline code, the program will continue to execute while the ESP8266 is powered on.

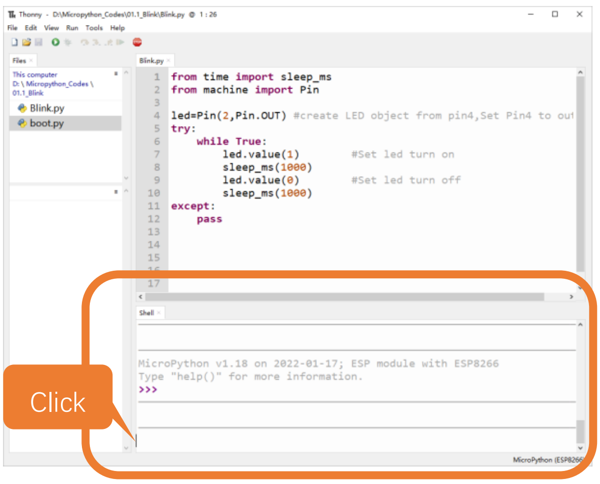


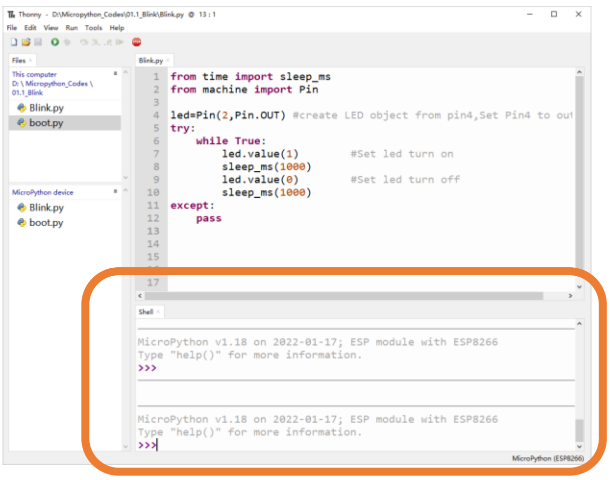
When you run offline code, you can exit the running program by pressing "CTRL" and "C" at the same time.

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.



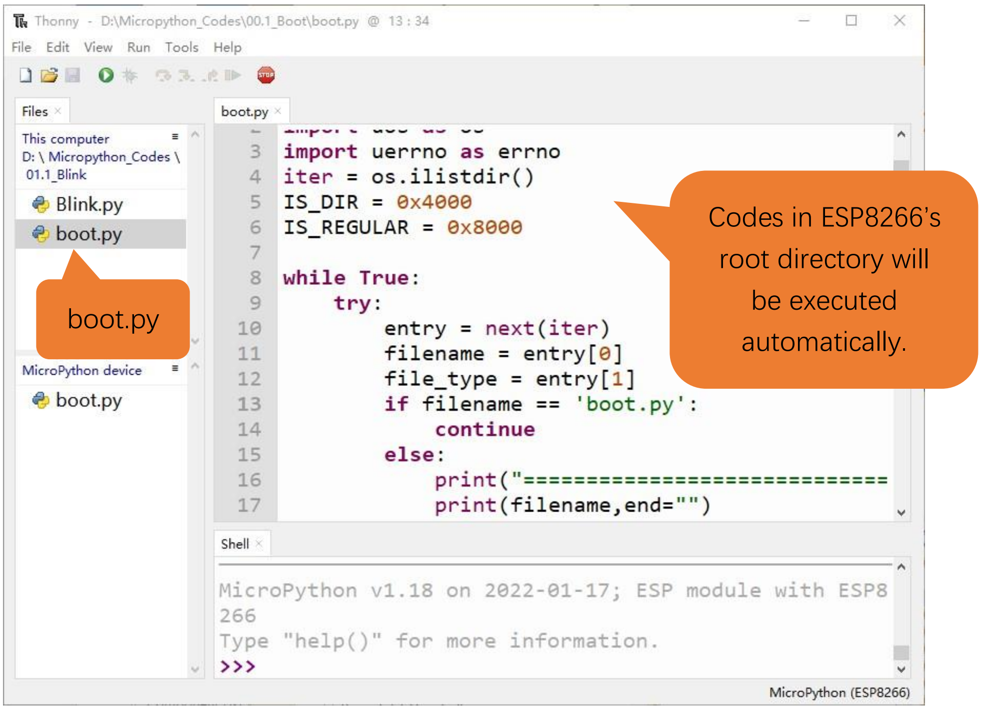


If the ESP8266 does not work properly, you can press CTRL and C at the same time to observe whether the Shell responds. If the ESP8266 still does not work properly, you can also rewrite the Micropython firmware and perform related operations again.

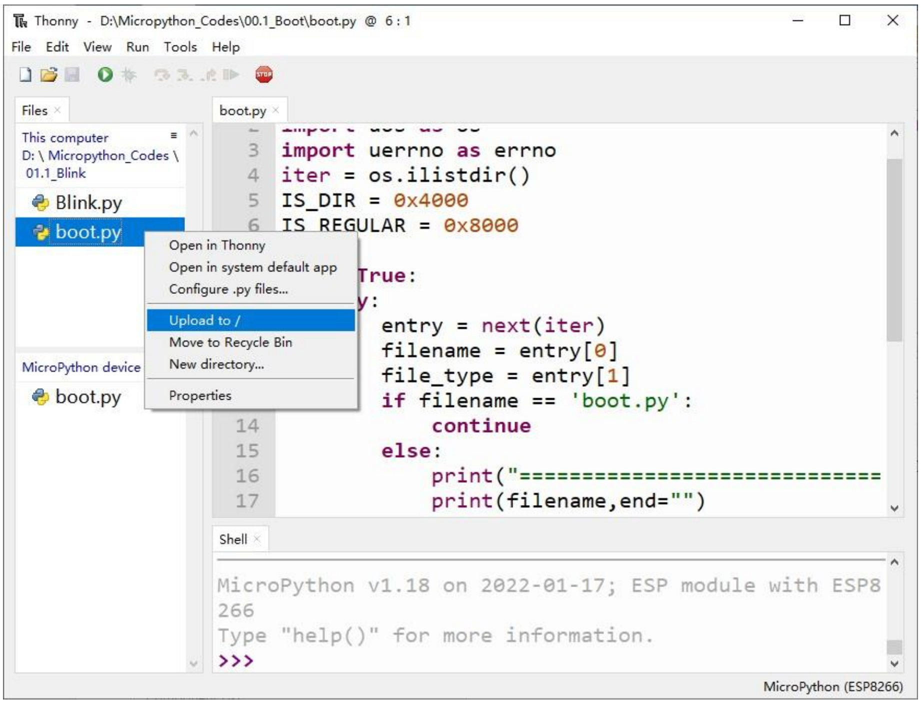
0.6 Thonny Common Operation

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.

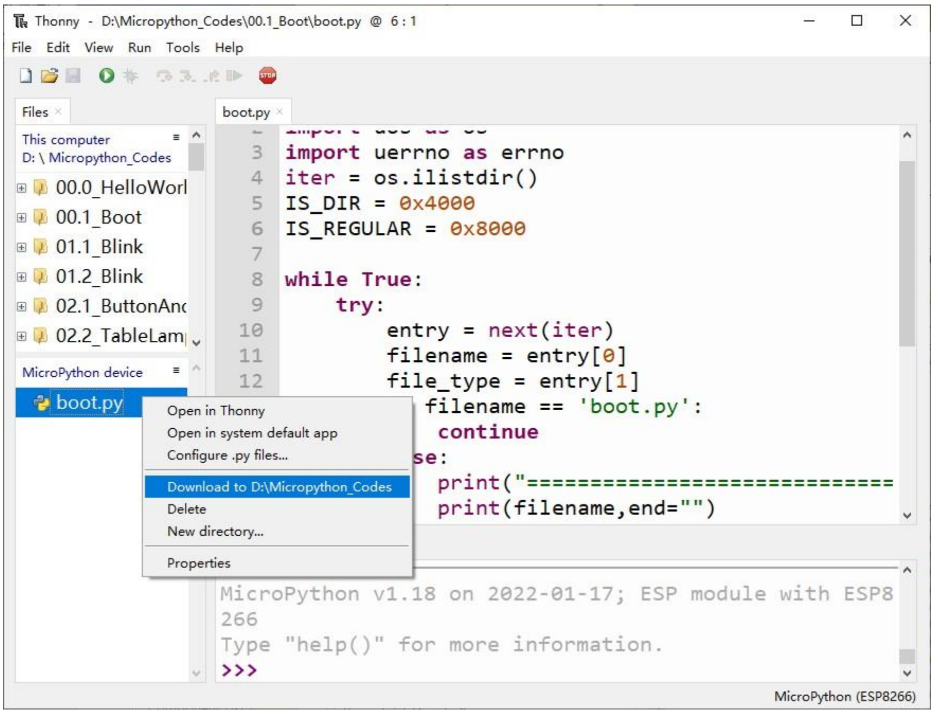


Select “Blink.py” in “01.1_Blink”, right-click your mouse and select “Upload to /” to upload code to ESP8266’s root directory.



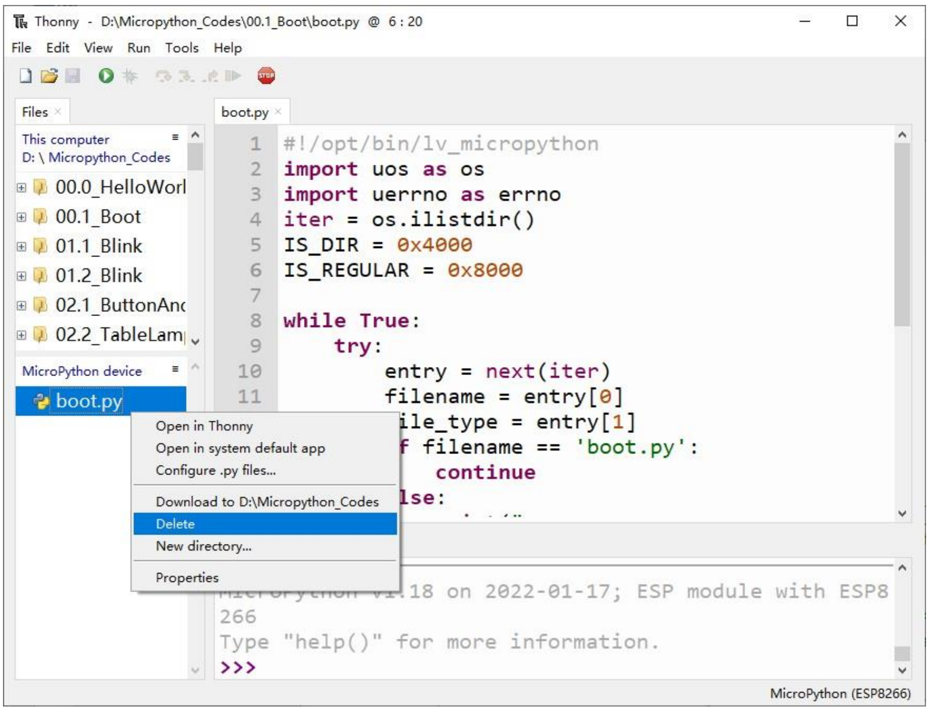
Downloading Code to Computer

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



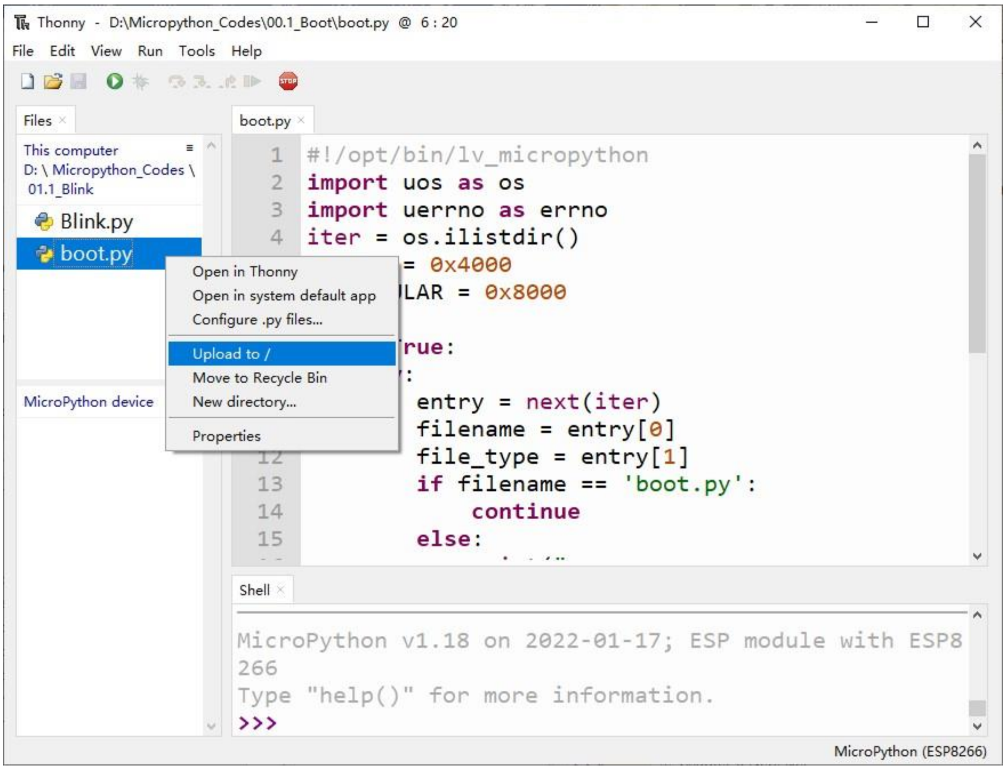
Deleting Files from ESP8266’s Root Directory

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



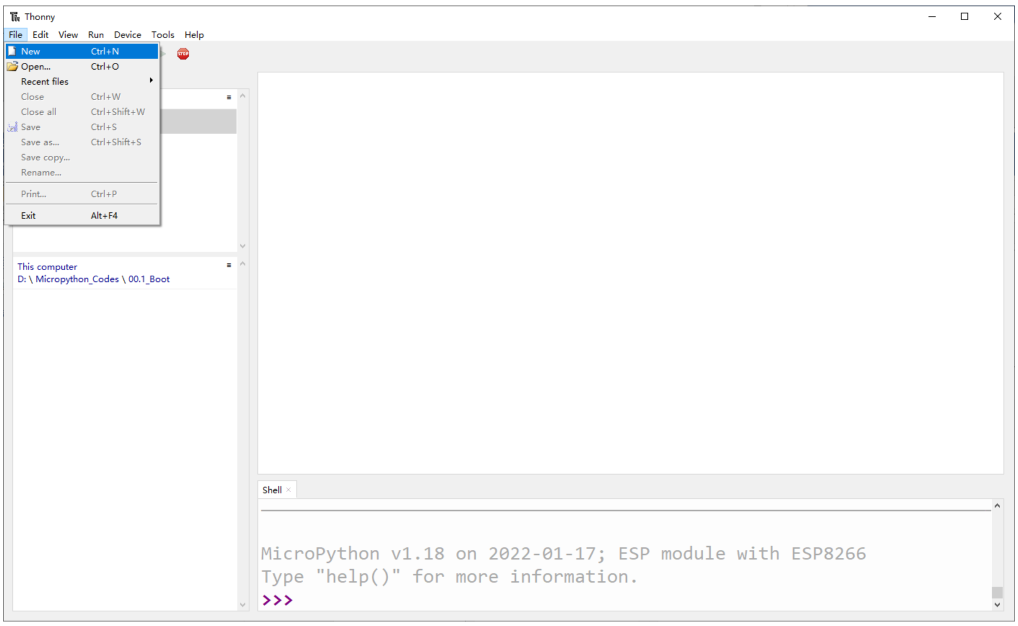
Deleting Files from your Computer Directory

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

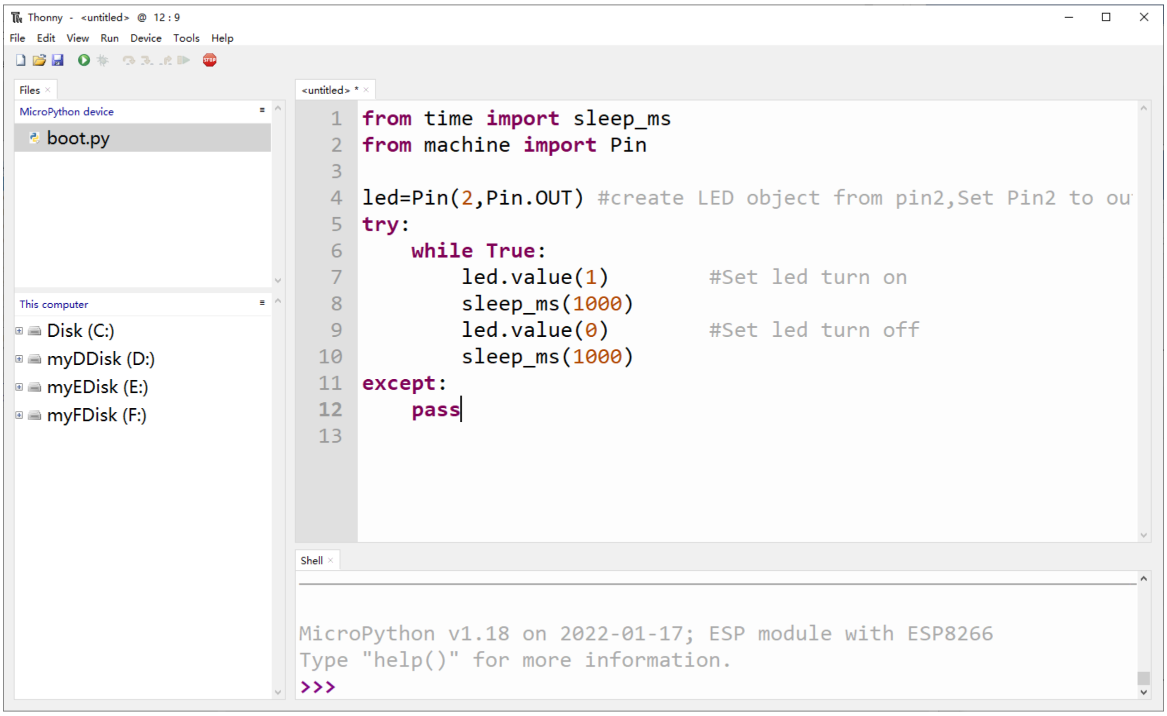


Creating and Saving the code

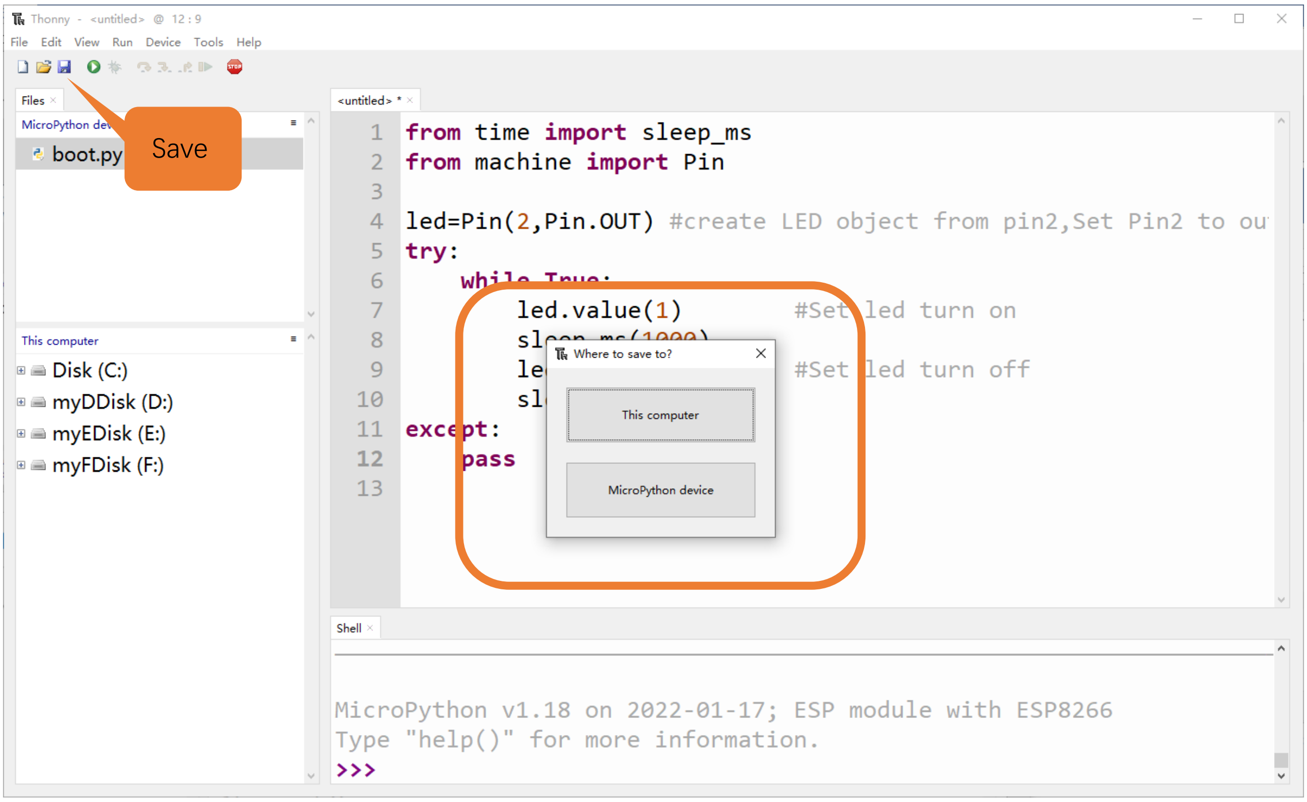
Click "File"→"New" to create and write codes.



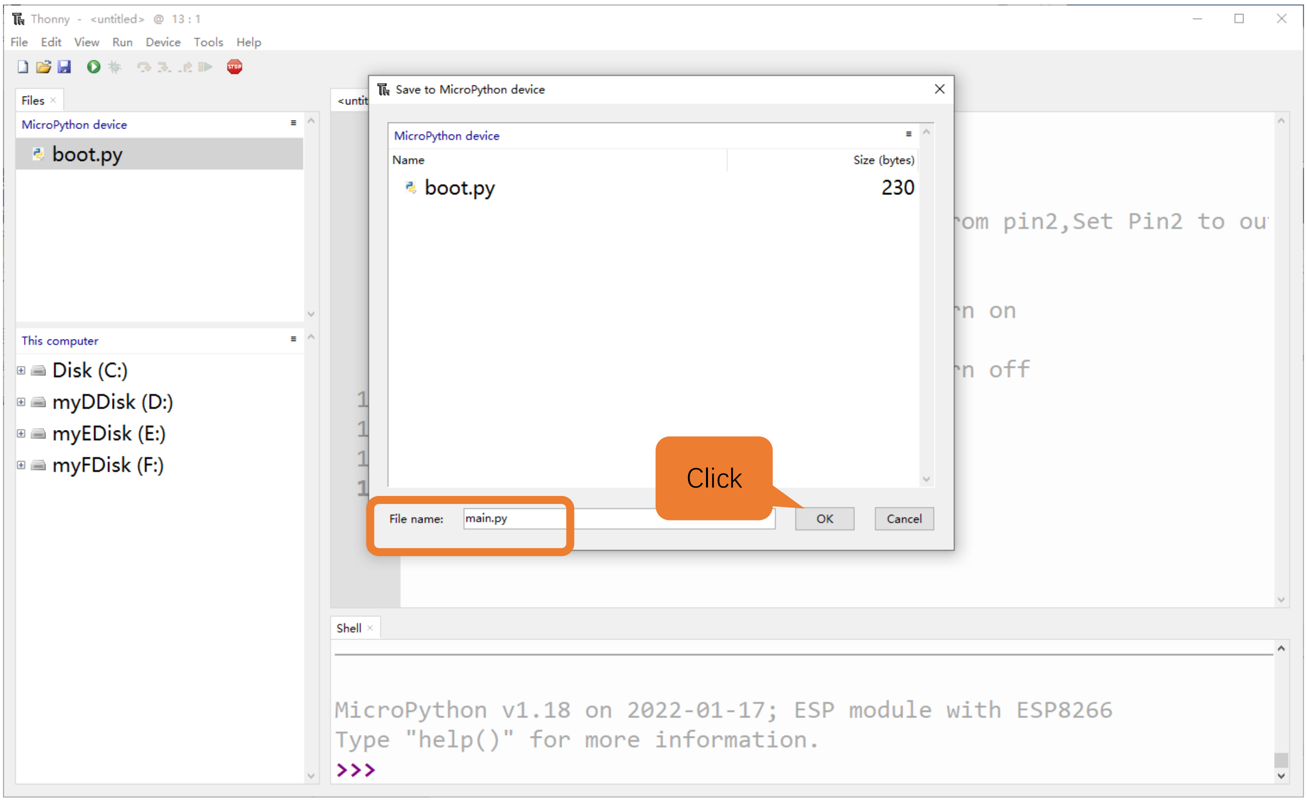
Enter codes in the newly opened file. Here we use codes of "01.1_Blink.py" as an example.



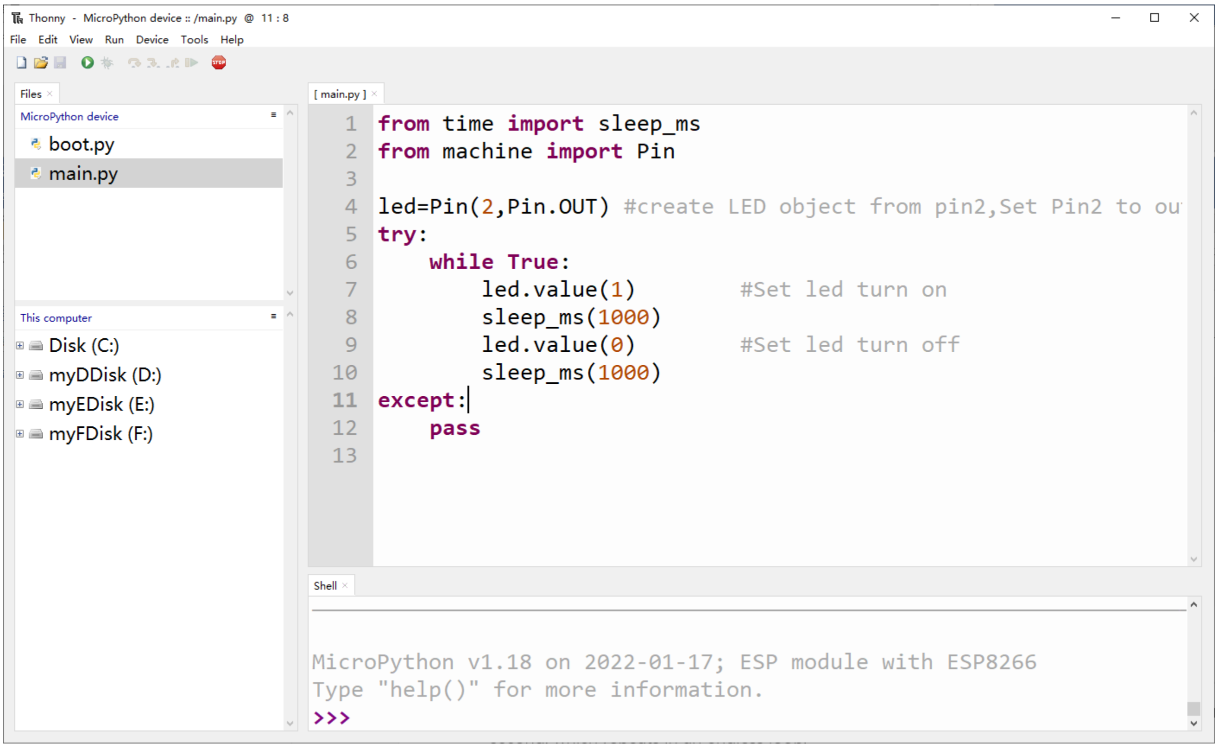
Click "Save" on the menu bar. You can save the codes either to your computer or to ESP8266.

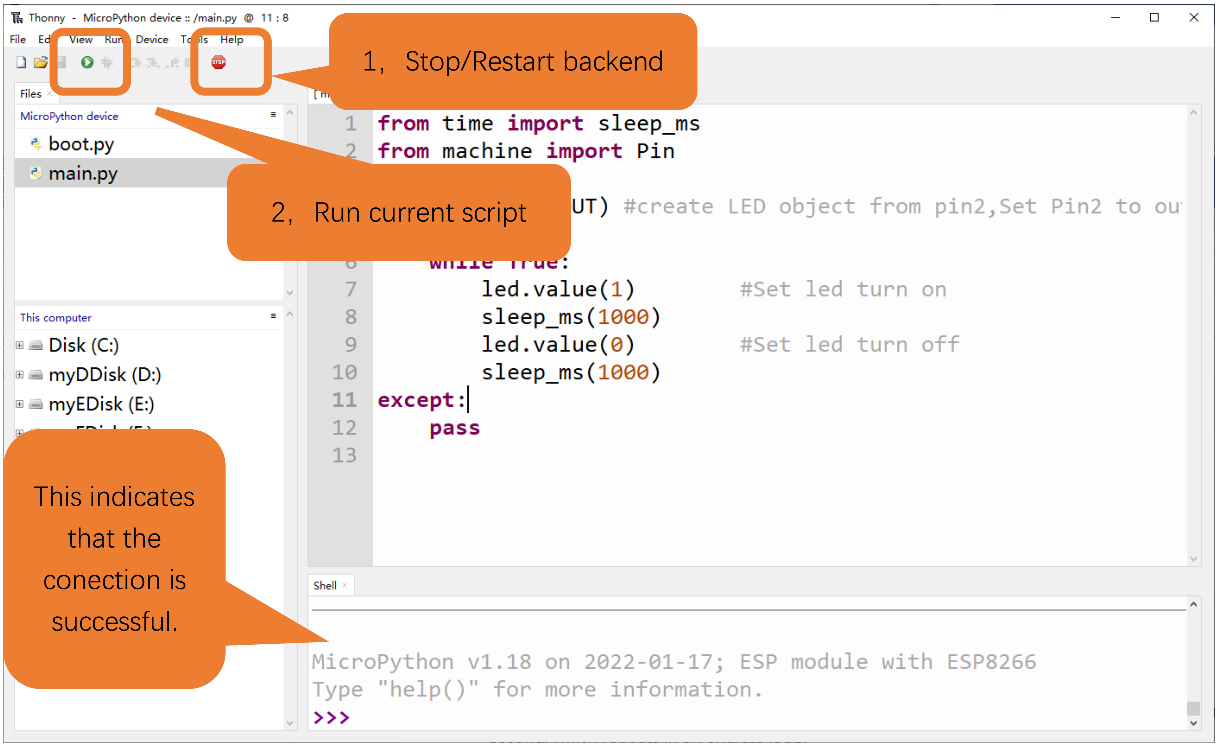


Select "MicroPython device", enter "main.py" in the newly pop-up window and click "OK".

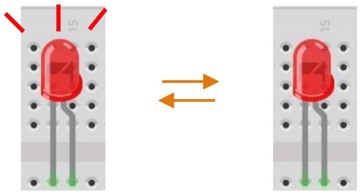


You can see that codes have been uploaded to ESP8266.





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.



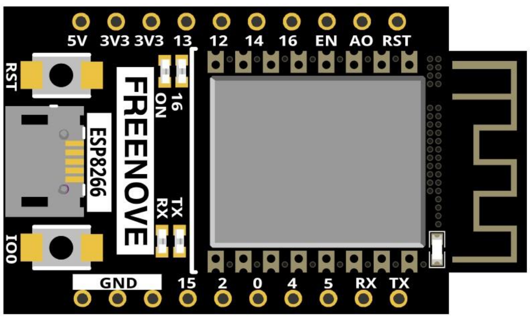
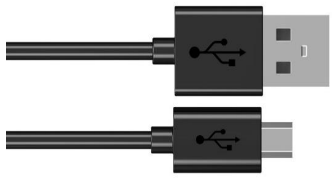
Chapter 1 LED (Important)

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

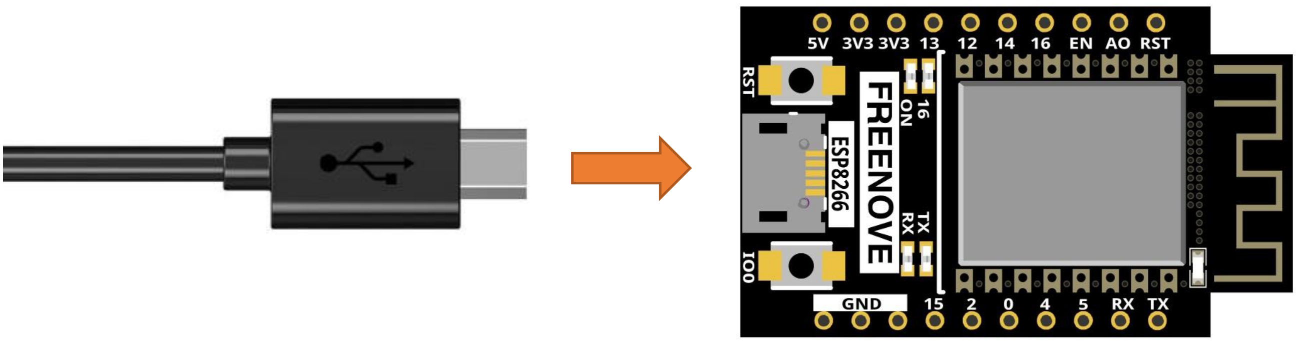
In this project, we will use ESP8266 to control blinking a common LED.
If you have not yet installed Thonny, click here.
If you have not yet downloaded Micropython Firmware, click here.
If you have not yet loaded Micropython Firmware, click here.

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.



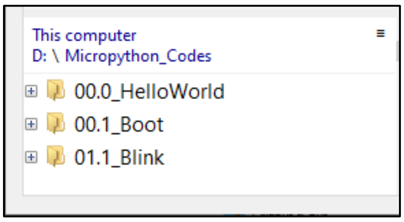
In the following projects, we only use USB cable to power ESP8266 development board by default.

Code

Codes used in this tutorial are saved in "**Freenove_Ultimate_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**".

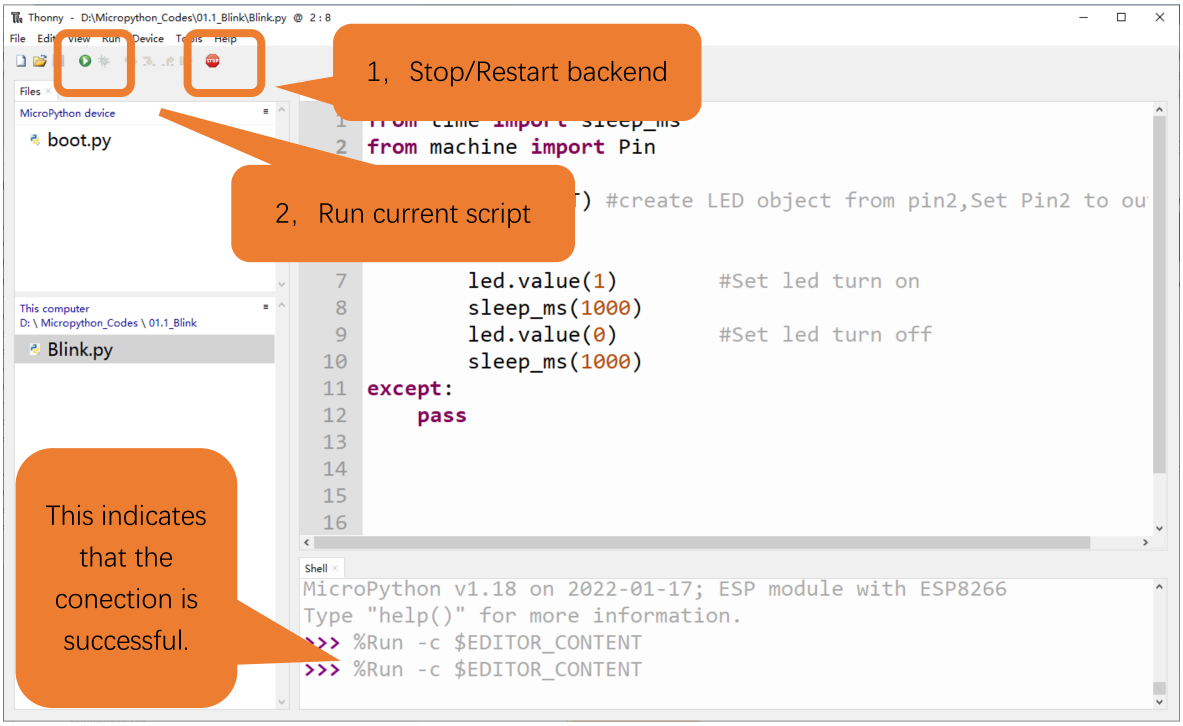
01.1_Blink

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

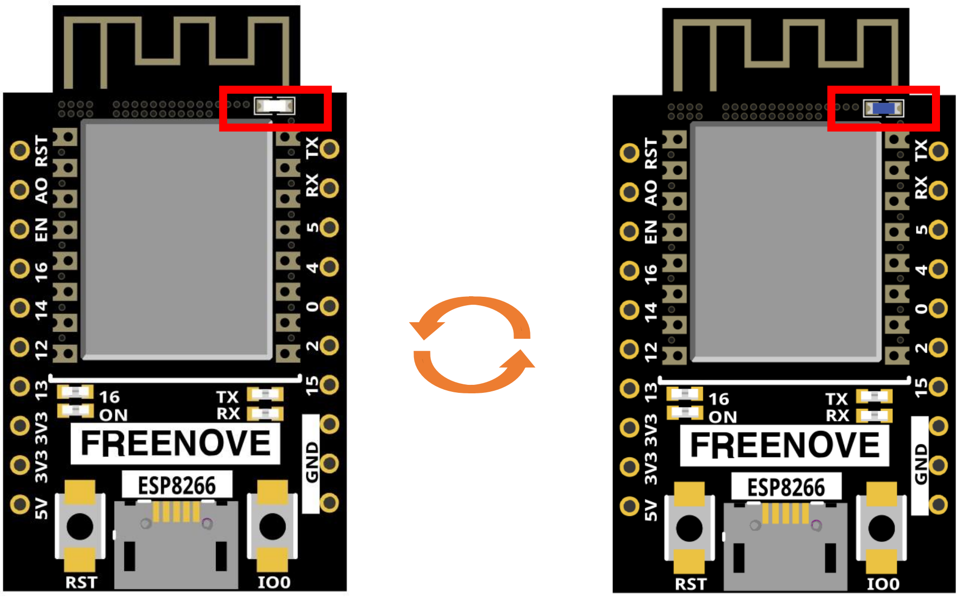


Expand folder "01.1_Blink" and double click "Blink.py" to open it. As shown in the illustration below.

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.

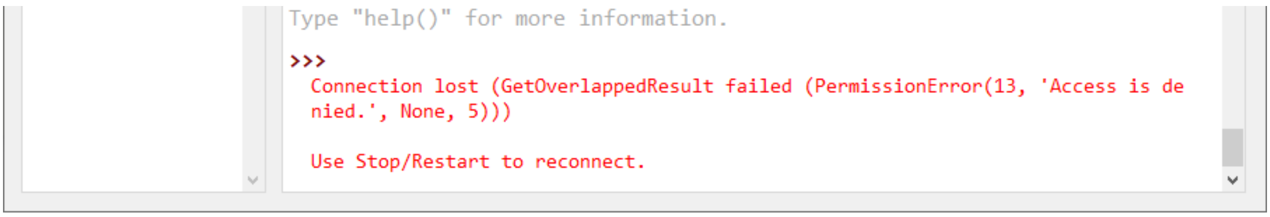


Click "Run current script" shown in the box above, the code starts to be executed and the LED in the circuit starts to blink.



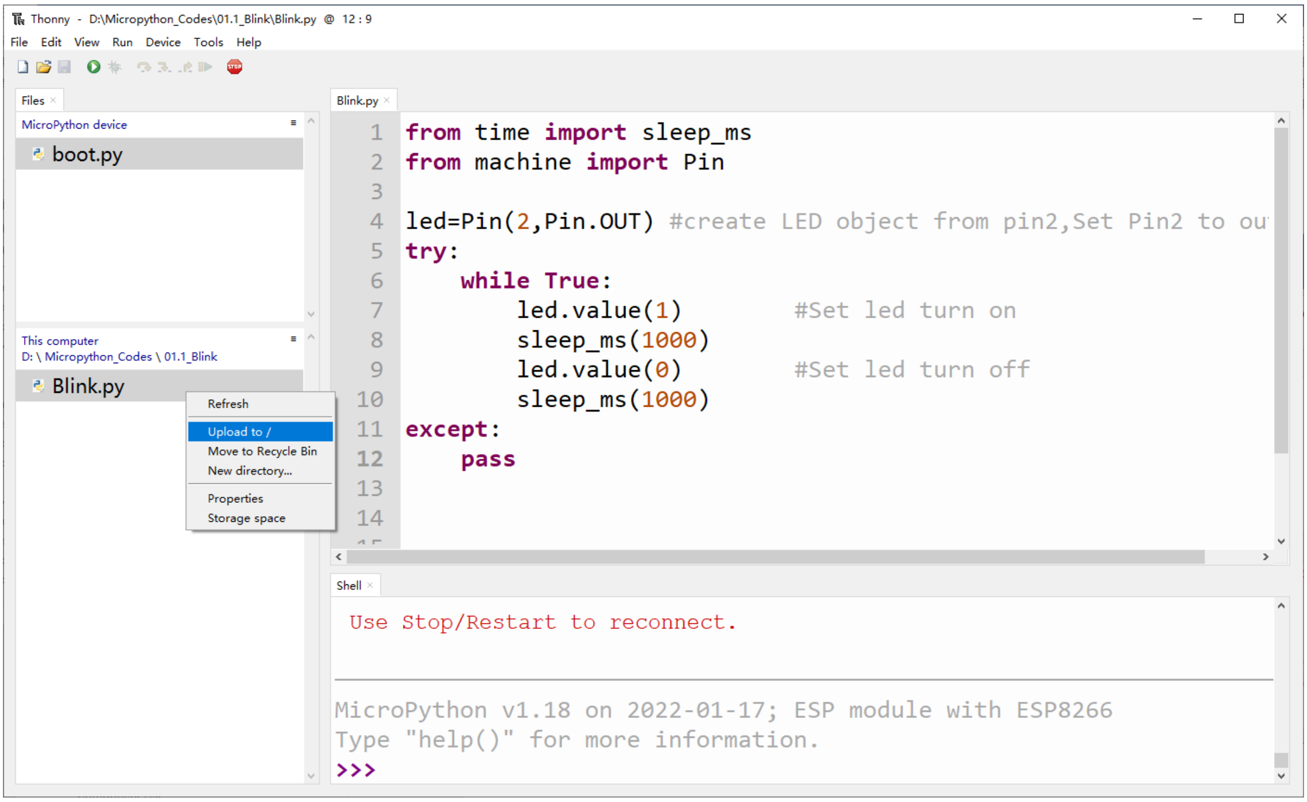
Note:

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.

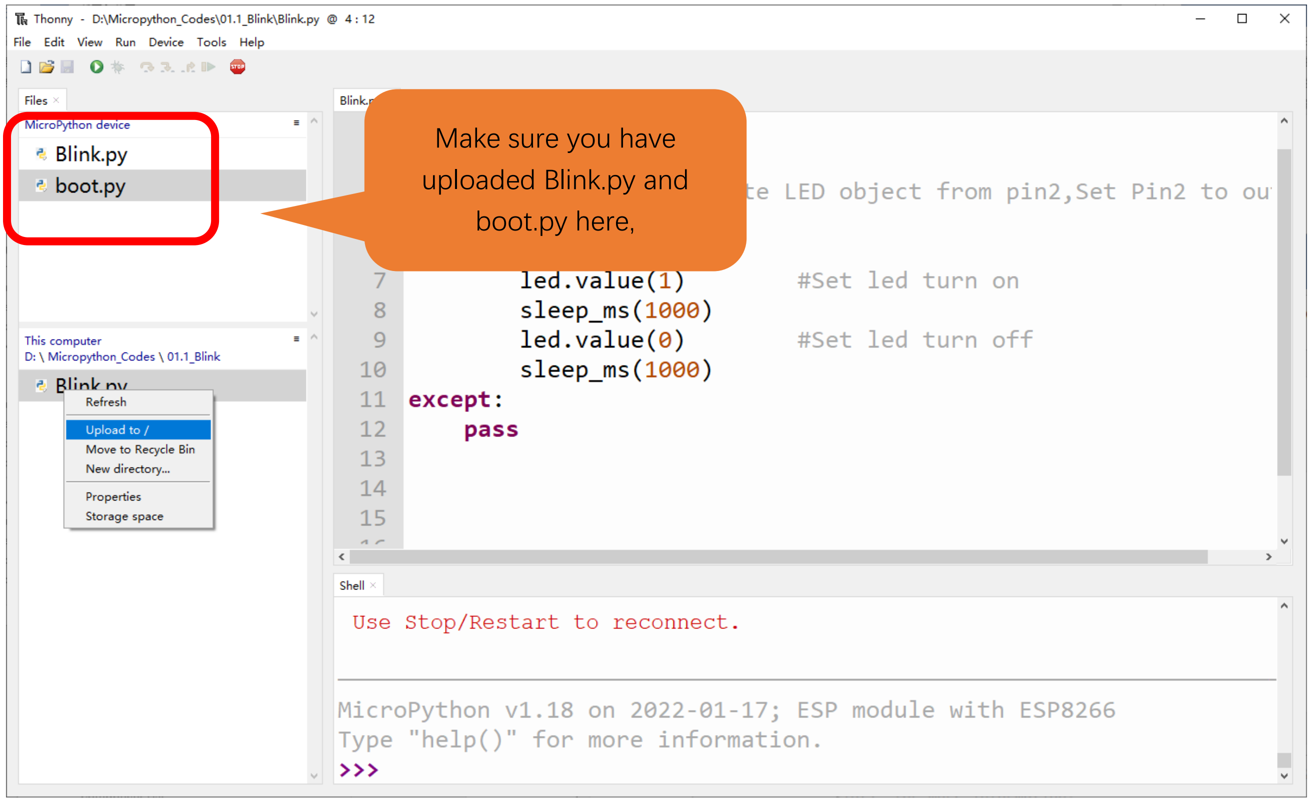


Uploading code to ESP8266

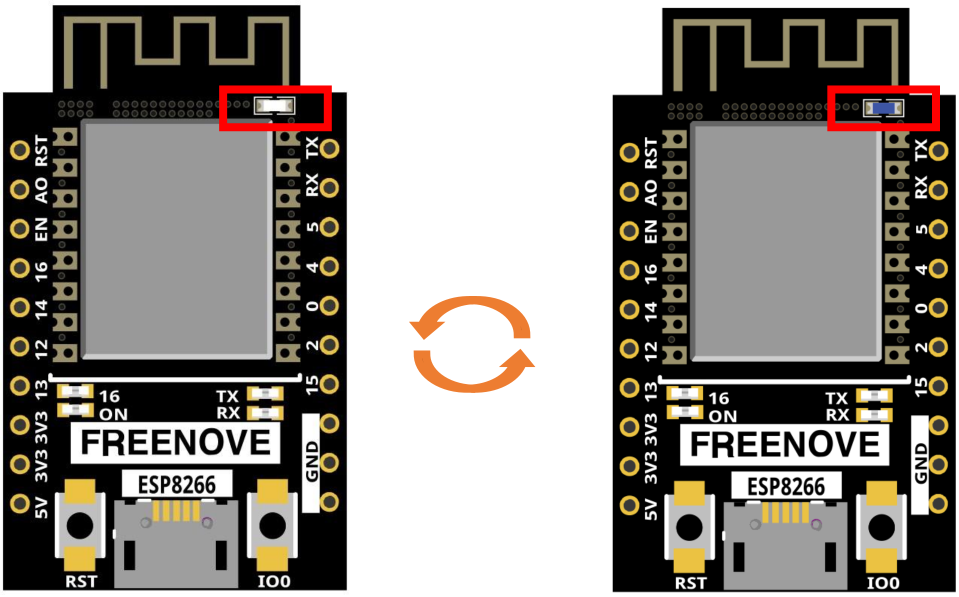
As shown in the following illustration, right-click the file Blink.py and select “Upload to /” to upload code to ESP8266.



Upload boot.py in the same way.

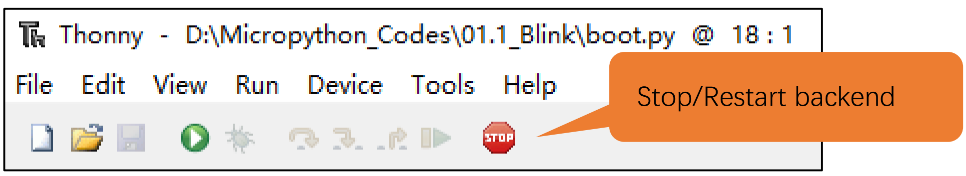


Press the reset key of ESP8266 and you can see LED is ON for one second and then OFF for one second, which repeats in an endless loop.



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.

Setup

Loop

```
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.

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

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

```
#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.

```
    while True:
        led.value(1) #Set led turn on
        sleep_ms(1000)
        led.value(0) #Set led turn off
        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().

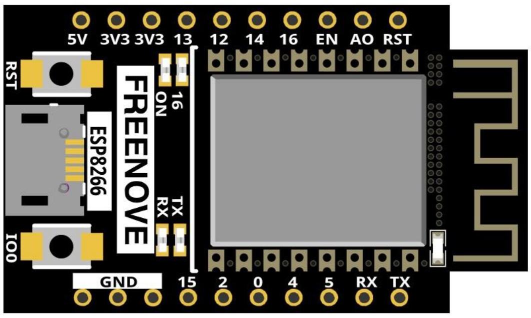
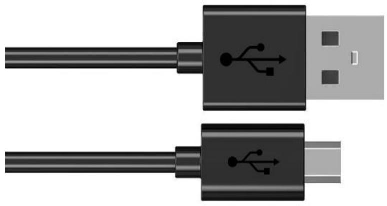
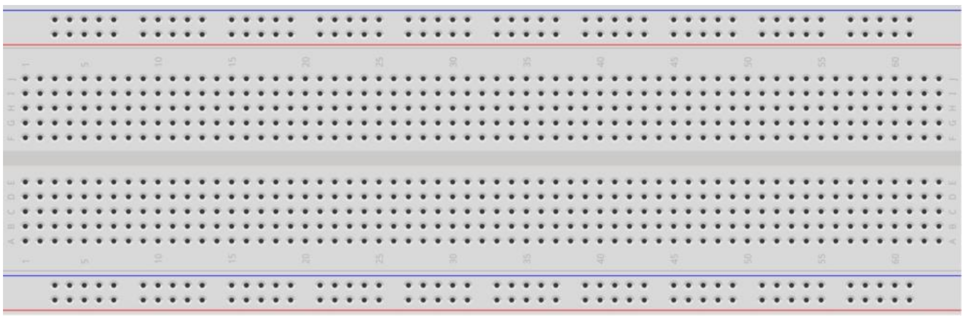
old_t: Starting time

new_t: Ending time

Project 1.2 Blink

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

Component List

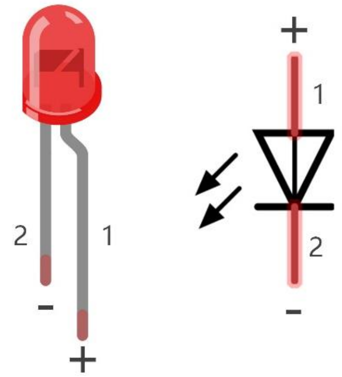
<table>
<tr><td colspan="3">ESP8266 x1

</td><td colspan="2">USB cable
</td></tr>
<tr><td colspan="5">Breadboard x1
</td></tr>
<tr><td>LED x1
</td><td>Resistor 220Ω x1
</td><td colspan="3">Jumper wire M/M x3
</td></tr>
</table>

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.



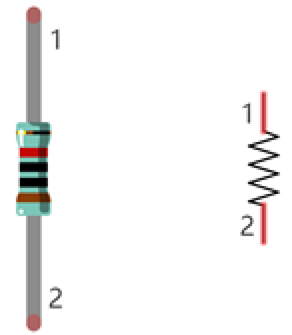
LED	Voltage	Maximum current	Recommended current
Red	1.9-2.2V	20mA	10mA
Green	2.9-3.4V	10mA	5mA
Blue	2.9-3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

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.

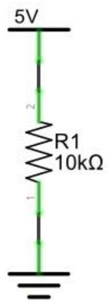
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



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.

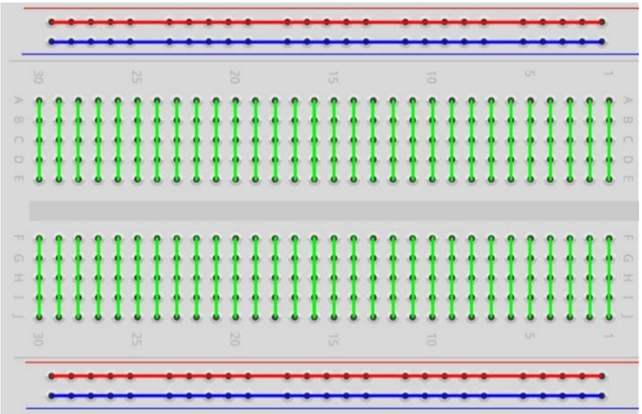


WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

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)

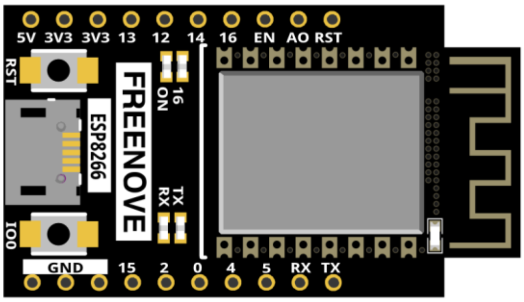
Breadboard

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.



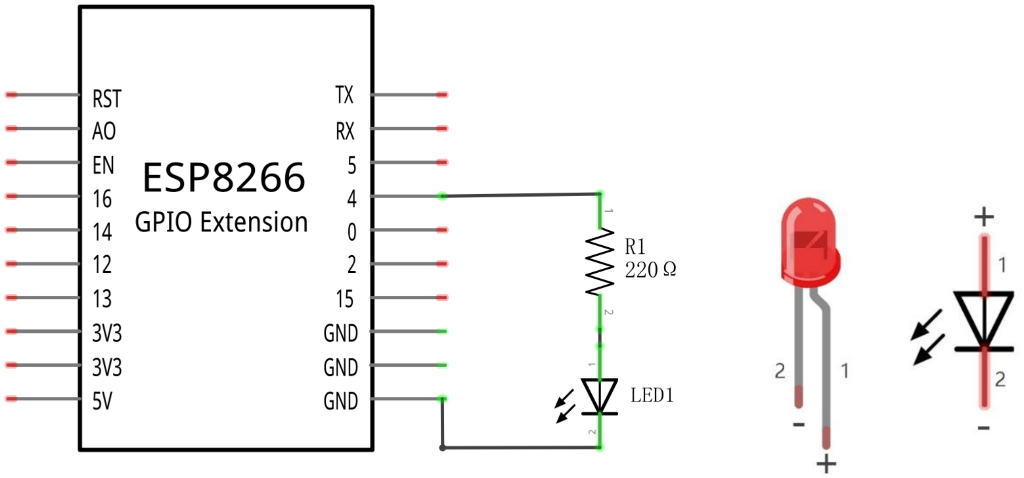
Later, we only use USB cable to power ESP8266 in default.

Circuit

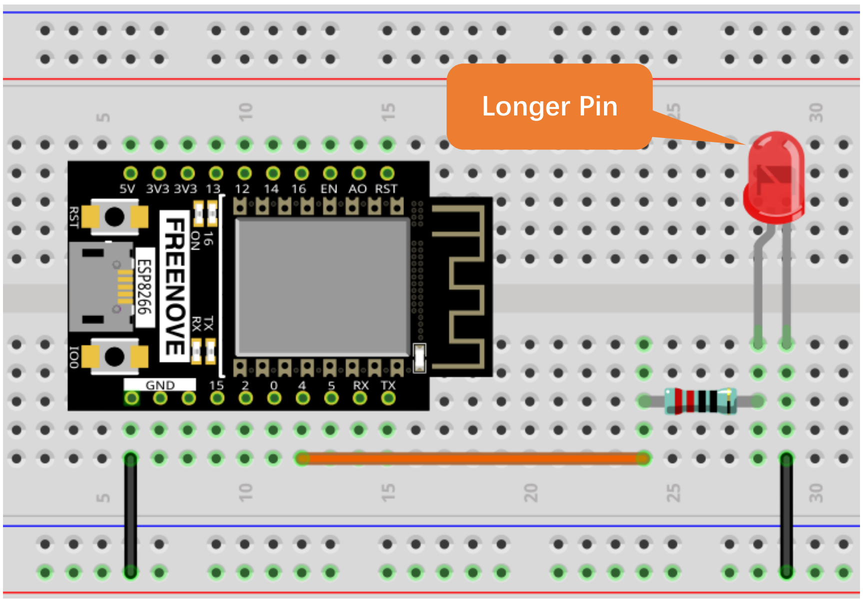
First, disconnect all power from the ES8266. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the PC to ESP8266.

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



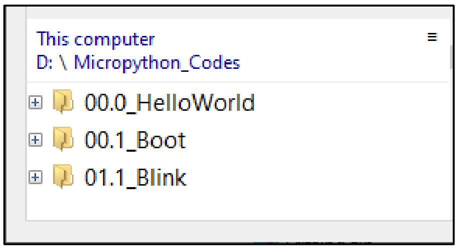
Code

Codes used in this tutorial are saved in "**Freenove_Ultimate_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**".

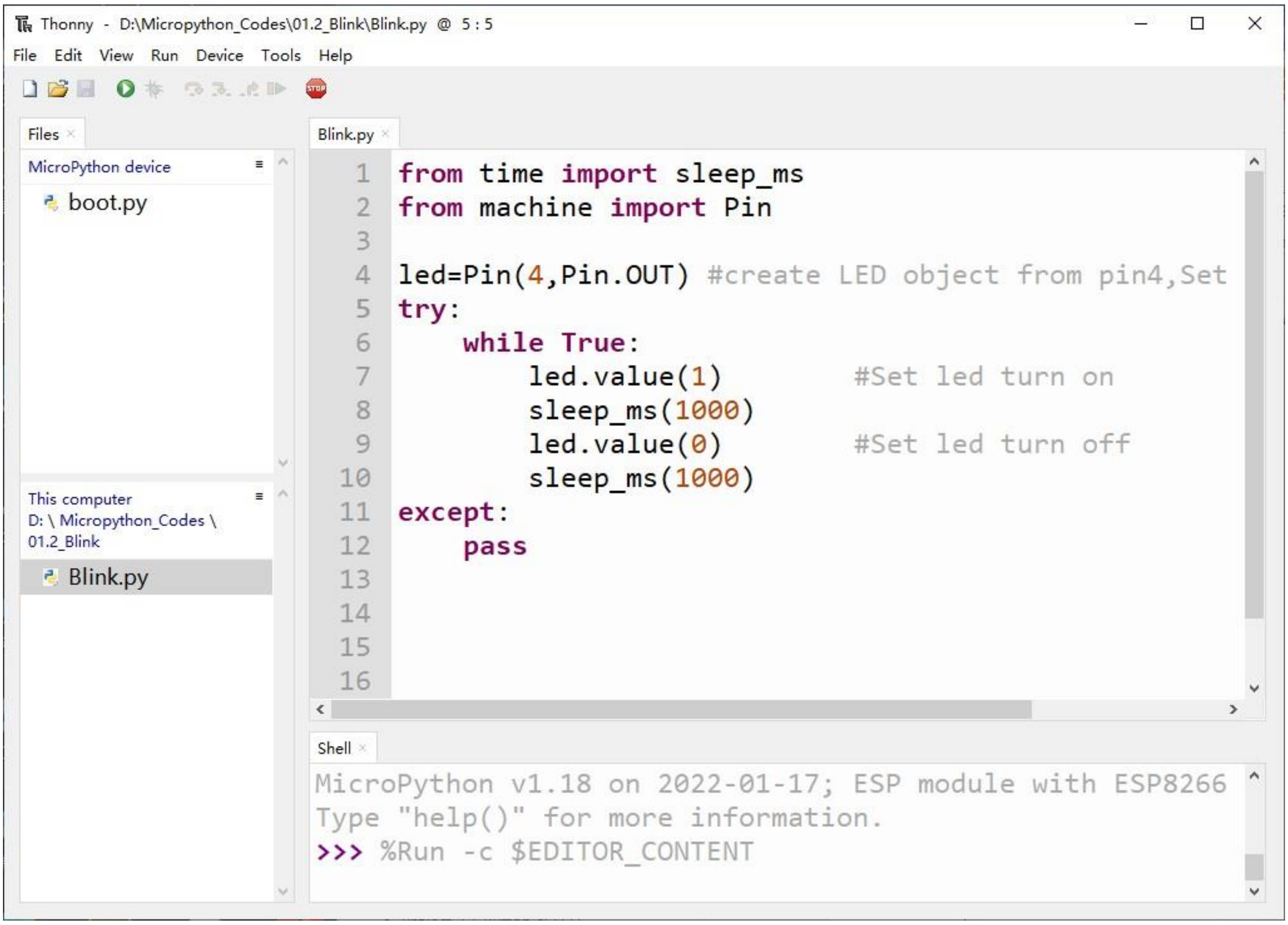
Any concerns? ✉ **support@freenove.com**

01.2_Blink

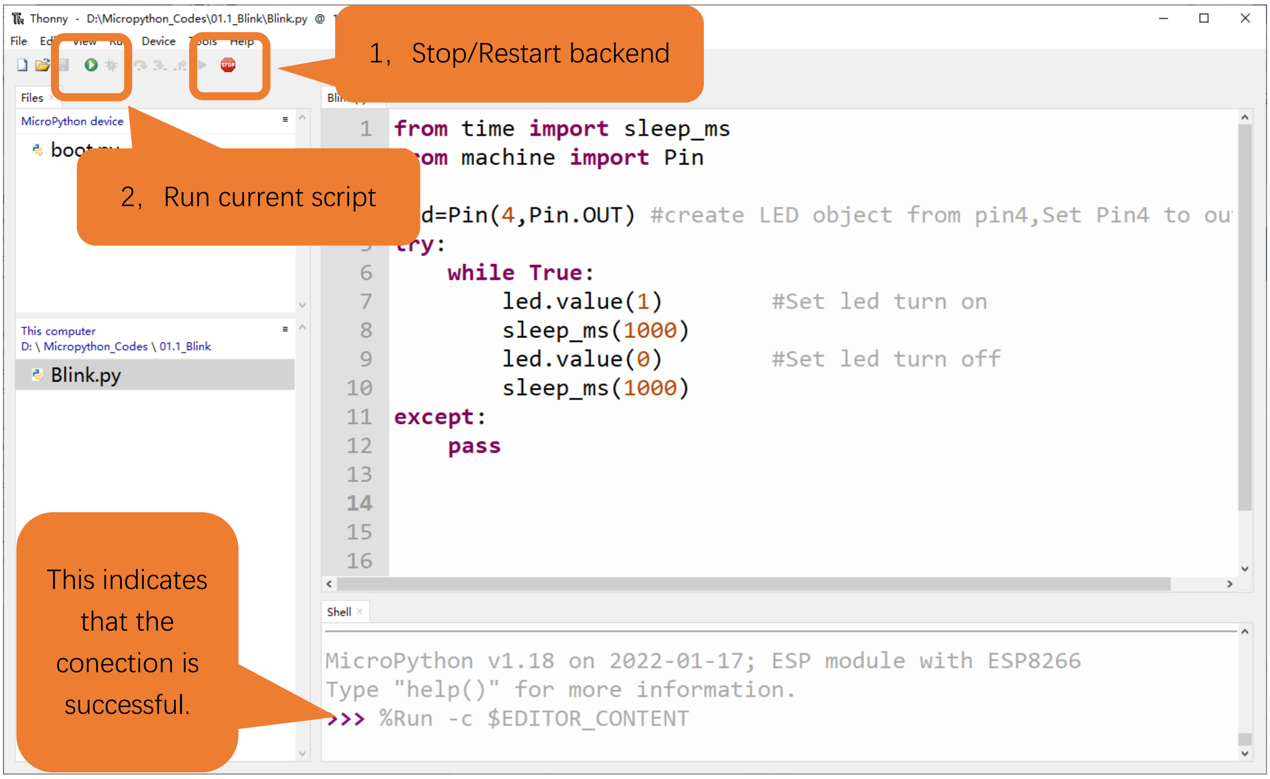
Open "Thonny", click "This computer"→"D:"→"Micropython_Codes".



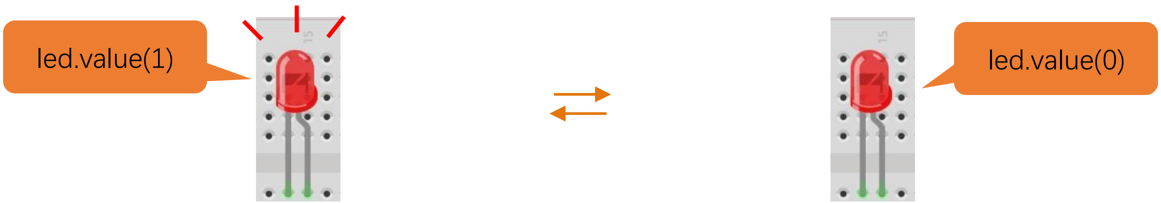
Expand folder "01.2_Blink" and double click "Blink.py" to open it. As shown in the illustration below.



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.

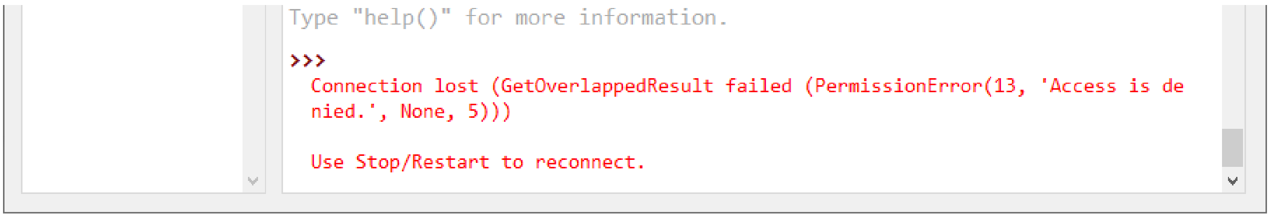


Click “Run current script” shown in the box above, the code starts to be executed and the LED in the circuit starts to blink.



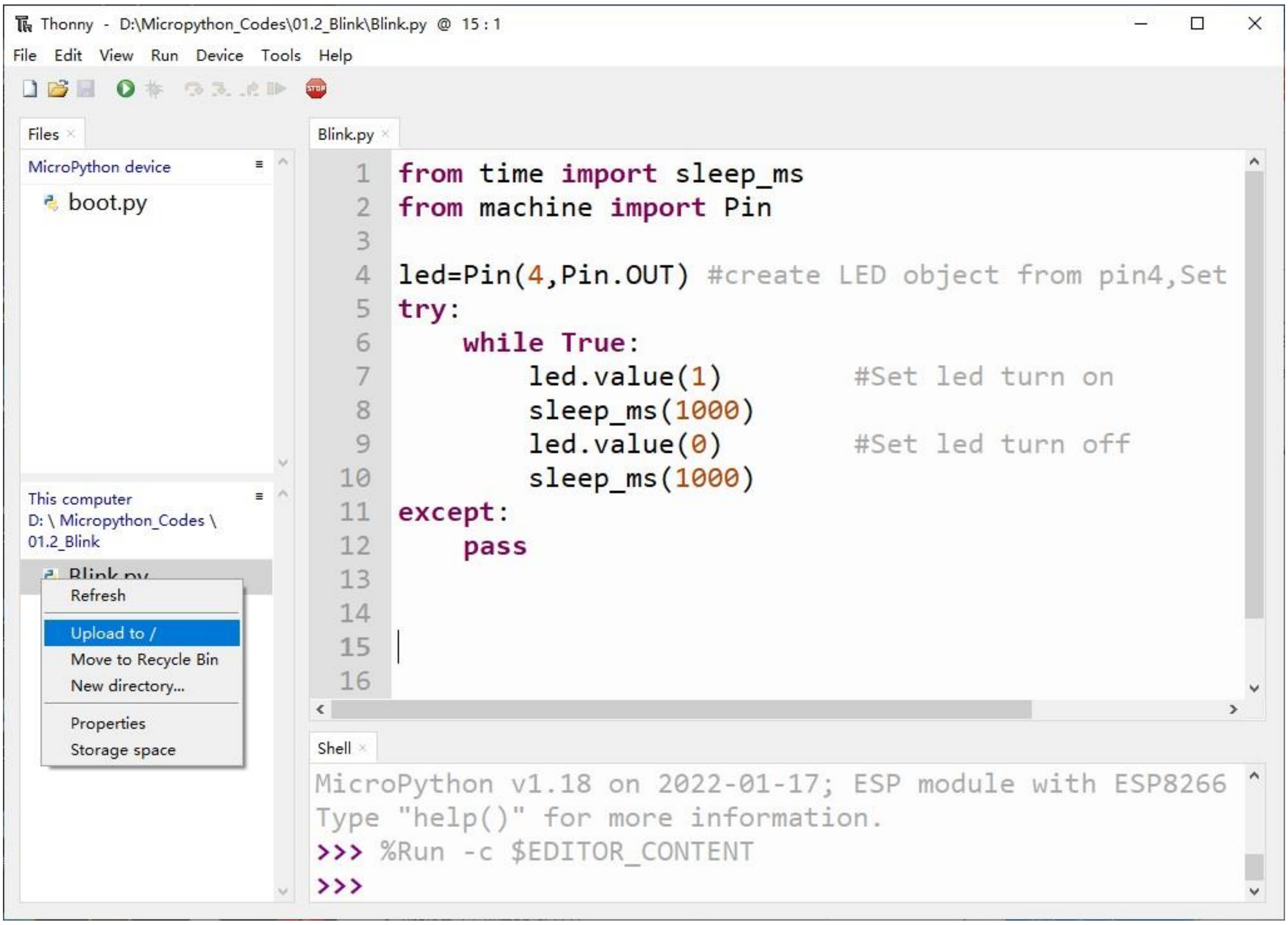
Note:

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.

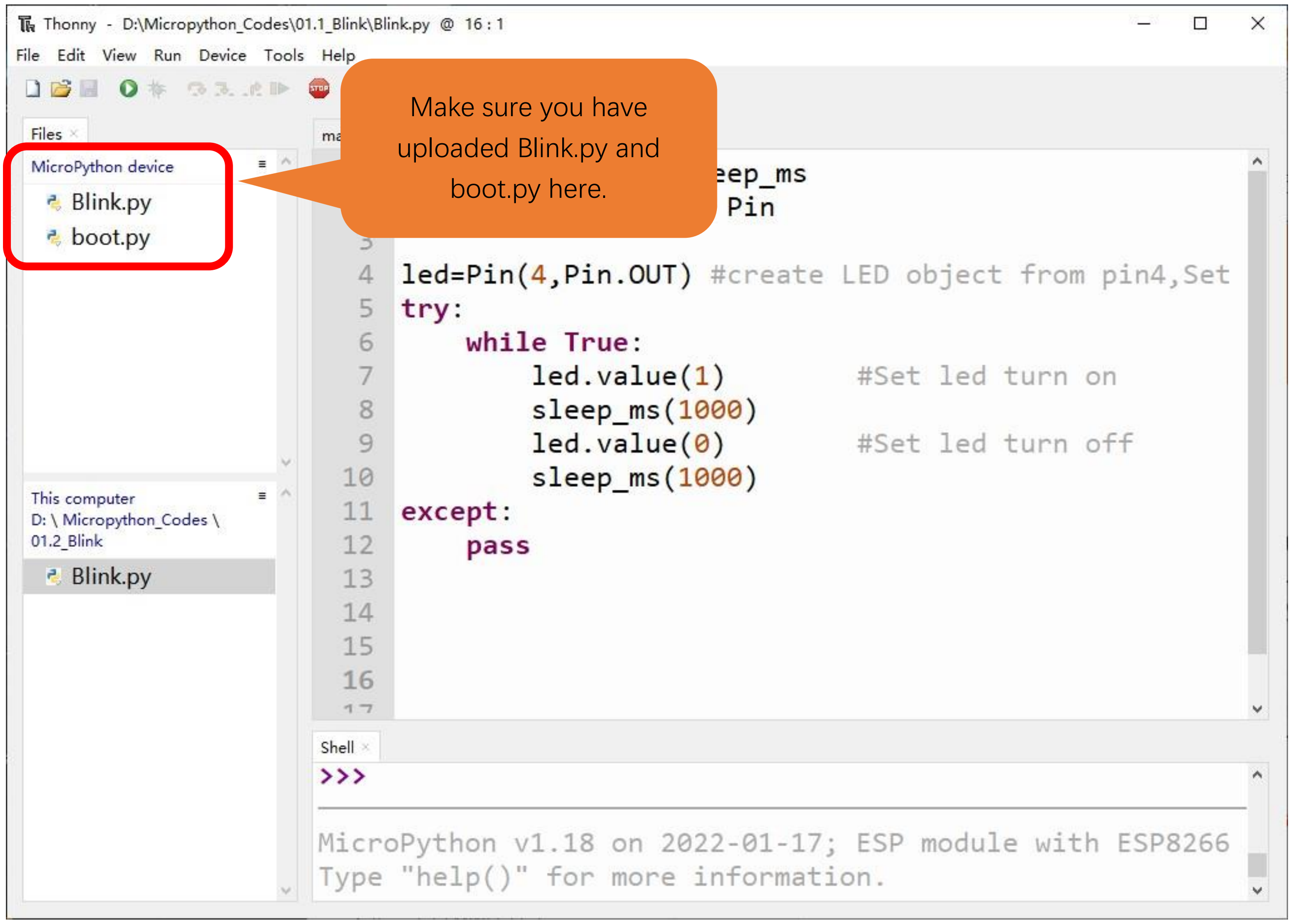


Uploading code to ESP8266

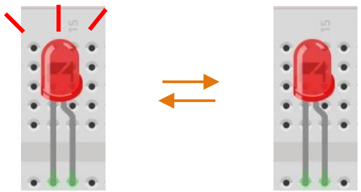
As shown in the following illustration, right-click the file Blink.py and select “Upload to /” to upload code to ESP8266.


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
Blink.py
Refresh
Upload to /
Move to Recycle Bin
New directory...
Properties
Storage space
Blink.py
from time import sleep_ms
from machine import Pin
led=Pin(4,Pin.OUT) #create LED object from pin4,Set
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
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.

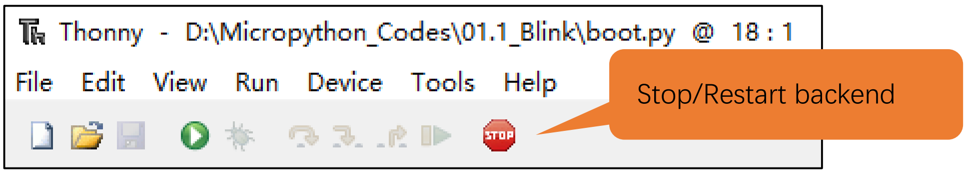


Press the reset key of ESP8266 and you can see LED is ON for one second and then OFF for one second, which repeats in an endless loop.



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

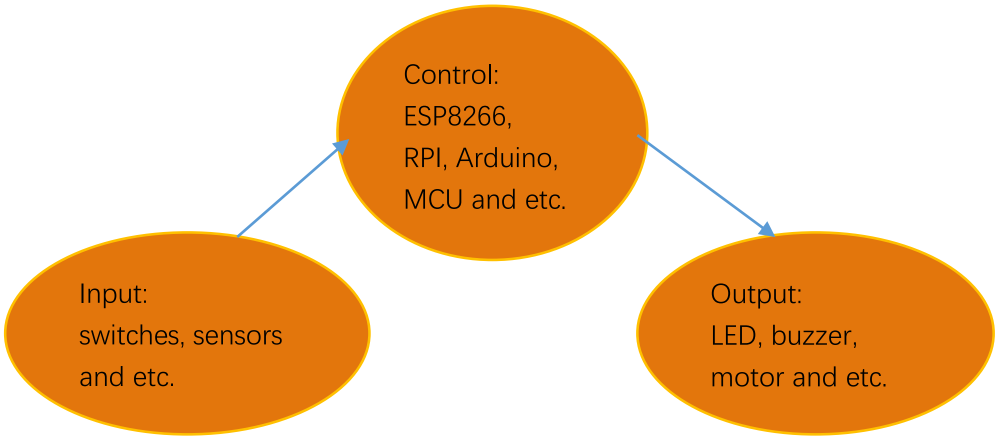
The following is the program code:

```
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.

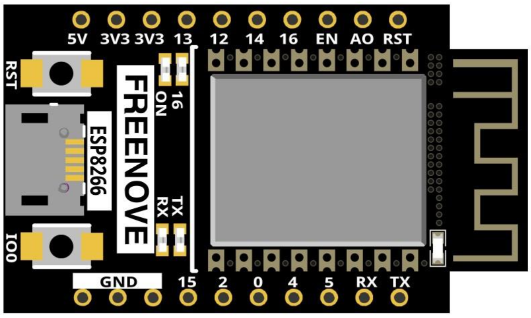
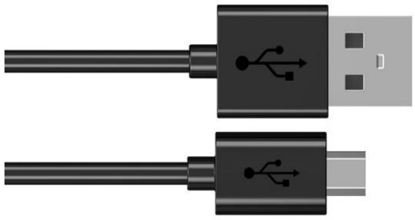
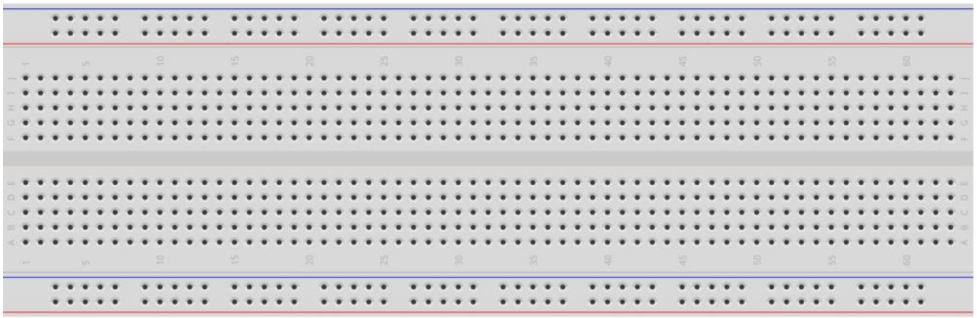


Next, we will build a simple control system to control an LED through a push button switch.

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.

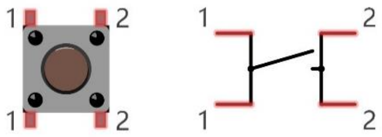
Component List

<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>Jumper wire M/M x6
</td><td>LED x1
</td><td>Resistor 220Ω x1
</td><td>Resistor 10kΩ x2
</td><td>Push button x1
</td></tr>
</table>

Component knowledge

Push button

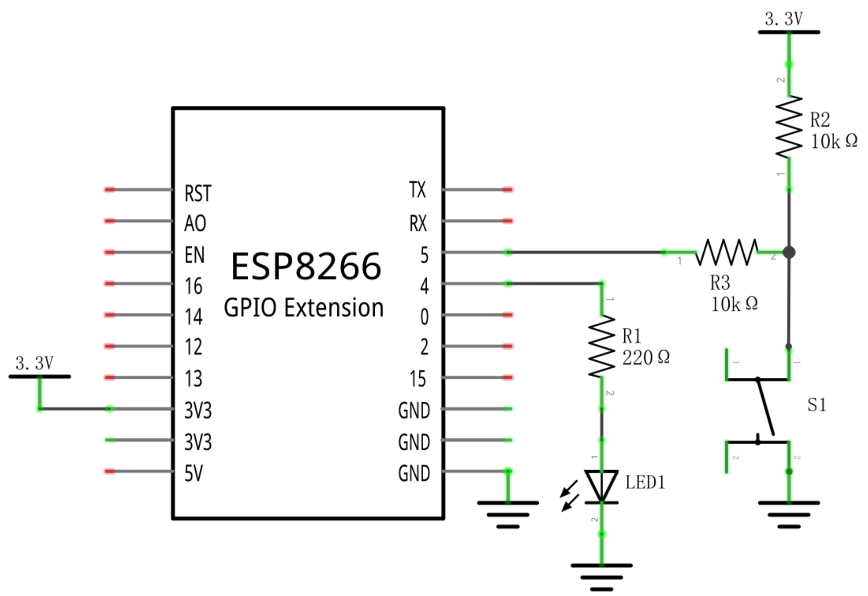
This type of Push Button Switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



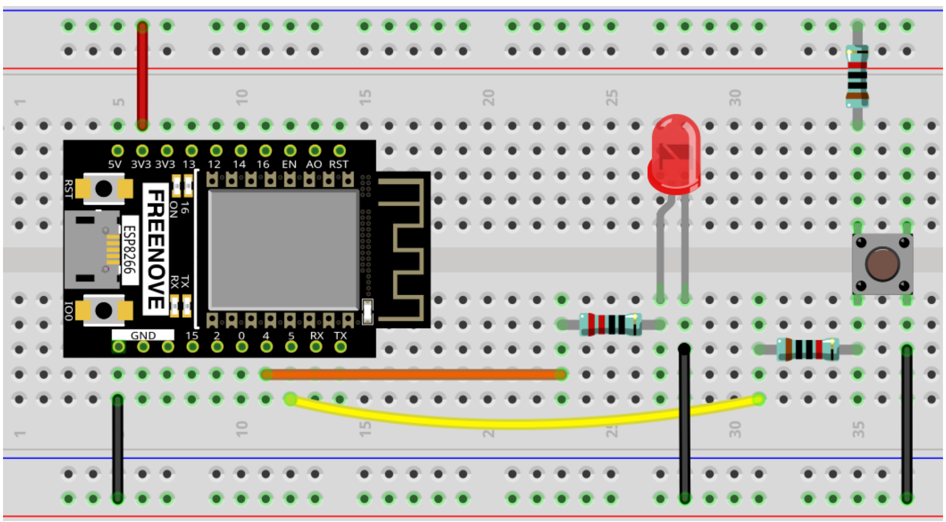
When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

Circuit

Schematic diagram



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



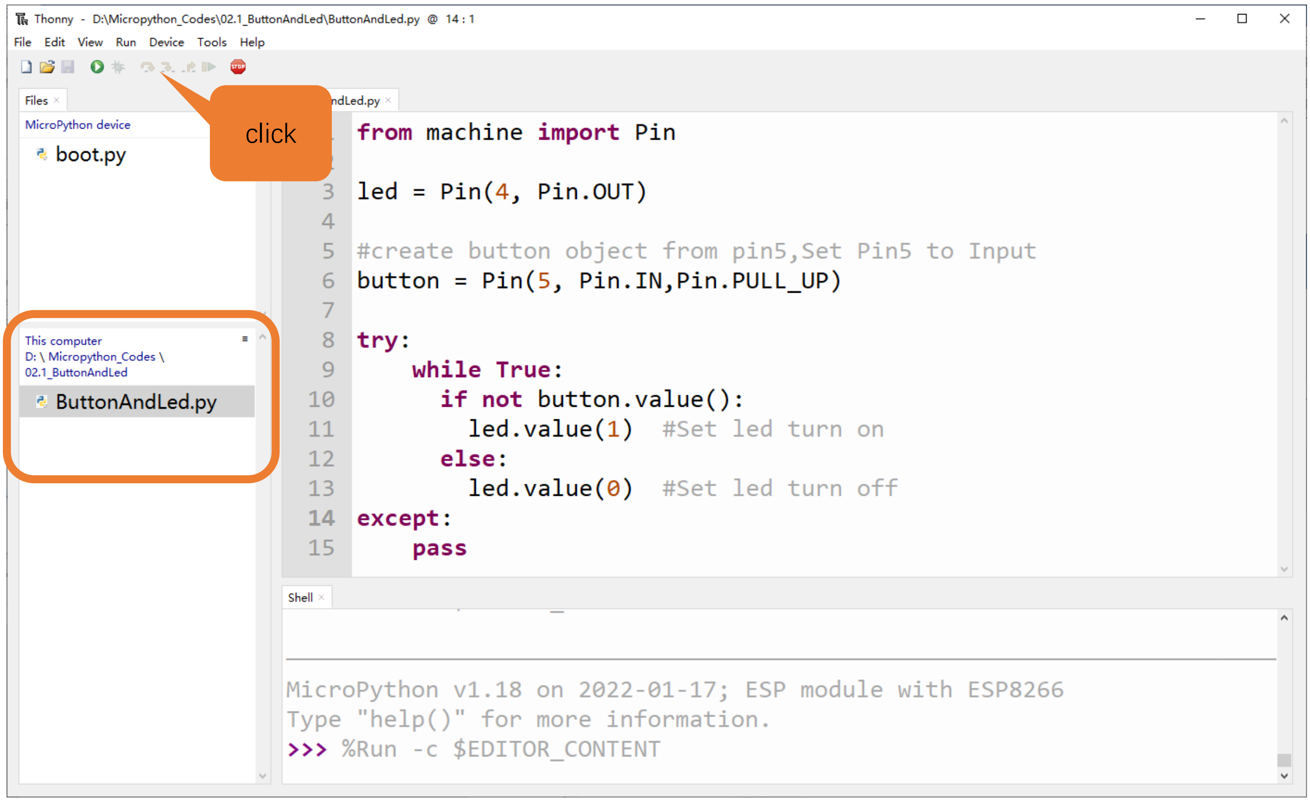
Code

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.

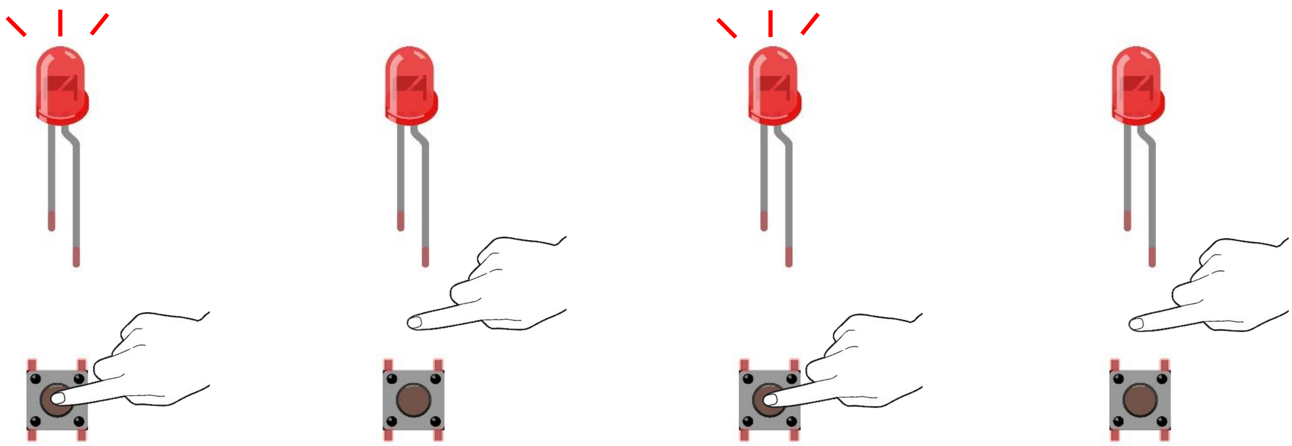
Move the program folder "**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

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

02.1_ButtonAndLed

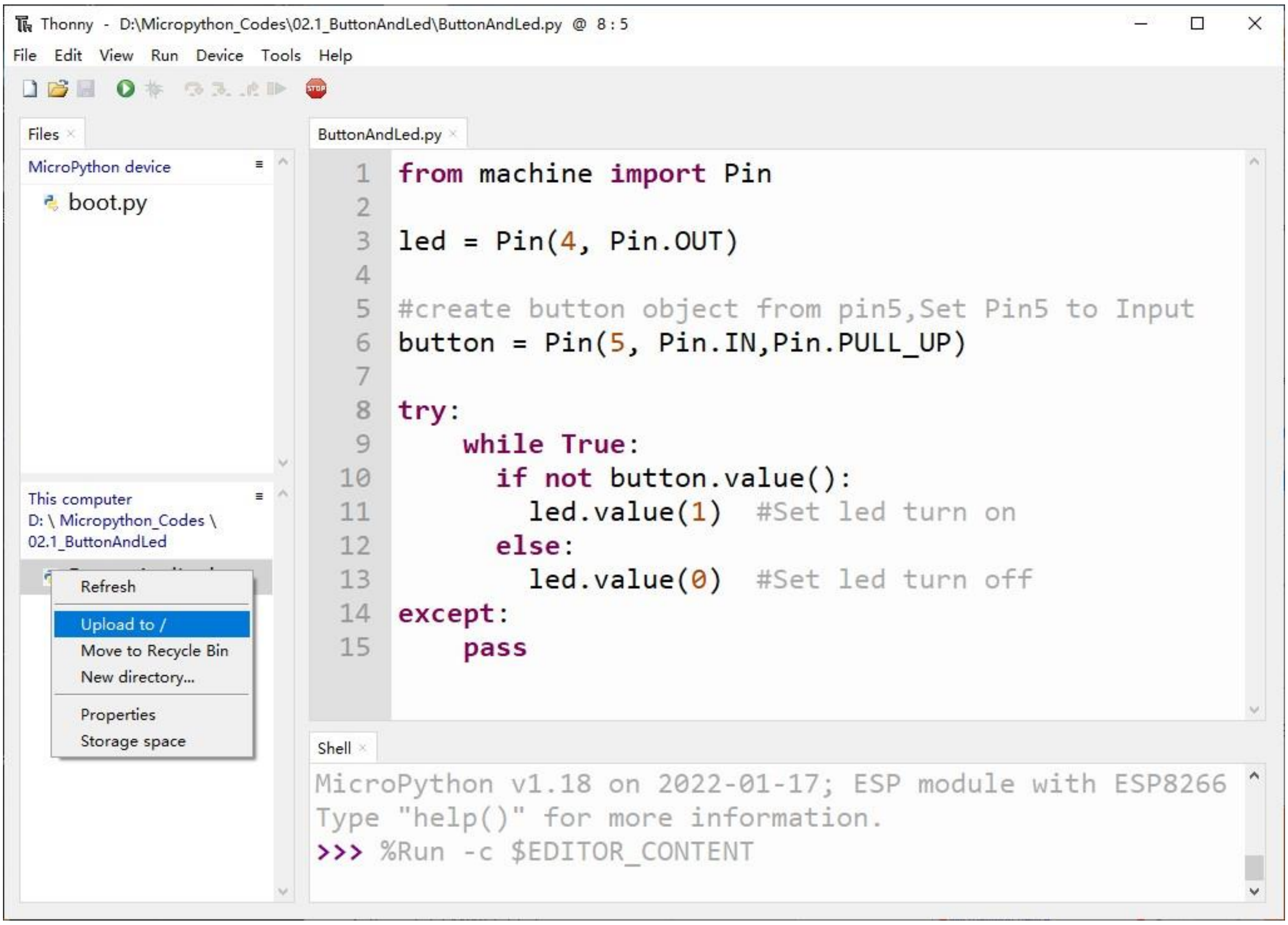


Click "Run current script" shown in the box of the above illustration, press the push button switch, LED turns ON; release the switch, LED turns OFF.

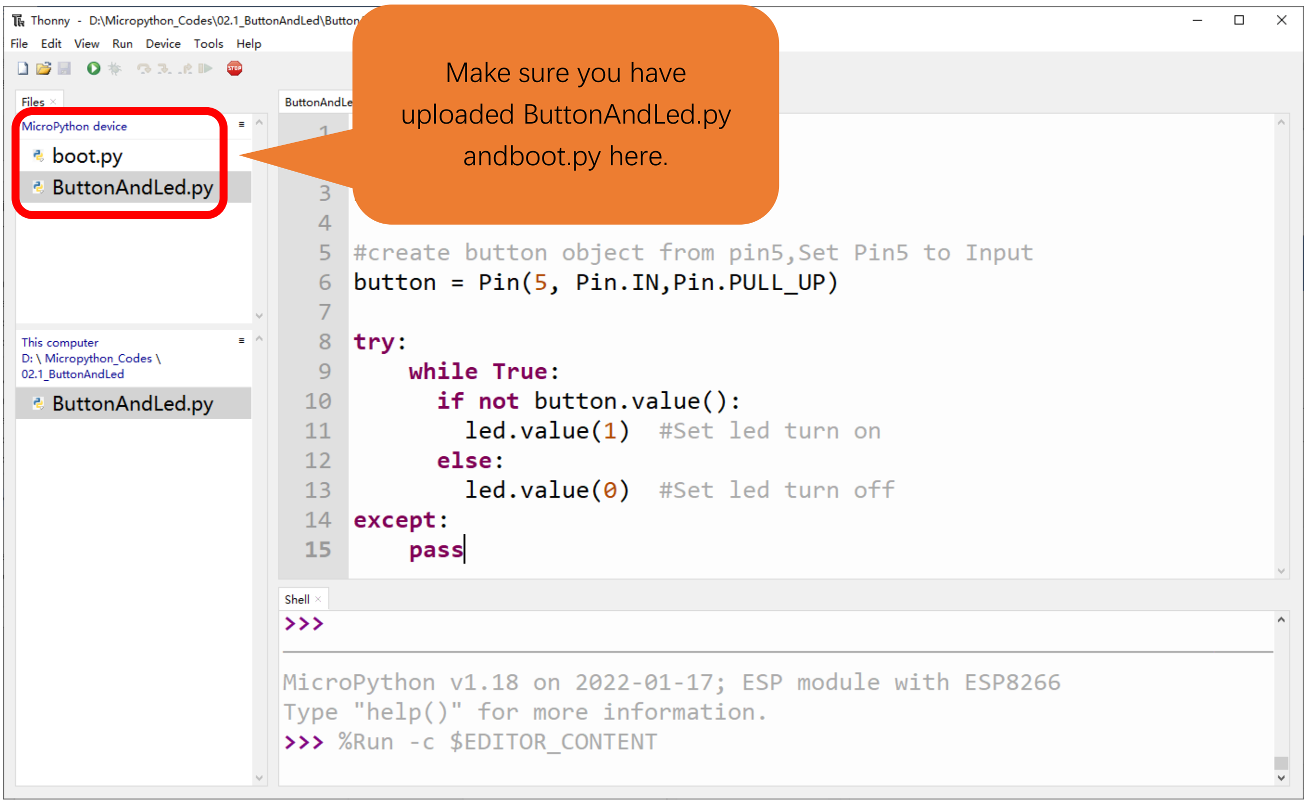


Upload Code to ESP8266

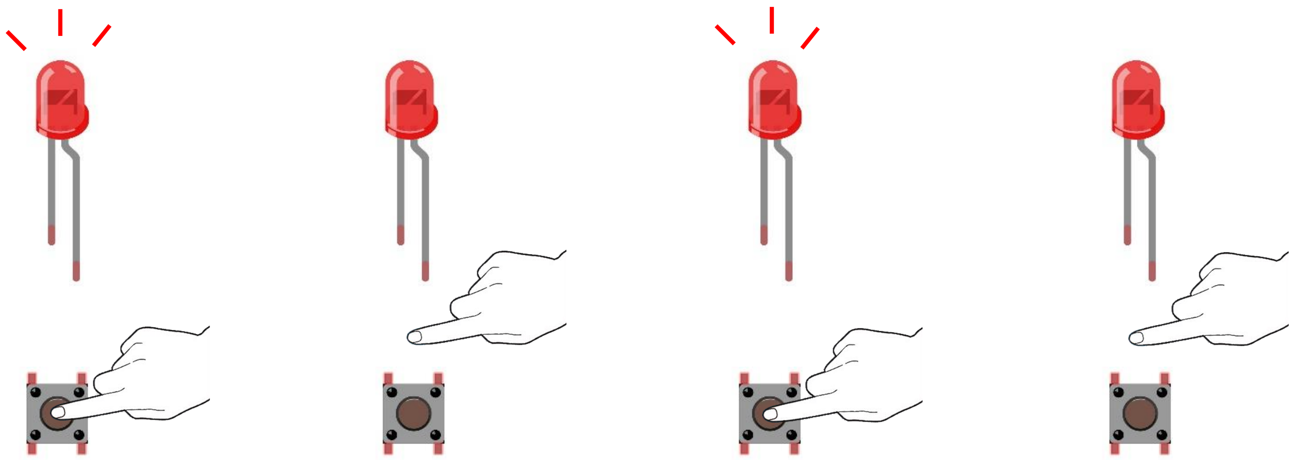
As shown in the following illustration, right-click file 02.1_ButtonAndLed and select “Upload to /” to upload code to ESP8266.



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.

```
    while True:
        if not button.value():
            led.value(1)   #Set led turn on
        else:
            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.

```
if not button.value():
    …
else:
    …
```

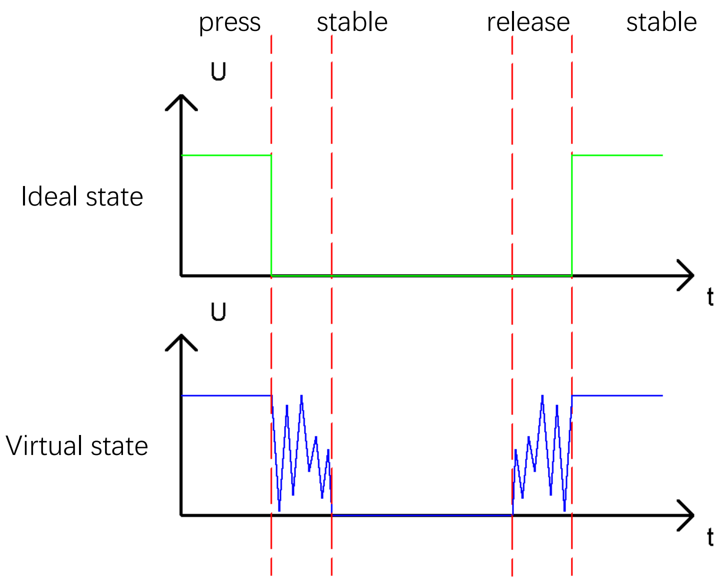
Project 2.2 MINI table lamp

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).

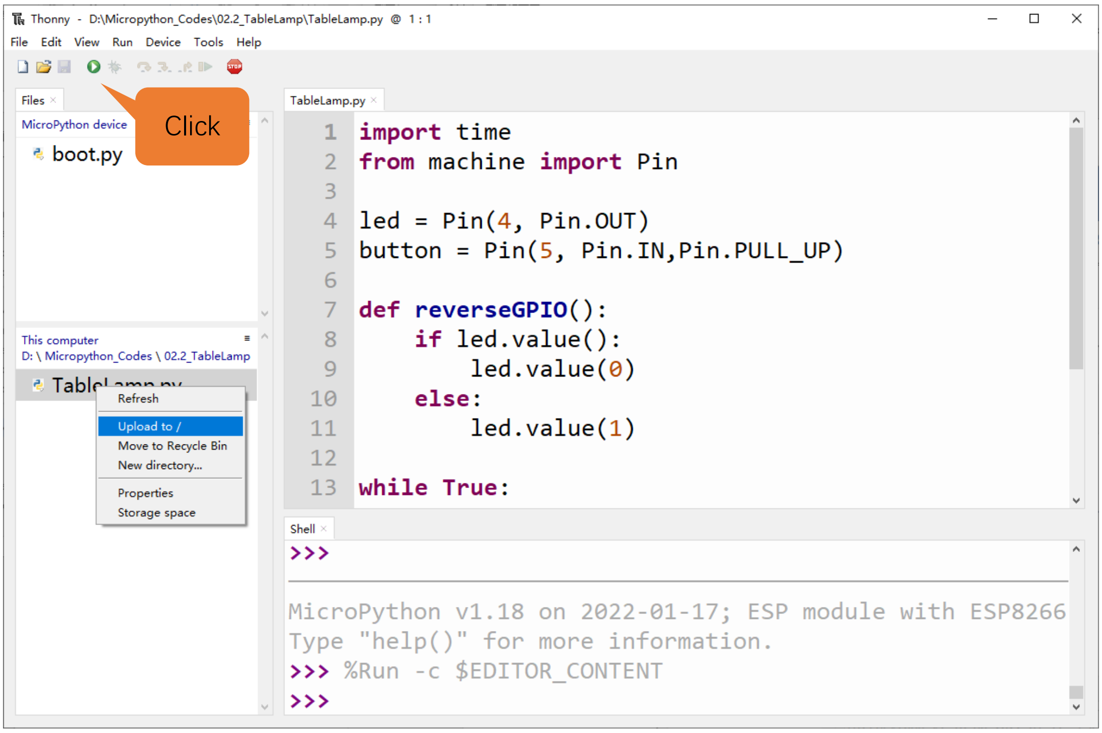
This project needs the same components and circuits as we used in the previous section.

Code

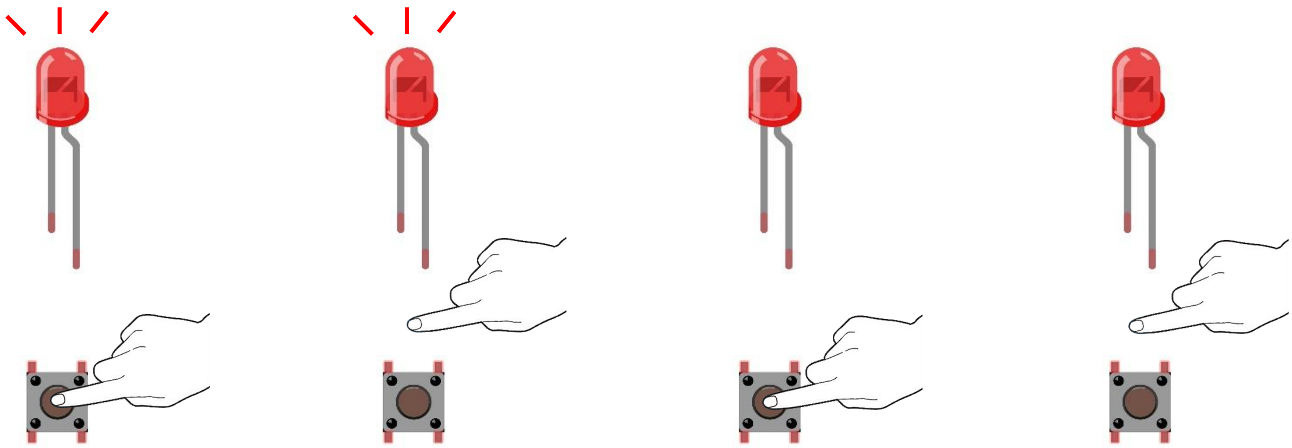
02.2_Tablelamp

Move the program folder "**Freenove_Ultimate_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".



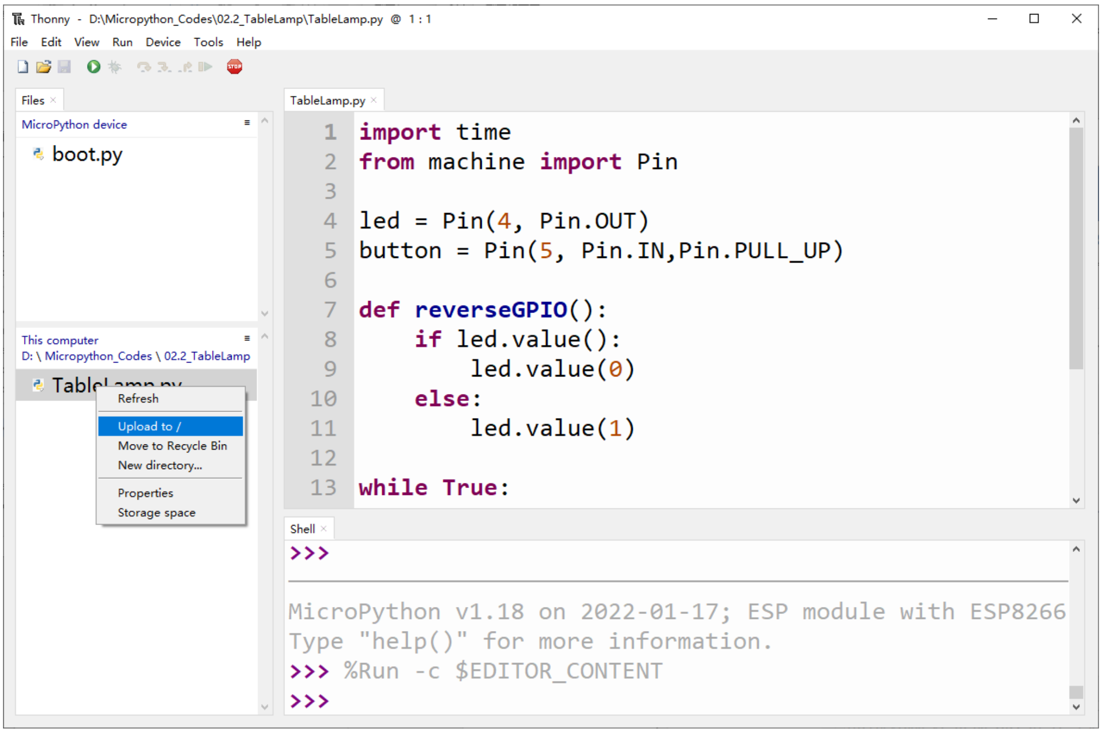
Click "Run current script" shown in the box of the above illustration, press the push button switch, LED turns ON; press it again, LED turns OFF.



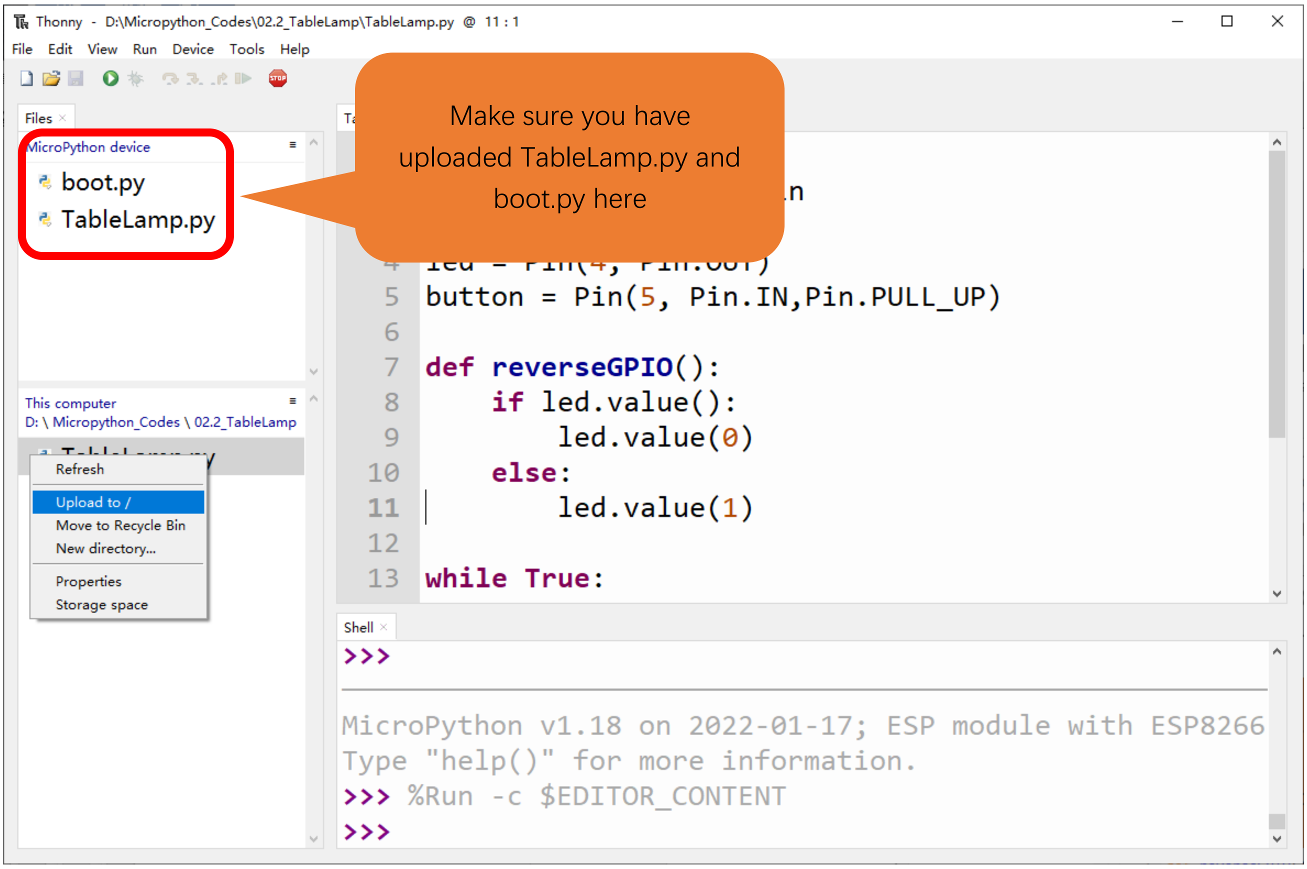
If you have any concerns, please contact us via: support@freenove.com

Upload code to ESP8266

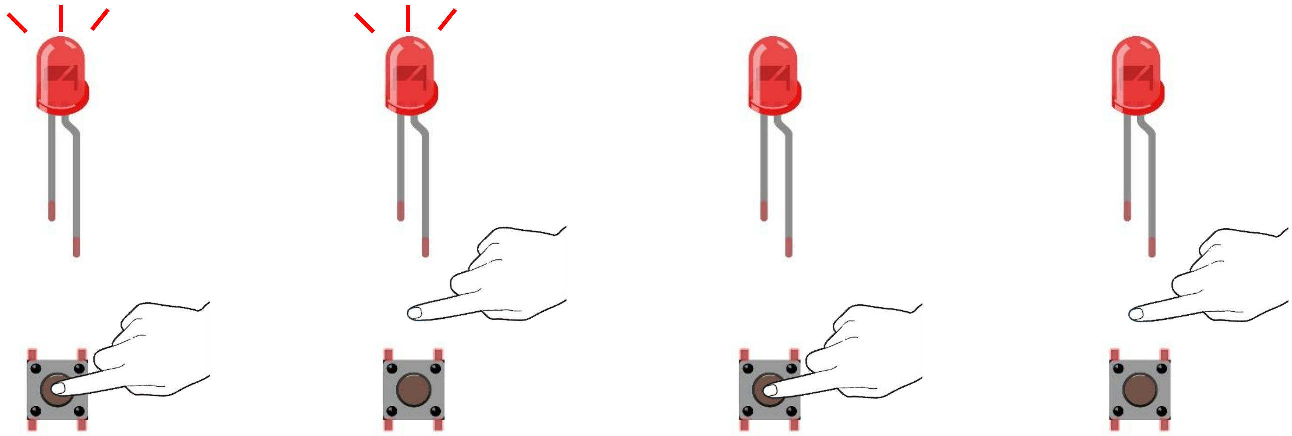
As shown in the following illustration, right-click file 02.2_TableLamp and select "Upload to /" to upload code to ESP8266.



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.

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

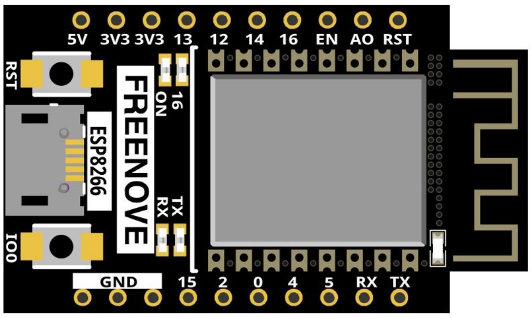
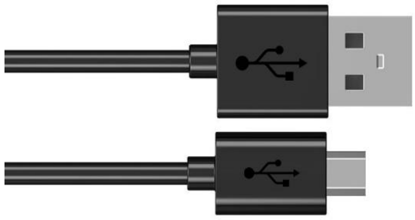
Chapter 3 LED Bar

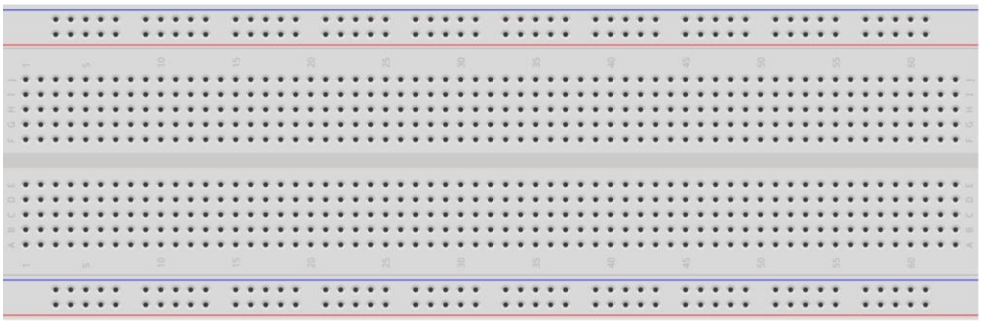
We have learned how to control a LED blinking, next we will learn how to control a number of LEDs.

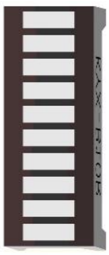
Project 3.1 Flowing Light

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

Component List

ESP8266 x1	USB cable
	

Breadboard x1


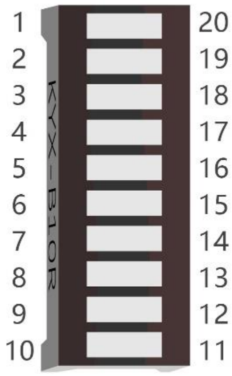
Jumper wire M/M x10	LED bar graph x1	Resistor 220Ω x9
		

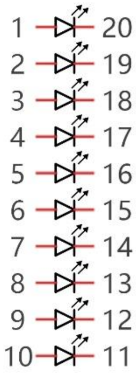
Component knowledge

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

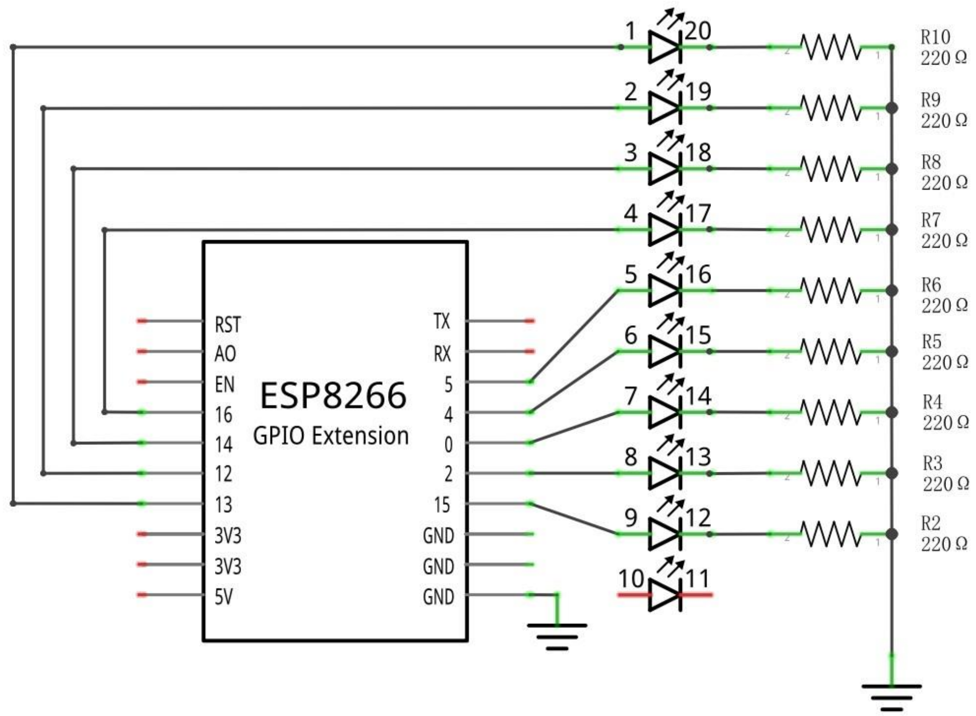
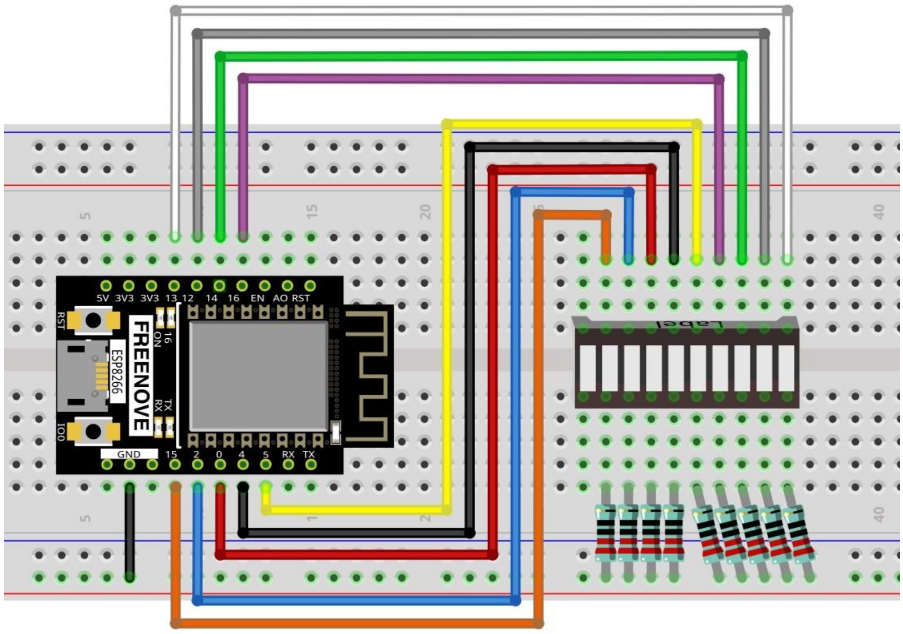
LED bar

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.

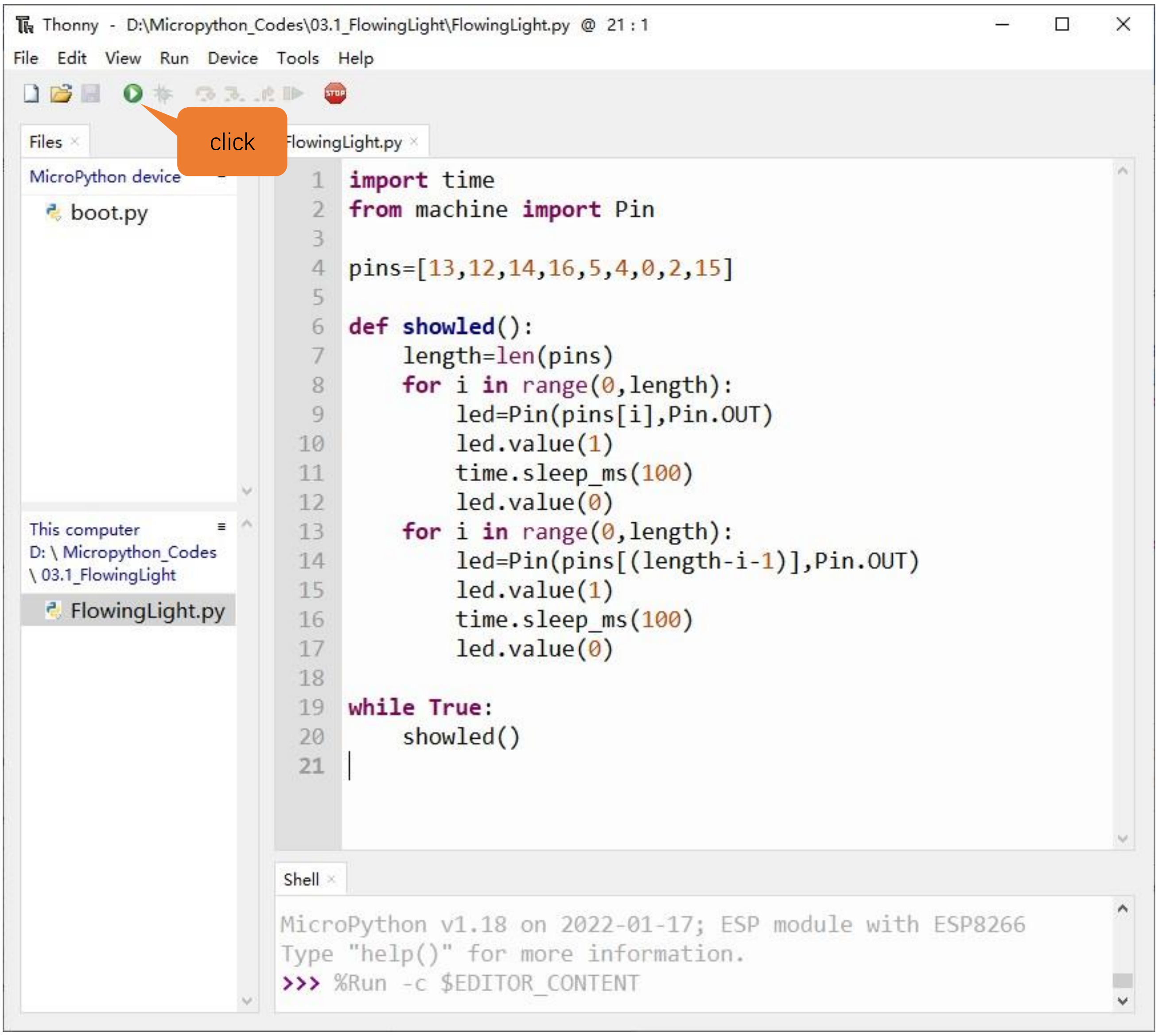
Code

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_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**” to disk(D) in advance with the path of “**D:/Micropython_Codes**”.

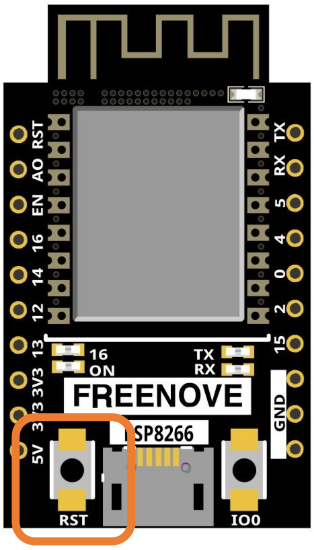
Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “03.1_FlowingLight” and double click “FlowingLight.py”.



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

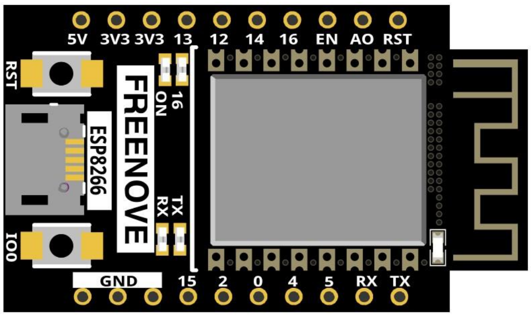
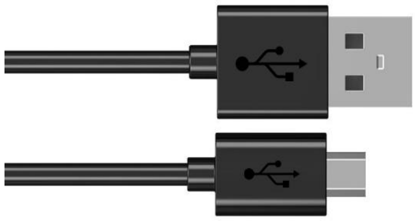
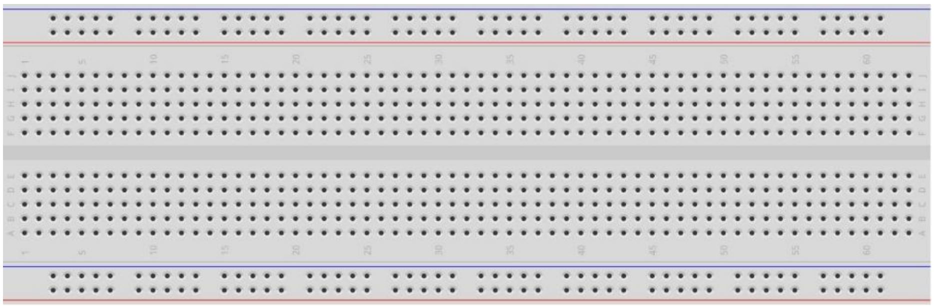
Chapter 4 Analog & PWM

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.

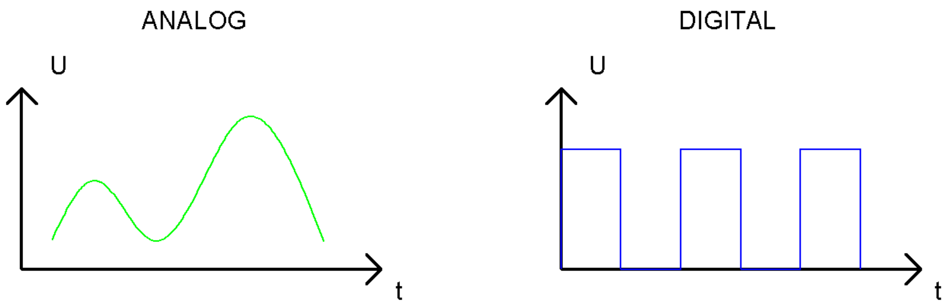
Component List

<table>
<tr><td colspan="2">ESP8266 x1

</td><td>USB cable
</td></tr>
<tr><td colspan="3">Breadboard x1
</td></tr>
<tr><td>LED x1
</td><td>Resistor 220Ω x1
</td><td>Jumper wire M/M x3
</td></tr>
</table>

Related knowledge

Analog & Digital

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



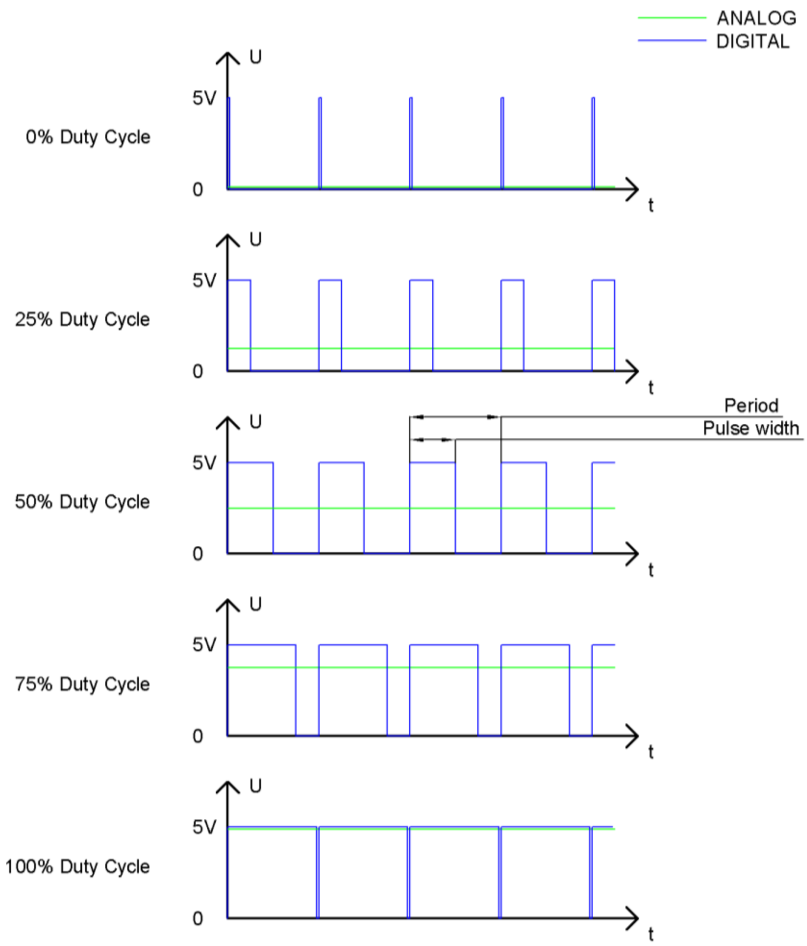
In practical application, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1), it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

PWM

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

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

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

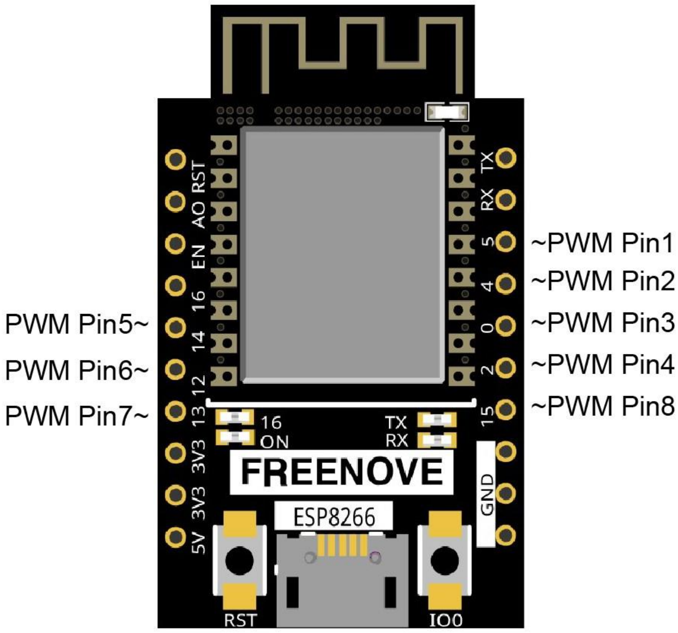


The longer the PWM duty cycle is, the higher the output power will be. Now that we understand this relationship, we can use PWM to control the brightness of an LED or the speed of DC motor and so on.
It is evident from the above that PWM is not real analog, and the effective value of the voltage is equivalent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

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:

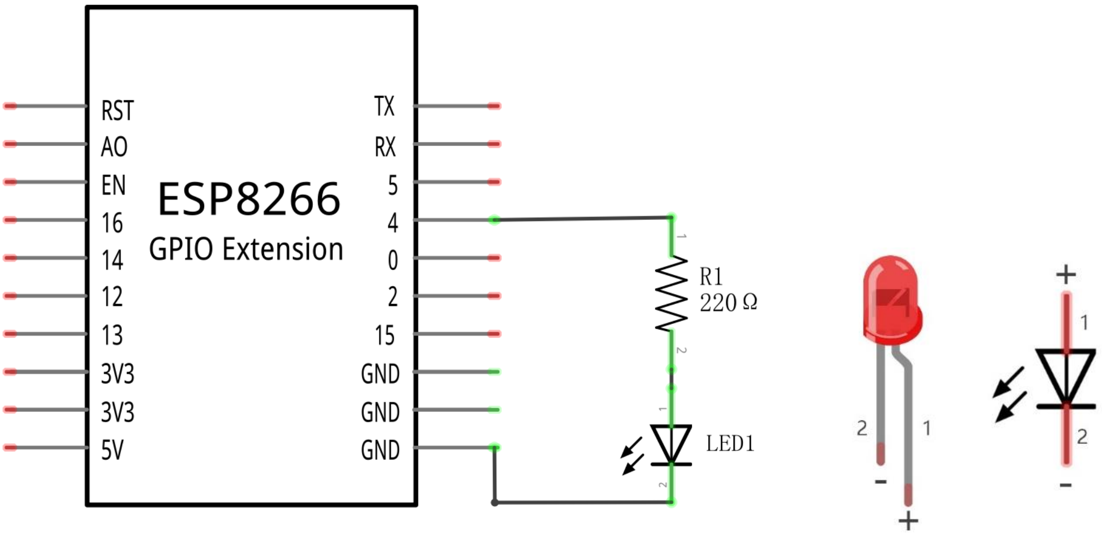


The longer the PWM duty cycle is, the higher the output power will be. Now that we understand this relationship, we can use PWM to control the brightness of an LED or the speed of DC motor and so on.

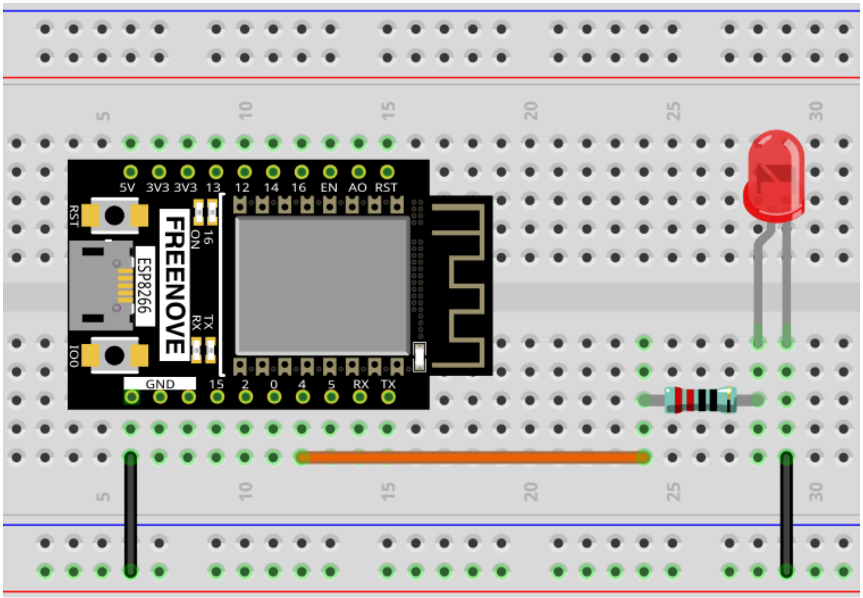
Circuit

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**



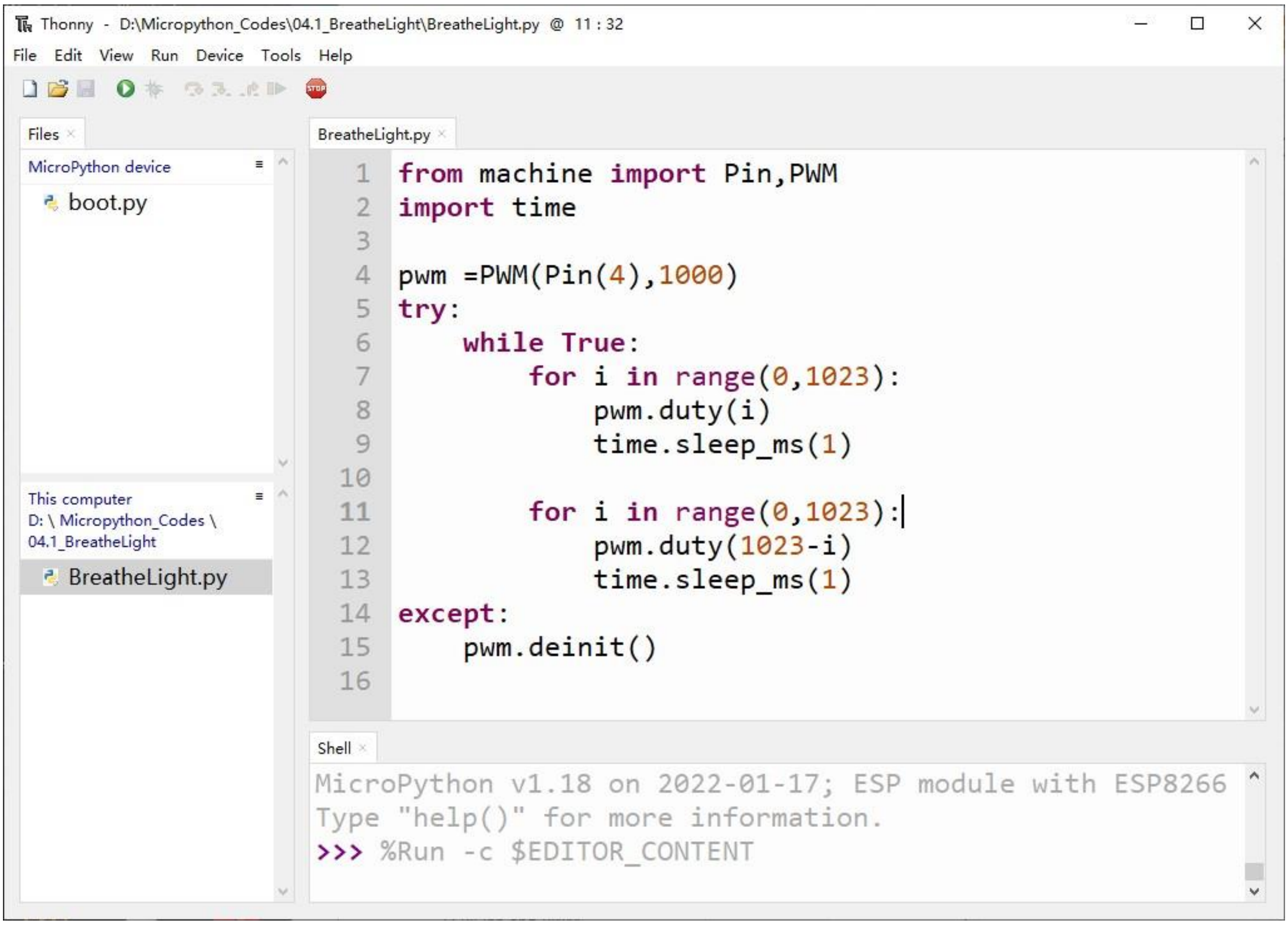
Code

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_Ultimate_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.

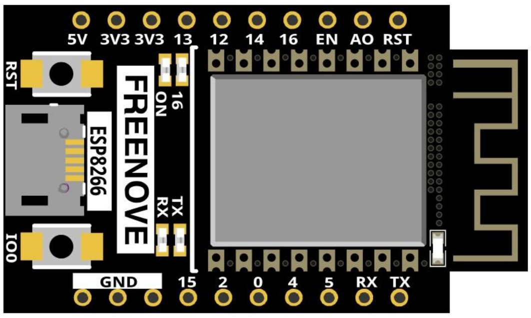
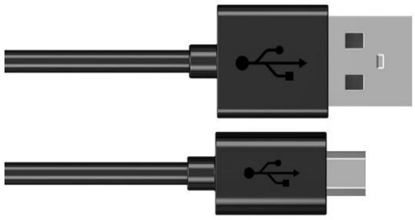
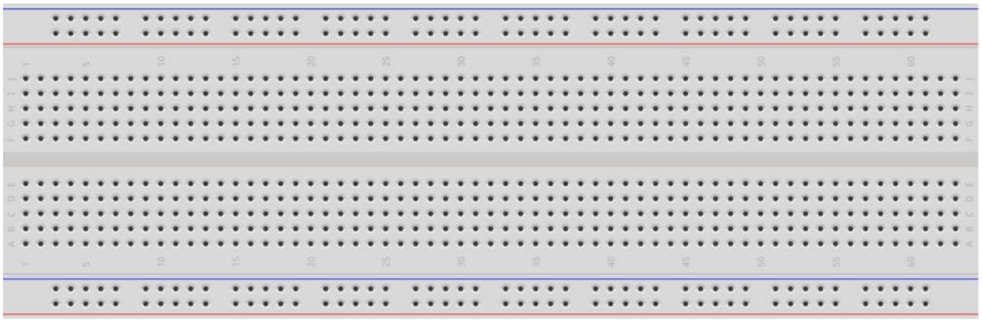
Reference

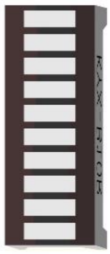
Class PWM(pin, freq)
Before each use of PWM module, please add the statement "**from machine import PWM**" to the top of the python file. **pin**: PWM can be enabled on all pins except pin (16), such as Pin(0)、Pin(2)···. **freq**: Output frequency, with the range of 0-1000 Hz **duty**: Duty cycle, with 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.

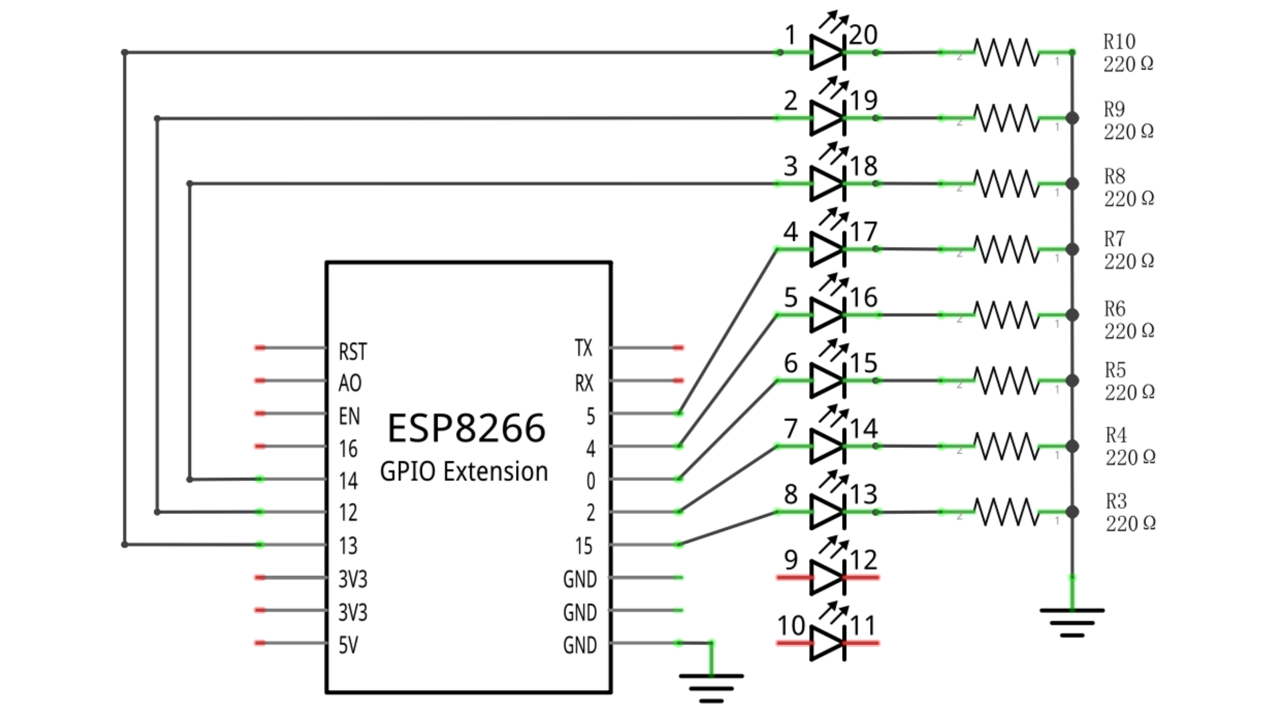
Component List

ESP8266 x1 	USB cable 
Breadboard x1 	

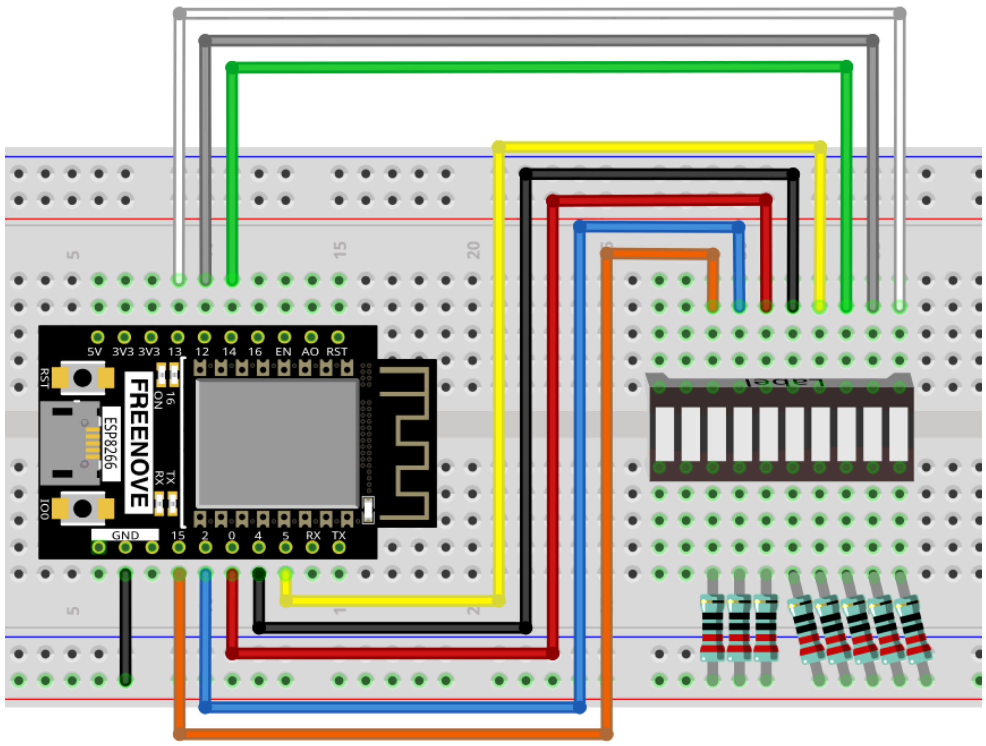
Jumper wire M/M x9 	LED bar graph x1 	Resistor 220Ω x8 

Circuit

Schematic diagram



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



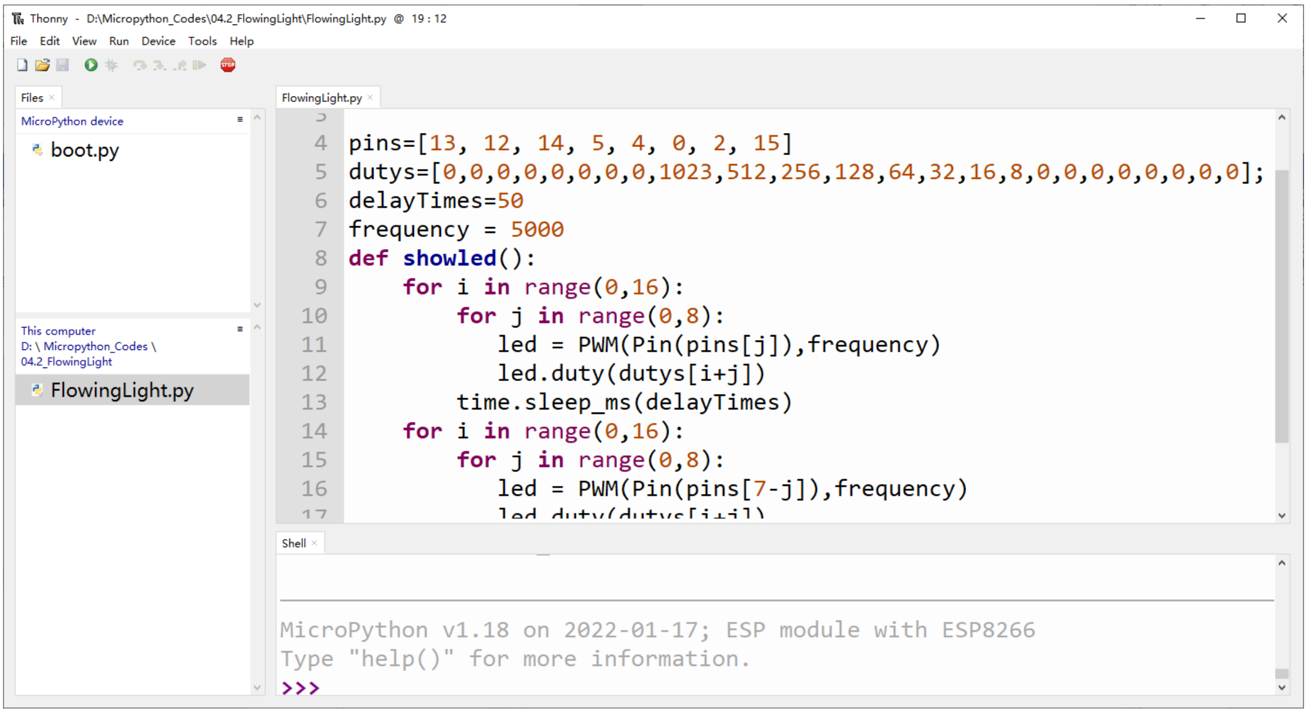
If LEDbar doesn't work, try to rotate LEDbar for 180°. The label is random.

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.

```
    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)
```

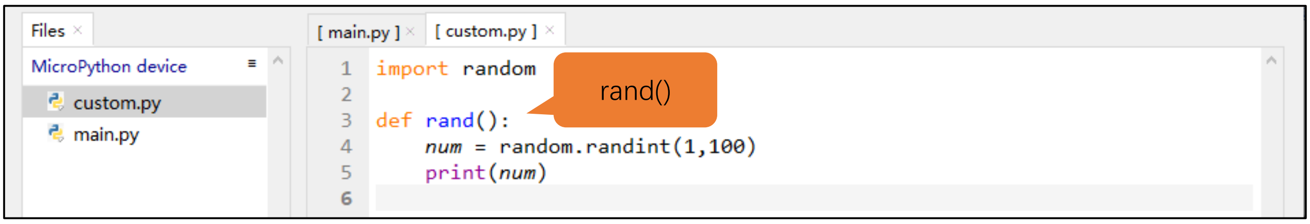
In the main function, a nested for loop is used to control the pulse width of the PWM. Every time i in the first for loop increases by 1, the LED Bar Graph will move one grid, and gradually change according to the value in the array dutys. As shown in the following table, the value in the second row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

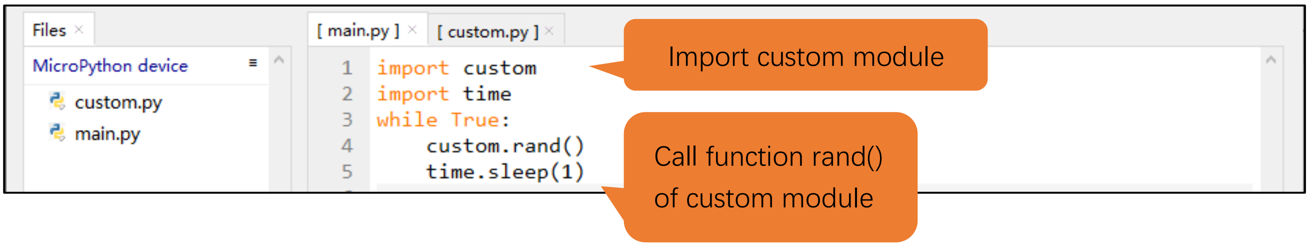
How to import a custom python module

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



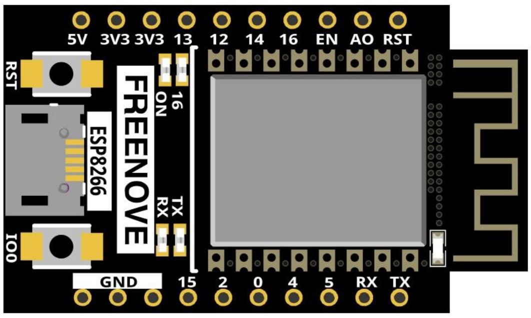
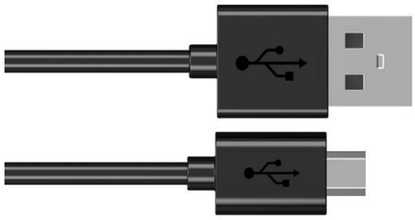
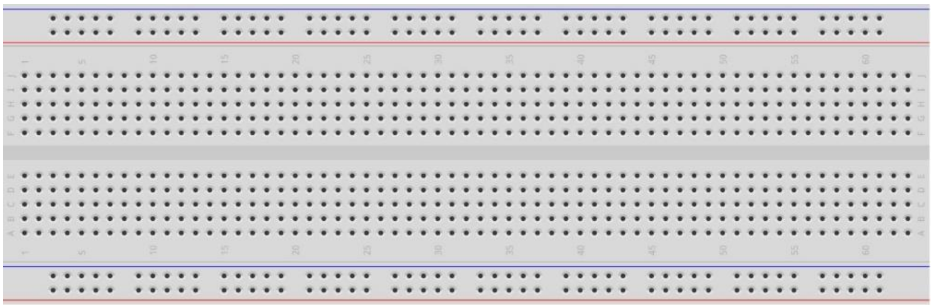
Chapter 5 RGBLED

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

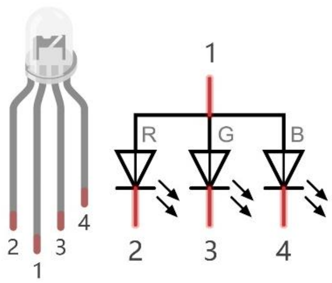
In this project, we will make a multicolored LED. And we can control RGBLED to switch different colors automatically.

Component List

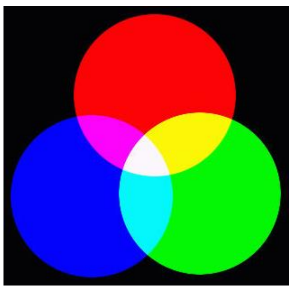
ESP8266 x1 		USB cable 
Breadboard x1 		
RGBLED x1 	Resistor 220Ω x3 	Jumper wire M/M x5 

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.

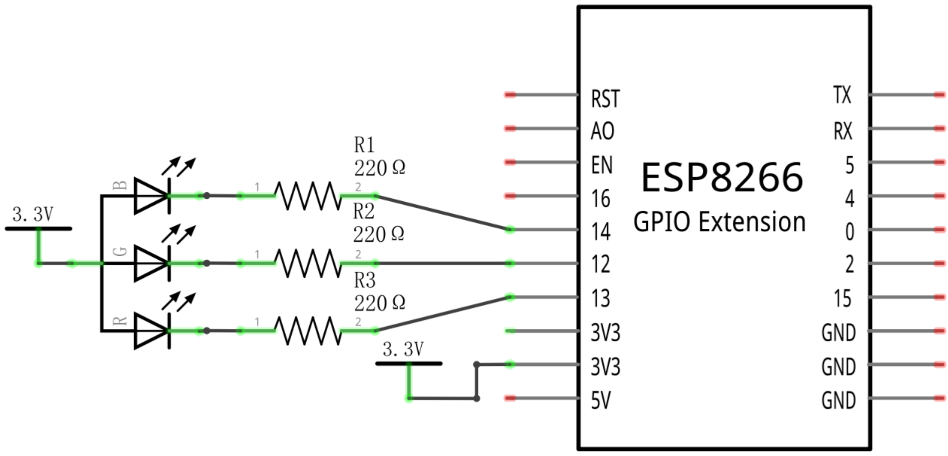


RGB

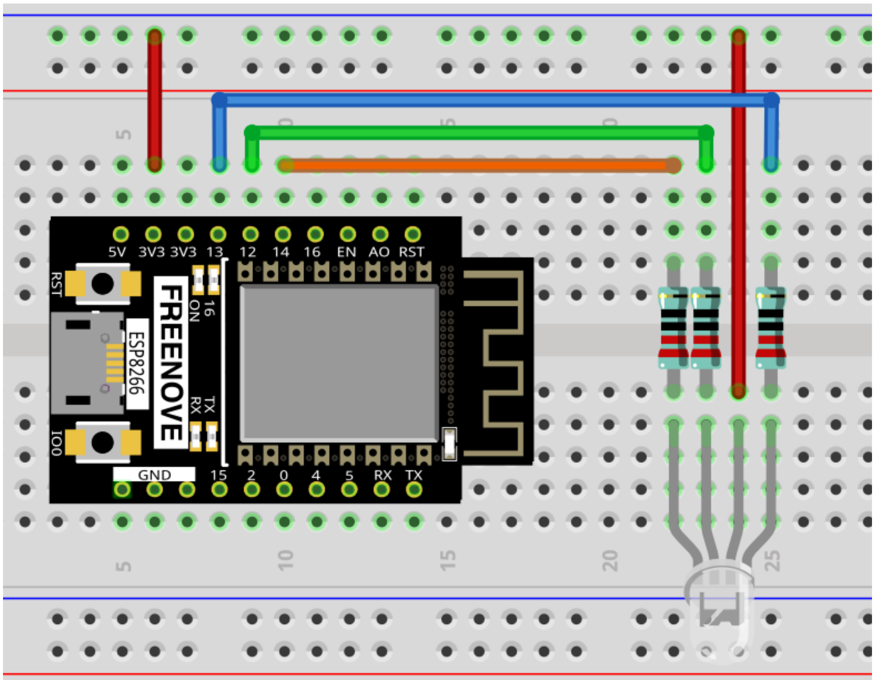
If we use three 10-bit PWM to control the RGBLED, in theory, we can create $2^{10}*2^{10}*2^{10}$= 1,073,741,824(1 billion) colors through different combinations.

Circuit

Schematic diagram



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



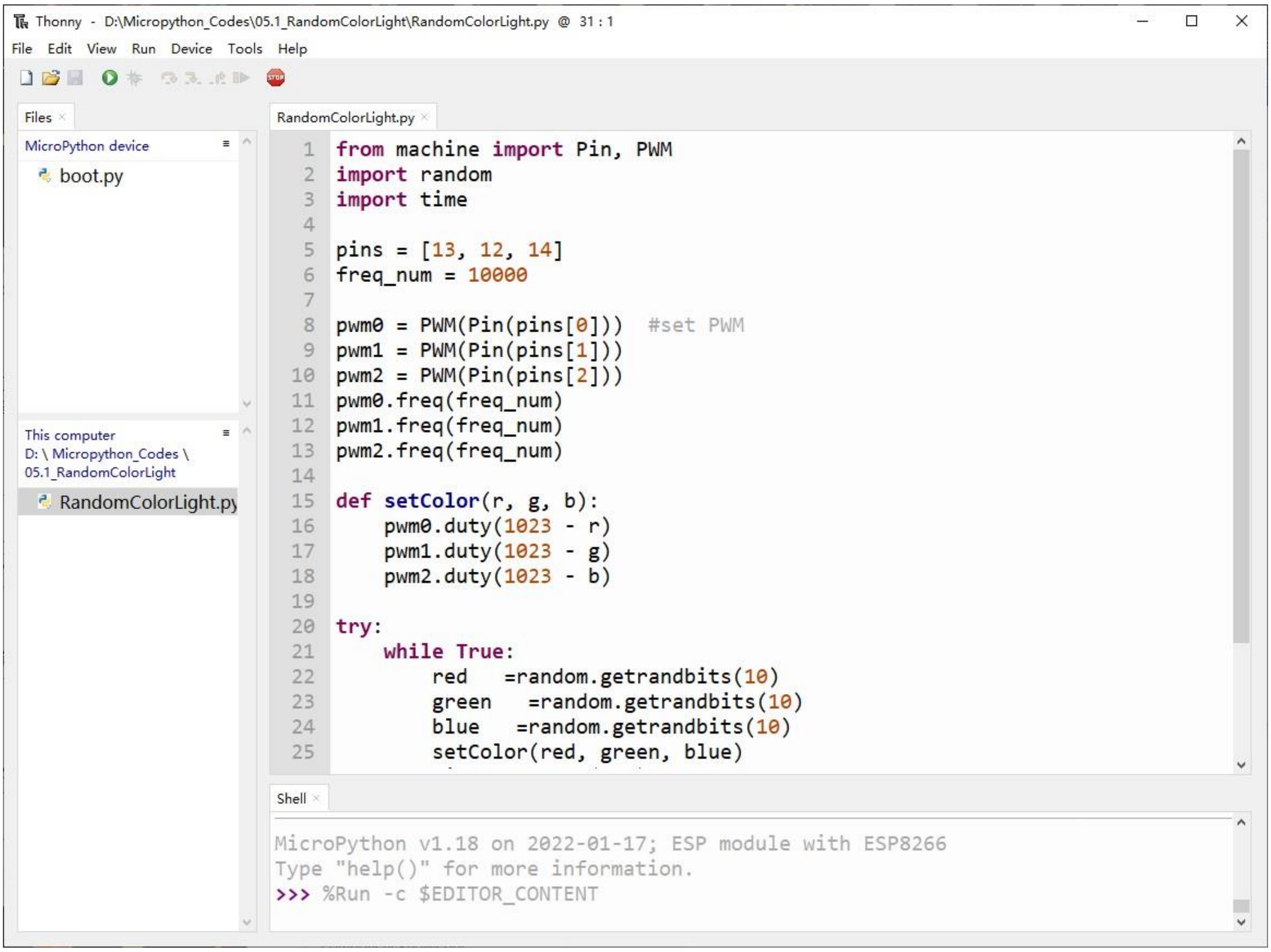
Code

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

Move the program folder "**Freenove_Ultimate_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.

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

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

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

```
    while True:
        red   =random.getrandbits(10)
        green =random.getrandbits(10)
        blue  =random.getrandbits(10)
        setColor(red,green,blue)
        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 program code:

```
from machine import Pin,PWM
import time

pins=[14,12,13];

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

red=0                    #red
```

```
green=0                 #green
blue=0                  #blue

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

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

    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)
```

```
    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
```

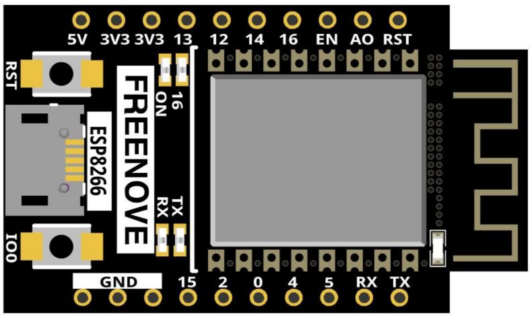
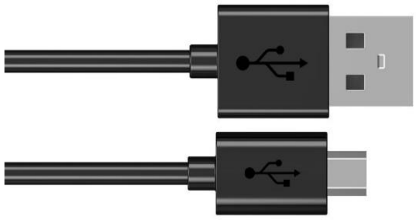
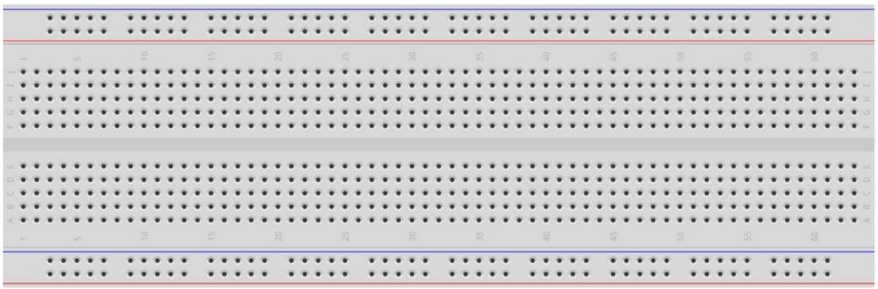
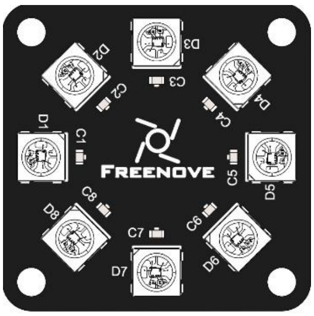
Chapter 6 NeoPixel

This chapter will help you learn to use a more convenient RGBLED lamp, which requires only one GPIO control and can be connected in infinite series in theory. Each LED can be controlled independently.

Project 6.1 NeoPixel

Learn the basic usage of NeoPixel and use it to flash red, green, blue and white.

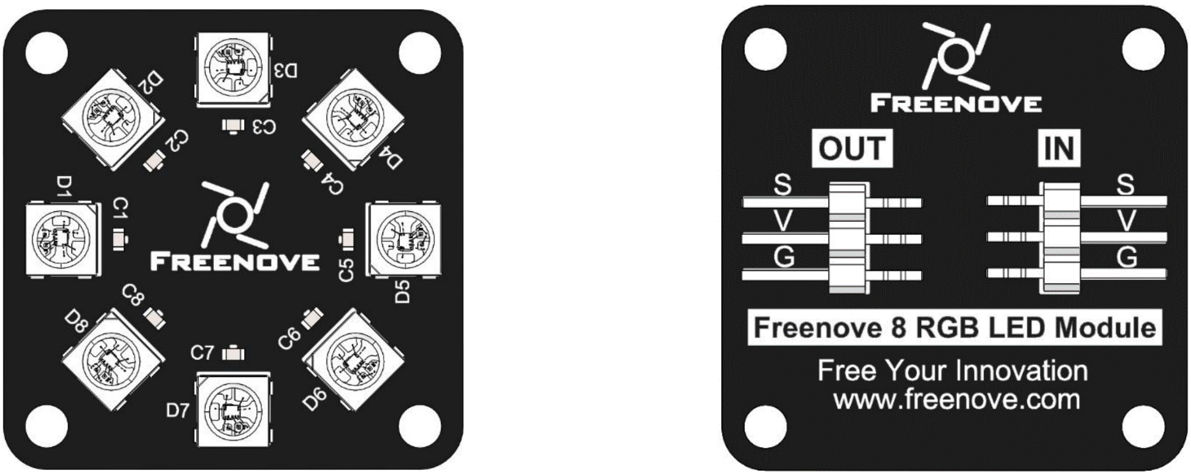
Component List

ESP8266 x1 	USB cable 
Breadboard x1 	
Freenove 8 RGB LED Module x1 	Jumper wire F/M x4 

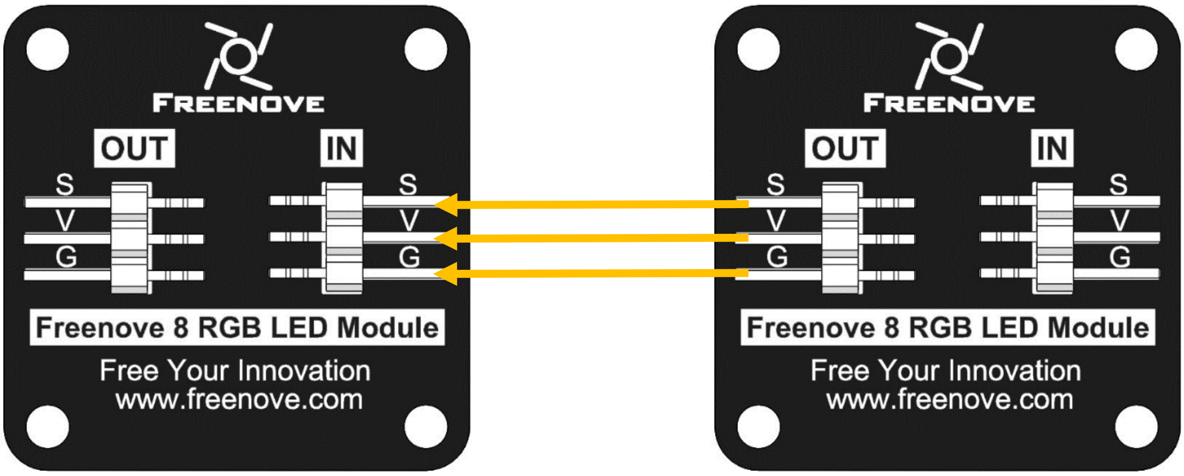
Related knowledge

Freenove 8 RGB LED Module

The Freenove 8 RGB LED Module is as below. You can use only one data pin to control eight LEDs on the module. As shown below:



And you can also control many modules at the same time. Just connect OUT pin of one module to IN pin of another module. In this way, you can use one data pin to control 8, 16, 32 … LEDs.

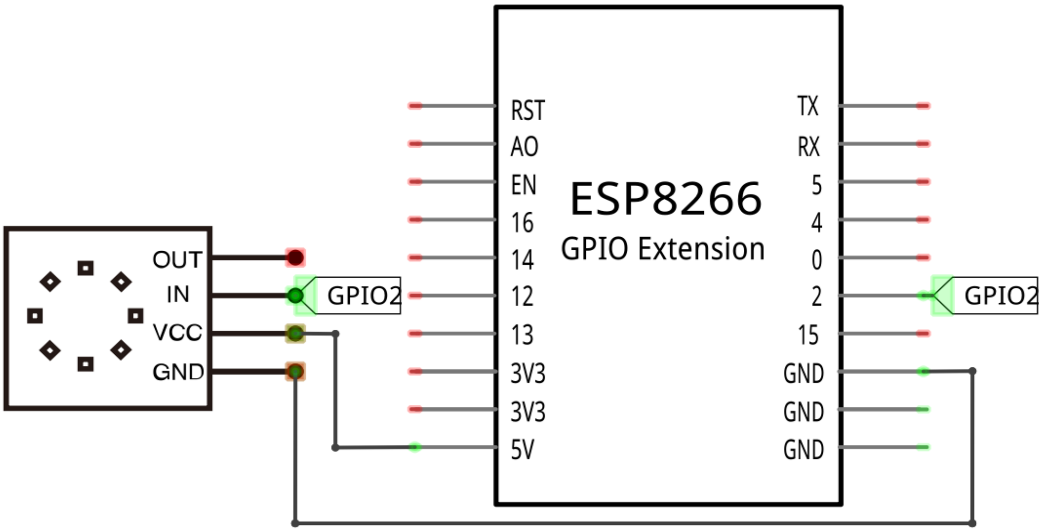


Pin description:

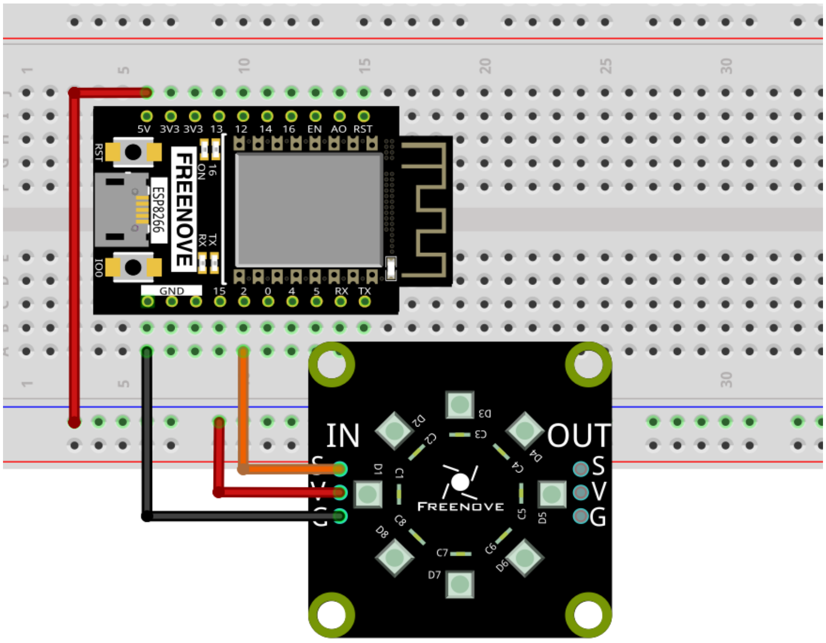
(IN)		(OUT)	
symbol	Function	symbol	Function
S	Input control signal	S	Output control signal
V	Power supply pin, +3.5V~5.5V	V	Power supply pin, +3.5V~5.5V
G	GND	G	GND

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_Ultimate_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" → "06.1_Neopixel" and double click "Neopixel.py".

06.1_Neopixel

Click "Run current script", and Neopixel begins to light up in red, green, blue, white and black.

The following is the program code:

```
from machine import Pin
import neopixel
import time
pin = Pin(2, Pin.OUT)
np = neopixel.NeoPixel(pin, 8)

#brightness :0-255
brightness=10
colors=[[brightness,0,0],                    #red
        [0,brightness,0],                    #green
        [0,0,brightness],                    #blue
        [brightness,brightness,brightness],  #white
        [0,0,0]]                             #close

while True:
    for i in range(0,5):
        for j in range(0,8):
            np[j]=colors[i]
            np.write()
            time.sleep_ms(50)
        time.sleep_ms(500)
    time.sleep_ms(500)
```

Import Pin, neopiexl and time modules.

```
from machine import Pin
import neopixel
import time
```

Define the number of pin and LEDs connected to neopixel.

```
pin = Pin(2, Pin.OUT)
np = neopixel.NeoPixel(pin, 8)
```

Define the brightness of neopixel's LED and an array to store color.

```
#brightness :0-255
brightness=10
colors=[[brightness,0,0],                    #red
        [0,brightness,0],                    #green
        [0,0,brightness],                    #blue
        [brightness,brightness,brightness],  #white
        [0,0,0]]                             #close
```

Assign the color data to the array np and call function write() to send np array data to neopixel module.

```
            np[j]=colors[i]
            np.write()
```

Nest two for loops to make the module repeatedly display five states of red, green, blue, white and OFF.

```
while True:
    for i in range(0,5):
        for j in range(0,8):
            np[j]=colors[i]
            np.write()
            time.sleep_ms(50)
        time.sleep_ms(500)
    time.sleep_ms(500)
```

Reference

Class neopixel

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

NeoPixel(pin, n): Define the number of output pins and LEDs of neopixel module

  pin: Output pins

  n: The number of LEDs.

NeoPixel.write(): Write data to LEDs.

Project 6.2 Rainbow Light

In the previous project, we have mastered the usage of NeoPixel. This project will realize a slightly complicated Rainbow Light. The component list and the circuit are exactly the same as the project fashionable Light.

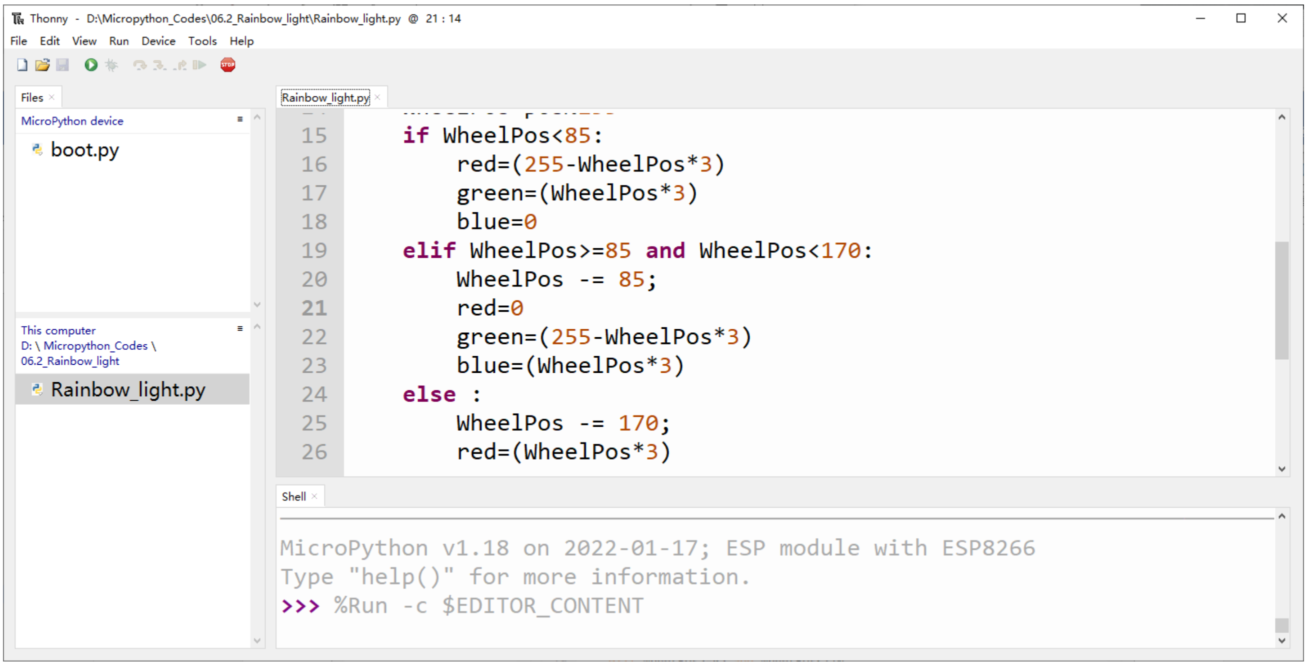
Code

Continue to use the following color model to equalize the color distribution of the 8 leds and gradually change.

Move the program folder "**Freenove_Ultimate_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" → "06.2_Rainbow_light" and then double click "Rainbow_light.py".

06.2_Rainbow_light



Click “Run current script”, and the Freenove 8 RGB LED Strip displays different colors and the color changes gradually.

The following is the program code:

```
from machine import Pin
import neopixel
import time
pin = Pin(2, Pin.OUT)
np = neopixel.NeoPixel(pin, 8)

brightness=0.1          #brightness: 0 - 1.0
red=0                   #red
green=0                 #green
blue=0                  #blue

def wheel(pos):
    global red,green,blue
    WheelPos=pos%255
    if WheelPos<85:
        red=(255-WheelPos*3)
        green=(WheelPos*3)
        blue=0
    elif WheelPos>=85 and WheelPos<170:
        WheelPos -= 85;
        red=0
        green=(255-WheelPos*3)
        blue=(WheelPos*3)
    else :
        WheelPos -= 170;
        red=(WheelPos*3)
        green=0
        blue=(255-WheelPos*3)

while True:
```

```
    for i in range(0,255):
        for j in range(0,8):
            wheel(i+j*255//8)
            np[j]=(int(red*brightness),int(green*brightness),int(blue*brightness))
            np.write()
        time.sleep_ms(5)
```

Define a wheel() function to process the color data of neopixel module.

```
def wheel(pos):
    global red,green,blue
    WheelPos=pos%255
    if WheelPos<85:
        red=(255-WheelPos*3)
        green=(WheelPos*3)
        blue=0
    elif WheelPos>=85 and WheelPos<170:
        WheelPos -= 85;
        red=0
        green=(255-WheelPos*3)
        blue=(WheelPos*3)
    else :
        WheelPos -= 170;
        red=(WheelPos*3)
        green=0
        blue=(255-WheelPos*3)
```

Set the color brightness of the module.

```
brightness=0.1          #brightness: 0 - 1.0
```

Use a nesting of two for loops. The first for loop makes the value of i increase from 0 to 255 automatically and the wheel() function processes the value of i into data of the module's three colors; the second for loop writes the color data to the module.

```
    for i in range(0,255):
        for j in range(0,8):
            wheel(i+j*255//8)
            np[j]=(int(red*brightness),int(green*brightness),int(blue*brightness))
            np.write()
        time.sleep_ms(5)
```

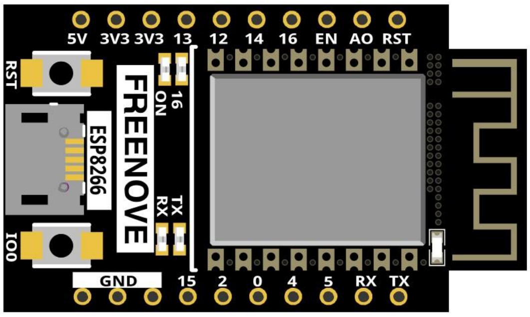
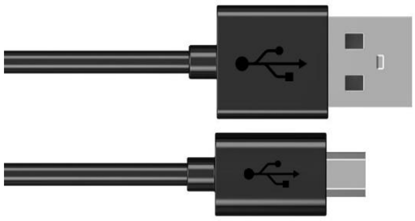
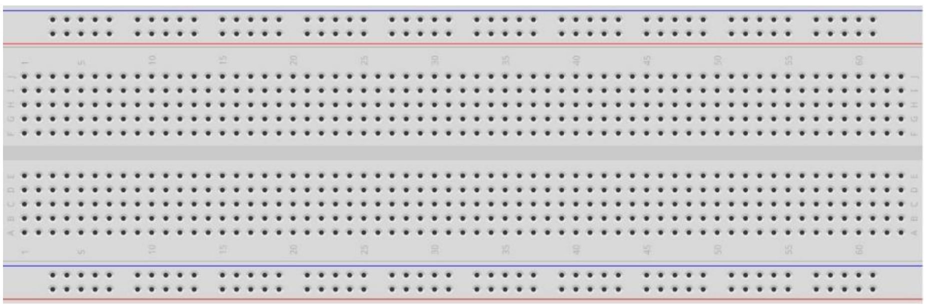
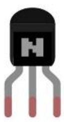
Chapter 7 Buzzer

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

Project 7.1 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

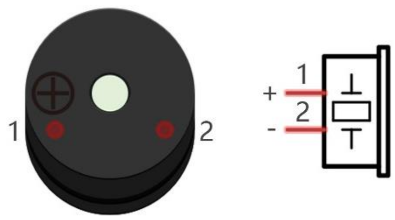
<table>
<tr><td colspan="3">ESP8266 x1

</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

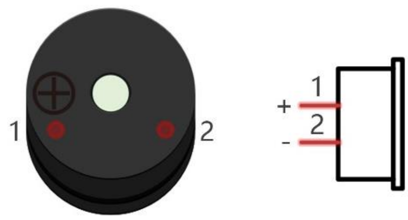
Buzzer

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

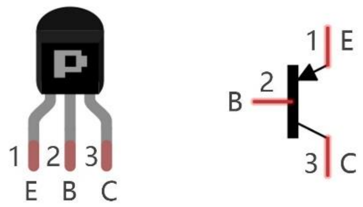
Passive buzzer

Transistor

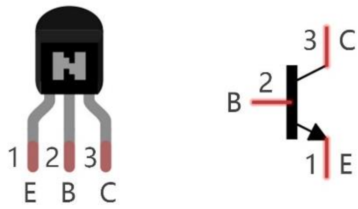
Because the buzzer requires such large current that GPIO of ESP8266 output capability cannot meet the requirement, a transistor of NPN type is needed here to amplify the current.

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

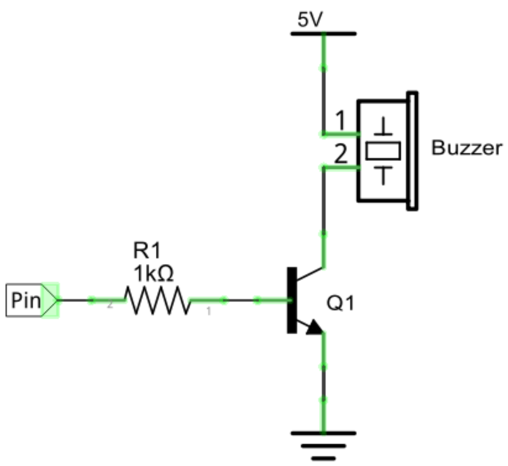
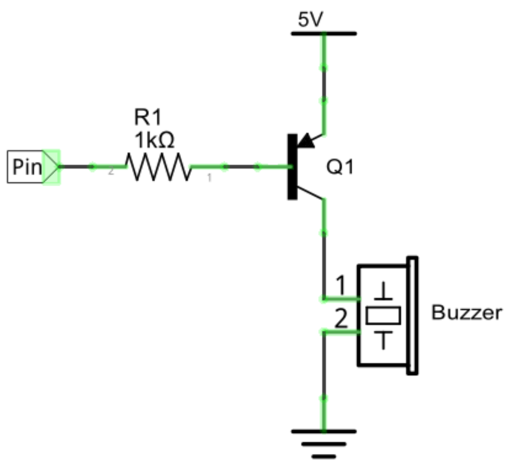


In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

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.

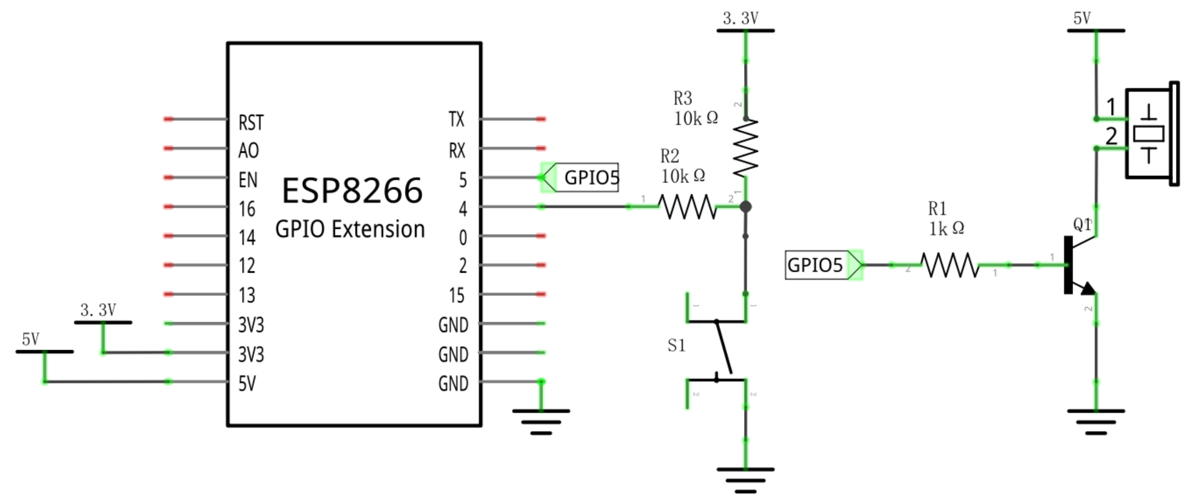
When using NPN transistor to drive buzzer, we often adopt the following method. If GPIO outputs high level, current will flow through R1, the transistor will get conducted, and the buzzer will sound. If GPIO outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer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

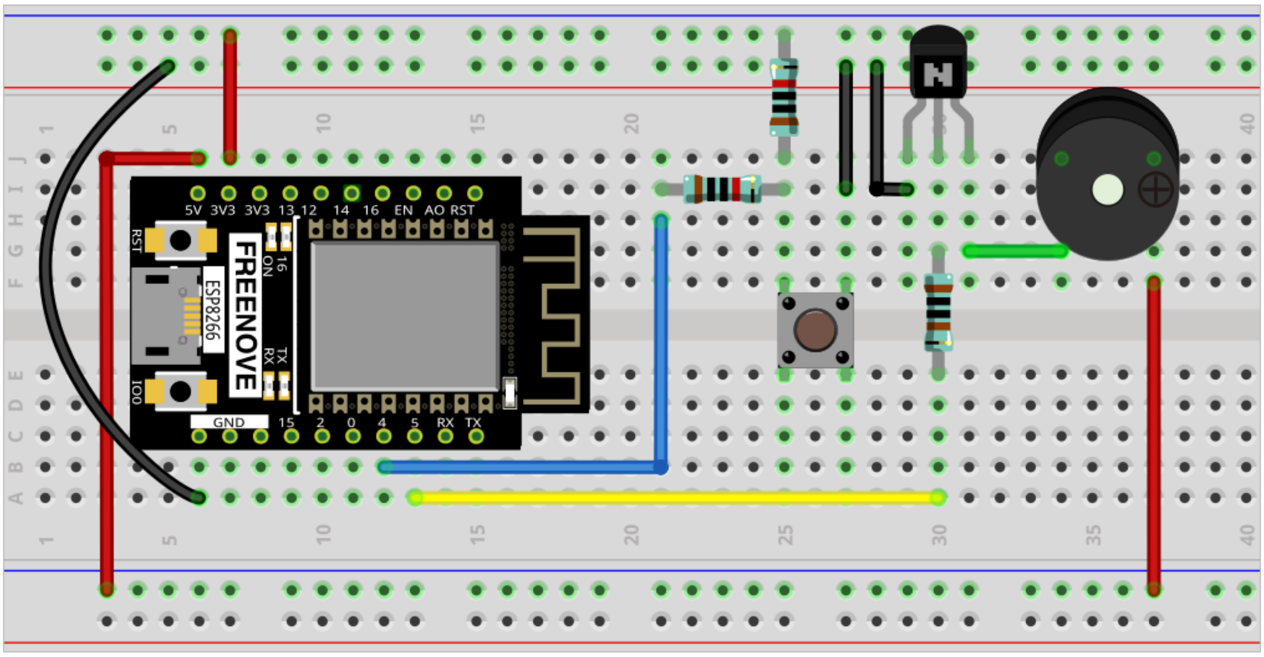
NPN transistor to drive buzzer	PNP transistor to drive buzzer
	

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.

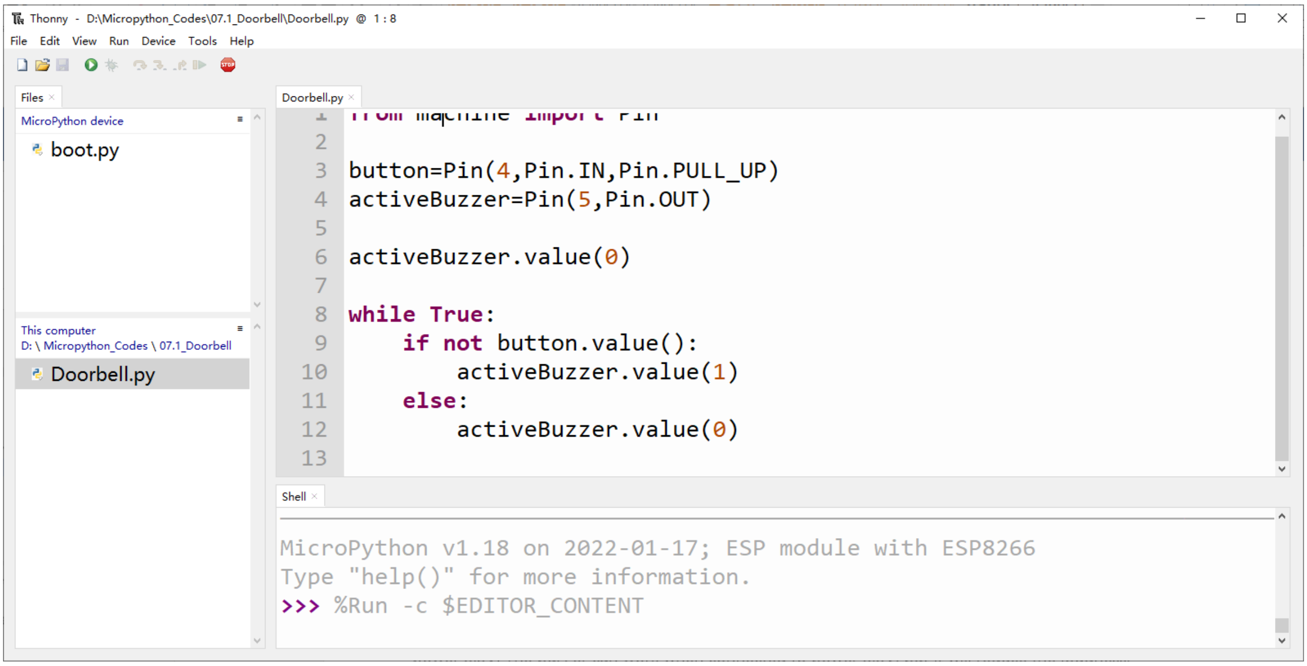
Code

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.

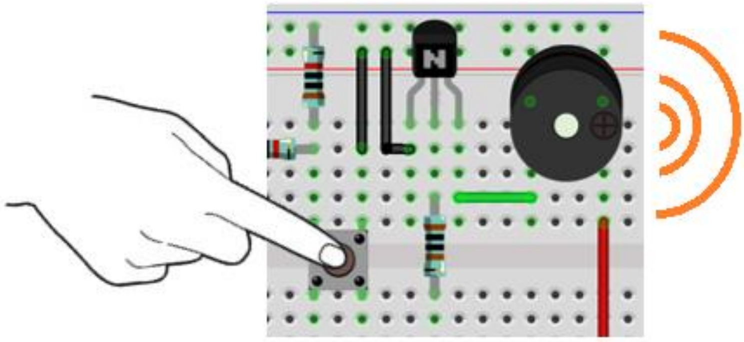
Move the program folder "**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

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 7.2 Alertor

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

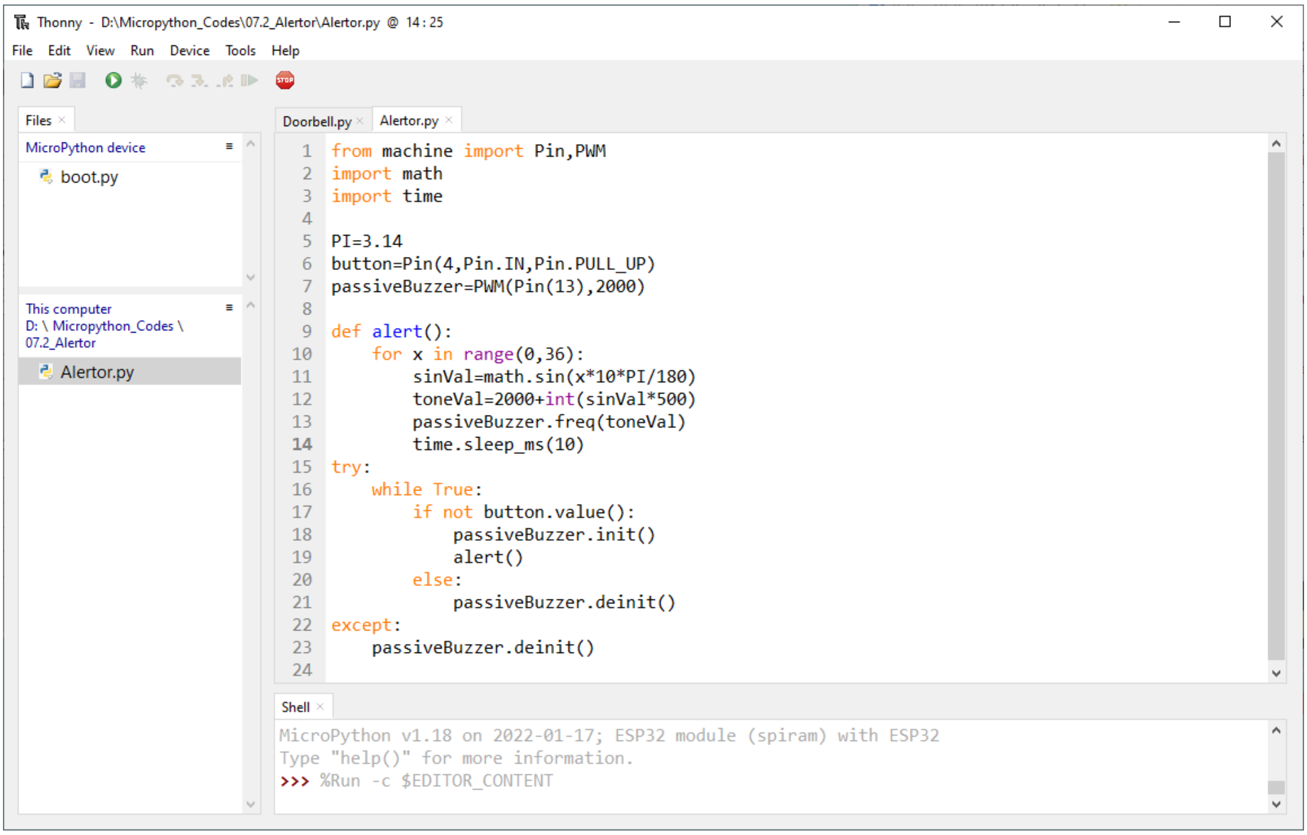
Component list and the circuit part is similar to last section. In the Doorbell circuit only the **active buzzer** needs to be **replaced** with a **passive buzzer.**

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

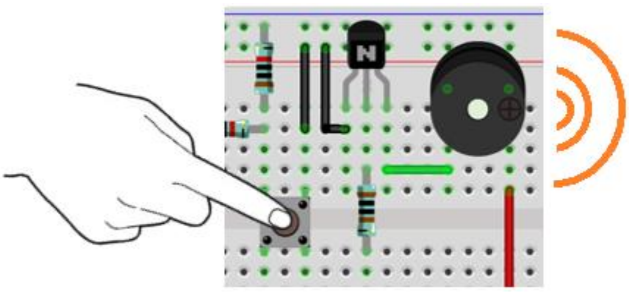



```python
from machine import Pin,PWM
import math
import time

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

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

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



The following is the program code:

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

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

def alert():
    for x in range(0,36):
        sinVal=math.sin(x*10*PI/180)
        toneVal=500+int(sinVal*500)
        passiveBuzzer.duty(1000)
        passiveBuzzer.freq(toneVal)
        time.sleep_ms(10)
try:
    while True:
        if not button.value():
            passiveBuzzer.init()
            alert()
        else:
            passiveBuzzer.duty(0)
            passiveBuzzer.deinit()
except:
    passiveBuzzer.deinit()
```

Import PWM, Pin, math and time modules.

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

Define the pins of the button and passive buzzer.

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

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

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

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

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

```
        passiveBuzzer.deinit()
```

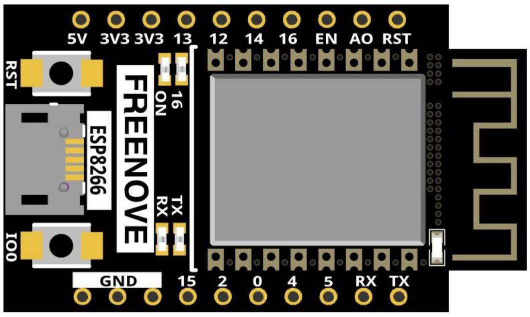
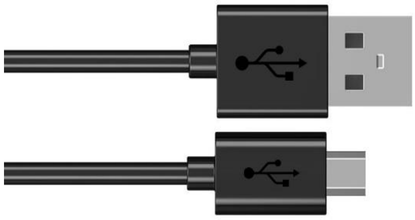
Chapter 8 Serial Communication

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

Project 8.1 Serial Print

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

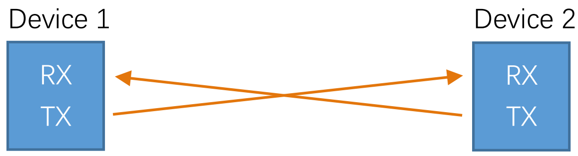
Component List

ESP8266 x1	USB cable
	

Related knowledge

Serial communication

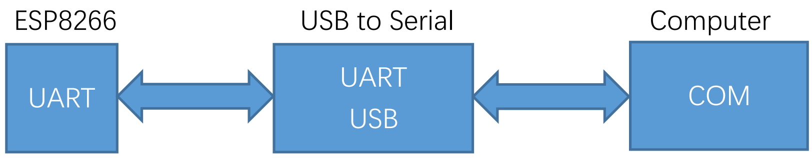
Serial communication generally refers to the Universal Asynchronous Receiver/Transmitter (UART), which is commonly used in electronic circuit communication. It has two communication lines, one is responsible for sending data (TX line) and the other for receiving data (RX line). The serial communication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

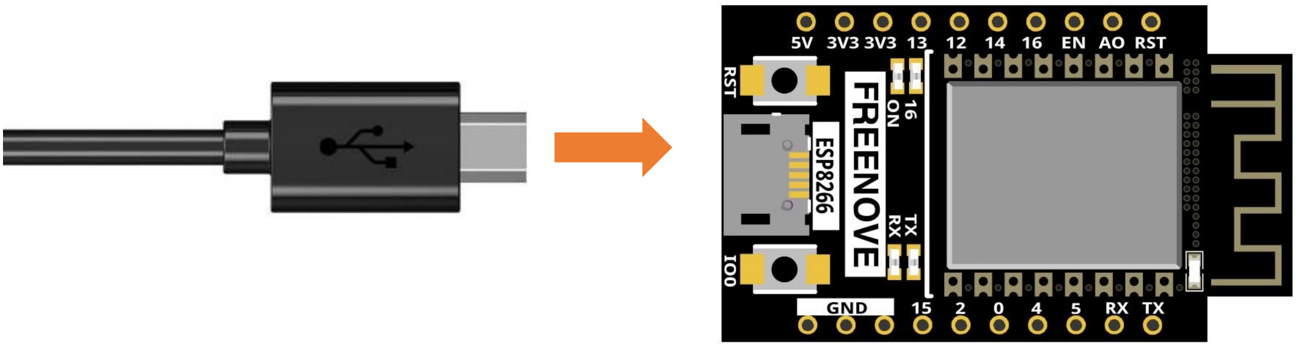
Serial port on ESP8266

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.

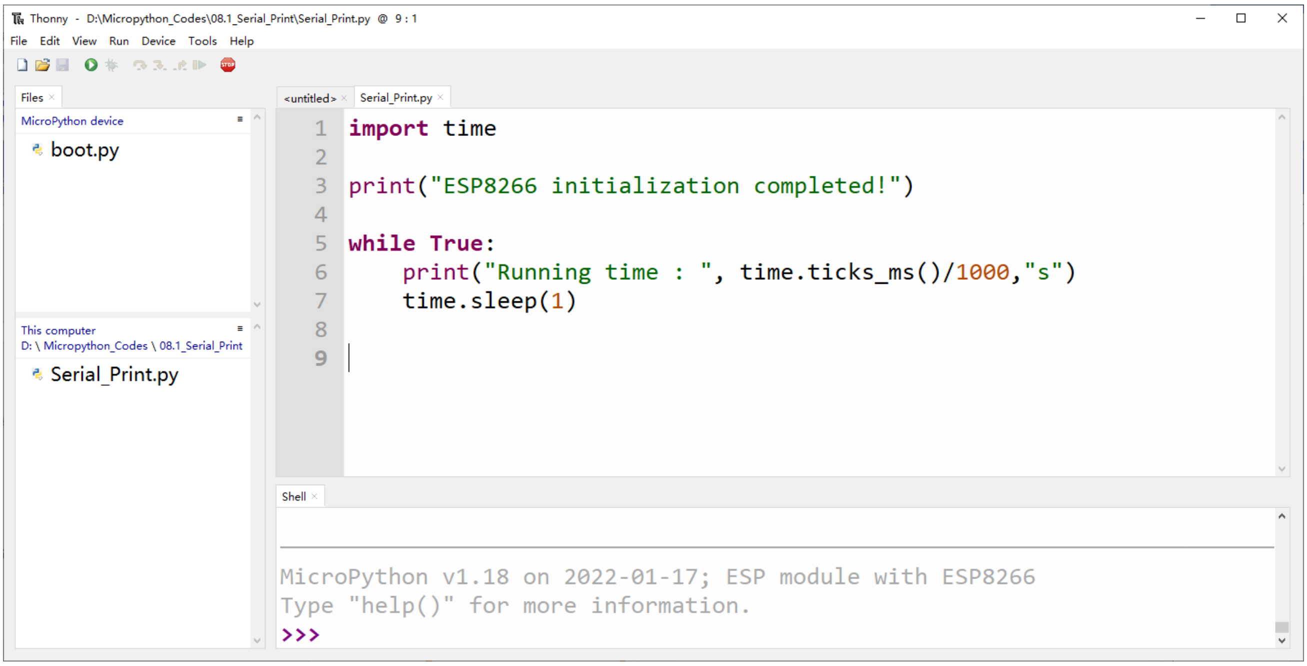


Code

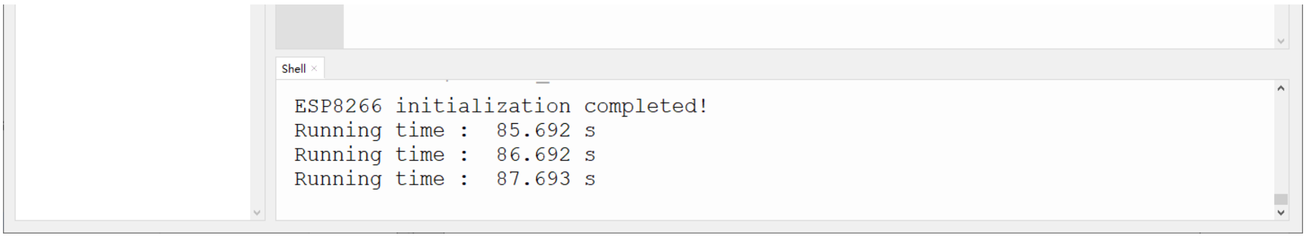
Move the program folder "**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

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:

```
import time

print("ESP8266 initialization completed!")

while True:
    print("Running time : ", time.ticks_ms()/1000, "s")
    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 8.2 Serial Read and Write

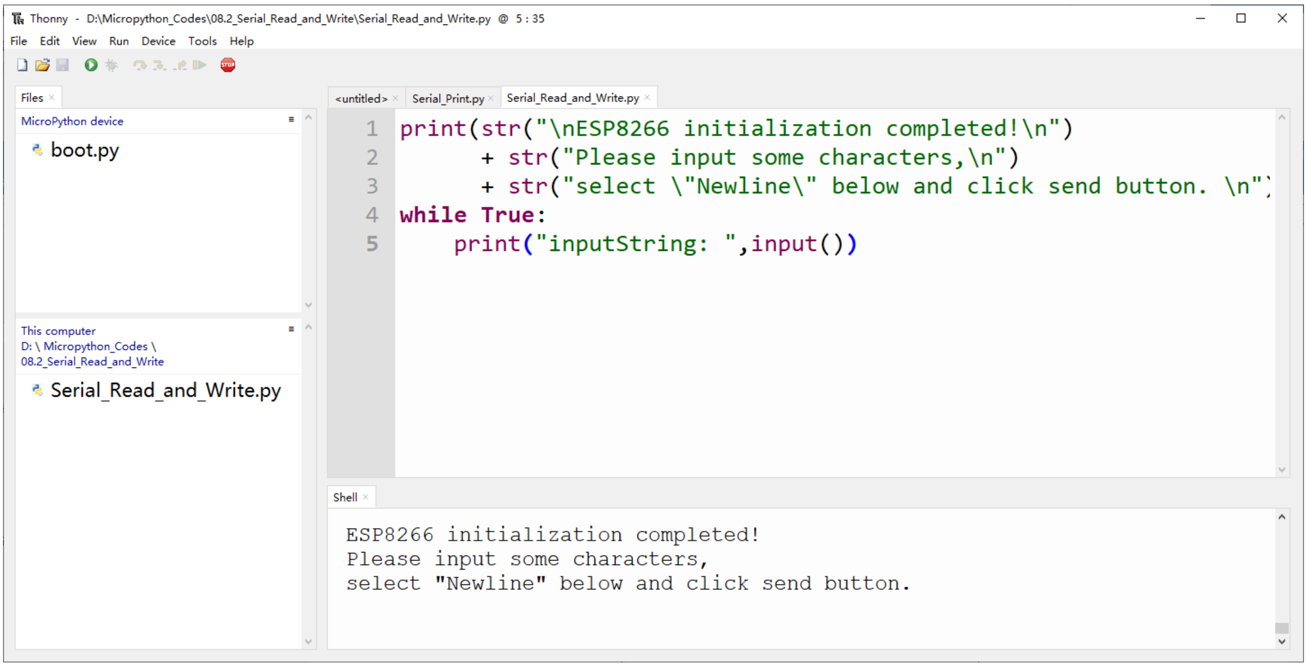
From last section, we use Serial port on Freenove ESP8266 to send data to a computer, now we will use that to receive data from computer.

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

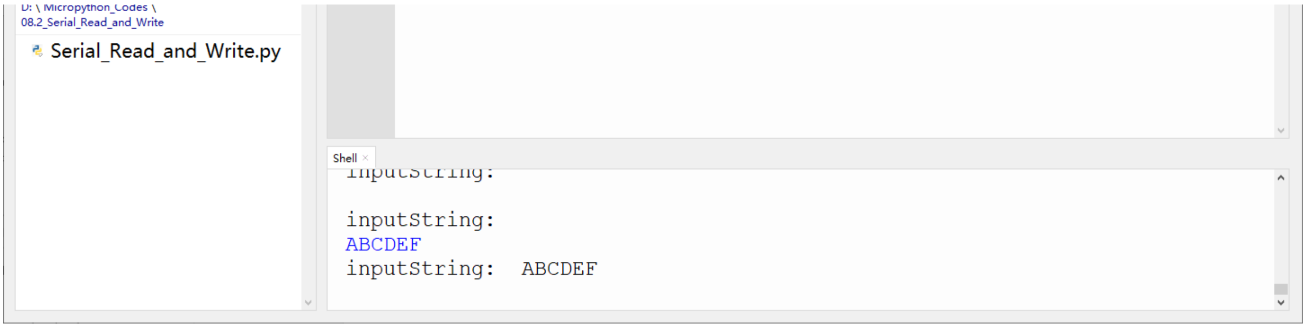
Code

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())
```

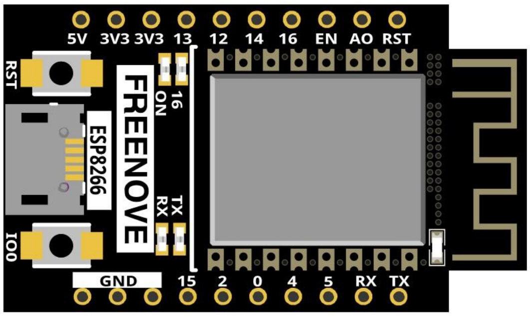
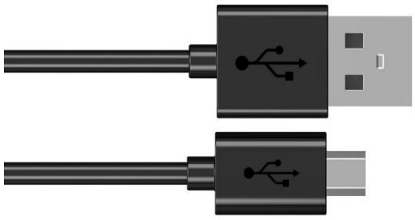
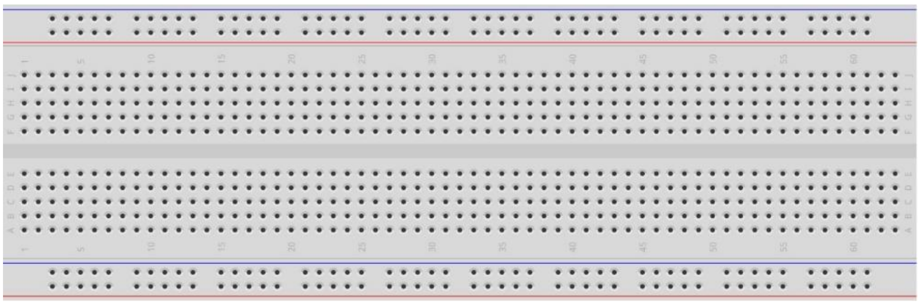
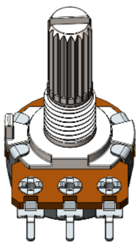
Chapter 9 ADC Converter

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 9.1 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.

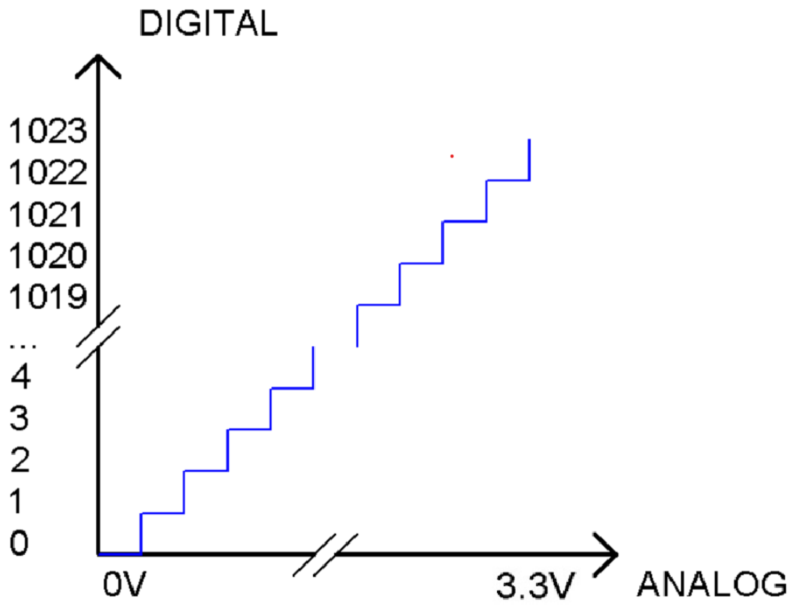
Component List

<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:

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

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!

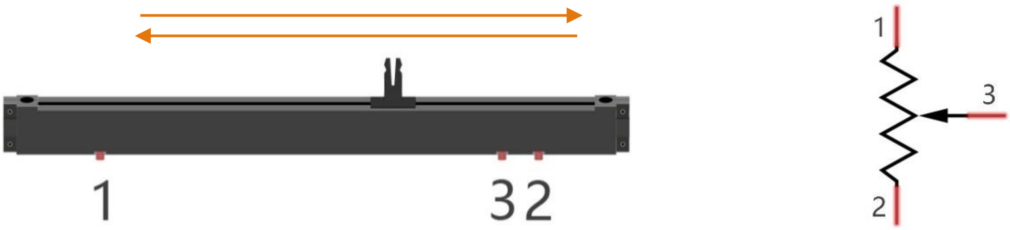
Pin number in ESP8266
A0

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

Component knowledge

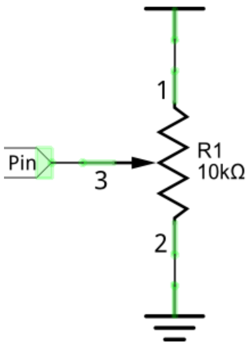
Potentiometer

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.



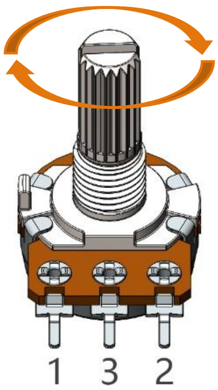
What between potentiometer pin 1 and pin 2 is the resistor body, and pins 3 is connected to brush. When brush moves from pin 1 to pin 2, the resistance between pin 1 and pin 3 will increase up to body resistance linearly, and the resistance between pin 2 and pin 3 will decrease down to 0 linearly.

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

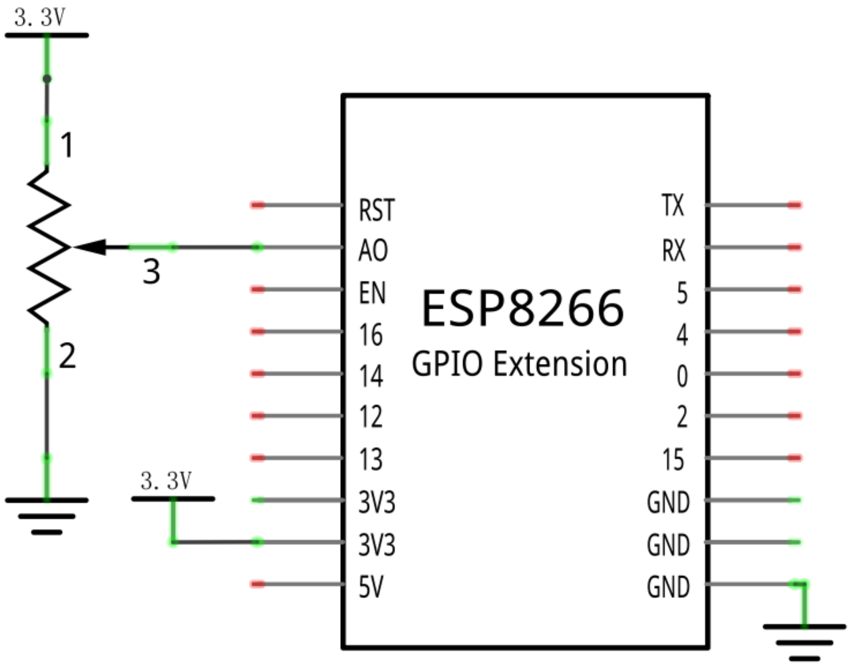
Rotary potentiometers and linear potentiometers have the same function; the only difference being the physical action being a rotational rather than a sliding movement.



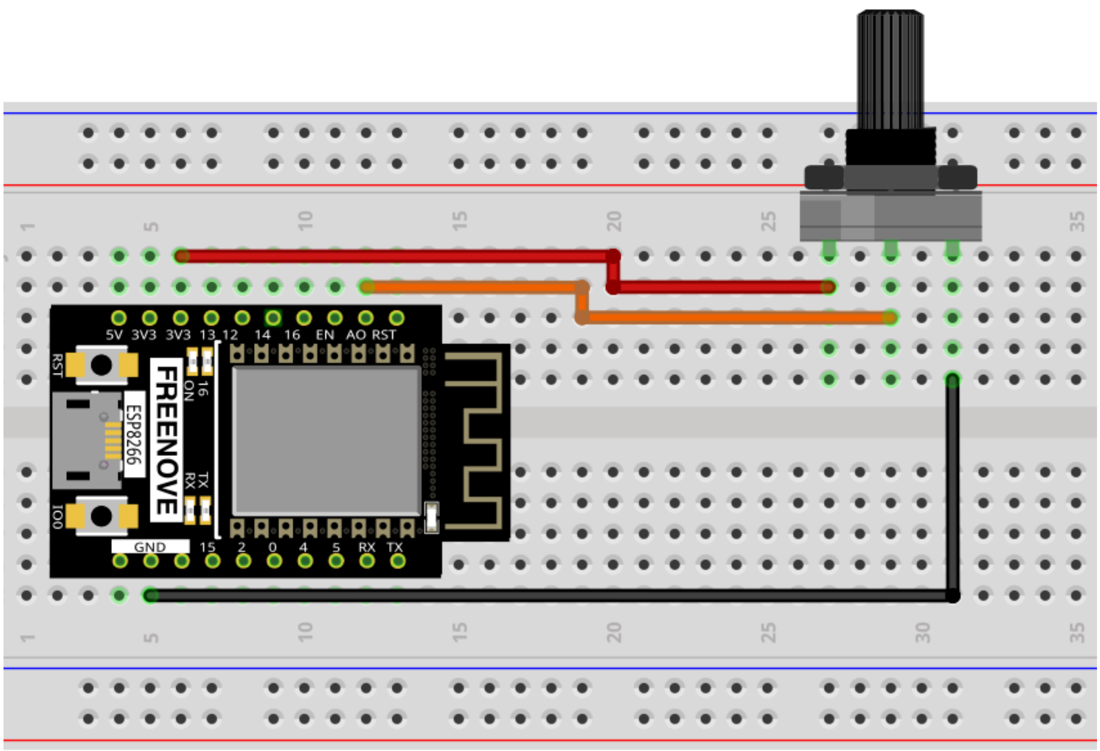
Circuit

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

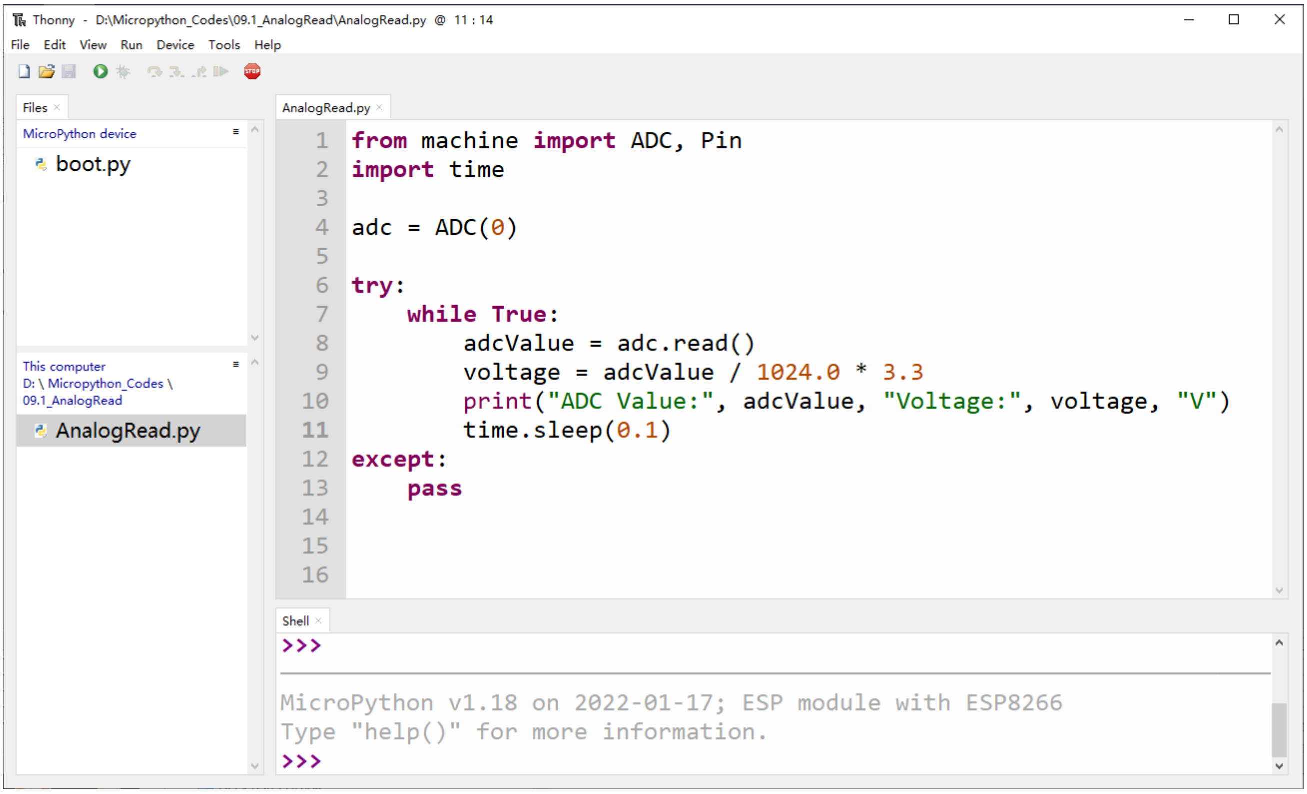


Code

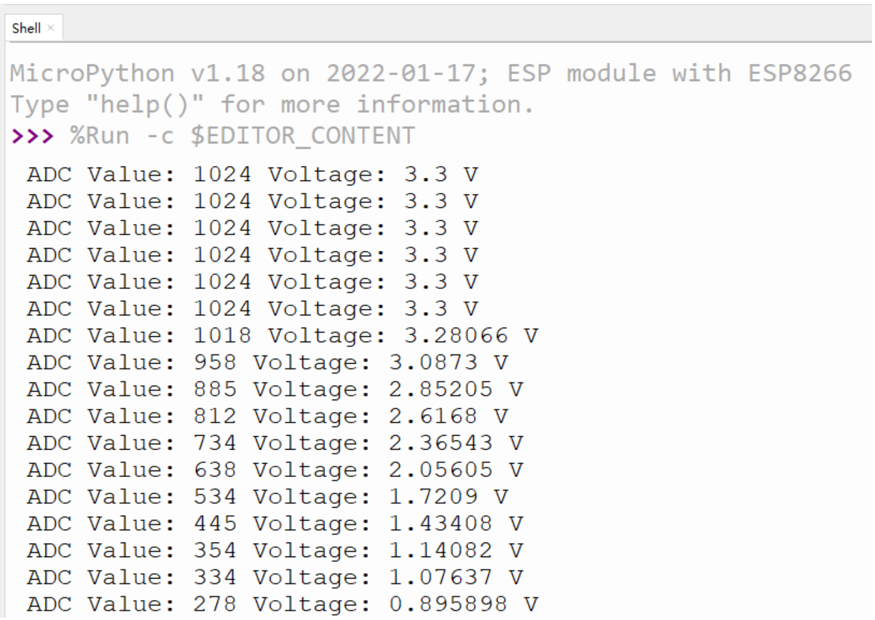
Move the program folder “**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**” to disk(D) in advance with the path of “**D:/Micropython_Codes**”.

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:

```
from machine import ADC, Pin
import time

adc = ADC(0)
try:
    while True:
        adcValue = adc.read()
        voltage = adcValue / 1024.0 * 3.3
        print("ADC Value:", adcValue, "Voltage:", voltage, "V")
        time.sleep(0.1)
except:
    pass
```

Import Pin, ADC and DAC modules.

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

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

```
        adcValue = adc.read()
        voltage = adcValue / 1024.0 * 3.3
        print("ADC Value:", adcValue, "Voltage:", voltage, "V")
        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.

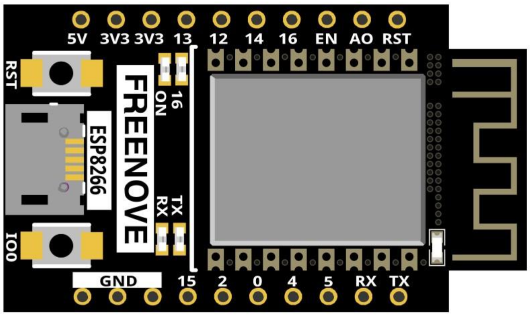
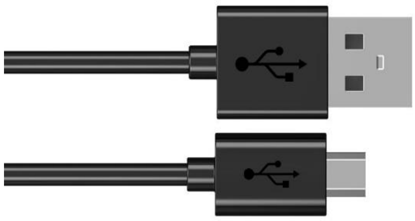
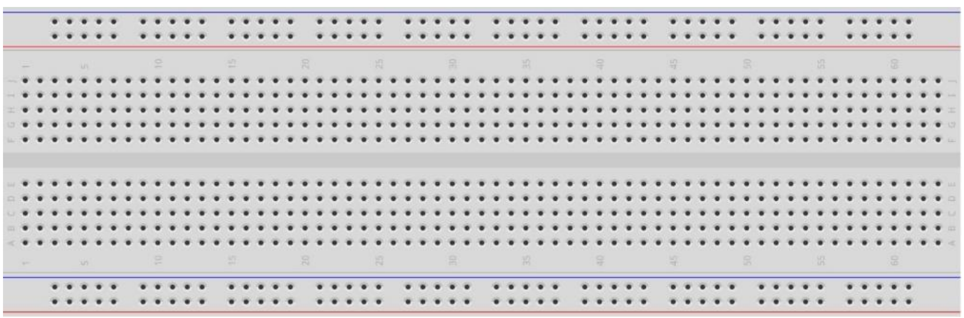
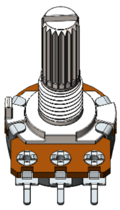
Chapter 10 Potentiometer & LED

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 10.1 Soft Light

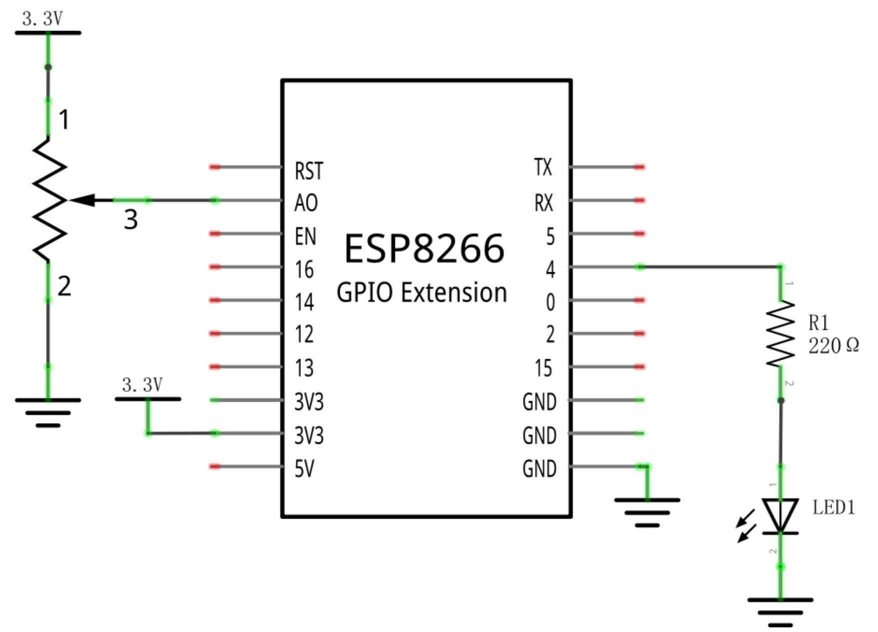
In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle of the PWM used to control the brightness of a LED. Then you can change the brightness of a LED by adjusting the potentiometer.

Component List

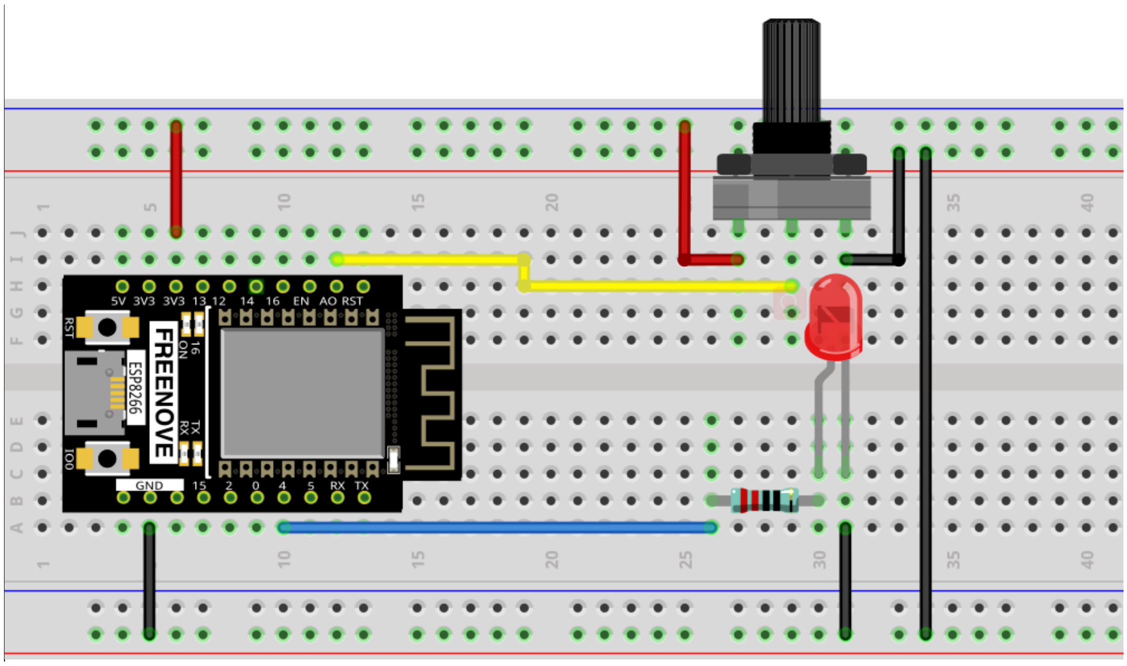
<table>
<tr><td colspan="2">ESP8266 x1

</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

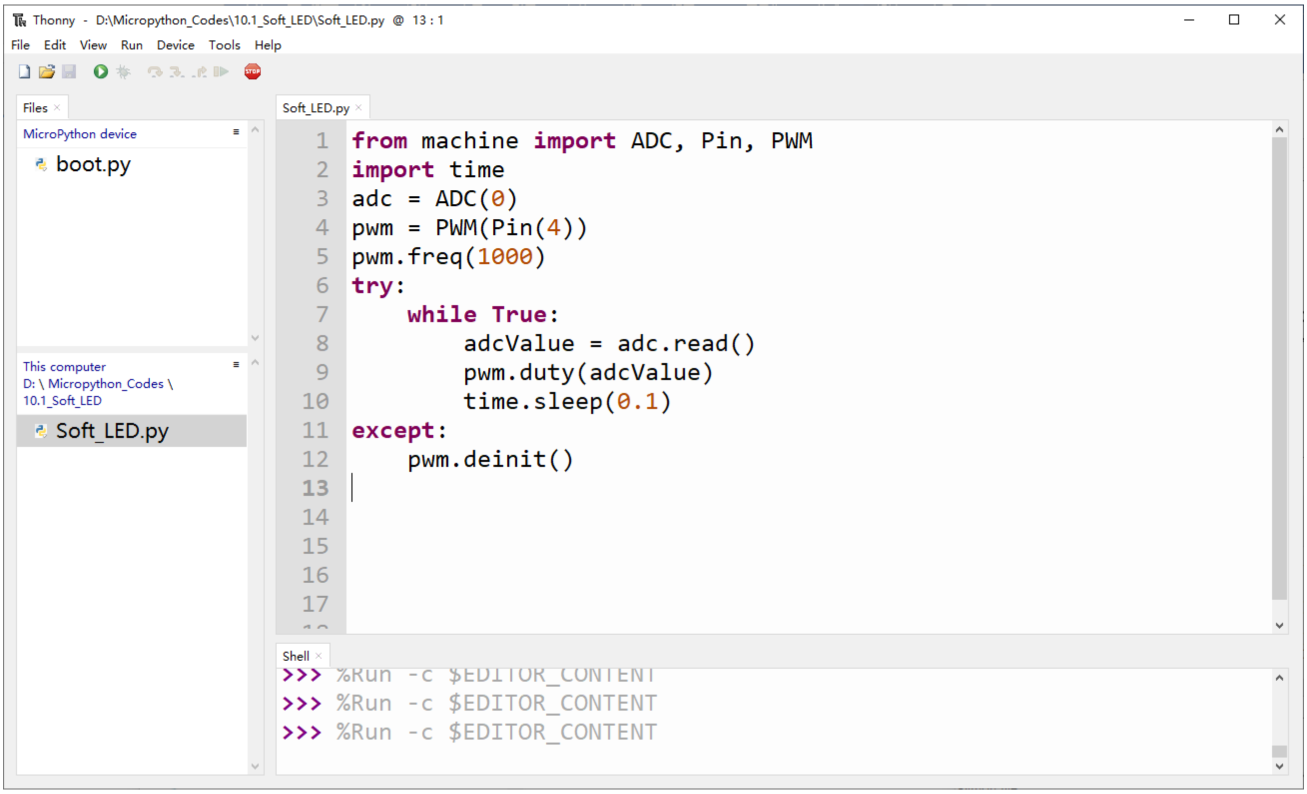


Code

Move the program folder "**Freenove_Ultimate_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:

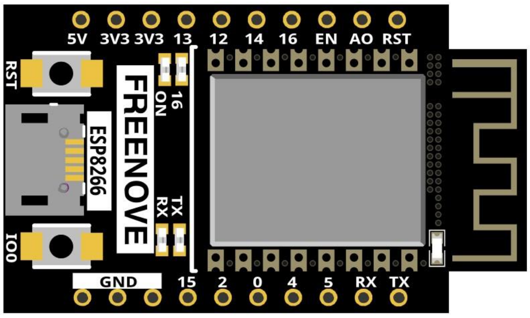
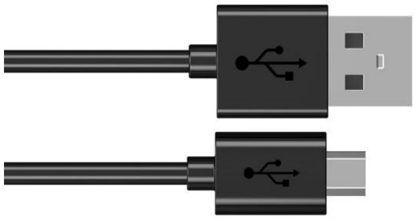
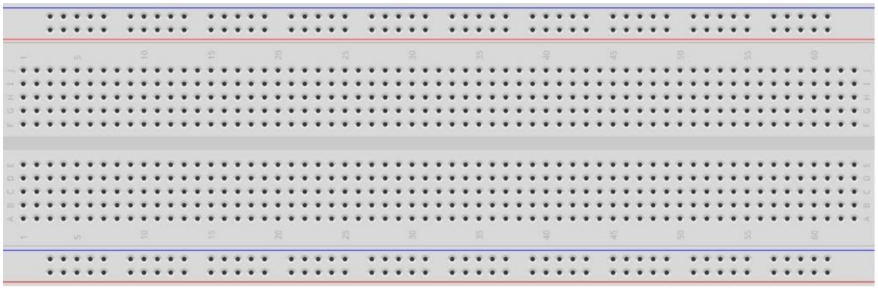
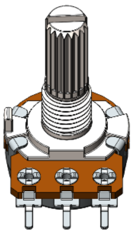
```
from machine import ADC, Pin, PWM
import time
adc = ADC(0)
pwm = PWM(Pin(4))
pwm.freq(1000)
try:
    while True:
        adcValue = adc.read()
        pwm.duty(adcValue)
        time.sleep(0.1)
except:
    pwm.deinit()
```

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

Project 10.2 Color Light

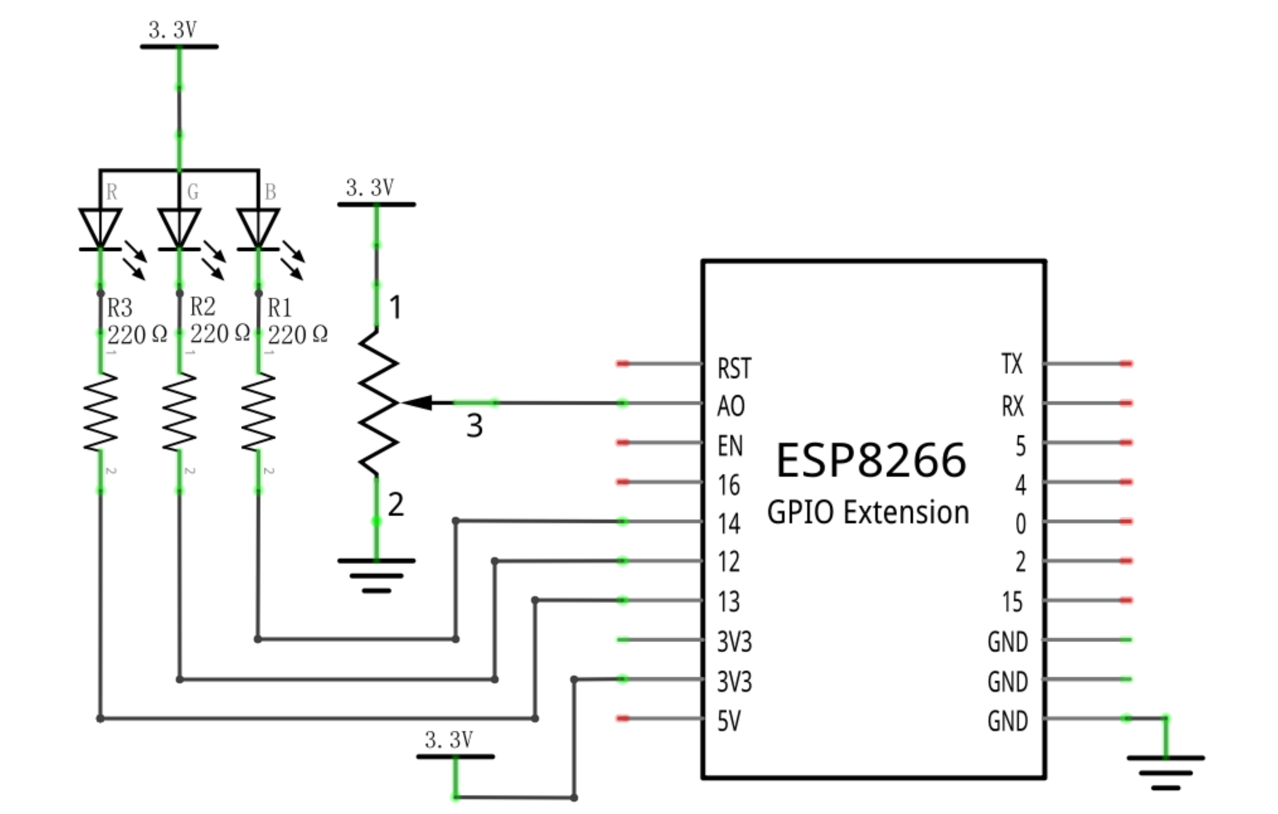
In this project, a potentiometer is used to control the RGB LED. The RGB LED is bright red when the potentiometer is near the midpoint, green when the potentiometer rotates to the "left" and blue when the potentiometer rotates to the "right".

Component List

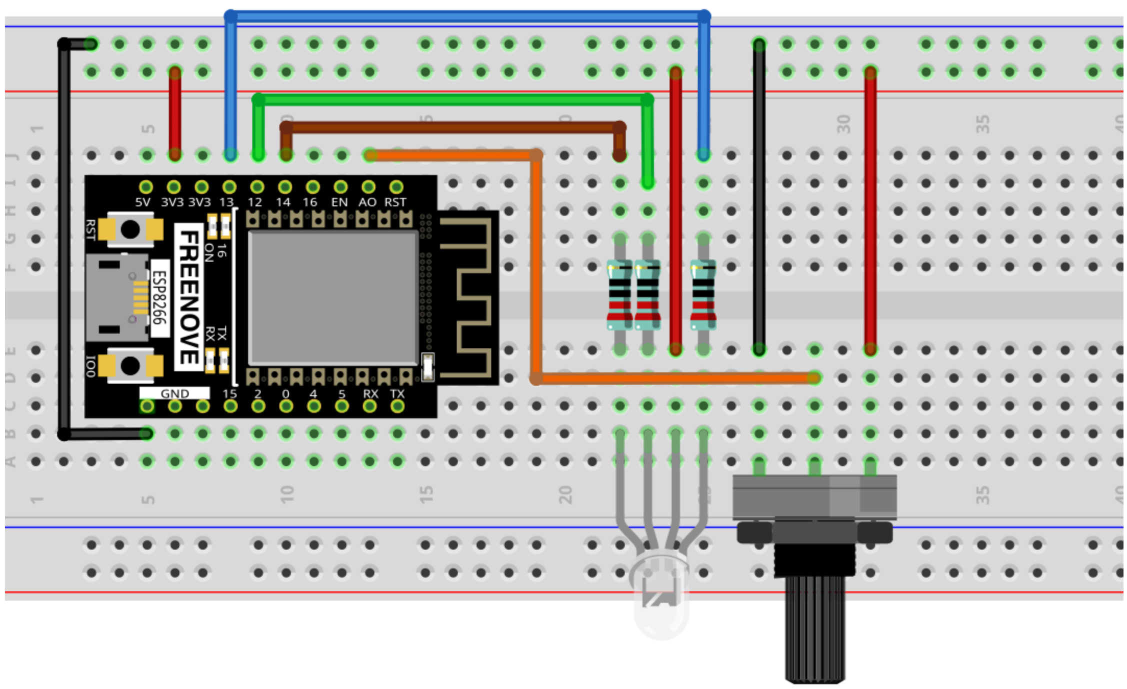
<table>
<tr><td colspan="2">ESP8266 x1

</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 x3
</td><td>RGBLED x1
</td><td>Jumper wire M/M x9
</td></tr>
</table>

Circuit

Schematic diagram



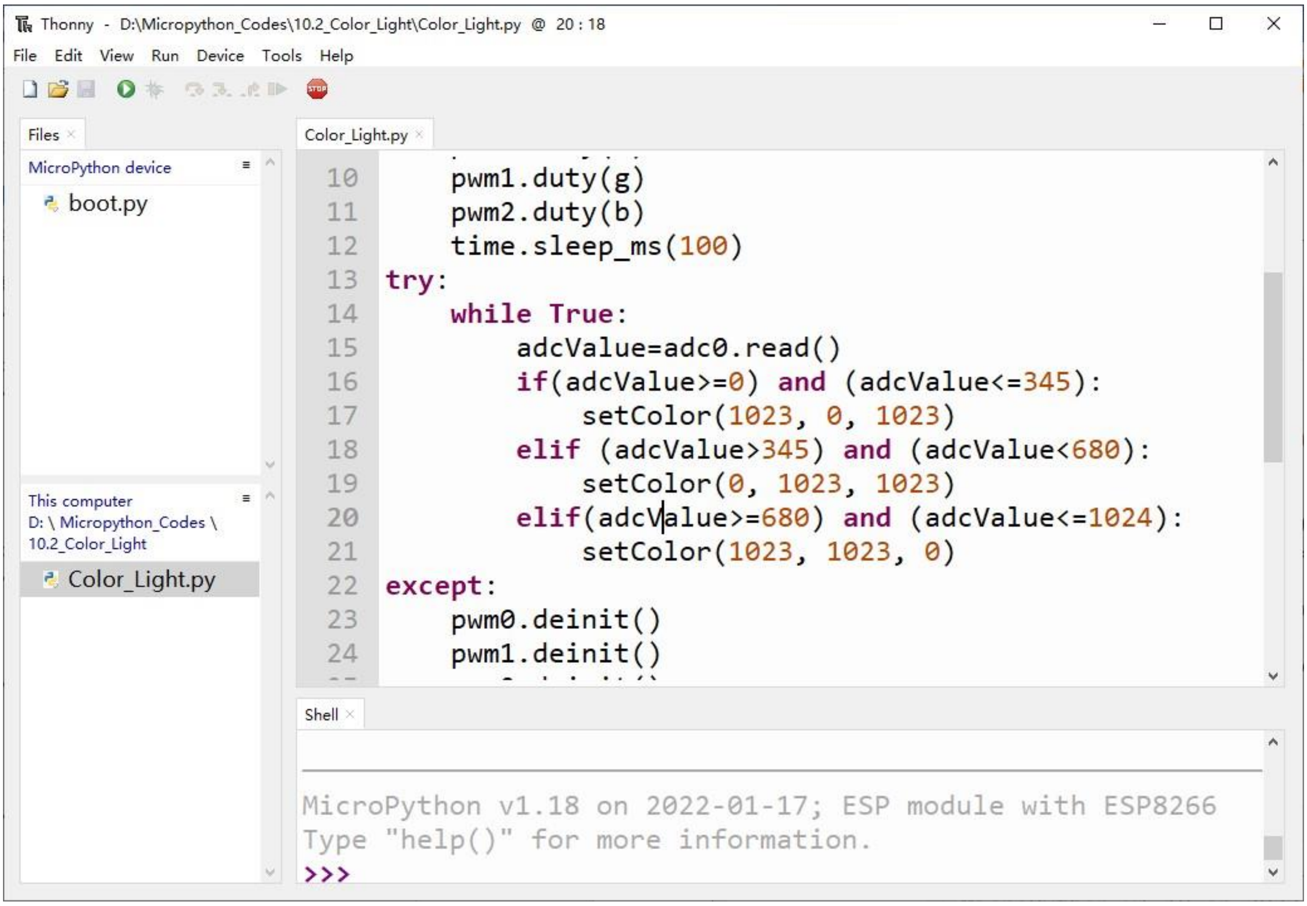
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



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:
```

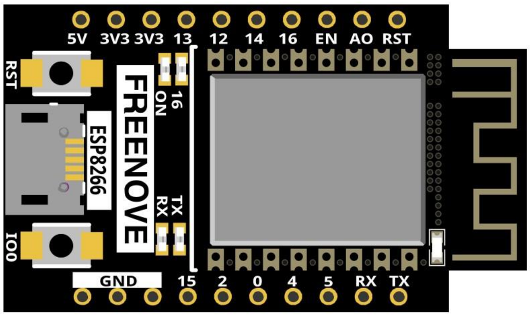
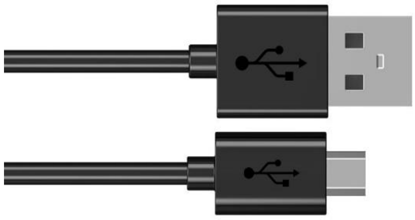
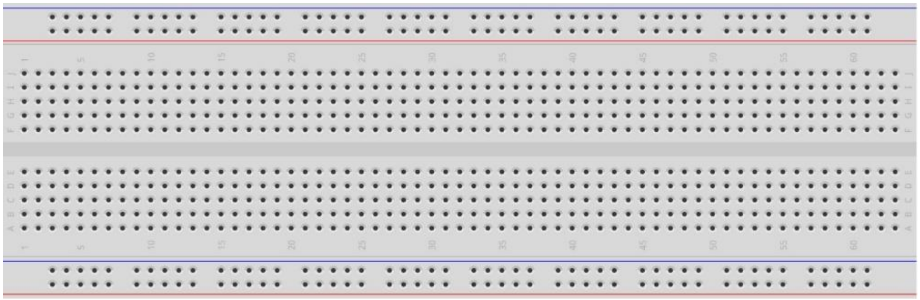
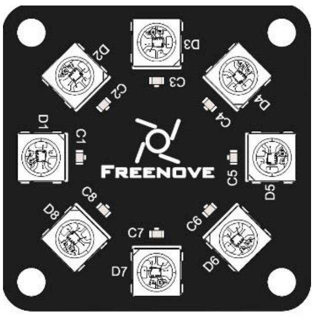
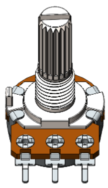
```
    while True:
        adcValue=adc0.read()
        if(adcValue>=0) and (adcValue<=345):
            setColor(1023, 0, 1023)
        elif (adcValue>345) and (adcValue<680):
            setColor(0, 1023, 1023)
        elif(adcValue>=680) and (adcValue<=1024):
            setColor(1023, 1023, 0)
except:
    pwm0.deinit()
    pwm1.deinit()
    pwm2.deinit()
```

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

Project 10.3 Soft Rainbow Light

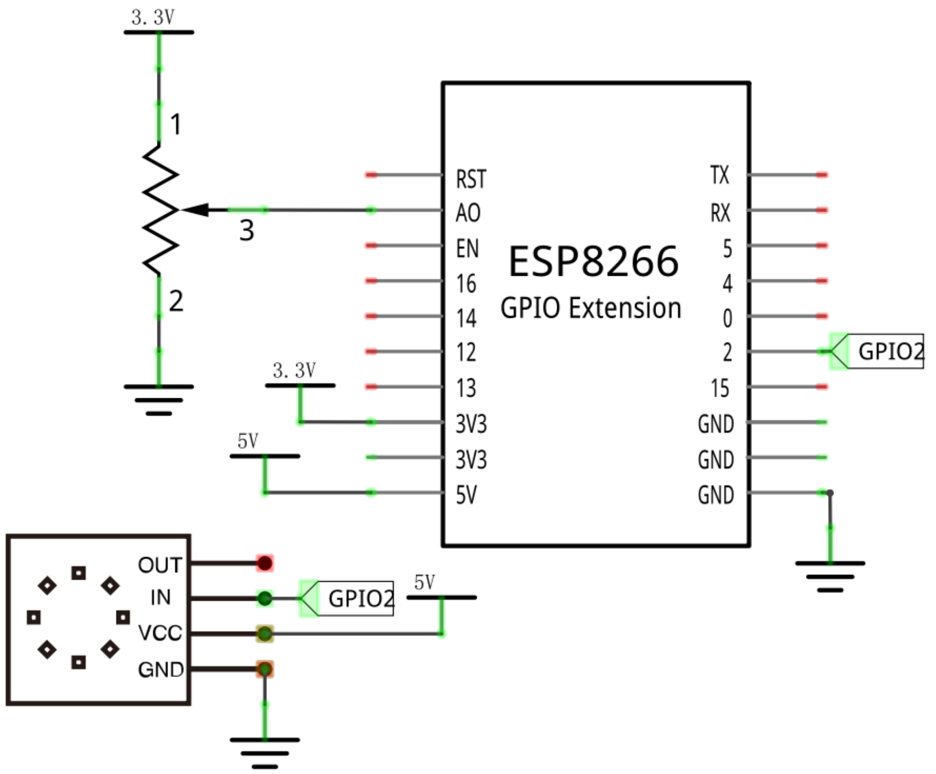
In this project, we use a potentiometer to control Freenove 8 RGBLED Module.

Component List

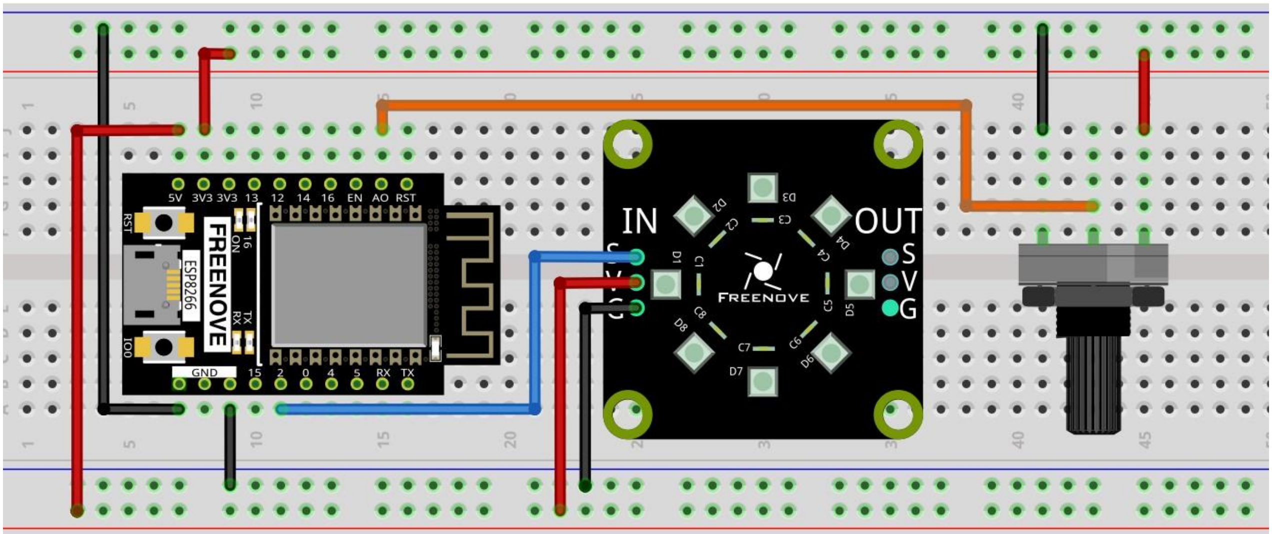
ESP8266 x1 	USB cable 	
Breadboard x1 		
Freenove 8 RGB LED Module x1 	Rotary potentiometer x1 	Jumper wire F/M x3 Jumper wire M/M x7 

Circuit

Schematic diagram



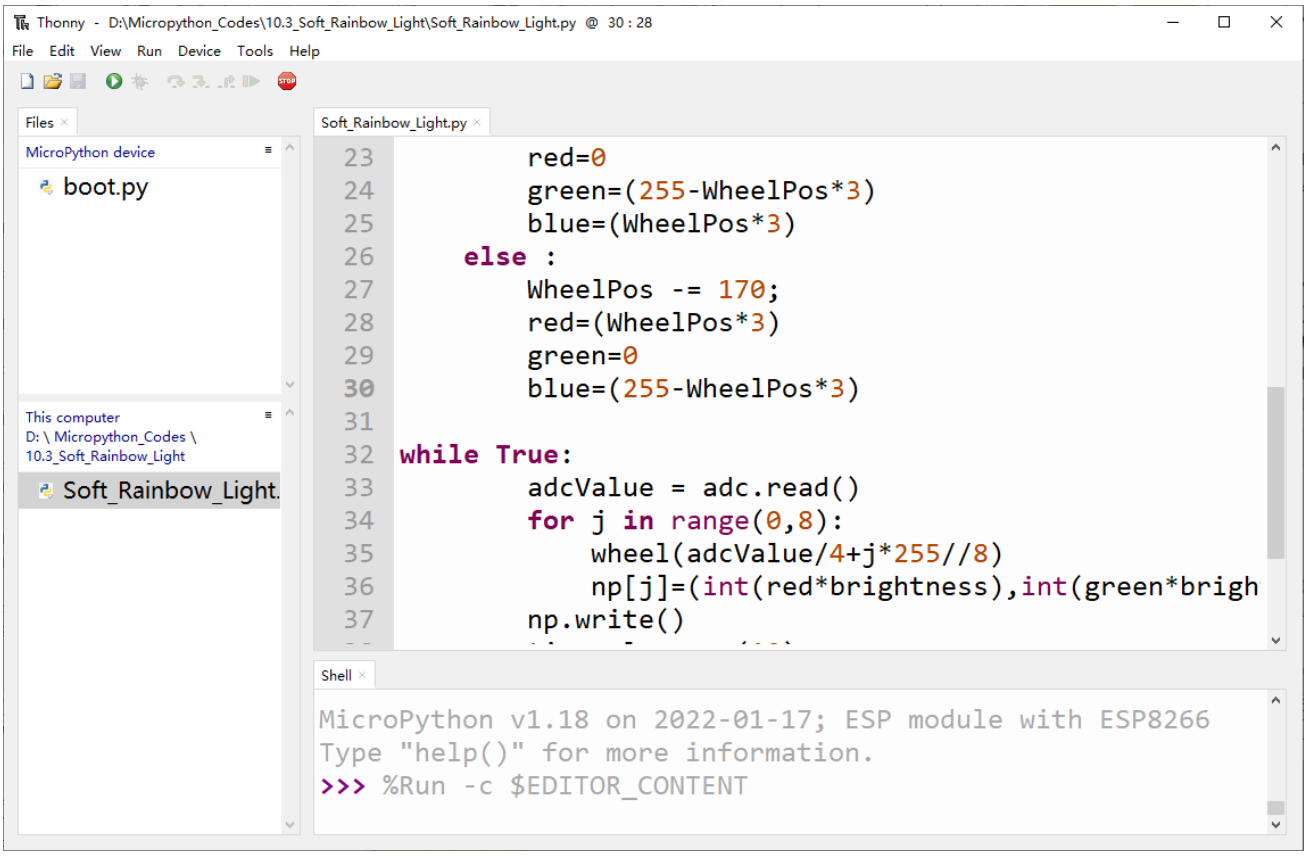
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “10.3_Soft_Rainbow_Light” and double click “Soft_Rainbow_Light.py”.

10.3_Soft_Rainbow_Light



Click “Run current script”. Rotate the handle of potentiometer and the color of the lights will change.

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

The following is the program code:

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

np = neopixel.NeoPixel(Pin(2, Pin.OUT), 8)

brightness=0.1          #brightbess
red=0                   #red
green=0                 #green
blue=0                  #blue

adc0=ADC(0)

def wheel(pos):
    global red,green,blue
    WheelPos=pos%255
    if WheelPos<85:
        red=(255-WheelPos*3)
        green=(WheelPos*3)
        blue=0
    elif WheelPos>=85 and WheelPos<170:
        WheelPos -= 85;
        red=0
        green=(255-WheelPos*3)
        blue=(WheelPos*3)
    else :
        WheelPos -= 170;
        red=(WheelPos*3)
        green=0
        blue=(255-WheelPos*3)

while True:
        adcValue = adc.read()
        for j in range(0,8):
            wheel(adcValue/4+j*255//8)
            np[j]=(int(red*brightness),int(green*brightness),int(blue*brightness))
        np.write()
        time.sleep_ms(10)
```

The logic of the code is basically the same as the previous project Rainbow Light. The difference is that in this code, the starting point of the color is controlled by the potentiometer.

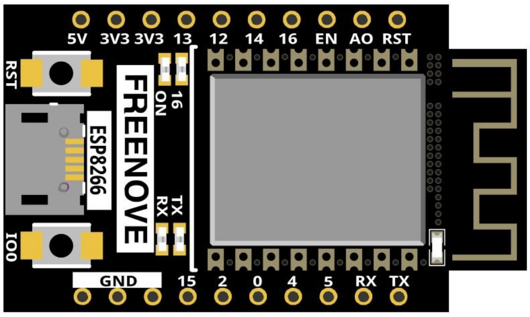
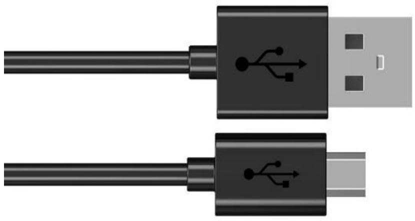
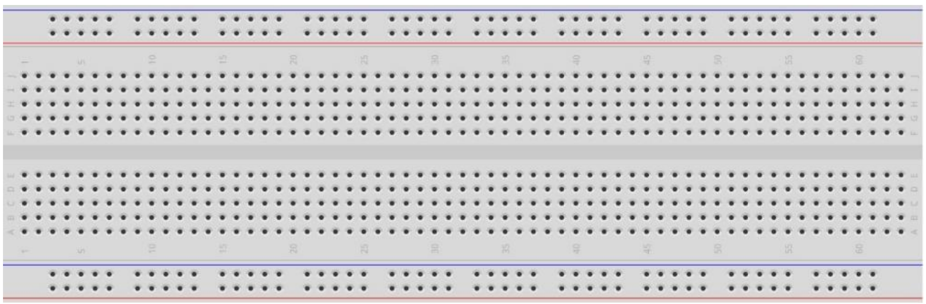
Chapter 11 Photoresistor & LED

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

Project 11.1 NightLamp

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

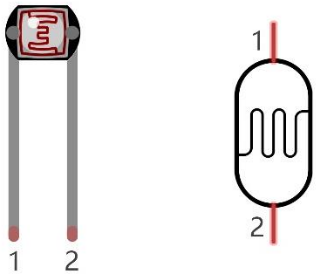
Component List

<table>
<tr><td colspan="3">ESP8266 x1

</td><td colspan="2">USB cable
</td></tr>
<tr><td colspan="5">Breadboard x1
</td></tr>
<tr><td rowspan="2">Photoresistor x1
</td><td colspan="2">Resistor</td><td rowspan="2">LED x1
</td><td rowspan="2">Jumper wire M/M x7
</td></tr>
<tr><td>220Ω x1
</td><td>10KΩ x1
</td></tr>
</table>

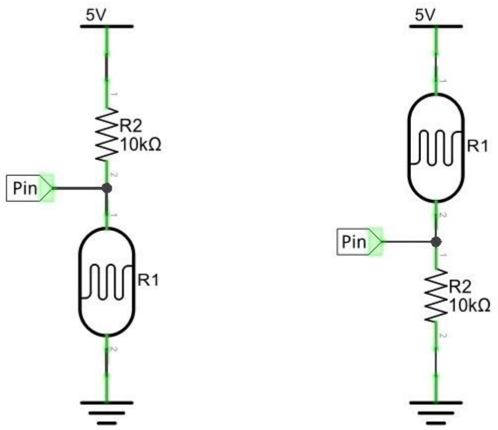
Component knowledge

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:

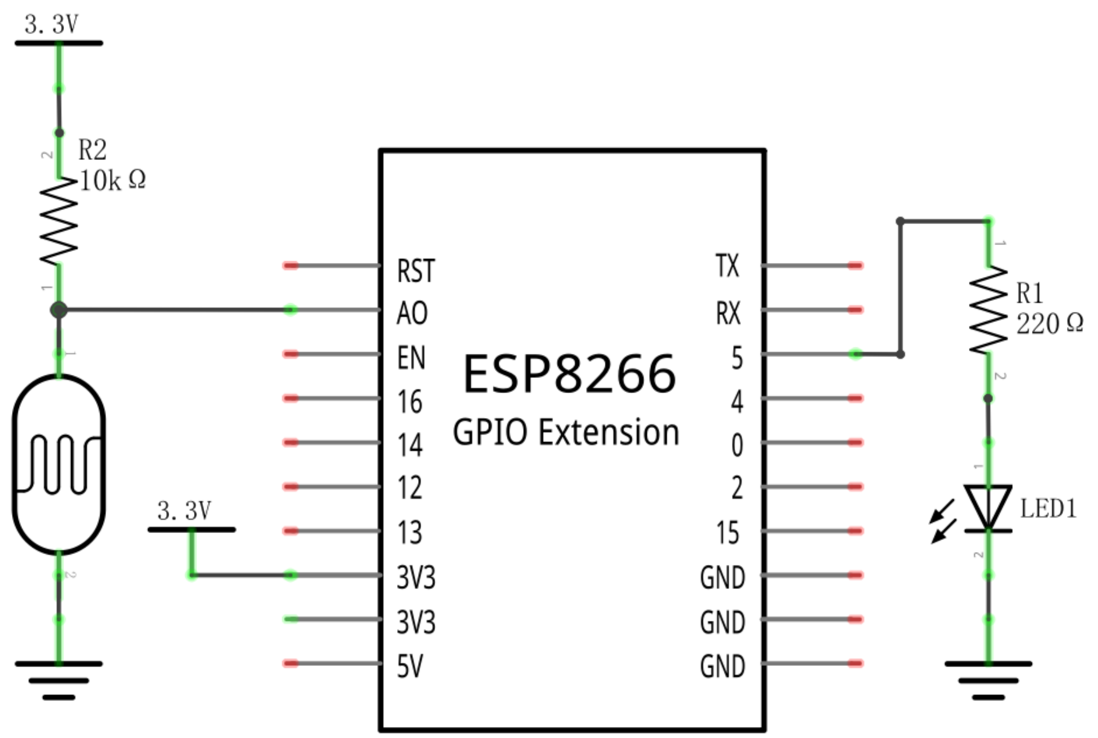


In the above circuit, when a Photoresistor's resistance vale changes due to a change in light intensity, the voltage between the Photoresistor and Resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

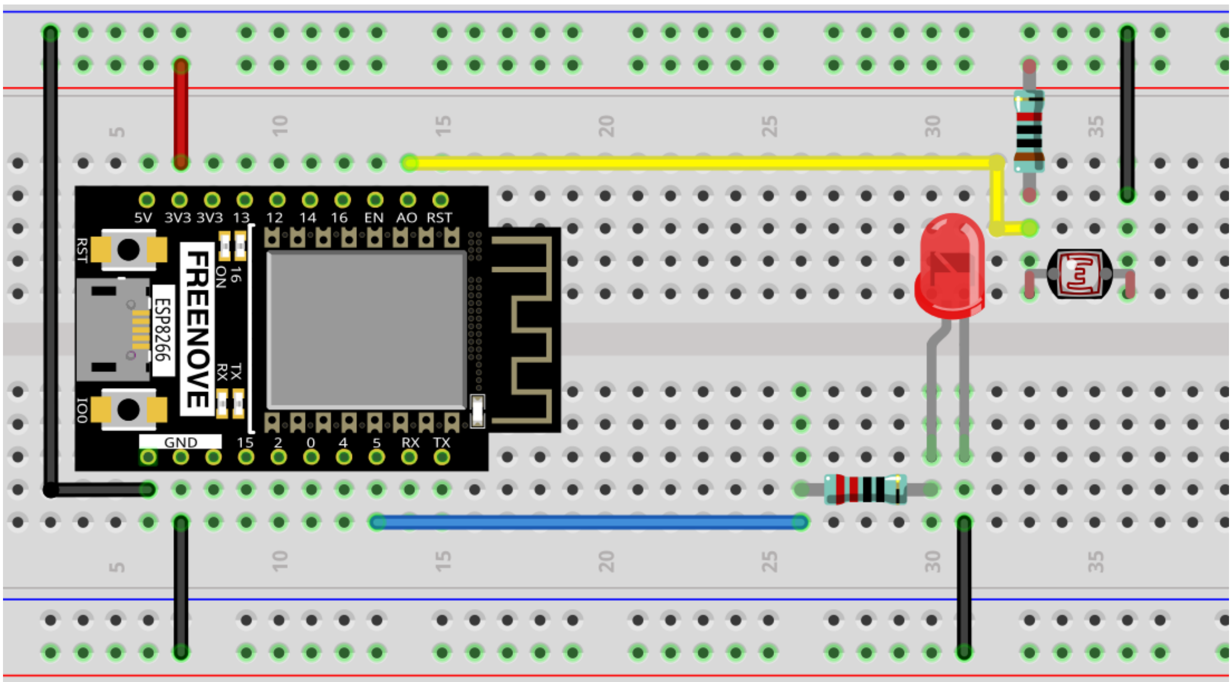
Circuit

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

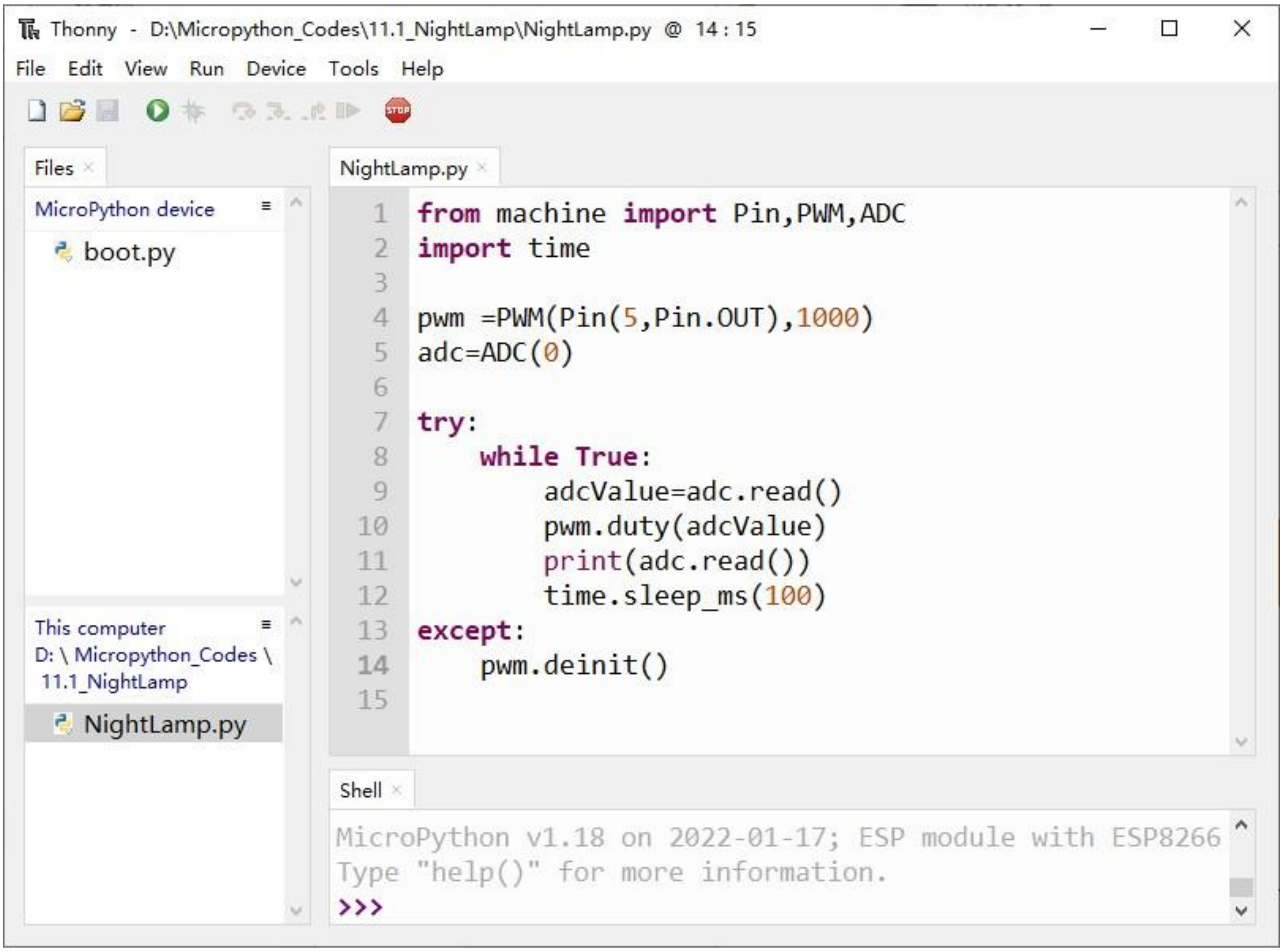


Code

Move the program folder “**Freenove_Ultimate_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:

```python
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())
```

```
        time.sleep_ms(100)
except:
    pwm.deinit()
```

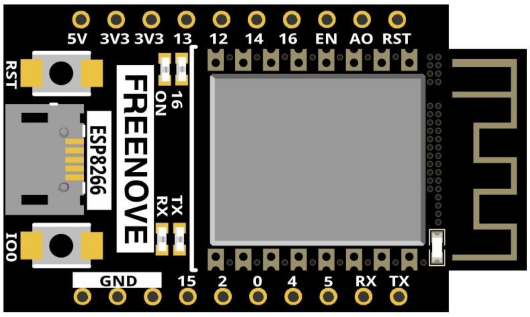
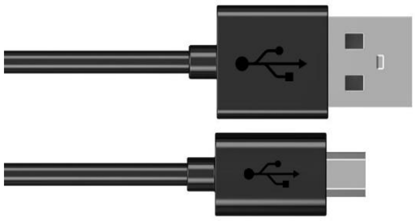
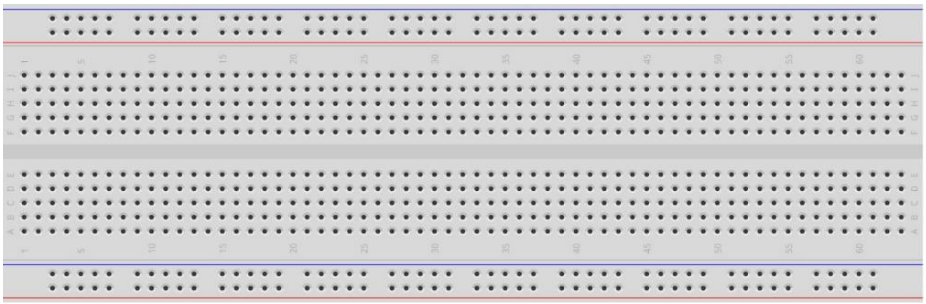
Chapter 12 Thermistor

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

Project 12.1 Thermometer

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

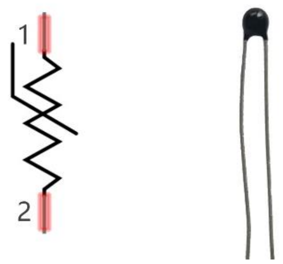
Component List

ESP8266 x1 		USB cable 
Breadboard x1 		
Thermistor x1 	Resistor 1kΩ x1 	Jumper wire M/M x4 

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:

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

Where:

Rt is the thermistor resistance under T2 temperature;

R is the nominal resistance of thermistor under T1 temperature;

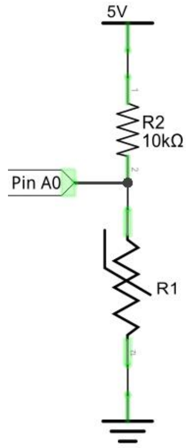
EXP[n] is nth power of e;

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.

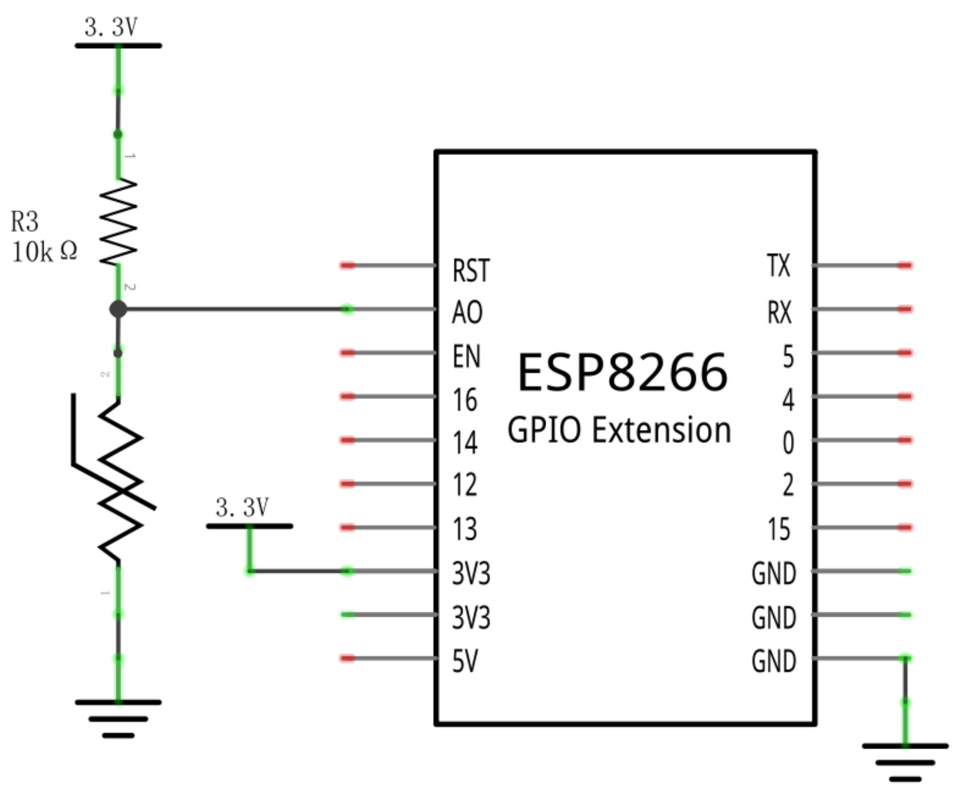
Therefore, the temperature formula can be derived as:

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

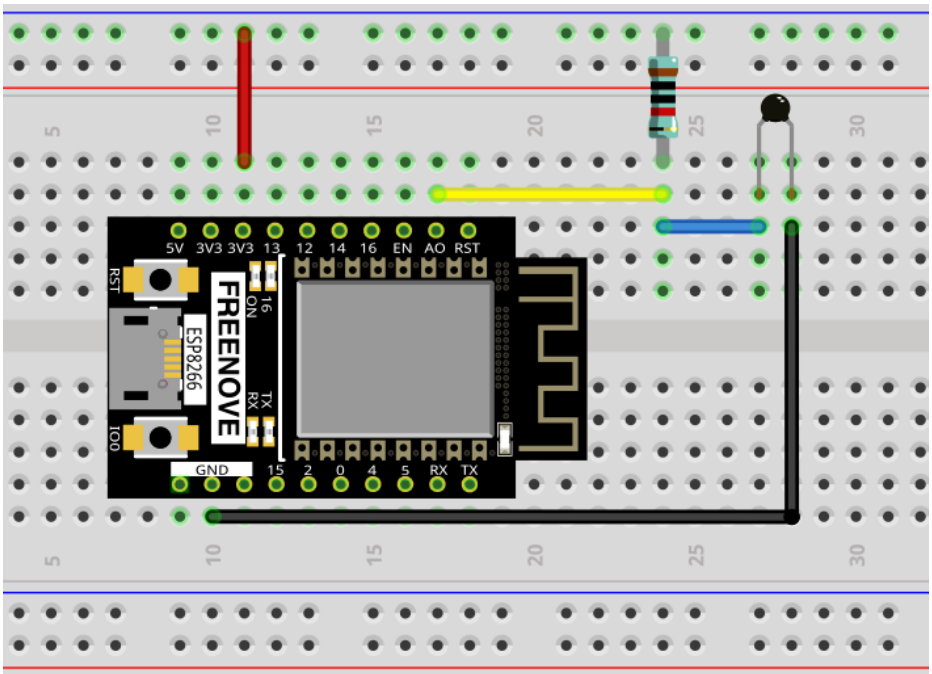
Circuit

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

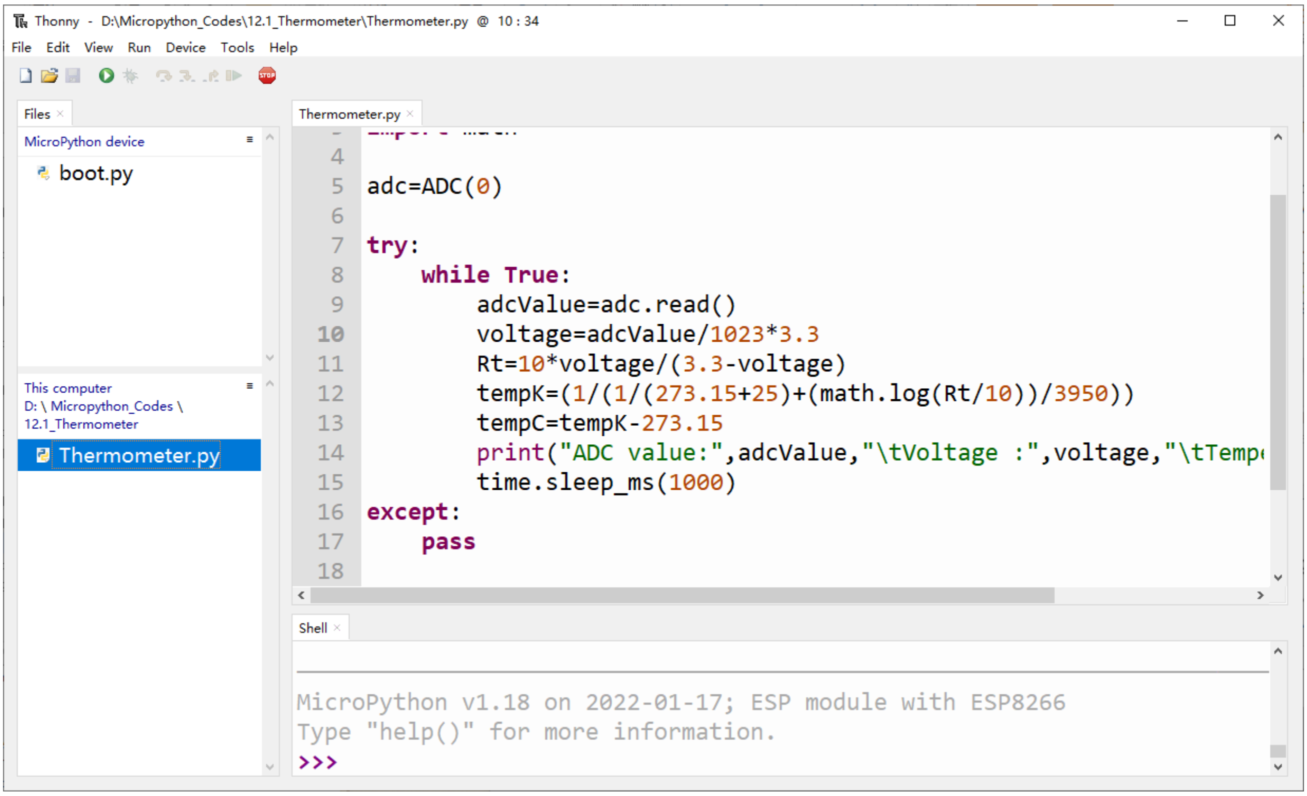


Code

Move the program folder "**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

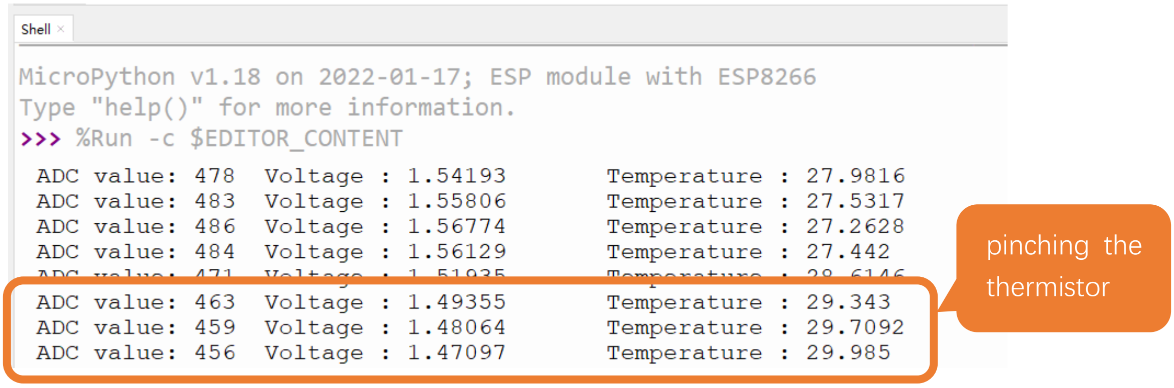
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "12.1_Thermometer" and double click "Thermometer.py".

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.

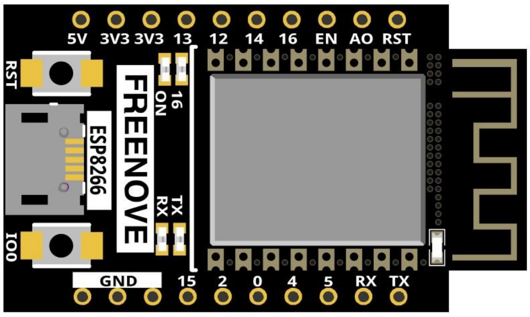
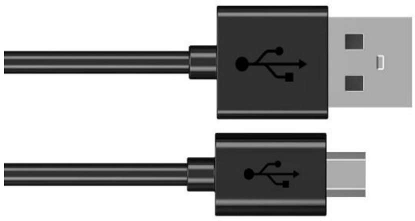
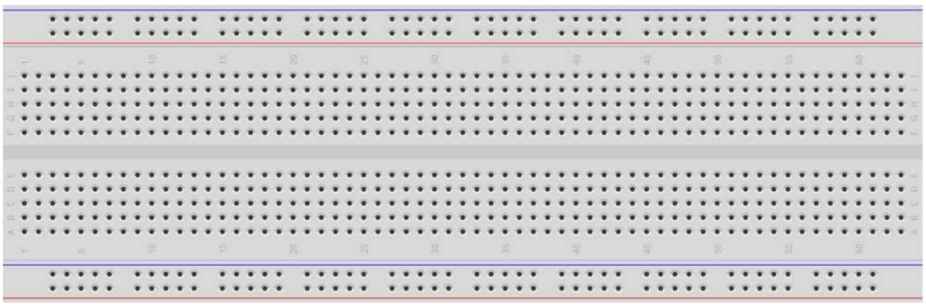
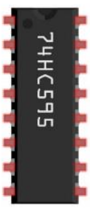
Chapter 13 74HC595 & LED Bar Graph

We have used LED Bar Graph to make a flowing water light, in which 10 GPIO ports of ESP8266 is occupied. More GPIO ports mean that more peripherals can be connected to ESP8266, so GPIO resource is very precious. Can we make flowing water light with less GPIO? In this chapter, we will learn a component, 74HC595, which can achieve the target.

Project 13.1 Flowing Water Light

Now let’s learn how to use the 74HC595 IC Chip to make a flowing water light using less GPIO.

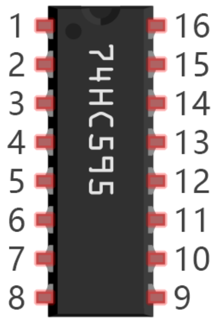
Component List

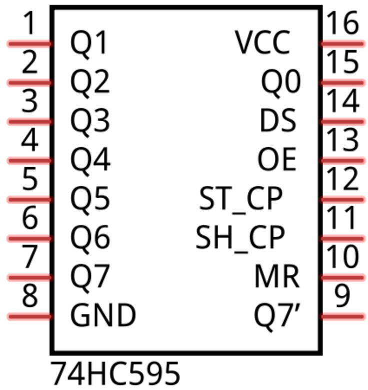
<table>
<tr><td colspan="2">ESP8266 x1
</td><td colspan="2">USB cable
</td></tr>
<tr><td colspan="4">Breadboard x1
</td></tr>
<tr><td>74HC595 x1
</td><td>LED Bar Graph x1
</td><td>Resistor 220Ω x8
</td><td>Jumper wire M/M x17
</td></tr>
</table>

Related knowledge

74HC595

A 74HC595 chip is used to convert serial data into parallel data. A 74HC595 chip can convert the serial data of one byte into 8 bits, and send its corresponding level to each of the 8 ports correspondingly. With this characteristic, the 74HC595 chip can be used to expand the IO ports of an ESP8266. At least 3 ports are required to control the 8 ports of the 74HC595 chip.





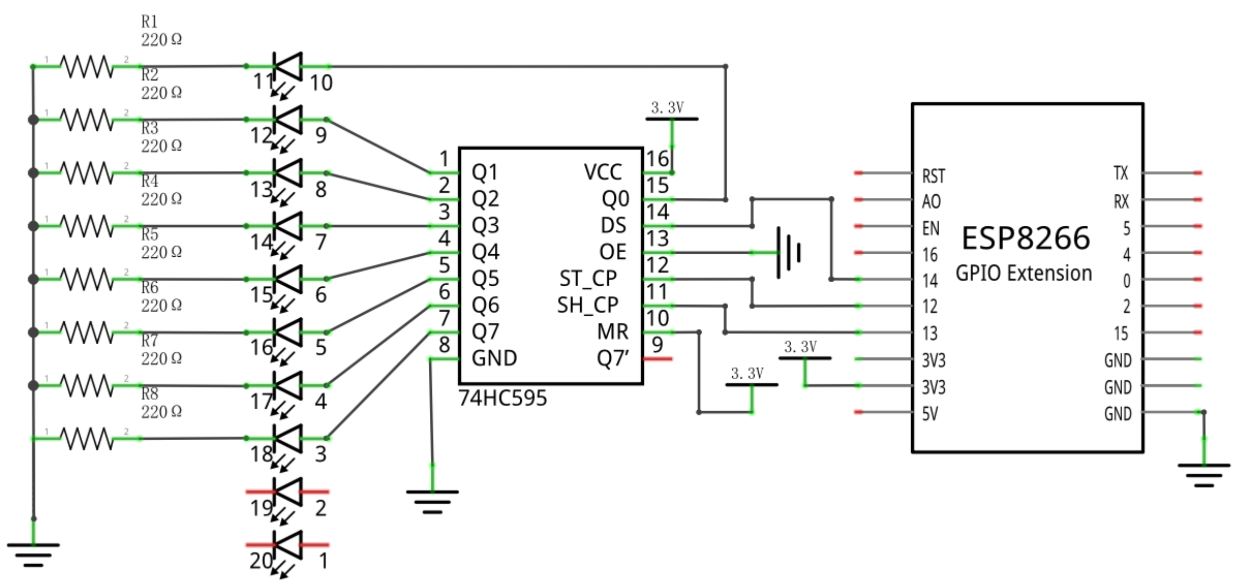
The ports of the 74HC595 chip are described as follows:

Pin name	GPIO number	Description
Q0-Q7	15, 1-7	Parallel data output
VCC	16	The positive electrode of power supply, the voltage is 2~6V
GND	8	The negative electrode of power supply
DS	14	Serial data Input
OE	13	Enable output, When this pin is in high level, Q0-Q7 is in high resistance state When this pin is in low level, Q0-Q7 is in output mode
ST_CP	12	Parallel Update Output: when its electrical level is rising, it will update the parallel data output.
SH_CP	11	Serial shift clock: when its electrical level is rising, serial data input register will do a shift.
MR	10	Remove shift register: When this pin is in low level, the content in shift register will be cleared.
Q7'	9	Serial data output: it can be connected to more 74HC595 in series.

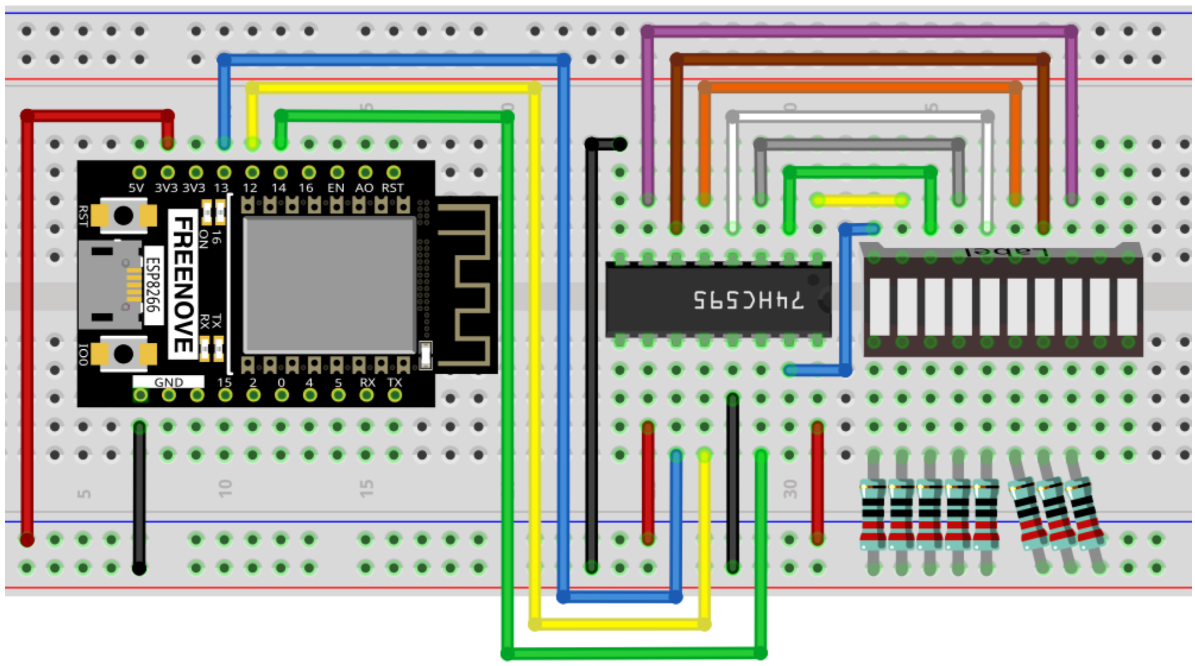
For more detail, please refer to the datasheet on the 74HC595 chip.

Circuit

Schematic diagram



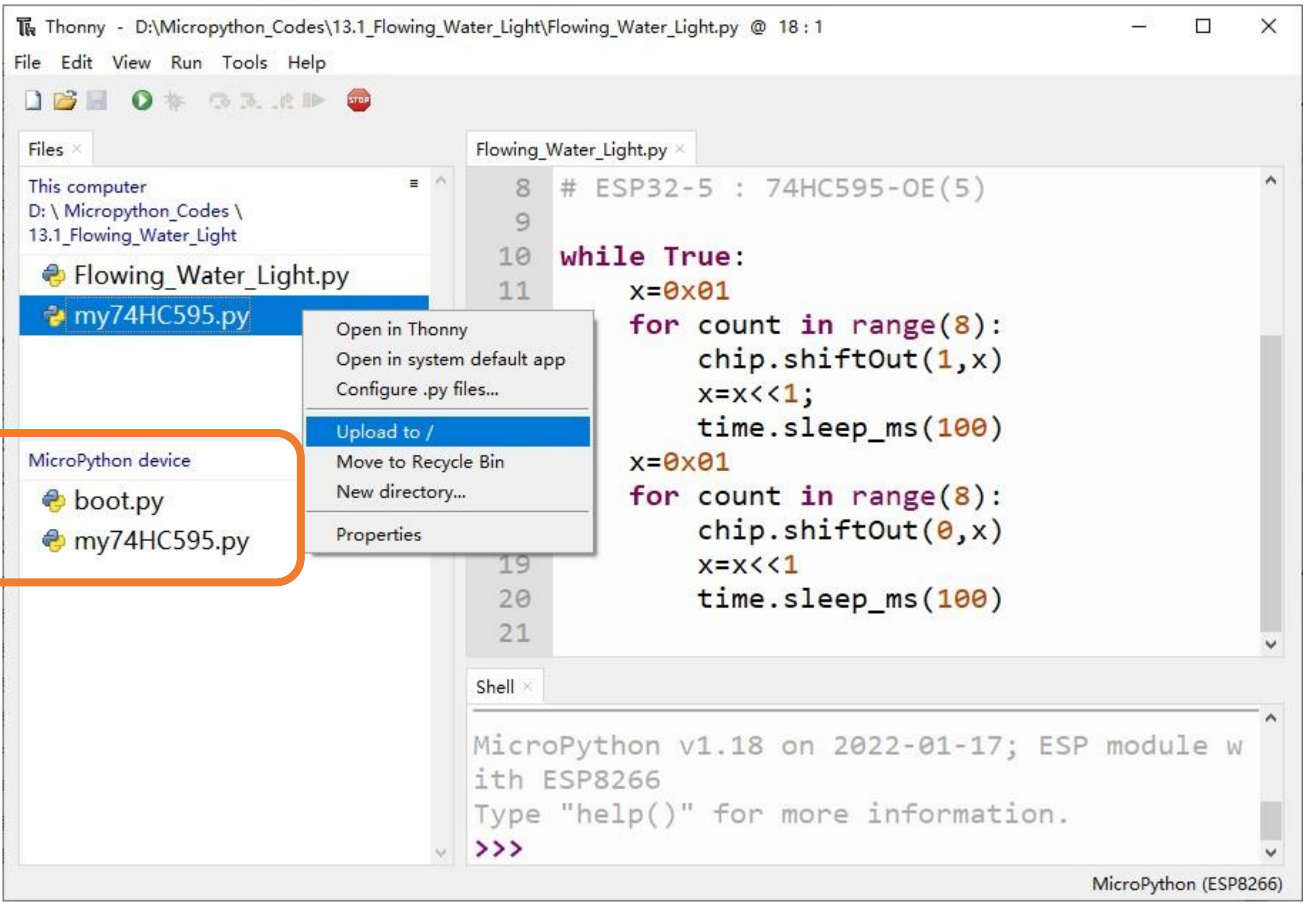
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

In this project, we will make a flowing water light with a 74HC595 chip to learn about its functions.
Move the program folder "**Freenove_Ultimate_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" → "13.1_Flowing_Water_Light".
Select "my74HC595.py", right click your mouse to select "Upload to /", wait for "my74HC595.py"to be uploaded to ESP8266 and then double click "Flowing_Water_Light.py".

13.1_Flowing_Water_Light



Click"Run current script" and you will see that Bar Graph LED starts with the flowing water pattern flashing from left to right and then back from right to left. If it displays nothing, maybe the LED Bar is connected upside down, please unplug it and then re-plug it reversely.

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

The following is the program code:

```
import time
from my74HC595 import Chip74HC595

chip = Chip74HC595(14,12,13)
# ESP8266-14: 74HC595-DS(14)
# ESP8266-12: 74HC595-STCP(12)
# ESP8266-13: 74HC595-SHCP(11)

while True:
    x=0x01
    for count in range(8):
        chip.shiftOut(1,x)   #High bit is sent first
        x=x<<1
        time.sleep_ms(100)
    x=0x01
    for count in range(8):
        chip.shiftOut(0,x)   #Low bit is sent first
        x=x<<1
        time.sleep_ms(100)
```

Import time and my74HC595 modules.

```
import time
from my74HC595 import Chip74HC595
```

Assign pins for ESP8266 to connect to 74HC595.

```
chip = Chip74HC595(14,12,13)
```

The first for loop makes LED Bar display separately from left to right while the second for loop make it display separately from right to left.

```
    x=0x01
    for count in range(8):
        chip.shiftOut(1,x)   #High bit is sent first
        x=x<<1
        time.sleep_ms(100)
    x=0x01
    for count in range(8):
        chip.shiftOut(0,x)   #Low bit is sent first
        x=x<<1
        time.sleep_ms(100)
```

Reference

Class Chip74HC595

Before each use of the object **Chip74HC595**, make sure my74HC595.py has been uploaded to “/” of ESP8266, and then add the statement “**from my74HC595 import Chip74HC595**” to the top of the python file.

Chip74HC595():An object. By default, 74HC595’s DS pin is connected to Pin(14) of ESP8266, ST_CP pin is connected to ESP8266’s Pin(12) and OE pin is connected to ESP’s Pin(5). If you need to modify the pins, just do the following operations.

chip=Chip74HC595() or **chip=Chip74HC595(14,12,13,5)**

shiftOut(direction, data): Write data to 74HC595.

direction: 1/0. “1” presents that high-order byte will be sent first while “0” presents that low-order byte will be sent first.

data: The content that is sent, which is one-byte data.

clear(): Clear the latch data of 74HC595.

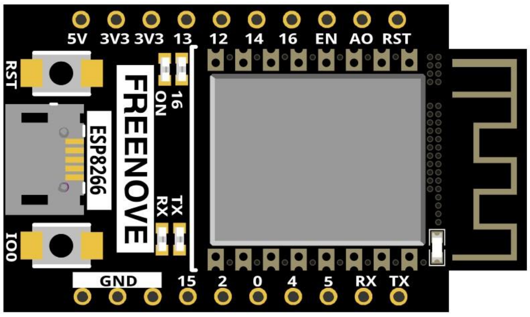
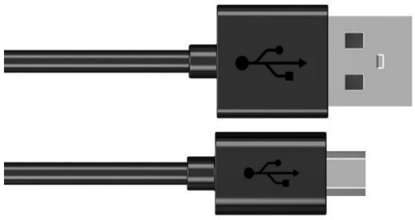
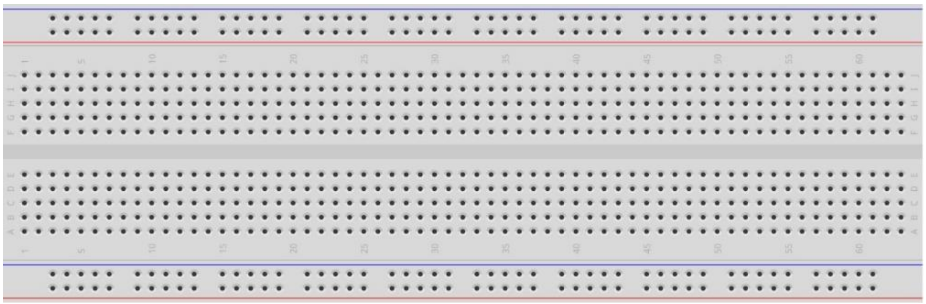
Chapter 14 74HC595 & 7-Segment Display.

In this chapter, we will introduce the 7-Segment Display.

Project 14.1 7-Segment Display.

We will use 74HC595 to control 7-segment display and make it display hexadecimal character "0-F".

Component List

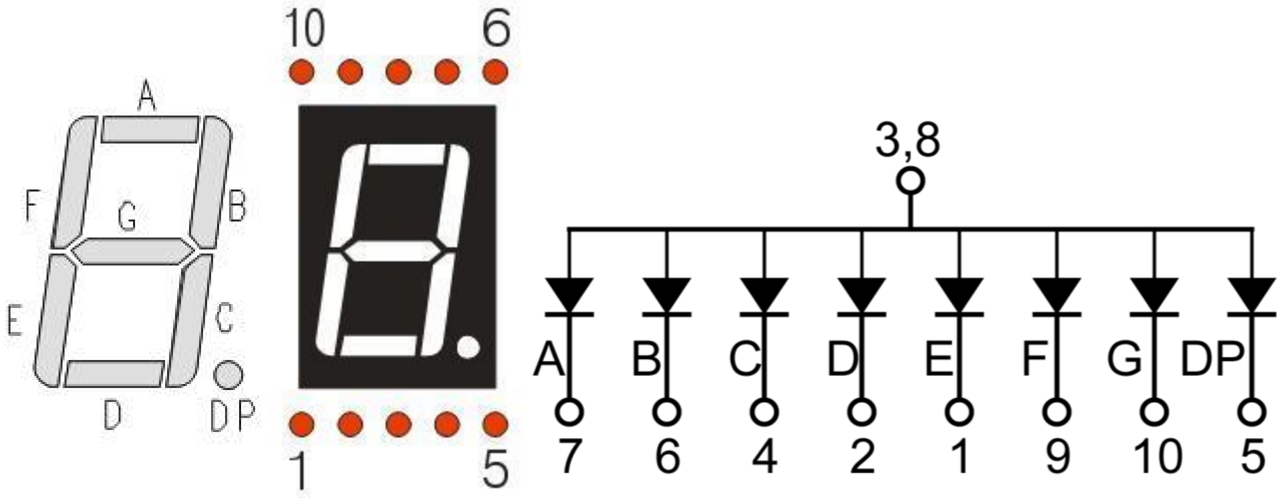
<table>
<tr><td colspan="2">ESP8266 x1

</td><td colspan="2">USB cable
</td></tr>
<tr><td colspan="4">Breadboard x1
</td></tr>
<tr><td>74HC595 x1

</td><td>7-segment display x1
</td><td>Resistor 220Ω x8
</td><td>Jumper wire M/M x19
</td></tr>
</table>

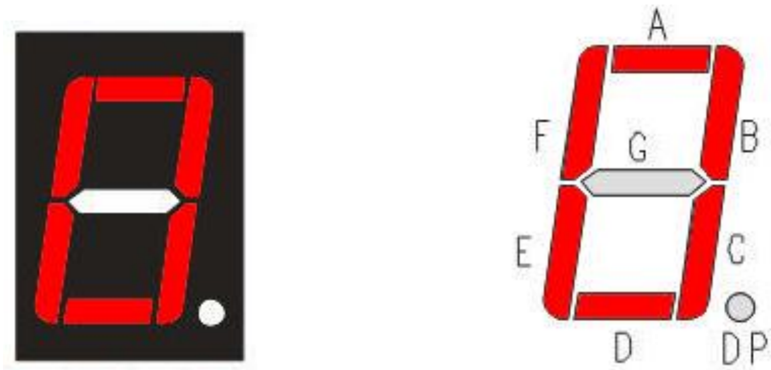
Component knowledge

7-segment display

A 7-Segment Display is a digital electronic display device. There is a figure "8" and a decimal point represented, which consists of 8 LEDs. The LEDs have a Common Anode and individual Cathodes. Its internal structure and pin designation diagram is shown below:



As we can see in the above circuit diagram, we can control the state of each LED separately. Also, by combining LEDs with different states of ON and OFF, we can display different characters (Numbers and Letters). For example, to display a “0”: we need to turn ON LED segments A, B, C, D, E and F, and turn OFF LED segments G and DP.



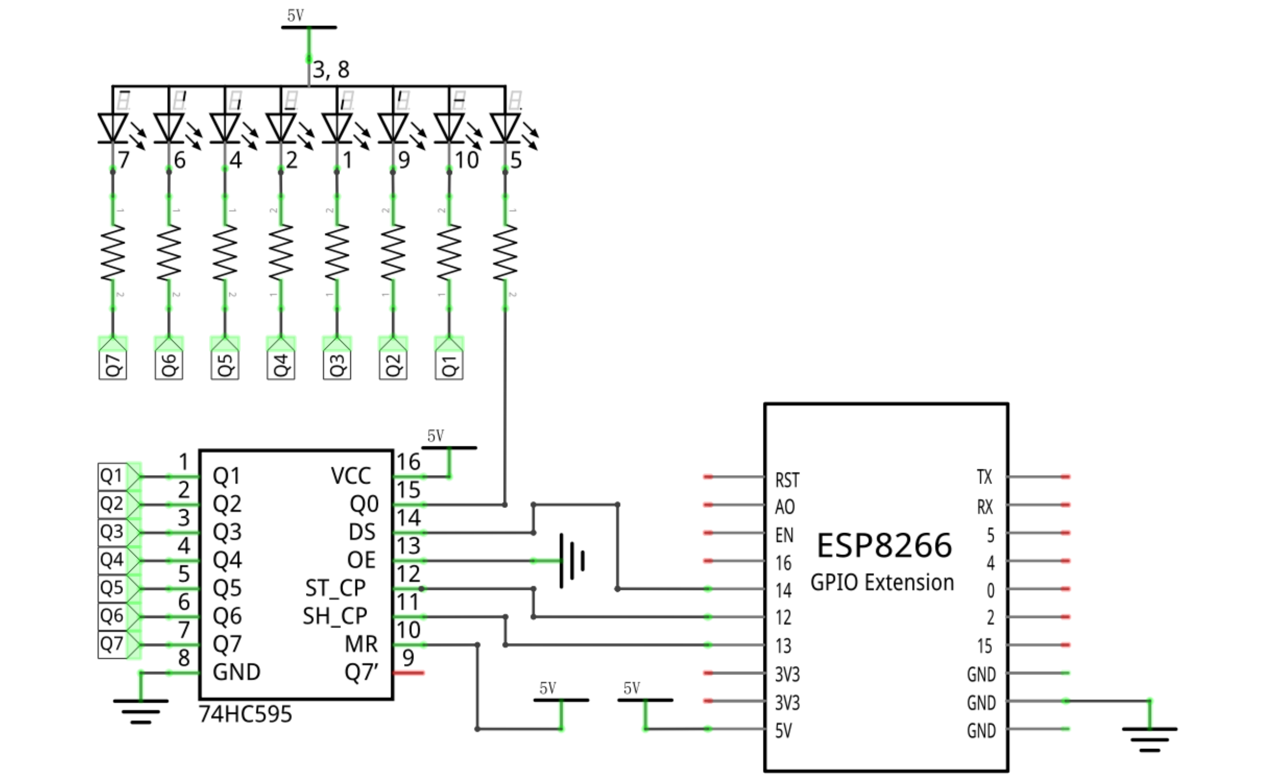
In this project, we will use a 7-Segment Display with a Common Anode. Therefore, when there is an input low level to an LED segment the LED will turn ON. Defining segment “A” as the lowest level and segment “DP” as the highest level, from high to low would look like this: “DP”, “G”, “F”, “E”, “D”, “C”, “B”, “A”. Character "0" corresponds to the code: 1100 0000b=0xc0.

For detailed code values, please refer to the following table (common anode).

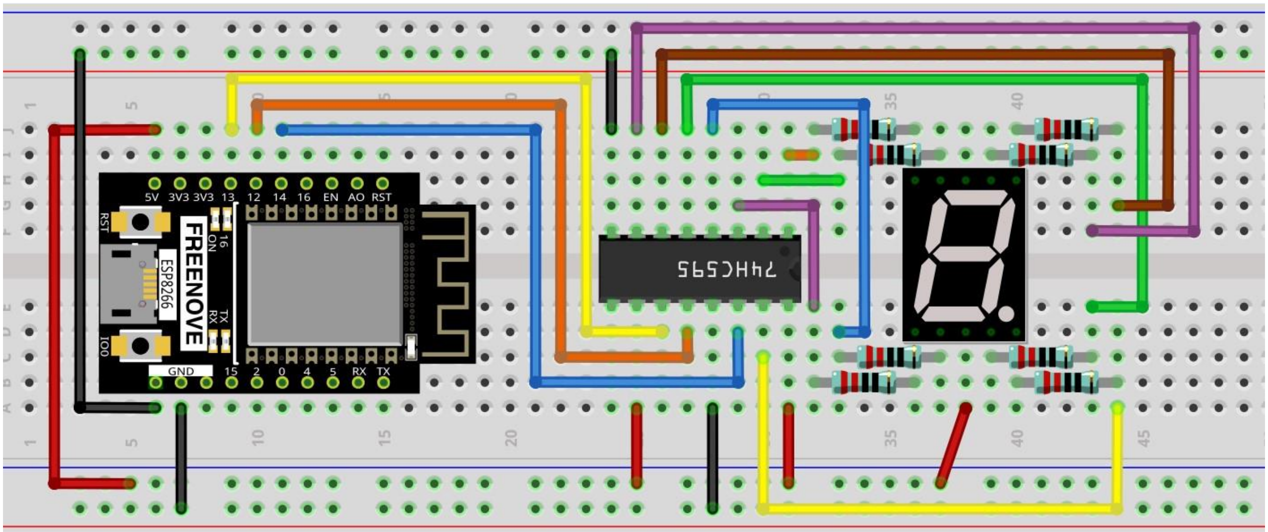
CHAR	DP	G	F	E	D	C	B	A	Hex	ASCII
0	1	1	0	0	0	0	0	0	0xc0	1100 0000
1	1	1	1	1	1	0	0	1	0xf9	1111 1001
2	1	0	1	0	0	1	0	0	0xa4	1010 0100
3	1	0	1	1	0	0	0	0	0xb0	1011 0000
4	1	0	0	1	1	0	0	1	0x99	1001 1001
5	1	0	0	1	0	0	1	0	0x92	1001 0010
6	1	0	0	0	0	0	1	0	0x82	1000 0010
7	1	1	1	1	1	0	0	0	0xf8	1111 1000
8	1	0	0	0	0	0	0	0	0x80	1000 0000
9	1	0	0	1	0	0	0	0	0x90	1001 0000
A	1	0	0	0	1	0	0	0	0x88	1000 1000
B	1	0	0	0	0	0	1	1	0x83	1000 0011
C	1	1	0	0	0	1	1	0	0xc6	1100 0110
D	1	0	1	0	0	0	0	1	0xa1	1010 0001
E	1	0	0	0	0	1	1	0	0x86	1000 0110
F	1	0	0	0	1	1	1	0	0x8e	1000 1110

Circuit

Schematic diagram



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



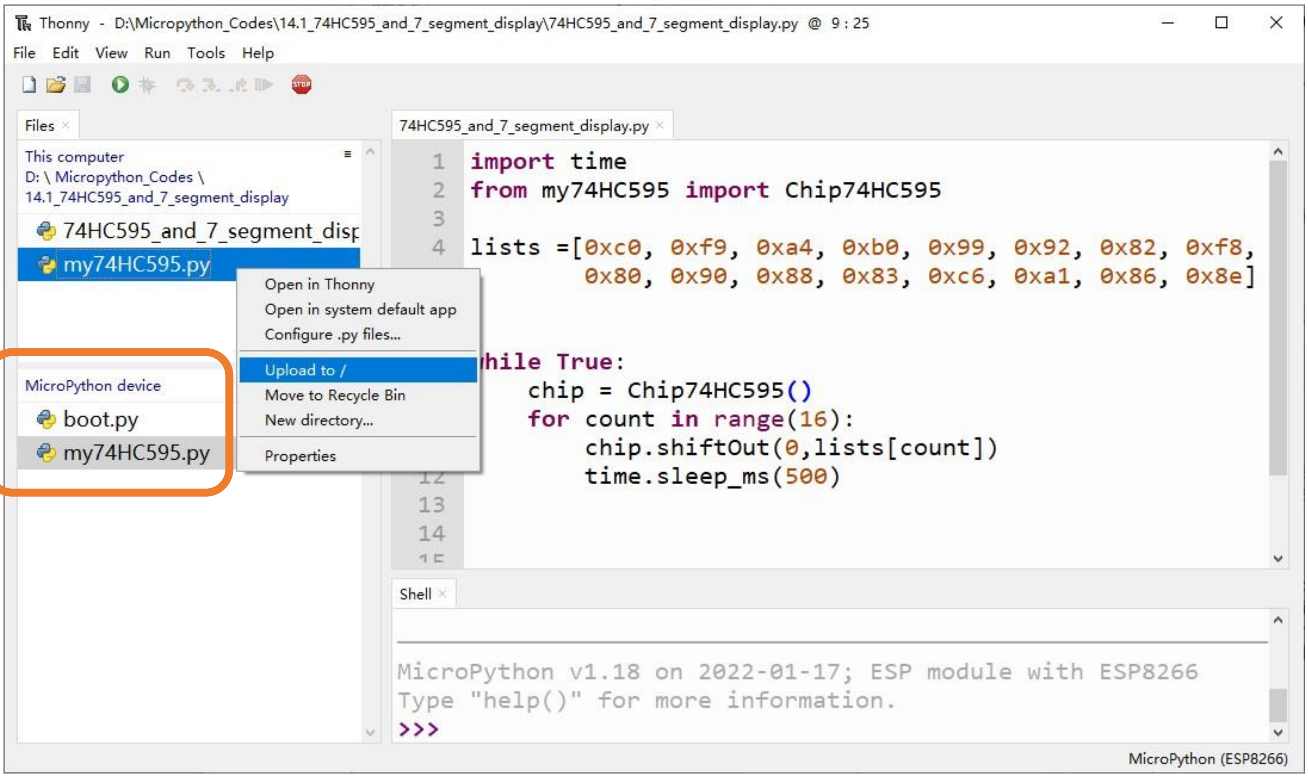
Code

In this section, the 74HC595 is used in the same way as in the previous section, but with different values transferred. We can learn how to master the digital display by sending the code value of "0" - "F".
Move the program folder "**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

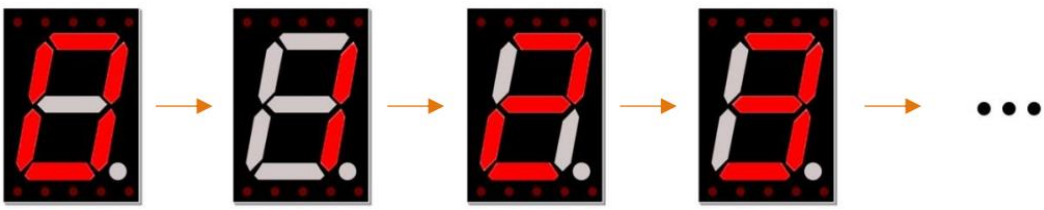
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "14.1_74HC595_and_7_segment_display".

Select "my74HC595.py", right click your mouse to select "Upload to /", wait for "my74HC595.py" to be uploaded to ESP8266 and then double click"74HC595_and_7_segment_display.py".

14.1_74HC595_and_7_segment_display



Click "Run current script", and you'll see a 1-bit, 7-segment display displaying 0-f in a loop.



The following is the program code:

```
import time
from my74HC595 import Chip74HC595

lists =[0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
        0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e]

chip = Chip74HC595(14,12,13)
try:
    while True:
        for count in range(16):
            chip.shiftOut(0,lists[count])
            time.sleep_ms(500)
except:
    pass
```

Import time and my74HC595 modules.

```
import time
from my74HC595 import Chip74HC595
```

Put the encoding **"0" - "F"** into the list.

```
lists =[0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
        0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e]
```

Define an object, whose pins applys default configuration, to drive 74HC595.

```
chip = Chip74HC595(14,12,13)
```

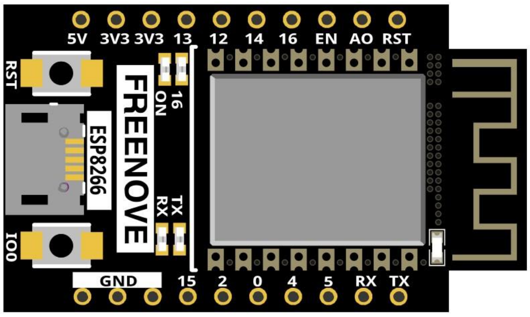
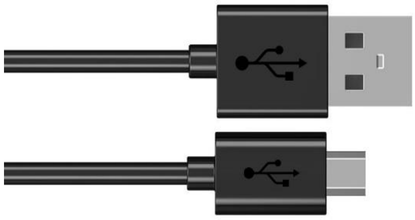
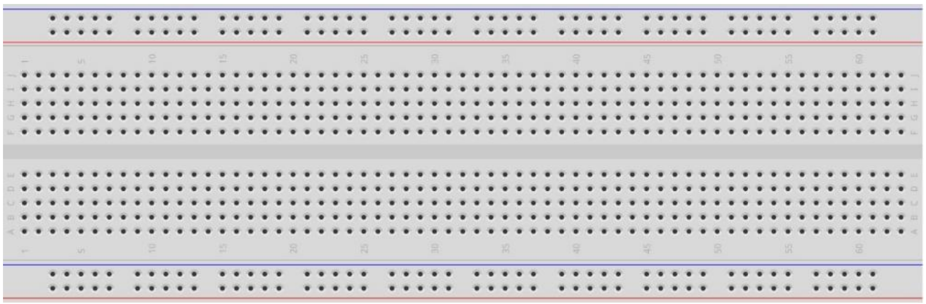
Send data of digital tube to 74HC595 chip.

```
chip.shiftOut(0,lists[count])
```

Project 14.2 4-Digit 7-Segment Display

Now, let's try to control a more-digit 7-segment display.

Component List

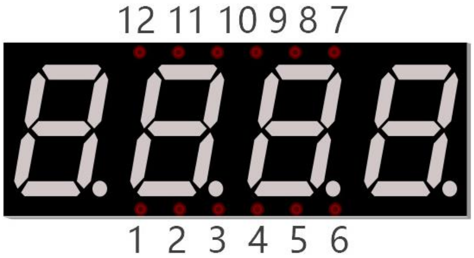
<table>
<tr><td colspan="2">ESP8266 x1

</td><td colspan="2">USB cable
</td></tr>
<tr><td colspan="4">Breadboard x1
</td></tr>
<tr><td>74HC595 x1

</td><td>7-segment display x1
</td><td>Resistor 220Ω x8
</td><td>Jumper wire M/M x21
</td></tr>
</table>

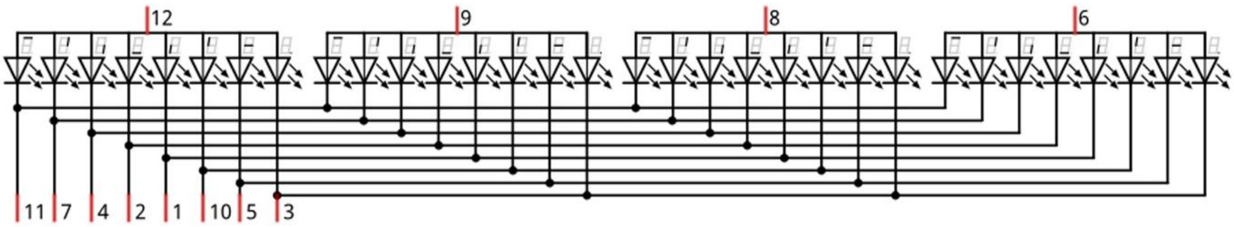
Component knowledge

4 Digit 7-Segment Display

A 4 Digit 7-segment display integrates four 7-Segment Displays into one module, therefore it can display more characters. All of the LEDs contained have a Common Anode and individual Cathodes. Its internal structure and pin designation diagram is shown below:



The internal electronic circuit is shown below, and all 8 LED cathode pins of each 7-Segment Display are connected together.

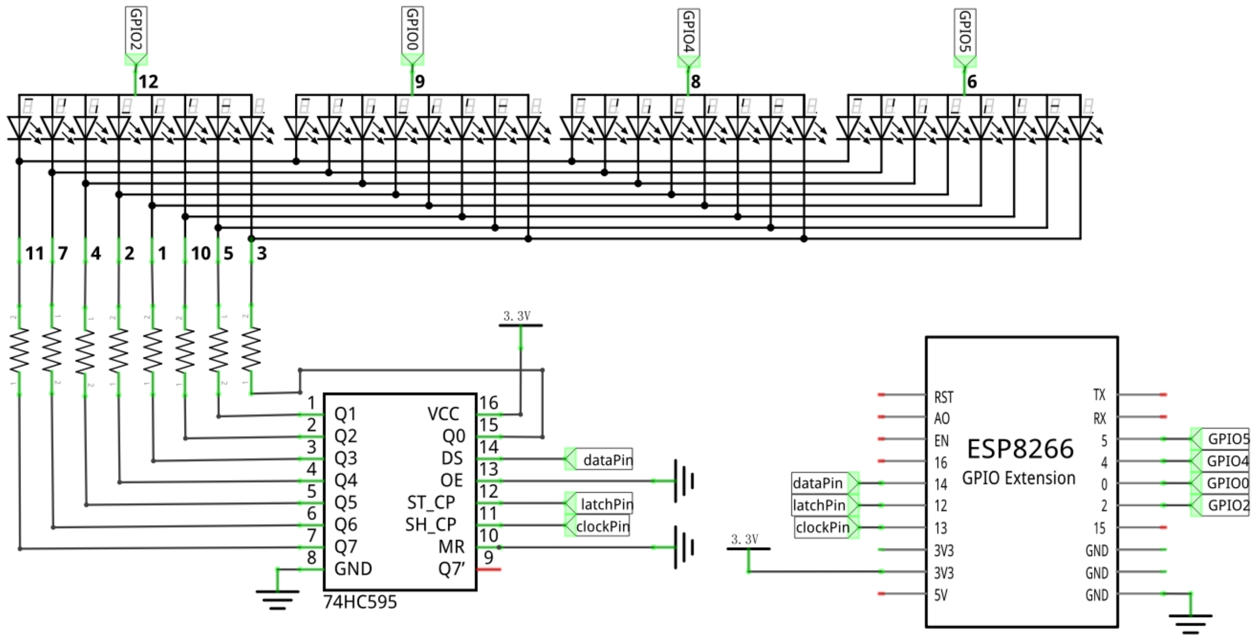


Display method of 4 Digit 7-segment display is similar to 1 Digit 7-segment display. The difference between them is that the 4-Digit displays each Digit visibly in turn, one by one and not together. We need to first send high level to the common end of the first Digit Display, and send low level to the remaining three common ends, and then send content to 8 LED cathode pins of the first Digit Display. At this time, the first 7-Segment Display will show visible content and the remaining three will be OFF.

Similarly, the second, third and fourth 7-Segment Displays will show visible content in turn by scanning the display. Although the four number characters are displayed in turn separately, this process is so fast that it is imperceptible to the naked eye. This is due to the principle of optical afterglow effect and the vision persistence effect in human sight. This is how we can see all 4 number characters at the same time. However, if each number character is displayed for a longer period, you will be able to see that the number characters are displayed separately.

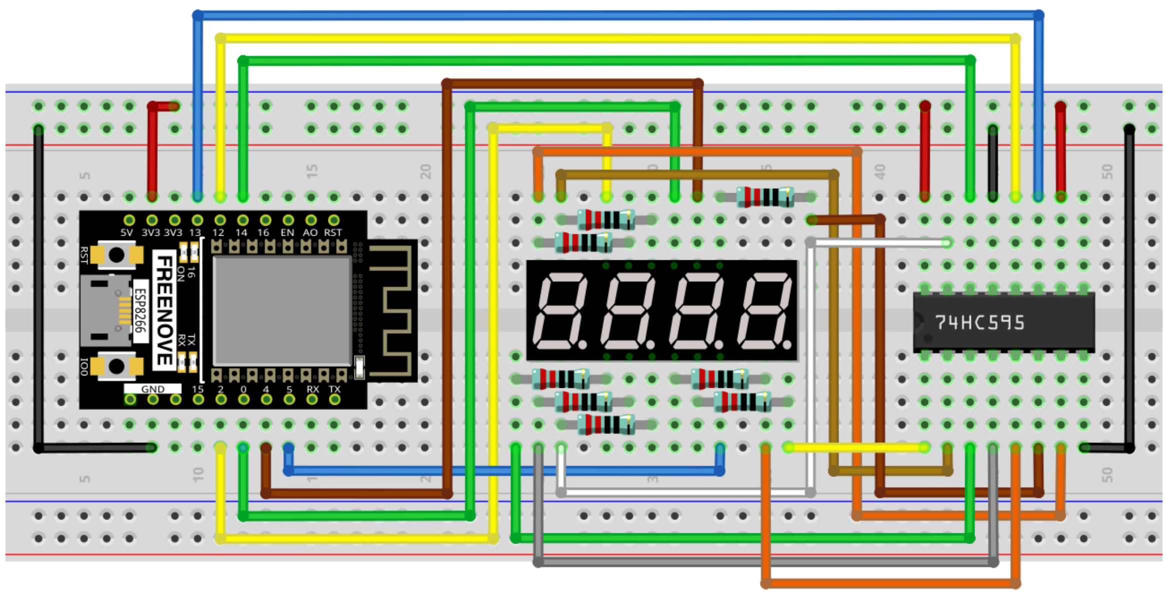
Circuit

Schematic diagram



Q0-Q7 connecting in inverted sequence

Hardware connection：

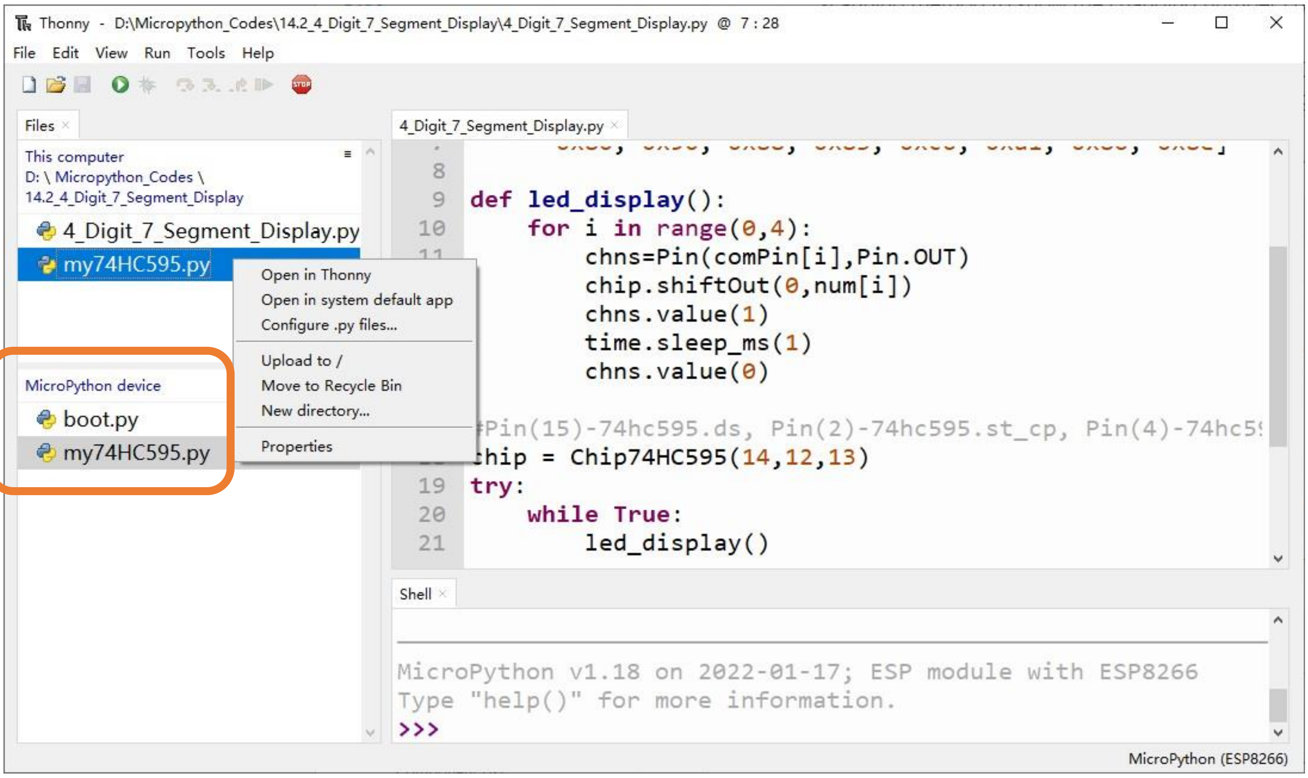


Code

In this code, we use the 74HC595 IC Chip to control the 4-Digit 7-Segment Display, and use the dynamic scanning method to show the changing number characters.

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "14.2_4_Digit_7_Segment_Display". Select "my74HC595.py", right click your mouse to select "Upload to /", wait for "my74HC595.py" to be uploaded to ESP8266 and double click "4_Digit_7_Segment_Display.py".

14.2_4_Digit_7_Segment_Display



Click "Run current script", and the Nixie tube display as shown in the image below.

The following is the program code:

```
import time
from my74HC595 import Chip74HC595
from machine import Pin

comPin = [2,0,4,5];
num = [0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
       0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e]

def led_display():
    for i in range(0,4):
        chns=Pin(comPin[i],Pin.OUT)
        chip.shiftOut(0,num[i])
        chns.value(1)
        time.sleep_ms(1)
        chns.value(0)

#Pin(15)-74hc595.ds, Pin(2)-74hc595.st_cp, Pin(4)-74hc595.sh_cp
chip = Chip74HC595(14,12,13)
try:
    while True:
        led_display()
except:
    pass
```

Import time, my74HC595 and Pin modules.

```
import time
from my74HC595 import Chip74HC595
from machine import Pin
```

Define common anode pins for digital tubes and request a list to put character encodings in it.

```
comPin = [2,0,4,5];
num = [0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8,
       0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e]
```

Request an object to drive 74HC595 and associate pins with it.

```
chip = Chip74HC595(14,12,13)
```

Make the digital tube display “0123”.

```
def led_display():
    for i in range(0,4):
        chns=Pin(comPin[i],Pin.OUT)
        chip.shiftOut(0,num[i])
        chns.value(1)
        time.sleep_ms(1)
        chns.value(0)
```

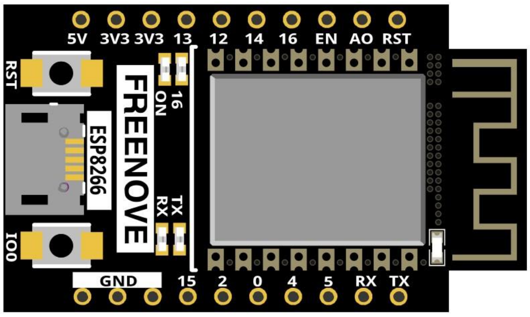
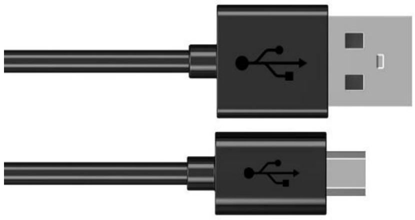
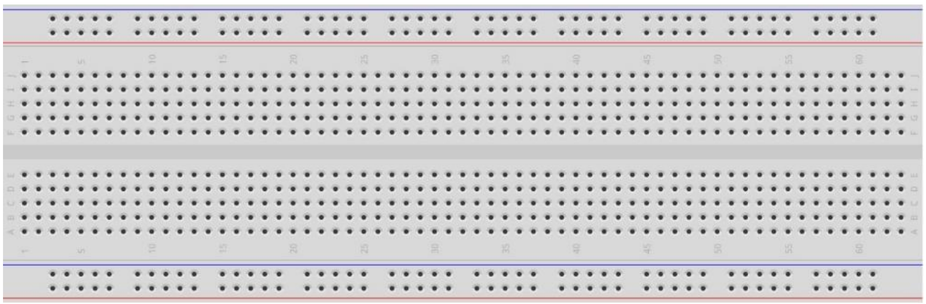
Chapter 15 74HC595 & LED Matrix

Thus far we have learned how to use the 74HC595 IC Chip to control the Bar Graph LED and the 7-Segment Display. We will now use 74HC595 IC Chips to control an LED Matrix.

Project 15.1 LED Matrix

In this project, we will use two 74HC595 IC chips to control a monochrome (one color) (8X8) LED Matrix to make it display both simple graphics and characters.

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>74HC595 x2

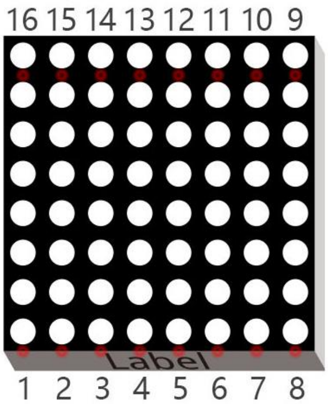
</td><td>8*8 LEDMatrix x1

</td><td>Resistor 220Ω x8
</td><td>Jumper wire M/M
</td></tr>
</table>

Component knowledge

LED matrix

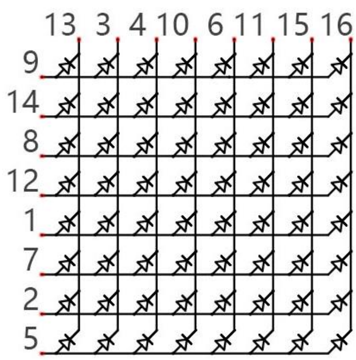
A LED Matrix is a rectangular display module that consists of a uniform grid of LEDs. The following is an 8X8 monochrome (one color) LED Matrix containing 64 LEDs (8 rows by 8 columns).



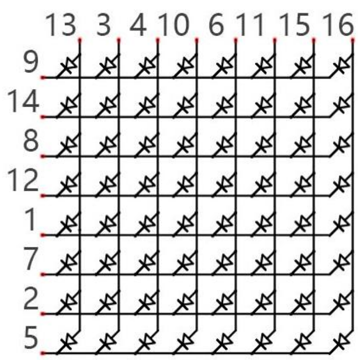
In order to facilitate the operation and reduce the number of ports required to drive this component, the Positive Poles of the LEDs in each row and Negative Poles of the LEDs in each column are respectively connected together inside the LED Matrix module, which is called a Common Anode. There is another arrangement type. Negative Poles of the LEDs in each row and the Positive Poles of the LEDs in each column are respectively connected together, which is called a Common Cathode.

The LED Matrix that we use in this project is a Common Anode LED Matrix.

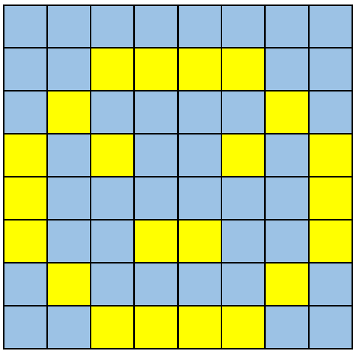
Connection mode of common anode



Connection mode of common cathode



Here is how a Common Anode LED Matrix works. First, choose 16 ports on ESP8266 board to connect to the 16 ports of LED Matrix. Configure one port in columns for low level, which makes that column the selected port. Then configure the eight port in the row to display content in the selected column. Add a delay value and then select the next column that outputs the corresponding content. This kind of operation by column is called Scan. If you want to display the following image of a smiling face, you can display it in 8 columns, and each column is represented by one byte.



1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	1	0
1	0	1	0	0	1	0	1
1	0	0	0	0	0	0	1
1	0	0	1	1	0	0	1
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0

Column	Binary	Hexadecimal
1	0001 1100	0x1c
2	0010 0010	0x22
3	0101 0001	0x51
4	0100 0101	0x45
5	0100 0101	0x45
6	0101 0001	0x51
7	0010 0010	0x22
8	0001 1100	0x1c

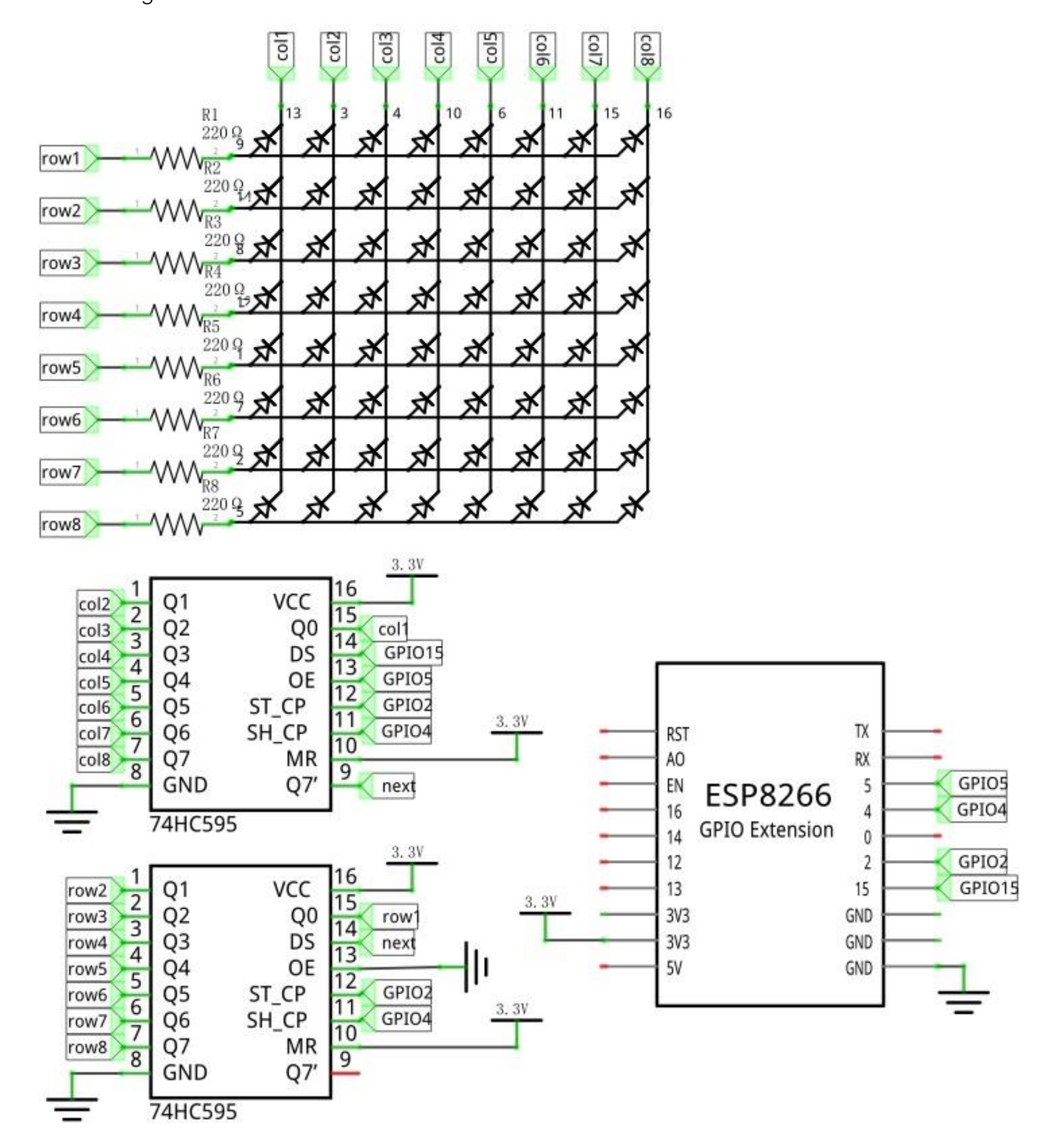
To begin, display the first column, then turn off the first column and display the second column. (and so on) .... turn off the seventh column and display the 8th column, and then start the process over from the first column again like the control of LED Bar Graph project. The whole process will be repeated rapidly in a loop. Due to the principle of optical afterglow effect and the vision persistence effect in human sight, we will see a picture of a smiling face directly rather than individual columns of LEDs turned ON one column at a time (although in fact this is the reality we cannot perceive).

Then, to save the number of GPIO, we use a 74HC595. When the first column is turned ON, set the lights that need to be displayed in the first column to "1", otherwise to "0", as shown in the above example, where the value of the first column is 0x1c. This value is sent to 74HC595 to control the display of the first column of the LEDMatrix. Following the above idea, turn OFF the display of the first column, then turn ON the second column, and then send the value of the second column to 74HC595 ...... Until each column is displayed, the LEDMatrix is displayed again from the first column.

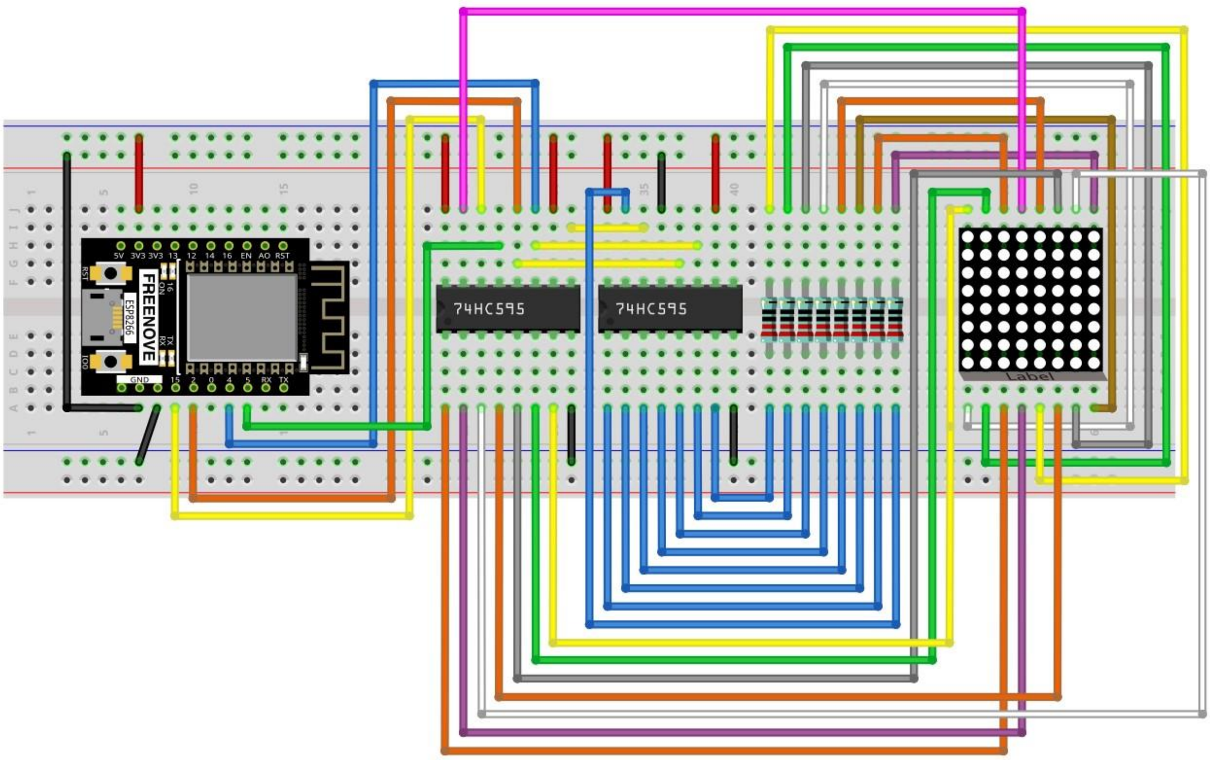
Circuit

In circuit of this project, the power pin of the 74HC595 IC Chip is connected to 3.3V. It can also be connected to 5V to make LED Matrix brighter.

Schematic diagram



Hardware connection:

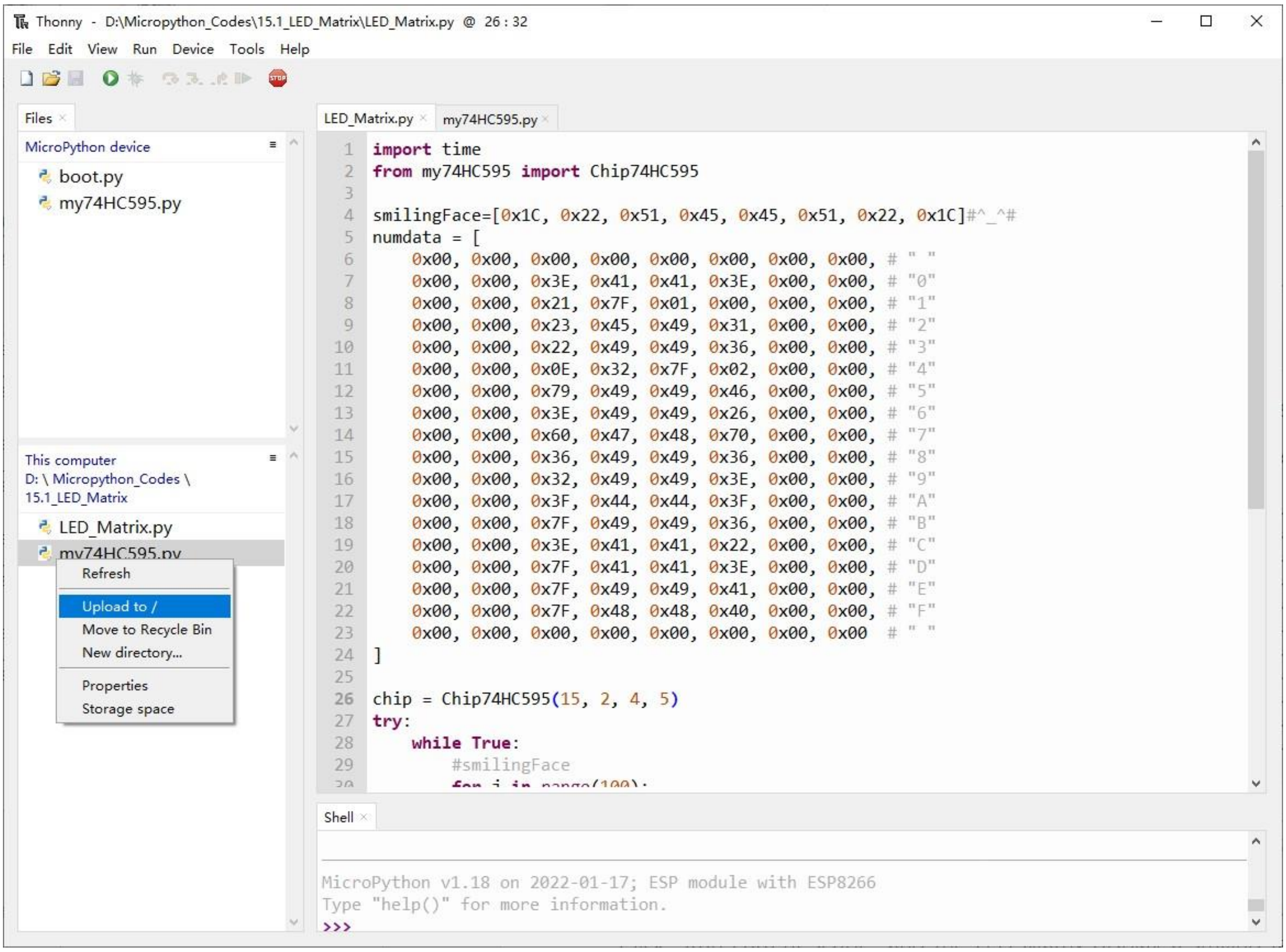


Code

The following code will make LEDMatrix display a smiling face, and then display scrolling character "0-F".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "Micropython_Codes".Select "HC595.py", right click your mouse to select "Upload to /", wait for "HC595.py" to be uploaded to ESP8266 and double click "LED_Matrix.py".

15.1_LED_Matrix



Click “Run current script”, and the LED Matrix display a smiling face, and then display characters "0 to F" scrolling in a loop on the LED Matrix.

The following is the program code:

```
import time
from my74HC595 import Chip74HC595

smilingFace=[0x1C, 0x22, 0x51, 0x45, 0x45, 0x51, 0x22, 0x1C]#^_^#
numdata = [
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, # " "
    0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, # "0"
    0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, # "1"
    0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, # "2"
    0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, # "3"
    0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, # "4"
    0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, # "5"
    0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, # "6"
    0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, # "7"
    0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, # "8"
    0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, # "9"
    0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, # "A"
    0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, # "B"
    0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, # "C"
    0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, # "D"
    0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, # "E"
    0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00, # "F"
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00  # " "
]

chip = Chip74HC595(15, 2, 4, 5)
try:
    while True:
        #smilingFace
        for j in range(30):
            cols = 0x01
            for i in range(8):
                chip.disable()
                chip.shiftOut(1, smilingFace[i])
                chip.shiftOut(0, ~cols)
                cols <<= 1
                chip.enable()
                time.sleep_us(50)
        #numdata
        for i in range(136):
            for _ in range(3):
                cols = 0x01
                for j in range(i, 8+i):
```

```
                    chip.disable()
                    chip.shiftOut(1, numdata[j])
                    chip.shiftOut(0, ~cols)
                    cols <<= 1
                    chip.enable()
                    time.sleep_us(50)
except:
    pass
```

Import time and my 74HC595 modules.

```
import time
from my74HC595 import Chip74HC595
```

Use a nesting of two for loops to display a smiling face.

```
        for j in range(100):
            cols = 0x01
            for i in range(8):
                chip.disable()
                chip.shiftOut(1, smilingFace[i])
                chip.shiftOut(0, ~cols)
                cols <<= 1
                chip.enable()
                time.sleep_us(500)
```

Use a nesting of three for loops to display "0"-"F".

```
        for i in range(136):
            for _ in range(5):
                cols = 0x01
                for j in range(i, 8+i):
                    chip.disable()
                    chip.shiftOut(1, numdata[j])
                    chip.shiftOut(0, ~cols)
                    cols <<= 1
                    chip.enable()
                    time.sleep_us(500)
```

The amount of pins of ESP8266 is limited, so we need to find ways to save pins. If we use ESP8266's GPIO to control the LEDMatrix instead of 74HC595, we need 16 pins to drive LED matrix. In this example, we use two 74HC595 chips to drive the LED matrix, requiring only three pins, so that we could save the rest of 13 pins.

Reference

Class HC595

Before each use of HC595, please make sure HC595.py has been uploaded to "/" of ESP8266, and then add the statement "**import HC595**" to the top of the python file.

Chip74HC595(): The object to control LEDMatrix.

chip=Chip74HC595() or **chip=Chip74HC595(15,2,4,5)**

set_bit_data(data): Write data to 74HC595.

clear(): Clear the latch data of 74HC595.

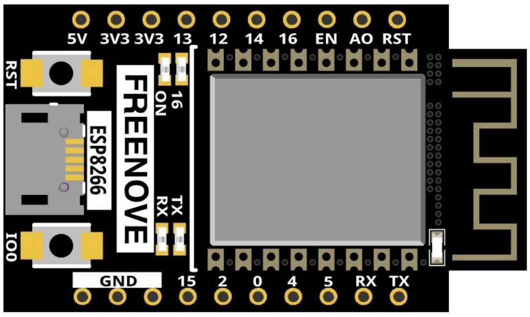
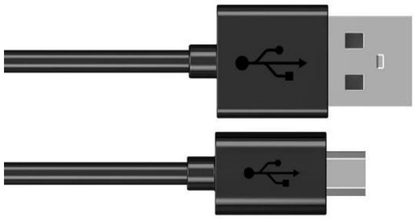
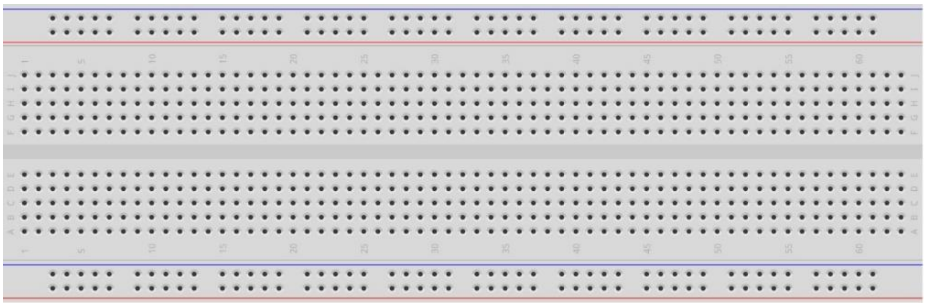
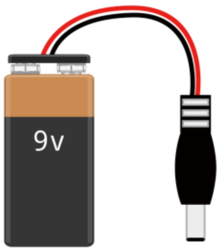
Chapter 16 Relay & Motor

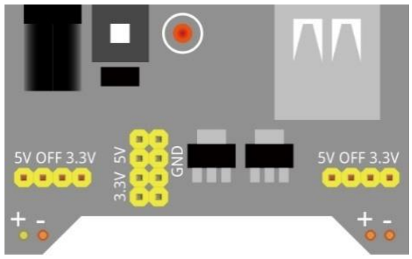
In this chapter, we will learn a kind of special switch module, Relay Module.

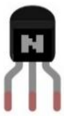
Project 16.1 Relay & Motor

In this project, we will use a Push Button Switch indirectly to control the motor via a Relay.

Component List

ESP8266 x1 	USB cable 
Breadboard x1 	
Jumper wire M/M x16 	9V battery (prepared by yourself) & battery line 

Resistor 10kΩ x2	Resistor 1kΩ x1	Resistor 220Ω x1	Breadboard Power module x1
			

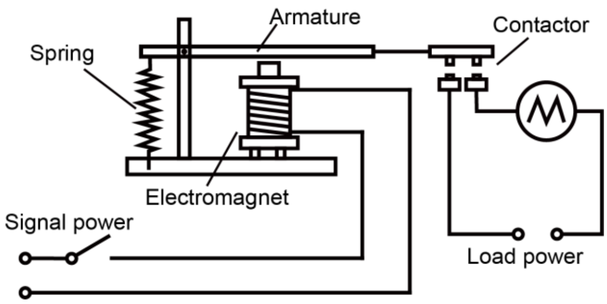
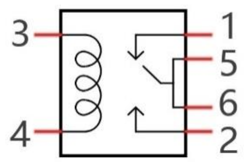
NPN transistor x1	Relay x1	Motor x1	Push button x1	LED x1	Diode x1
					

Component knowledge

Relay

A relay is a safe switch which can use low power circuit to control high power circuit. It consists of electromagnet and contacts. The electromagnet is controlled by low power circuit and contacts are used in high power circuit. When the electromagnet is energized, it will attract contacts.

The following is a schematic diagram of a common relay and the feature and circuit symbol of a 5V relay used in this project:

Diagram	Feature:	Symbol
		

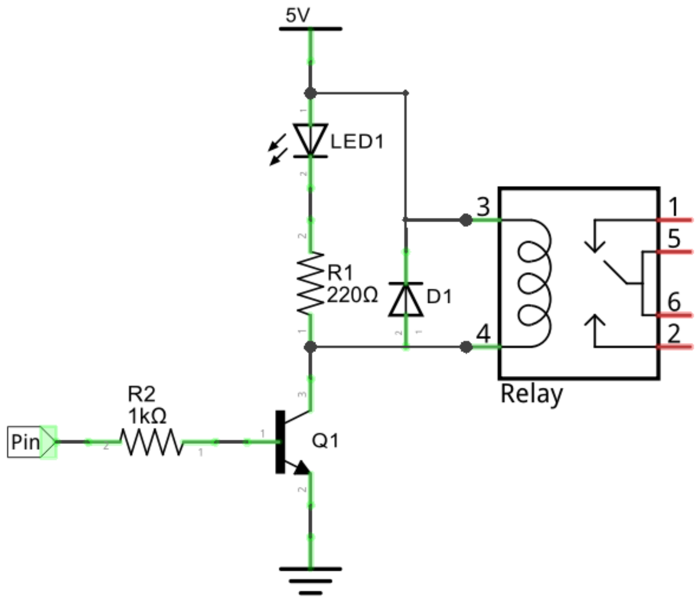
Pin 5 and pin 6 are connected to each other inside. When the coil pin 3 and 4 get connected to 5V power supply, pin 1 will be disconnected to pin 5&6 and pin 2 will be connected to pin 5&6. So pin 1 is called close end, pin 2 is called open end.

Inductor

The symbol of Inductance is "L" and the unit of inductance is the "Henry" (H). Here is an example of how this can be encountered: 1H=1000mH, 1mH=1000μH.

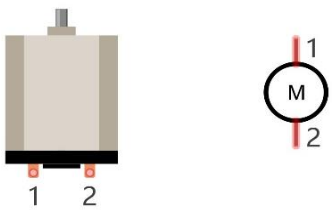
An inductor is an energy storage device that converts electrical energy into magnetic energy. Generally, it consists of winding coil, with a certain amount of inductance. Inductors hinder the change of current passing through it. When the current passing through it increases, it will attempt to hinder the increasing trend of current; and when the current passing through it decreases, it will attempt to hinder the decreasing trend of current. So the current passing through inductor is not transient.

The reference circuit for relay is as follows. The coil of relays can be equivalent to that of inductors, when the transistor disconnects power supply of the relay, the current in the coil of the relay can't stop immediately, causing an impact on power supply. So a parallel diode will get connected to both ends of relay coil pin in reversing direction, then the current will pass through diode, avoiding the impact on power supply.

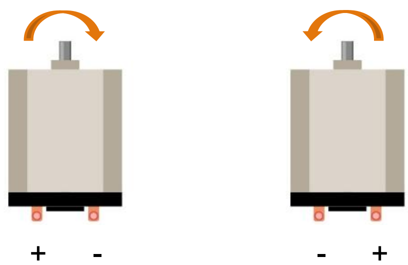


Motor

A motor is a device that converts electrical energy into mechanical energy. Motor consists of two parts: stator and rotor. When motor works, the stationary part is stator, and the rotating part is rotor. Stator is usually the outer case of motor, and it has terminals to connect to the power. Rotor is usually the shaft of motor, and can drive other mechanical devices to run. Diagram below is a small DC motor with two pins.

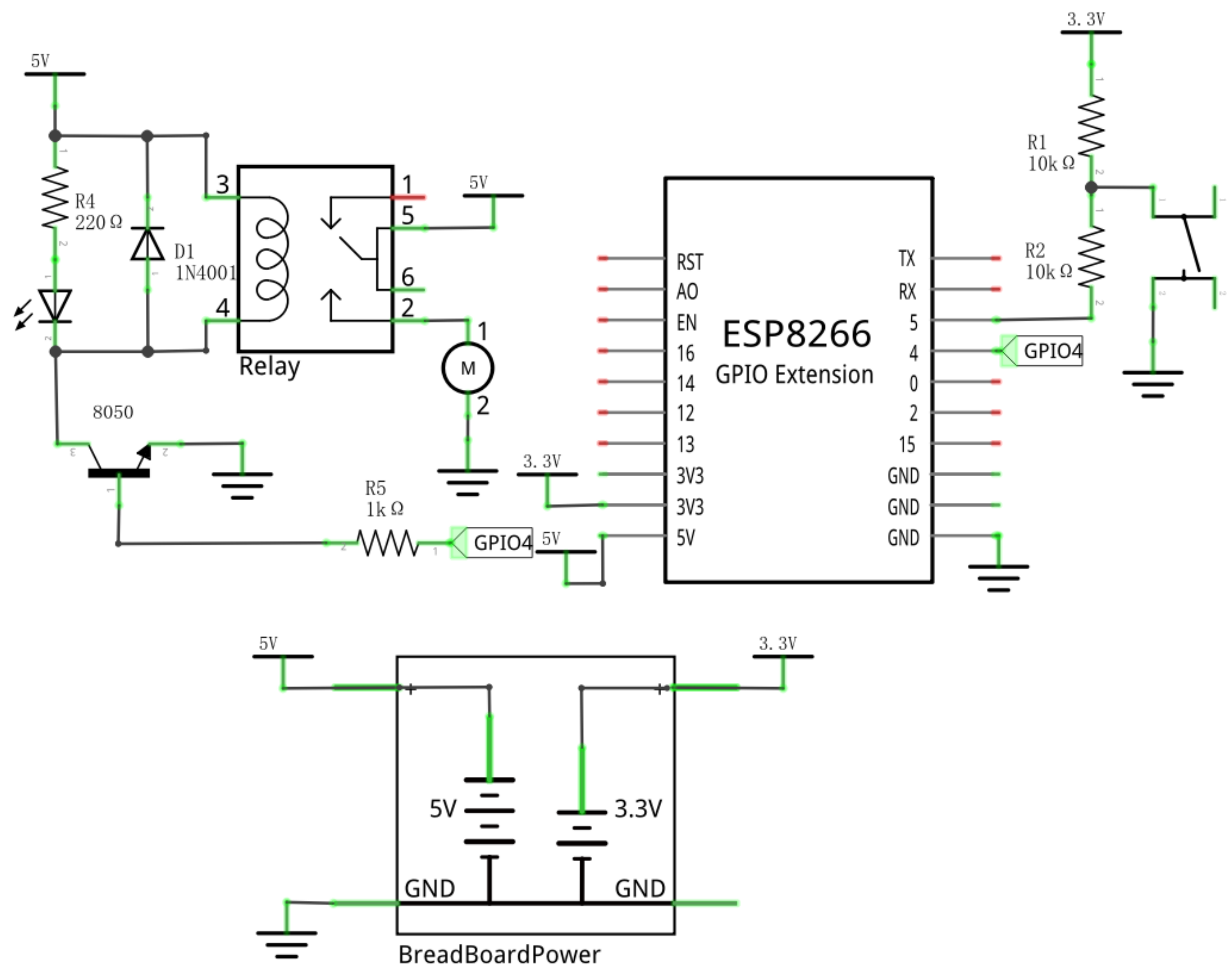


When a motor gets connected to the power supply, it will rotate in one direction. Reverse the polarity of power supply, then the motor rotates in opposite direction.

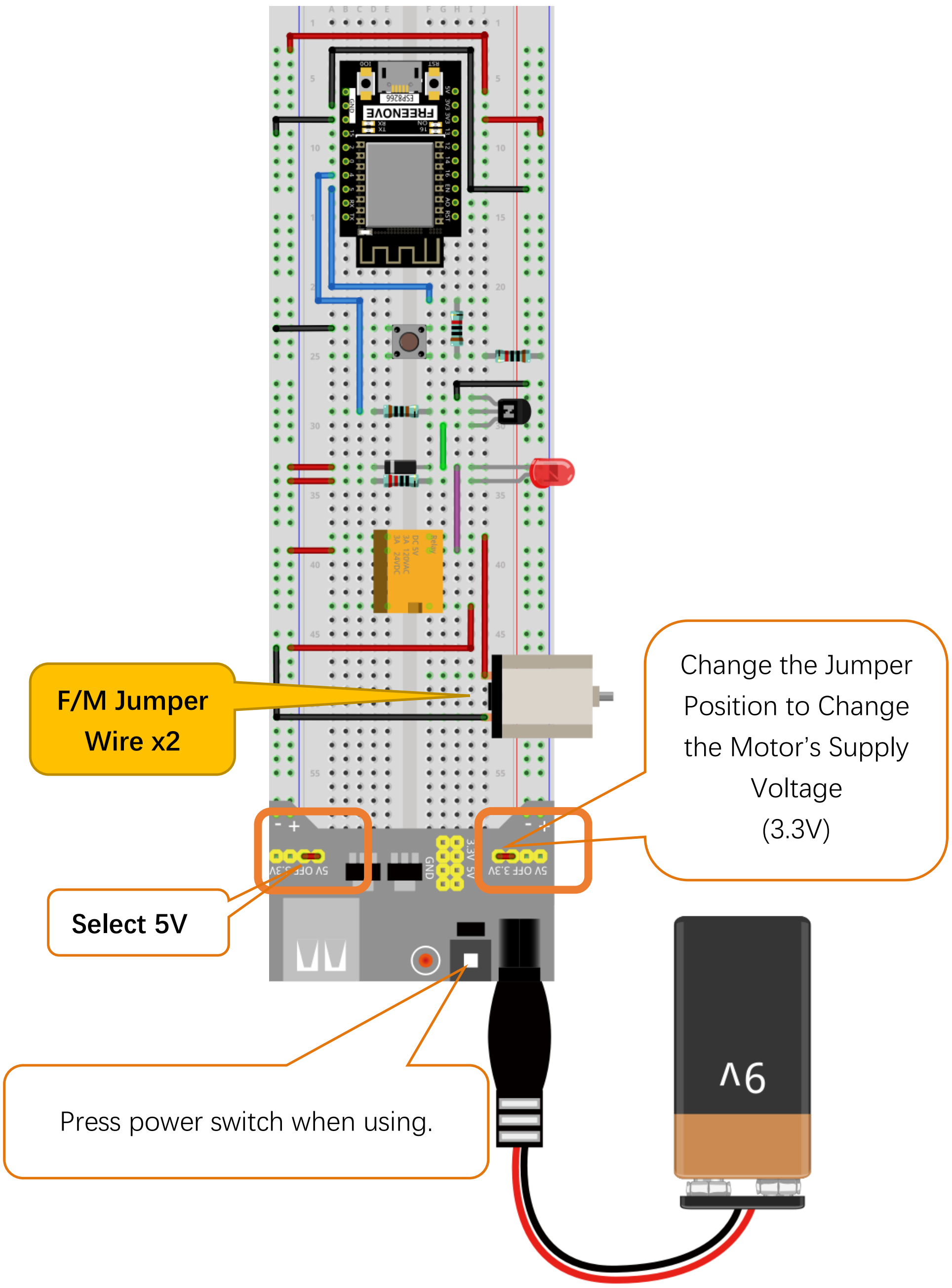


Circuit

Schematic diagram



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



Note: the motor circuit uses A large current, about 0.2-0.3A without load. We recommend that you use a 9V battery to power your system.

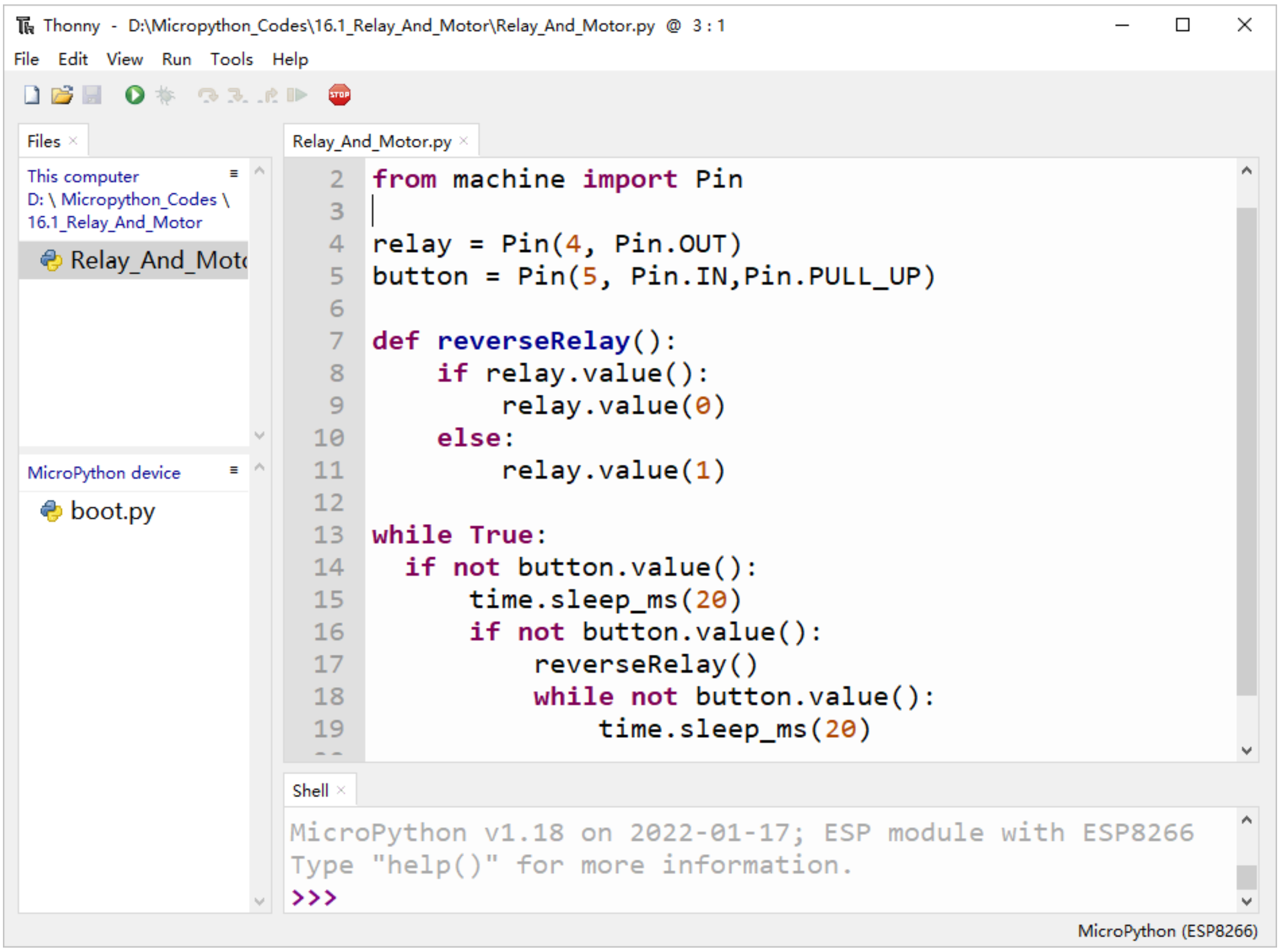
Code

Use buttons to control the relays and motors.

Move the program folder "**Freenove_Ultimate_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" → "16.1_Relay_And_Motor" and double click "Relay_And_Motor.py".

16.1_Relay_And_Motor



Click "Run current script". When the DC Motor is connected to a power supply, it will rotate in one direction. If you reverse the polarity of the power supply, the DC Motor will rotate in opposite direction.

The following is the program code:

```
import time
from machine import Pin

relay = Pin(4, Pin.OUT)
button = Pin(5, Pin.IN,Pin.PULL_UP)

def reverseRelay():
    if relay.value():
        relay.value(0)
    else:
        relay.value(1)

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

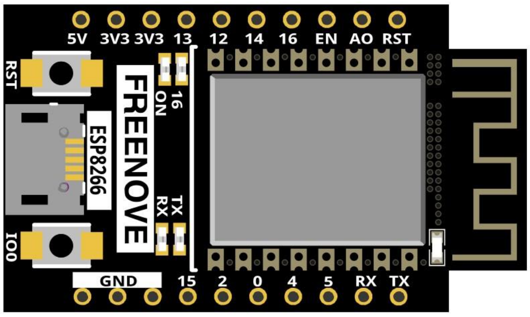
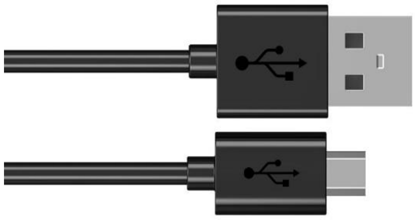
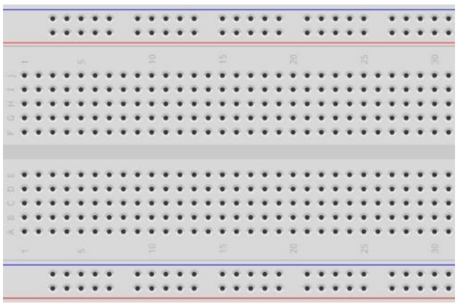
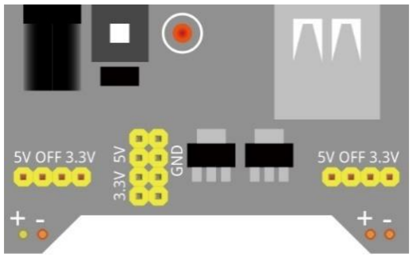
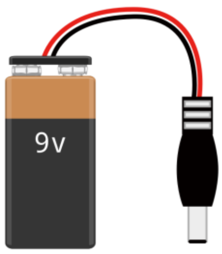
This section of code is basically the same as that of project Tablelamp. If you don't understand the program, you can click here to go back to the Tablelamp and study again.

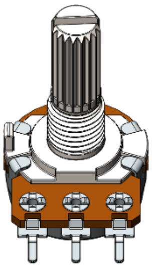
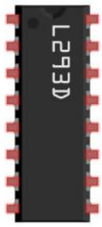
Chapter 17 Motor & Driver

Project 17.1 Control Motor with Potentiometer

Control the direction and speed of the motor with a potentiometer.

Component List

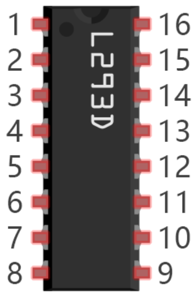
ESP8266 x1 	USB cable 
Breadboard x1 	Breadboard Power module x1 
Jumper wire M/M x12 	9V battery (prepared by yourself) & battery line 

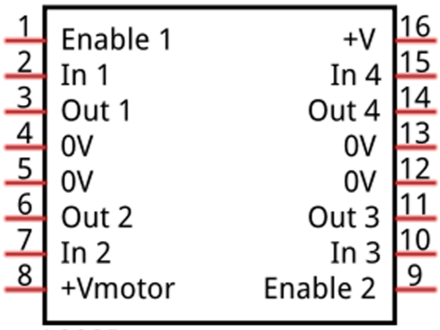
Rotary potentiometer x1	Motor x1	L293D
		

Component knowledge

L293D

L293D is an IC Chip (Integrated Circuit Chip) with a 4-channel motor drive. You can drive a Unidirectional DC Motor with 4 ports or a Bi-Directional DC Motor with 2 ports or a Stepper Motor (Stepper Motors are covered later in this Tutorial).





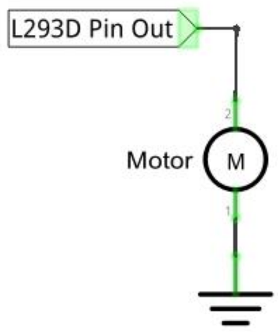
L293D

Port description of L293D module is as follows:

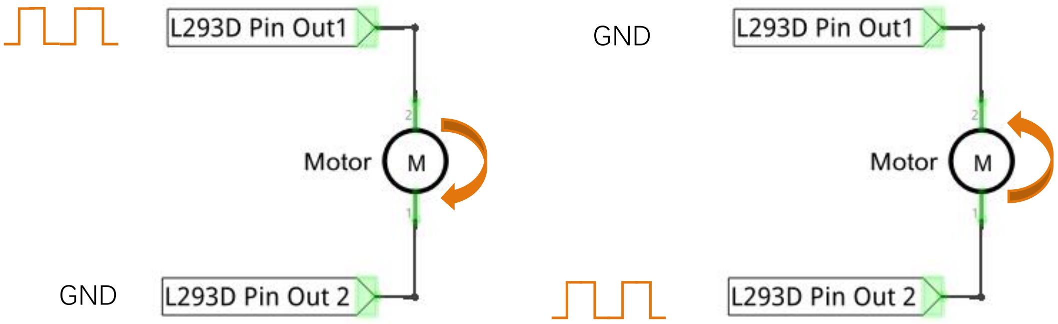
Pin name	Pin number	Description
In x	2, 7, 10, 15	Channel x digital signal input pin
Out x	3, 6, 11, 14	Channel x output pin, input high or low level according to In x pin, get connected to +Vmotor or 0V
Enable1	1	Channel 1 and channel 2 enable pin, high level enable
Enable2	9	Channel 3 and channel 4 enable pin, high level enable
0V	4, 5, 12, 13	Power cathode (GND)
+V	16	Positive electrode (VCC) of power supply, supply voltage 3.0~36V
+Vmotor	8	Positive electrode of load power supply, provide power supply for the Out pin x, the supply voltage is +V~36V

For more detail, please refer to the datasheet for this IC Chip.

When using L293D to drive DC motor, there are usually two connection options.
The following connection option uses one channel of the L239D, which can control motor speed through the PWM, However the motor then can only rotate in one direction.


L293D Pin Out
Motor
M

The following connection uses two channels of the L239D: one channel outputs the PWM wave, and the other channel connects to GND, therefore you can control the speed of the motor. When these two channel signals are exchanged, not only can they control the speed of motor, but also control the direction of the motor.

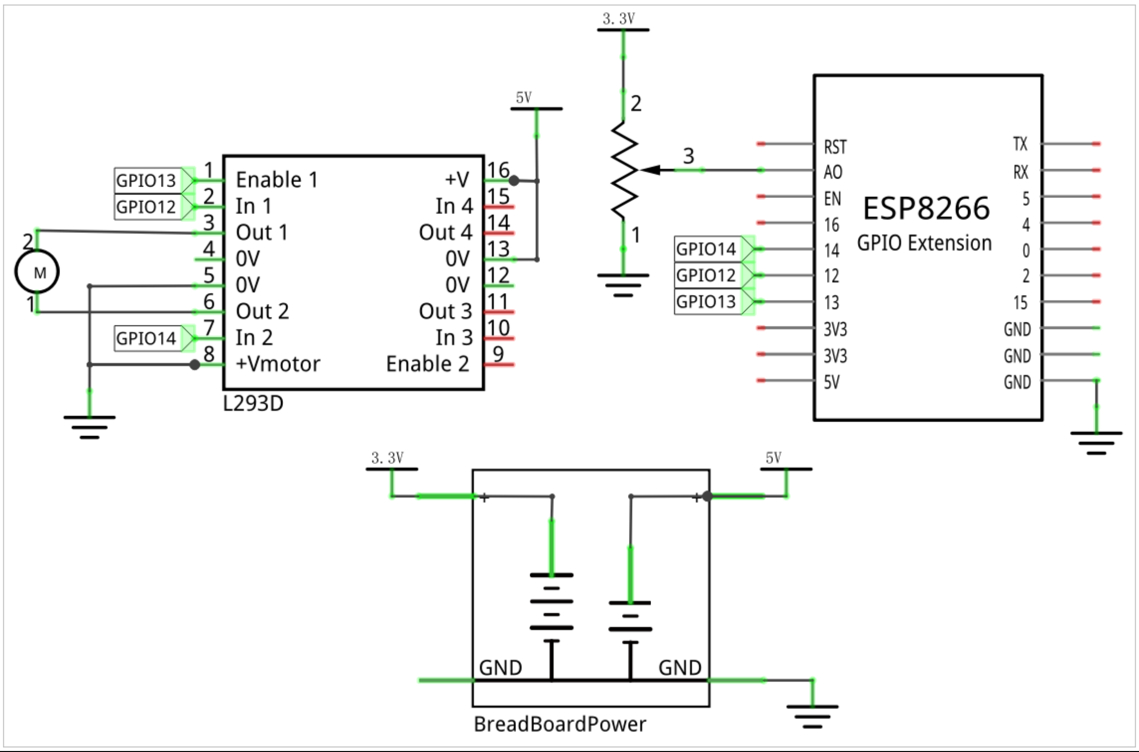


In practical use the motor is usually connected to channels 1 and 2 by outputting different levels to in1 and in2 to control the rotational direction of the motor, and output to the PWM wave to Enable1 port to control the motor's rotational speed. If the motor is connected to channel 3 and 4 by outputting different levels to in3 and in4 to control the motor's rotation direction, and output to the PWM wave to Enable2 pin to control the motor's rotational speed.

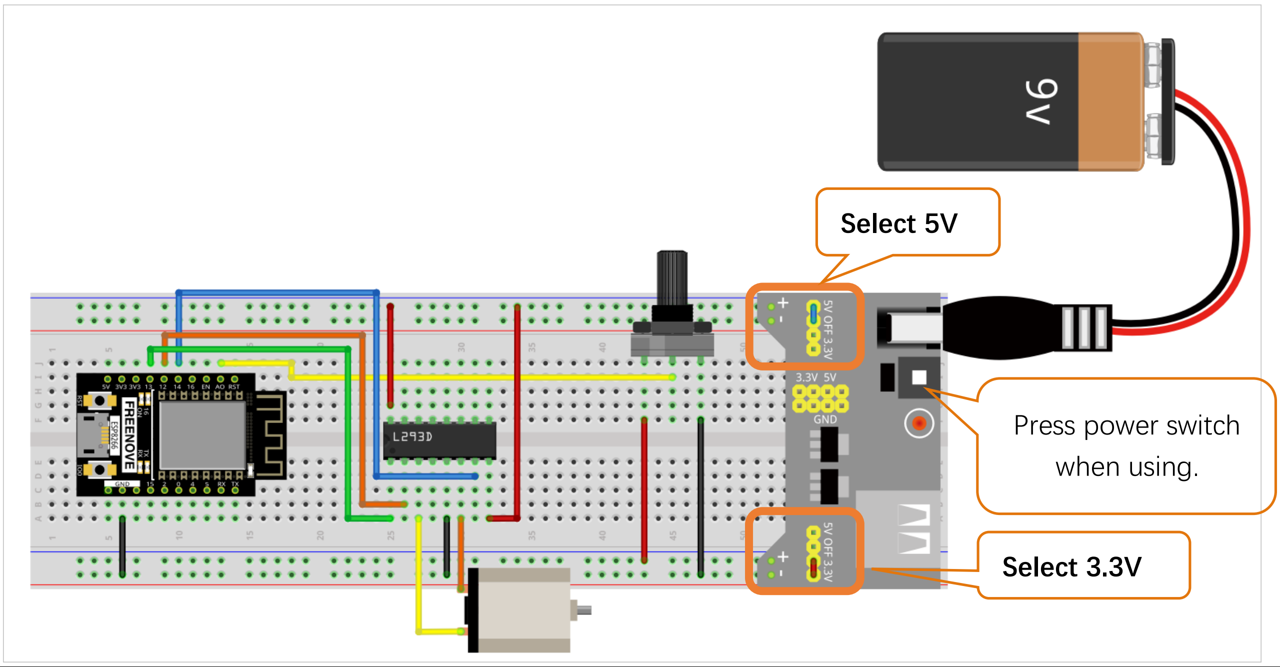
Circuit

Use caution when connecting this circuit because the DC Motor is a high-power component. **Do not use the power provided by the ESP8266 to power the motor directly, as this may cause permanent damage to your RPi!** The logic circuit can be powered by the ESP8266's power or an external power supply, which should share a common ground with ESP8266.

Schematic diagram



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

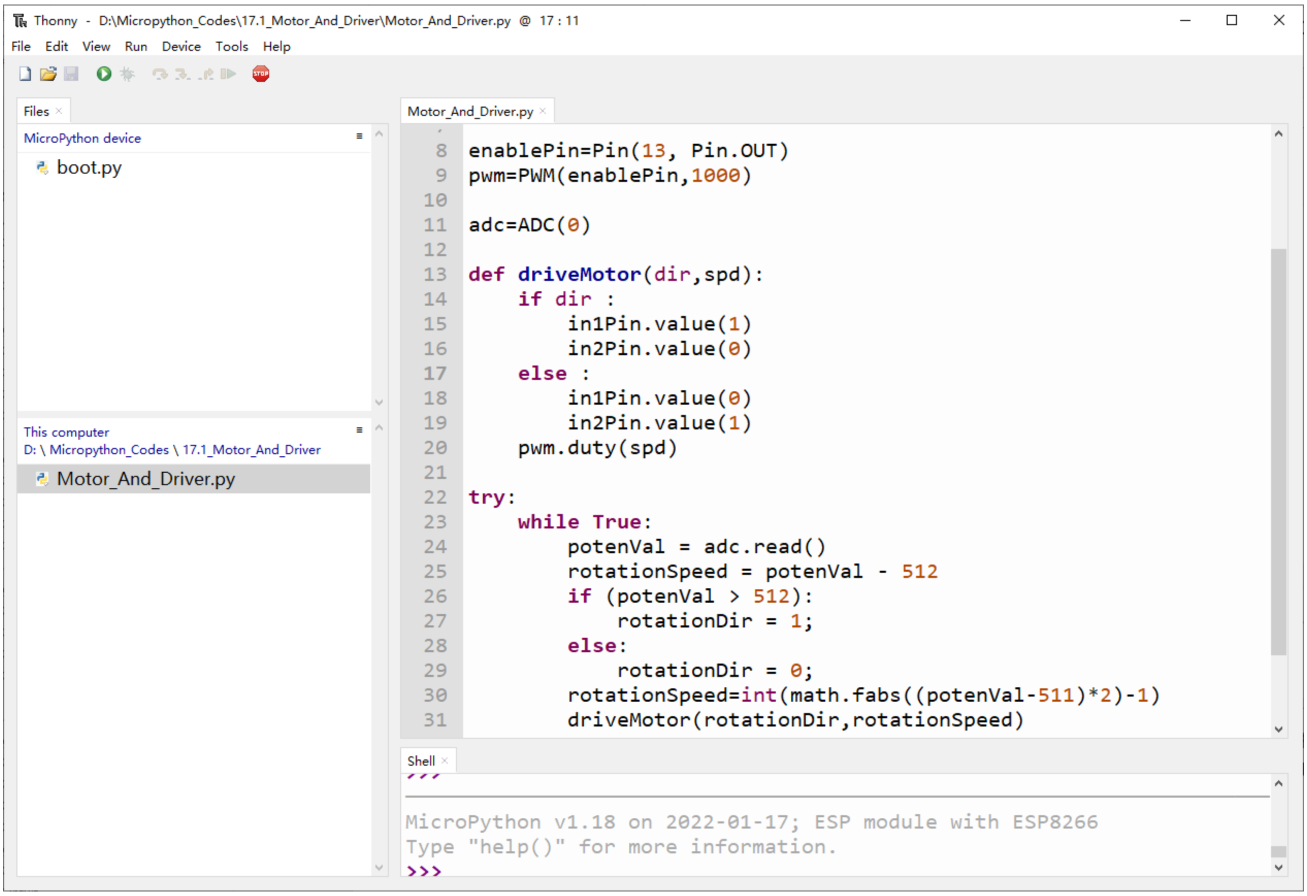


Code

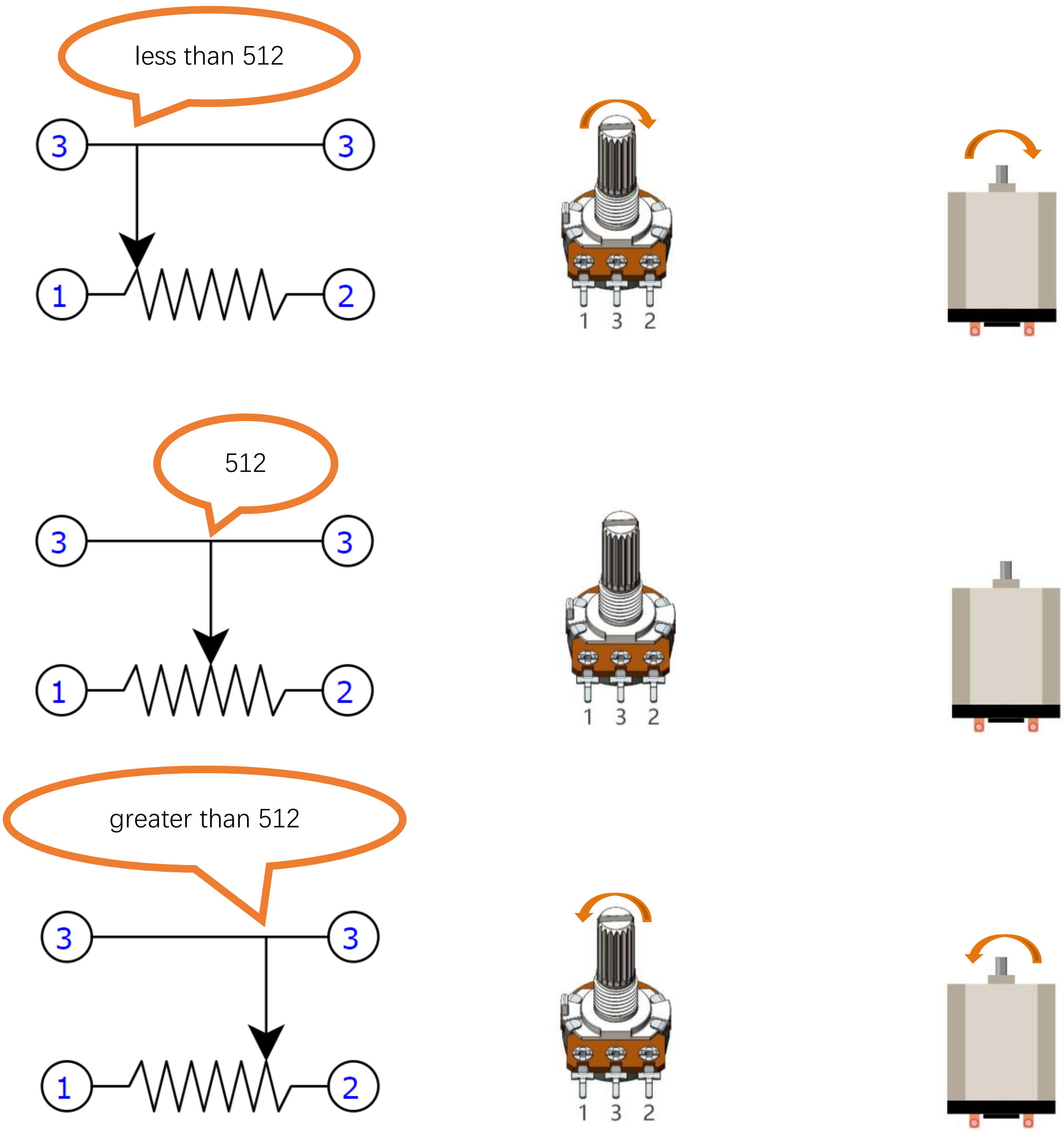
Move the program folder "**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "17.1_Motor_And_Driver" and double click "Motor_And_Driver.py".

17.1_Motor_And_Driver



Click "Run current script", rotate the potentiometer in one direction and the motor speeds up slowly in one direction. Rotate the potentiometer in the other direction and the motor will slow down to stop. And then rotate it in the original direction to accelerate the motor.


less than 512
3
3
1
2
1 3 2
512
3
3
1
2
1 3 2
greater than 512
3
3
1
2
1 3 2

The following is the Code:

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

in1Pin=Pin(12, Pin.OUT)
in2Pin=Pin(14, Pin.OUT)

enablePin=Pin(13, Pin.OUT)
pwm=PWM(enablePin,1000)
adc=ADC(0)

def driveMotor(dir,spd):
    if dir :
        in1Pin.value(1)
        in2Pin.value(0)
    else :
        in1Pin.value(0)
        in2Pin.value(1)
    pwm.duty(spd)

try:
    while True:
        potenVal = adc.read()
        rotationSpeed = potenVal - 512
        if (potenVal > 512):
            rotationDir = 1;
        else:
            rotationDir = 0;
        rotationSpeed=int(math.fabs((potenVal-511)*2)-1)
        driveMotor(rotationDir,rotationSpeed)
        time.sleep_ms(10)
except:
    pass
```

The ADC of ESP8266 has a 10-bit accuracy, corresponding to a range from 0 to 1023. In this program, set the number 512 as the midpoint. If the value of ADC is less than 512, make the motor rotate in one direction. If the value of ADC is greater than 512, make the motor rotate in the other direction. Subtract 512 from the ADC value and take the absolute value, and then divide this result by 2 to be the speed of the motor.

```
        potenVal = adc.read()
        rotationSpeed = potenVal - 512
        if (potenVal > 512):
            rotationDir = 1;
        else:
            rotationDir = 0;
        rotationSpeed=int(math.fabs((potenVal-511)*2)-1)
        driveMotor(rotationDir,rotationSpeed)
        time.sleep_ms(10)
```

Initialize pins of L293D chip.

```
in1Pin=Pin(12, Pin.OUT)
in2Pin=Pin(14, Pin.OUT)

enablePin=Pin(13, Pin.OUT)
pwm=PWM(enablePin,1000)
```

Initialize ADC pins, set the range of voltage to 0-3.3V and the acquisition width of data to 0-1023.

```
adc=ADC(0)
```

Function driveMotor is used to control the rotation direction and speed of the motor. The **dir** represents direction while **spd** refers to speed.

```
def driveMotor(dir,spd):
    if dir :
        in1Pin.value(1)
        in2Pin.value(0)
    else :
        in1Pin.value(0)
        in2Pin.value(1)
    pwm.duty(spd)
```

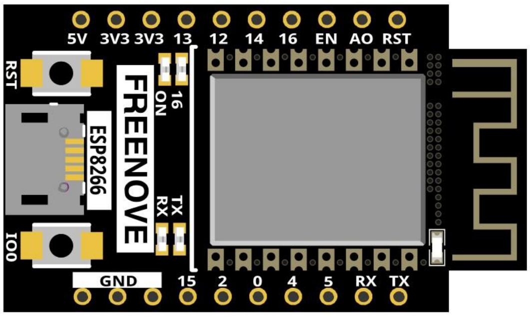
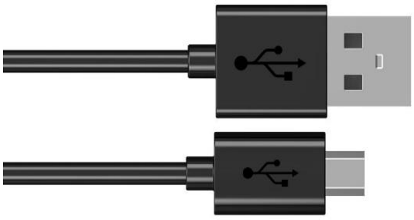
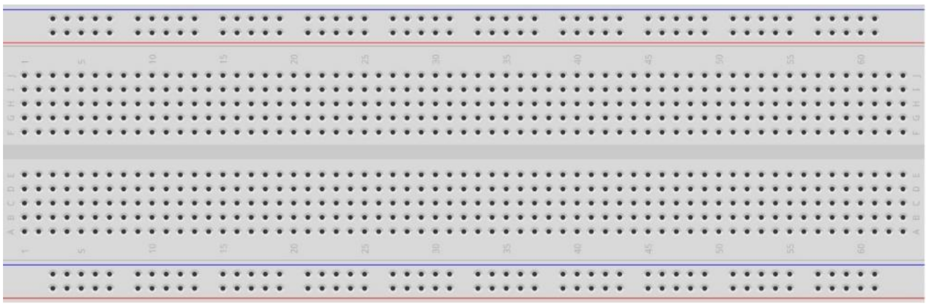
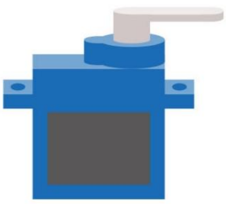
Chapter 18 Servo

Previously, we learned how to control the speed and rotational direction of a Motor. In this chapter, we will learn about Servos which are a rotary actuator type motor that can be controlled to rotate to specific angles.

Project 18.1 Servo Sweep

First, we need to learn how to make a Servo rotate.

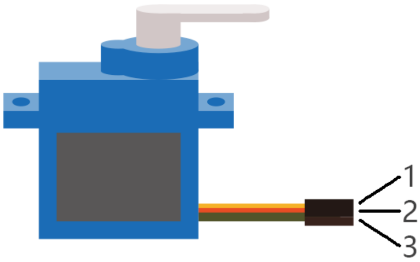
Component List

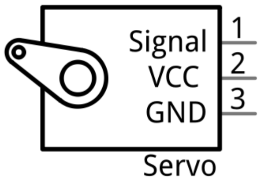
ESP8266 x1 	USB cable 
Breadboard x1 	
Servo x1 	Jumper wire M/M x5 

Component knowledge

Servo

Servo is a compact package which consists of a DC Motor, a set of reduction gears to provide torque, a sensor and control circuit board. Most Servos only have a 180-degree range of motion via their “horn”. Servos can output higher torque than a simple DC Motor alone and they are widely used to control motion in model cars, model airplanes, robots, etc. Servos have three wire leads which usually terminate to a male or female 3-pin plug. Two leads are for electric power: Positive (2-VCC, Red wire), Negative (3-GND, Brown wire), and the signal line (1-Signal, Orange wire) as represented in the Servo provided in your Kit.





We will use a 50Hz PWM signal with a duty cycle in a certain range to drive the Servo. The time interval of 0.5ms-2.5ms of PWM single cycle high level corresponds to the Servo angle 0 degrees - 180 degree linearly. Part of the corresponding values are as follows:

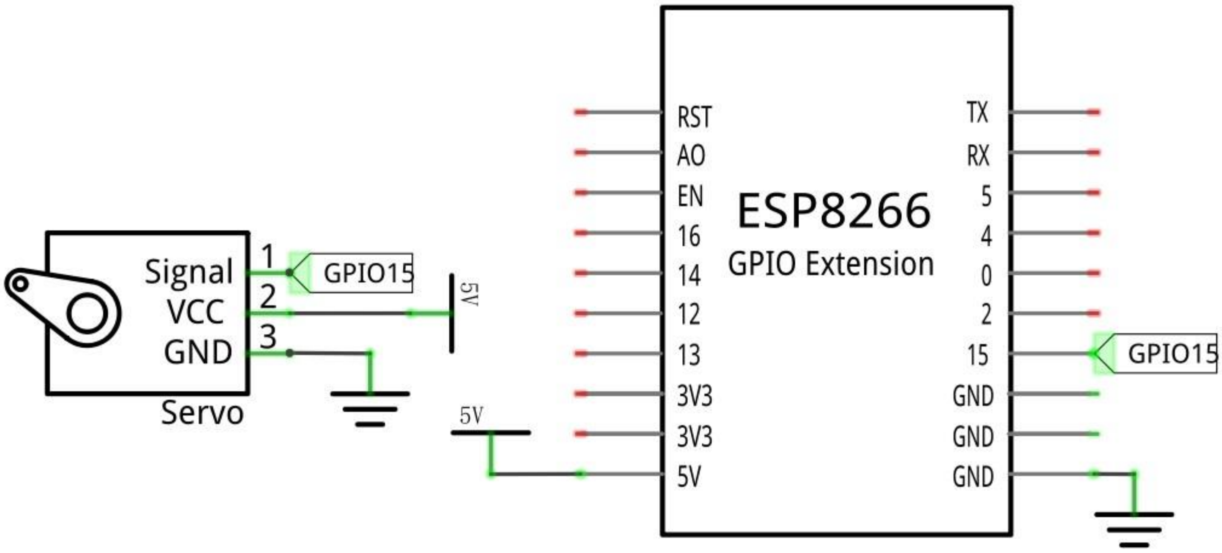
High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	0 degree
2ms	45 degree
2.5ms	180 degree

When you change the Servo signal value, the Servo will rotate to the designated angle.

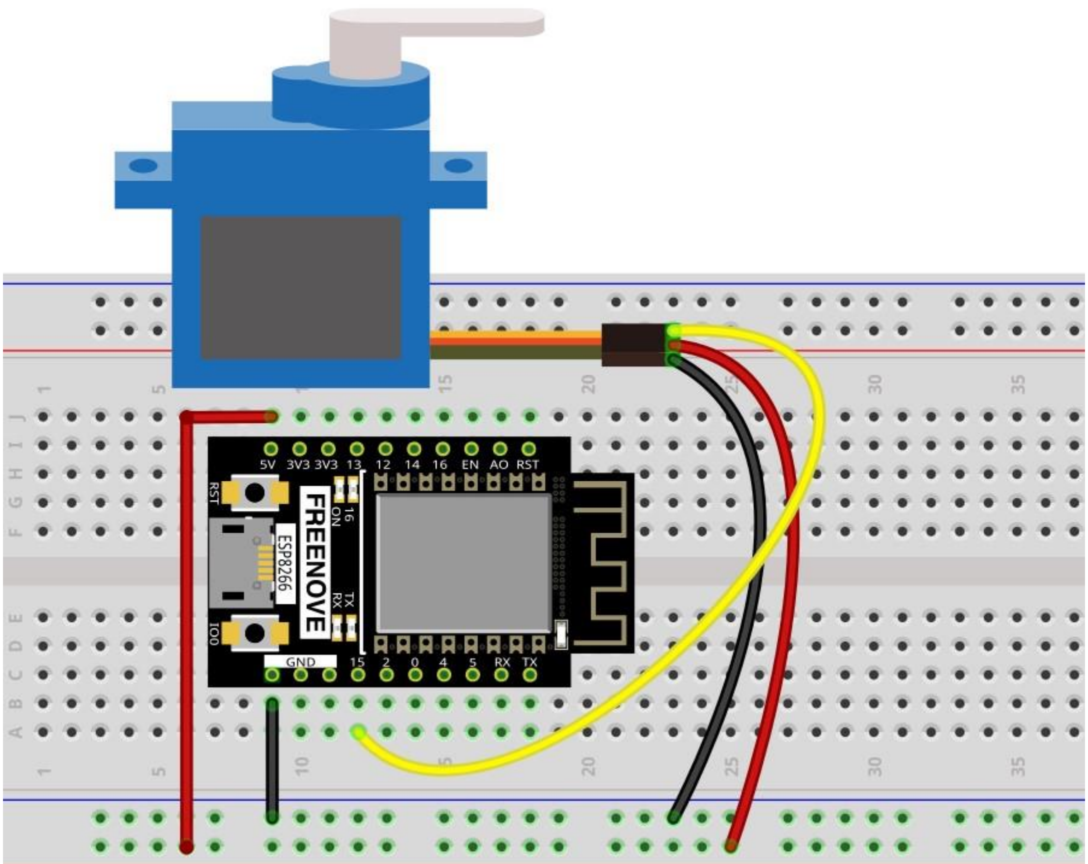
Circuit

Use caution when supplying power to the Servo, it should be 5V. Make sure you do not make any errors when connecting the Servo to the power supply.

Schematic diagram



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

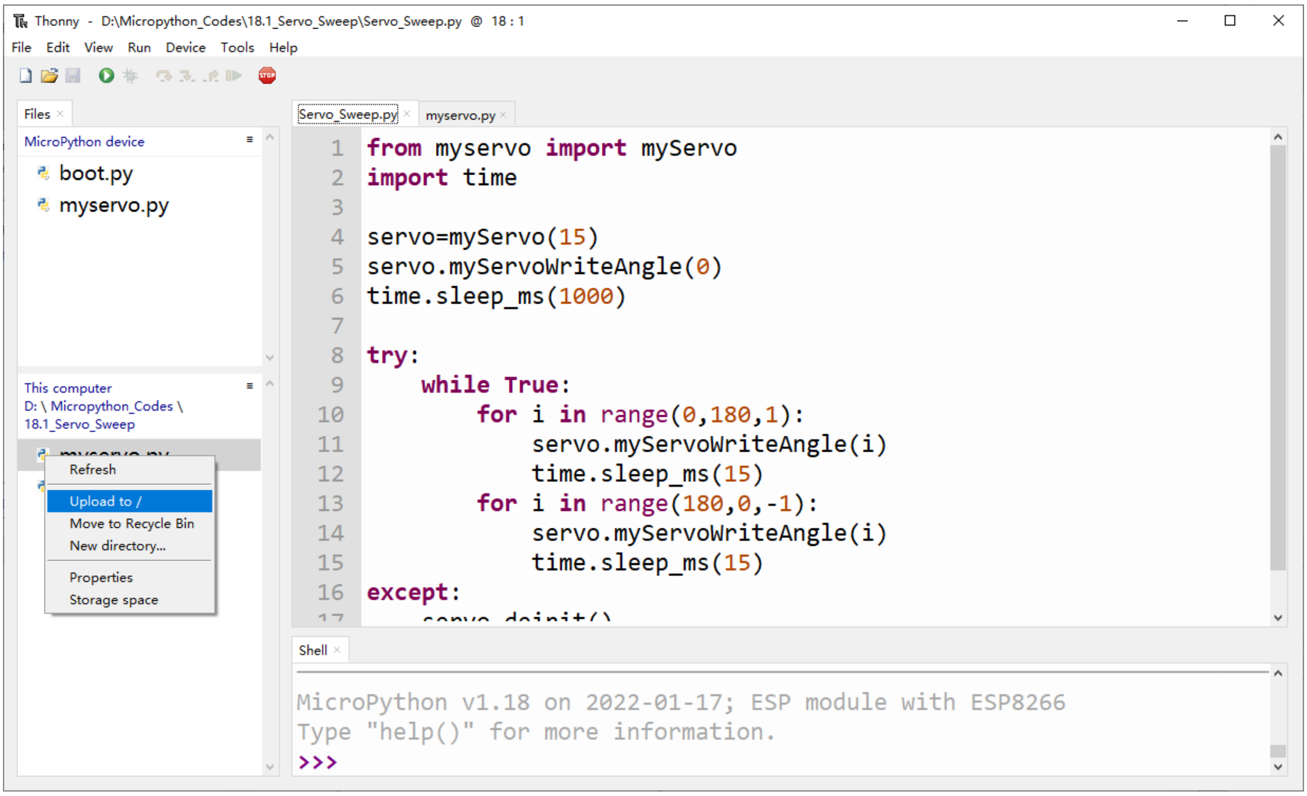


Code

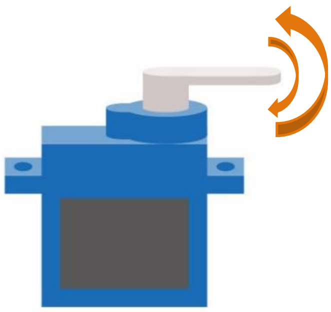
Move the program folder “**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**” to disk(D) in advance with the path of “**D:/Micropython_Codes**”.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “18.1_Servo_Sweep”. Select “myservo.py”, right click your mouse to select “Upload to /”, wait for “myservo.py” to be uploaded to ESP8266 and then double click “Servo_Sweep.py”.

18.1_Servo_Sweep



Click “Run current script”, the Servo will rotate from 0 degrees to 180 degrees and then reverse the direction to make it rotate from 180 degrees to 0 degrees and repeat these actions in an endless loop.



The following is the program code:

```
from myservo import myServo
import time

servo=myServo(15)
servo.myServoWriteAngle(0)
time.sleep_ms(1000)

try:
    while True:
        for i in range(0,180,1):
            servo.myServoWriteAngle(i)
            time.sleep_ms(15)
        for i in range(180,0,-1):
            servo.myServoWriteAngle(i)
            time.sleep_ms(15)
except:
    servo.deinit()
```

Import myservo module.

```
from myservo import myServo
```

Initialize pins of the servo and set the starting point of the servo to 0 degree.

```
servo=myServo(15)
servo.myServoWriteAngle(0)
time.sleep_ms(1000)
```

Control the servo to rotate to a specified angle within the range of 0-180 degrees.

```
servo.myServoWriteAngle(i)
```

Use two for loops. The first one controls the servo to rotate from 0 degree to 180 degrees while the other controls it to rotate back from 180 degrees to 0 degree.

```
        for i in range(0,180,1):
            servo.myServoWriteAngle(i)
            time.sleep_ms(15)
        for i in range(180,0,-1):
            servo.myServoWriteAngle(i)
            time.sleep_ms(15)
```

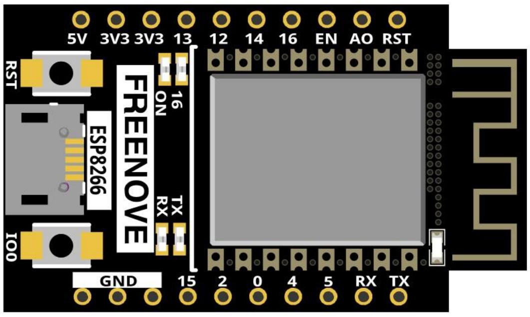
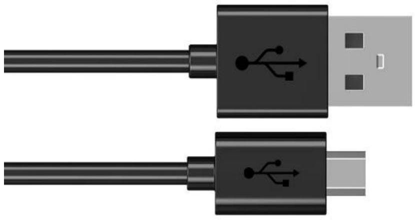
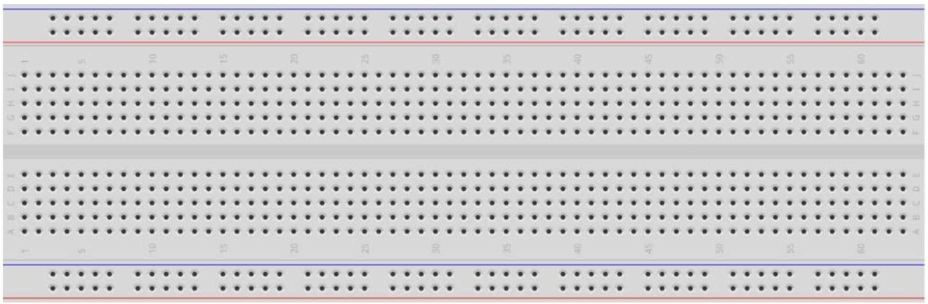
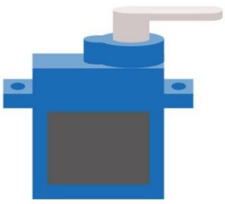
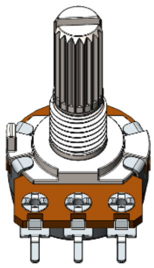
Reference

class myServo
Before each use of **myServo**, please make sure myservo.py has been uploaded to “/” of ESP8266, and then add the statement “**from myservo import myServo**” to the top of the python file. **myServo ()**: The object that controls the servo, with the default pin Pin(15), default frequency 50Hz and default duty cycle 512. **myServoWriteDuty(duty)**: The function that writes duty cycle to control the servo. **duty**: Range from 26 to 128, with 26 corresponding to the servo's 0 degree and 128 corresponding to 180 degrees. **myServoWriteAngle(pos)**: Function that writes angle value to control the servo. **pos**: Ranging from 0-180, corresponding the 0-180 degrees of the servo. **myServoWriteTime(us)**: Writes time to control the servo. **us**: Range from 500-2500, with 500 corresponding to the servo's 0 degree and 2500 corresponding to 180 degrees.

Project 18.2 Servo Knop

Use a potentiometer to control the servo motor to rotate at any angle.

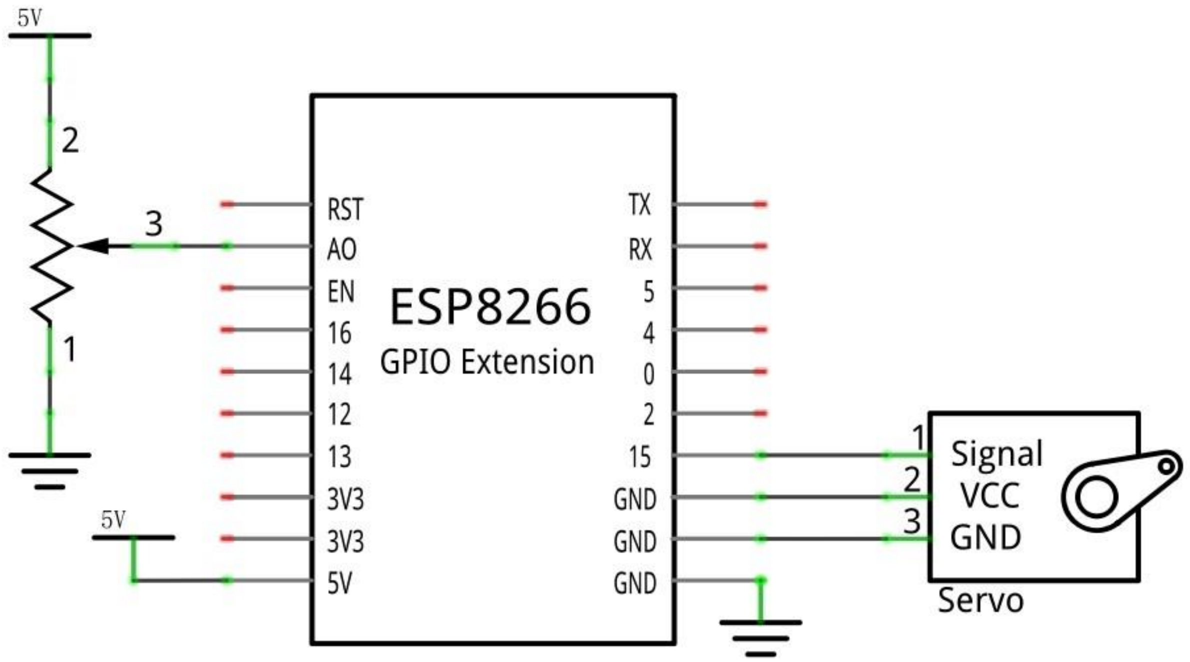
Component List

ESP8266 x1		USB cable
		
Breadboard x1		
		
Servo x1	Jumper wire M/M x10	Rotary potentiometer x1
		

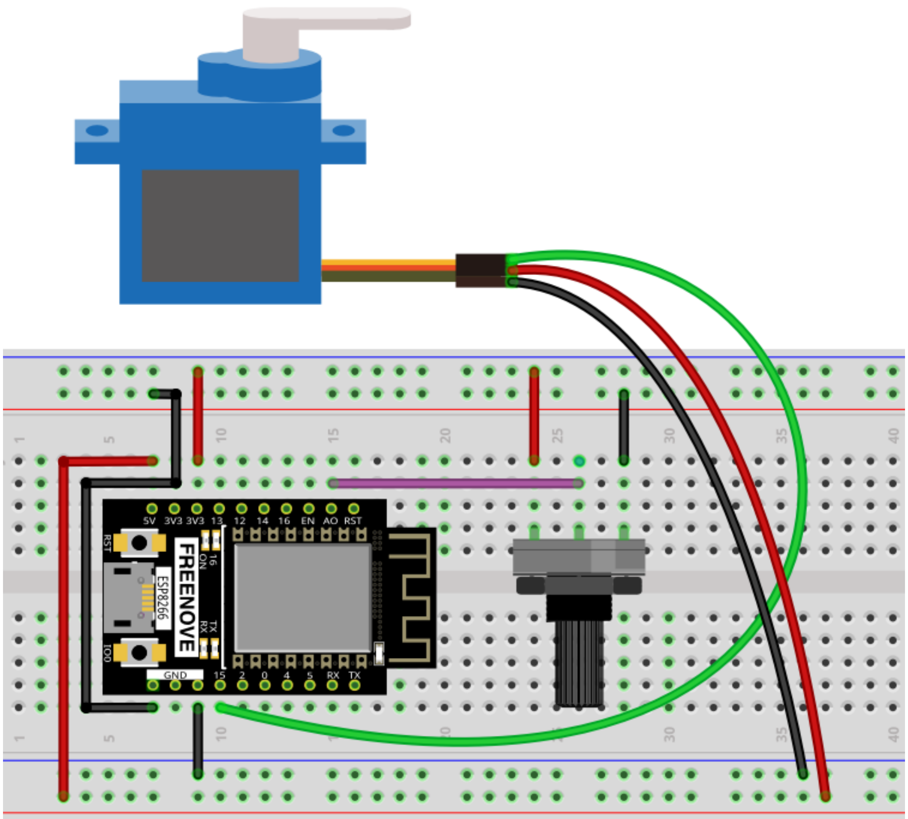
Circuit

Use caution when supplying power to the Servo, it should be 5V. Make sure you do not make any errors when connecting the Servo to the power supply.

Schematic diagram



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

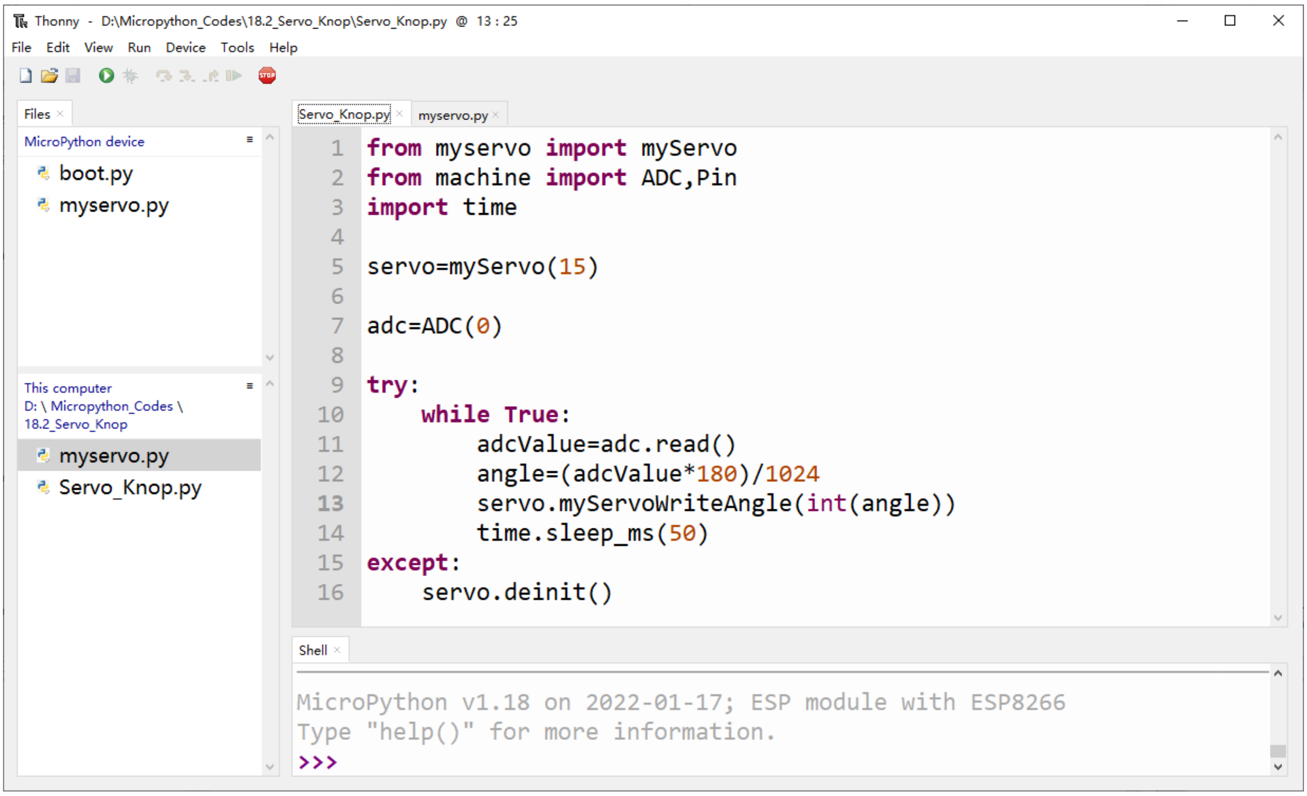


Code

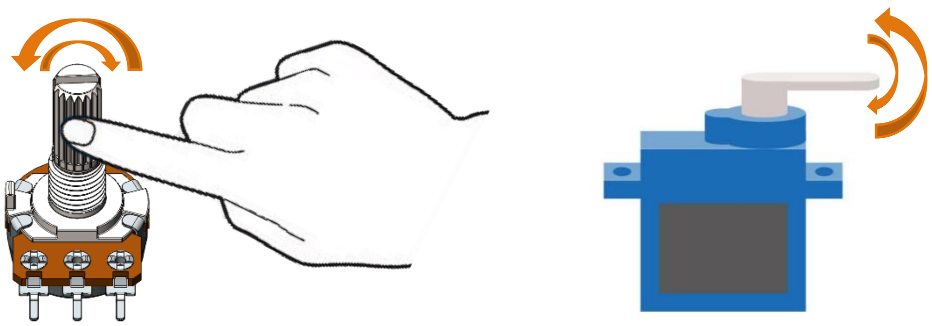
Move the program folder “**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**” to disk(D) in advance with the path of “**D:/Micropython_Codes**”.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “18.2_Servo_Knop”. Select “myservo.py”, right click your mouse to select “Upload to /”, wait for “myservo.py” to be uploaded to ESP8266 and then double click “Servo_Knop.py”.

18.2_Servo_Knop



Click “Run current script”, twist the potentiometer back and forth, and the servo motor rotates accordingly.



The following is the program code:

```
from myservo import myServo
from machine import ADC,Pin
import time

servo=myServo(15)

adc=ADC(0)

try:
    while True:
        adcValue=adc2.read()
        angle=(adcValue*180)/1024
        servo.myServoWriteAngle(int(angle))
        time.sleep_ms(50)
except:
    servo.deinit()
```

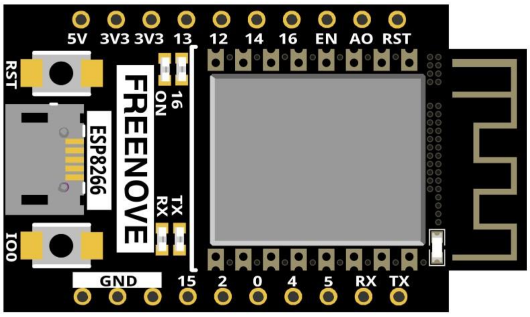
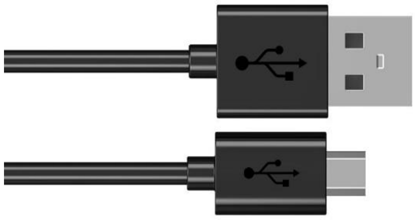
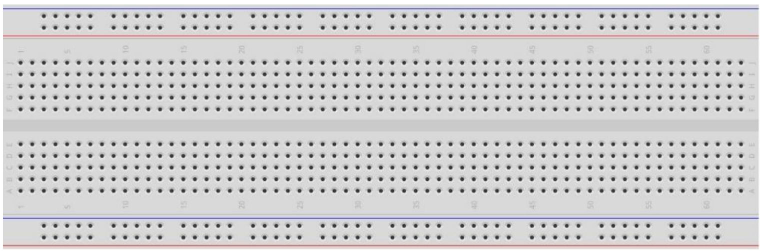
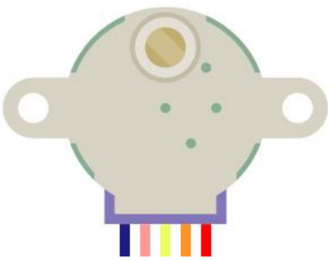
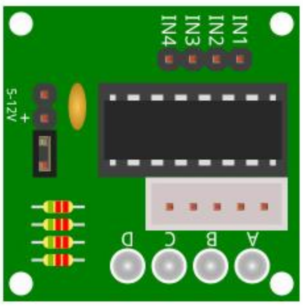
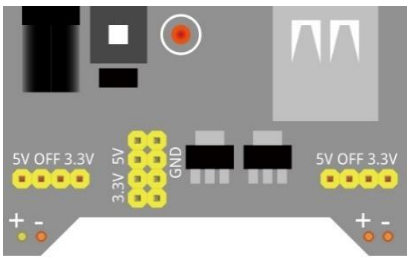
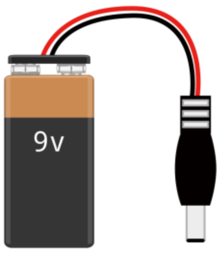
In this project, we will use ADC0 of ESP8266 to read the ADC value of the rotary potentiometer and then convert it to the angle value required by the servo and control the servo to rotate to the corresponding angle.

Chapter 19 Stepper Motor

In this project, we will learn how to drive a Stepper Motor, and understand its working principle.

Project 19.1 Stepping Motor

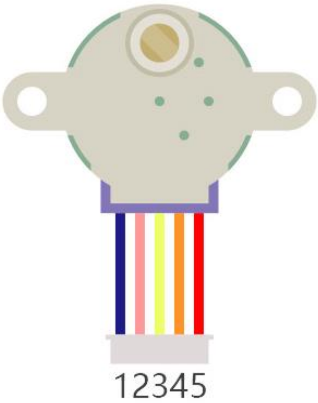
Component List

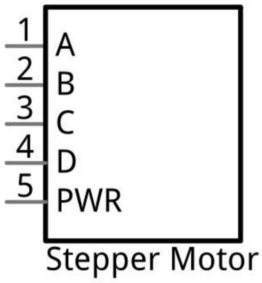
<table>
<tr><td colspan="3">ESP8266 x1
</td><td colspan="3">USB cable
</td></tr>
<tr><td colspan="6">Breadboard x1
</td></tr>
<tr><td colspan="2">Stepping Motor x1
</td><td colspan="2">ULN2003 Stepping motorDriver x1
</td><td colspan="2">Jumper wire F/M x8
</td></tr>
<tr><td colspan="3">Breadboard Power module x1
</td><td colspan="3">9V battery (prepared by yourself) & battery line
</td></tr>
</table>

Component knowledge

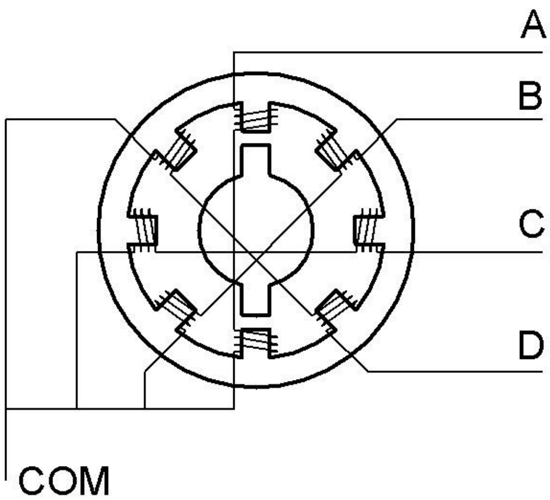
Stepper Motor x1

Stepper Motors are an open-loop control device, which converts an electronic pulse signal into angular displacement or linear displacement. In a non-overload condition, the speed of the motor and the location of the stops depend only on the pulse signal frequency as well as the number of pulses and are not affected by changes in load as with a DC Motor. A small Four-Phase Deceleration Stepper Motor is shown here:



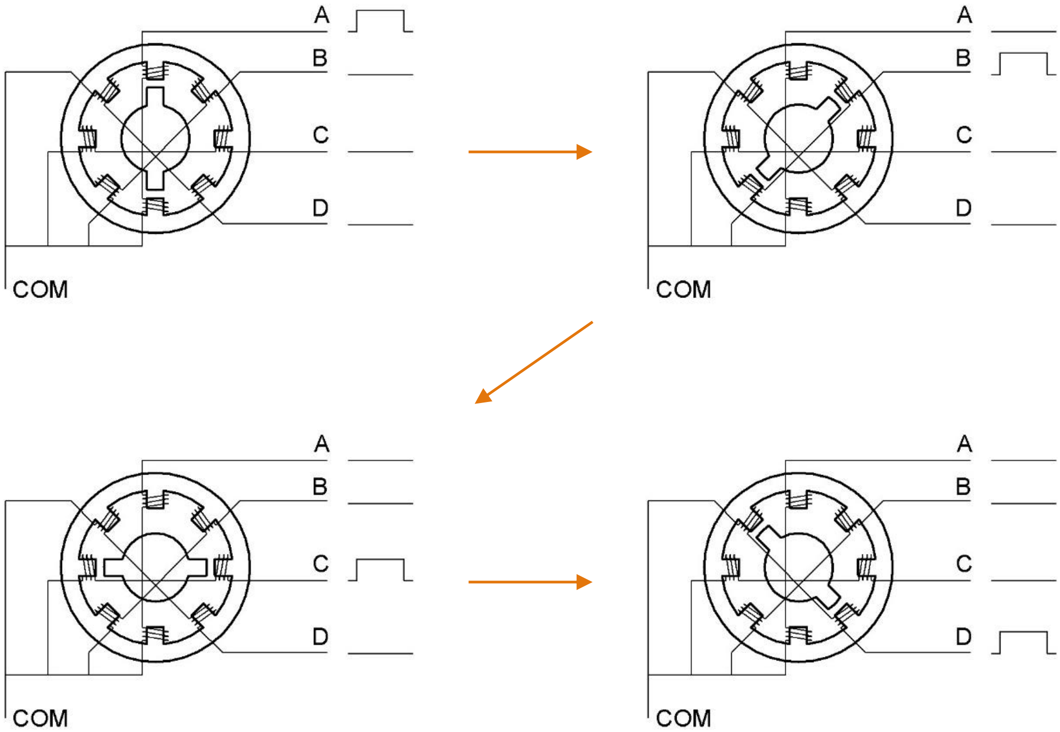


The electronic schematic diagram of a Four-Phase Stepper Motor is shown below:



The outside case or housing of the Stepper Motor is the Stator and inside the Stator is the Rotor. There are a specific number of individual coils, usually an integer multiple of the number of phases the motor has, when the Stator is powered ON, an electromagnetic field will be formed to attract a corresponding convex diagonal groove or indentation in the Rotor's surface. The Rotor is usually made of iron or a permanent magnet. Therefore, the Stepper Motor can be driven by powering the coils on the Stator in an ordered sequence (producing a series of "steps" or stepped movements).

A common driving process is as follows:



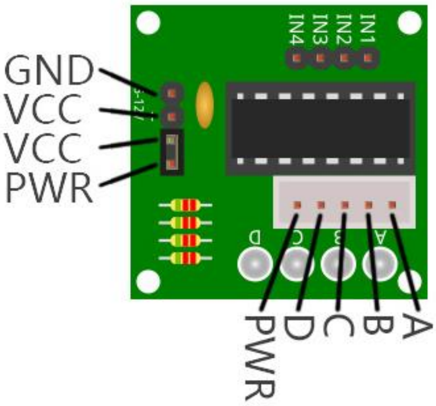
In the course above, the stepping motor rotates a certain angle once, which is called a step. By controlling the number of rotation steps, you can control the stepping motor rotation angle. By controlling the time between two steps, you can control the stepping motor rotation speed. When rotating clockwise, the order of coil powered on is: A→B→C→D→A→…… . And the rotor will rotate in accordance with the order, step by step down, called four steps four pats. If the coils is powered on in the reverse order, D→C→B→A→D→… , the rotor will rotate in anti-clockwise direction.

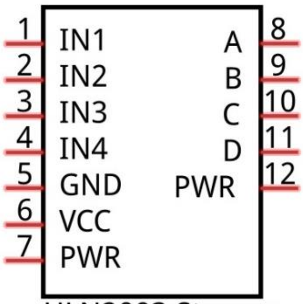
There are other methods to control Stepper Motors, such as: connect A phase, then connect A B phase, the stator will be located in the center of A B, which is called a half-step. This method can improve the stability of the Stepper Motor and reduces noise. The sequence of powering the coils looks like this: A→ AB→B →BC→C →CD→D→DA→A→……, the rotor will rotate in accordance to this sequence at a half-step at a time, called four-steps, eight-part. Conversely, if the coils are powered ON in the reverse order the Stepper Motor will rotate in the opposite direction.

The stator in the Stepper Motor we have supplied has 32 magnetic poles. Therefore, to complete one full revolution requires 32 full steps. The rotor (or output shaft) of the Stepper Motor is connected to a speed reduction set of gears and the reduction ratio is 1:64. Therefore, the final output shaft (exiting the Stepper Motor's housing) requires 32 X 64 = 2048 steps to make one full revolution.

ULN2003 Stepping motor driver

A ULN2003 Stepper Motor Driver is used to convert weak signals into more powerful control signals in order to drive the Stepper Motor. In the illustration below, the input signal IN1-IN4 corresponds to the output signal A-D, and 4 LEDs are integrated into the board to indicate the state of these signals. The PWR interface can be used as a power supply for the Stepper Motor. By default, PWR and VCC are connected.



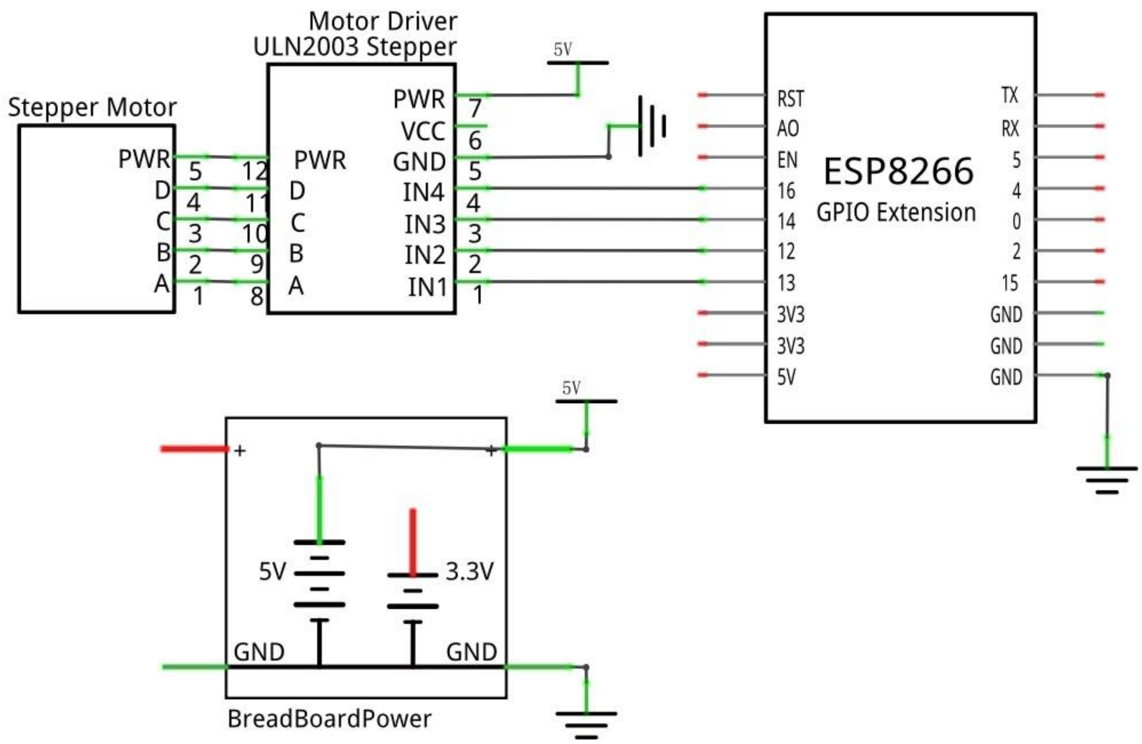


ULN2003 Stepper Motor Driver

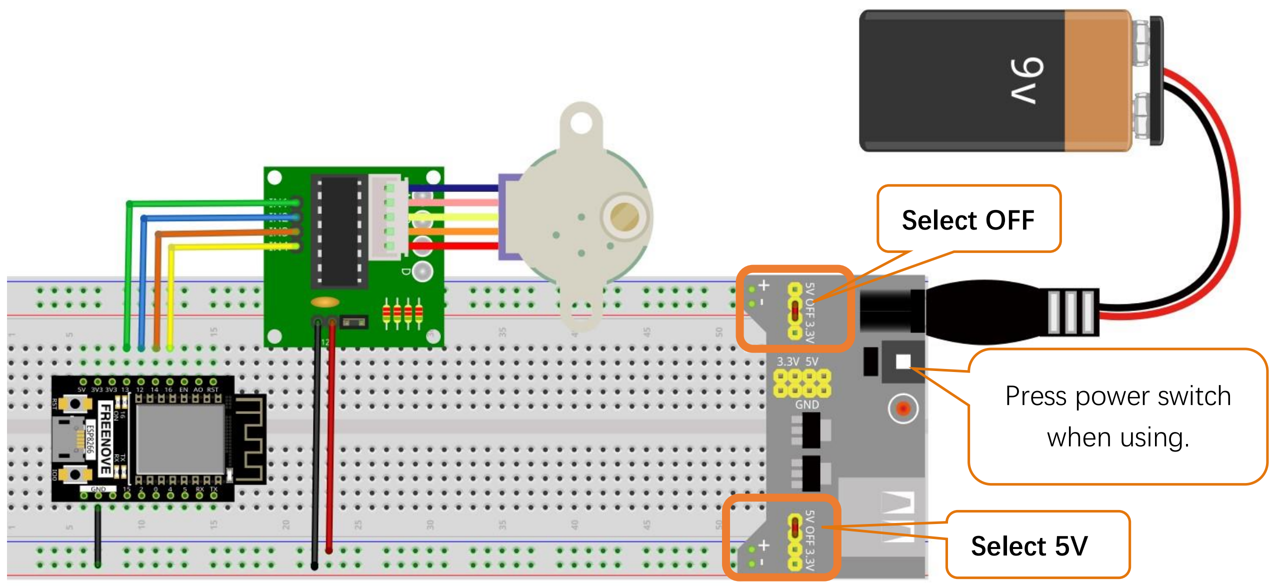
Circuit

When building the circuit, note that rated voltage of the Stepper Motor is 5V, and we need to use the Breadboard power supply independently. Additionally, the Breadboard power supply needs to share Ground with ESP8266.

Schematic diagram



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



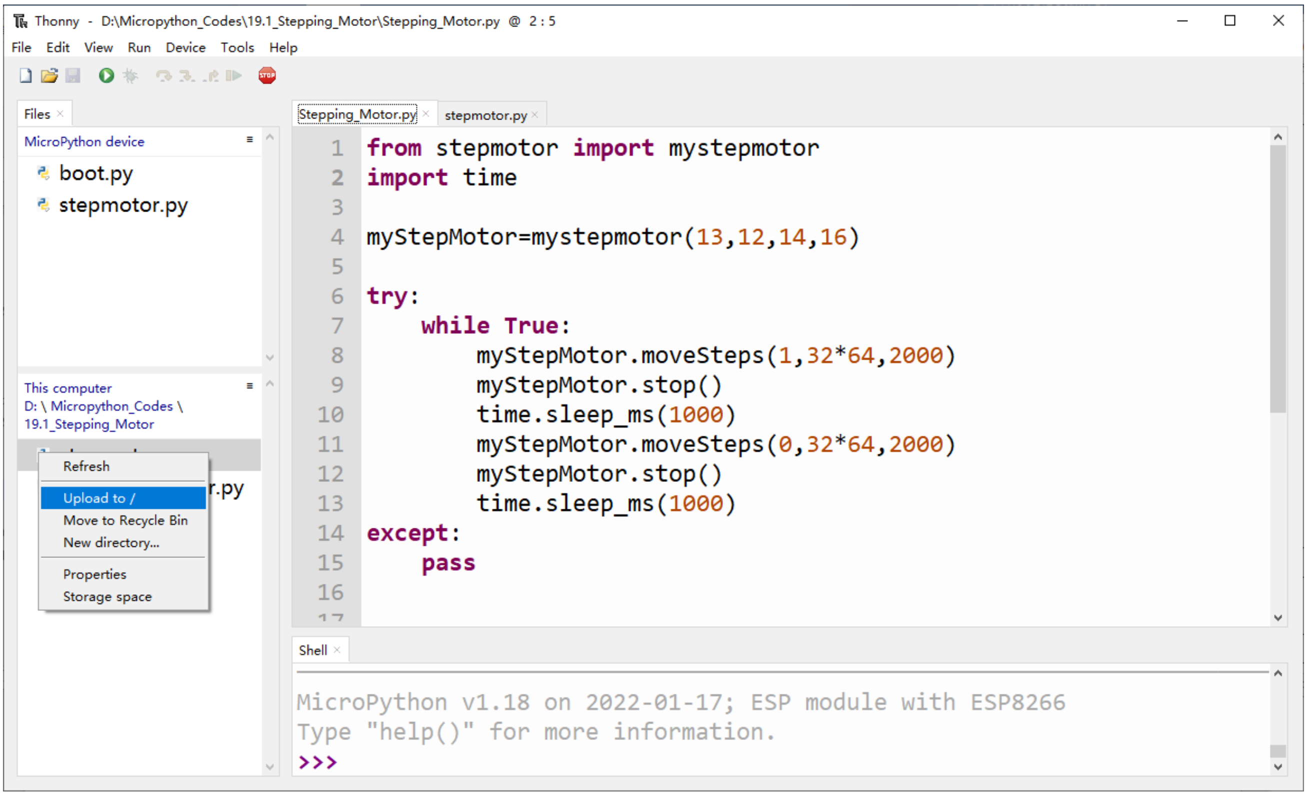
Code

This code uses the four-step, four-part mode to drive the Stepper Motor in the clockwise and anticlockwise directions.

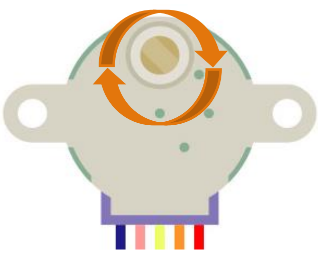
Move the program folder "**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

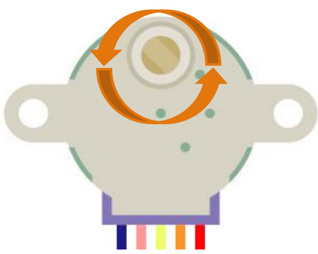
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "19.1_Stepping_Motor". Select "stepmotor.py", right click your mouse to select "Upload to /", wait for "myservo.py" to be uploaded to ESP8266 and then double click "Stepping_Motor.py".

19.1_Stepping_Motor



Click "Run current script", the stepper motor will rotate 360° clockwise and stop for 1s, and then rotate 360° anticlockwise and stop for 1s. And it will repeat this action in an endless loop.





The following is the program code:

```
from stepmotor import mystepmotor
import time

myStepMotor=mystepmotor(13,12,14,16)

try:
    while True:
        myStepMotor.moveSteps(1,32*64,2000)
        myStepMotor.stop()
        time.sleep_ms(1000)
        myStepMotor.moveSteps(0,32*64,2000)
        myStepMotor.stop()
        time.sleep_ms(1000)
except:
    pass
```

Import time and stepmotor modules.

```
from stepmotor import mystepmotor
import time
```

In this project, we define four pins to drive the stepper motor.

```
myStepMotor=mystepmotor(13,12,14,16)
```

Call the function moveSteps to control the stepper motor to rotate for 360°and then call function stop() to stop it.

```
        myStepMotor.moveSteps(1,32*64,2000)
        myStepMotor.stop()
```

Repeatedly control the stepmotor to rotate 360° clockwise and then rotate 360° anti-clockwise.

```
    while True:
        myStepMotor.moveSteps(1,32*64,2000)
        myStepMotor.stop()
        time.sleep_ms(1000)
        myStepMotor.moveSteps(0,32*64,2000)
        myStepMotor.stop()
        time.sleep_ms(1000)
```

Reference

class myServo

Before each use of the object **mystepmotor**, please make sure that stepmotor.py has been uploaded to "/" of ESP8266, and then add the statement "**from stepmotor import mystepmotor**" to the top of the python file.
mystepmotor(): The object to control the stepper motor. The default control pins are Pin(13), Pin(12), Pin(14) and Pin(16).
moveSteps(direction,steps,us): Control the stepper motor to rotate a specified number of steps.
    direction: The rotation direction of stepper motor.
    Steps: Rotation steps of the stepper motor.
    us: Time required by the stepper motor to rotate for one step.
moveAround(direction,turns,us): Control the stepper motor to rotate a specific number of turns.
    Turns: Number of turns that the stepper motor rotates.
moveAngle(direction,angles,us): Control the stepper motor to rotate a specific angle.
    Angles: Rotation angles that the stepper motor rotates.
stop(): Stop the stepper motor.

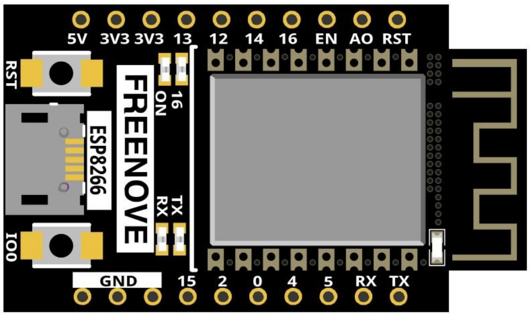
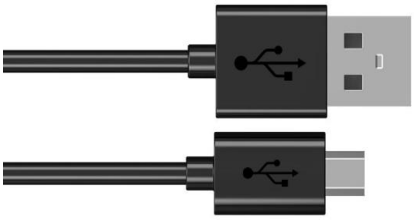
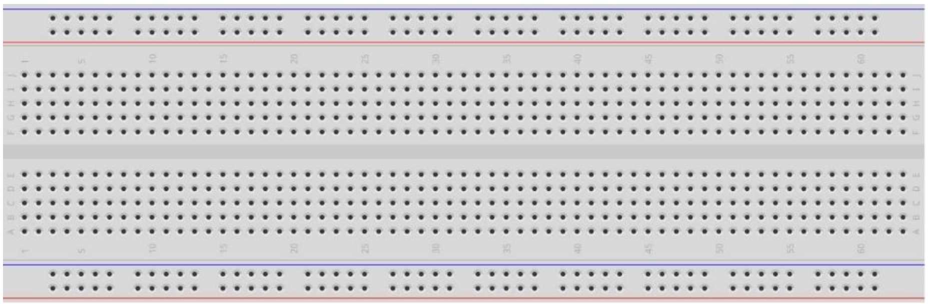
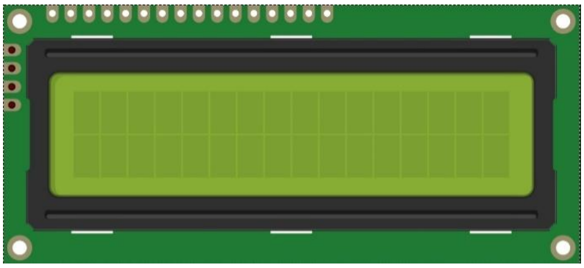
Chapter 20 LCD1602

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

Project 20.1 LCD1602

In this section we learn how to use lcd1602 to display something.

Component List

ESP8266 x1 	USB cable 
Breadboard x1 	
LCD1602 Module x1 	Jumper wire F/M x6 

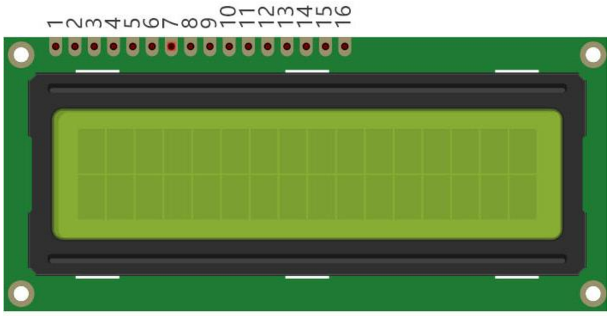
Component knowledge

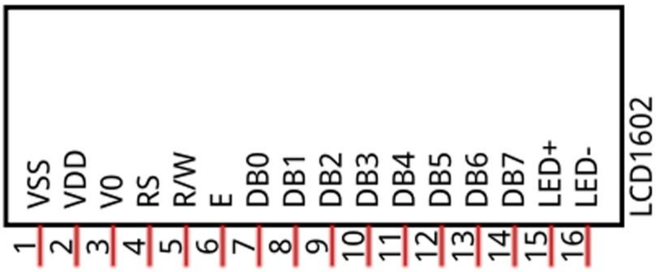
I2C communication

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

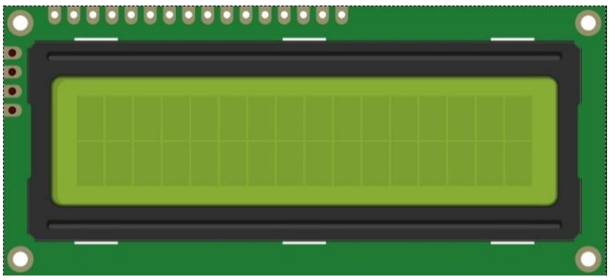
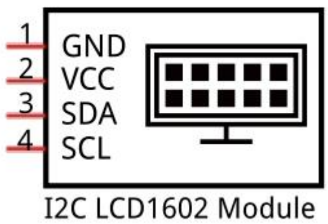
LCD1602 communication'

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





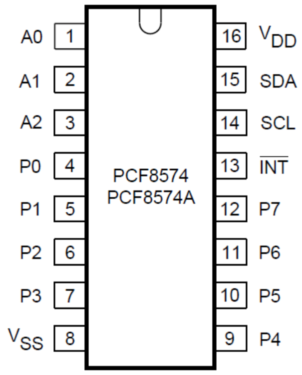
I2C LCD1602 Display Screen integrates an I2C interface, which connects the serial-input & parallel-output module to the LCD1602 Display Screen. This allows us to use only 4 lines to the operate the LCD1602.

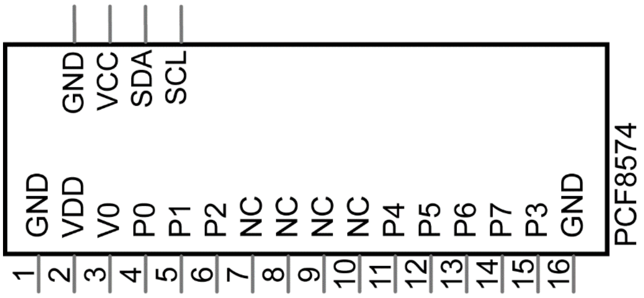
The serial-to-parallel IC chip used in this module is PCF8574T (PCF8574AT), and its default I2C address is 0x27(0x3F).

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

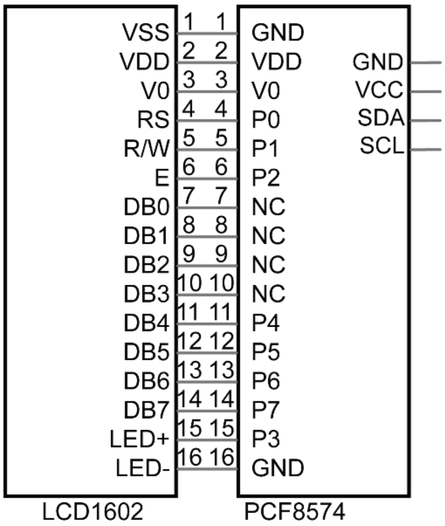
PCF8574 chip pin diagram:



PCF8574 module pin diagram



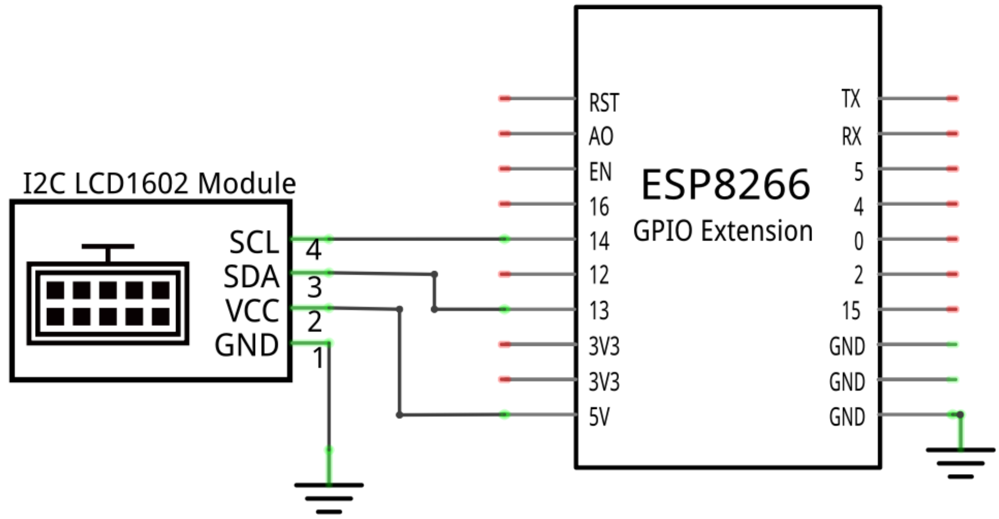
PCF8574 module pin and LCD1602 pin are corresponding to each other and connected with each other:



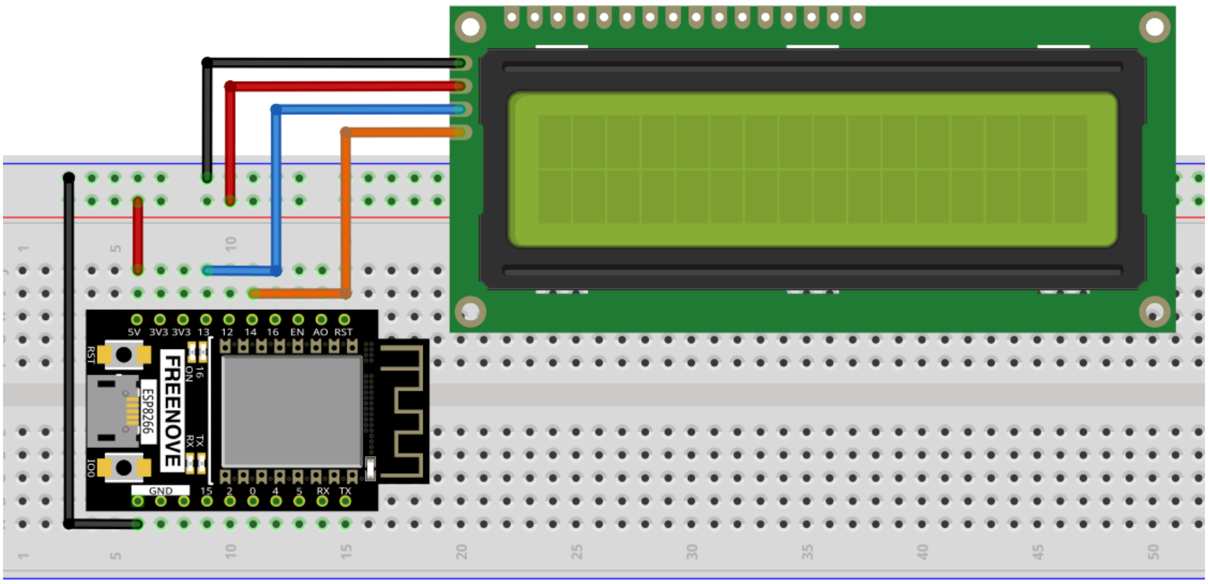
So we only need 4 pins to control the 16 pins of the LCD1602 Display Screen through the I2C interface.
In this project, we will use the I2C LCD1602 to display some static characters and dynamic variables.

Circuit

Schematic diagram



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

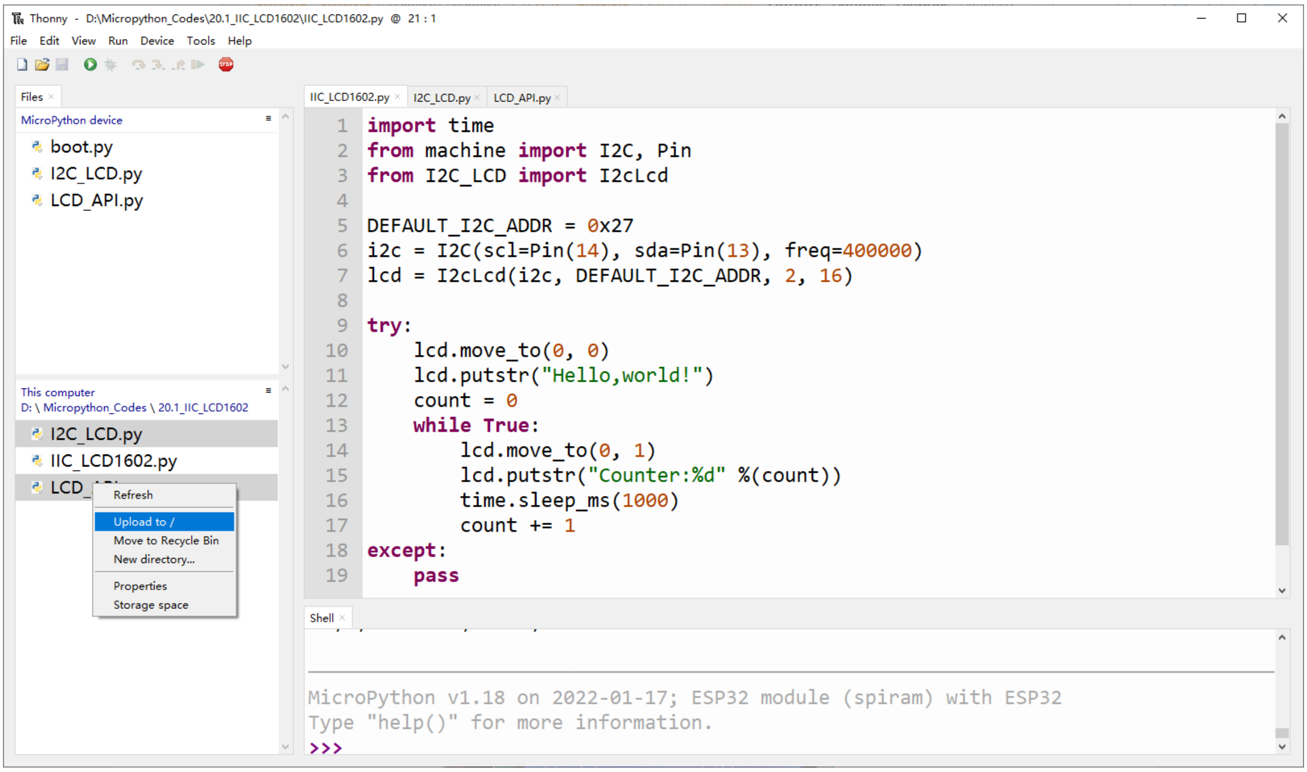


Code

Move the program folder "**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

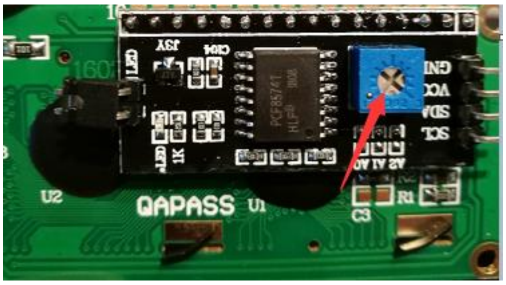
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "20.1_I2C_LCD1602". Select "I2C_LCD.py"and "LCD_API.py", right click your mouse to select "Upload to /", wait for "I2C_LCD.py" and "LCD_API.py" to be uploaded to ESP8266 and then double click "I2C_LCD1602.py".

20.1_I2C_LCD1602



Click "Run current script" and LCD1602 displays some characters.

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



The following is the program code:

```
import time
from machine import I2C, Pin
from I2C_LCD import I2cLcd

DEFAULT_I2C_ADDR = 0x27
i2c = I2C(scl=Pin(14), sda=Pin(13), freq=400000)
lcd = I2cLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)

try:
    lcd.move_to(0, 0)
    lcd.putstr("Hello,world!")
    count = 0
    while True:
        lcd.move_to(0, 1)
        lcd.putstr("Counter:%d" %(count))
        time.sleep_ms(1000)
        count += 1
except:
    pass
```

Import time, I2C and I2C_LCD modules.

```
import time
from machine import I2C, Pin
from I2C_LCD import I2cLcd
```

Instantiate the I2C LCD1602 screen. It should be noted here that if your LCD driver chip uses PCF8574T, set the I2C address to 0x27, and if uses PCF8574AT, set the I2C address to 0x3F.

```
DEFAULT_I2C_ADDR = 0x27
```

Initialize I2C pins and associate them with I2CLCD module, and then set the number of rows and columns for LCD1602.

```
i2c = I2C(scl=Pin(14), sda=Pin(13), freq=400000)
lcd = I2cLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)
```

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

```
    lcd.move_to(0, 0)
    lcd.putstr("Hello,world!")
```

The second line of LCD1602 continuously prints the number of seconds after the ESP8266 program runs.

```
    while True:
        lcd.move_to(0, 1)
        lcd.putstr("Counter:%d" %(count))
        time.sleep_ms(1000)
        count += 1
```

Reference

Class I2cLcd

Before each use of the object **I2cLcd**, please make sure that **I2C_LCD.py** and **LCD_API.py** have been uploaded to “/” of ESP8266, and then add the statement “**from I2C_LCD import I2cLcd**” to the top of the python file.

clear(): Clear the LCD1602 screen display.

show_cursor(): Show the cursor of LCD1602.

hide_cursor(): Hide the cursor of LCD1602.

blink_cursor_on(): Turn on cursor blinking.

blink_cursor_off(): Turn off cursor blinking.

display_on(): Turn on the display function of LCD1602.

display_off(): Turn on the display function of LCD1602.

backlight_on(): Turn on the backlight of LCD1602.

backlight_off(): Turn on the backlight of LCD1602.

move_to(cursor_x, cursor_y): Move the cursor to a specified position.

  cursor_x: Column cursor_x

  cursor_y: Row cursor_y

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

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

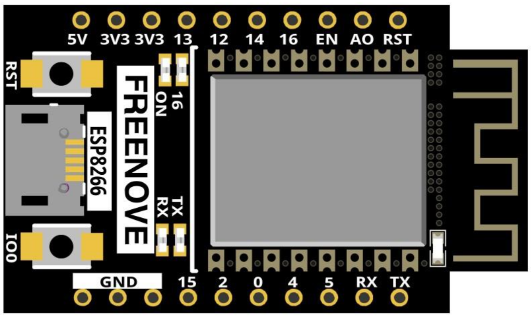
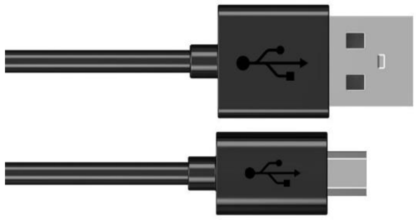
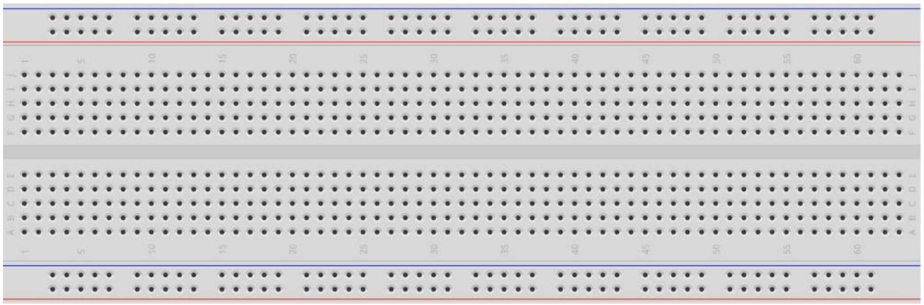
Chapter 21 Ultrasonic Ranging

In this chapter, we learn a module which use ultrasonic to measure distance, HC SR04.

Project 21.1 Ultrasonic Ranging

In this project, we use ultrasonic ranging module to measure distance, and print out the data in the terminal.

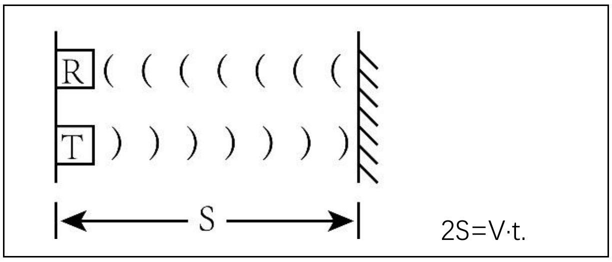
Component List

<table>
<tr><td>ESP8266 x1

</td><td colspan="2">USB cable
</td></tr>
<tr><td colspan="3">Breadboard x1
</td></tr>
<tr><td colspan="2">Jumper wire F/M x6
</td><td>HC SR04 x1

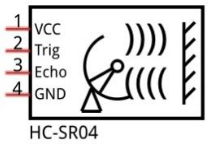
</td></tr>
</table>

Component Knowledge

The Ultrasonic Ranging Module uses the principle that ultrasonic waves will reflect when they encounter any obstacles. This is possible by counting the time interval between when the ultrasonic wave is transmitted to when the ultrasonic wave reflects back after encountering an obstacle. Time interval counting will end after an ultrasonic wave is received, and the time difference (delta) is the total time of the ultrasonic wave's journey from being transmitted to being received. Because the speed of sound in air is a constant, and is about v=340m/s, we can calculate the distance between the Ultrasonic Ranging Module and the obstacle: s=vt/2.



The HC-SR04 Ultrasonic Ranging Module integrates both an ultrasonic transmitter and a receiver. The transmitter is used to convert electrical signals (electrical energy) into high frequency (beyond human hearing) sound waves (mechanical energy) and the function of the receiver is opposite of this. The picture and the diagram of the HC SR04 Ultrasonic Ranging Module are shown below:



Pin description:

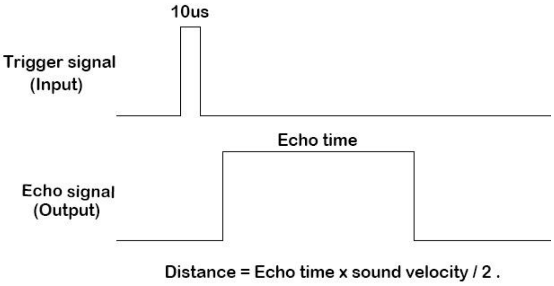
Pin	Description
VCC	power supply pin
Trig	trigger pin
Echo	Echo pin
GND	GND

Technical specs:

Working voltage: 5V Working current: 12mA

Minimum measured distance: 2cm Maximum measured distance: 200cm

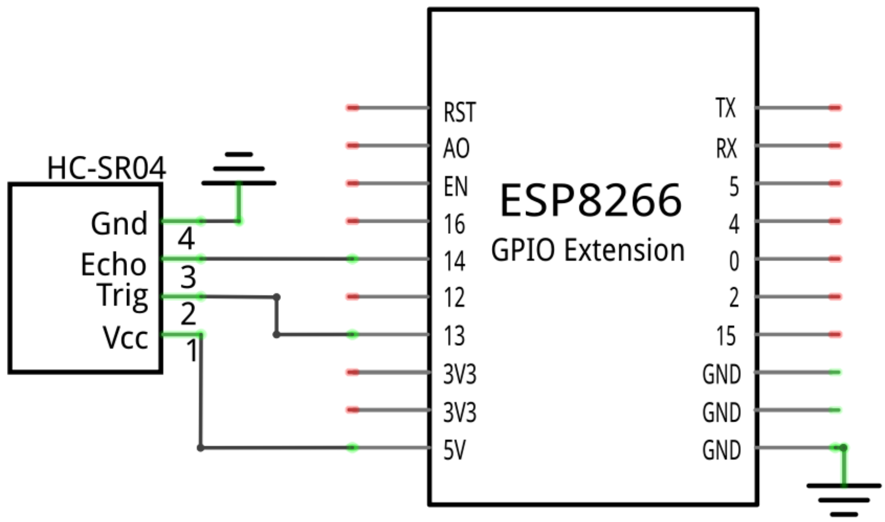
Instructions for Use: output a high-level pulse in Trig pin lasting for least 10us, the module begins to transmit ultrasonic waves. At the same time, the Echo pin is pulled up. When the module receives the returned ultrasonic waves from encountering an obstacle, the Echo pin will be pulled down. The duration of high level in the Echo pin is the total time of the ultrasonic wave from transmitting to receiving, s=vt/2.



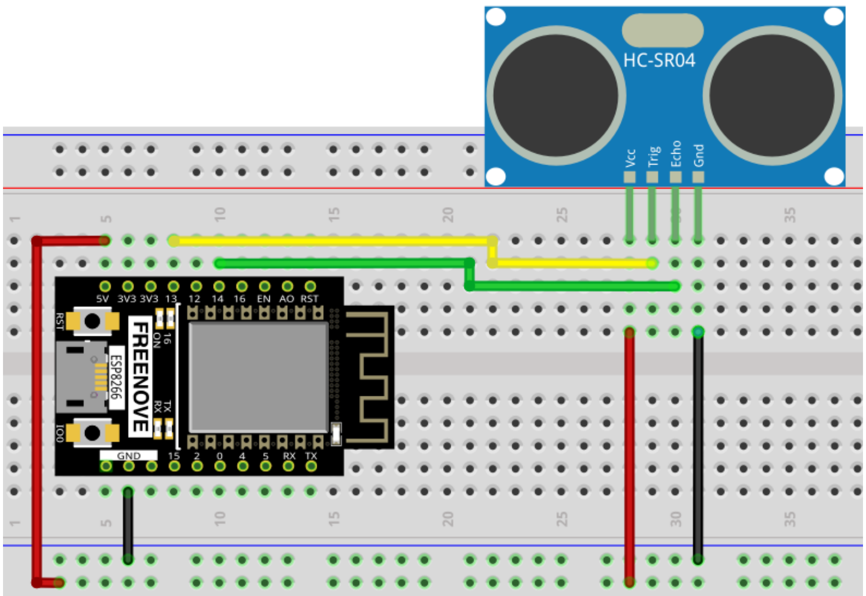
Circuit

Note that the voltage of ultrasonic module is 5V in the circuit.

Schematic diagram



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

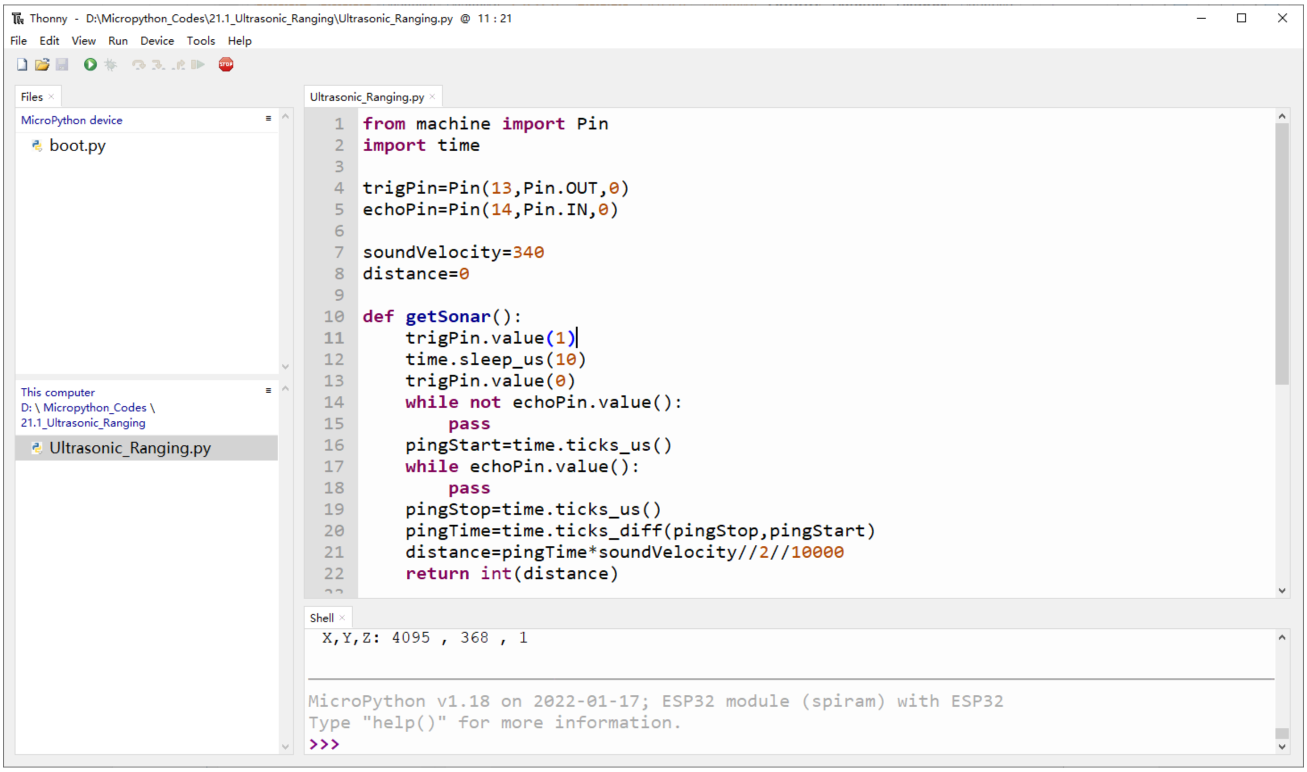


Code

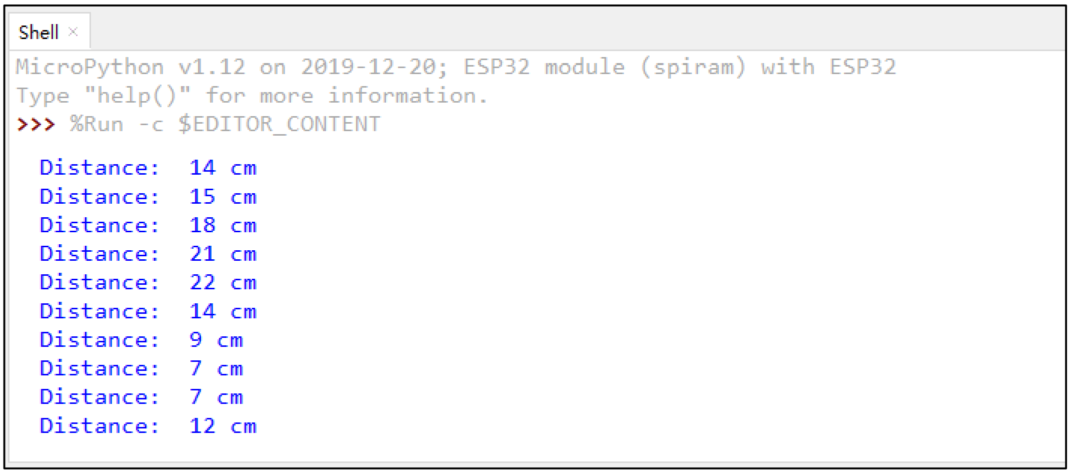
Move the program folder "**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "21.1_Ultrasonic_Ranging" and double click "Ultrasonic_Ranging.py".

21.1_Ultrasonic_Ranging



Click “Run current script”, you can use it to measure the distance between the ultrasonic module and the object. As shown in the following figure:



The following is the program code:

```
from machine import Pin
import time

trigPin=Pin(13,Pin.OUT,0)
echoPin=Pin(14,Pin.IN,0)

soundVelocity=340
distance=0

def getSonar():
    trigPin.value(1)
    time.sleep_us(10)
    trigPin.value(0)
    while not echoPin.value():
        pass
    pingStart=time.ticks_us()
    while echoPin.value():
        pass
    pingStop=time.ticks_us()
    pingTime=time.ticks_diff(pingStop,pingStart)
    distance=pingTime*soundVelocity//2//10000
    return int(distance)

time.sleep_ms(2000)
while True:
    time.sleep_ms(500)
    print(‘Distance: ’,getSonar(),‘cm’)
```

Define the control pins of the ultrasonic ranging module.

```
trigPin=Pin(13,Pin.OUT,0)
echoPin=Pin(14,Pin.IN,0)
```

Set the speed of sound.

```
soundVelocity=340
distance=0
```

Subfunction getSonar() is used to start the Ultrasonic Module to begin measurements, and return the measured distance in centimeters. In this function, first let trigPin send 10us high level to start the Ultrasonic Module. Then use pulseIn() to read the Ultrasonic Module and return the duration time of high level. Finally, the measured distance according to the time is calculated.

```
def getSonar():
    trigPin.value(1)
    time.sleep_us(10)
    trigPin.value(0)
    while not echoPin.value():
        pass
```

```
    pingStart=time.ticks_us()
    while echoPin.value():
        pass
    pingStop=time.ticks_us()
    pingTime=time.ticks_diff(pingStop,pingStart)
    distance=pingTime*soundVelocity//2//10000
    return int(distance)
```

Delay for 2 seconds and wait for the ultrasonic module to stabilize. Print data obtained from ultrasonic module every 500 milliseconds

```
time.sleep_ms(2000)
while True:
    time.sleep_ms(500)
    print(‘Distance: ’,getSonar(),‘cm’)
```

Project 21.2 Ultrasonic Ranging

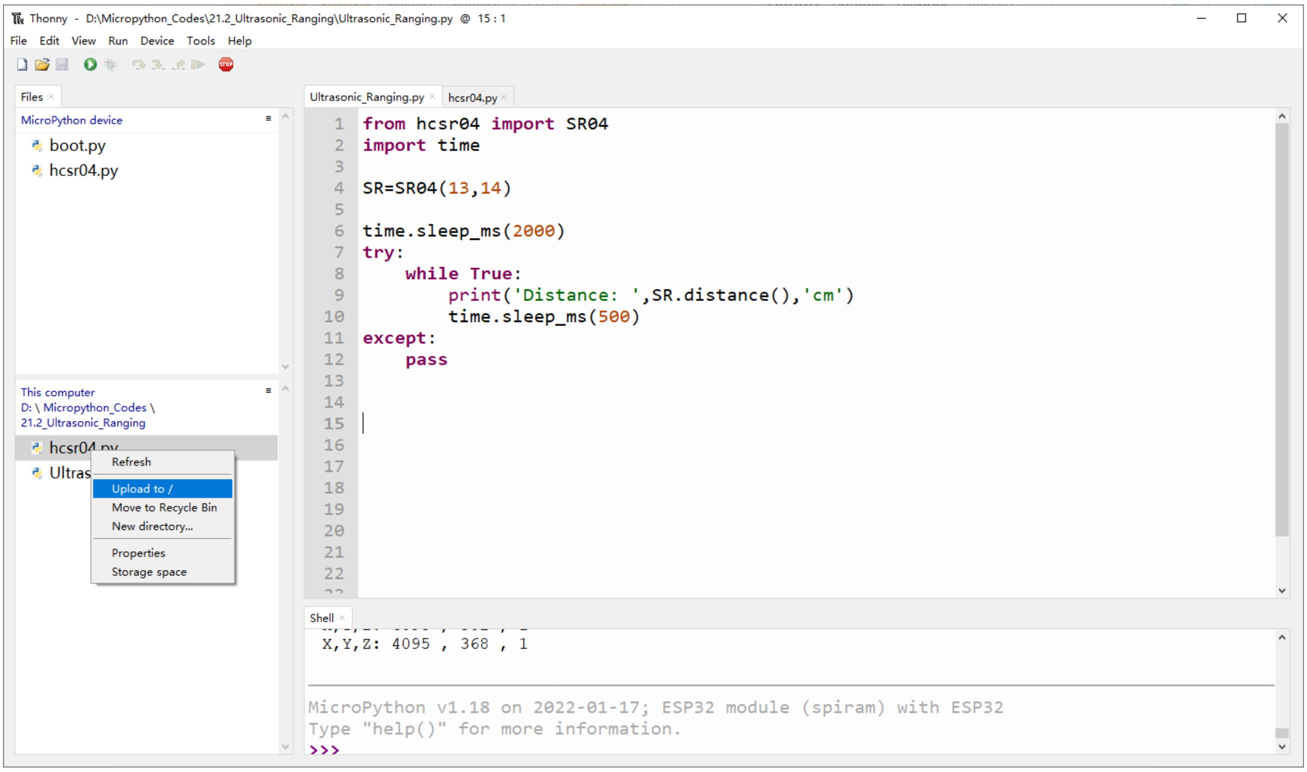
Component List and Circuit

Component List and Circuit are the same as the previous section.

Code

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "21.2_Ultrasonic_Ranging". Select "hcsr04.py", right click your mouse to select "Upload to /", wait for "hcsr04.py" to be uploaded to ESP8266 and then double click "Ultrasonic_Ranging.py".

21.2_Ultrasonic_Ranging



Click "Run current script". Use the ultrasonic module to measure distance. As shown in the following figure:

```
Shell
Distance:  6.647 cm
Distance:  5.151 cm
Distance:  5.151 cm
Distance:  6.052 cm
Distance:  7.225 cm
Distance:  7.531 cm
Distance:  8.721 cm
Distance:  12.121 cm
Distance:  11.798 cm
Distance:  13.6 cm
Distance:  14.467 cm
Distance:  16.116 cm
Distance:  17.442 cm
Distance:  19.652 cm
Distance:  22.015 cm
```

The following is the program code:

```
from hcsr04 import SR04
import time

SR=SR04(13,14)

time.sleep_ms(2000)
try:
    while True:
        print('Distance: ',SR.distance(),'cm')
        time.sleep_ms(500)
except:
    pass
```

Import hcsr04 module.

```
from hcsr04 import SR04
```

Define an ultrasonic object and associate with the pins.

```
SR=SR04(13,14)
```

Obtain the distance data returned from the ultrasonic ranging module.

```
SR.distance()
```

Obtain the ultrasonic data every 500 milliseconds and print them out in "Shell".

```
    while True:
        print('Distance: ',SR.distance(),'cm')
        time.sleep_ms(500)
```

Reference

Class hcsr04
Before each use of object **SR04**, please add the statement "**from hcsr04 import SR04**" to the top of python file. **SR04()**: Object of ultrasonic module. By default, trig pin is Pin(13) and echo pinis Pin(14). **distanceCM()**: Obtain the distance from the ultrasonic to the measured object with the data type being int type, and the unit being cm. **distanceMM()**: Obtain the distance from the ultrasonic to the measured object with the data type being int type, and the unit being mm. **distance()**: Obtain the distance from the ultrasonic to the measured object with the data type being float type, and the unit being cm.

Chapter 22 Matrix Keypad

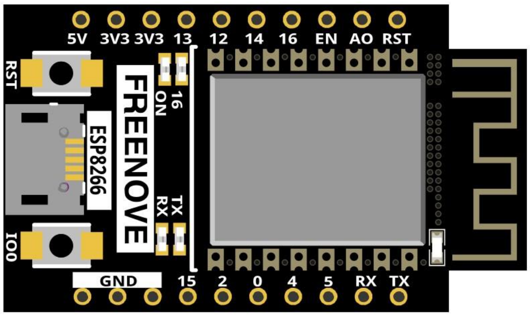
Earlier we learned about a single Push Button Switch. In this chapter, we will learn about Matrix Keyboards, which integrates a number of Push Button Switches as Keys for the purposes of Input.

Project 22.1 Matrix Keypad

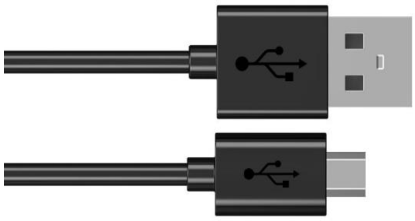
In this project, we will attempt to get every key code on the Matrix Keypad to work.

Component List

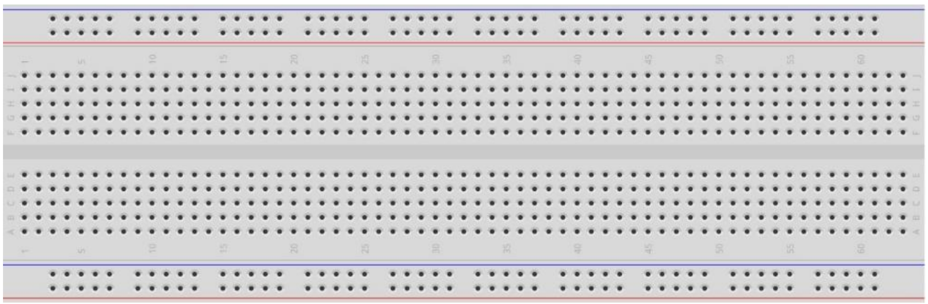
ESP8266 x1



USB cable

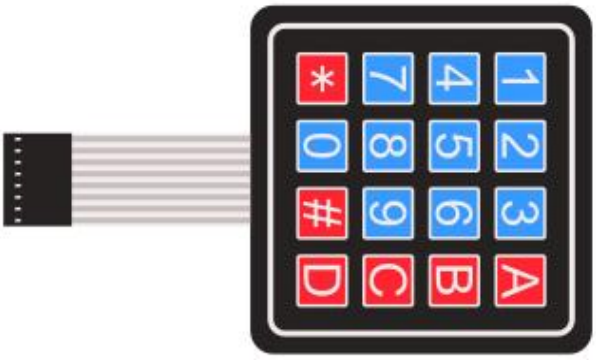


Breadboard x1



Jumper wire M/M x8

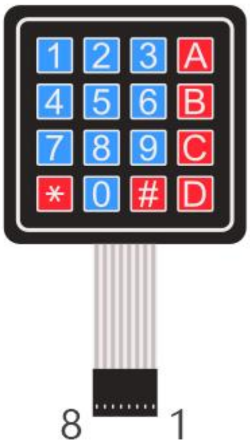
4x4 Matrix Keypad x1

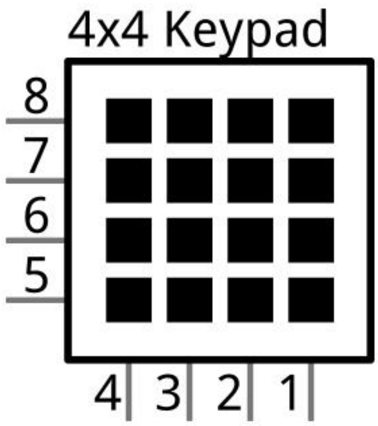


Component knowledge

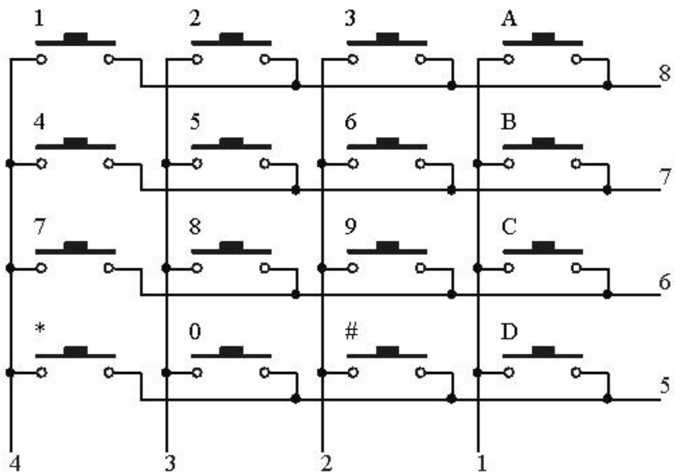
4x4 Matrix Keypad

A Keypad Matrix is a device that integrates a number of keys in one package. As is shown below, a 4x4 Keypad Matrix integrates 16 keys:





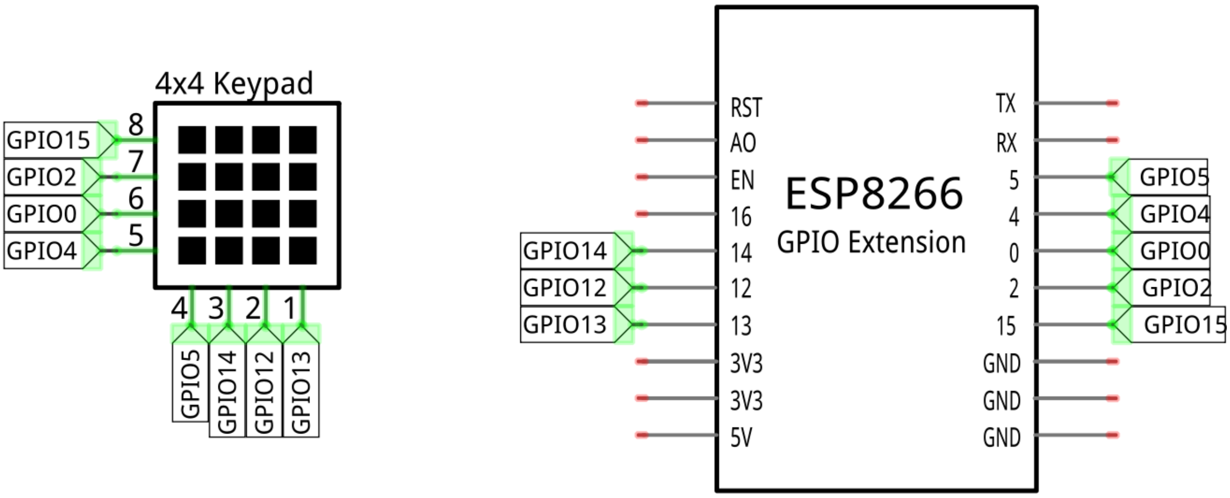
Similar to the integration of an LED Matrix, the 4x4 Keypad Matrix has each row of keys connected with one pin and this is the same for the columns. Such efficient connections reduce the number of processor ports required. The internal circuit of the Keypad Matrix is shown below.



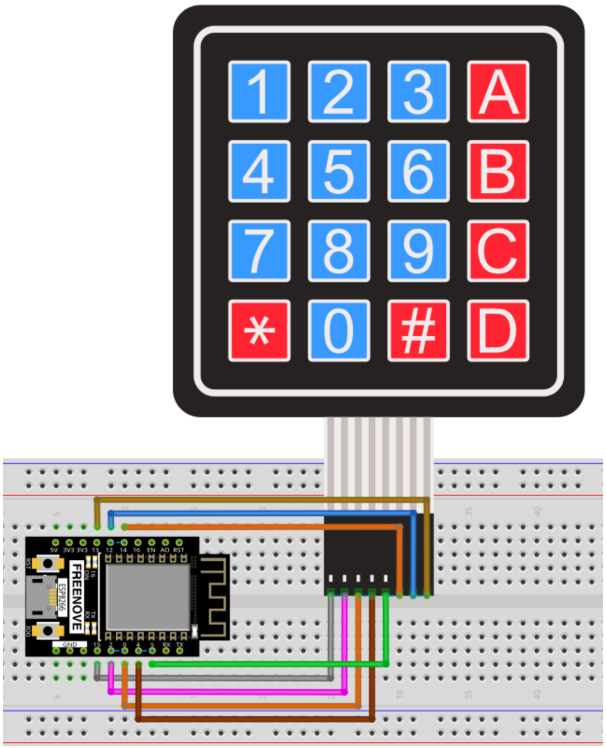
The method of usage is similar to the Matrix LED, by using a row or column scanning method to detect the state of each key's position by column and row. Take column scanning method as an example, send low level to the first 1 column (Pin1), detect level state of row 5, 6, 7, 8 to judge whether the key A, B, C, D are pressed. Then send low level to column 2, 3, 4 in turn to detect whether other keys are pressed. By this means, you can get the state of all of the keys.

Circuit

Schematic diagram



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



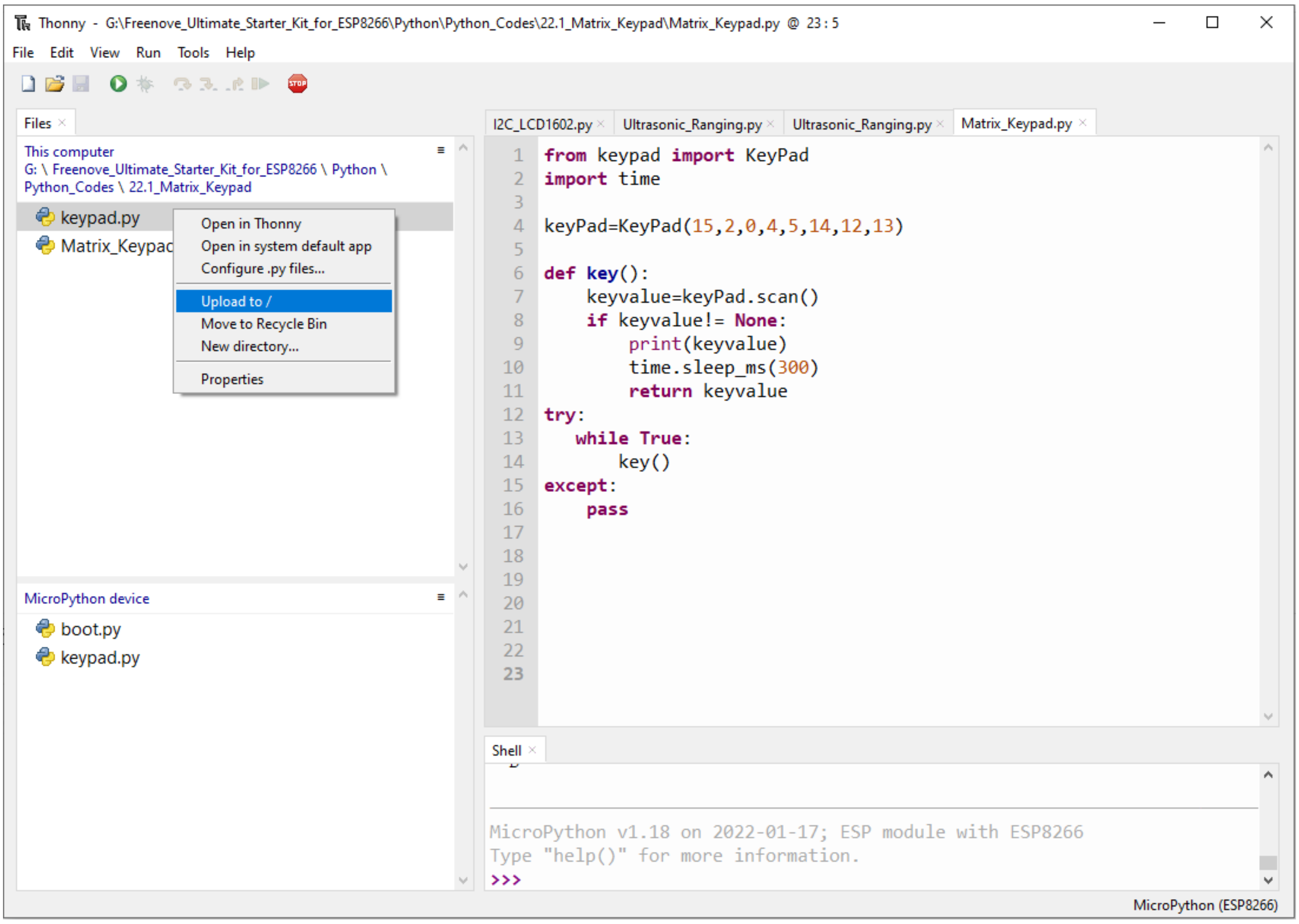
Code

This code is used to obtain all key codes of the 4x4 Matrix Keypad, when one of the keys is pressed, the key code will be printed out via serial port.

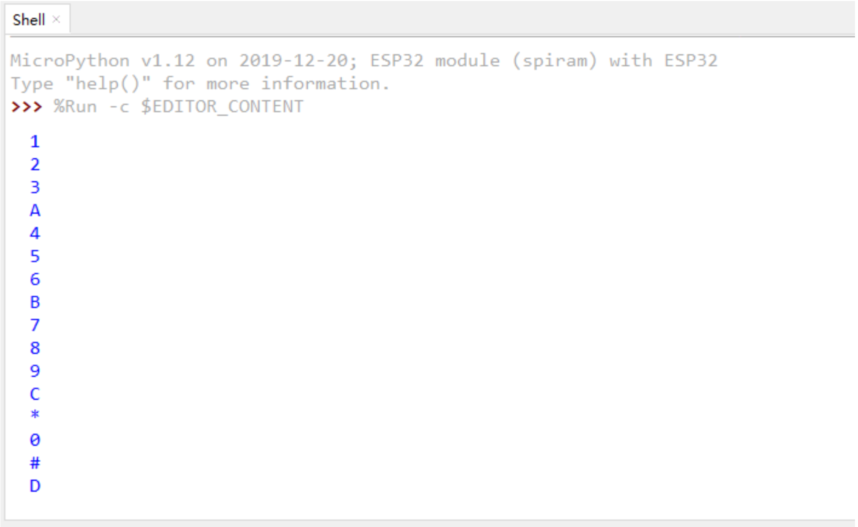
Move the program folder "**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "22.1_Matrix_Keypad". Select "keypad.py", right click your mouse to select "Upload to /", wait for "keypad.py" to be uploaded to ESP8266 and then double click "Matrix_Keypad.py".

22.1_Matrix_Keypad



Click "Run current script", push the key board and the key value will be printed in "Shell". As shown in the illustration below:



The following is the program code:

```
from keypad import KeyPad
import time

keyPad=KeyPad(15, 2, 0, 4, 5, 14, 12, 13)

def key():
    keyvalue=keyPad.scan()
    if keyvalue!= None:
        print(keyvalue)
        time.sleep_ms(300)
        return keyvalue

while True:
    key()
```

Import keypad module.

```
from keypad import KeyPad
```

Associate the keypad module to ESP8266 pins.

```
keyPad=KeyPad(15, 2, 0, 4, 5, 14, 12, 13)
```

Call function keypad.scan() of the keypad module. When the keypad module detects that the key is pressed, it returns the value of the pressed key; when no key is pressed, the return value is None.

```
keyPad.scan()
```

Call function keyPan.scan() to obtain the value of the pressed key. Once it is obtained, print it out.

```
def key():
    keyvalue=keyPad.scan()
    if keyvalue!= None:
        print(keyvalue)
        time.sleep_ms(300)
        return keyvalue
```

Reference

Class keypad

Before each use of the object **KeyPad**, please make sure **keypad.py** has been uploaded to "/" of ESP8266 and then add the statement "**from keypad import KeyPad**" to the top of python file.

KeyPad(row1,row2,row3,row4,col1,col2,col3,col4): Intialize keypad module and associate its pins with ESP8266.

scan(): Non-blocking keypad scan function. If no key is pressed, it returns None; Otherwise, it returns the valued of the pressed key.

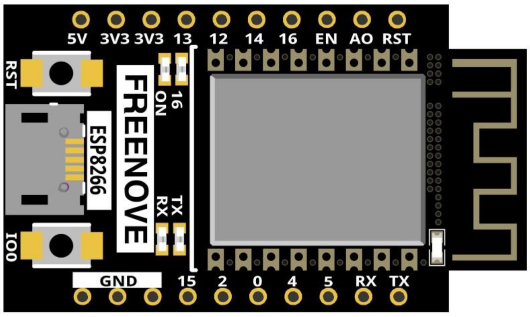
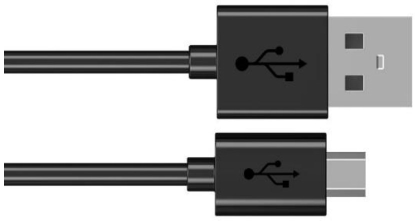
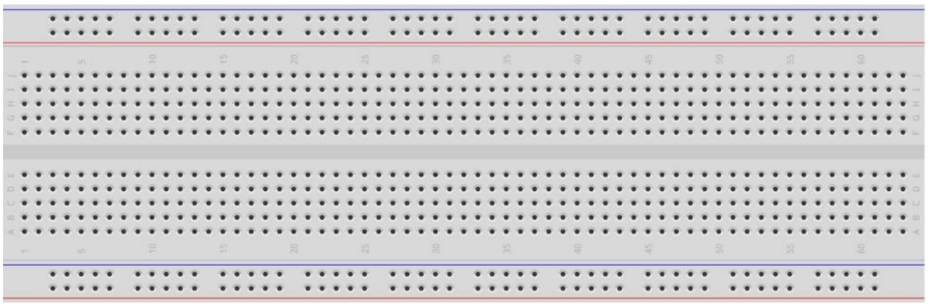
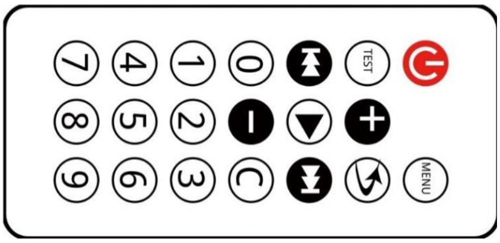
Chapter 23 Infrared Remote

In this chapter, we'll learn how to use an infrared remote control, and control a LED.

Project 23.1 Infrared Remote Control

First, we need to understand how infrared remote control works, then get the command sent from infrared remote control.

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 colspan="2">Jumper wire M/M x6
</td><td colspan="2" rowspan="2">Infrared Remote x1
(May need CR2025 battery x1, please check the holder)
</td></tr>
<tr><td>Infrared Remote x1
</td><td>Resistor 10kΩ x1
</td></tr>
</table>

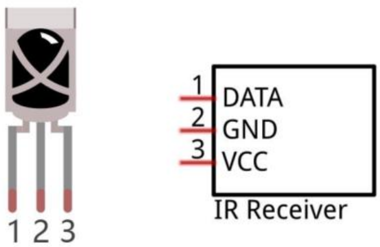
Component knowledge

Infrared Remote

An infrared(IR) remote control is a device with a certain number of buttons. Pressing down different buttons will make the infrared emission tube, which is located in the front of the remote control, send infrared ray with different command. Infrared remote control technology is widely used in electronic products such as TV, air conditioning, etc. Thus making it possible for you to switch TV programs and adjust the temperature of the air conditioning when away from them. The remote control we use is shown below:

Infrared receiver

An infrared(IR) receiver is a component which can receive the infrared light, so we can use it to detect the signal emitted by the infrared remote control. DATA pin here outputs the received infrared signal.



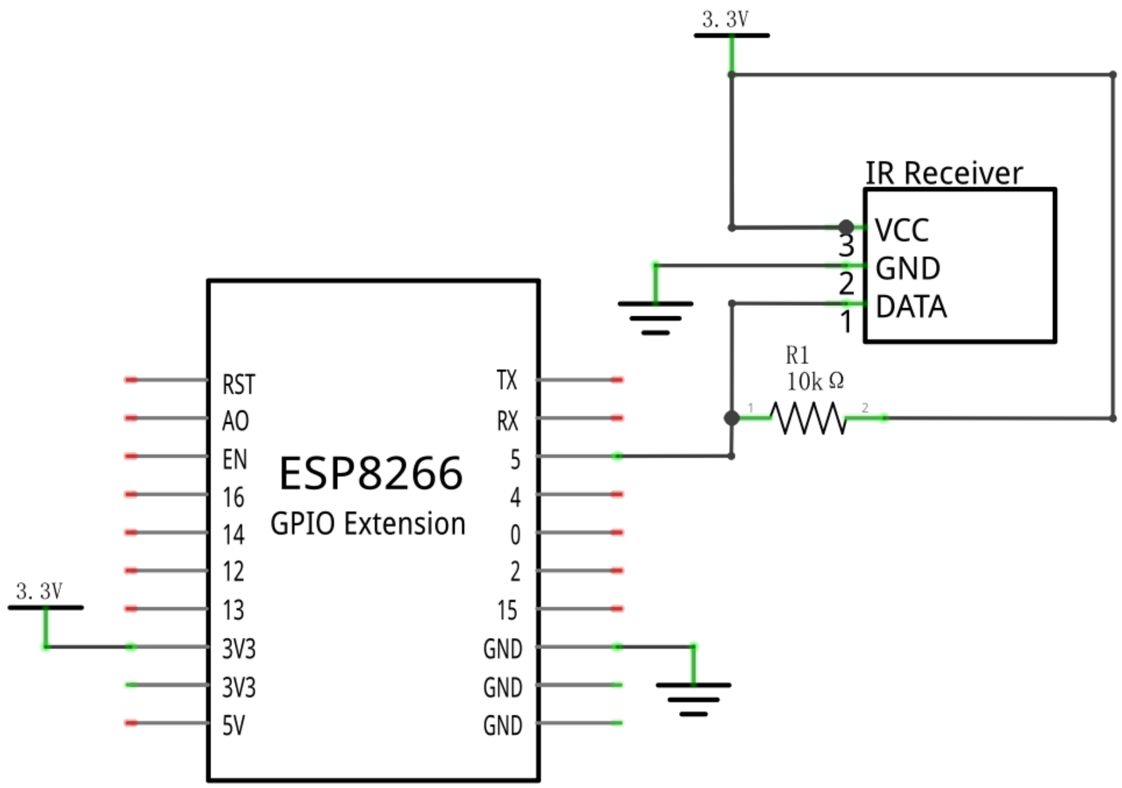
When you use the infrared remote control, the infrared remote control sends a key value to the receiving circuit according to the pressed keys. We can program the ESP8266 to do things like lighting, when a key value is received.

The following is the key value that the receiving circuit will receive when each key of the infrared remote control is pressed.

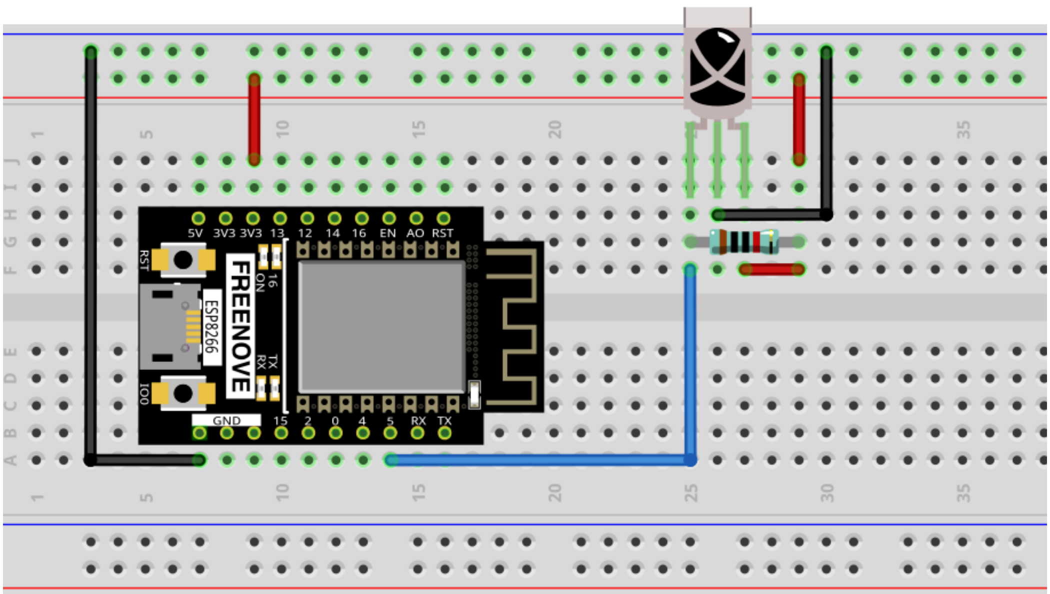
ICON	KEY Value	ICON	KEY Value
⏻	FFA25D	C	FFB04F
MENU	FFE21D	1	FF30CF
TEST	FF22DD	2	FF18E7
+	FF02FD	3	FF7A85
↩	FFC23D	4	FF10EF
⏮	FFE01F	5	FF38C7
▶	FFA857	6	FF5AA5
⏭	FF906F	7	FF42BD
0	FF6897	8	FF4AB5
−	FF9867	9	FF52AD

Circuit

Schematic diagram



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

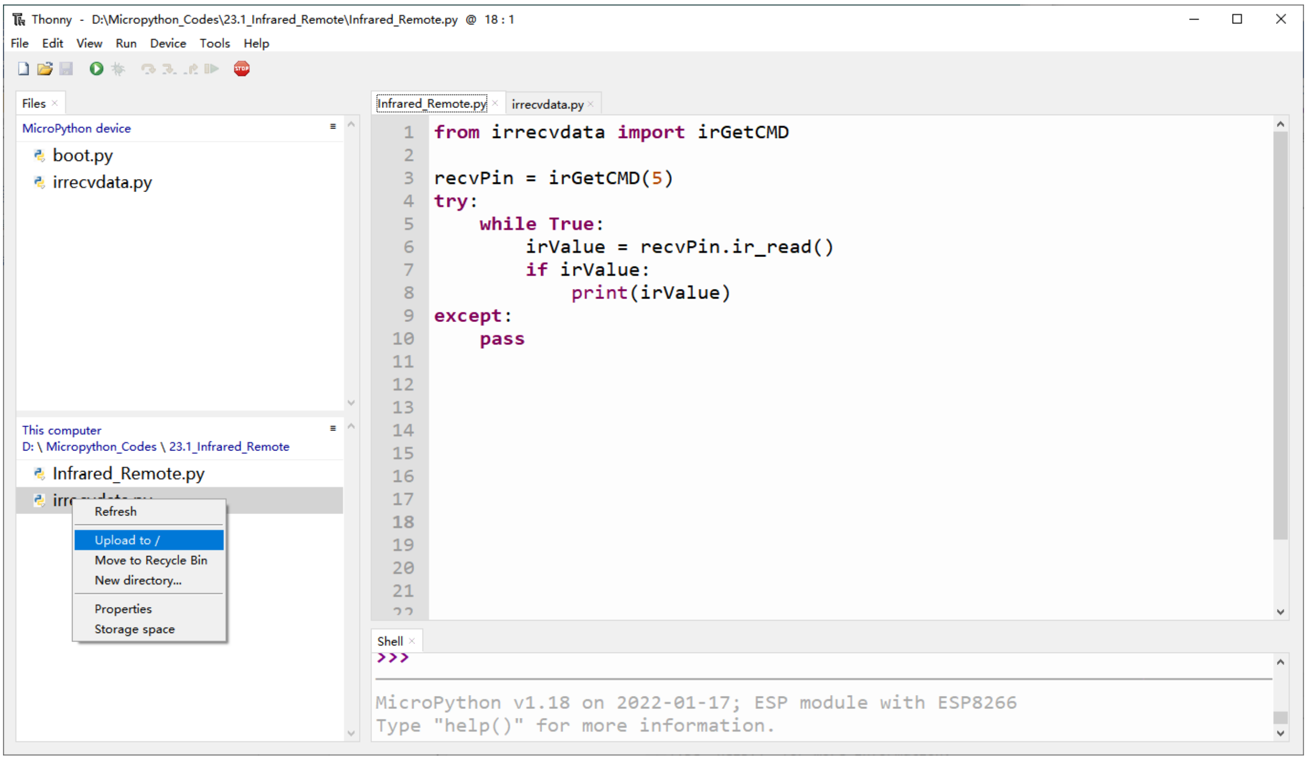


Code

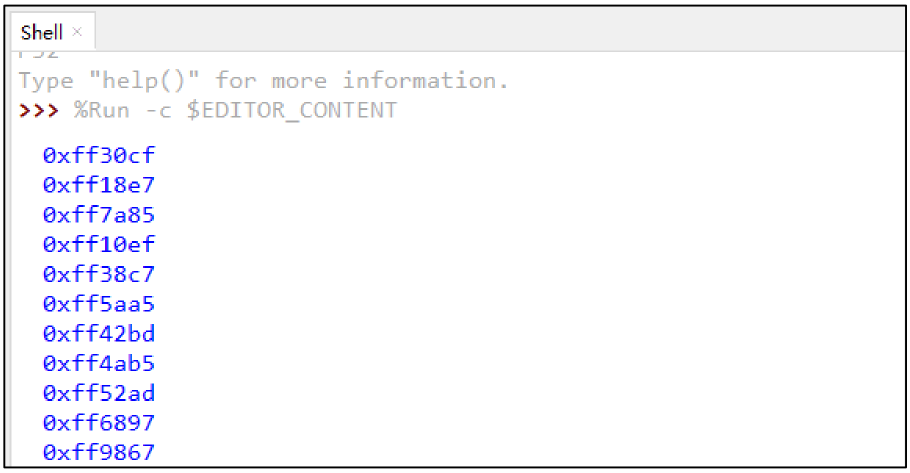
Move the program folder "**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "23.1_Infrared_Remote". Select "irrecvdata.py", right click your mouse to select "Upload to /", wait for "irrecvdata.py" to be uploaded to ESP8266 and then double click "Infrared_Remote.py".

23.1_Infrared_Remote



Click "Run current script", press the key of the infrared remote and the key value will be printed in "Shell". As shown in the illustration below:



The following is the program code:

```
from irrecvdata import irGetCMD

recvPin = irGetCMD(5)
try:
    while True:
        irValue = recvPin.ir_read()
        if irValue:
            print(irValue)
except:
    pass
```

Import the infrared decoder.

```
from irrecvdata import irGetCMD
```

Associate the infrared decoder with Pin(5).

```
recvPin = irGetCMD(5)
```

Call ir_read() to read the value of the pressed key and assign it to IRValue.

```
irValue = recvPin.ir_read()
```

When infrared key value is obtained, print it out in" Shell".

```
    while True:
        irValue = recvPin.ir_read()
        if irValue:
            print(irValue)
```

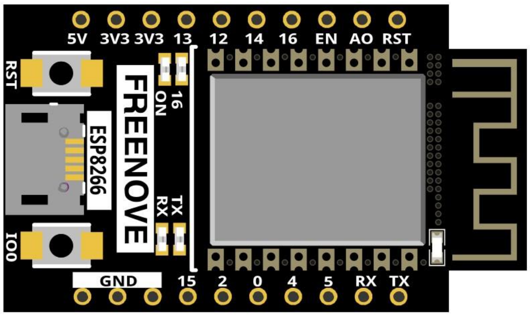
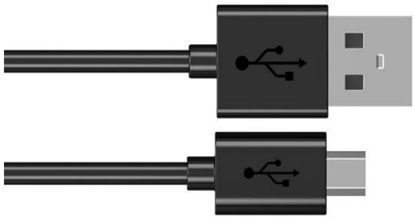
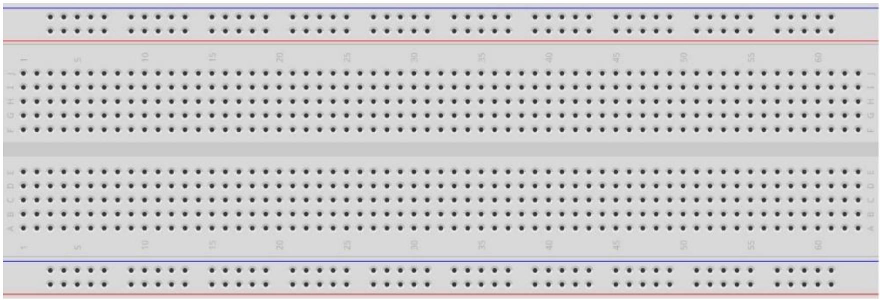
Reference

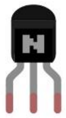
Class irrecvdata
Before each use of the object **irrecvdata**, please add the statement “**from irrecvdata import irGetCMD**” to the top of the python file. **irGetCMD()**: Object of infrared encoder, which is associated with Pin(15) by default. **ir_read()**: The function that reads the key value of infrared remote. When the value is read, it will be returned; when no value is obtained, character **None** will be returned.

Project 23.2 Control LED through Infrared Remote

In this project, we will control the brightness of LED lights through an infrared remote control.

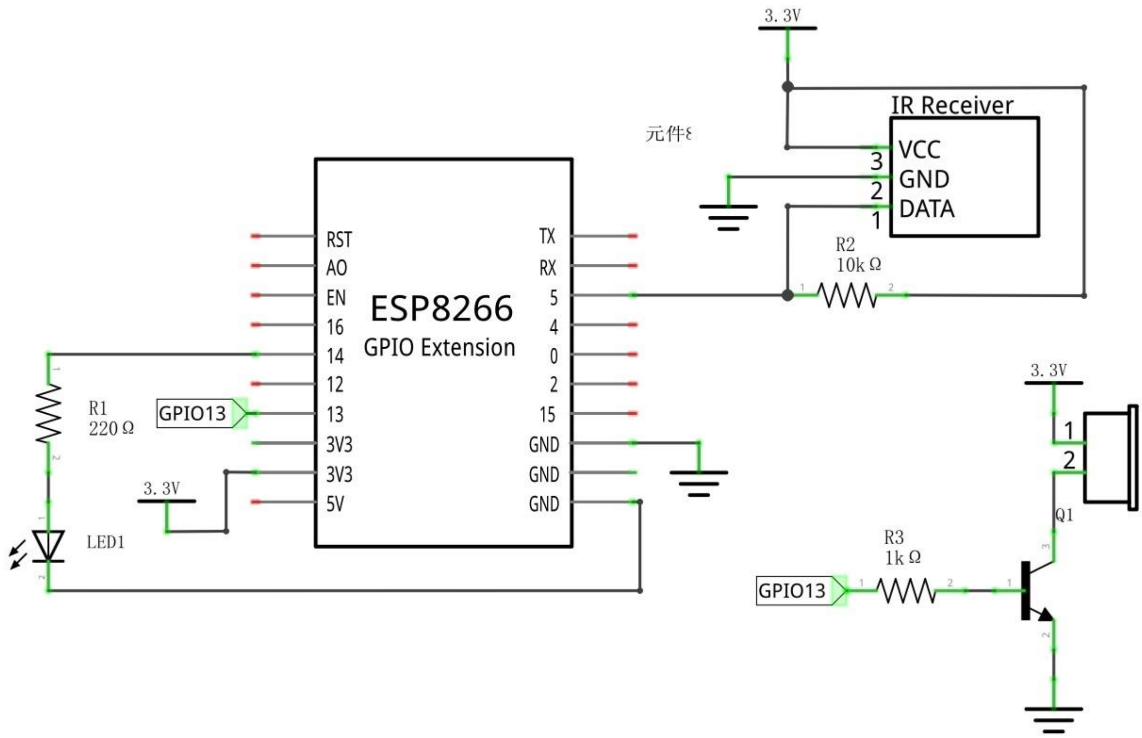
Component List

<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 colspan="3">Jumper wire M/M x12
</td><td rowspan="3">Infrared Remote x1
(May need CR2025 battery x1, please check the holder)
</td></tr>
<tr><td>LED x1
</td><td colspan="2">Resistor 220Ω x1
</td></tr>
<tr><td>Infrared receiver x1
</td><td colspan="2">Resistor 10kΩ x1
</td></tr>
</table>

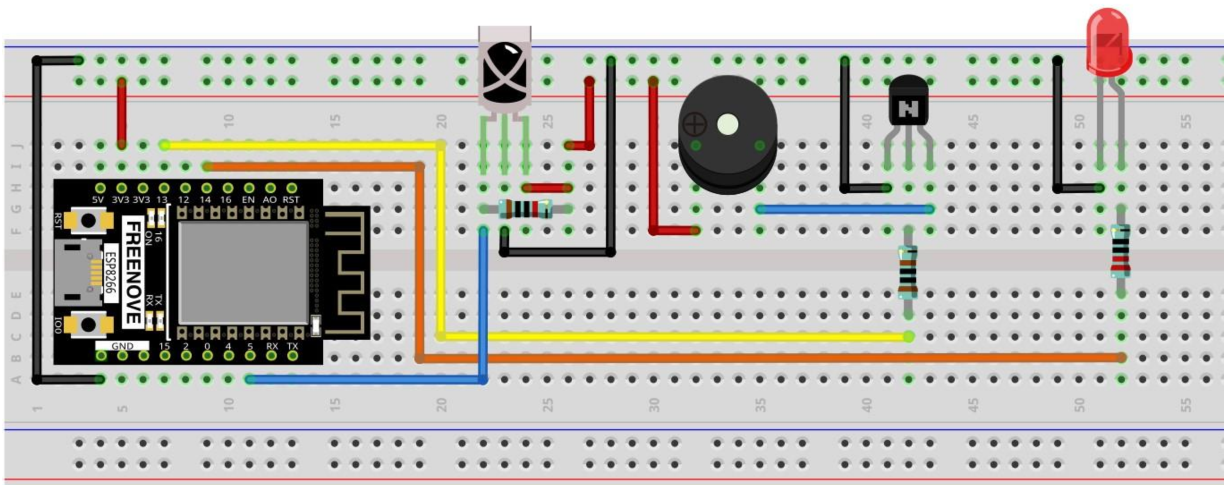
NPN transistorx1 (S8050)	Active buzzer x1	Resistor 1kΩ x1
		

Circuit

Schematic diagram



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



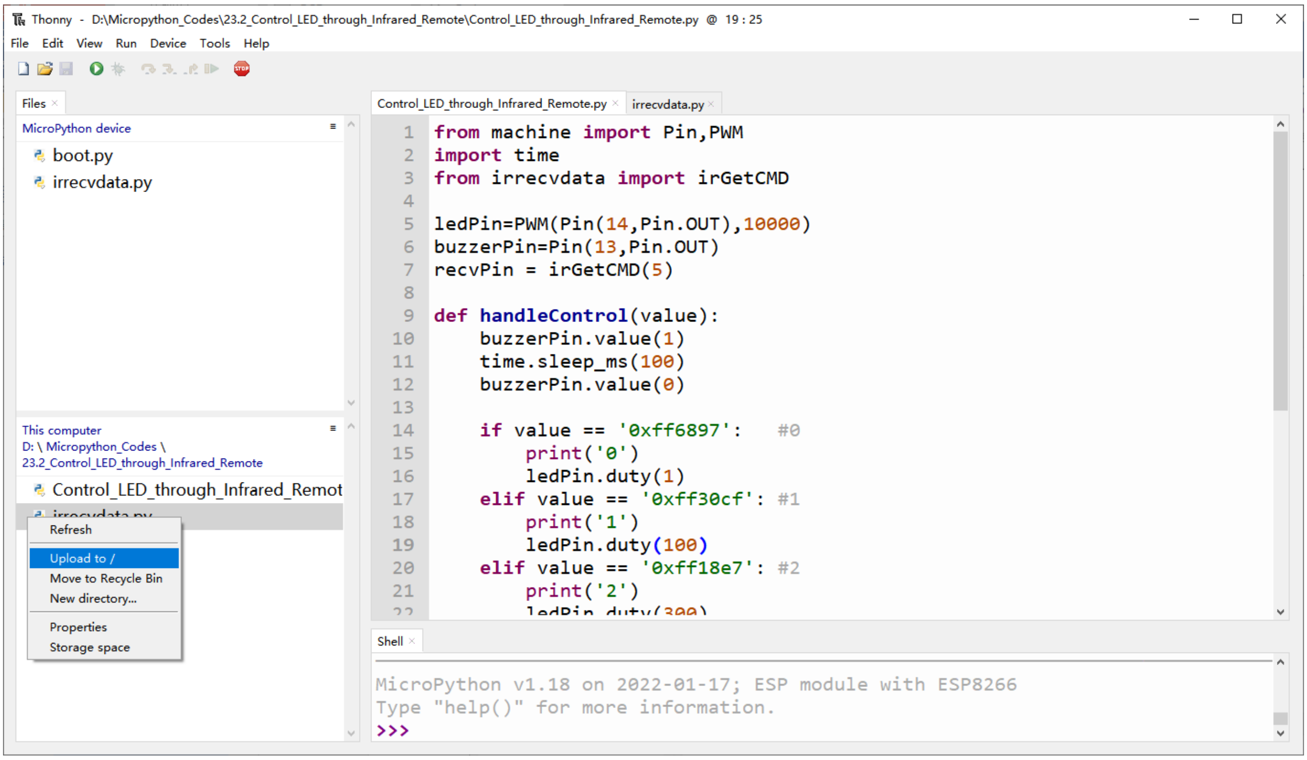
Code

The Code controls the brightness of the LED by determining the key value of the infrared received.

Move the program folder “**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**” to disk(D) in advance with the path of “**D:/Micropython_Codes**”.

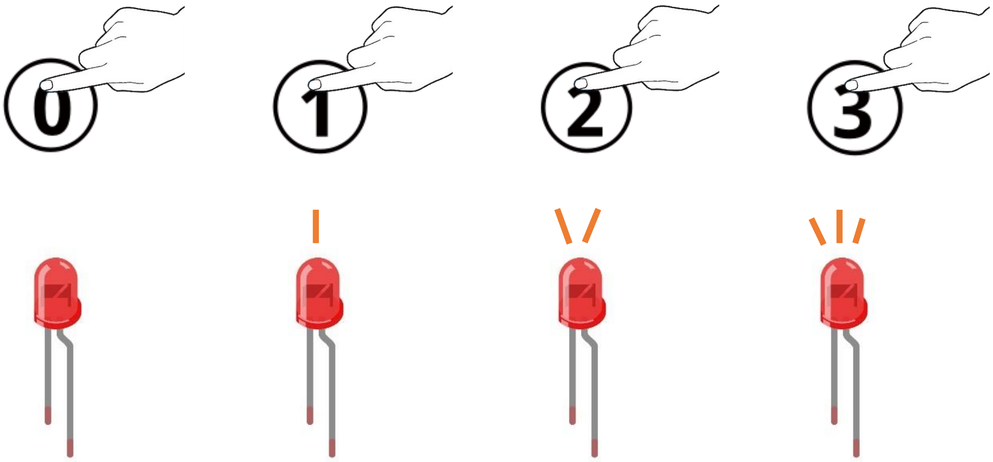
Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “23.2_Control_LED_through_Infrared_Remote”. Select “irrecvdata.py”, right click your mouse to select “Upload to /”, wait for “irrecvdata.py” to be uploaded to ESP8266 and then double click “Control_LED_through_Infrared_Remote.py”.

23.2_Control_LED_through_Infrared_Remote



Click “Run current script”. When pressing "0", "1", "2", "3" of the infrared remote control, the buzzer will sound once, and the brightness of the LED light will change correspondingly.

Rendering



The following is the program code:

```
from machine import Pin,PWM
import time
from irrecvdata import irGetCMD

ledPin=PWM(Pin(14,Pin.OUT),10000)
buzzerPin=Pin(13,Pin.OUT)
recvPin = irGetCMD(5)

def handleControl(value):
    buzzerPin.value(1)
    time.sleep_ms(100)
    buzzerPin.value(0)

    if value == '0xff6897':   #0
        print('0')
        ledPin.duty(1)
    elif value == '0xff30cf': #1
        print('1')
        ledPin.duty(100)
    elif value == '0xff18e7': #2
        print('2')
        ledPin.duty(300)
    elif value == '0xff7a85': #3
        print('3')
        ledPin.duty(1000)
    else:
        return
```

```
try:
    while True:
        irValue = recvPin.ir_read()
        if irValue:
            print(irValue)
            handleControl(irValue)
except:
    ledPin.deinit()
```

The handleControl() function is used to execute events corresponding to infrared code values. Every time when the function is called, the buzzer sounds once and determines the brightness of the LED based on the infrared key value. If the key value is not "0", "1", "2", "3", the buzzer sounds once, but the brightness of LED will not change.

```
def handleControl(value):
    buzzerPin.value(1)
    time.sleep_ms(100)
    buzzerPin.value(0)

    if value == '0xff6897':   #0
        print('0')
        ledPin.duty(1)
    elif value == '0xff30cf': #1
        print('1')
        ledPin.duty(100)
    elif value == '0xff18e7': #2
        print('2')
        ledPin.duty(300)
    elif value == '0xff7a85': #3
        print('3')
        ledPin.duty(1000)
    else:
        return
```

Each time the key value of IR remote is received, function handleControl() will be called to process it.

```
try:
    while True:
        irValue = recvPin.ir_read()
        if irValue:
            print(irValue)
            handleControl(irValue)
except:
    ledPin.deinit()
```

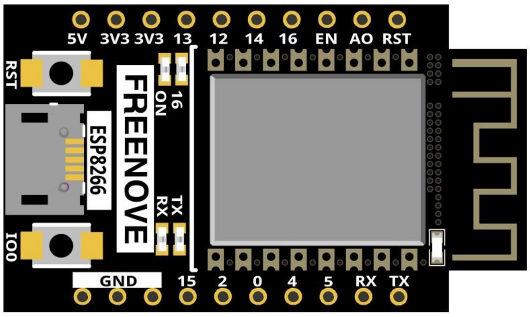
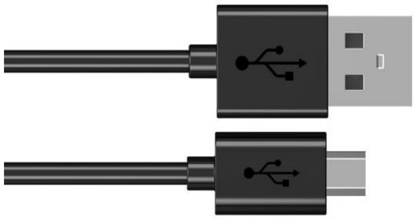
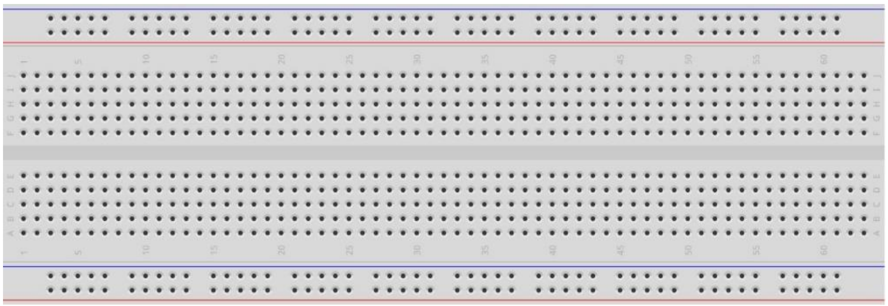
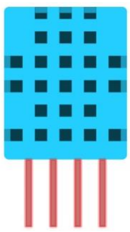
Chapter 24 Hygrothermograph DHT11

In this chapter, we will learn about a commonly used sensor called a Hygrothermograph DHT11

Project 24.1 Hygrothermograph

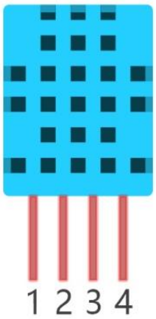
Hygrothermograph is an important tool in our lives to give us data on the temperature and humidity in our environment. In this project, we will use the ESP8266 to read Temperature and Humidity data of the DHT11 Module.

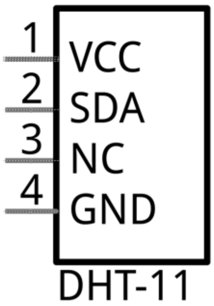
Component List

<table>
<tr><td colspan="2">ESP8266 x1

</td><td>USB cable
</td></tr>
<tr><td colspan="3">Breadboard x1
</td></tr>
<tr><td>Jumper wire M/M x6
</td><td>DHT11 x1
</td><td>Resistor 10kΩ x1
</td></tr>
</table>

Component knowledge

The Temperature & Humidity Sensor DHT11 is a compound temperature & humidity sensor, and the output digital signal has been calibrated by its manufacturer.

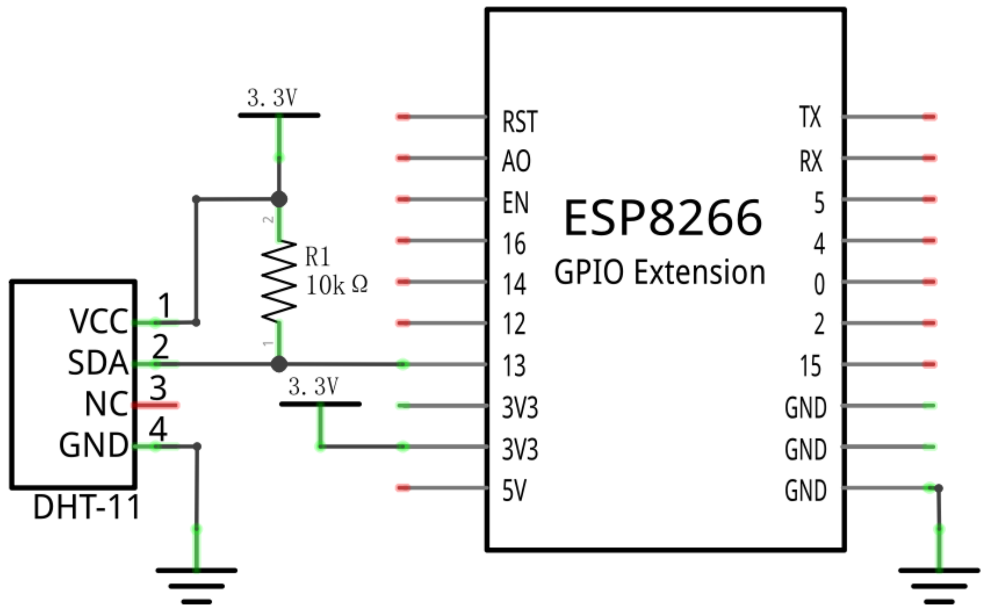




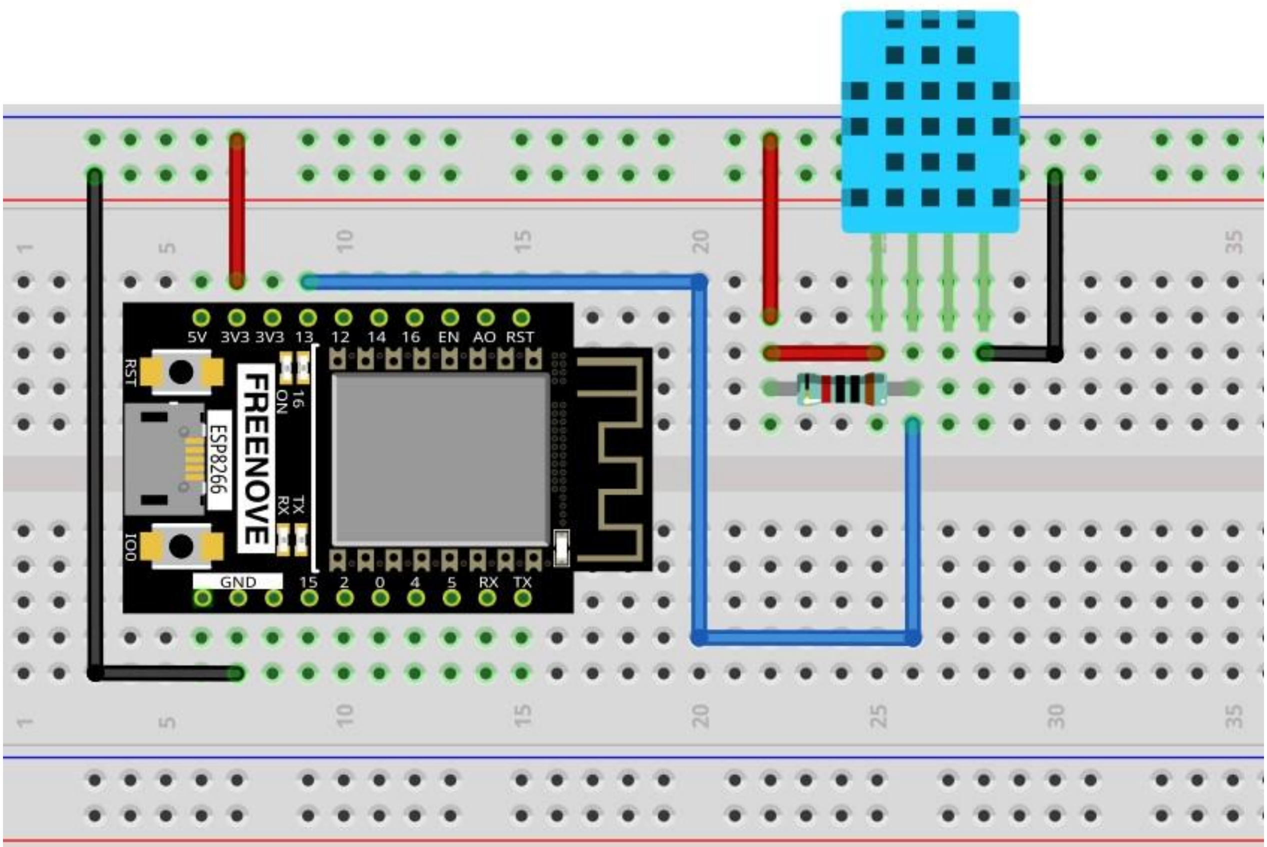
DHT11 uses customized single-line communication protocol, so we can use the library to read data more conveniently.
After being powered up, it will initialize in 1S's time. Its operating voltage is within the range of 3.3V-5.5V.
The SDA pin is a data pin, which is used to communicate with other devices.
The NC pin (Not Connected Pin) is a type of pin found on various integrated circuit packages. Those pins have no functional purpose to the outside circuit (but may have an unknown functionality during manufacture and test). Those pins should not be connected to any of the circuit connections.

Circuit

Schematic diagram



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



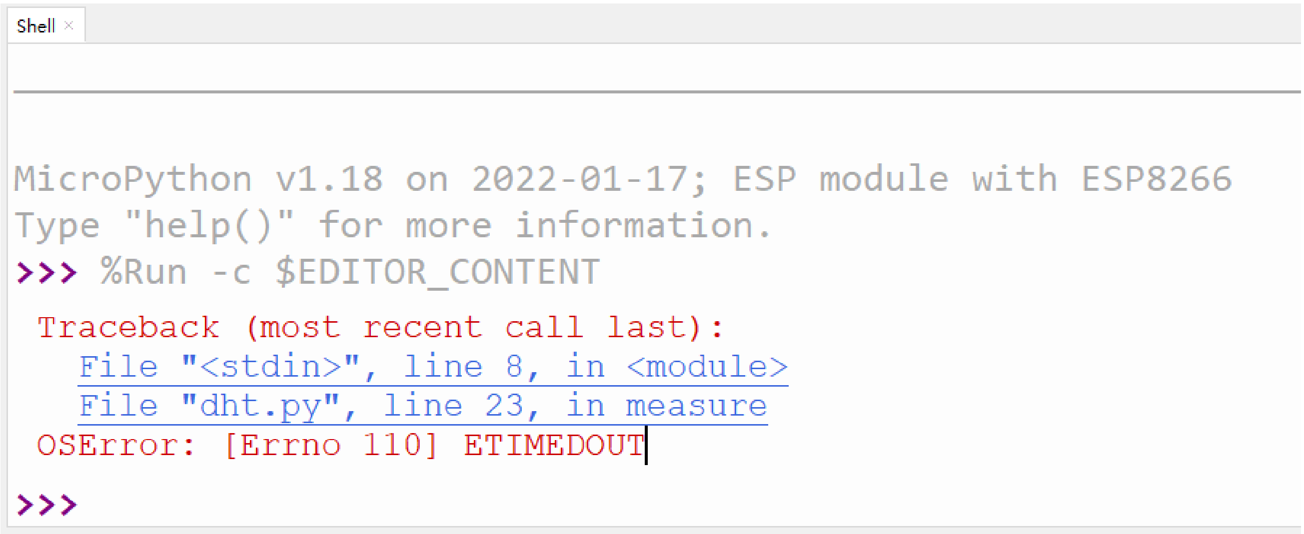
Code

Move the program folder "**Freenove_Ultimate_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" → "24.1_Hygrothermograph" and double click "Hygrothermograph.py".

24.1_Hygrothermograph

Click "Run current script". If your DHT11 is connected incorrectly, the following information will be printed in "Shell".



Make sure your circuit is correctly connected and you will see the following messages printed in “Shell”.

```
Shell
MicroPython v1.18 on 2022-01-17; ESP module with ESP8266
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
 temperature: 28 C humidity: 59 %
 temperature: 26 C humidity: 60 %
 temperature: 26 C humidity: 60 %
 temperature: 26 C humidity: 60 %
 temperature: 26 C humidity: 60 %
 temperature: 26 C humidity: 60 %
 temperature: 26 C humidity: 60 %
 temperature: 26 C humidity: 60 %
 temperature: 26 C humidity: 60 %
 temperature: 26 C humidity: 60 %
 temperature: 26 C humidity: 60 %
 temperature: 26 C humidity: 60 %
 temperature: 26 C humidity: 60 %
```

The following is the program code:

```python
import machine
import time
import dht

DHT = dht.DHT11(machine.Pin(13))

while True:
    DHT.measure()
    print('temperature:',DHT.temperature(),'humidity:',DHT.humidity())
    time.sleep_ms(1000)
```

Import machine, time and dht modules.

```python
import machine
import time
import dht
```

Associate DHT11 with Pin(13).

```python
DHT = dht.DHT11(machine.Pin(13))
```

Start DHT11 to measure data once.

```python
DHT.measure()
```

Call the built-in function of DHT to obtain temperature and humidity data and print them in “Shell”.

```python
print('temperature:',DHT.temperature(),'humidity:',DHT.humidity())
```

Obtain temperature and humidity data once per second and print them out.

```python
while True:
    DHT.measure()
    print('temperature:',DHT.temperature(),'humidity:',DHT.humidity())
```

```
        time.sleep_ms(1000)
```

Reference

Class dht

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

DHT11(): Object of DHT11

DHT12(): Object of DHT12

DHT11.measure(): Start DHT11 to measure temperature and humidity data once.

DHT11.temperature(): Return temperature data obtained by DHT11.

DHT11.humidity(): Return humidity data obtained by DHT11.

DHT12.measure(): Start DHT12 to measure temperature and humidity data once

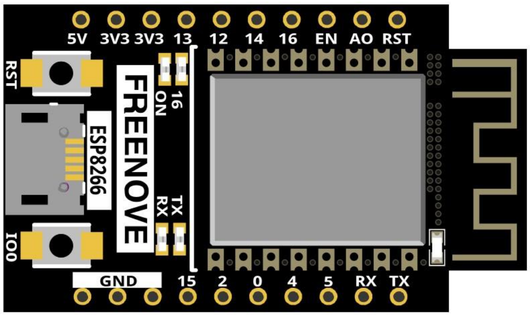
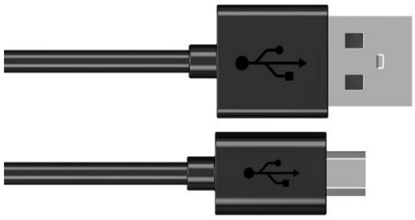
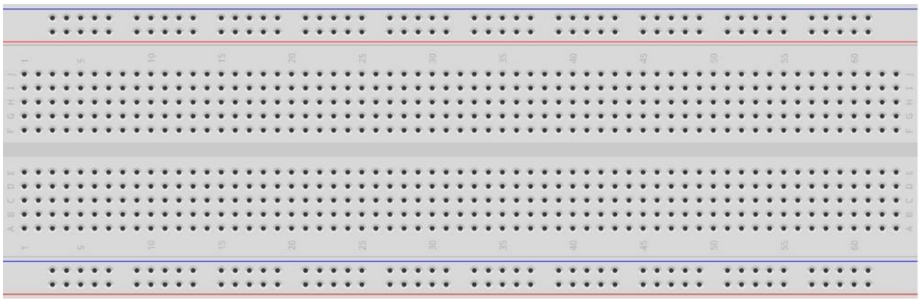
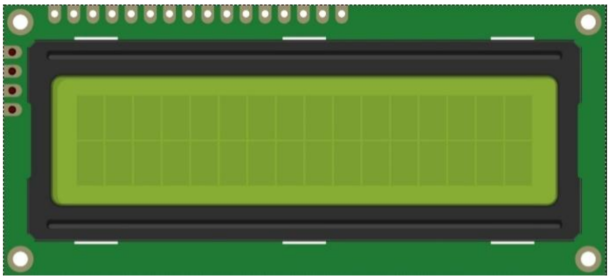
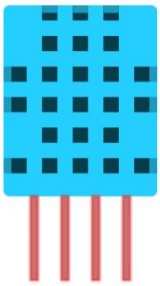
DHT12.temperature(): Return temperature data obtained by DHT12.

DHT12.humidity(): Return humidity data obtained by DHT12.

Project 24.2 Hygrothermograph

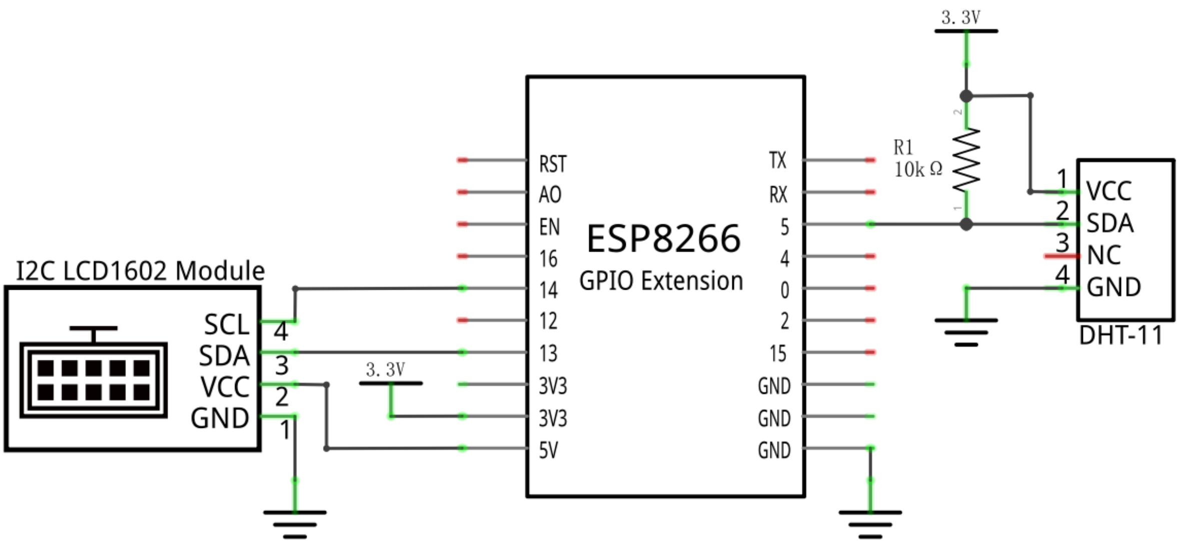
In this project, we use L2C-LCD1602 to display data collected by DHT11.

Component List

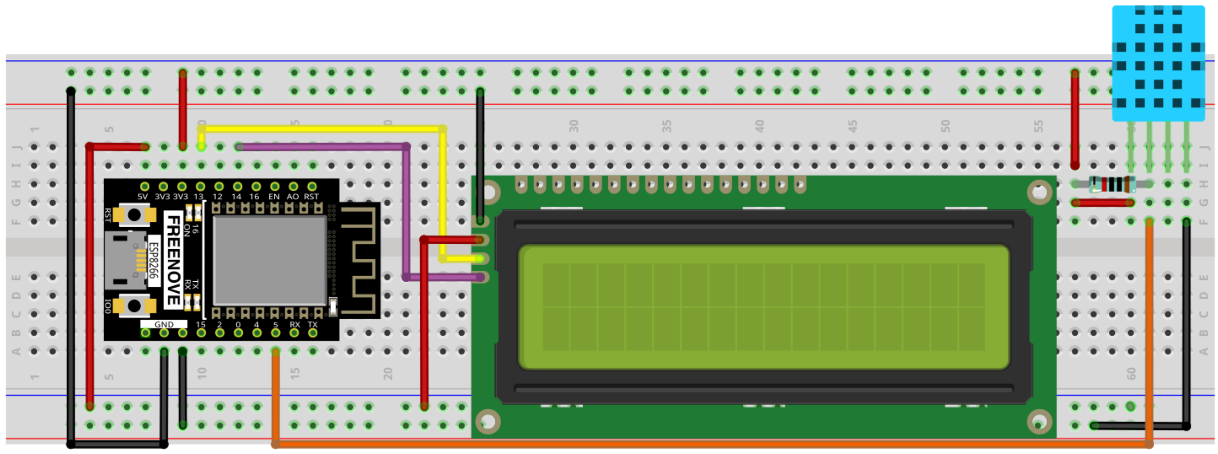
ESP8266 x1 	USB cable 
Breadboard x1 	
LCD1602 Module x1 	Resistor 10kΩ x1 
Jumper wire F/M x4 Jumper wire M/M x8 	DHT11 x1 

Circuit

Schematic diagram



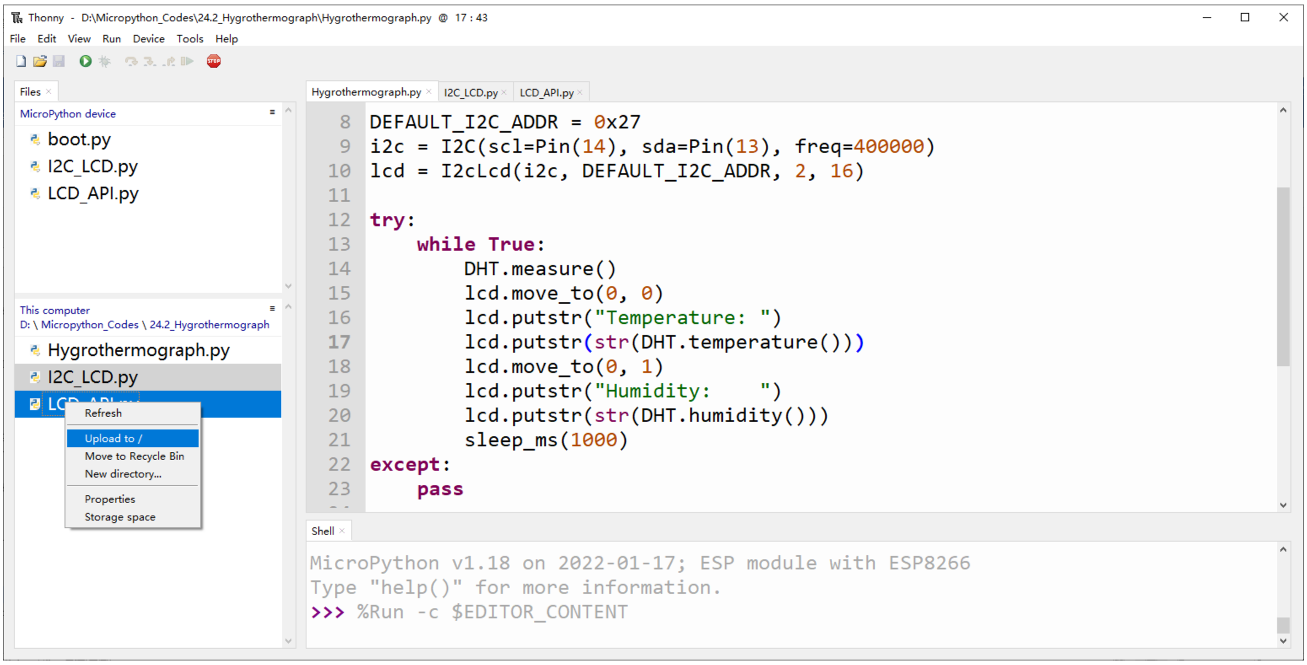
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



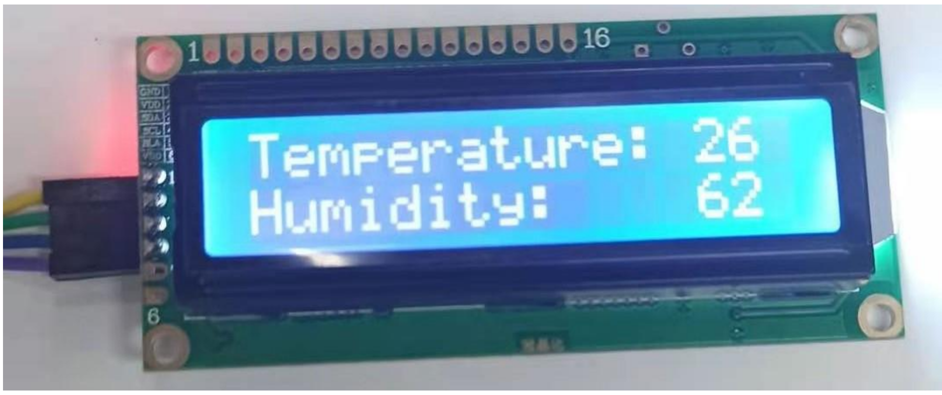
Code

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → 24.2_Hygrothermograph".
Select "I2C_LCD.py" and "LCD_API.py", right click your mouse to select "Upload to /", wait for "I2C_LCD.py" and "LCD_API.py" to be uploaded to ESP8266 and then double click "Hygrothermograph.py".

24.2_Hygrothermograph



Click "Run current script". The first row of LCD1602 is temperature value and the second row is humidity. Try to "pinch" the DHT11 (without touching the leads) with your index finger and thumb for a brief time, you should see that the displayed value on LCD1602 changes.



The following is the program code:

```
from time import sleep_ms
from machine import I2C, Pin
from I2C_LCD import I2cLcd
import dht

DHT = dht.DHT11(Pin(5))
DEFAULT_I2C_ADDR = 0x27
i2c = I2C(scl=Pin(14), sda=Pin(13), freq=400000)
lcd = I2cLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)

try:
    while True:
        DHT.measure()
        lcd.move_to(0, 0)
        lcd.putstr("Temperature: ")
        lcd.putstr(str(DHT.temperature()))
        lcd.move_to(0, 1)
        lcd.putstr("Humidity:    ")
        lcd.putstr(str(DHT.humidity()))
        sleep_ms(1000)
except:
    pass
```

Import DHT11 and I2C LCD1602 modules.

```
from time import sleep_ms
from machine import I2C, Pin
from I2C_LCD import I2cLcd
import dht
```

Assign Pin(5) to DHT11, Pin(13) and Pin(14) to LCD1602.

```
DHT = dht.DHT11(Pin(5))
DEFAULT_I2C_ADDR = 0x27
i2c = I2C(scl=Pin(14), sda=Pin(13), freq=400000)
lcd = I2cLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)
```

Obtain data of Hygrothermograph every second and display them on LCD1602. The first line displays temperature and the second line displays humidity.

```
    while True:
        DHT.measure()
        lcd.move_to(0, 0)
        lcd.putstr("Temperature: ")
        lcd.putstr(str(DHT.temperature()))
        lcd.move_to(0, 1)
        lcd.putstr("Humidity:    ")
        lcd.putstr(str(DHT.humidity()))
        sleep_ms(1000)
```

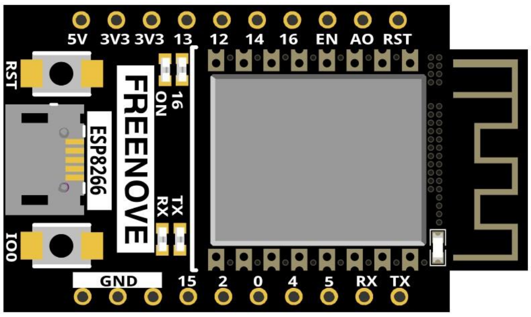
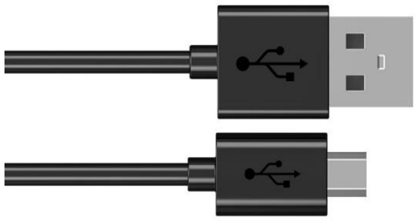
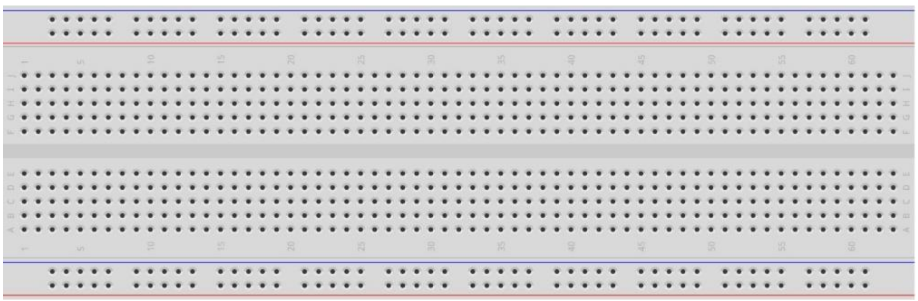
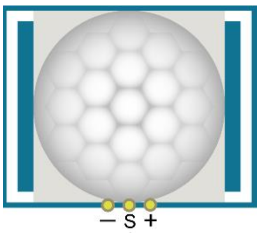
Chapter 25 Infrared Motion Sensor

In this chapter, we will learn a widely used sensor, Infrared Motion Sensor.

Project 25.1 Infrared Motion Detector with LED Indicator

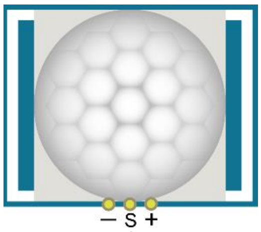
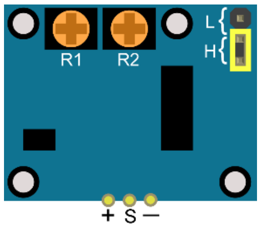
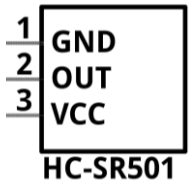
In this project, we will make a Motion Detector, with the human body infrared pyroelectric sensors.
When someone is in close proximity to the Motion Detector, it will automatically light up and when there is no one close by, it will be out.
This Infrared Motion Sensor can detect the infrared spectrum (heat signatures) emitted by living humans and animals.

Component List

<table>
<tr><td colspan="3">ESP8266 x1
</td><td>USB cable
</td></tr>
<tr><td colspan="4">Breadboard x1
</td></tr>
<tr><td>HC SR501 x1
</td><td>LED x1
</td><td>Resistor 220Ω x1
</td><td>Jumper wire F/M x3
Jumper wire M/M x4
</td></tr>
</table>

Component knowledge

The following is the diagram of the infrared Motion sensor（HC SR-501）:

Top	Bottom	Schematic
		

Description:

Working voltage: 5v-20v(DC) Static current: 65uA.

Automatic Trigger. When a living body enters into the active area of sensor, the module will output high level (3.3V). When the body leaves the sensor's active detection area, it will output high level lasting for time period T, then output low level(0V). Delay time T can be adjusted by the potentiometer R1.

According to the position of Jumper wire cap, you can choose non-repeatable trigger mode or repeatable mode.

L: non-repeatable trigger mode. The module output high level after sensing a body, then when the delay time is over, the module will output low level. During high level time, the sensor no longer actively senses bodies.

H: repeatable trigger mode. The distinction from the L mode is that it can sense a body until that body leaves during the period of high level output. After this, it starts to time and output low level after delaying T time.

Induction block time: the induction will stay in block condition and does not induce external signal at lesser time intervals (less than delay time) after outputting high level or low level.

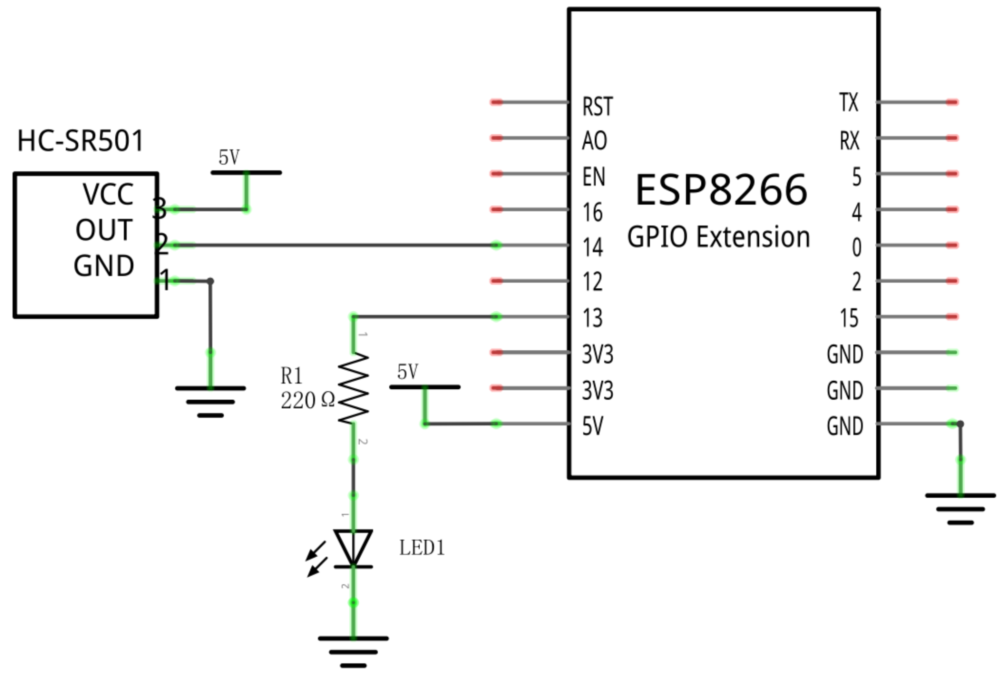
Initialization time: the module needs about 1 minute to initialize after being powered ON. During this period, it will alternately output high or low level.

One characteristic of this sensor is when a body moves close to or moves away from the sensor's dome edge, the sensor will work at high sensitivity. When a body moves close to or moves away from the sensor's dome in a vertical direction, the sensor cannot detect well (please take note of this deficiency). Note: The Sensing Range (distance before a body is detected) is adjusted by the potentiometer.

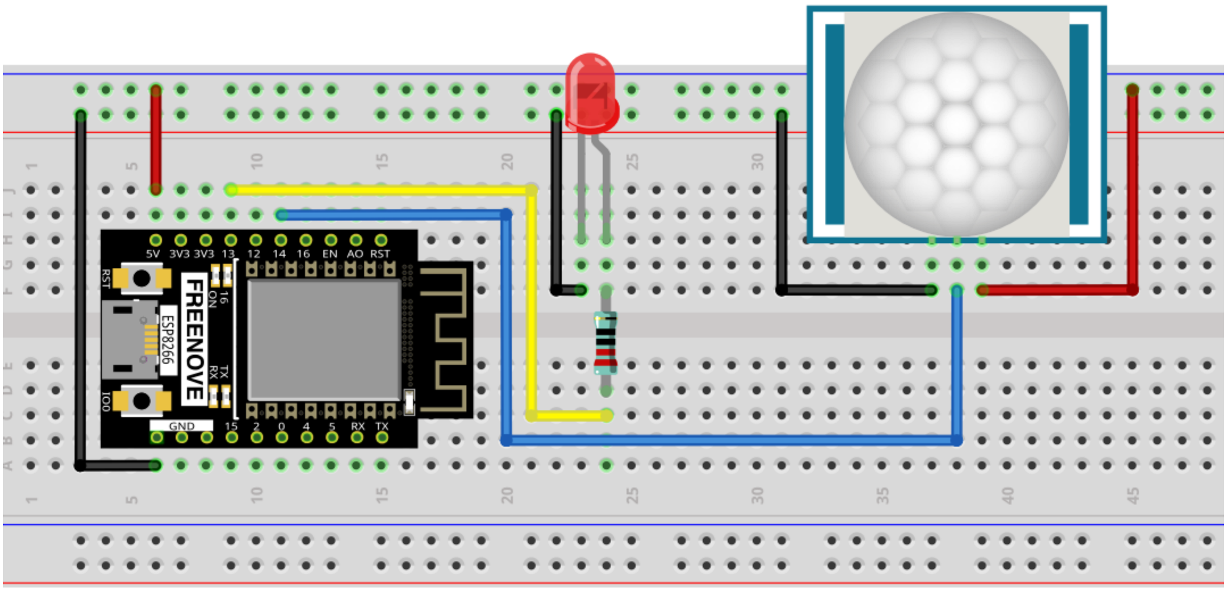
We can regard this sensor as a simple inductive switch when in use.

Circuit

Schematic diagram



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



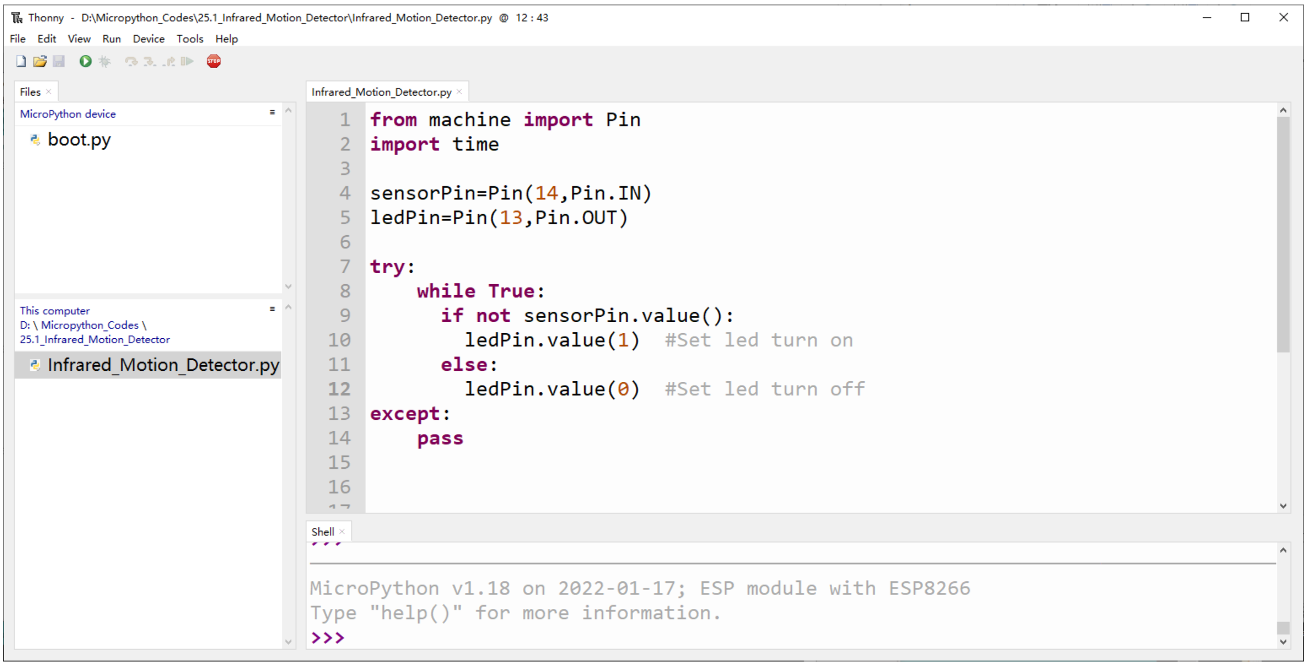
Code

In this project, we will use an infrared motion sensor to trigger an LED, essentially using the infrared motion sensor as a motion switch. So the code of this project is similar to that of project "Button & Led". The difference is when infrared motion sensor detects changes, it will out high level; when it detects nothing, it will output low level. When the sensor outputs high level, LED turns ON; Otherwise, LED turns OFF.

Move the program folder "**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

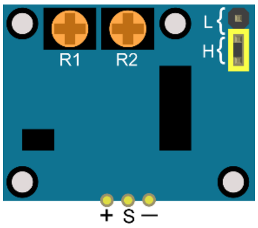
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "Infrared_Motion_Detector" and then double click"Infrared_Motion_Detector.py".

25.1_Infrared_Motion_Detector



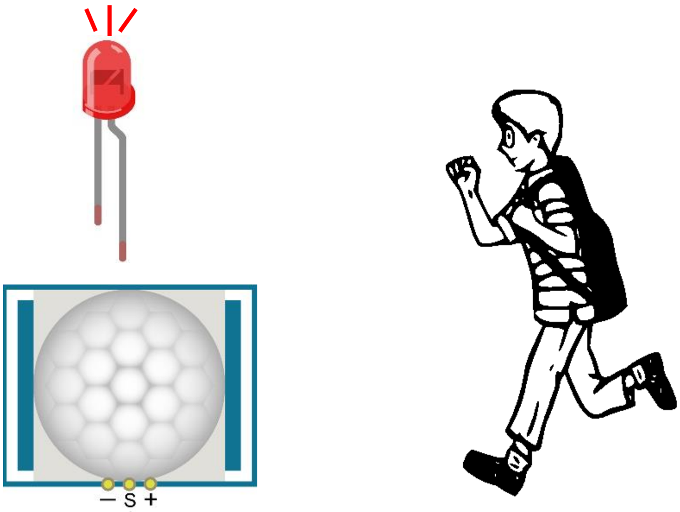
Click "Run current script". Put the sensor on a stationary table and wait for about a minute. Then try to move away from or move closer to the Infrared Motion Sensor and observe whether the LED turns ON or OFF automatically.

You can rotate the potentiometer on the sensor to adjust the detection effect, or use different modes by changing the jumper.



Apart from that, you can also use this sensor to control some other modules to implement different functions by reediting the code, such as the induction lamp, induction door.

Move to the Infrared Motion Sensor



Move away from the Infrared Motion Sensor

Description:

1. You can choose non repeatable trigger modes or repeatable modes.

L: nonrepeatable trigger mode. The module output high level after sensing a body, then when the delay time is over, the module will output low level. During high level time, the sensor no longer actively senses bodies.

H: repeatable trigger mode. The distinction from the L mode is that it can sense a body until that body leaves. After this, it starts to time and output low level after delaying T time.

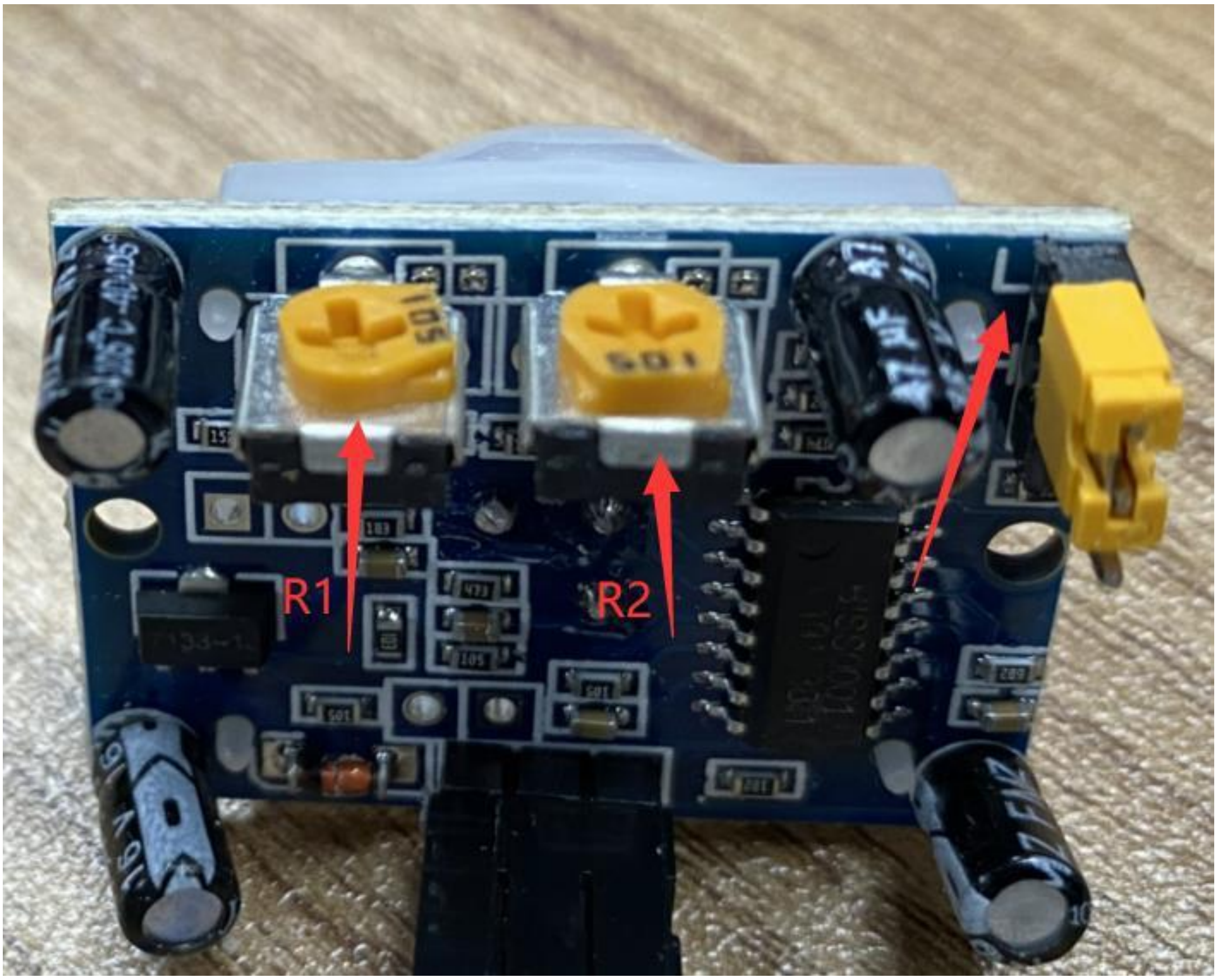
2. R1 is used to adjust HIGH level last ing time when sensor detect s human motion , 1.2 s 320 s

3. R 2 is used to adjust the maxmum distance the sensor can detect, 3~5m.

Here we connect L and adjust R1 and R2 like below to do this project.

Put you hand close and away from the sensor slowly. Obsever the LED in previous circuit.

It need some time between two detections.



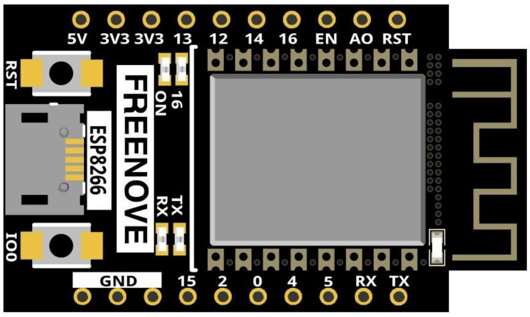
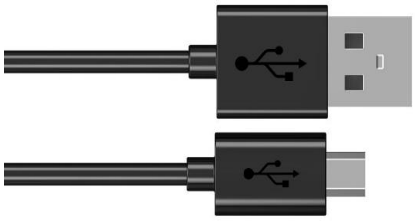
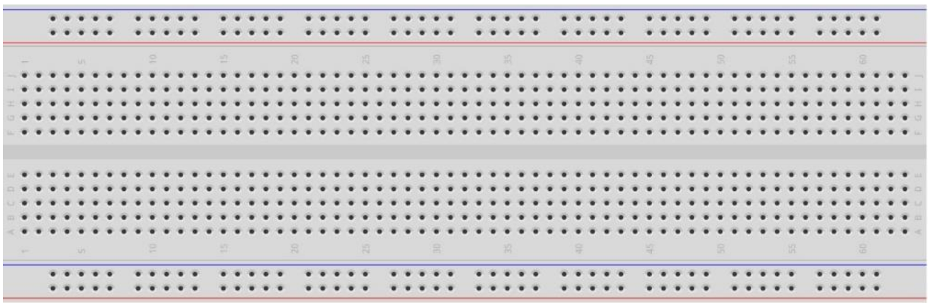
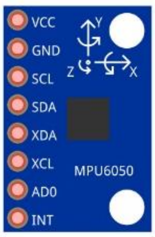
Chapter 26 Attitude Sensor MPU6050

In this chapter, we will learn about a MPU6050 Attitude Sensor which integrates an Accelerometer and Gyroscope.

Project 26.1 Read a MPU6050 Sensor Module

In this project, we will read Acceleration and Gyroscope Data of the MPU6050 Sensor

Component List

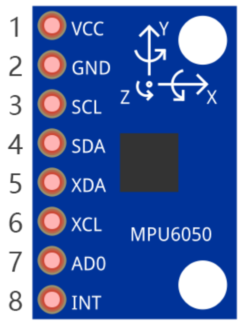
ESP8266 x1 	USB cable 
Breadboard x1 	
Jumper wire F/M x6 	MPU6050 x1 

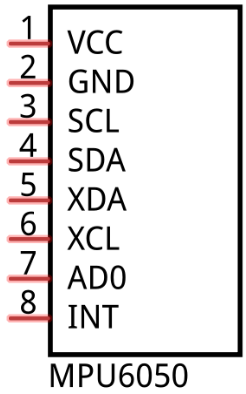
Component knowledge

MPU6050

MPU6050 Sensor Module is a complete 6-axis Motion Tracking Device. It combines a 3-axis Gyroscope, a 3-axis Accelerometer and a DMP (Digital Motion Processor) all in a small package. The settings of the Accelerometer and Gyroscope of MPU6050 can be changed. A precision wide range digital temperature sensor is also integrated to compensate data readings for changes in temperature, and temperature values can also be read. The MPU6050 Module follows the I2C communication protocol and the default address is 0x68.

MPU6050 is widely used to assist with balancing vehicles, robots and aircraft, mobile phones and other products which require stability to control stability and attitude or which need to sense same.





The port description of the MPU6050 module is as follows:

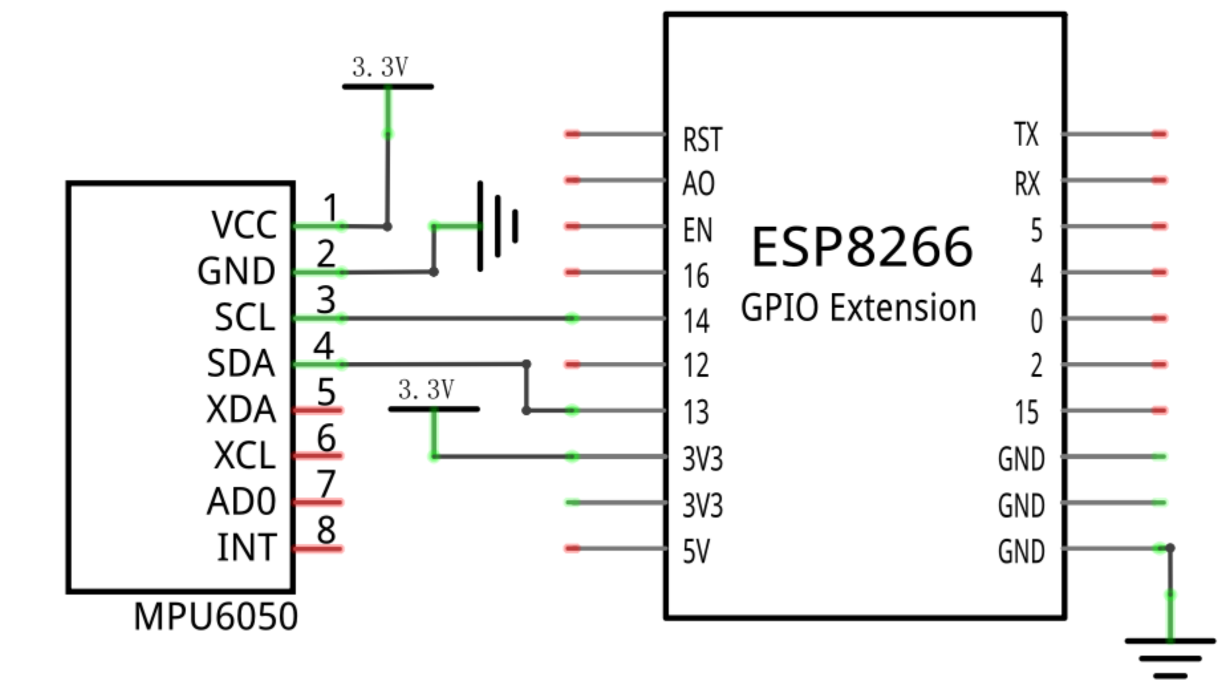
Pin name	Pin number	Description
VCC	1	Positive pole of power supply with voltage 5V
GND	2	Negative pole of power supply
SCL	3	I2C communication clock pin
SDA	4	I2C communication clock pin
XDA	5	I2C host data pin which can be connected to other devices.
XCL	6	I2C host clock pin which can be connected to other devices.
AD0	7	I2C address bit control pin. Low level: the device address is 0x68 High level: the device address is 0x69
INT	8	Output interrupt pin

For more detail, please refer to datasheet.

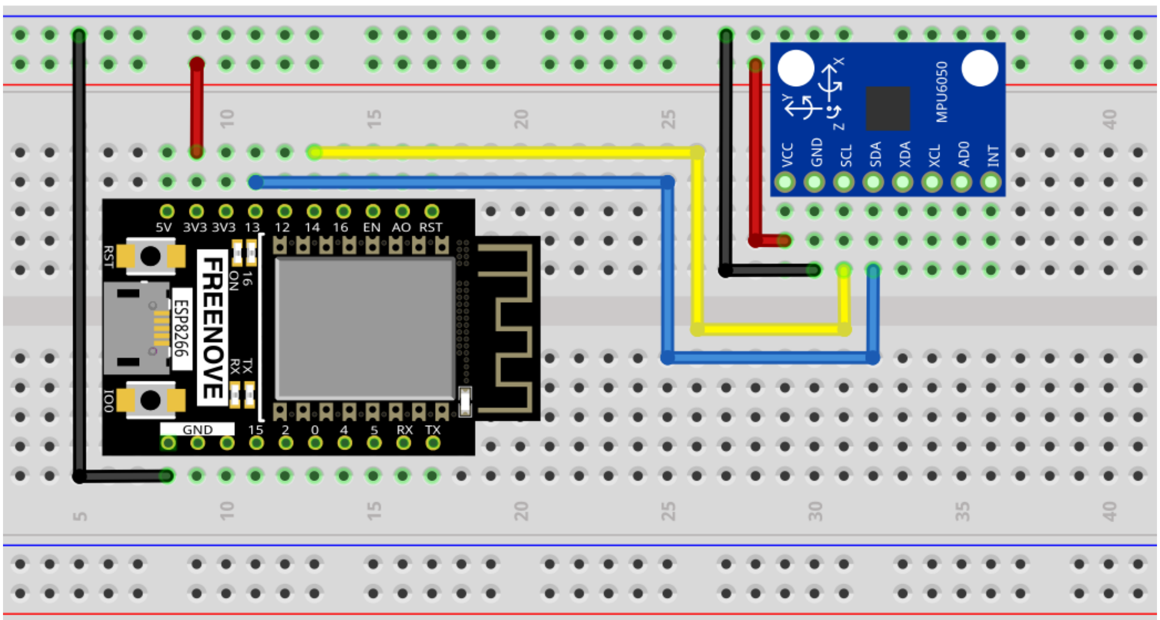
Circuit

Note that the power supply voltage for MPU6050 module is 5V in the circuit.

Schematic diagram



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

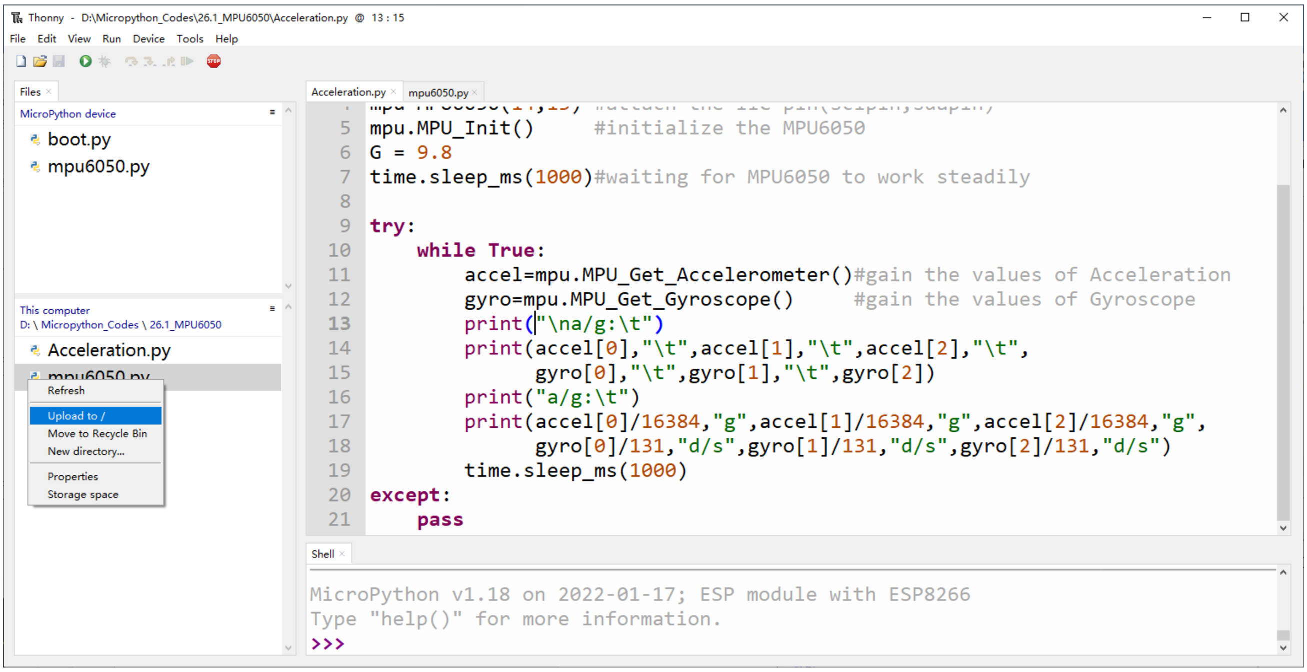


Code

Move the program folder “**Freenove_Ultimate_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” → “26.1_MPU6050”. Select “mpu6050.py”, right click your mouse to select “Upload to /”, wait for “mpu6050.py” to be uploaded to ESP8266 and then double click “MPU6050.py”.

26.1_MPU6050



Click “Run current script”. When ESP8266 obtains acceleration data and gyroscope data of MPU605, it will print them in “Shell”.

```
Shell

a/g:
1102      506      16422    -60      39       8
a/g:
0.0672607 g 0.0308838 g 1.00232 g -0.458015 d/s 0.29771 d/s 0.0610687 d/s

a/g:
972       436      16408    -59      39       7
a/g:
0.0593262 g 0.0266113 g 1.00146 g -0.450382 d/s 0.29771 d/s 0.0534351 d/s

a/g:
948       498      16412    -60      39       8
a/g:
0.0578613 g 0.0303955 g 1.00171 g -0.458015 d/s 0.29771 d/s 0.0610687 d/s

a/g:
950       440      16372    -59      38       7
a/g:
0.0579834 g 0.0268555 g 0.999268 g -0.450382 d/s 0.290076 d/s 0.0534351 d/s
```

Note: The data transmission of MPU6050 is very sensitive. Therefore, when using it, please make sure the jumper wire is in good contact, otherwise the data may fail to be obtained.

The following is the program code:

```
from mpu6050 import MPU6050
import time

mpu=MPU6050(14,13) #attach the IIC pin(sclpin,sdapin)
mpu.MPU_Init()     #initialize the MPU6050
G = 9.8
time.sleep_ms(1000)#waiting for MPU6050 to work steadily
try:
    while True:
        accel=mpu.MPU_Get_Accelerometer()#gain the values of Acceleration
        gyro=mpu.MPU_Get_Gyroscope()     #gain the values of Gyroscope
        print("\na/g:\t")
        print(accel[0],"\t",accel[1],"\t",accel[2],"\t",
              gyro[0],"\t",gyro[1],"\t",gyro[2])
        print("a/g:\t")
        print(accel[0]/16384,"g",accel[1]/16384,"g",accel[2]/16384,"g",
              gyro[0]/16384,"d/s",gyro[1]/16384,"d/s",gyro[2]/16384,"d/s")
        time.sleep_ms(1000)
except:
    pass
```

Import MPU6050 and time modules.

```
from mpu6050 import MPU6050
import time
```

Set I2C pins and associate them with MPU6050 module, and then initialize MPU6050 and wait for the initialization to complete.

```
mpu=MPU6050(14,13) #attach the IIC pin(sclpin,sdapin)
mpu.MPU_Init()     #initialize the MPU6050
G = 9.8
time.sleep_ms(1000)#waiting for MPU6050 to work steadily
```

Obtain the acceleration data of MPU6050 and store it in accel. Obtain the gyroscope data and store it in gyro.

```
        accel=mpu.MPU_Get_Accelerometer()#gain the values of Acceleration
        gyro=mpu.MPU_Get_Gyroscope()     #gain the values of Gyroscope
```

Update and collect the original data of the gyroscope every second and print the original data and processed acceleration and angular velocity data in “Shell”.

```
        accel=mpu.MPU_Get_Accelerometer()#gain the values of Acceleration
        gyro=mpu.MPU_Get_Gyroscope()     #gain the values of Gyroscope
        print("\na/g:\t")
        print(accel[0],"\t",accel[1],"\t",accel[2],"\t",
              gyro[0],"\t",gyro[1],"\t",gyro[2])
        print("a/g:\t")
        print(accel[0]/16384,"g",accel[1]/16384,"g",accel[2]/16384,"g",
```

```
            gyro[0]/16384,"d/s",gyro[1]/16384,"d/s",gyro[2]/16384,"d/s")
        time.sleep_ms(1000)
```

Reference

Class mpu6050

Before each use of **mpu6050**, please add the statement "**from mpu6050 import MPU6050**" to the top of the python file.

MPU6050(sclpin,sdapin): Create an object MPU6050 and associate I2C pin with it.

MPU_Init(): Intialize MPU6050 module.

MPU_Get_Accelerometer(): Obtain original data of MPU6050's acceleration

MPU_Get_Gyroscope(): Obtain original data of MPU6050's Gyroscope

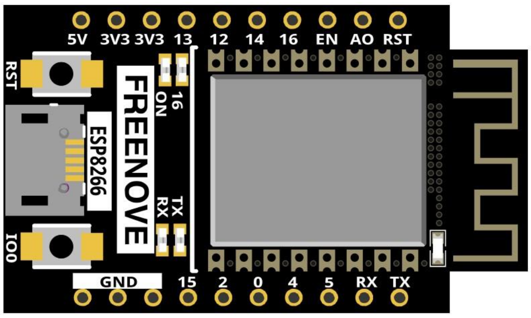
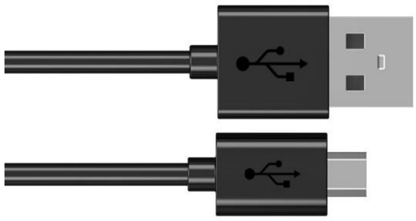
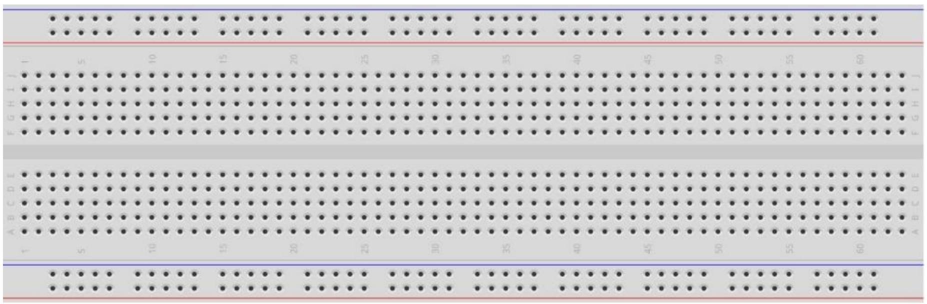
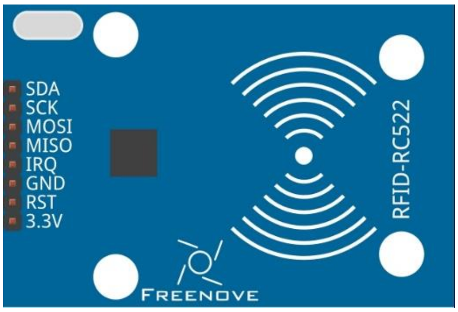
Chapter 27 RFID

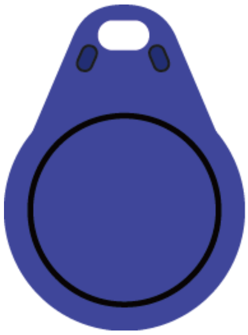
Now, we will learn to use the RFID (Radio Frequency Identification) wireless communication technology.

Project 27.1RFID read UID

In this project, we will read the unique ID number (UID) of the RFID card, recognize the type of the RFID card and display the information through serial port.

Component List

ESP8266 x1 	USB cable 
Breadboard x1 	
Jumper wire M/M x8 	RC522 module x1 

Mifare1 S50 Standard card x1	Mifare1 S50 Non-standard card x1
	

Component knowledge

RFID

RFID (Radio Frequency Identification) is a wireless communication technology. A complete RFID system is generally composed of the responder and reader. Generally, we use tags as responders, and each tag has a unique code, which is attached to the object to identify the target object. The reader is a device for reading (or writing) tag information.

Products derived from RFID technology can be divided into three categories: passive RFID products, active RFID products and semi active RFID products. And Passive RFID products are the earliest, the most mature and most widely used products in the market among others. It can be seen everywhere in our daily life such as, the bus card, dining card, bank card, hotel access cards, etc., and all of these belong to close-range contact recognition. The main operating frequency of Passive RFID products are: 125KHZ (low frequency), 13.56MHZ (high frequency), 433MHZ (ultrahigh frequency), 915MHZ (ultrahigh frequency). Active and semi active RFID products work at higher frequencies.

The RFID module we use is a passive RFID product with the operating frequency of 13.56MHz.

MFRC522 RFID Module

The MFRC522 is a highly integrated reader/writer IC for contactless communication at 13.56MHz.
The MFRC522's internal transmitter is able to drive a reader/writer antenna designed to communicate with ISO/IEC 14443 A/MIFARE cards and transponders without additional active circuitry. The receiver module provides a robust and efficient implementation for demodulating and decoding signals from ISO/IEC 14443 A/MIFARE compatible cards and transponders. The digital module manages the complete ISO/IEC 14443A framing and error detection (parity and CRC) functionality.
This RFID Module uses MFRC522 as the control chip, and SPI (Peripheral Interface Serial) as the reserved interface.

Technical specs:

Operating Voltage	13-26mA(DC)\3.3V
Idle current	10-13mA(DC)\3.3V
Sleep current in the	<80uA
Peak current	<30mA
Operating frequency	13.56MHz
Supported card type	Mifare1 S50、Mifare1 S70、Mifare Ultralight、Mifare Pro、Mifare Desfire
Size	40mmX60mm
Operation temperature	20-80 degrees(Celsius)
Storage temperature	40-85 degrees (Celsius)
Operation humidity	5%-95%(Relative humidity)

Mifare1 S50 Card

Mifare S50 is often called Mifare Standard with the capacity of 1K bytes. And each card has a 4-bytes global unique identifier number (USN/UID), which can be rewritten 100 thousand times and read infinite times. Its storage period can last for 10 years. The ordinary Mifare1 S50 Card and non-standard Mifare1 S50 Card equipped for this kit are shown below.

The Mifare S50 capacity (1K byte) is divided into 16 sectors (Sector0-Sector15). Each sector contains 4 data block (Block0-Block3. 64 blocks of 16 sectors will be numbered according absolute address, from 0 to 63). And each block contains 16 bytes (Byte0-Byte15), 64*16=1024. As is shown in the following table:

Sector No.	Block No.	Storage area	Block type	Absolute block No.
sector 0	block 0	vendor code	vendor block	0
	block 1		data block	1
	block 2		data block	2
	block 3	Password A-access control-password B	control block	3
sector 1	block 0		data block	4
	block 1		data block	5
	block 2		data block	6
	block 3	Password A-access control-password B	control block	7
……	……	……	……	
sector 15	block 0		data block	60
	block 1		data block	61
	block 2		data block	62
	block 3	Password A-access control-password B	control block	63

Each sector has a set of independent password and access control put in its last block, that is, Block 3, which is also known as sector trailer. Sector 0, block 0 (namely absolute address 0) of S50 is used to store the card serial number and vendor code, which has been solidified and can't be changed. Except the manufacturer and the control block, the rest of the cards are data blocks, which can be used to store data. Data block can be used for two kinds of applications:

(1) used as general data storage and can be operated for reading and writing data.

(2) used as data value, and can be operated for initializing, adding, subtracting and reading the value.
The sector trailer block in each sector is the control block, including a 6-byte password A, a 4-byte access control and a 6-byte password B. For example, the control block of a brand new card is as follows:

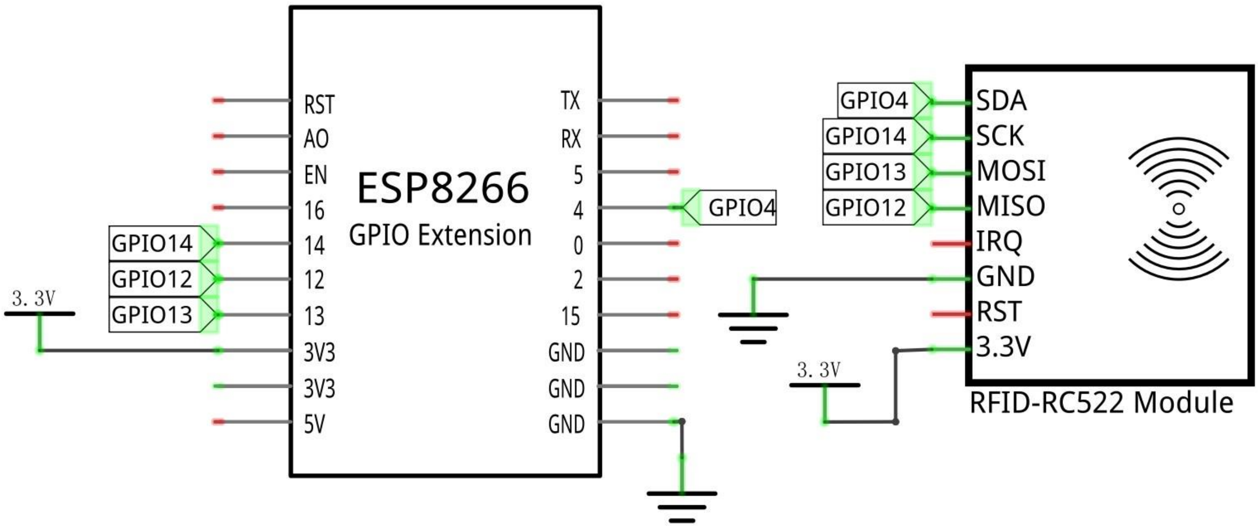
A0 A1 A2 A3 A4 A5	FF 07 80 69	B0 B1 B2 B3 B4 B5
password A	access control	password B

The default password of a brand new card is generally 0A1A2A3A4A5 for password A and B0B1B2B3B4B5 for password B, or both the password A and password B are 6 FF. Access control is used to set the access conditions for each block (including the control block itself) in a sector.
Blocks of S50 are divided into data blocks and control blocks. There are four operations, "read", "write", "add value", "subtract value (including transmission and storage)" for data blocks, and there are two operations, "read" and "write" for control blocks.
For more details about how to set data blocks and control blocks, please refer to Datasheet.
By default, after verifying password A or password B, we can do reading or writing operation to data blocks. And after verifying password A, we can do reading or writing operation to control blocks. But password A can never be read, so if you choose to verify password A but forget the password A, the block will never be able to read again. **It is highly recommended that beginners should not try to change the contents of control blocks.**
For Mifare1 S50 card equipped in Freenove RFID Kit, the default password A and B are both FFFFFFFFFFFF.

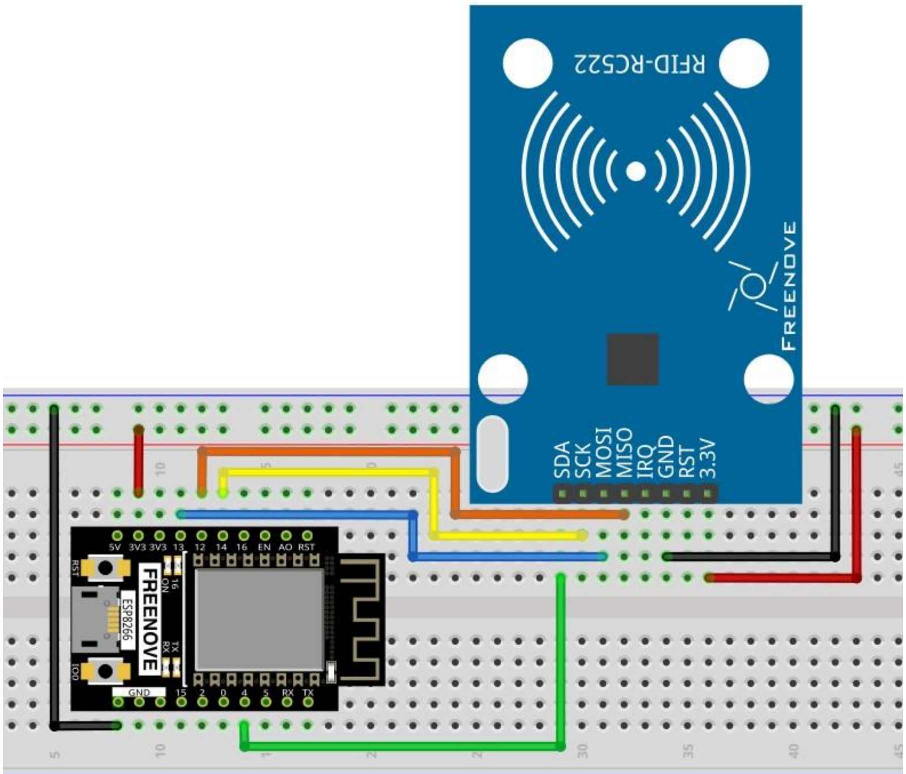
Circuit

The connection of control board and RFID module is shown below.

Schematic diagram



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



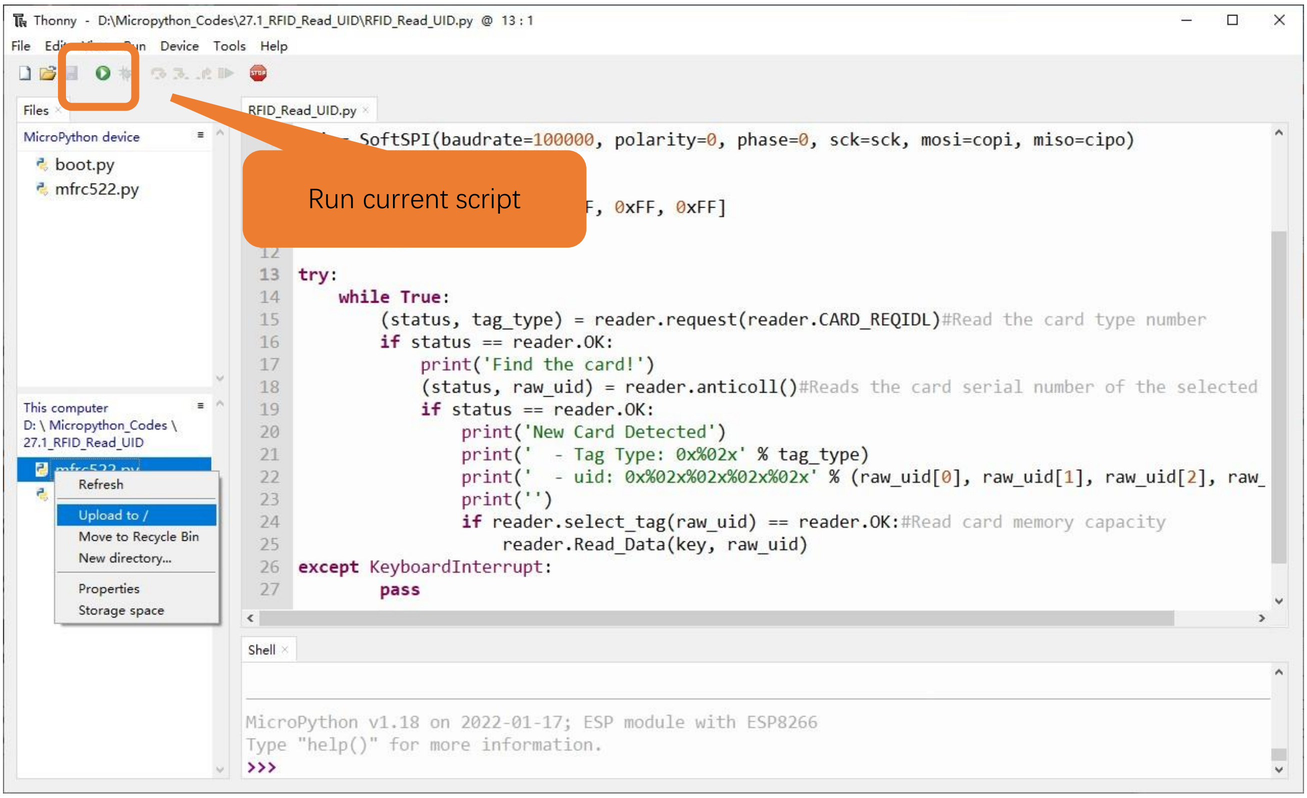
Code

In this project, we will read the unique ID number (UID) of the RFID card, recognize the type of the RFID card and display the information through serial port.

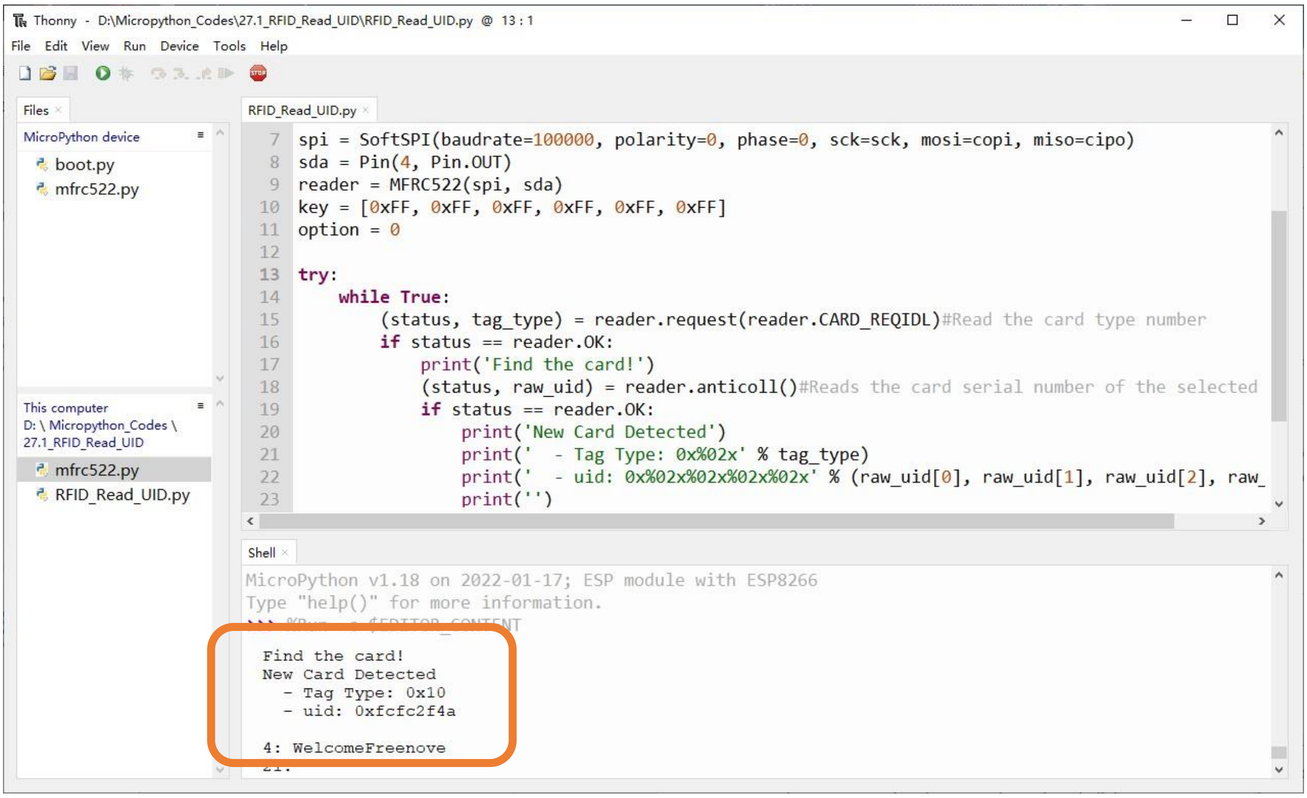
Move the program folder "**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "27.1_Infrared_Motion_Detector" and then double click"Infrared_Motion_Detector.py".
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "27.1_RFID_Read_UID". Select "mfrc522.py", right click to select "Upload to /", wait for "mfrc522.py" to be uploaded to ESP8266 and then double click "RFID_Read_UID.py".

27.1_RFID_Read_UID

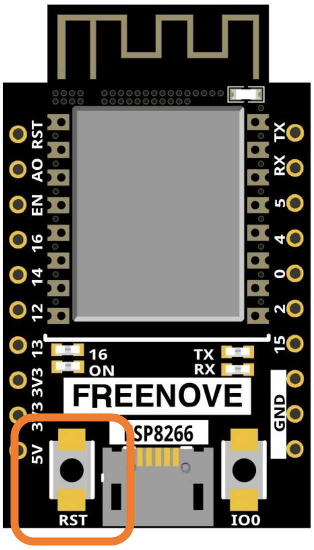


Click "Run current script", put the white or blue magnetic card near the RFID module, and Pico will print out the read information such as card type, card UID and card address data to "Shell". Press the "RST" button on the ESP8266 development board and exit the program. You can also click "Run current script" again.



If your "Shell" output is incorrect, check that the hardware connection is correct and press the "RST" button on the ESP8266 development board and click "Run current script" again.

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



The following is the program code:

```
from machine import Pin, SoftSPI
from mfrc522 import MFRC522

sck = Pin(14, Pin.OUT)
copi = Pin(13, Pin.OUT) # Controller out, peripheral in
cipo = Pin(12, Pin.OUT) # Controller in, peripheral out
spi = SoftSPI(baudrate=100000, polarity=0, phase=0, sck=sck, mosi=copi, miso=cipo)
```

```
sda = Pin(4, Pin.OUT)
reader = MFRC522(spi, sda)
key = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF]

try:
    while True:
        (status, tag_type) = reader.request(reader.CARD_REQIDL)#Read the card type number
        if status == reader.OK:
            print('Find the card!')
            (status, raw_uid) = reader.anticoll()#Reads the card serial number of the selected card
            if status == reader.OK:
                print('New Card Detected')
                print('  - Tag Type: 0x%02x' % tag_type)
                print('  - uid: 0x%02x%02x%02x%02x' % (raw_uid[0], raw_uid[1], raw_uid[2], raw_uid[3]))
                print('')
                if reader.select_tag(raw_uid) == reader.OK:#Read card memory capacity
                    reader.Read_Data(key, raw_uid)
except KeyboardInterrupt:
        pass
```

Import Pin, SoftSPI and MFRC522 modules

```
from machine import Pin, SoftSPI
from mfrc522 import MFRC522
```

Set SPI Pins and associate them with RFID-RC522 module, and then set MFRC522 module.

```
sck = Pin(14, Pin.OUT)
copi = Pin(13, Pin.OUT) # Controller out, peripheral in
cipo = Pin(12, Pin.OUT) # Controller in, peripheral out
spi = SoftSPI(baudrate=100000, polarity=0, phase=0, sck=sck, mosi=copi, miso=cipo)
sda = Pin(4, Pin.OUT)
reader = MFRC522(spi, sda)
```

Call the request() function to determine whether the RFID module detects the card. If so, read the model information of the card.

```
        (status, tag_type) = reader.request(reader.CARD_REQIDL)#Read the card type number
```

Call anticol1() function to read serial number of the selected card and save the data to the raw_uid variable.

```
            (status, raw_uid) = reader.anticoll()#Reads the card serial number of the selected card
```

Call select_tag() function to determine whether the memory information of the card can be read.

Call the auth() function to verify whether the card's password information is correct.

```
                reader.select_tag(raw_uid)
```

Read the data of the card.

```
                    reader.Read_Data(key, raw_uid)
```

The process of reading card information:

First determine whether the RFID module detects the card. If it does, read the model information and UID of the card, and then determine whether the memory information of the card is readable. If it is readable, first verify the password of the card. If the password is correct, read the address data of the card, and print out all the read information to the "Shell".

```
    while True:
        (status, tag_type) = reader.request(reader.CARD_REQIDL)#Read the card type number
        if status == reader.OK:
            print('Find the card!')
            (status, raw_uid) = reader.anticoll()#Reads the card serial number of the selected card
            if status == reader.OK:
                print('New Card Detected')
                print('  - Tag Type: 0x%02x' % tag_type)
                print('  - uid: 0x%02x%02x%02x%02x' % (raw_uid[0], raw_uid[1], raw_uid[2], raw_uid[3]))
                print('')
                if reader.select_tag(raw_uid) == reader.OK:#Read card memory capacity
                    reader.Read_Data(key, raw_uid)
```

Reference

Class MFRC 522()

Before each use of mfrcC522 module, please add the statement "**from mfrc522 import MFRC522**" to the top of the python file.

MFRC522(spi, cs): Creat an object MFRC522.

spi: imstance of the SoftSPI class.

cs: Chip select.

Default variables in the MFRC522 class:

MFRC522.OK = 0

MFRC522.NO_TAG_ERR = 1

MFRC522.ERR = 2

MFRC522.CARD_REQIDL = 0x26

MFRC522.AUTH = 0x60

MFRC522.init(): Initialize MFRC522 module.

MFRC522.reset(): Reset MFRC522 module.

MFRC522.request(mode): Request to read card data. The function returns a status code and an int. The status code is used to check whether a card is being read. The data of the int type is the model data of the card.

mode: CARD_REQIDL = 0x26.

MFRC522.anticol1(): Read the serial number of the card. The function returns a list and a status code. The data in the list is the serial number of the card.

MFRC522.select_tag(): Check whether the memory information of the card can be read.

MFRC522.auth(mode, addr, sect, serial_number): The function is used to verify the card password, and returns a boolean value. If it is correct, it returns True, if it is wrong, it returns False.

mode: AUTH = 0x60.

addr: data address.

sect: password of the card.

serial_number: serial number of the card.
MFRC522.read(addr): read data in addr address.
MFRC522.write(addr, data): Function to write card data. addr is the address of the data to be written and data is the data to be written. Returns a status code.
MFRC522.Read_Data(sect, serial_number): Read all the data written to the card and print it out.
sect: password of the card.
serial_number: serial number of the card.
MFRC522.Write_Data(sect, serial_number, datas): Write datas data into the card, the original data in the card will be overwritten.
MFRC522.Clear_Data(sect, serial_number): Clear the data written to the card.
MFRC522.Write_Row_Data(sect, serial_number, addr, datas): Write the datas data to the addr address.
MFRC522.Read_Row_Data(sect, serial_number, addr): Read the data at the addr address and print it out.
MFRC522.Clear_Row_Data(sect, serial_number, addr): Clear the data of the addr address.

Class SoftSPI

Before each use of SoftSPI module, please add the statement "**from machine import SoftSPI**" to the top of the python file.
Machine.SoftSPI(id, baudrate, polarity, phase, bits, sck, mosi, miso): construct a new software SPI object. Additional parameters must be give, usually at least id, baudrate, sck, mosi, miso.
SoftSPI.init(baudrate, polarity, phase, bits, sck, mosi, miso):
Baudrate: the SCK clock rate.
Polarity: It is set as 0 or 1, which represents the level of the clock line when it is idle.
Phase: It is set as 0 or 1, to sample data on the first or second clock edge, respectively.
sck, mosi, miso: Corresponding pins of SPI bus in ESP8266.
SoftSPI.deinit(): Close the SoftSPI bus.
SoftSPI.read(nbytes, write=0): Read a number of bytes specified by nbytes while continuously writing the single byte given by write. Returns a bytes object with the data that was read.
SoftSPI.readinto(buf, write=0): Read into the buffer specified by buf while continuously writing the single byte given by write. Returns None.
SoftSPI.write(buf): Write the bytes contained in buf. Returns None.
SoftSPI.write_readinto(write_buf, read_buf): Write the bytes from write_buf while reading into read_buf. The buffers can be the same or different, but both buffers must have the same length. Returns None.

Project 27.2RFID Read and Write

In this project, we will do reading and writing operations to the card.

Component List

Same with last section.

Circuit

Same with last section.

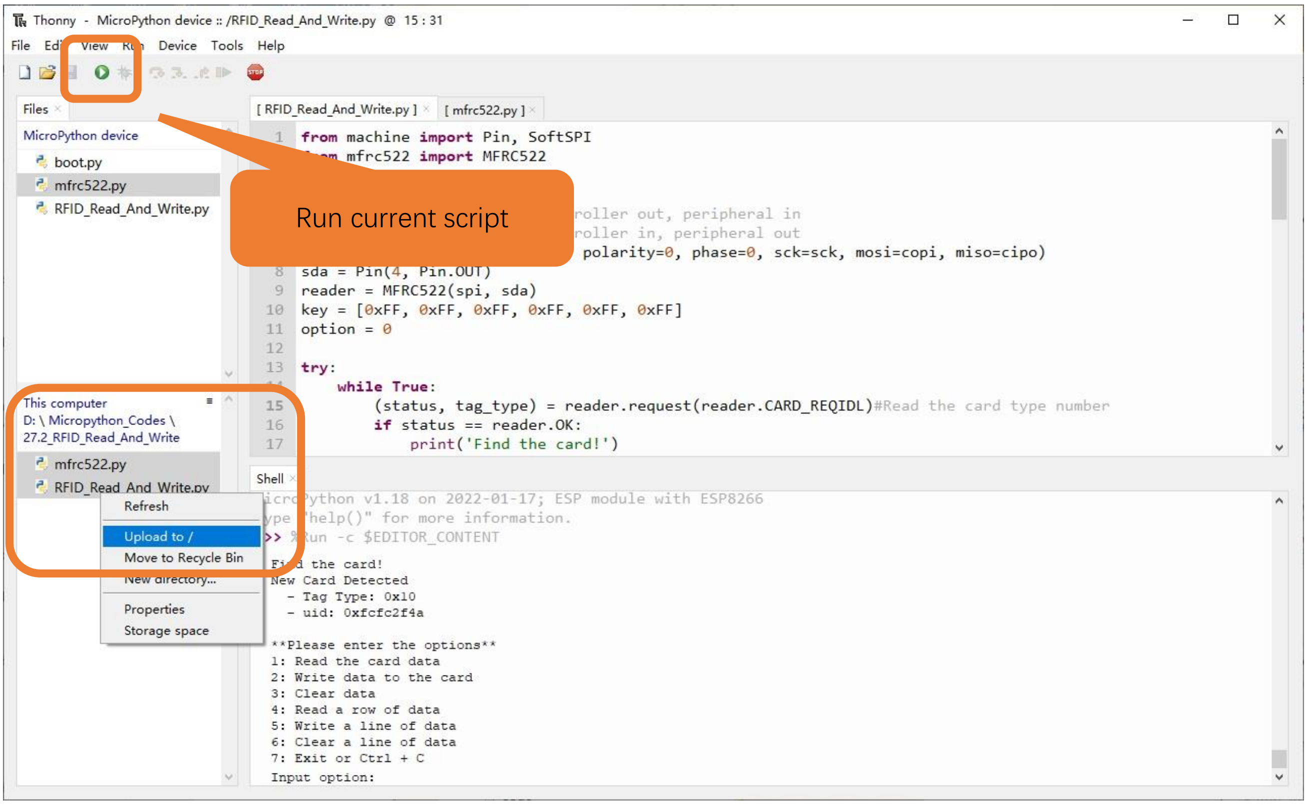
Code

In this project, we will do reading and writing operations to the card.

Move the program folder "**Freenove_Ultimate_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" → "27.2_RFID_Read_And_Write". Select"mfrc522.py", right click to select "Upload to /", wait for "mfrc522.py" to be uploaded to ESP8266 and then double click "RFID_Read_And_Write.py".

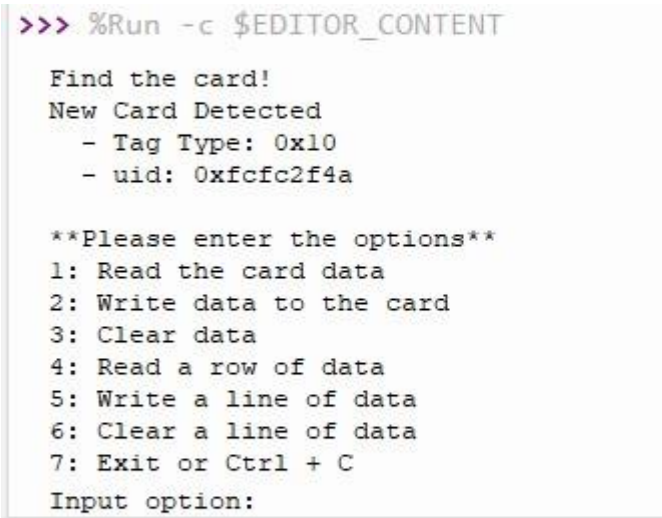
27.2_RFID_Read_UID



Click "Run current script", put the white magnetic card or blue magnetic card close to the induction area of the RFID module, the "Shell" window prints the read information such as card UID data.

It also provides the option to perform regular read and write operations on the card. Press Ctrl+C or click "Stop/Restart backend" to exit the program.

In the experiment, it is recommended to place the white or blue magnetic card above the sensor area of the RFID module, and then remove it after the experiment. This is conducive to relevant operations and prevent the occurrence of false alarms.



For example: Option 1 is to read all the data of the card.

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

 Find the card!
 New Card Detected
   - Tag Type: 0x10
   - uid: 0xfcfc2f4a

 **Please enter the options**
 1: Read the card data
 2: Write data to the card
 3: Clear data
 4: Read a row of data
 5: Write a line of data
 6: Clear a line of data
 7: Exit or Ctrl + C
 Input option: 1

 1: WelcomeFreenove!
 reading end
```

Line number

Data stored in the card

Option2 is to write data to the card.

```
3: Clear data
4: Read a row of data
5: Write a line of data
6: Clear a line of data
7: Exit or Ctrl + C
Input option: 2

plase input datas: Welcome
Write card OK!

**Please enter the options**
1: Read the card data
2: Write data to the card
3: Clear data
4: Read a row of data
5: Write a line of data
6: Clear a line of data
7: Exit or Ctrl + C
Input option: 1

1: Welcome
reading end
```

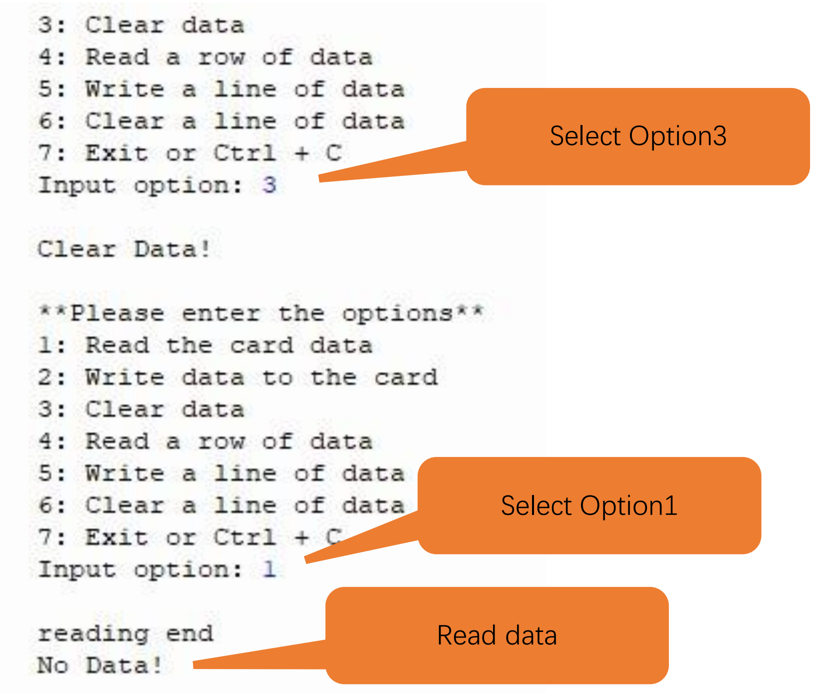
Select Option2

Input datas

Select Option1

Read the data

Option3 is to clear data.



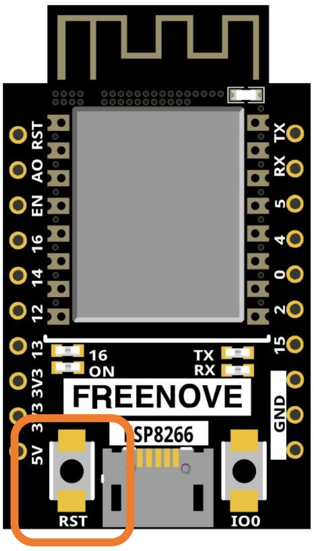
```
3: Clear data
4: Read a row of data
5: Write a line of data
6: Clear a line of data
7: Exit or Ctrl + C
Input option: 3

Clear Data!

**Please enter the options**
1: Read the card data
2: Write data to the card
3: Clear data
4: Read a row of data
5: Write a line of data
6: Clear a line of data
7: Exit or Ctrl + C
Input option: 1

reading end
No Data!
```


If your "Shell" output is incorrect or the output error alert, please check that the hardware connection is correct and press the "RST" button on the ESP8266 development board and click "Run current script" again. If you need any support, please feel free to contact us via: support@freenove.com



The following is the program code:

```
from machine import Pin, SoftSPI
from mfrc522 import MFRC522

sck = Pin(14, Pin.OUT)
copi = Pin(13, Pin.OUT) # Controller out, peripheral in
cipo = Pin(12, Pin.OUT) # Controller in, peripheral out
spi = SoftSPI(baudrate=100000, polarity=0, phase=0, sck=sck, mosi=copi, miso=cipo)
sda = Pin(4, Pin.OUT)
```

```
reader = MFRC522(spi, sda)
key = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF]
option = 0

try:
    while True:
        (status, tag_type) = reader.request(reader.CARD_REQIDL)#Read the card type number
        if status == reader.OK:
            print('Find the card!')
            (status, raw_uid) = reader.anticoll()#Reads the card serial number of the selected card
            if status == reader.OK:
                print('New Card Detected')
                print('  - Tag Type: 0x%02x' % tag_type)
                print('  - uid: 0x%02x%02x%02x%02x' % (raw_uid[0], raw_uid[1], raw_uid[2], raw_uid[3]))
                print('')
                if reader.select_tag(raw_uid) == reader.OK:#Read card memory capacity
                    while option != "7":
                        print("**Please enter the options**")
                        print("1: Read the card data")
                        print("2: Write data to the card")
                        print("3: Clear data")
                        print("4: Read a row of data")
                        print("5: Write a line of data")
                        print("6: Clear a line of data")
                        print("7: Exit or Ctrl + C")
                        option = input("Input option: ")
                        print("")
                        if option == "1":
                            reader.Read_Data(key, raw_uid)
                        elif option == "2":
                            datas = input("plase input datas: ")
                            reader.Write_Data(key, raw_uid, datas)
                        elif option == "3":
                            reader.Clear_Data(key, raw_uid)
                        elif option == "4":
                            row_index = int(input("Please enter the row index: "))
                            if (row_index+1)%4 == 0 or row_index == 0 or row_index > 63:
                                print("\nPermission Denied")
                            if reader.auth(0x60, row_index, key, raw_uid) == reader.OK: #Verification card password
                                if bytearray(reader.read(row_index)) == bytearray(16):
                                    print("No Data")
```

```
                                else:
                                    print(str(row_index) + ": " + str(bytearray(reader.read(row_index)).decode()))
                            else:
                                print("ERROR")
                        elif option == "5":
                            row_index = int(input("Please enter the row index: "))
                            datas = input("plase input datas: ")
                            reader.Write_Row_Data(key, raw_uid, row_index, datas)
                        elif option == "6":
                            row_index = int(input("Please enter the row index: "))
                            reader.Clear_Row_Data(key, raw_uid, row_index)
                        elif option == "7":
                            reader.stop_crypto1()
                            print("Exit the success")
                        else:
                            print("Input error")
                        print("")
                    break
except KeyboardInterrupt:
    pass
```

Print the function of each option, use the option variable to receive the options input by the user.

```
                        print("**Please enter the options**")
                        print("1: Read the card data")
                        print("2: Write data to the card")
                        print("3: Clear data")
                        print("4: Read a row of data")
                        print("5: Write a line of data")
                        print("6: Clear a line of data")
                        print("7: Exit or Ctrl + C")
                        option = input("Input option: ")
```

Read_Data() function reads data in the card (excluding the password).

```
                            reader.Read_Data(key, raw_uid)
```

Call Write_Data() to continuously write data to the card. Call the function again will overwrite data previously input.

```
                            reader.Write_Data(key, raw_uid, datas)
```

Clear_Data() function is used to clear all input data in the card.

```
                            reader.Clear_Data(key, raw_uid)
```

Call Read_Row_Data() function to read data in designated address.

```
                            if (row_index+1)%4 == 0 or row_index == 0 or row_index > 63:
                                print("\nPermission Denied")
                            if reader.auth(0x60, row_index, key, raw_uid) == reader.OK: #Verification card password
                                if bytearray(reader.read(row_index)) == bytearray(16):
```

```
                print("No Data")
            else:
                print(str(row_index) + ": " +
str(bytearray(reader.read(row_index)).decode()))
        else:
            print("ERROR")
```

Write_Row_Data() function writes data to designated address.

```
        reader.Write_Row_Data(key, raw_uid, row_index, datas)
```

Clear_Row_Data() clears data of designated address.

```
        reader.Clear_Row_Data(key, raw_uid, row_index)
```

Based on the prompt, users can input option in "Shell" to read, write, or erase data on the card, and output the result in the "Shell". You can exit the program by inputting 7 or pressing Ctrl + C.

```
while option != "7":
    print("**Please enter the options**")
    print("1: Read the card data")
    print("2: Write data to the card")
    print("3: Clear data")
    print("4: Read a row of data")
    print("5: Write a line of data")
    print("6: Clear a line of data")
    print("7: Exit or Ctrl + C")
    option = input("Input option: ")
    print("")
    if option == "1":
        reader.Read_Data(key, raw_uid)
    elif option == "2":
        datas = input("plase input datas: ")
        reader.Write_Data(key, raw_uid, datas)
    elif option == "3":
        reader.Clear_Data(key, raw_uid)
    elif option == "4":
        row_index = int(input("Please enter the row index: "))
        if (row_index+1)%4 == 0 or row_index == 0 or row_index > 63:
            print("\nPermission Denied")
        if reader.auth(0x60, row_index, key, raw_uid) == reader.OK:
#Verification card password
            if bytearray(reader.read(row_index)) == bytearray(16):
                print("No Data")
            else:
                print(str(row_index) + ": " +
str(bytearray(reader.read(row_index)).decode()))
        else:
            print("ERROR")
    elif option == "5":
```

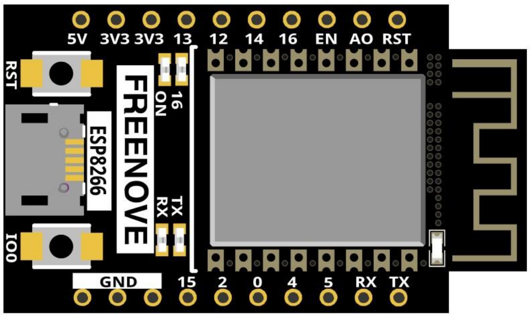
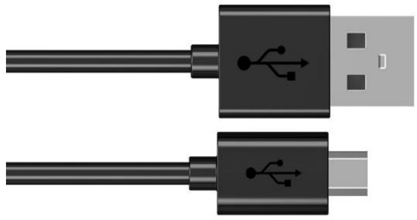
```
                    row_index = int(input("Please enter the row index: "))
                    datas = input("plase input datas: ")
                    reader.Write_Row_Data(key, raw_uid, row_index, datas)
                elif option == "6":
                    row_index = int(input("Please enter the row index: "))
                    reader.Clear_Row_Data(key, raw_uid, row_index)
                elif option == "7":
                    reader.stop_crypto1()
                    print("Exit the success")
                else:
                    print("Input error")
                print("")
            break
```

Chapter 28 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 28.1 Station mode

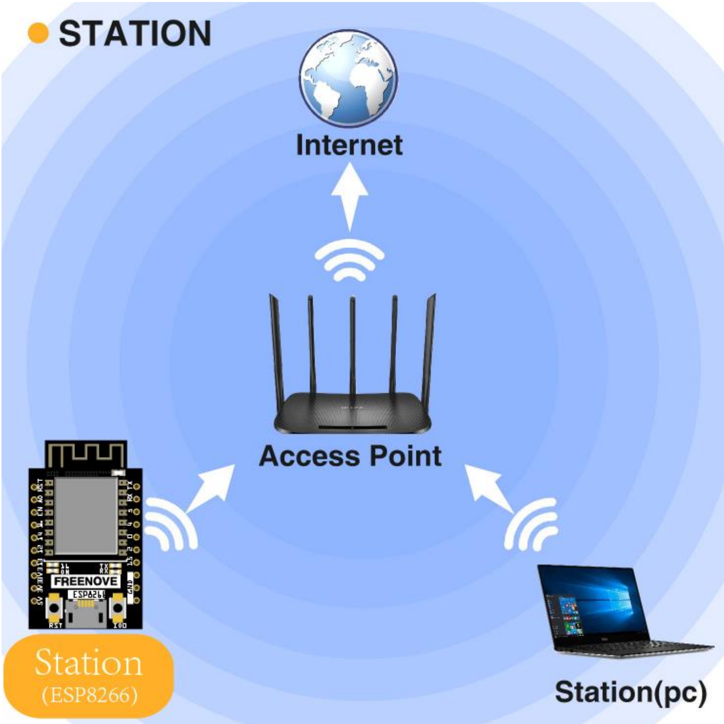
Component List

ESP8266 x1	USB cable
	

Component knowledge

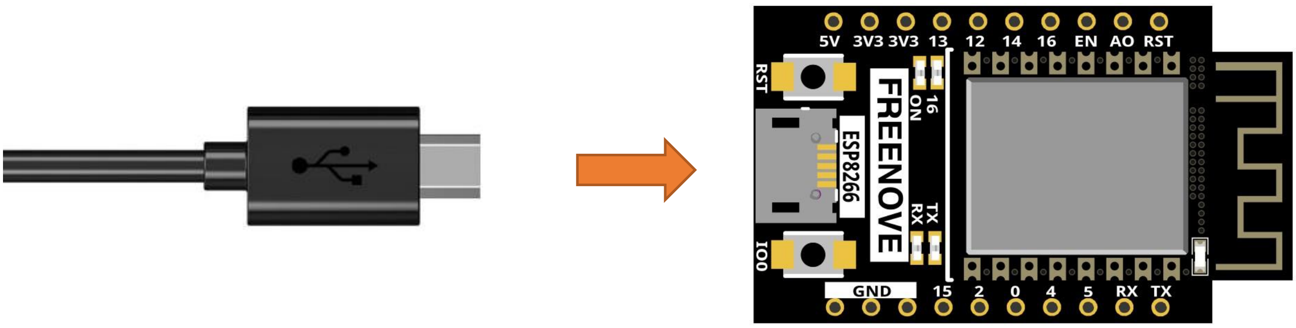
Station mode

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.

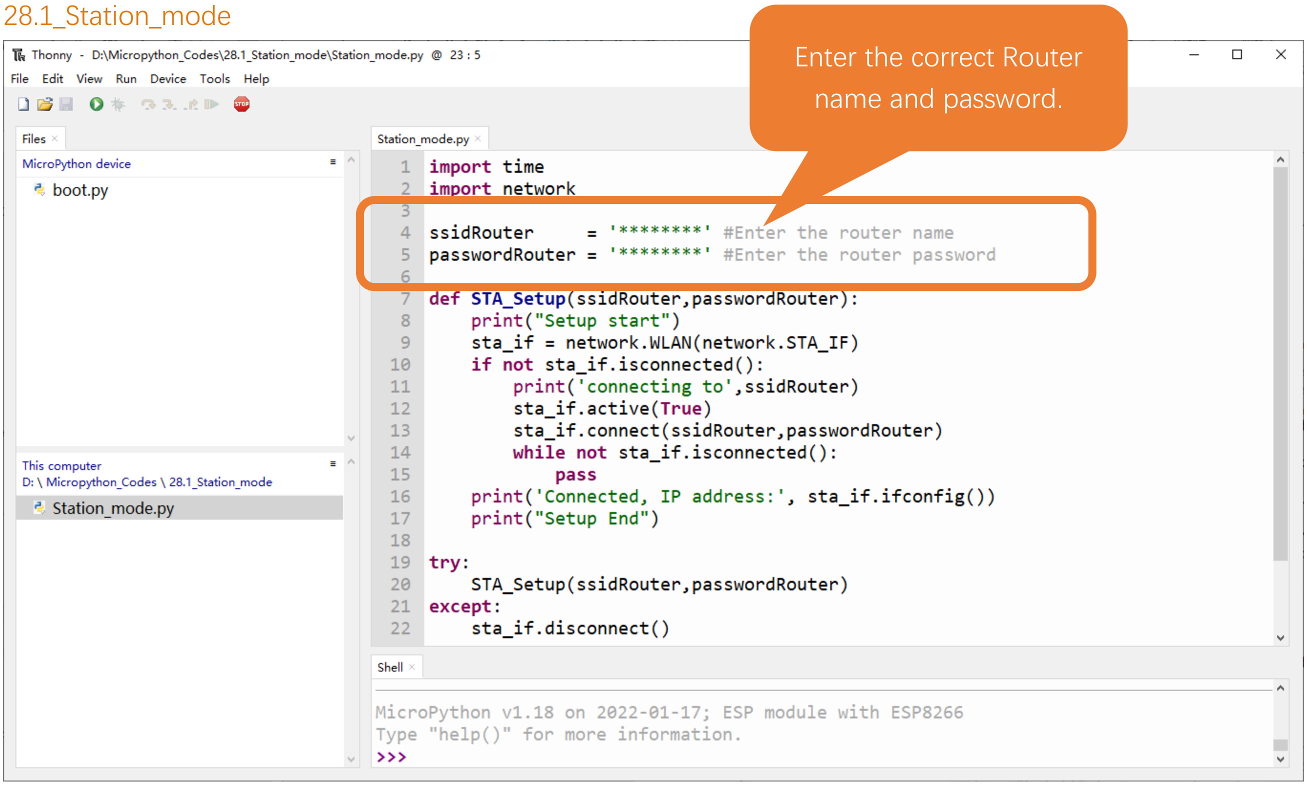


Code

Move the program folder "**Freenove_Ultimate_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:

```
import time
import network

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

def STA_Setup(ssidRouter,passwordRouter):
    print("Setup start")
    sta_if = network.WLAN(network.STA_IF)
    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")

try:
    STA_Setup(ssidRouter,passwordRouter)
except:
    sta_if.disconnect()
```

Import network module.

```
import network
```

Enter correct router name and password.

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

Set ESP8266 in Station mode.

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

Activate ESP8288 Station mode, initiate a connection request to the router and enter the password to connect.

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

Wait for ESP8266 to connect to router until they connect to each other successfully.

```
        while not sta_if.isconnected():
            pass
```

Print the IP address assigned to ESP8266 in "Shell".

```
    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 28.2 AP mode

Component List & Circuit

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

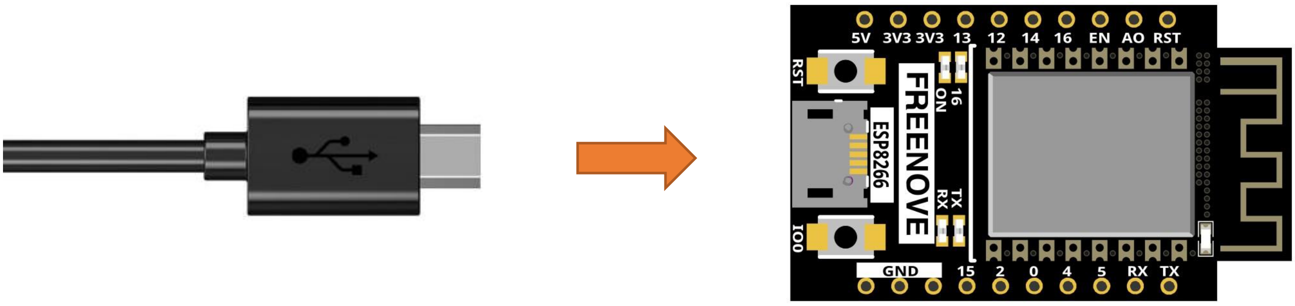
Component knowledge

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.

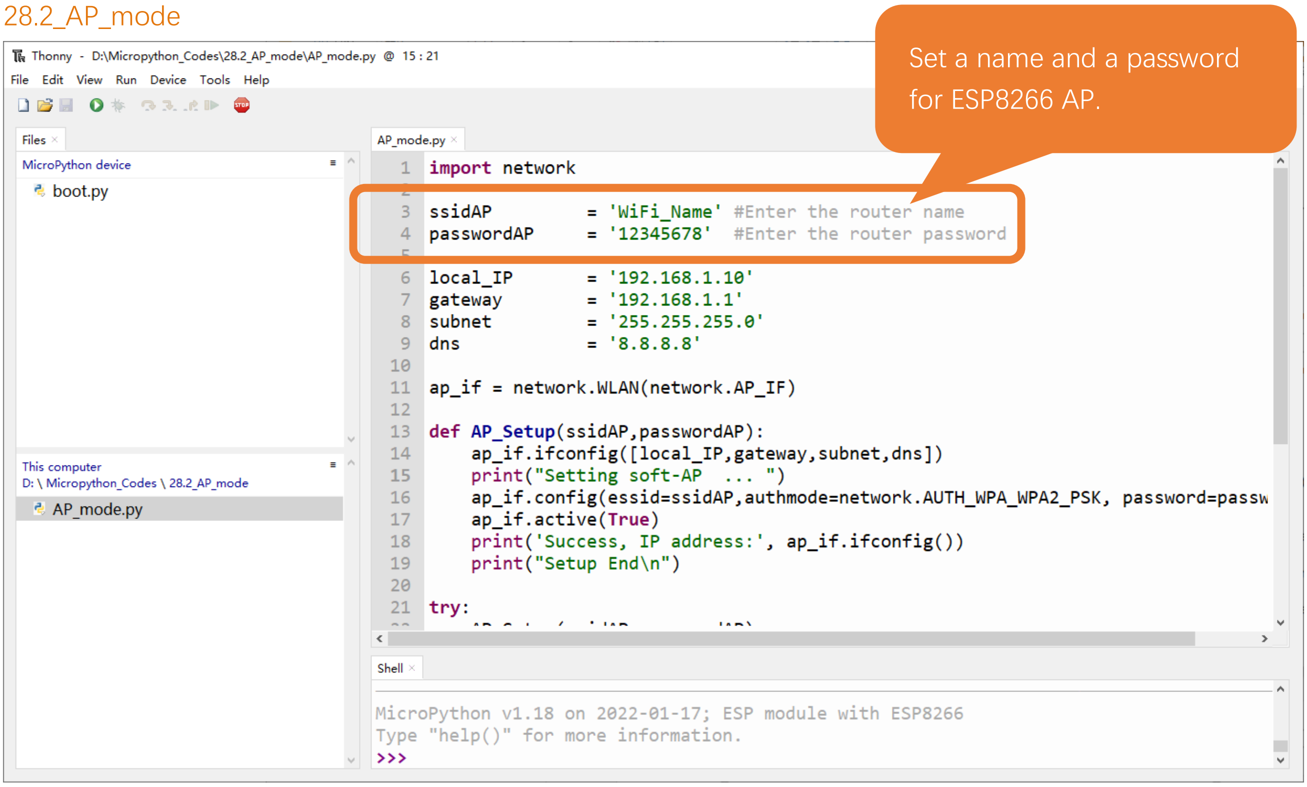


Code

Move the program folder "**Freenove_Ultimate_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.

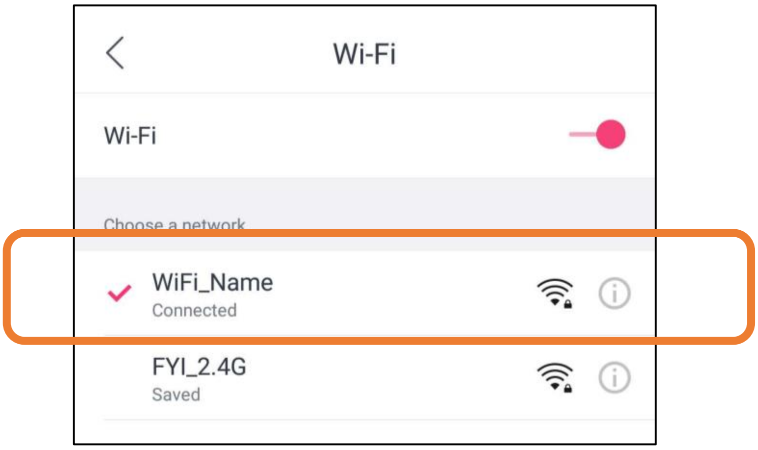
Click "Run current script", open the AP function of ESP8266 and print the access point information.

```
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: ('192.168.1.10', '192.168.1.1', '255.255.255.0', '8.8.8.8')
  Setup End

>>>
```

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

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

Set ESP8266 in AP mode.

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

Configure IP address, gateway and subnet mask for ESP8266.

```
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.

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

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

```
    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

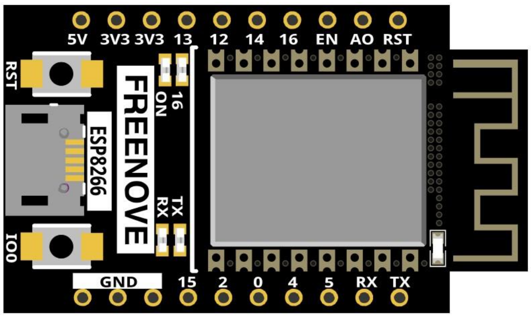
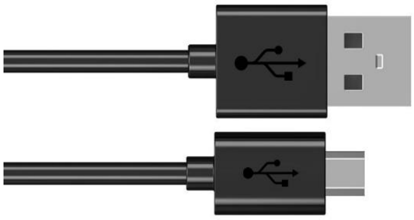
DNS_server: DNSserver

disconnect(): Disconnect from the currently connected wireless network

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

Project 28.3 AP+Station mode

Component List

ESP8266 x1	Micro USB Wire x1
	

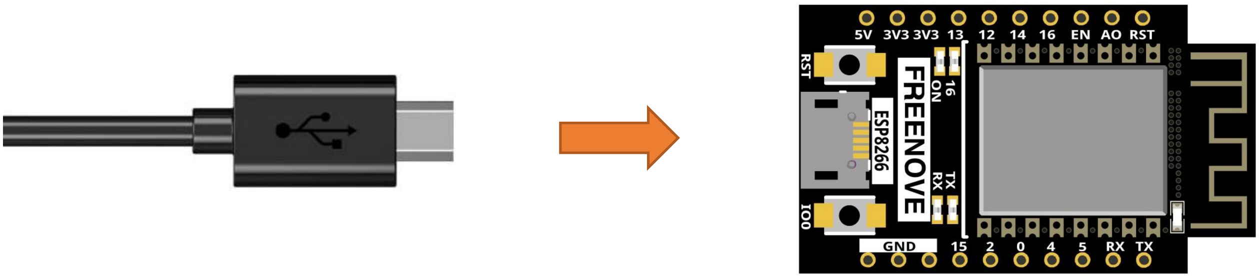
Component knowledge

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.

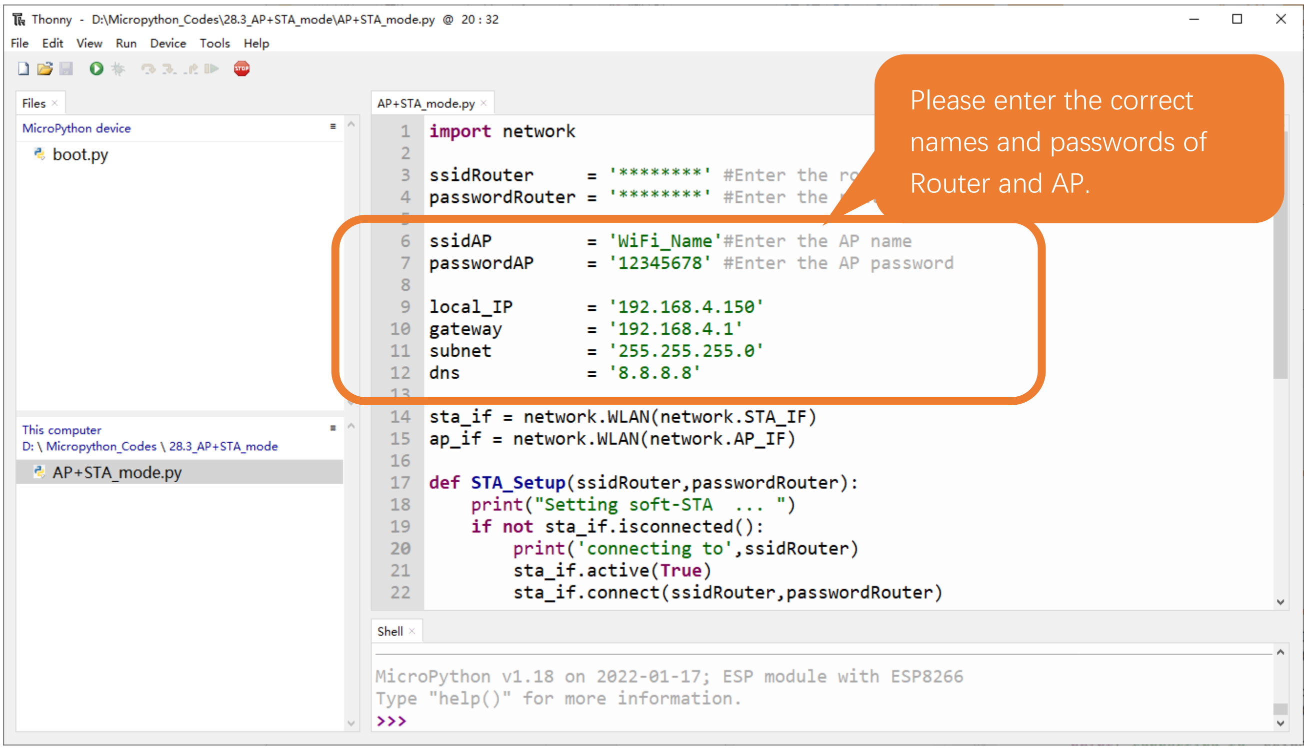


Code

Move the program folder "**Freenove_Ultimate_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 28.1 and project 28.2. 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:

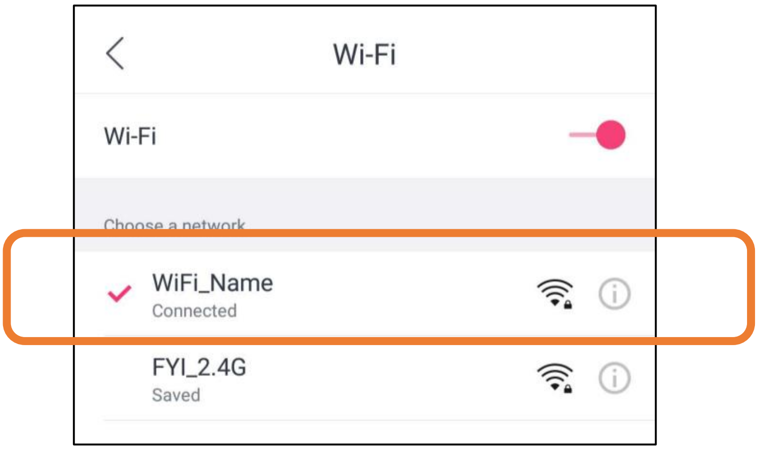
```
Shell
MicroPython v1.18 on 2022-01-17; ESP module with ESP8266
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

 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  ... ")
```

```
    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)
    STA_Setup(ssidRouter,passwordRouter)
except:
    sta_if.disconnect()
    ap_if.idsconnect()
```

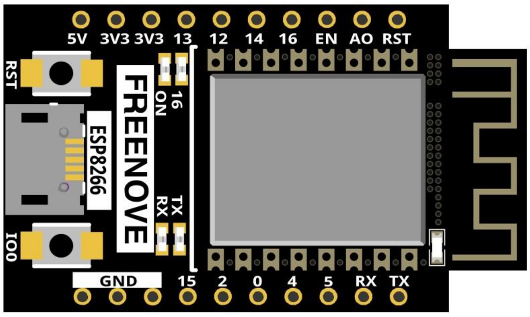
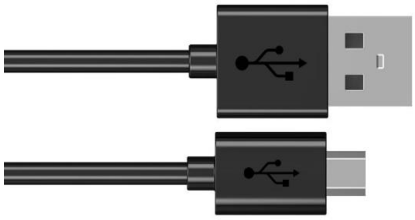
Chapter 29 TCP/IP

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 29.1 As Client

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

Component List

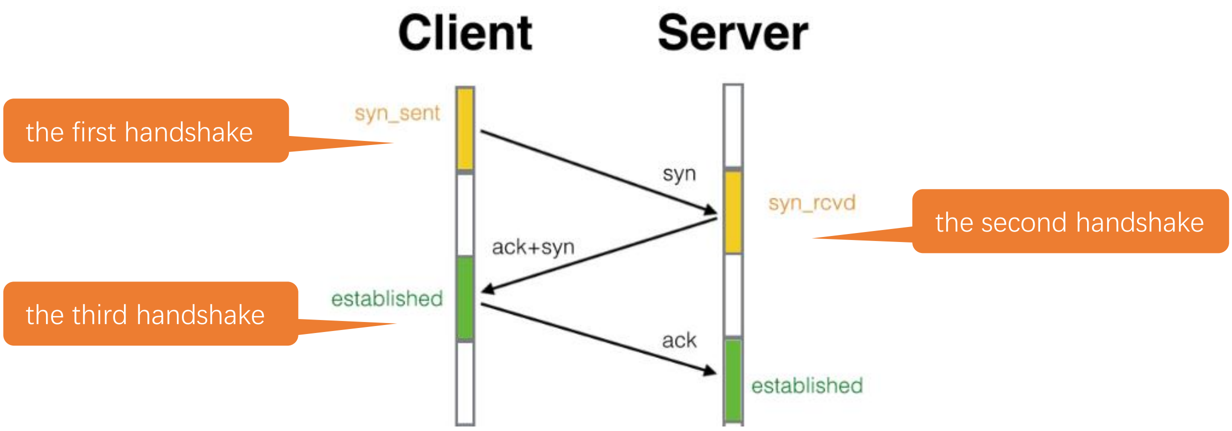
ESP8266 x1	USB cable
	

Component knowledge

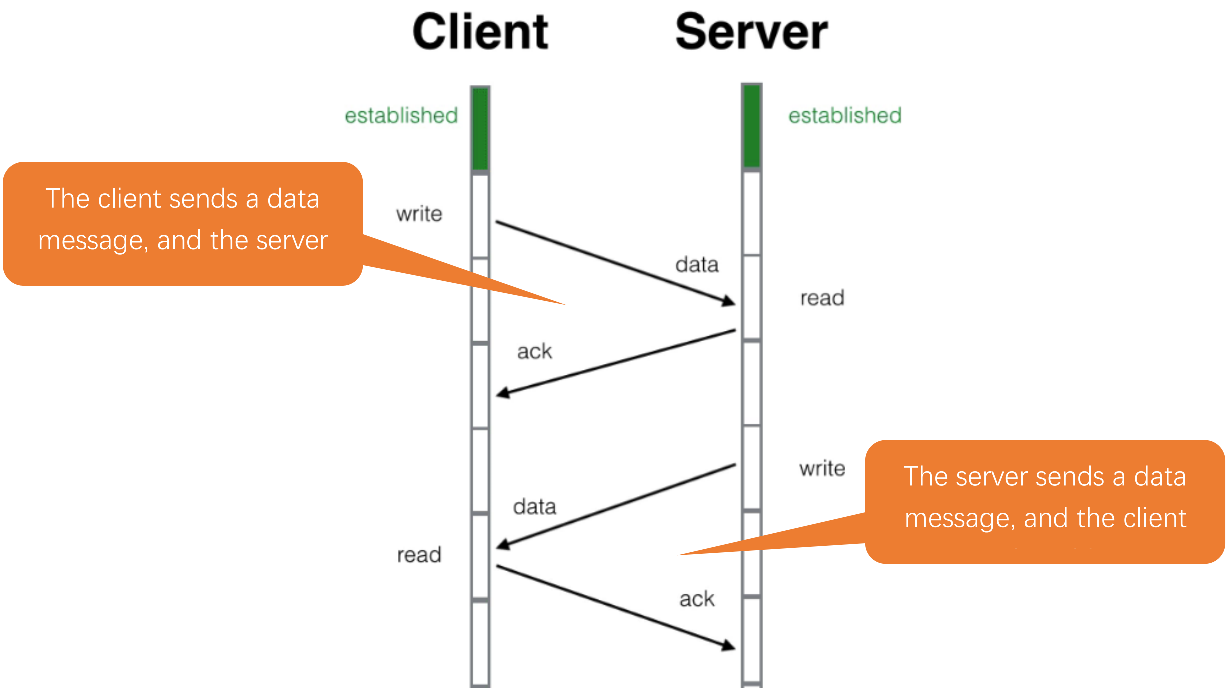
TCP connection

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.

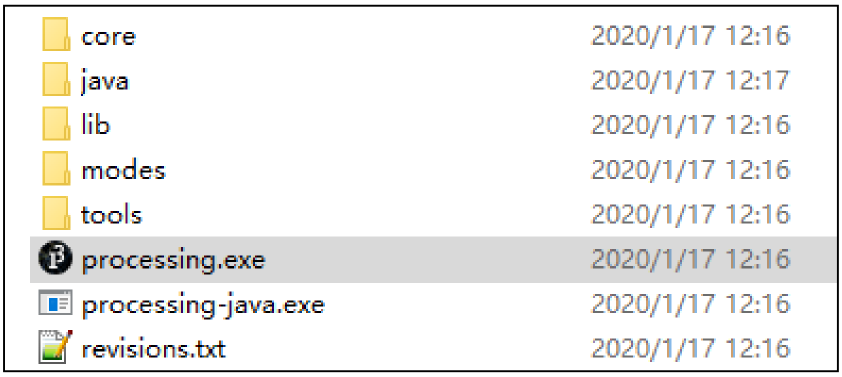


Install Processing

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.

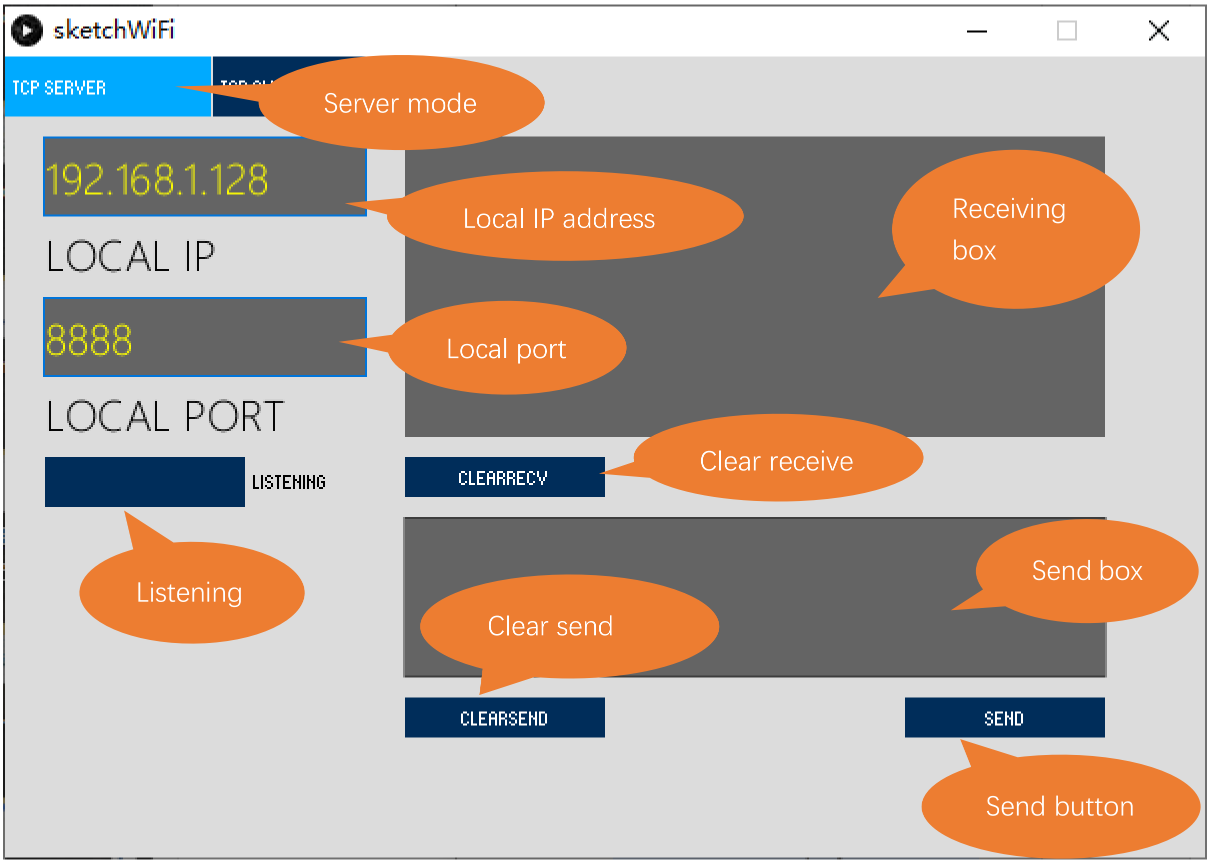
Unzip the downloaded file to your computer. Click "processing.exe" as the figure below to run this software.



Use Server mode for communication

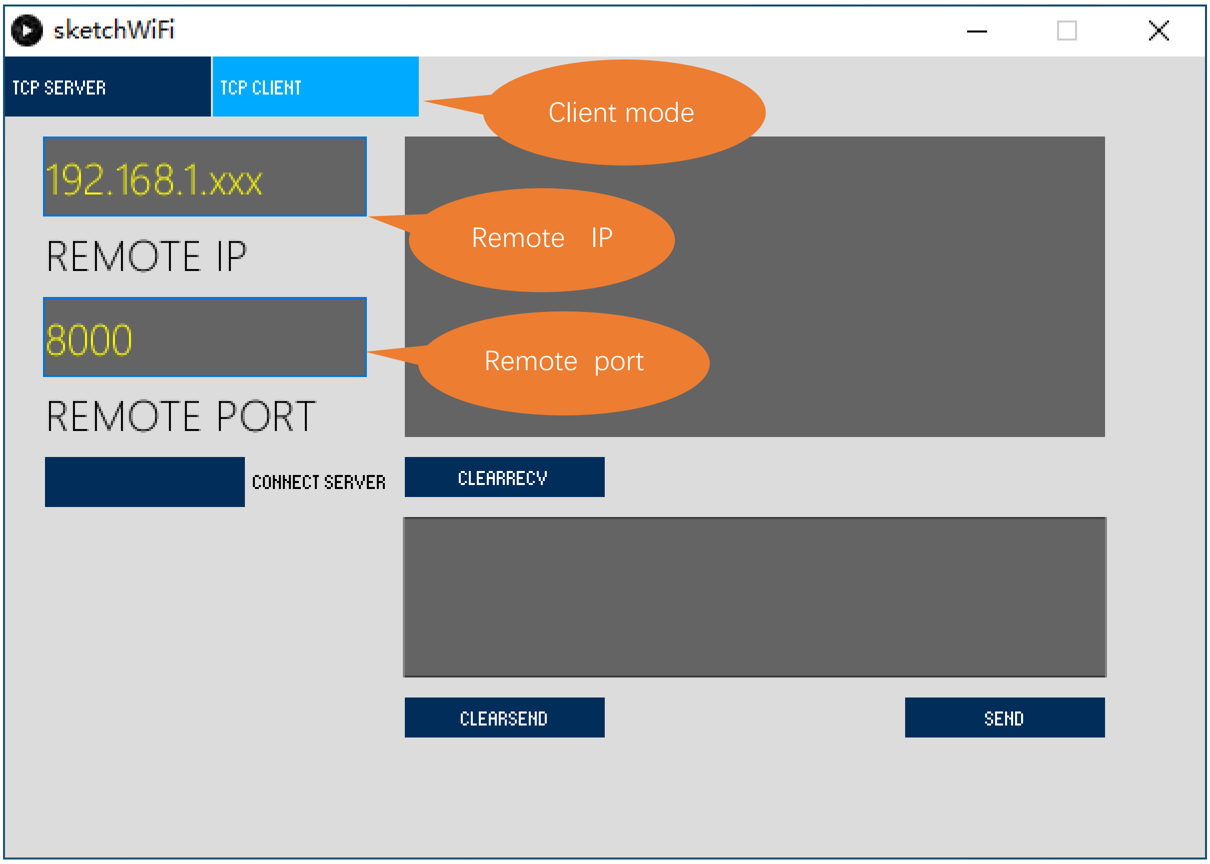
Open the "**Freenove_Ultimate_Starter_Kit_for_ESP8266/Codes/Micropython_Codes/29.1_TCP_as_Client/sketchWiFi/sketchWiFi.pde**". Click "Run".

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.

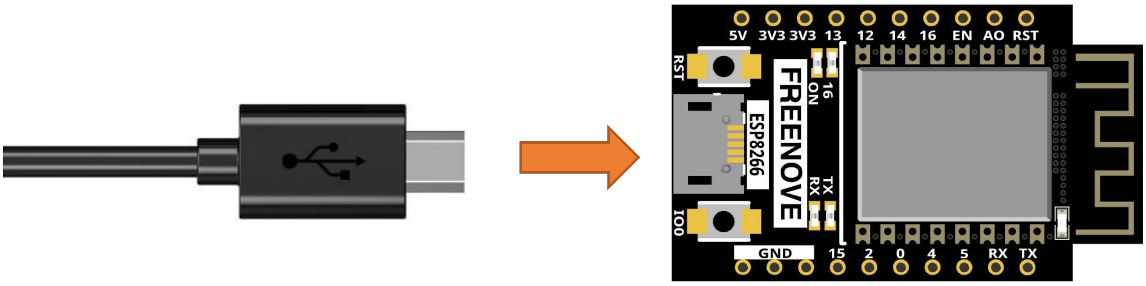
clear receive: clear out the content in the receiving text box

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

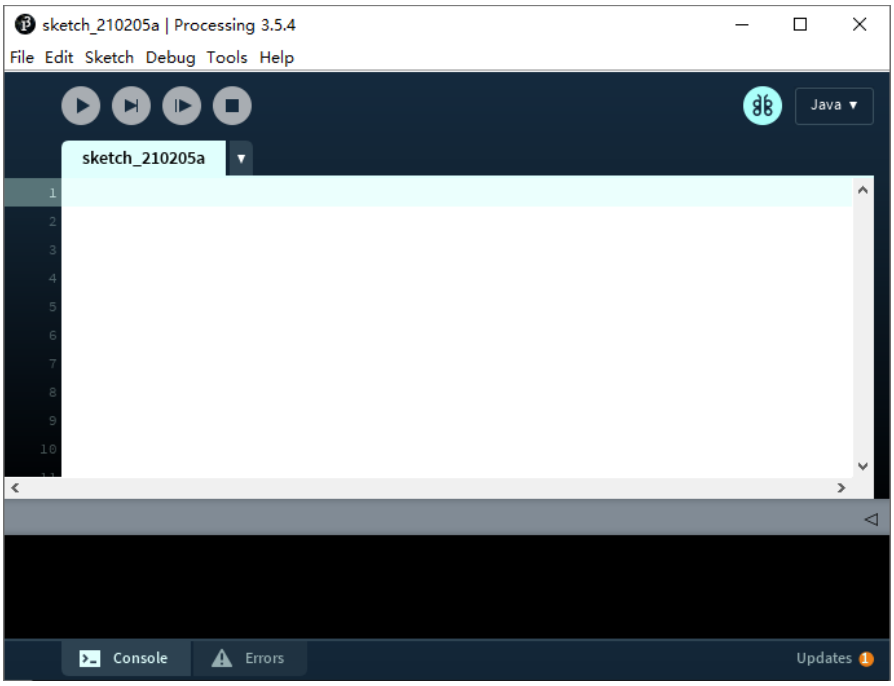
Connect Freenove ESP8266 to the computer using USB cable.



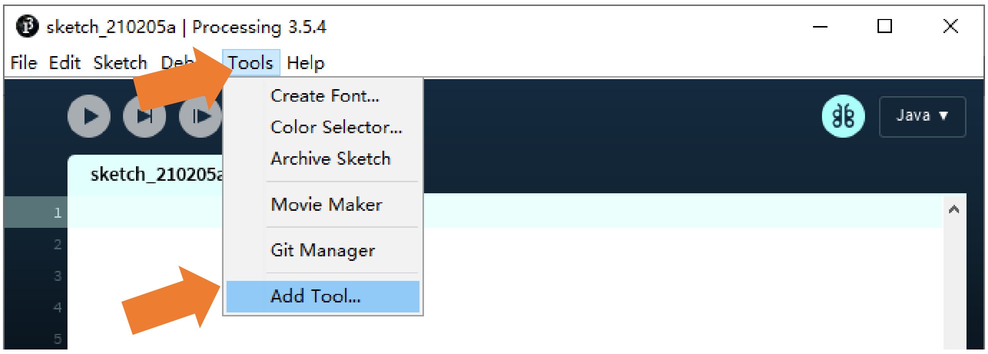
Code

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

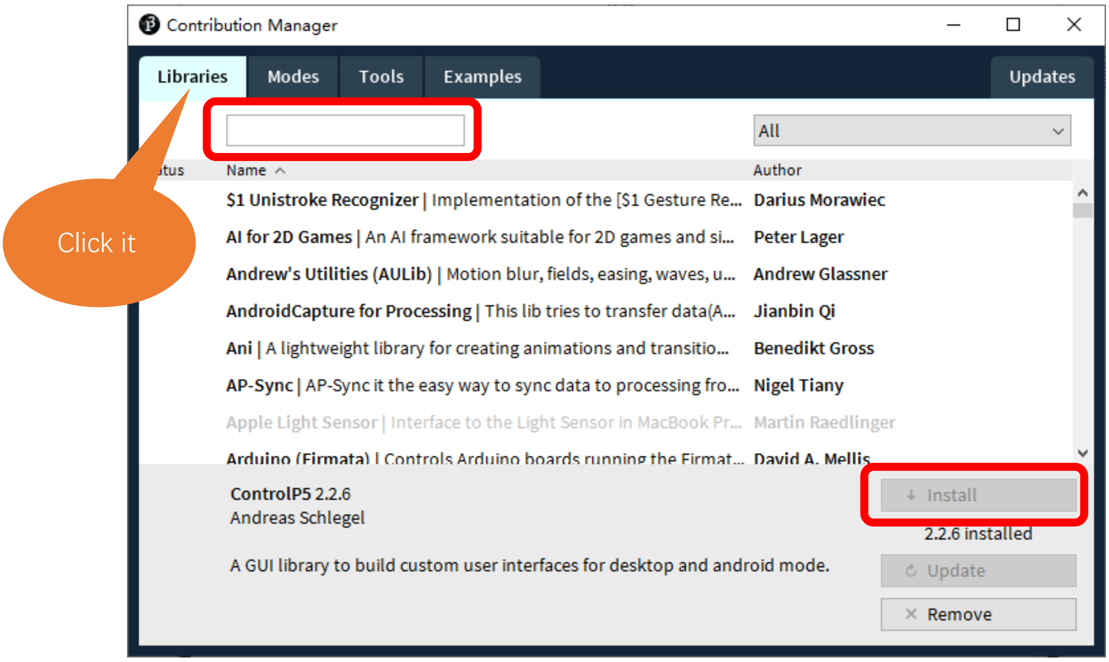
Open Processing.



Click Add Tool under Tools.

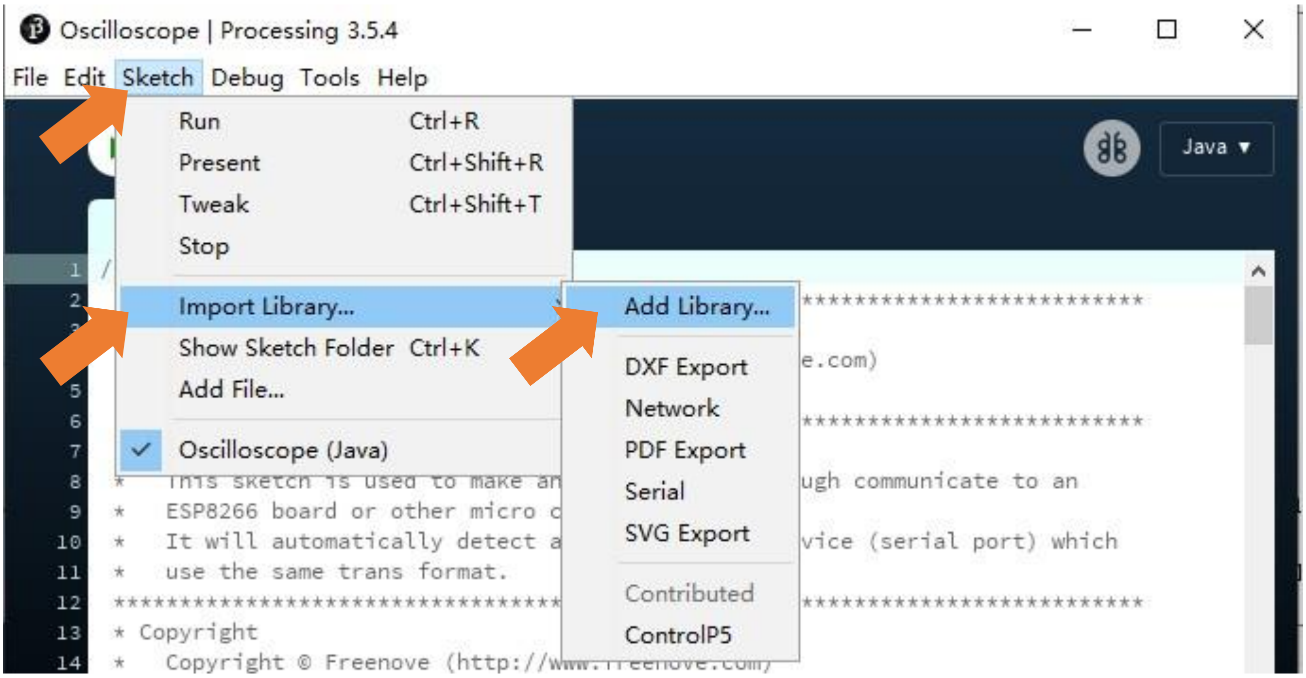


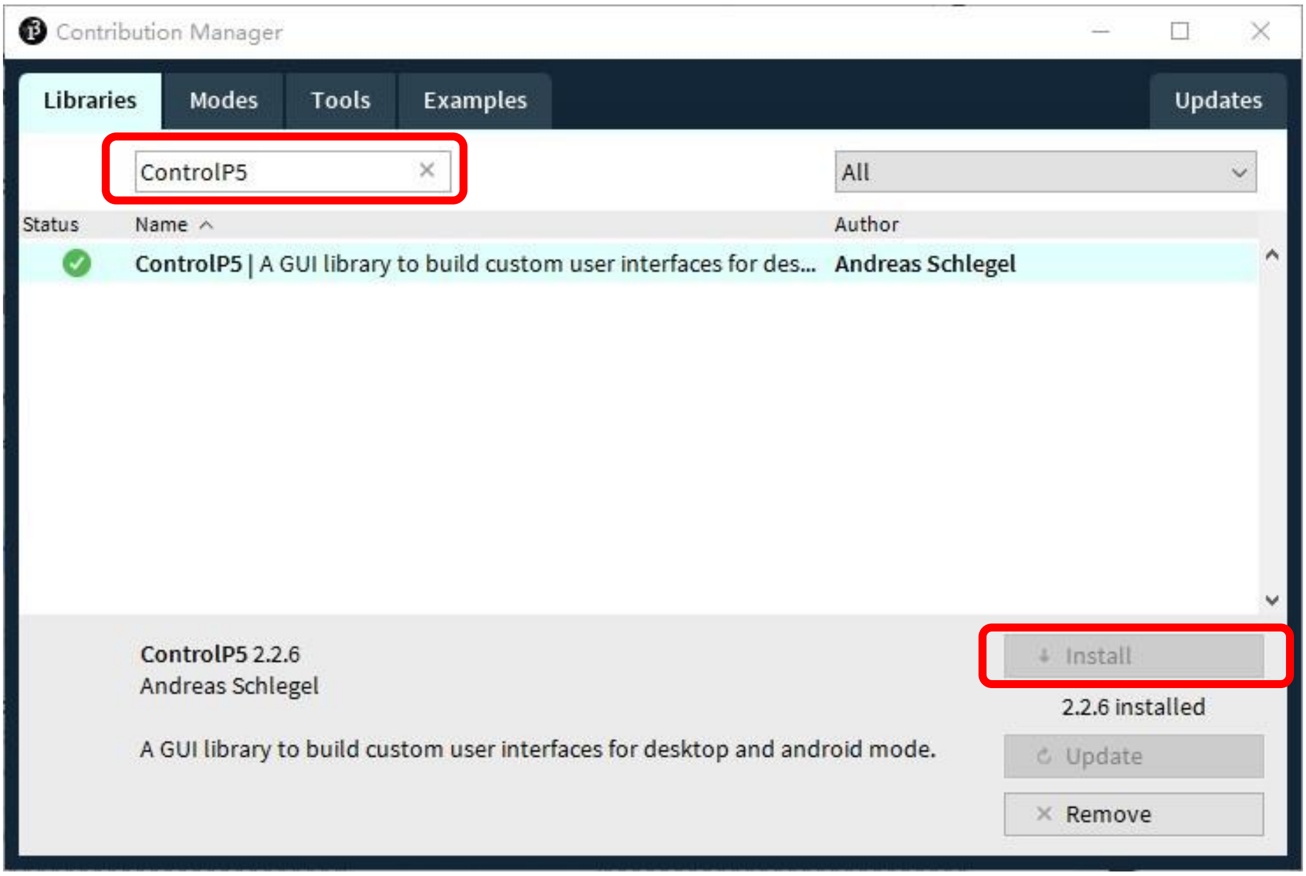
Select Libraries in the pop-up window.



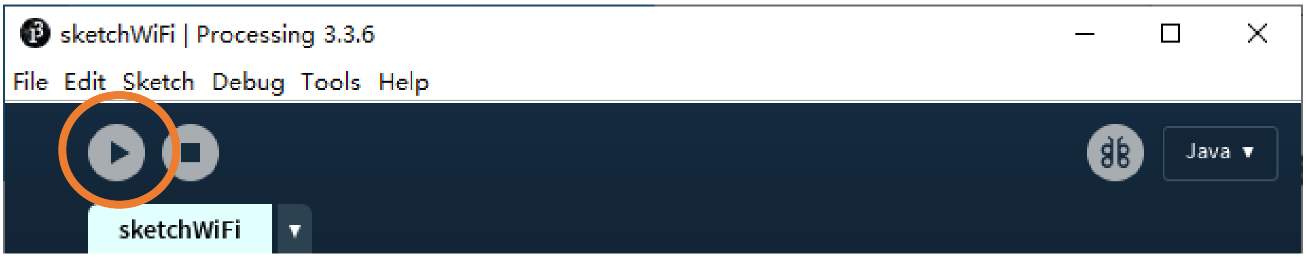
Input "ControlP5" in the searching box, and then select the option as below. Click "Install" and wait for the installation to finish.

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

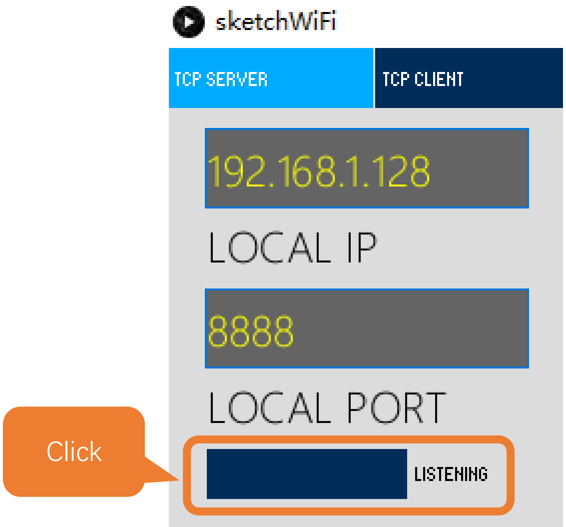

Oscilloscope | Processing 3.5.4
File Edit Sketch Debug Tools Help
Run Ctrl+R
Present Ctrl+Shift+R
Tweak Ctrl+Shift+T
Stop
Import Library...
Show Sketch Folder Ctrl+K
Add File...
Oscilloscope (Java)
Add Library...
DXF Export
Network
PDF Export
Serial
SVG Export
Contributed
ControlP5
Java


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”。

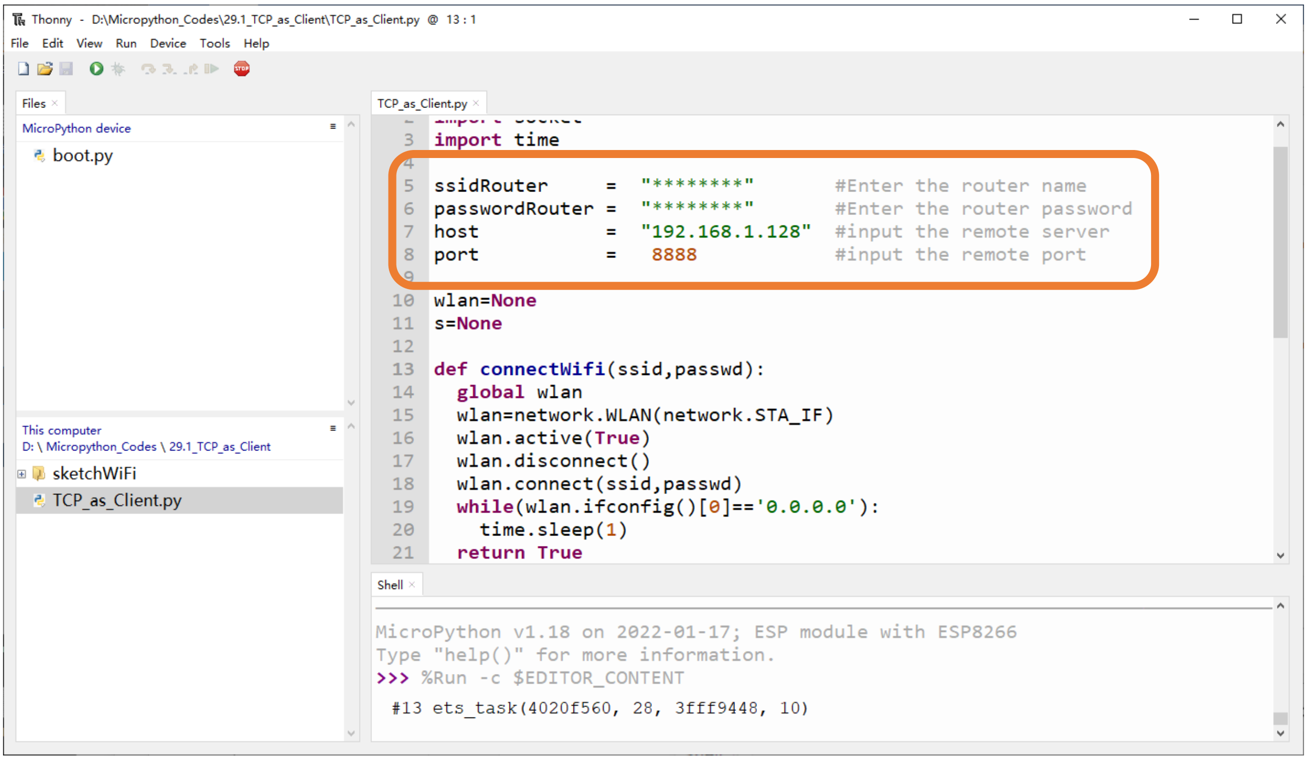


Move the program folder "**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

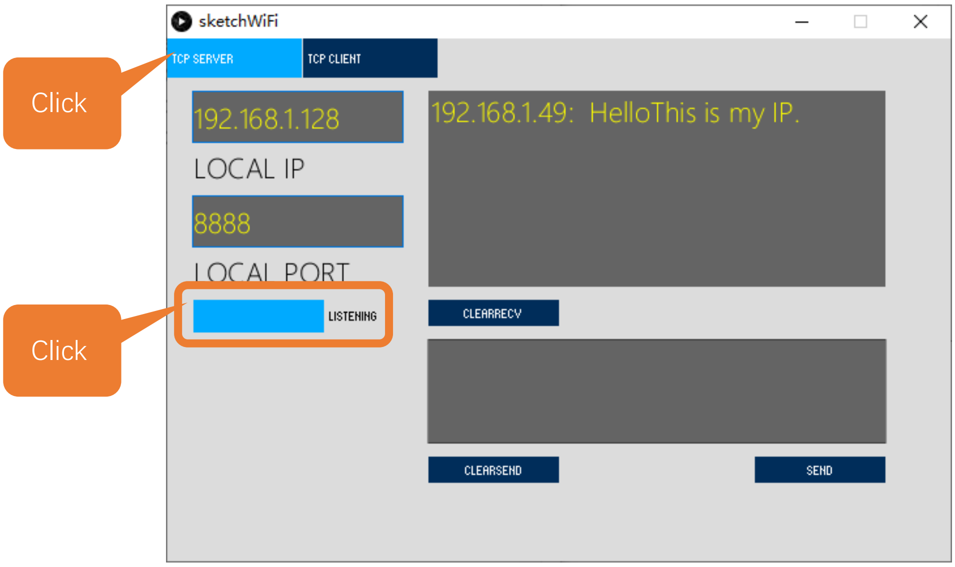
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "29.1_TCP_as_Client" and double click "TCP_as_Client.py".

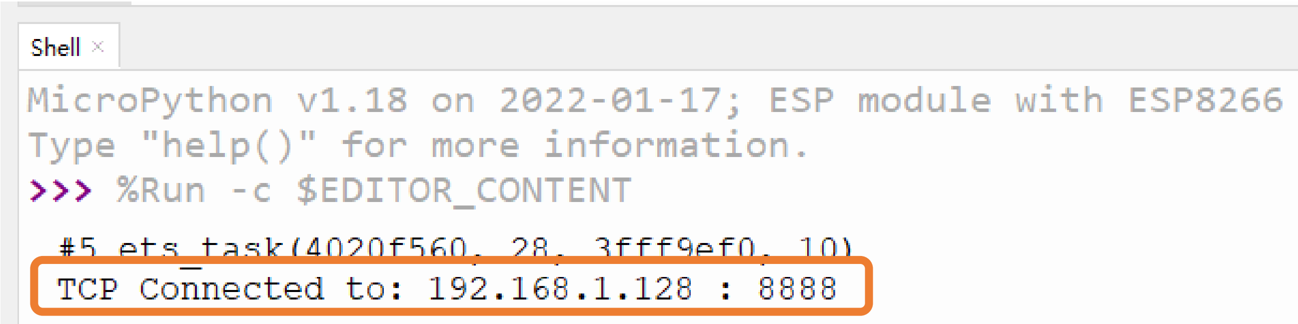
Before clicking "Run current script", please modify the name and password of your router and fill in the "host" and "port" according to the **IP information in processing app** shown in the box below:

29.1_TCP_as_Client

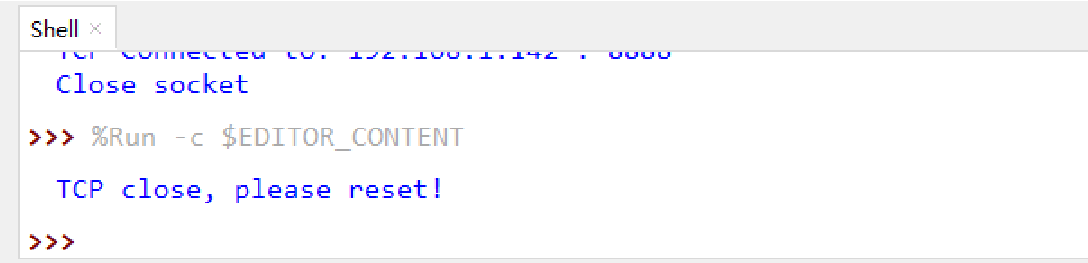


Click "Run current script" and in "Shell", you can see ESP8266 automatically connects to sketchWiFi.

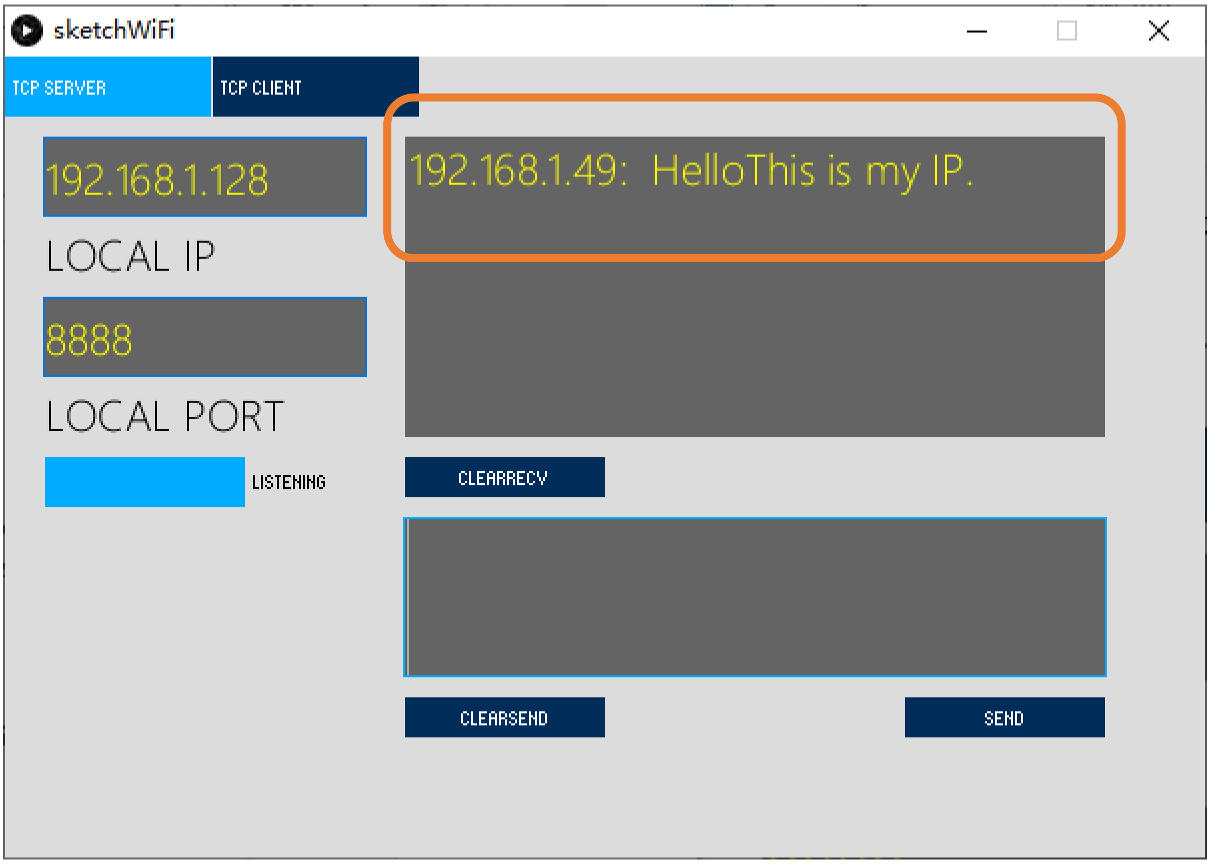




If you don't click "Listening" for sketchWiFi, ESP8266 will fail to connect and will print information as follows:



ESP8266 connects with TCP SERVER, and TCP SERVER receives messages from ESP8266, as shown in the figure below.



The following is the program code:

```
import network
import socket
import time
```

```

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

wlan=None
s=None

def connectWifi(ssid,passwd):
  global wlan
  wlan= network.WLAN(network.STA_IF)
  wlan.active(True)
  wlan.disconnect()
  wlan.connect(ssid,passwd)
  while(wlan.ifconfig()[0]=='0.0.0.0'):
    time.sleep(1)
  return True
try:
  connectWifi(ssidRouter,passwordRouter)
  s = socket.socket()
  s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
  s.connect((host,port))
  print("TCP Connected to:", host, ":", port)
  s.send('Hello')
  s.send('This is my IP.')
  while True:
    data = s.recv(1024)
    if(len(data) == 0):
      print("Close socket")
      s.close()
      break
    print(data)
    ret=s.send(data)
except:
  print("TCP close, please reset!")
  if (s):
    s.close()
  wlan.disconnect()
  wlan.active(False)
```

Import network、socket、time modules.

```
import network
import socket
import time
```

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

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

Connect specified Router until it is successful.

```
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
```

Connect router and then connect it to remote server.

```
  connectWifi(ssidRouter,passwordRouter)
  s = socket.socket()
  s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
  s.connect((host,port))
  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.

```
  s.send('Hello')
  s.send('This is my IP.')
  while True:
    data = s.recv(1024)
    if(len(data) == 0):
      print("Close socket")
      s.close()
      break
    print(data)
    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.

```
  print("TCP close, please reset!")
  if (s):
    s.close()
  wlan.disconnect()
  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.

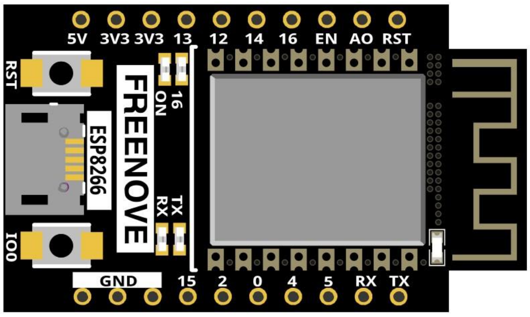
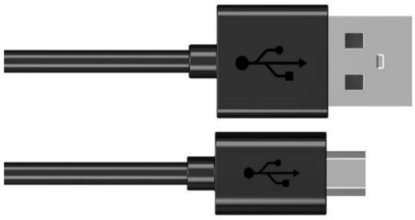
close(): Close socket.

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

Project 29.2 As Server

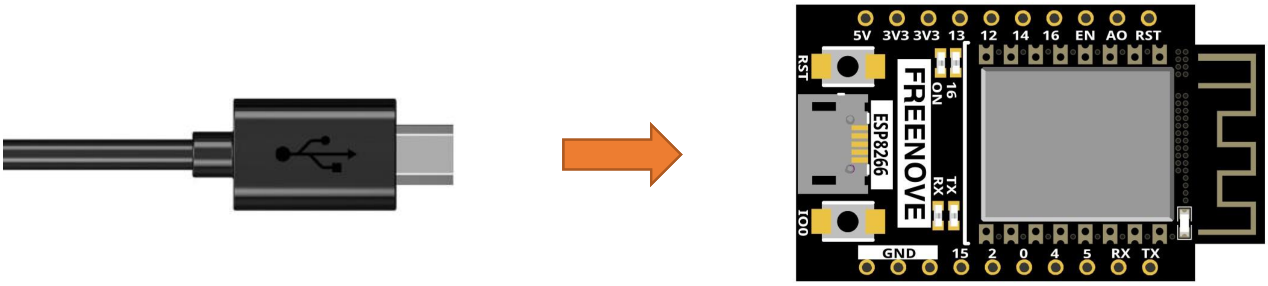
In this section, ESP8266 is used as a Server to wait for the connection and communication with Client on the same LAN.

Component List

ESP8266 x1	USB cable
	

Circuit

Connect Freenove ESP8266 to the computer using the USB cable.



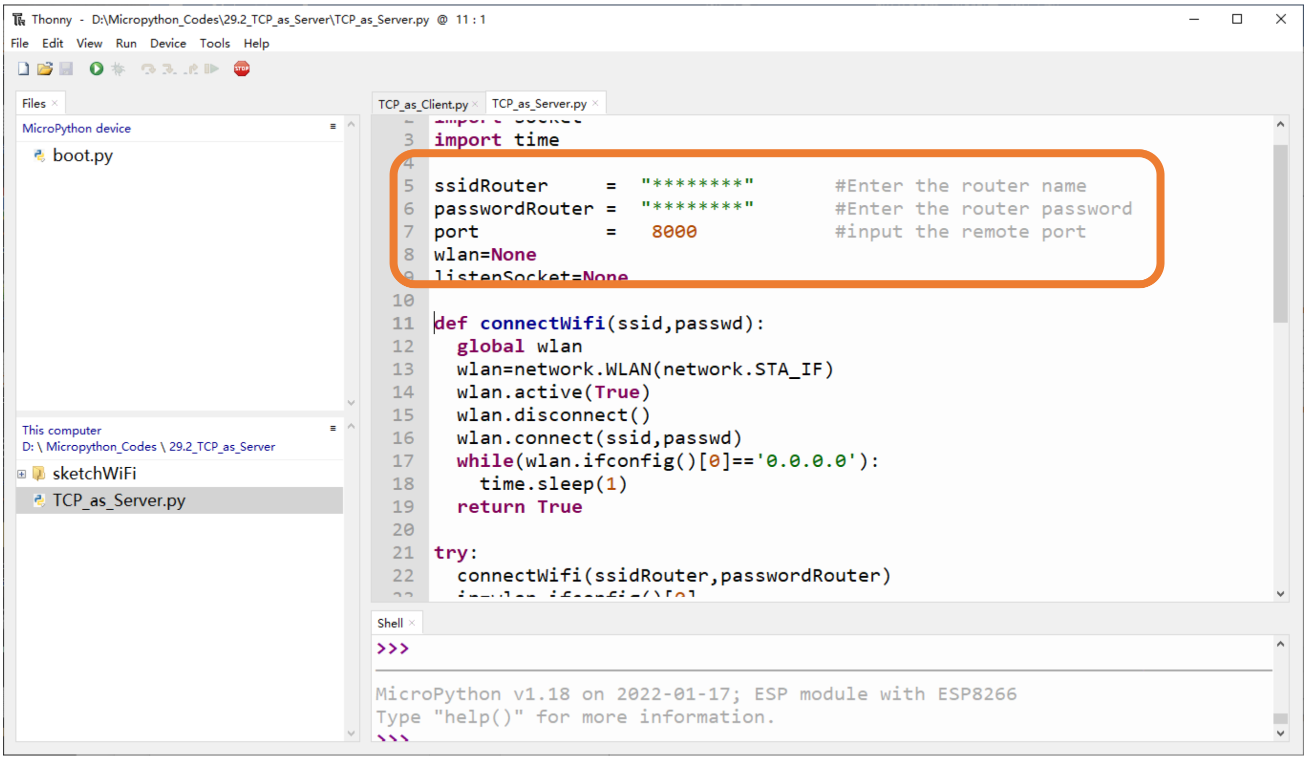
Code

Move the program folder “**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**” to disk(D) in advance with the path of “**D:/Micropython_Codes**”.

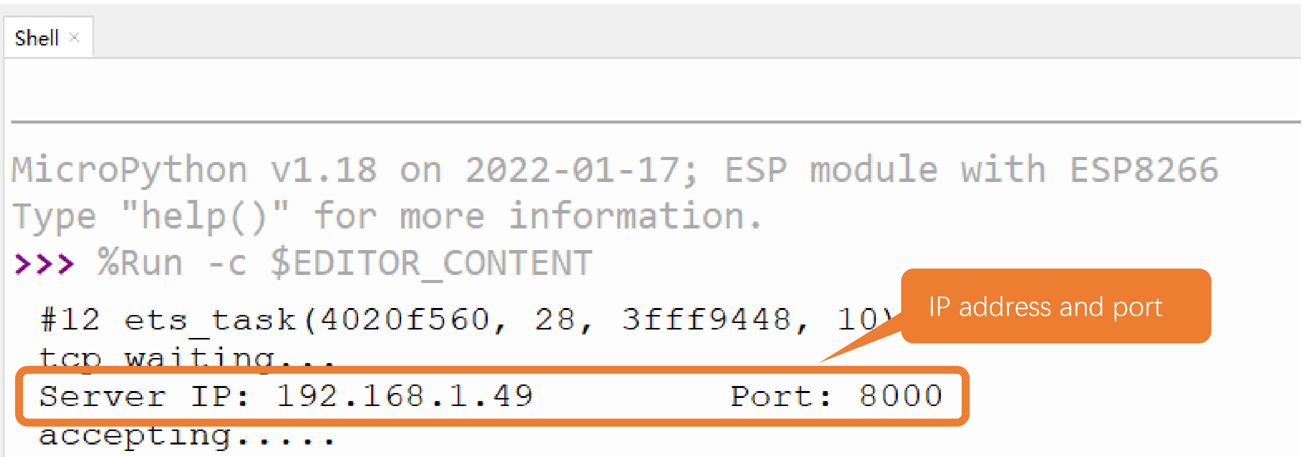
Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “29.2_TCP_as_Server” and double click “TCP_as_Server.py”.

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

29.2_TCP_as_Server



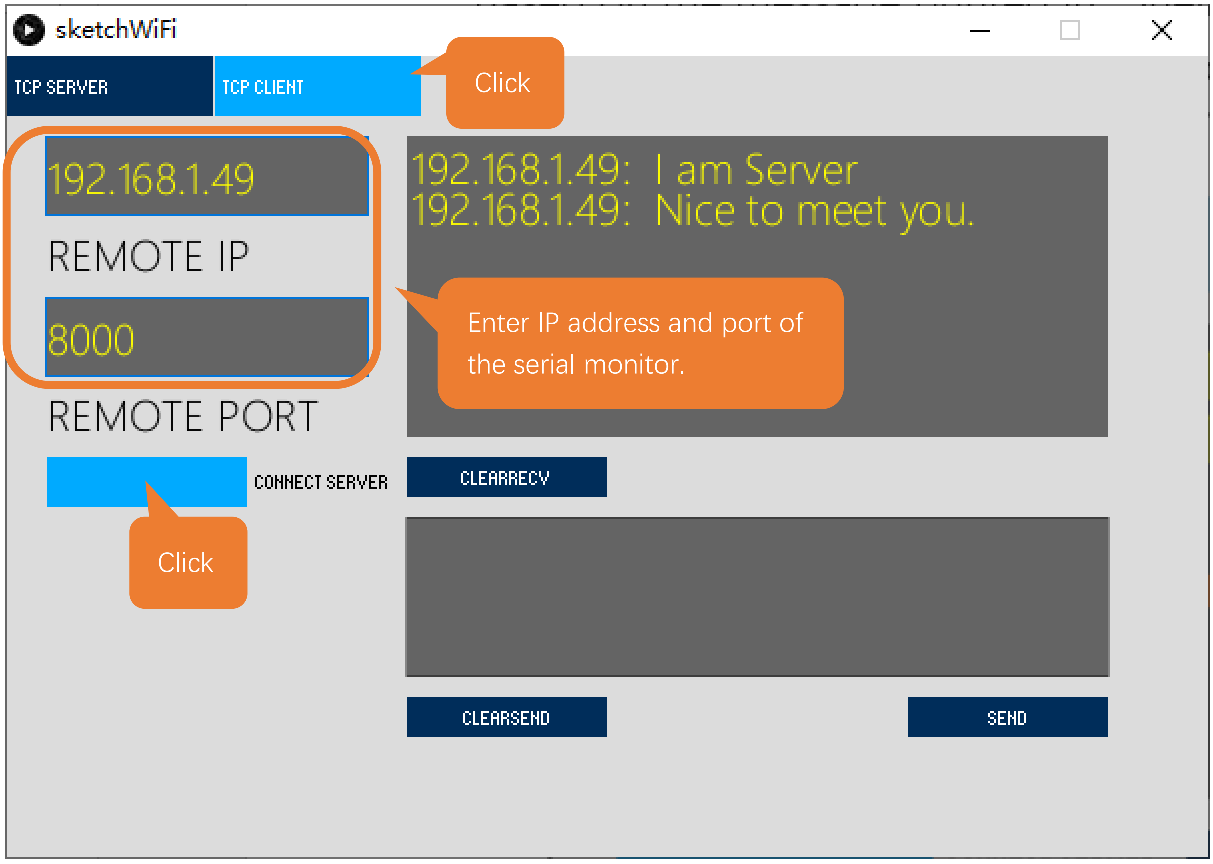
After making sure that the router’s name and password are correct, click “Run current script” and in “Shell”, you can see a server opened by the ESP8266 waiting to connecting to other network devices.



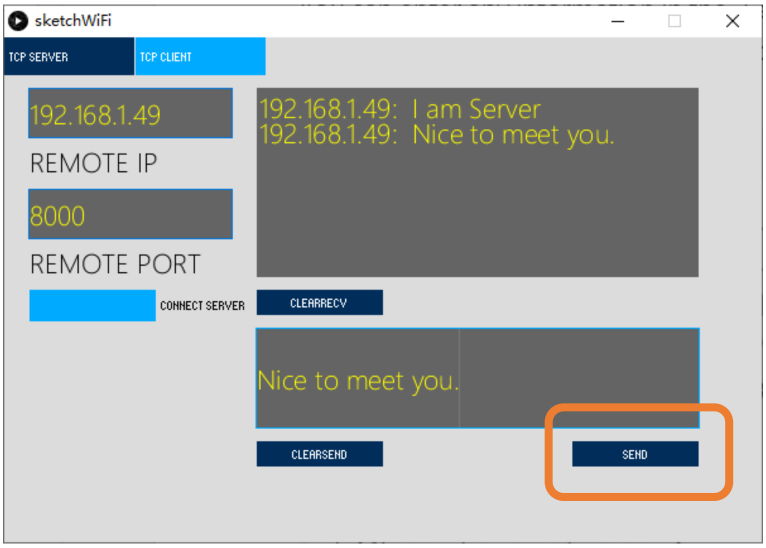
Processing:

Open the "**Freenove_Ultimate_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)
```

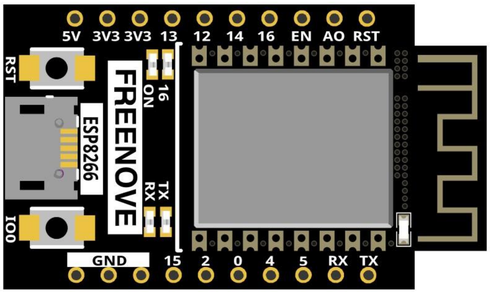
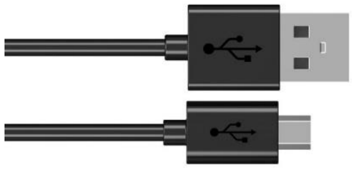
Chapter 30 Smart Home

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 30.1 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

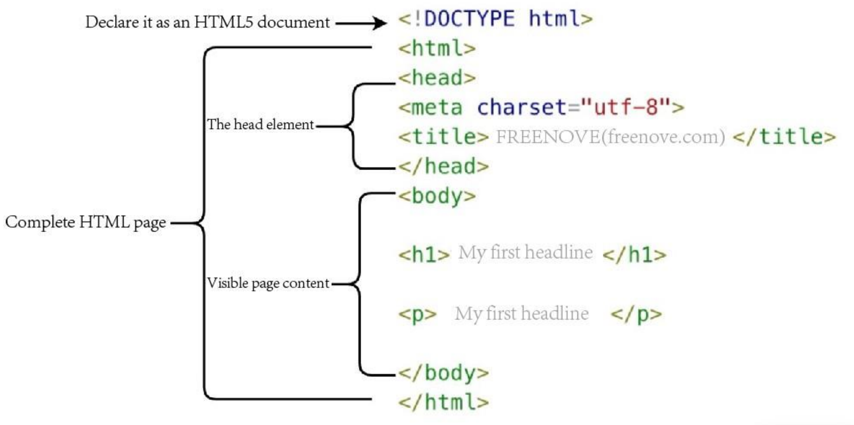
HTML

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

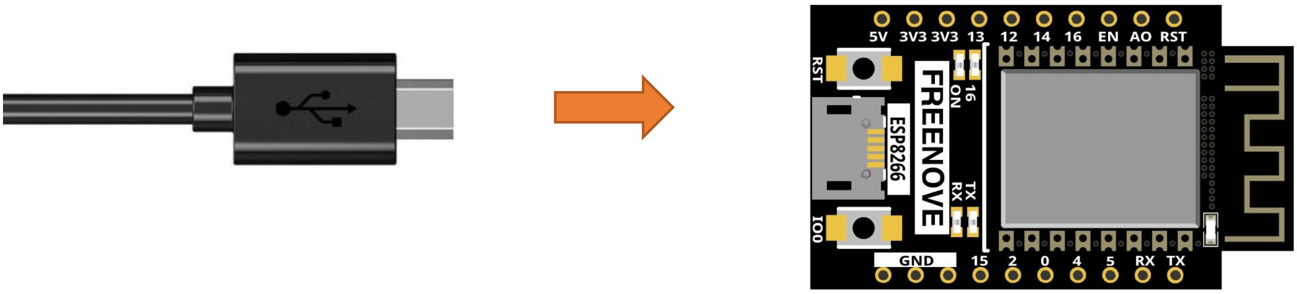
<h1>: Define a big heading

<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.

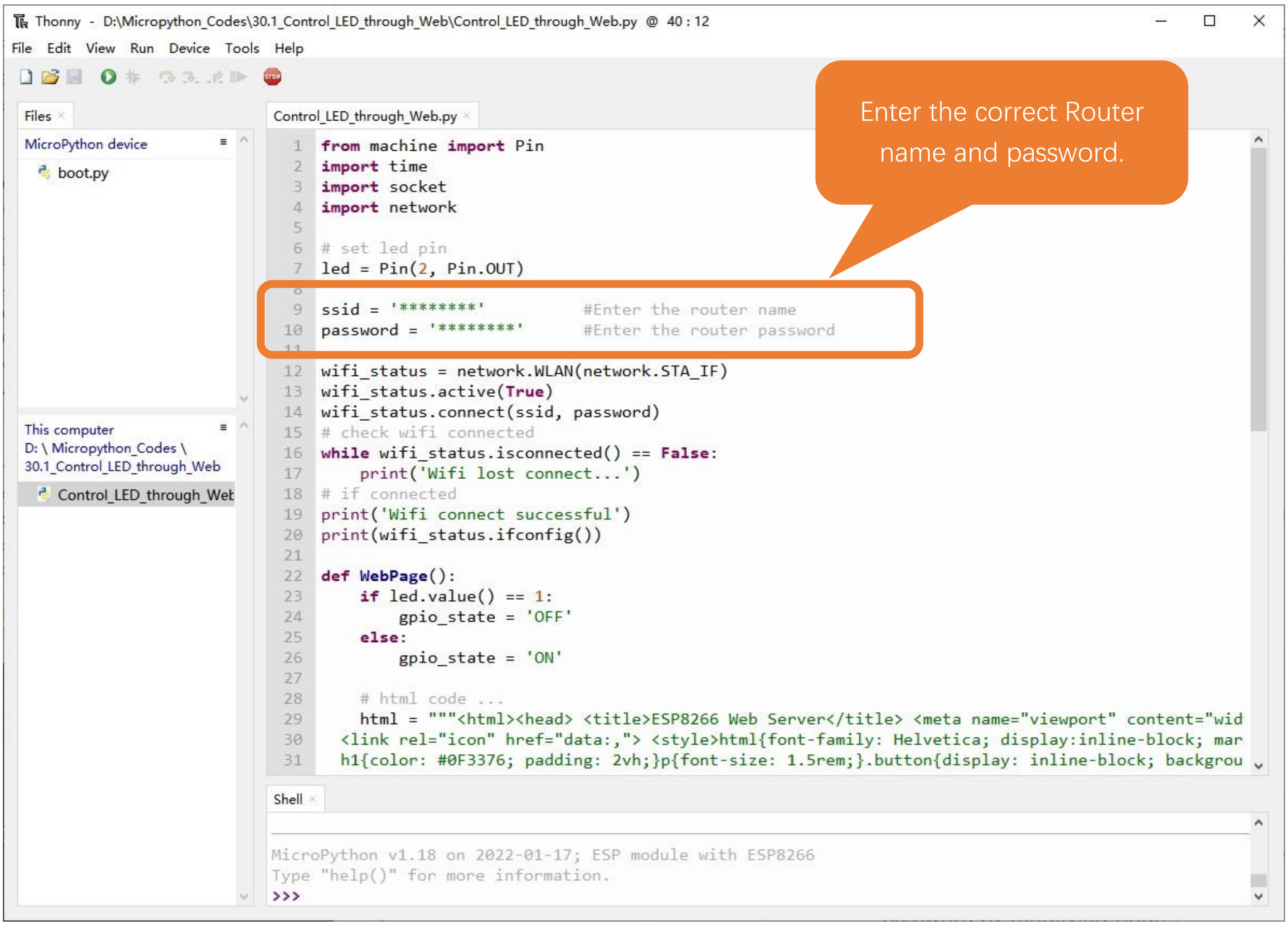


Code

Move the program folder "**Freenove_Ultimate_Starter_Kit_for_ESP8266/Python/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

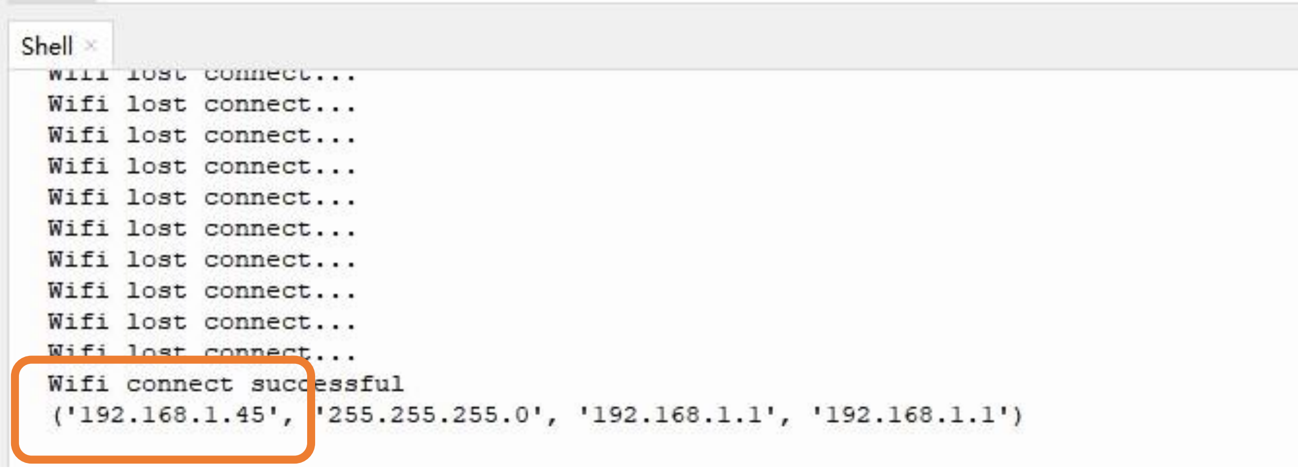
Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "30.1_Control_LED_through_Web". and double click "Control_LED_through_Web".

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".



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:

```
from machine import Pin
import time
import socket
import network

# set led pin
led = Pin(2, Pin.OUT)

ssid = '********'            #Enter the router name
password = '********'        #Enter the router password

```

```
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)
```

```
s.bind(('', 80))
s.listen(5)
try:
    while True:
        conn, addr = s.accept()
        print('Connection: %s' % str(addr))
        req = conn.recv(1024)
        req = str(req)
        print('Connect = %s' % req)
        led_on = req.find('/?led=on')
        led_off = req.find('/?led=off')
        if led_on == 6:
            print('LED ON')
            led.value(0)
        else:
            print('LED OFF')
            led.value(1)
        if led.value() == 1:
            gpio_state = 'OFF'
        else:
            gpio_state = 'ON'
        response = WebPage()
        conn.send('HTTP/1.1 200 OK\n')
        conn.send('Content-Type: text/html\n')
        conn.send('Connection: close\n\n')
        conn.sendall(response)
        conn.close()
except:
    pass
```

Import socket module and Import network module.

```
import socket
import network
```

Enter correct AP name and password.

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

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

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

"Shell" displays the IP address assigned to ESP8266.

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

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

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

```
            led.value(0)
        else:
            print('LED OFF')
            led.value(1)
        if led.value() == 1:
            gpio_state = 'OFF'
        else:
            gpio_state = 'ON'
```

Chapter 31 Soldering Circuit Board

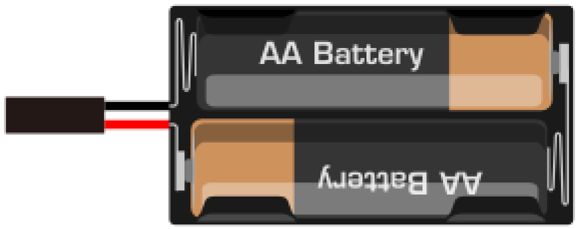
Project 31.1 Soldering a Buzzer

We have tried to use a buzzer in a previous chapter, and now we will solder a circuit that when the button is pressed, the buzzer sounds.

This circuit doesn't need programming and can work when it is powered on. And when the button is not pressed, there is no power consumption.

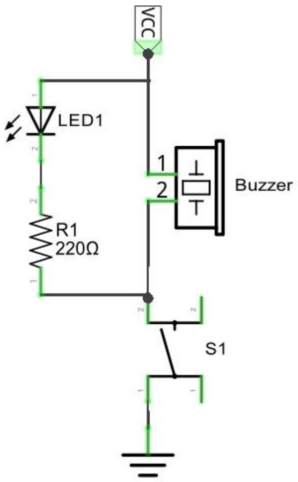
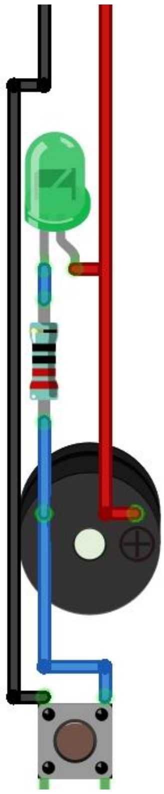
You can install it on your bike, bedroom door or any other places where it is needed.

Component List

Pin header x2	LED x1	Resistor 220Ω x1	Active buzzer x1	Push button x1
				
AA Battery Holder x1				
				

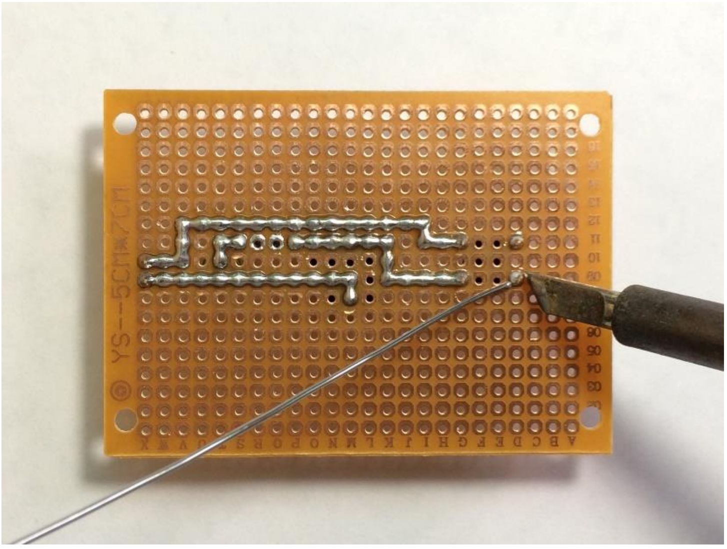
Circuit

We will solder the following circuit on the main board.

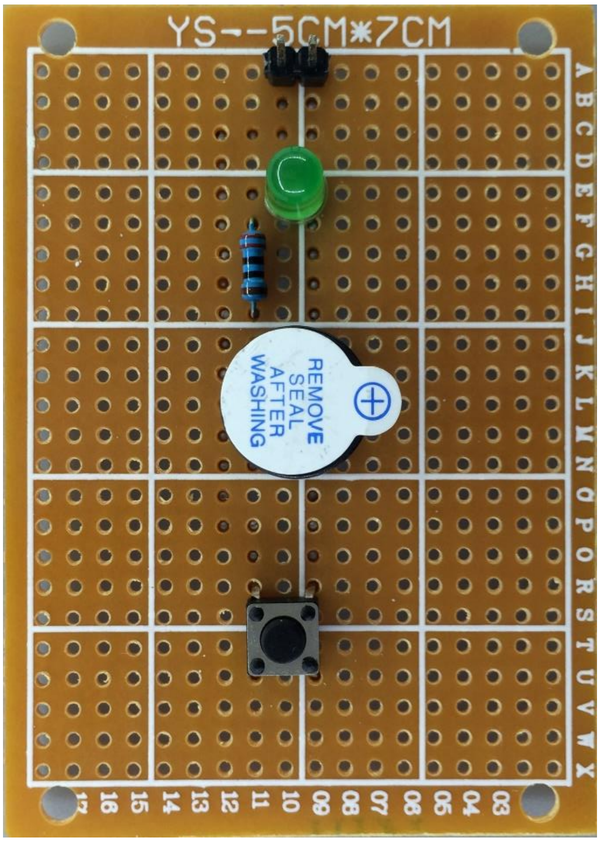
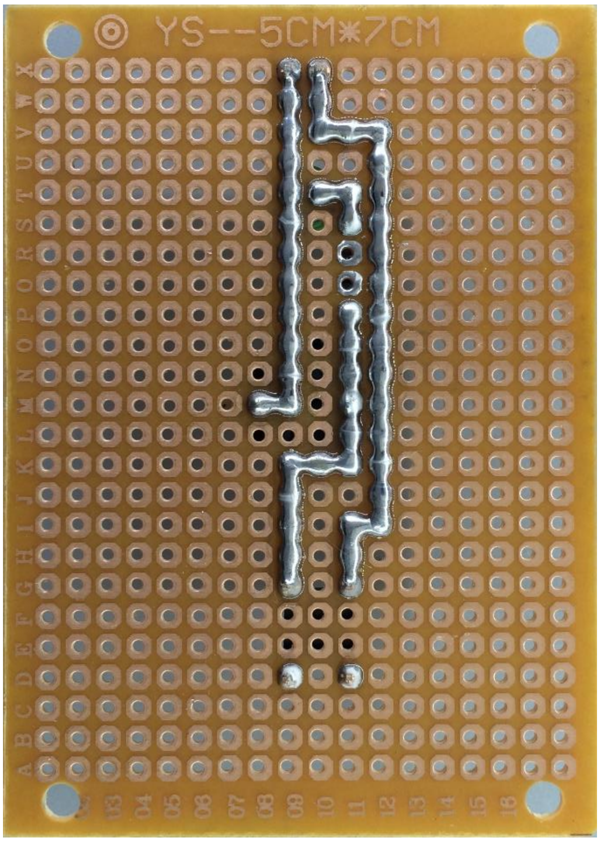
Schematic diagram	Hardware connection. If you need any support, please feel free to contact us via: **support@freenove.com**
	

Solder the Circuit

Insert the components on the main board and solder the circuit on its back.

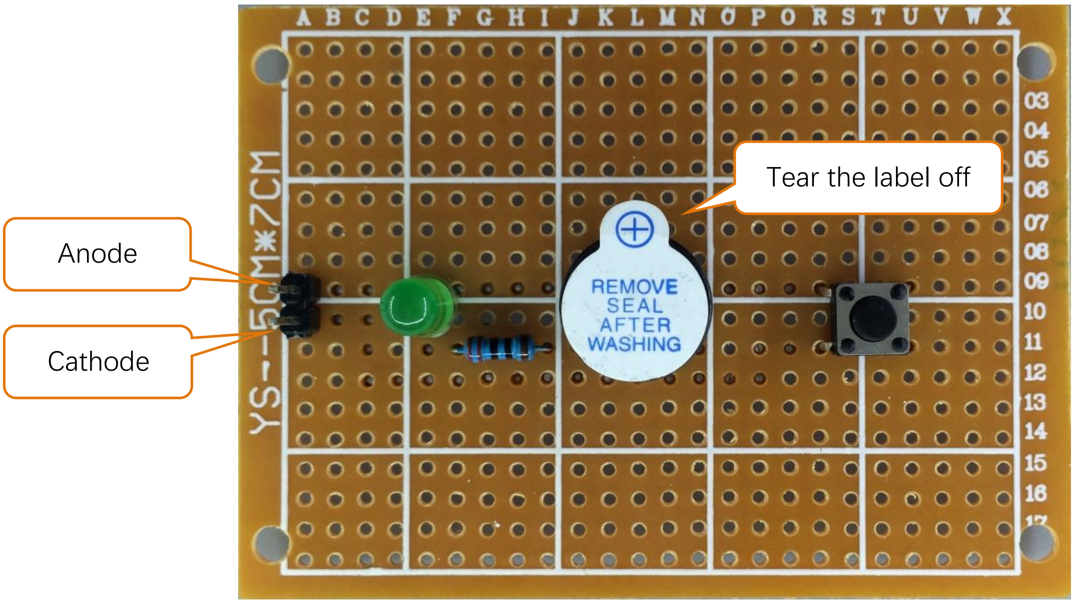


Rendering after soldering:

Front	Back
	

Test circuit

Connect the circuit board to power supply (3~5V). You can use ESP8266 board or battery box as the power supply.

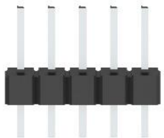
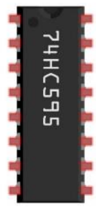


Press the push button after connecting the power, and then the buzzer will make a sound.

Project 31.2 Soldering a Flowing Water Light

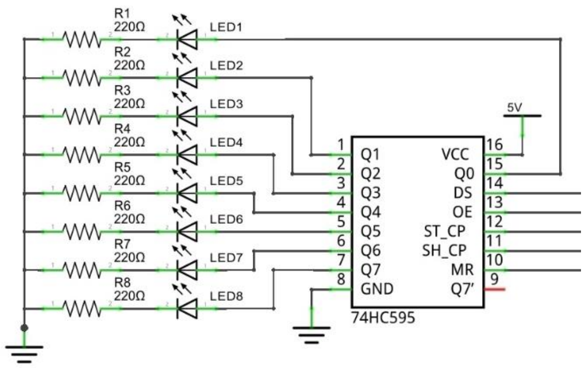
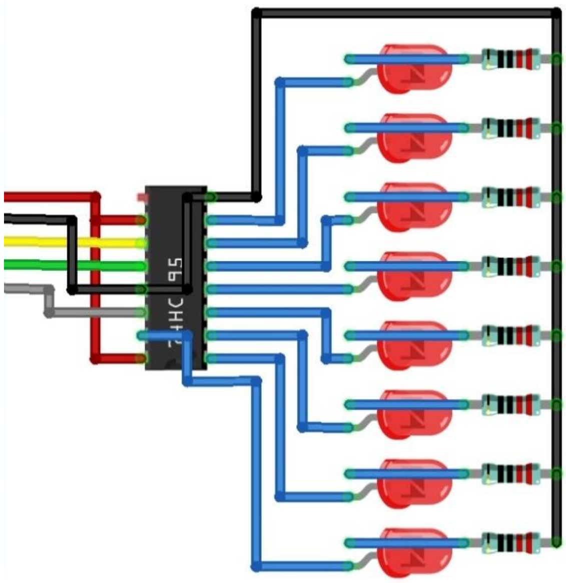
From previous chapter, we have learned to make a flowing water light with LED. Now, we will solder a circuit board, and use the improved code to make a more interesting flowing water light.

Component List

Pin header x5	Resistor 220Ω x8	LED x1	74HC595 x1
			

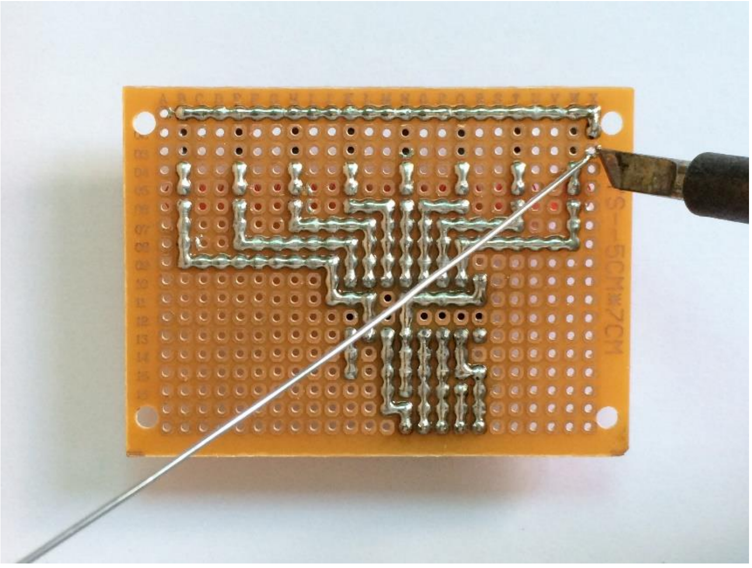
Circuit

Solder the following circuit on the main board.

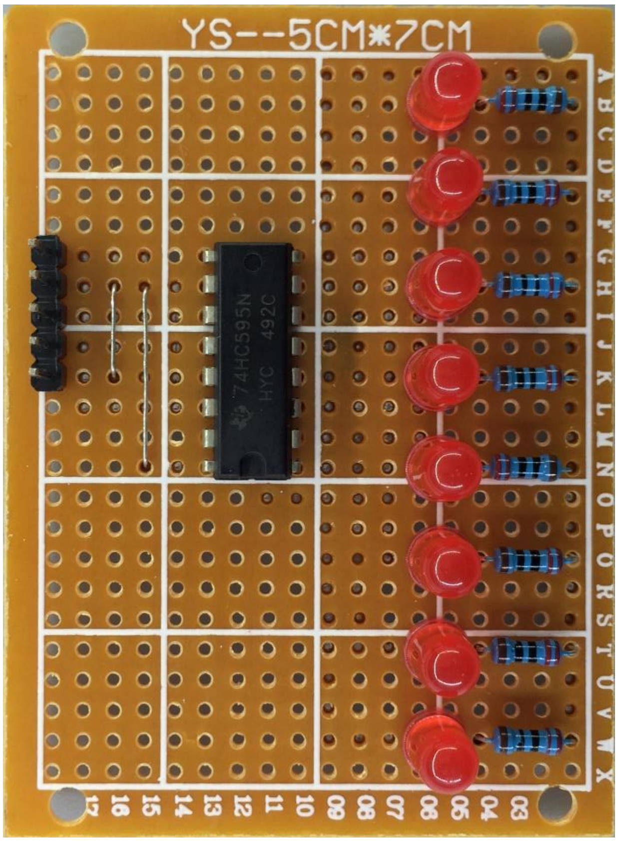
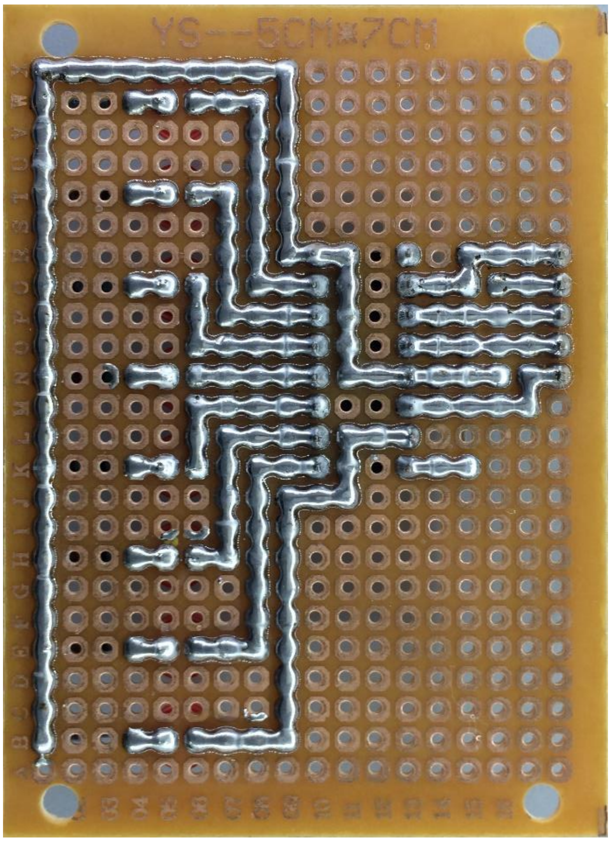
Schematic diagram	Hardware connection
	

Soldering the Circuit

Insert the components on the main board and solder the circuit on its back.

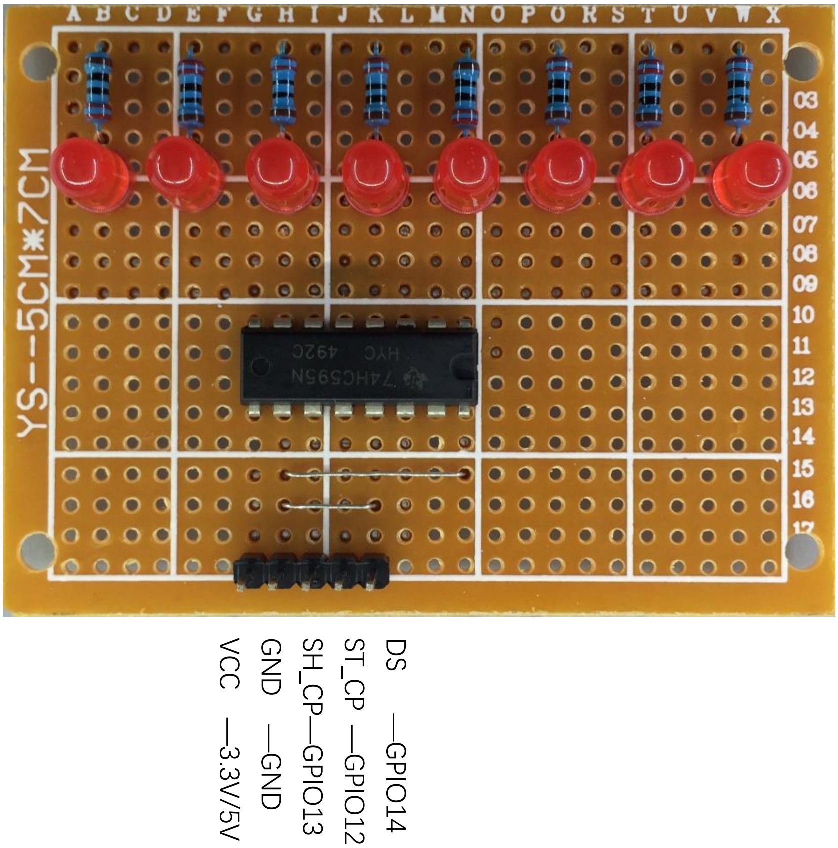


Rendering after soldering:

Front	Back
	

Connect the Circuit

Connect the board to ESP8266 with jumper wire in the following way.



Code

The following is the program code:

```
import time
from my74HC595 import Chip74HC595

chip = Chip74HC595(14,12,13)
# ESP8266-14: 74HC595-DS(14)
# ESP8266-12: 74HC595-STCP(12)
# ESP8266-13: 74HC595-SHCP(11)

while True:
    x=0x01
    for count in range(8):
        chip.shiftOut(1,x)   #High bit is sent first
        x=x<<1
```

```
        time.sleep_ms(300)
    x=0x01
    for count in range(8):
        chip.shiftOut(0,x)  #Low bit is sent first
        x=x<<1
        time.sleep_ms(300)
```

What's next?(73)

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 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.