Get Started MaESP ESP32 in Thonny

Introduction

The Makerfabs MaESP ESP32 Kit intended to help learners getting started with MicroPython/Arduino on the ESP32. This Kits includes the most common electronic components and modules, for the starters to learn the basic knowledge of MicroPython, usage of Thonny IDE and usage of Arduino IDE. 12 basic lessons help starters to learn the usage and program skills about MicroPython/Arduino.

The total learning time for this kits& lessons is about 20 hours.

What is MicroPython?

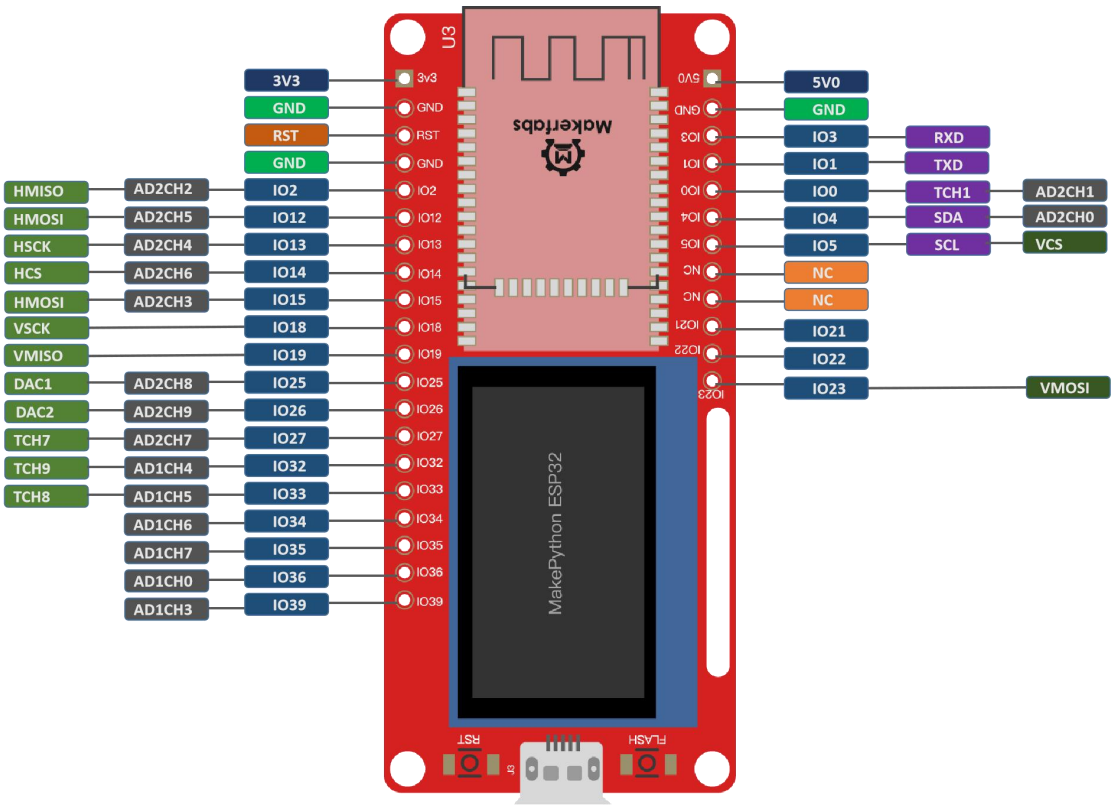
Python is one of the most widely use simple and easy-to-learn programming languages around. MicroPython is a lean implementation of the Python 3 programming language that has been pared down to run efficiently on microcontrollers. It is a relatively simple but powerful language that is easy for beginners to pick up and has been gaining popularity in schools as an introductory language. MicroPython has nearly all of the features of Python, which means that interacting with hardware is now easily accessible to beginners and seasoned Python programmers alike.

MicroPython/Arduino goal is to make programming digital electronic as simple as possible, so it can be used by anyone. Currently, MicroPython/Arduino is used by hobbyists, researchers, teachers, educators, and even in commercial products.

Makerfabs MaESP ESP32 can be compiled in both micropython/GCC environments. Our kit comes with micropython firmware pre-installed at the factory, so you can compile micropython programs directly. If you want to use Arduino routines, you can burn them directly. The Arduino IDE rewrites Flash, but once you've burned the Arduino routines, you need to re-download the micropython firmware to recompile the micropython program. Micropython firmware download address https://micropython.org/download/esp32/

Product List:

Makerfabs MaESP ESP32 Pin figure:



Directory

Introduction....1
I.MicroPython Development Tool....4
II.MaESP Lessons in Thonny....9
1.Lesson1 LED Control....9
2.Lesson2 Running LED....11
3.Lesson3 Button....13
4.Lesson4 PWM Control....15
5.Lesson5 Voice Control....18
6.Lesson6 OLED Display....21
7.Lesson7 Temperature Monitor DS18B20....23
8.Lesson8 Digital LED WS2812....25
9.Lesson9 Pulse Sensor....28
10.Lesson10 AnalogRead....30
11.Lesson11 Ultrasonic Ranging....33
12.Lesson12 WiFi....38
III.Troubleshooting....48
IV.Flash/Upload MicroPython Firmware to ESP32....49

I. MicroPython Development Tool

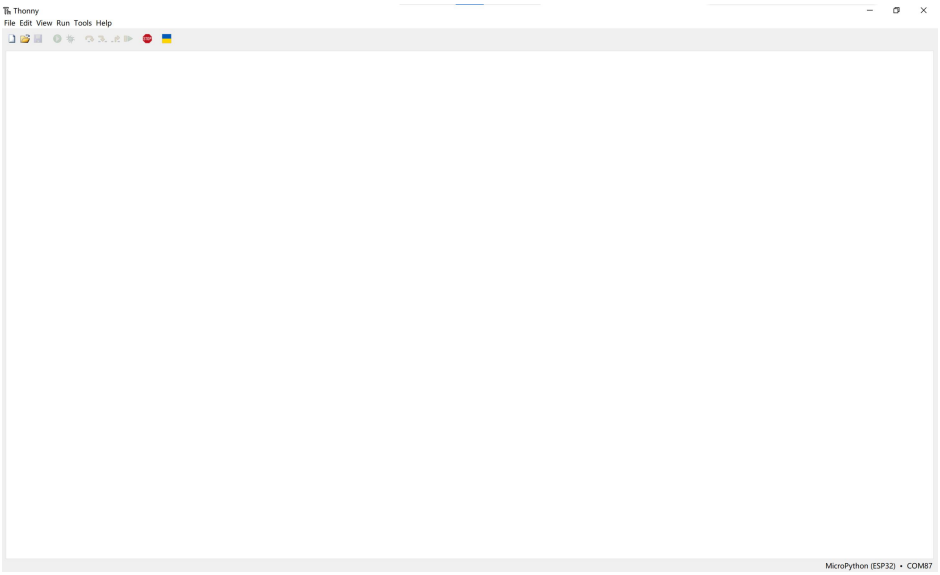
Installing Thonny IDE Windows PC

There many codes& programming methods with MicroPython. For this tutorial, we use Thonny IDE, which is the most simple& easy way for starts to skip into MicroPython. You can learn more about Thonny IDE on their GitHub repository or explore the Thonny IDE source code. Click this link to download Thonny IDE for Windows:https://thonny.org/ .

WARNING: Please choose the thonny download version according to your computer's operating system.

After a few seconds you should see a similar file (thonny-4.0.2.exe) in your Downloads folder

Double-click that file and install it. Open the Thonny, A new window opens with the Thonny IDE software.



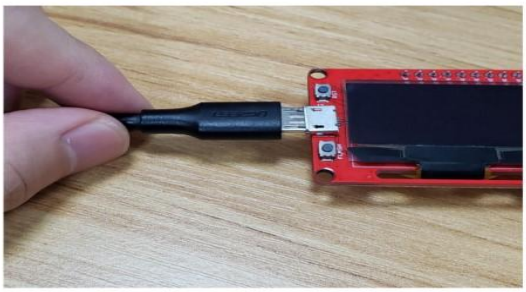
A Simple code for blinking an LED on ESP32 as follows:

Get Start with Thonny IDE

Open Thonny IDE. Let's learn how to run micropython programs on Thonny IDE:

1. Establishing a communication with the board

After unpacking the MaESP ESP32, connect it to the PC with a Type-C USB cable.

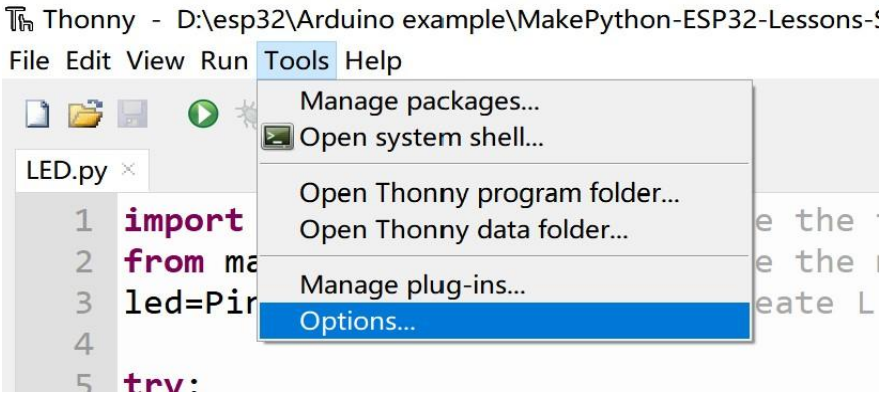


2. Installed the USB driver

Go to ***Tools > option>>Interperter*** and select the port your MaESP ESP32 connected (the like for downloading the USB driver is: https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers)

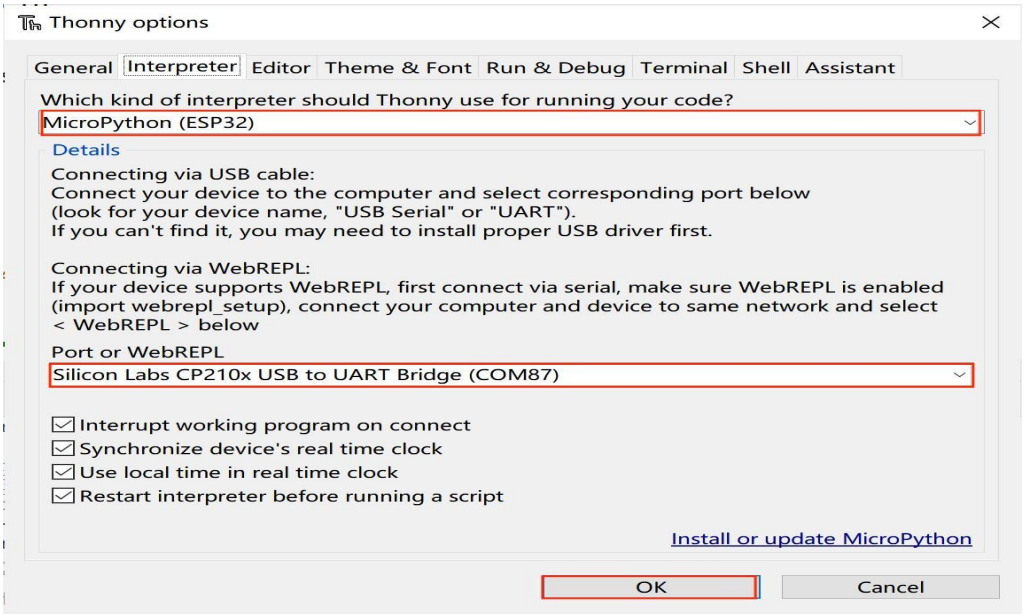
3. Click the tool, and choose the Options.

It can choose the interpreter to run micropython, and you also need to choose the corresponding serial port.



4. Selecting the Board and serial port

Users need to choose the right interpreter and port. This is to correctly identify that ESP32 communicates with the computer through the serial port, and it can only be successfully compiled with Thonny.



5. Creating new file on your board

1. Press the "***New file***" button to create a new file.
2. Press the "***Save file***" button to save the file in your computer.
3. When save the file, it will pop-up window. Users can choose the flie save to This computer or Micropython device.

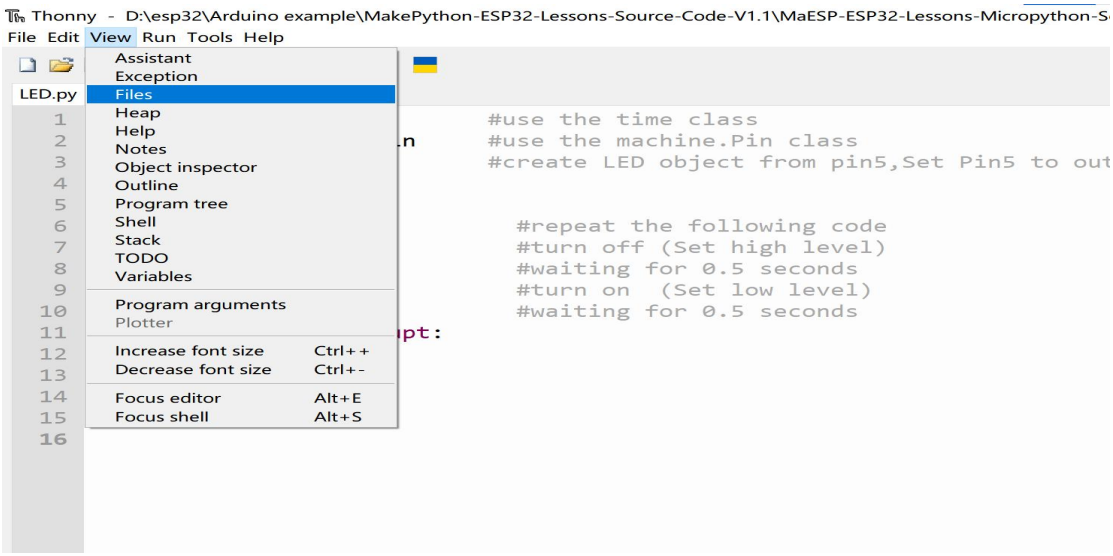
Note: If the program is saved on the computer, it needs to be uploaded to the Mrcropython device when it is to run. If the program is saved on the Mrcropython device, when the internal flash of the device is erased, the program cannot be retrieved from the Flash.

4. Name the file and click the "OK" button. And the file save type is **".py"** .

Uploading The program: Micropython Lessons

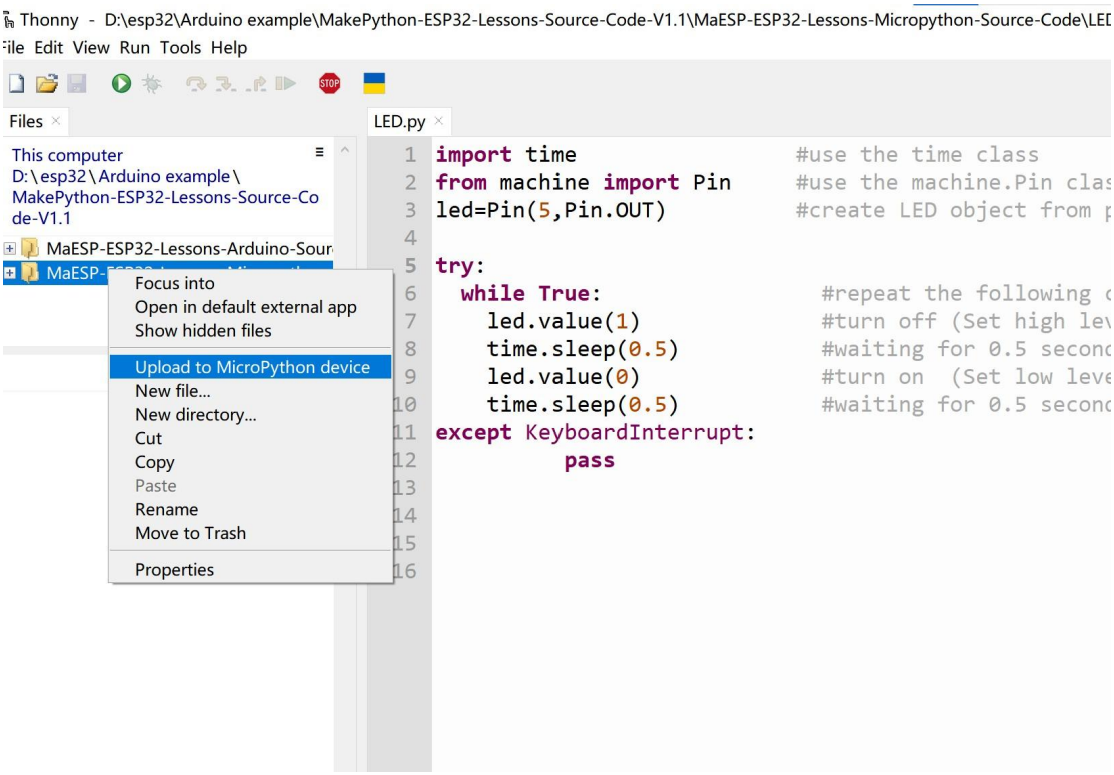
1. Click the View and choose the files.

After having the MicroPython firmware installed (MicroPython Firmware already installed when you get Makerfabs MaESP ESP32), follow the steps: It can open the local file, now you can upload the micropython program to the MaESP ESP32.



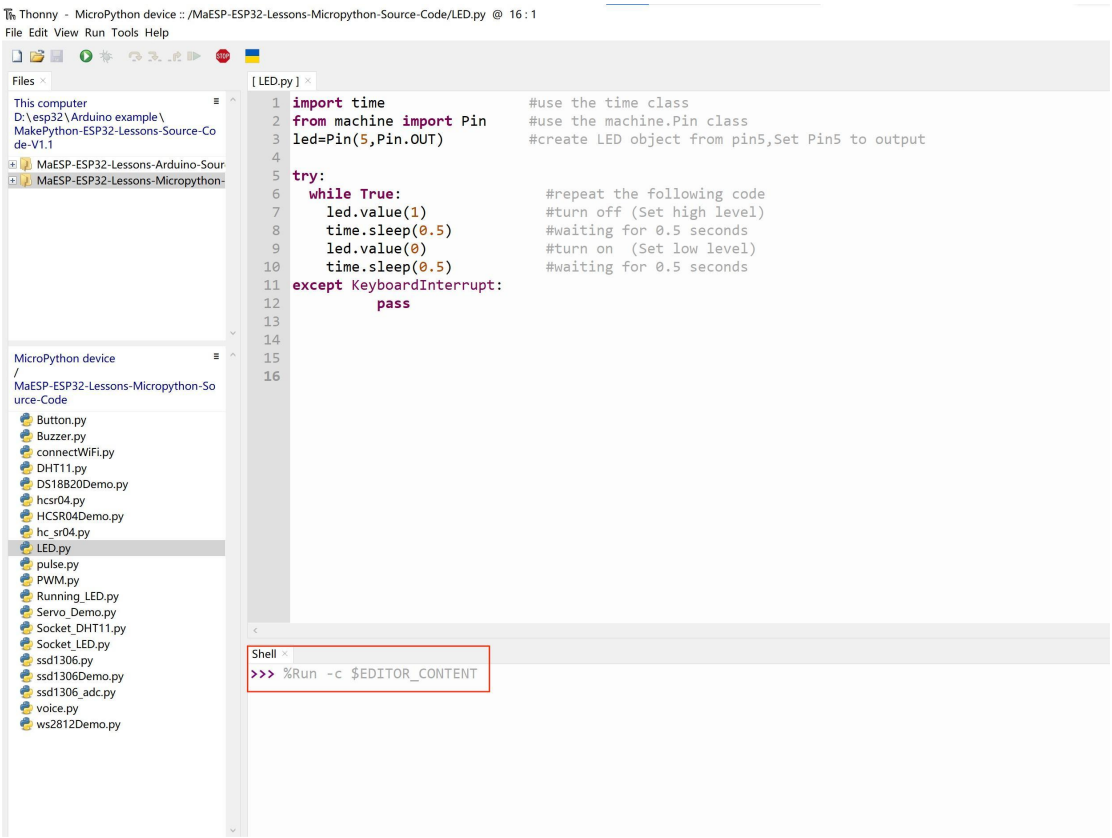
2. Upload the micropython program to the MaESP ESP32

Locate the local path to the downloaded resource code, select that folder, and upload it to the micropython controller.



3. Running the first demo in Thonny

Finished to upload the program, choose one of the demos, click the running on , users can open the demo to achieve the project.



We can see that the shell already has commands that run successfully.

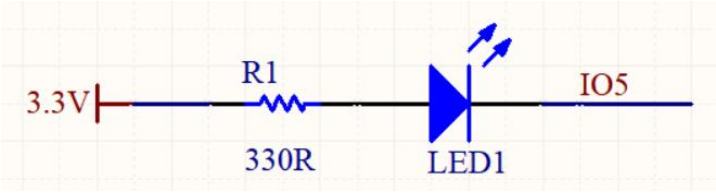
II. MaESP Lessons in Thonny

1. Lesson1: LED Control

This is the most basic lesson, in which we will learn how to control the digital output pins with MicroPython, by control the lighting of an LED.

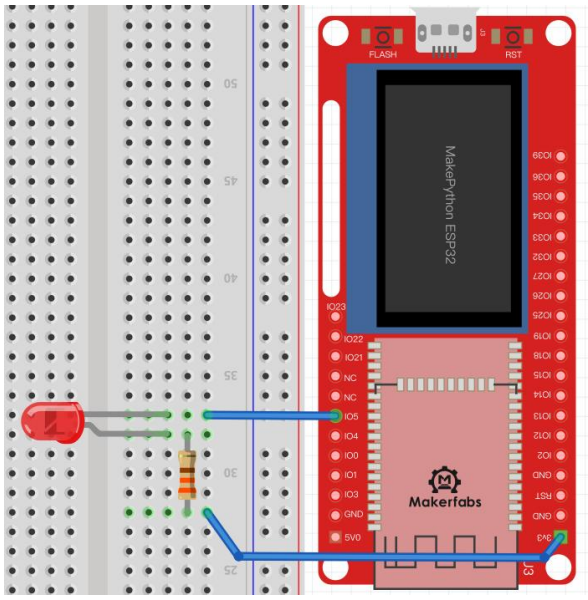
Material:

- 1*LED;
- Resistor: 1*330R;
- Jump wires;



Instructions: The short pin on the left is negative and the long pin on the right is positive.
The purpose of resistor is to limit the current, avoid burning LED lamp with too much current.

Wiring: Connect the long LED pin to the end of the 330R resistor, the other end of the resistance is connected to 3.3v, the short pin of the LED lamp is connected to the IO5 of ESP32. As follows:



Create a file：Open the Thonny IDE by clicking , and click save ,to name the Settings file, select your own save path. You can then program in the LED documentation:

Note:* *The first time you open the **Thonny IDE, the **files window** does not show. When you click the **View**, the Files can be choose in the menu. Advice that all the program save the locate computer, then upload the program to the board. it can ensure that the program is not lost.*

Code and comments:

```
import time                    #use the time class
from machine import Pin        #use the machine.Pin class
led=Pin(5,Pin.OUT)             #create LED object from pin5,Set Pin5 to output

try:
  while True:                  #repeat the following code
    led.value(1)               #turn off (Set high level)
    time.sleep(0.5)            #waiting for 0.5 seconds
    led.value(0)               #turn on    (Set low level)
    time.sleep(0.5)            #waiting for 0.5 seconds
except KeyboardInterrupt:
        pass
```

Grammar explanation:

- led=Pin(5,Pin.OUT):

Create an LED object and set it to output mode

- while:

Used to loop through a program, that is, under certain conditions, loop through a segment of a program to handle the same task that needs to be processed repeatedly

- led.value(1):

Set the led pin value to high level. Since the other pin is connected to 3.3v, there is no voltage difference between the two pins.

- led.value(0):

Set the pin value to low level, there is a voltage difference between two pins, there is a current through, the LED light is on.

- time.sleep(0.5):

Delay 0.5 seconds, in this time the control will be sleep and do nothing.

Results:

Upload the code file to the Micropython, click the to save and the to run, you will see the LED lights blinks with interval of 0.5 seconds. You can try changing it to other pins yourself. For example, change to ***led=Pin(4,Pin.OUT***) , then connect the wire to IO4, and the light will flash, or the delay is changed to ***time.sleep(1),***The light flickers slowly, with interval of1s. You can also try to increase the number of LED lights and keep them on and off.

Extension:

You can also control the LED lighting by sending command through serial port.

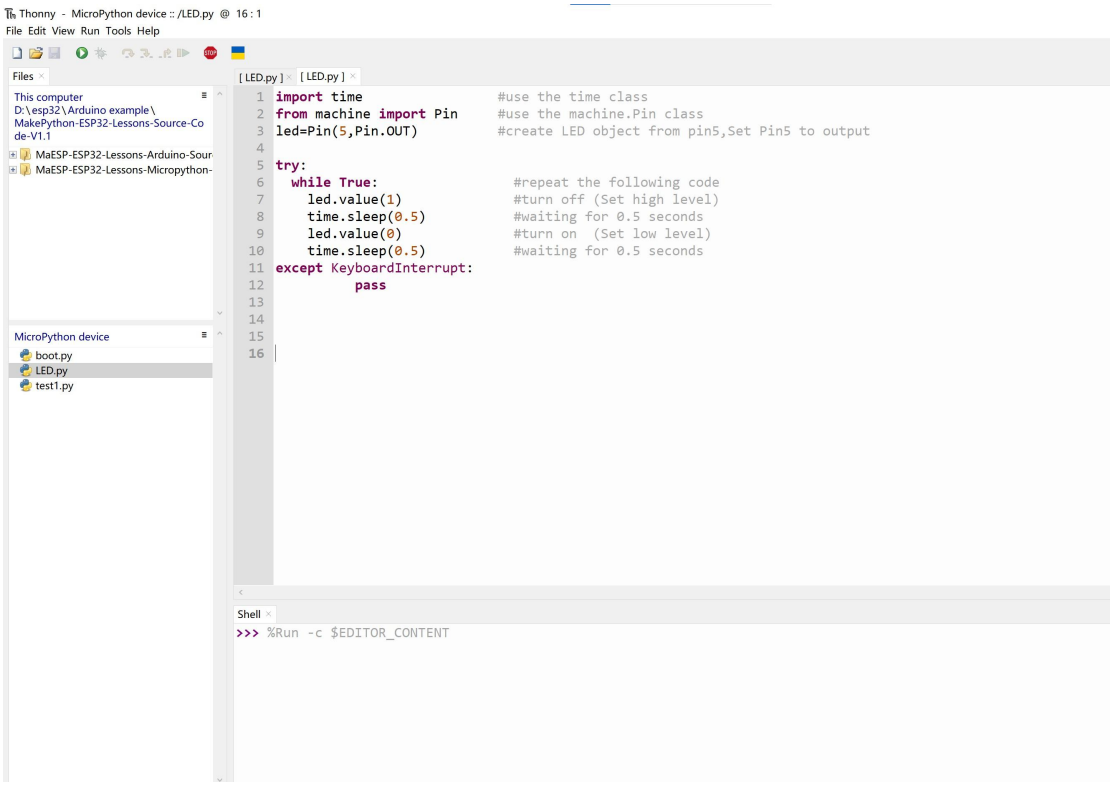
**Note: If your lights are still flashing, that means the code is still running, press to* *stop the program running.*

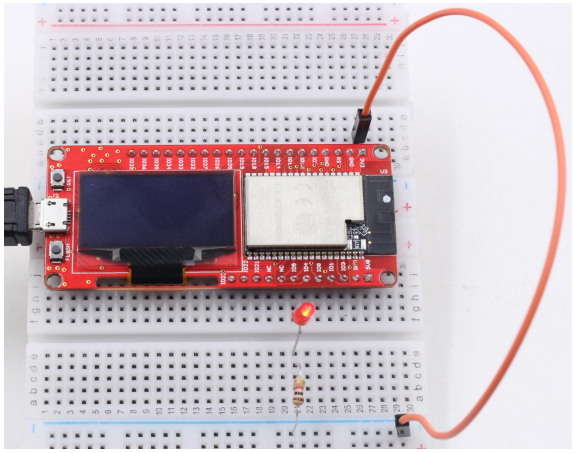
*Write the code after **">>>"** and press **enter** after each line:*

code：

```
>>>from machine import Pin            #Press enter to jump the cursor to the next line
>>>led=Pin(5,Pin.OUT)
>>>led.value(1)      #Press enter, led off
>>>led.value(0)      #Press enter, led on
```

** Each time you finish writing the code, you need to press enter and move the cursor to the next line behind ">>>"*





---

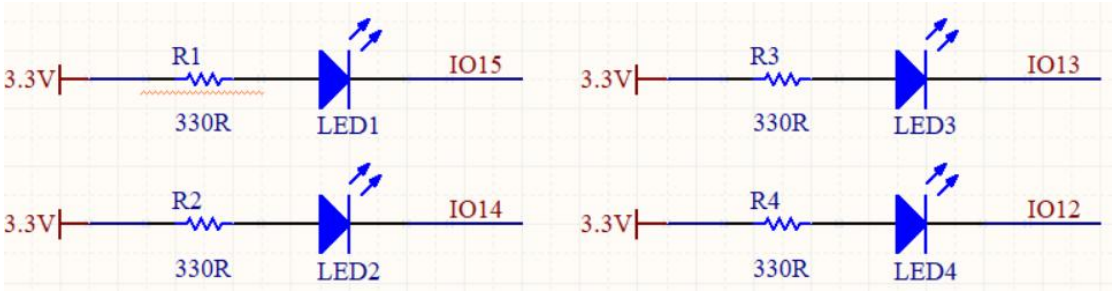
2. Lesson2: Running LED

In this experiment, multiple LED lights are controlled to deepen the understanding of board GPIO port, compilation environment and program framework.

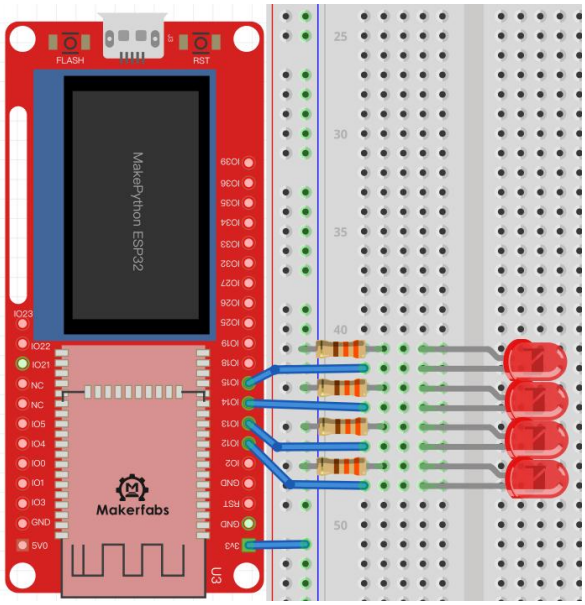
Material:

- 4*LEDs;
- resistance:4*330R;
- Jump wires;

Schematic:



Wiring: Connect the short LED pin to the IO15, IO14, IO13 and IO12 of ESP32, and connect the long pin to one end of the 330R resistor and the other end of the resistor to 3.3V:



Create a new **running_LED** file with the following code and comments:

```
from machine import Pin
import time

#Create an array of Pin15,Pin14,Pin13,Pin12 leds
leds = [Pin(15,Pin.OUT),Pin(14,Pin.OUT),Pin(13,Pin.OUT),Pin(12,Pin.OUT)]
n=0
try:
  while True:
    n = (n+1)%4          #The remainder sign % guarantees that n is between 0 and 3
    leds[n].value(1)     #The value of the NTH LED is high level,turn off
    time.sleep_ms(30)    #Delay of 30 milliseconds
    leds[n].value(0)     #The value of the NTH LED is low level,turn on
    time.sleep_ms(30)    #Delay of 30 milliseconds
except KeyboardInterrupt:
        pass
```

Grammar:

- leds = [Pin(15,Pin.OUT),Pin(14,Pin.OUT),Pin(13,Pin.OUT),Pin(12,Pin.OUT)]:

Create an object for each LED lamp and control them separately, declaring a list form that is easy to understand

- n = (n+1)%4:

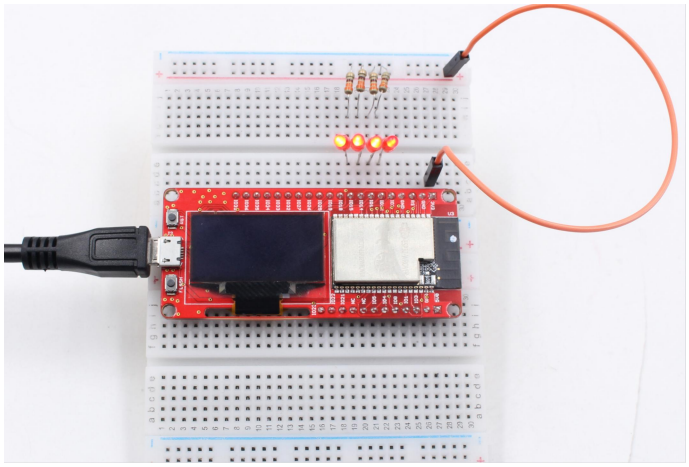
n represents the current LED and we can get the value of the next n after each loop execution (the residual symbol % ensures that the value of n is between 0 and 3)

- led.value():

1 for high level, 0 for low level

Results:

After running the code, you can see four LED lights go on and off in cycles. You can appropriately create more LED objects in the list to make the running lights look better. For example, change to leds=[Pin(15,Pin.OUT),Pin(14,Pin.OUT),Pin(13,Pin.OUT),Pin(12,Pin.OUT),Pin(18,Pin.OUT),Pin(19,Pin.OUT)],n = (n+1)%6 . You can see a row of lights dancing. You can even add more. Try it yourself.



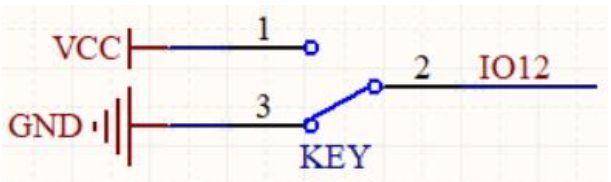
---

3. Lesson3: Button

The LED light in previous lesson is about the usage of output GPIO port. In the lesson, we will learn how to input the signals to the board, by learning the usage of input of GPIO port by pressing buttons. The MaESP ESP32 board will sense the button input, and thus to control the LED ON/OFF.

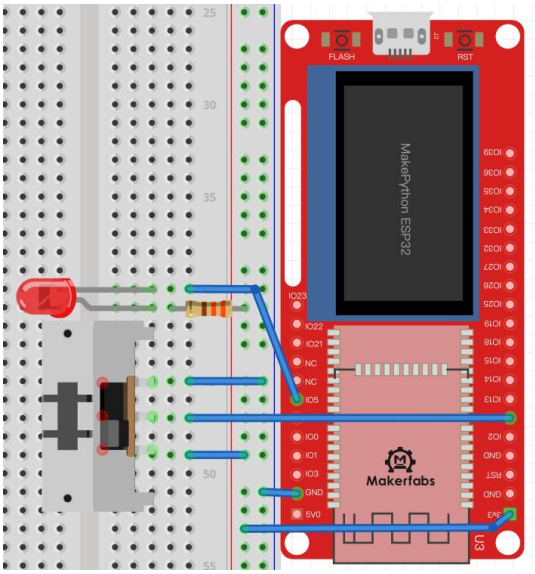
Material:

- 1*Button Module;
- 1* LED;
- resistance:1*330R;
- Jump wires;



Instructions: When the button is pressed, VCC is connected to OUT, and OUT pin outputs high level; when the key is released, GND is connected to OUT, and OUT pin outputs low level

Wiring: LED wiring is the same as Lesson1. Button VCC pin connects board 3V3, GND connects board GND, OUT pin connects IO12:



Create a new **Button** file with the following code and comments:

```
from machine import Pin
import time
button=Pin(12,Pin.IN, pull=None)          #create Button object from pin12
led=Pin(5,Pin.OUT)                        #create LED object from pin5,Set Pin5 to output
try:
  while True:
    if(button.value() == 1):              #Press the button, the value is 1, release the button, the value is 0
      led.value(0)                        #turn on
    else:
      led.value(1)                        #turn off
except KeyboardInterrupt:
        pass
```

Grammar explanation:

- button=Pin(12,Pin.IN):

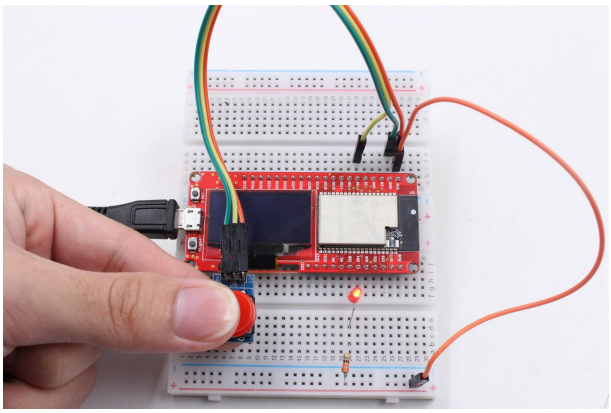
Create a key object and set it to input mode

- if...else:

Statement judge, If true, execute the statement after the if. Otherwise, execute the statement after the else

Results:

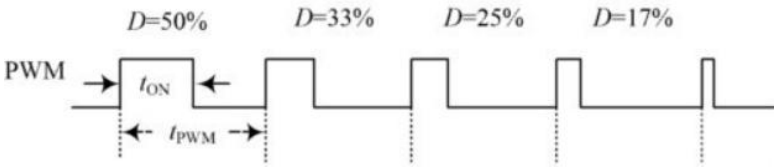
Press the button, the LED light is will be on, while release the button, the LED light is off;



---

4. Lesson4: PWM Control

Pulse width modulation (PWM) is a method of obtaining artificial analog output on digital pins. It does this by quickly switching pins from low to high. There are two related parameters: switching frequency and duty cycle. Duty cycle is defined as the time a pin is at a high level compared to the length of a single cycle (low level to high level time).The maximum duty cycle is that the guide foot is always at high level while the minimum duty cycle is always at low level



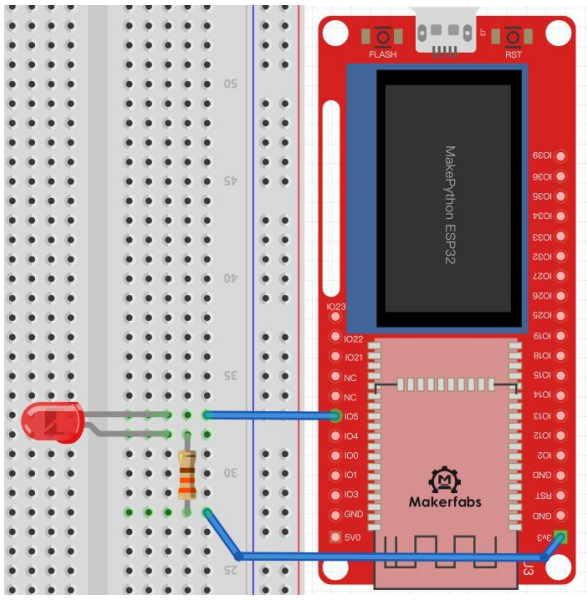
We will demonstrate how to use the ESP32 board to generate a PWM signal. For example, by changing duty cycle to reduce the brightness of LED, control steering gear rotation Angle.

Material:

- 1*Servomotor;
- 1*LED;
- 1*330R;
- Breadboard and jump wires;

Instructions: By adjusting the ratio of LED brightness to off in one cycle, the LED brightness can be adjusted. For example, the control cycle of PWM is 100ms, where 33ms is high level and 67ms is low level, and the duty cycle is 33/100=33%

Wiring: Connect the long LED pin to the end of the 330R resistor, the other end of the resistance is connected to 3.3v, the short pin of the LED lamp is connected to the IO5 of ESP32. As follows:



Create a new **PWM** file with the following code and comments:

```
from machine import Pin, PWM
import time
frequency = 5000                    #set frequency
led = PWM(Pin(5), frequency)        #create PWM object from pin5,Set frequency
try:
  while True:
    for duty_cycle in range(0, 1024):   #duty_cycle cycles between 0 and 1024
      led.duty(duty_cycle)              #duty cycle size of LED lights
      time.sleep_ms(2)                  #delay of 2 ms
except KeyboardInterrupt:
          pass
```

Grammar explanation:

- led = PWM(Pin(5), frequency):

Create the PWM object and set the default frequency

- range():

The range() function creates a list of integers and is used in a for loop

- *for:*

the for loops can traverse any sequence of items, such as a list or a string

- led.duty():

The larger the duty_cycle value of the led, the brighter the led.

Results:

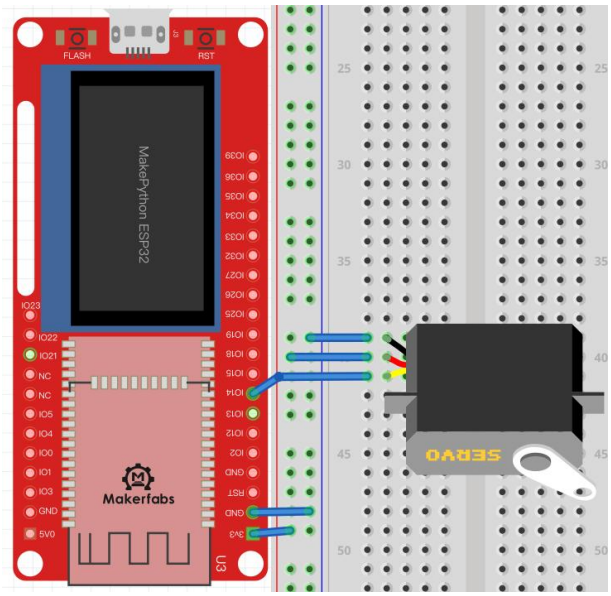
The LED brightness goes from full bright to dark, then full bright again, and the cycle around, like a breath, which we called “breathing LED”

Servo motor is widely used in robot applications. It is an automatic control system composed of dc motor, reduction gear, sensor and control circuit, the rotation angle of the output shaft determined by the input **PWM** signal.

Instructions: Orange wire: Signal , Red wire: Power , Brown wire: GND.

Users control the Servo by sending them a fixed frequency square wave signal (usually 50Hz for analog servers, but digital ones may also accept up to 300Hz), we use the duty method to set the Angle, calling the **servo.duty ()** method to change the set servo angle

Wiring: The orange line is connected to IO14, the red line is connected to 3.3v, and the brown line is connected to GND：



Create a new **Servo_Demo** file with the following code and comments:

```
import machine
import time
p14 = machine.Pin(14)
servo = machine.PWM(p14,freq=50)
try:
   for i in range(0, 180):
        servo.duty(i)
        time.sleep_ms(10)
except KeyboardInterrupt:
        pass
```

Grammar explanation:

- servo = machine.PWM(p14,freq=50):

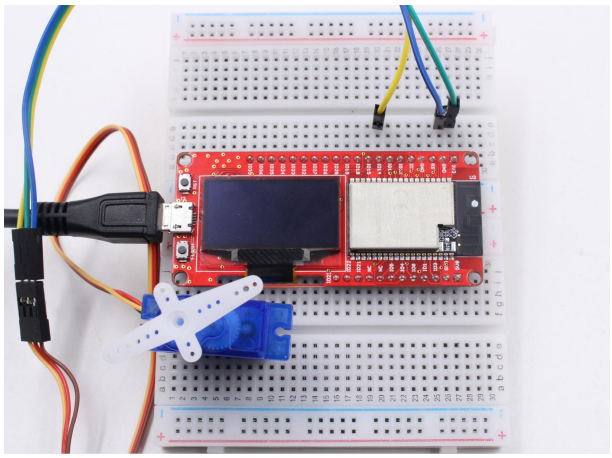
Create the PWM object and set the frequency to 50Hz

- servo.duty(i):

Adjust the duty cycle

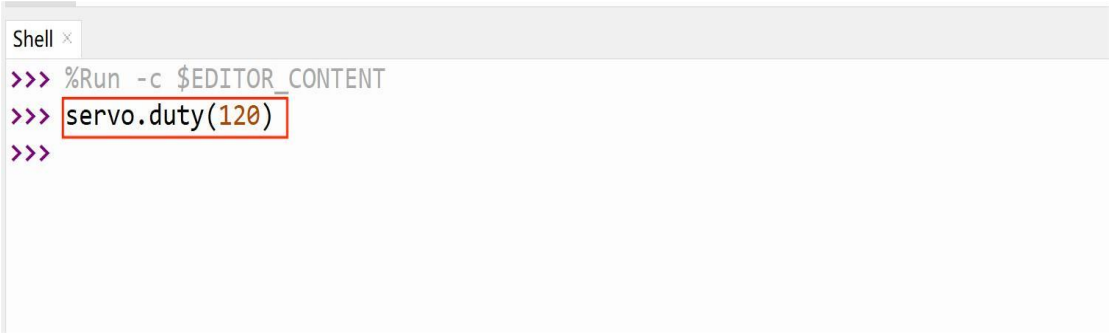
Results:

The servo motor keeps turning.



More important: You can also sent the commands to MaESP board to make the servo rotate to any position you want, which is much more convenient in robot application debugging:

```
>>>servo.duty(120)
```



Then the servo will rotate to the position of 120 degree. You can adjust the position by sending commands.

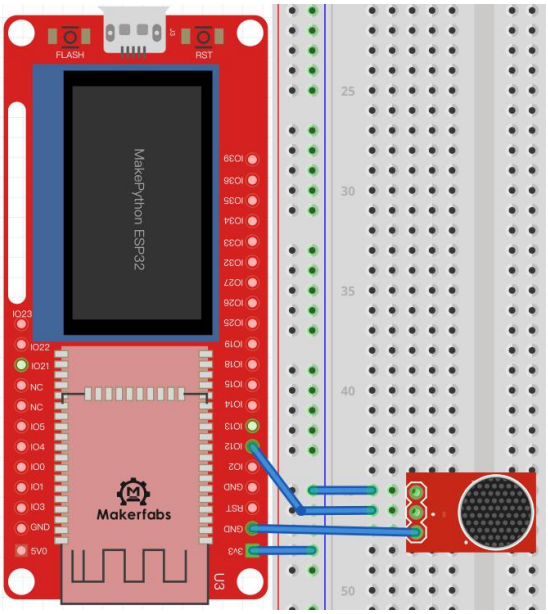
---

5. Lesson5: Voice Control

This experiment will use the buzzer and sound sensor module to do some experiments on sound.

Instructions: according to the driving mode is mainly divided into active buzzer and passive buzzer, active buzzer needs dc voltage can drive, passive buzzer needs a specific frequency of vibration signal to drive.The experiment use an active buzzer.

Wiring： VCC is connected to 3V3,GND is connected to GND, and I/O is connected to IO12:



Create a **Buzzer** file with code and comments:

```
from machine import Pin,PWM
import time
pwm=PWM(Pin(12,Pin.OUT))
def ambulenceSiren(pwm):
    pwm.freq(400)
    pwm.duty(512)
    time.sleep_ms(500)
    pwm.freq(800)
    time.sleep_ms(500)
try:
  while True:
      ambulenceSiren(pwm)
except KeyboardInterrupt:
      pwm.deinit()
      pass
```

Grammar explanation:

- Def:

Define your own functions, beginning with the def keyword, followed by the function identifier name and parentheses (), within which you can pass arguments and arguments.

- pwm.freq(400):

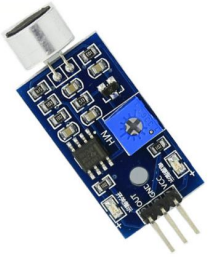
Set the frequency at 400Hz, different frequencies produce different sounds

Results:

You can hear the sound of an ambulance whistle, but you can also modify **pwm.freq()** to simulate other sounds.

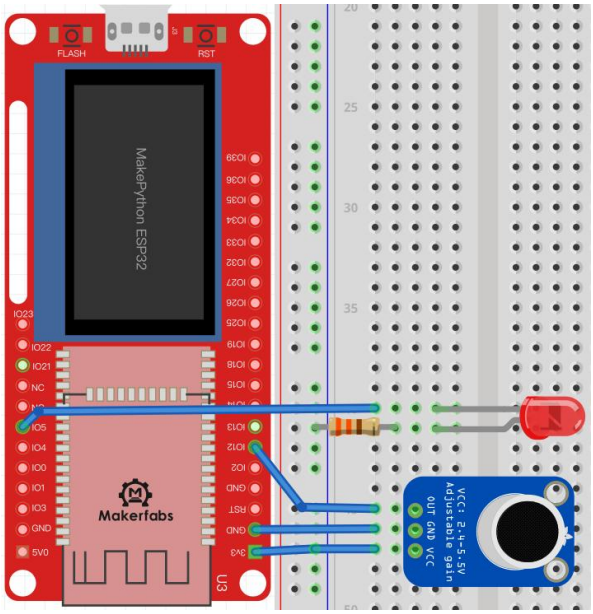
With the sound sensor in the kit, we can also make a “sound switch”, that controls an LED On/Off by

single sound:



Instructions: the sound intensity of the surrounding environment can be detected. When the sound of the external environment fails to reach the set threshold, OUT outputs a high level; when the sound intensity of the environment reaches the set threshold, module OUT outputs a low level. The blue digital potentiometer on the module can be used to adjust the sensitivity. If the led lights do not work, try to adjust the sensitivity of the potentiometer.

Wiring: VCC is connected to 3V3, GND is grounded, OUT is connected to IO12, and LED is connected to IO5：



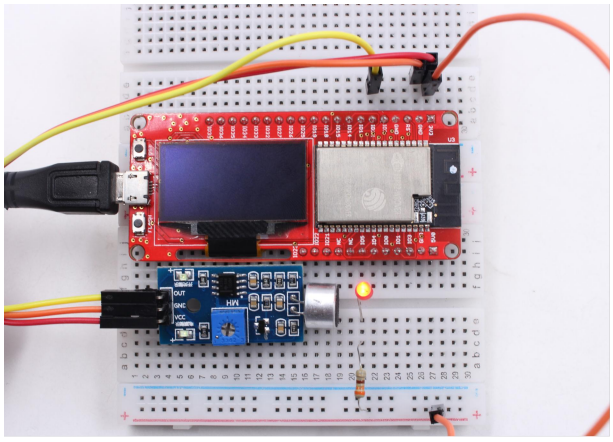
Create **Voice** files, code, and comments:

```
from machine import Pin
import time


voice=Pin(12,Pin.IN)
led=Pin(5,Pin.OUT)
try:
  while True:
    if (voice.value() == 1):
      led.value(0)
      time.sleep(1)
    else :
      led.value(1)
```

```
except KeyboardInterrupt:
        pass
```

Results:

When you clap your hands or snap your fingers, you'll notice that the light will be on for 1 second.



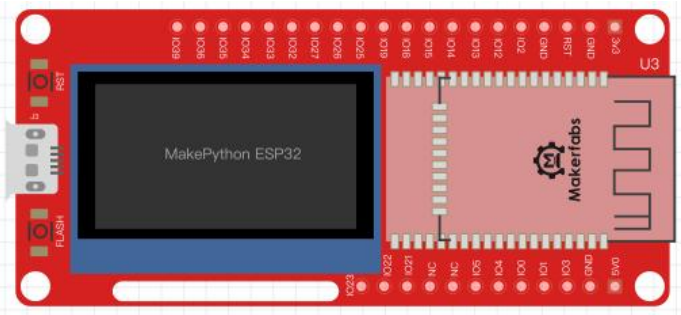
---

6. Lesson6: OLED Display

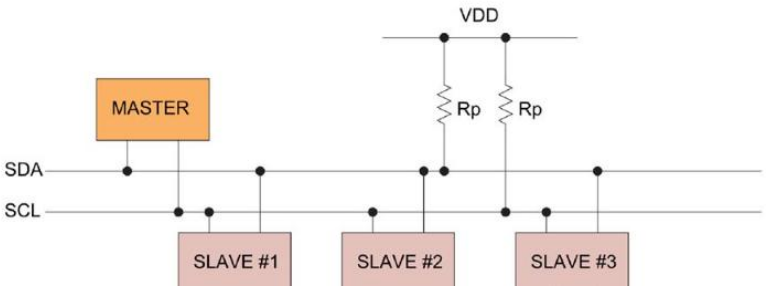
There an on-board OLED 1.3' OLED module on MaESP board, with 128x64 pixel.One pixel of a monochrome screen is a light-emitting diode. OLED is "self-illumination", the pixel itself is the light source, so the contrast is very high. OLED screens have I2C and SPI communication protocols, which are completely incompatible due to different protocols. In our lesson, the OLED is configured to compatible with I2C protocol

Material:

- MaESP ESP32:



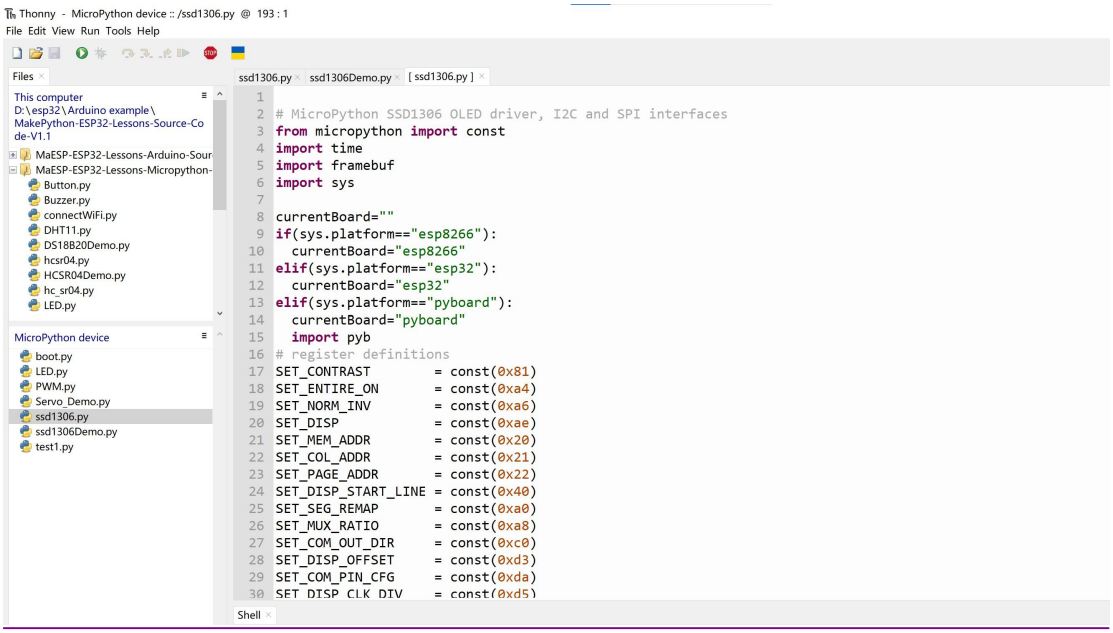
I2C communication: I2C is widely used for controller communicating with onboard components such as sensors/ displays. Data transmission can be completed by only two signal lines, respectively clock line SCL and signal line SDA. There is only one main device Master and several Slave devices on the I2C line. In order to ensure that both buses are at high level when idle, SDA and SCL must be connected with the pull resistor. The classical value of the pull resistor is 10K.



Download library function:

Since the OLED screen use the SSD1306 driver chip, we need to download the driver of SSD1306. You can go to the **github** website to search and download the library of SSD1306 or click the following link:

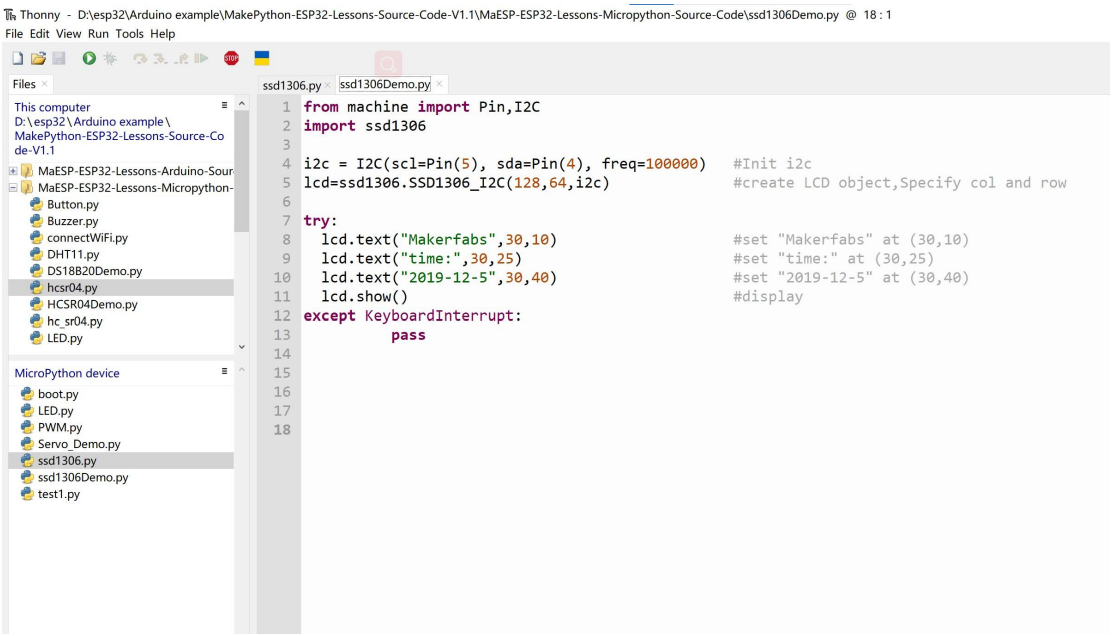
https://www.makerfabs.com/Makepython-esp32-starter-kit.html



Load library function:

After downloading, save **ssd1306.py** to the **Micropython device** file directory.

In the **Micropython device** directory.Users can be called with the **import ssd1306** statement. Create a new **ssd1306demo.py** file to write the code.the library file can be auto-loaded into the device directory click open the **ssd1306Demo.py** file and click run. So As shown in figure:



Create a new **ssd1306Demo** file, code and comments:

```
from machine import Pin,I2C
import ssd1306

i2c = I2C(scl=Pin(5), sda=Pin(4), freq=100000)     #Init i2c
lcd=ssd1306.SSD1306_I2C(128,64,i2c)                    #create LCD object,Specify col and row

try:
  lcd.text("Makerfabs",30,10)                          #set "Makerfabs" at (30,10)
  lcd.text("time:",30,25)                              #set "time:" at (30,25)
  lcd.text("2019-12-5",30,40)                          #set "2019-12-5" at (30,40)
  lcd.show()                                           #display
except KeyboardInterrupt:
          pass
```

Grammar explanation:

- I2C():

Initialize, configure the SCL and SDA pins, and set the frequency at 100000Hz.

- lcd.text():

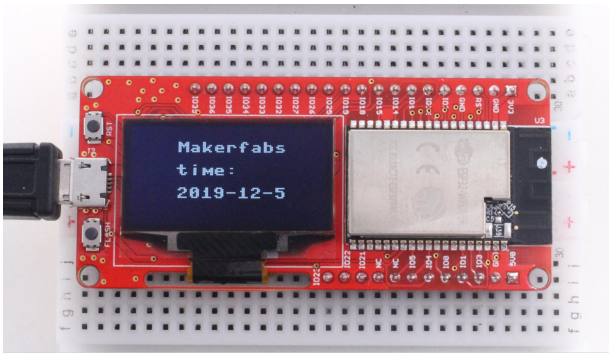
Something to display,The following Numbers are the X and Y axes of the display

- lcd.show():

Turn on display function

Results:

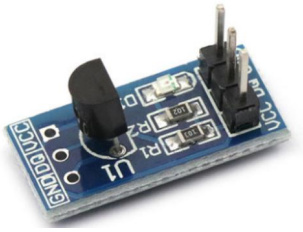
Save and click run, and you'll see Makerfabs/time:/2019-12-5; You can also make it display your own name, etc. Just use the LCD.text() statement to write what you want to display.

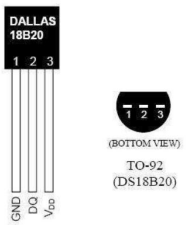


7. Lesson7: Temperature Monitor DS18B20

DS18B20 is a low-cost digital temperature sensor. Its measurement range is between -55 ° c and 125 ° c, with an inherent error of 1 ° c and an accuracy of 0.5 ° c between -10 ° c and 85 ° c, which can meet the daily temperature measurement needs.

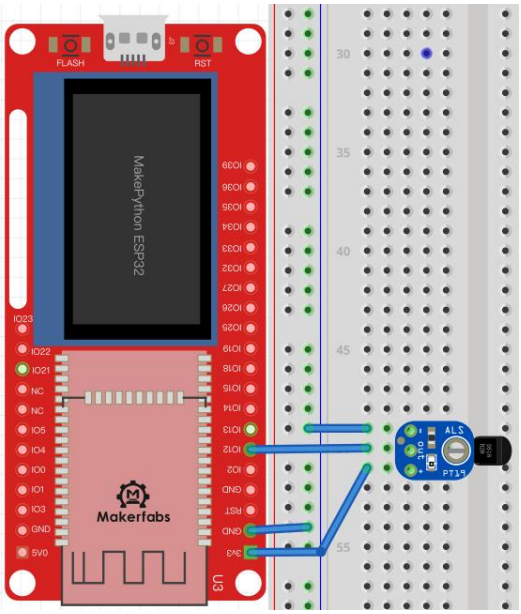
DS18B20 uses OneWire (single-wire) communication protocol, which only needs a signal line to complete the reading and writing of data. Power supply is also allowed to have two modes: independent power supply and data parasitic power supply. In the parasitic power supply mode, the power line is grounded, and the power is obtained from the data line, which can save a connection line. Here we show you how to read data using the OneWire protocol.





Instructions: 1:GND; 2:Data(need to connect the pull resistor a 10k resistor to 3.3v);3:VDD

Wiring: The GND pins are connected, VCC pin is connected to 3v3, and the DQ pin is connected to IO12.



** If there is no ssd1306.py file in the device directory, please download ssd1306.py first. Refer to lesson 6*

Create a new **DS18B20Demo** file, code and comments:

```
from machine import Pin,I2C
import ssd1306
import time
import machine
import onewire, ds18x20

dat = machine.Pin(12)                          # the device is on GPIO12
i2c = I2C(scl=Pin(5), sda=Pin(4), freq=100000)    #Init i2c
lcd=ssd1306.SSD1306_I2C(128,64,i2c)            #create LCD object,Specify col and row

ds = ds18x20.DS18X20(onewire.OneWire(dat))     # create the onewire object
roms = ds.scan()                               # scan for devices on the bus

try:
  while True:
      ds.convert_temp()
      time.sleep_ms(750)                       #The reading temperature needs at least 750ms
      for rom in roms:
          lcd.fill(0)
          lcd.text("temperatures:",10,16)
          lcd.text(str(ds.read_temp(rom)),24,40)
          lcd.show()
except KeyboardInterrupt:
          pass
```

Grammar explanation:

- scan():

scan for devices on the bus

- convert_temp():

Start temperature reading

- read_temp():

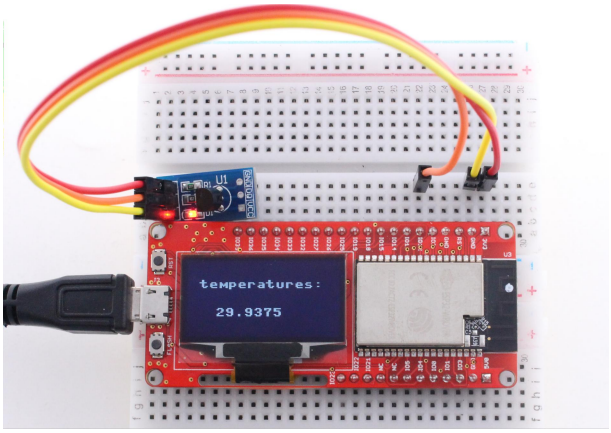
Read the collected temperature

- fill(n):

Empty the screen when n=0, and fill the screen when n> is 0

Results:

After running, the temperature collected on the display screen appears, which is refreshed every 750ms

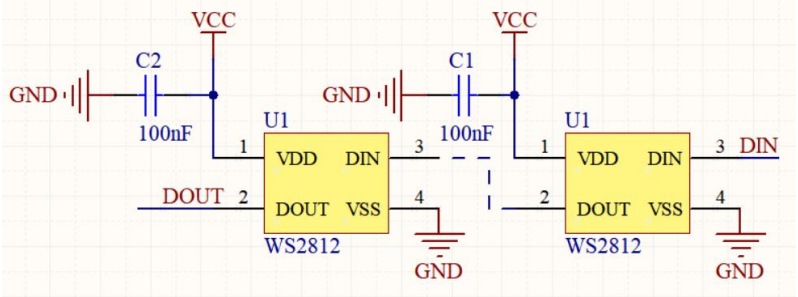


---

8. Lesson8: Digital LED WS2812

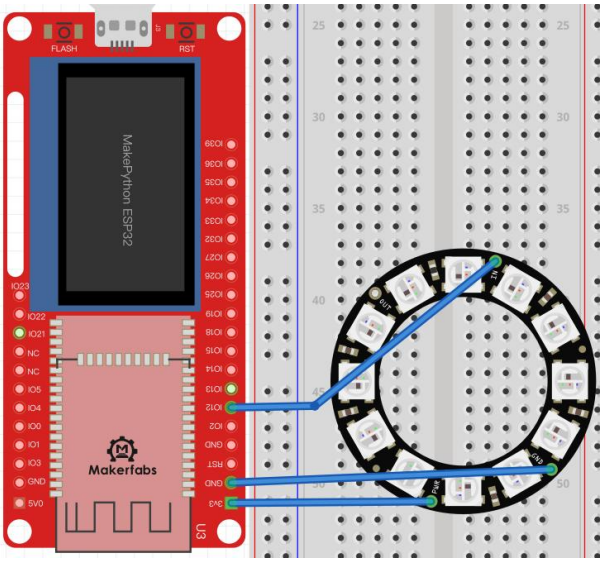
WS2812 is an integrated chip low-power RGB trichromatic lamp, chips and full-color legs wrapped together, only 4 pins, each one chip as a full-color "pixel", each pixel end, DIN client receive the data from the controller transfer to come over, every pixel of transmission, signal 24 bit to reduce, all the way.

This experiment will teach you how to make colorful LED light rings.



Wiring: VDD is connected to 3.3V/5V, VSS is connected to GND, DIN is connected to IO12 of the

board:



Create a new **ws2812Demo** file, code and comments:

```
import machine, neopixel
import time

n = 12                                          #Define LED quantity
np = neopixel.NeoPixel(machine.Pin(12), n)      #Create the NeoPixe object

def demo(np):
    # Circular effect.One pixel runs through all strip positions, while the others are closed.
    for i in range(4 * n):
        for j in range(n):
            np[j] = (0, 0, 0)
        np[i % n] = (255, 255, 255)
        np.write()
        time.sleep_ms(25)

    #Rebound effect and set color (R,G,B), as well as wait time.
    #Wait time determines the speed of the bounce effect.
    for i in range(4 * n):
        for j in range(n):
            np[j] = (0, 0, 128)
        if (i // n) % 2 == 0:
            np[i % n] = (0, 0, 0)
        else:
            np[n - 1 - (i % n)] = (0, 0, 0)
        np.write()
        time.sleep_ms(60)
try:
  while True:
```

```
    demo(np)
except KeyboardInterrupt:
        pass
```

Grammar explanation:

- np = neopixel.NeoPixel(machine.Pin(12), n):

Create the NeoPixe object from pin12,set LED quantity

- np[i] = (r, g, b):

r,g,b: red, green, and blue color channels are each divided into 256 degrees of brightness, with values ranging from 0 to 255, with 0 being the darkest and 255 being the brightest

- "*":

Multiply - multiply the two Numbers

- "%":

Mod - returns the remainder of division

- "//":

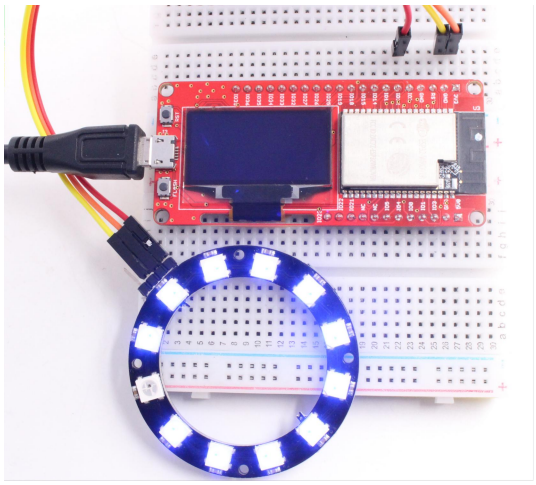
Take the integral part of the divisible - return quotient

- np.write():

Use the write() method to output the colors to the LED

Results:

Running the code, you can see the brightest white light in the loop, followed by the half-bright blue light bouncing back and forth. You can make colored lights by changing the **r**,**g**, and **b** values of np[i] to get different colors and brightness.



---

9. Lesson9: Pulse Sensor

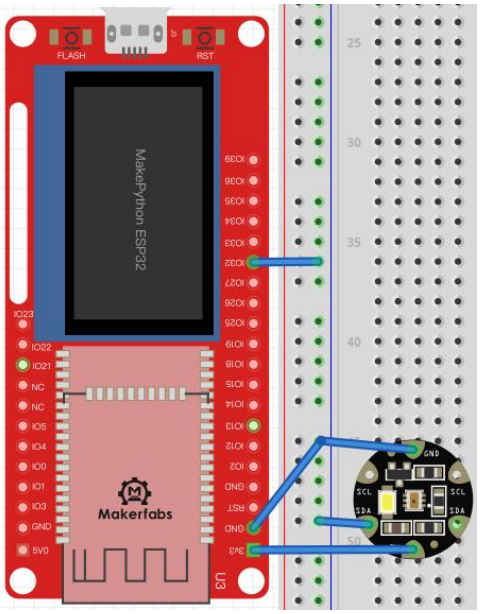
The Pulse Sensor is a photoelectric reflex analog sensor for pulse and heart rate measurement. Wear it on finger, earlobe and so on, make use of the body tissue in the vascular pulse caused by different transmittance to measure pulse. The sensor filters and amplifies the photoelectric signal, and finally outputs the simulated voltage value. Then the heart rate value can be obtained by simple calculation.

In this experiment, ADC will be learned through the Pulse Sensor module, and the results will be displayed on the OLED screen.

Instructions: On the left is the s-pin of the signal output, in the middle is the positive power supply VCC (3.3v ~5V), and on the right is the negative power supply GND.

On the ESP32 ADC functionality is available on Pins 32-39. Note that, when using the default configuration, input voltages on the ADC pin must be between 0.0v and 1.0v (anything above 1.0v will just read as 4095). Attenuation must be applied in order to increase this usable voltage range.

Wiring: The s-pin is connected to IO32, the middle pin is connected to 3V3, and the "-" pin is connected to GND:



** If there is no ssd1306.py file in the device directory, please download ssd1306.py first. Refer to lesson 6*

Create a new **pulse** file, code and comments:

```
from machine import Pin,ADC
import machine
from micropython import const
import time
import ssd1306
import math
```

```
pscl = machine.Pin(5, machine.Pin.OUT)
psda = machine.Pin(4, machine.Pin.OUT)
i2c = machine.I2C(scl=pscl, sda=psda)
oled = ssd1306.SSD1306_I2C(128, 64, i2c)

counttime=0
try:
  while True:
    adc = ADC(Pin(32))                    #Create ADC object on ADC pin
    adcValue=adc.read()                   #read value, 0-4095 across voltage range 0.0v - 1.0v
    adc.atten(ADC.ATTN_11DB)              #set 11dB input attenuation (voltage range roughly 0.0v - 3.6v)
    adc.width(ADC.WIDTH_10BIT)            # set 10 bit return values
    adcValue=adc.read()                   # read value using the newly configured attenuation and width
    oled.line(counttime,40,counttime,adcValue-420,1)      #Drew the heart rate
    oled.show()
    print(adcValue)
    time.sleep_ms(1)
    counttime+=1

    if(counttime>127):
      counttime=0
      oled.fill(0)                        #Clear the screen
      oled.show()
except KeyboardInterrupt:
      pass
```

Grammar explanation:

- counttime:

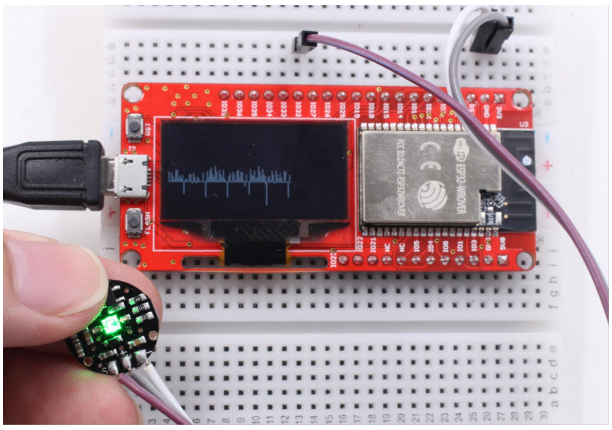
Cache x coordinates (x less than 128)

- line(x,y,x1,y1,c):

Line function,(x,y): starting point coordinate,(x1,y1): ending point coordinate,c: color

Results:

Touch the side printed with the heart with your finger, the screen will display your heart rate beat line, the module contact instability or vibration, will affect the measurement data, please save the static measurement.

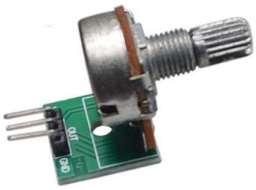


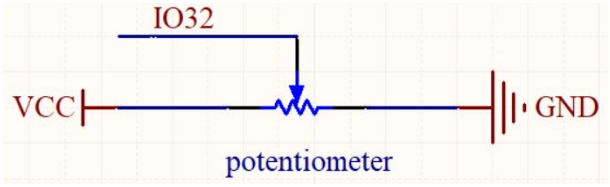
.

---

10. Lesson10: AnalogRead

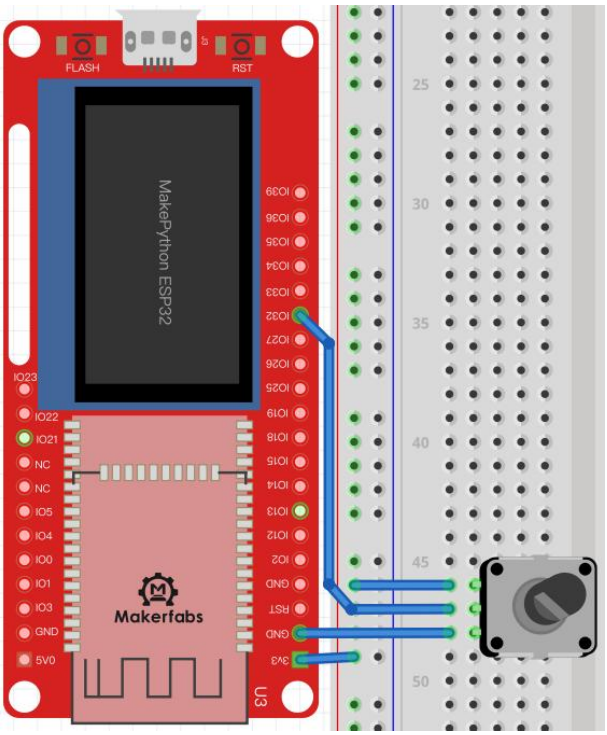
ADC is an Analog/Digital Converter that converts Analog signals into Digital. In the front control LED on, PWM inside, we know the difference between digital signal and analog signal. The signals we use in everyday life, such as light intensity, sound waves, and battery voltages, are all analog values. If we want to measure the analog signal (voltage, light intensity, sound wave) through the single-chip microcomputer and express it by digital signal, then we need ADC analog digital signal converter





Instructions: Potentiometer is an adjustable resistor with three leading ends and resistance values that can be adjusted according to a certain variation law. A potentiometer usually consists of a resistor body and a movable brush. When the brush moves along the resistance body, the resistance value or voltage in relation to the displacement is obtained at the output end.

Wiring : The pins on the left and right sides are connected to 3.3V and GND of the plate respectively, the middle pin is connected to IO32:



** if there is no ssd1306.py file in the device directory, please download ssd1306.py first. Refer to lesson 6*

Create a new **ssd1306_adc** file, code and comments:

```
from machine import Pin,ADC
import machine
from micropython import const
import time
import ssd1306
import math

pscl = machine.Pin(5, machine.Pin.OUT)
psda = machine.Pin(4, machine.Pin.OUT)
i2c = machine.I2C(scl=pscl, sda=psda)
oled = ssd1306.SSD1306_I2C(128, 64, i2c)

def circle(x,y,r,color,fill=0):
    if(fill==0):
            for i in range(x-r,x+r+1):
                oled.pixel(i,int(y-math.sqrt(r*r-(x-i)*(x-i))),color)
                oled.pixel(i,int(y+math.sqrt(r*r-(x-i)*(x-i))),color)
            for i in range(y-r,y+r+1):
                oled.pixel(int(x-math.sqrt(r*r-(y-i)*(y-i))),i,color)
                oled.pixel(int(x+math.sqrt(r*r-(y-i)*(y-i))),i,color)
    else:
            for i in range(x-r,x+r+1):
                a=int(math.sqrt(r*r-(x-i)*(x-i)))
```

```
                    oled.vline(i,y-a,a*2,color)

                for i in range(y-r,y+r+1):
                    a=int(math.sqrt(r*r-(y-i)*(y-i)))
                    oled.hline(x-a,i,a*2,color)

try:
  while True:
    adc = ADC(Pin(32))                  #Create ADC object on ADC pin
    adcValue=adc.read()                 #read value, 0-4095 across voltage range 0.0v - 1.0v
    adc.atten(ADC.ATTN_0DB)             #0dB attenuation, gives a maximum input voltage of 1v
    adc.width(ADC.WIDTH_12BIT)          # set 12 bit return values, this is the default configuration
    adcValue=adc.read()                 # read value using the newly configured attenuation and width
    time.sleep_ms(10)
    if(adc.read() == adcValue):
      oled.fill(0)
      circle(64,32,adcValue//64,1,1)
      oled.show()
except KeyboardInterrupt:
      pass
```

Grammar explanation:

- adc.read():

Read ADC sampled data

- adc.atten():

This method allows for the setting of the amount of attenuation on the input of the ADC. This allows for a wider possible input voltage range, at the cost of accuracy

- adc.width():

This method allows for the setting of the number of bits to be utilised and returned during ADC reads

- def circle():

Custom draw circle function that USES sqrt () function to calculate the radius of the circle

- math.sqrt(r):

Returns the square root of the number

- pixel(x,y,c):

Draw the point at (x,y)

- hline(x,y,w,c):

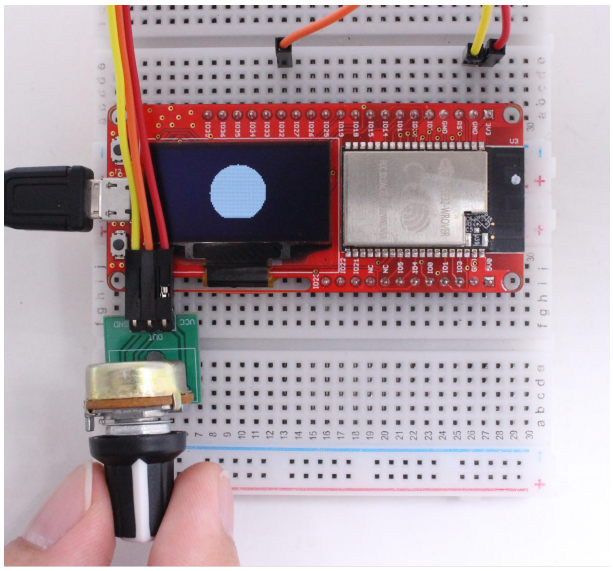
Draw a horizontal line, starting at (x,y), length w

- vline(x,y,w,c):

Draw a vertical line, starting at (x,y), with a height of w

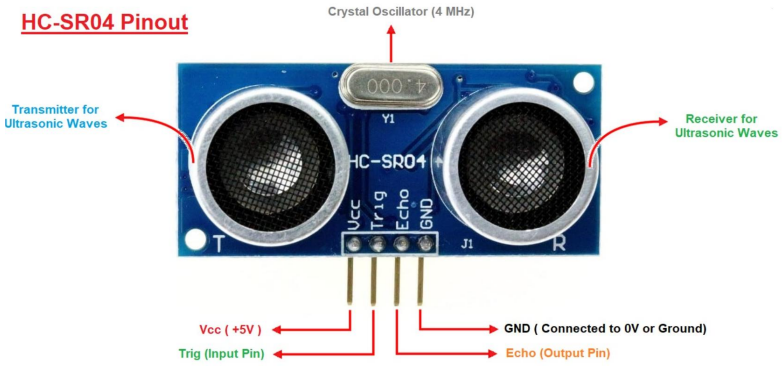
Results:

By rotating the potentiometer, the circle on the OLED display becomes larger or smaller.

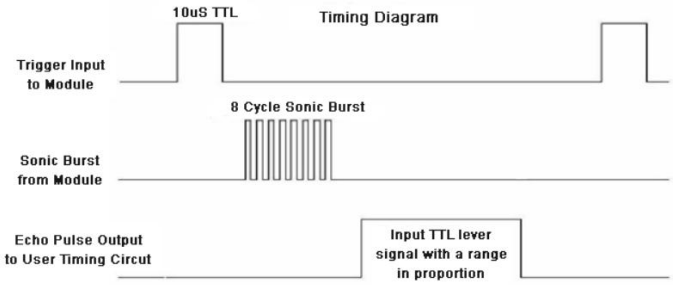


11. Lesson11: Ultrasonic Ranging

The ultrasonic module is for obstacle& distance measurement. It has stable performance, accurate measurement distance and high precision. The module includes ultrasonic transmitter, receiver and control circuit. Application: robot obstacle avoidance, object ranging, etc



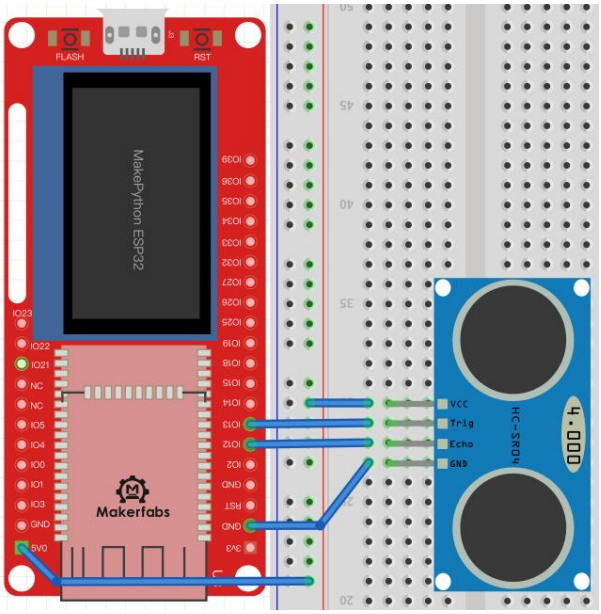
Sequence diagram:



Ranging principle: to module 1 at least 10 us high level, began after launch eight modules receive the high level of 40 KHZ sound waves, echo the feet will be changed from 0 to 1, MCU start timing, when the ultrasonic module receives the returned acoustic echo from 1 to 0, MCU stop timing, this time is the total

time range, in a voice the speed of 340 m/SEC, in addition to 2 is distance.

Wiring: VCC is connected to 5V, GND is connected, Trig is connected to IO13, and Echo is connected to IO12:



Library function download link:

https://www.makerfabs.com/Makepython-esp32-starter-kit.html

As before, place the **hcsr04.py** file in the Micropython directory, it can auto-load it into the device directory when rhe program name of hc_sr04.py running.

** if there is no ssd1306.py file in the device directory, please download ssd1306.py first. Refer to lesson 6*

Create a new **hc_sr04** file, code and comments:

```
from hcsr04 import HCSR04
from machine import Pin,I2C
import ssd1306

i2c = I2C(scl=Pin(5), sda=Pin(4), freq=100000)          #Init i2c
lcd=ssd1306.SSD1306_I2C(128,64,i2c)

sensor = HCSR04(trigger_pin=13, echo_pin=12,echo_timeout_us=1000000)

try:
  while True:
    distance = sensor.distance_cm()
    print(distance)
    lcd.fill(0)
    lcd.text("Distance:",30,20)
    lcd.text(str(distance),30,40)
    lcd.show()
except KeyboardInterrupt:
        pass
```

Grammar explanation:

- HCSR04():

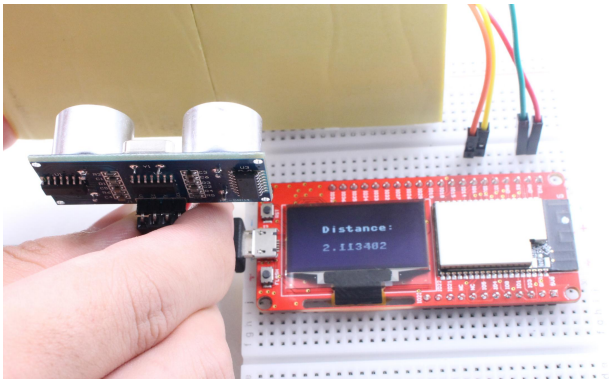
Set IO port and timeout reminder,

- sensor.distance_cm():

Returns the result of the test distance.

Results:

Point the module at objects in different directions and the OLED displays the distance.



Expand:

Calling the corresponding library is very convenient, but if you can't find the right library or do not have this library, you need to write the module driver and corresponding procedures, we use ultrasonic module to do experiments, write the driver to achieve its function.

First understand the working principle of the module, look at the sequence diagram, for better understanding. You can review how this module works. Step 1: you need to give the Trig pin a high level of 10us, step 2: calculate the time when the Echo pin is in the high level, and step 3: calculate the distance according to the time.

Create a new **HC-SR04Demo** file,**Code and comments:**

```
from machine import Pin,I2C
import ssd1306
import machine
import time
i2c = I2C(scl=Pin(5), sda=Pin(4), freq=100000)          #Init i2c
lcd = ssd1306.SSD1306_I2C(128, 64, i2c)

trig = Pin(13,Pin.OUT, pull=None)
echo = Pin(12, Pin.IN, pull=None)
trig.value(0)
echo.value(0)

def send_pulse_and_wait(echo_timeout_us=500*2*30):
        trig.value(0) # Stabilize the sensor
        time.sleep_us(5)
        trig.value(1)
```

```
        # Send a 10us pulse.
        time.sleep_us(10)
        trig.value(0)
        try:
            pulse_time = machine.time_pulse_us(echo, 1, echo_timeout_us)
            return pulse_time
        except OSError as ex:
            if ex.args[0] == 110: # 110 = ETIMEDOUT
                raise OSError('Out of range')
            raise ex

def distance_cm():
        pulse_time = send_pulse_and_wait()
        cms = (pulse_time / 2) / 29.1
        return cms

def main():
  try:
    while True:
      distance = distance_cm()
      print(distance)
      lcd.fill(0)
      lcd.text("Distance:",30,20)
      lcd.text(str(distance),30,40)
      lcd.show()
  except KeyboardInterrupt:
            pass

main()
```

Grammar explanation:

- time.ticks_us():

Returns an incrementing millisecond counter with any reference point that ends after some value (not specified).This value should be considered opaque and applies only to ticks_diff().

- time.ticks_diff(t2,t1):

Measure the period between successive calls to ticks_ms() and ticks_us(). Calculate the time from t1 to t2

- pass:

Empty statement, do nothing, used as a placeholder statement

Results:

As before, point the module at objects in different directions and the OLED displays the distance.

12. Lesson12: WiFi

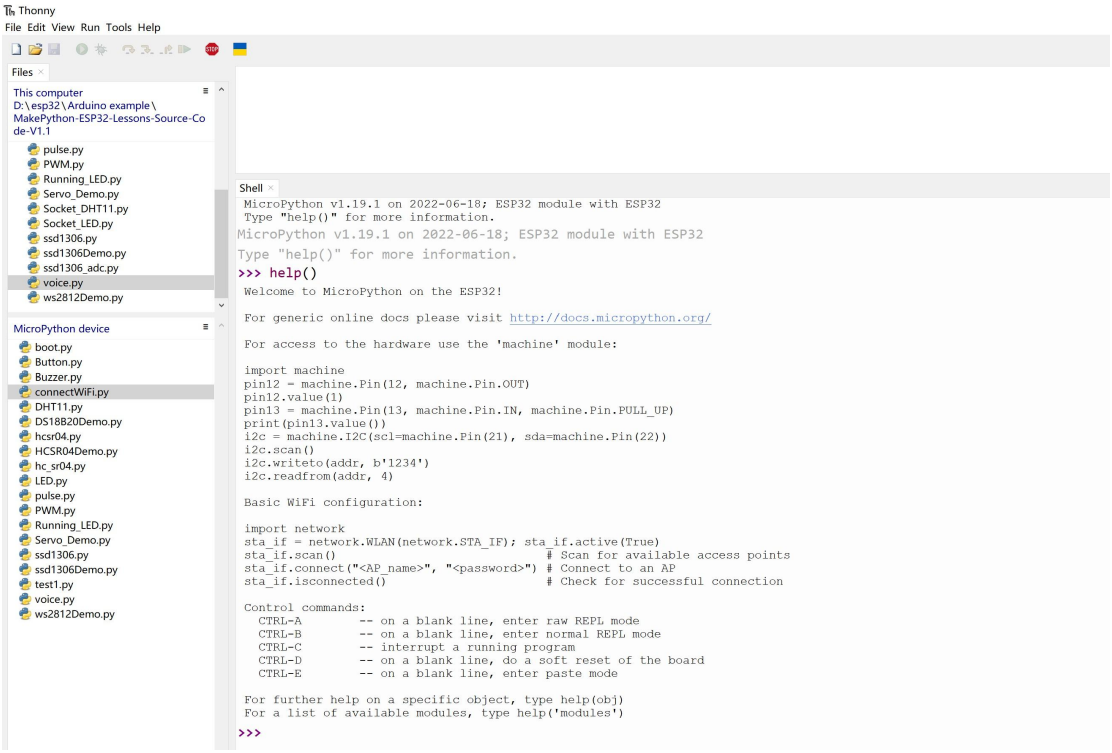
MaESP ESP32 devices have built-in Wifi& Bluetooth, which enables us to access it via Wifi, or connect to your home Wifi network. In this lesson, we will learn how to the files uploading/downloading, that users can remote control/upload MaESP, without the need of a USB cable, and also how to make the MaESP be a web-server, with socket communication.

1. WiFi connection

Enter **help()** in MicroPython Shell/terminal, press enter key, and the printed message will prompt you how to set **STA mode:**

- **STA Mode:**

Each terminal connected to a wireless network can be called a site.



You can enter the following code one by one in the MicroPython Shell/terminal to connect:

- Connect to Internet(STA):

```
>>> import network
>>> wlan=network.WLAN(network.STA_IF)     #create access-point interface
>>> wlan.active(True)                     #activate the interface
>>> wlan.scan()                           #Scan access point
>>> wlan.connect('ap_name','password')    #Connect to AP (local WIFI and password)
>>>wlan.isconnected()                     #Check that the workstation is connected to the AP
>>> wlan.ifconfig()                       #Get the IP/netmask/gw/DNS address of the interface
```

Grammar explanation:

- wlan.active(True):

The activation interface will display:#5 ets_task(4020f4d8, 28, 3fff9e30, 10)

- wlan.scan():

Scan the access point and display the network around you

- wlan.connect('ap_name','password'):

Change 'ap_name' and 'password' to your local network name and password to connect

- wlan.isconnected():

If the connection is successful, it will return True, if the connection is unsuccessful, it will return False. You need to check whether the network is correct

- wlan.ifconfig():

After the connection is successful, the address information of the interface is printed

- ap.config(essid='MKF-AP'):

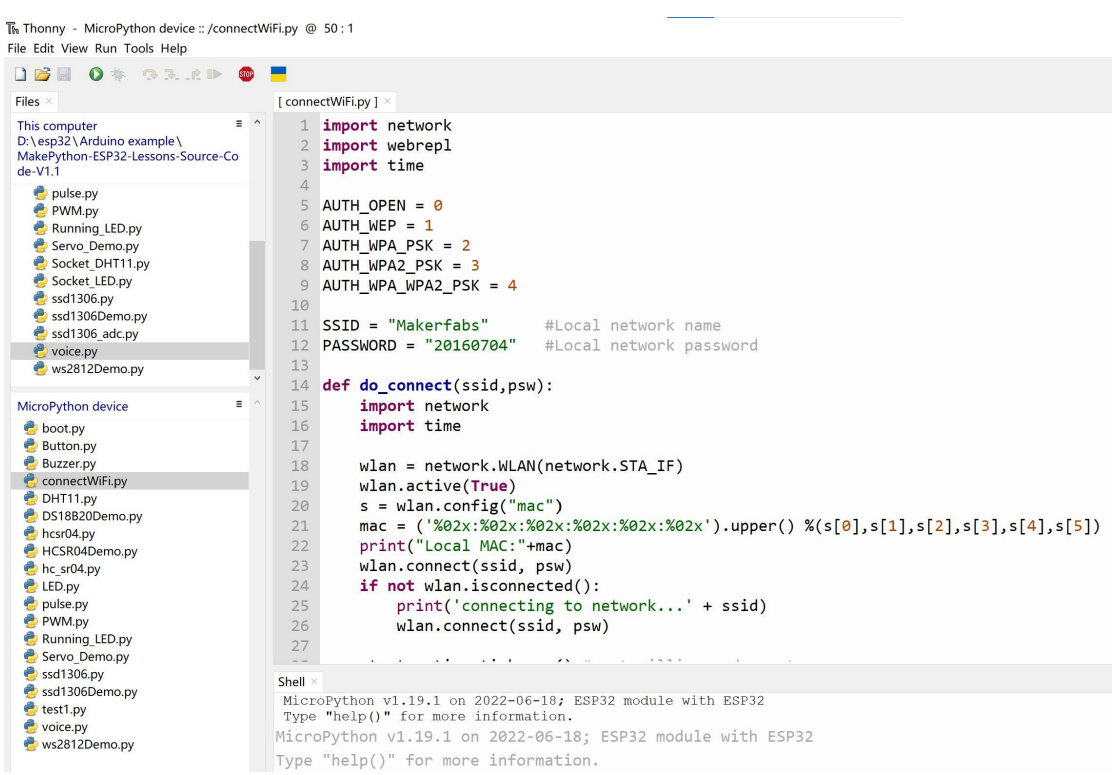
Set AP_name, the password defaults to micropythoN,or you can replace this code with:ap.config(essid='mkf-ap ',authmode=AUTH_WPA_WPA2_PSK,password='12345678') to set the password to12345678,or whatever.

Or you can set the connection to WiFi by calling the library provided by us. Here is the download link of the library:
https://www.makerfabs.com/Makepython-esp32-starter-kit.html

Click to download the **connectwifi.py** file, place it in the **Micropython** directory, open the file and change the **SSID** and **PASSWORD** to local WIF and password.

For example, my WIFI name and password are “Makerfabs” and “20160704”, changed to:

```
SSID = "Makerfabs"
PASSWORD = "20160704"
```



After saving and clicking run, you can see **network config**, followed by the **IP/netmask/gw/DNS** address of the interface:

```
>>> import network
>>> wlan=network.WLAN(network.STA_IF)
>>> wlan.active(True)
True
>>> wlan.scan()
[(b'Makerfabs', b'\xf4*}P\xd2\xec', 7, -57, 4, False), (b'Makerfabs_915_gateway', b'\xaa@A$\xbc
\xc8', 7, -60, 3, False), (b'MERCURY_043864', b'\xc0a\x18\x048d', 12, -71, 0, False), (b'DIRECT
-5B-HP Smart Tank 510', b'\xb2\\\xda\xc5\x16[', 7, -75, 3, False), (b'Makerfabs', b'T\xa7\x03\x
ef\x95i', 7, -79, 4, False), (b'hopecctv6', b'zD\xfd\x06\xd6\xd8', 7, -81, 4, False), (b'ChinaN
et-ALtw', b'\x90\x124\x02=f', 3, -82, 4, False), (b'DIRECT-7F-HP Laser 136w', b'\xf69\tV#\x7f',
1, -85, 3, False), (b'hopecctv6', b'\x9c\xa6\x15\xd0\x15\xc7', 7, -86, 4, False), (b'ChinaNet-2
.4G-5762', b',a\x04\xa8Wb', 12, -86, 3, False), (b'hopecctv62', b't"\xbb\\\xdd\xb4', 11, -89, 3
, False), (b'', b'\xf4\xbf\x80m\xa2\xa9', 11, -94, 3, False)]
>>> wlan.connect('Makerfabs','20160704')
>>> wlan.isconnected()
True
>>> wlan.ifconfig()
('192.168.1.121', '255.255.255.0', '192.168.1.1', '192.168.1.1')
>>>
```

2. Socket communication

Socket communication is the basis of Internet communication. It is the basic operation unit of network communication that supports TCP/IP protocol. To establish Socket communication requires a server side and a client side. This routine will use Python ESP32 as the client and computer browser as the server side. Both sides will use TCP protocol to transmit and receive data from each other.

Example_1: Socket Communication LED Remote control

In this example, we will make the MaESP ESP32 as a server with LED, so the customer can assess the LED control remotely with any computer, as he/she get the right IP address.

The **Socket_LED** source code:

```
#webserver.py
import network
import webrepl
import time
from machine import Pin
try:
  import usocket as socket
except:
  import socket

AUTH_OPEN = 0
AUTH_WEP = 1
AUTH_WPA_PSK = 2
AUTH_WPA2_PSK = 3
AUTH_WPA_WPA2_PSK = 4
```

```
SSID = "xxx"                    #Modify here with SSID
PASSWORD = "xxx"                #Modify here with PWD
led = Pin(5, Pin.OUT)

def web_page():
  html = """<html><head><meta name="viewport" content="width=device-width, initial-scale=1"></head>
  <body><h1>Makerfabs Web Server</h1><a href=\"?led=on\"><button>ON</button></a> 
  <a href=\"?led=off\"><button>OFF</button></a></body></html>"""
  return html

def do_connect(ssid,psw):
    import network
    import time

    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    s = wlan.config("mac")
    mac = ('%02x:%02x:%02x:%02x:%02x:%02x').upper() %(s[0],s[1],s[2],s[3],s[4],s[5])
    print("Local MAC:"+mac) #get mac
    wlan.connect(ssid, psw)
    if not wlan.isconnected():
        print('connecting to network...' + ssid)
        wlan.connect(ssid, psw)

    start = time.ticks_ms() # get millisecond counter
    while not wlan.isconnected():
        time.sleep(1) # sleep for 1 second
        if time.ticks_ms()-start > 20000:
            print("connect timeout!")
            break

    if wlan.isconnected():
        print('network config:', wlan.ifconfig())
    return wlan

def connect():
 do_connect(SSID,PASSWORD)

def app():
  connect()
  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  s.bind(('', 80))
  s.listen(5)
  while True:
```

```
    conn, addr = s.accept()
    print('Got a connection from %s' % str(addr))
    request = conn.recv(1024)
    request = str(request)
    print('Content = %s' % request)
    led_on = request.find('/?led=on')
    led_off = request.find('/?led=off')
    if led_on == 6:
      print('LED ON')
      led.value(0)
    if led_off == 6:
      print('LED OFF')
      led.value(1)
    response = web_page()
    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()

app()
```

Code download link:

https://www.makerfabs.com/Makepython-esp32-starter-kit.html

Grammar explanation:

- web_page():

Create a simple web interface

- s = socket.socket(socket.AF_INET, socket.SOCK_STREAM):

Create the socket object,AF_INET->IPV4,SOCK_STREAM->TCP

- s.bind(address):

Bindings for service roles

- s.listen(i):

Listen, for service roles, I: number of allowed connections, must be greater than 0

- s.accept():

Accept the connection for the service role

- conn.recv():

Receive data

- conn.send():

Send data

- conn.close():

Close data connection

Results:

Change SSID and Password to local network name and Password. The LED wiring is the same as Lesson1. When connecting to the router, you can see the IP address printed out of the serial port:

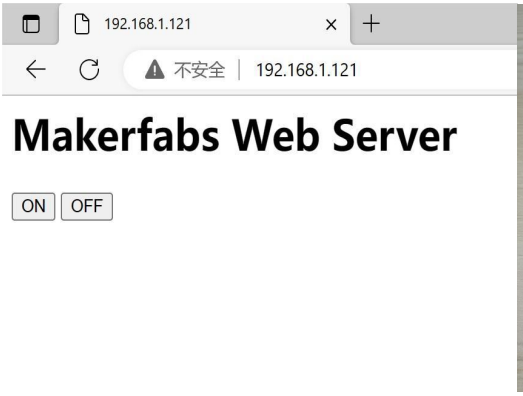
[Socket_LED.py]

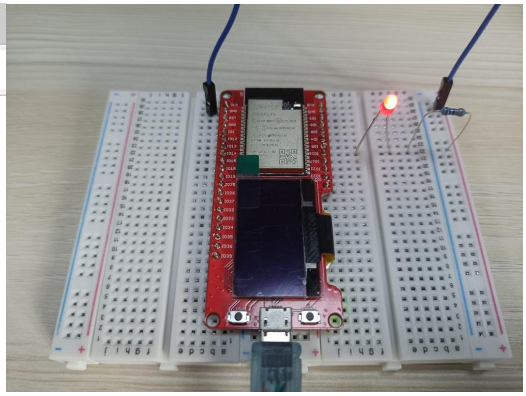
```
#webserver.py
import network
import webrepl
import time
from machine import Pin
try:
  import usocket as socket
except:
  import socket

AUTH_OPEN = 0
AUTH_WEP = 1
AUTH_WPA_PSK = 2
AUTH_WPA2_PSK = 3
AUTH_WPA_WPA2_PSK = 4

SSID = "Makerfabs"       #Modify here with SSID
PASSWORD = "20160704"    #Modify here with PWD
led = Pin(5, Pin.OUT)

```

Shell

```
>>> %Run -c $EDITOR_CONTENT
 Local MAC:24:62:AB:DA:01:28
 network config: ('192.168.1.121', '255.255.255.0', '192.168.1.1', '192.168.1.1')
```

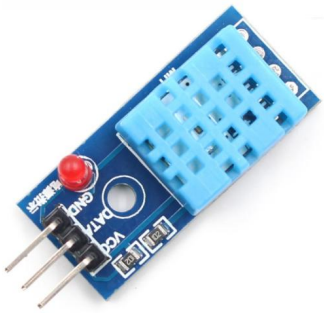
Enter the IP address printed from the serial port in any browser, with any computer in the same WIFI, to control the LED on/off:





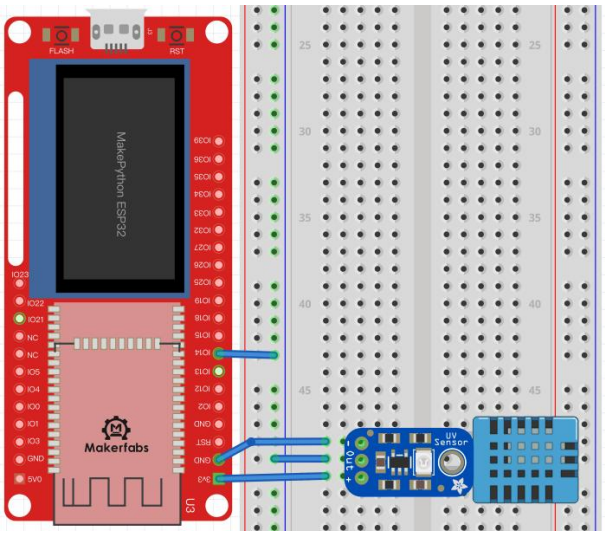
Example_2: Socket Communication Remote Monitoring

In addition to sending command and control LED lights, it can also receive measurement data. The DHT11 module is used in this experiment. With the Socket communication, the MaESP can be also a server to detect environment and report to the Internet for sharing. In this example, the MaESP be a temperature& humidity server, so any PC could assess it to get the related info.



Instructions: DHT11 digital temperature and humidity sensor is sensor containing calibrated digital signal output. It applies special digital module acquisition technology and temperature and humidity sensing technology. The sensor consists of a resistive moisture sensor and an NTC temperature sensor. Its precision humidity +-5%RH, temperature +-2℃, range humidity 20-90%RH, temperature 0~50℃.VCC power positive pole : 3V~ 5.5v,GND: power negative ground, DATA: serial DATA pin;

Wiring: VCC is connected to 3.3v, and DATA is connected to IO14:



Create a new DHT11 file, Socket_DHT11 Sample code:

```
#main.py
import network
import webrepl
import dht
import time
from machine import Pin

try:
    import usocket as socket
except:
    import socket

AUTH_OPEN = 0
AUTH_WEP = 1
```

```
AUTH_WPA_PSK = 2
AUTH_WPA2_PSK = 3
AUTH_WPA_WPA2_PSK = 4

SSID = "Makerfabs"
PASSWORD = "20160704"
sensor = dht.DHT11(Pin(14))

def read_sensor():
    global temp, temp_percentage, hum
    temp = temp_percentage = hum = 0
    try:
        sensor.measure()
        temp = sensor.temperature()
        hum = sensor.humidity()
        print(temp)
        print(hum)
        if (isinstance(temp, float) and isinstance(hum, float)) or (isinstance(temp, int) and isinstance(hum, int)):
            msg = (b'{0:3.1f},{1:3.1f}'.format(temp, hum))
            temp_percentage = (temp+6)/(40+6)*(100)
            hum = round(hum, 2)
            return(msg)
        else:
            return('Invalid sensor readings.')
    except OSError as e:
        return('Failed to read sensor.')

def web_page():

    html = """<html><head><meta name="viewport" content="width=device-width, initial-scale=1">
    <style>body{padding: 20px; margin: auto; width: 50%; text-align: center;}
    .progress{background-color: #F5F5F5;} .progress.vertical{position: relative;
    width: 25%; height: 60%; display: inline-block; margin: 20px;}
    .progress.vertical > .progress-bar{width: 100% !important;position: absolute;bottom: 0;}
    .progress-bar{background: linear-gradient(to top, #f5af19 0%, #f12711 100%);}
    .progress-bar-hum{background: linear-gradient(to top, #9CECFB 0%, #65C7F7 50%, #0052D4 100%);}
    p{position: absolute; font-size: 1.5rem; top: 50%; left: 50%; transform: translate(-50%, -50%); z-index:
5;}</style></head>
    <body><h1>Makerfabs DHT Sensor</h1><div class="progress vertical">
    <p>"""+str(temp)+"""C<p>
    <div role="progressbar" style="height: """+str(temp_percentage)+"""%;"
class="progress-bar"></div></div><div class="progress vertical">
    <p>"""+str(hum)+"""%</p>
```

```
   <div role="progressbar" style="height: """+str(hum)+"""%;" class="progress-bar
progress-bar-hum"></div></div></body></html>"""
   return html

def do_connect(ssid, psw):
      import network
      import time

      wlan = network.WLAN(network.STA_IF)
      wlan.active(True)
      s = wlan.config("mac")
      mac = ('%02x:%02x:%02x:%02x:%02x:%02x').upper() %(s[0],s[1],s[2],s[3],s[4],s[5])
      print(" Local MAC:"+mac)            #get mac
      wlan.connect(ssid, psw)
      if not wlan.isconnected():
            print('connecting to network...' + ssid)
            wlan.connect(ssid, psw)
      start = time.ticks_ms()                  # get millisecond counter
      while not wlan.isconnected():
            time.sleep(1) # sleep for 1 second
            if time.ticks_ms()-start > 20000:
                  print("connect timeout!")
                  break
      if wlan.isconnected():
            print('network config:', wlan.ifconfig())
      return wlan

def connect():
 do_connect(SSID,PASSWORD)

def app():
      connect()
      s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      s.bind(('', 80))
      s.listen(5)

      while True:
         conn, addr = s.accept()
         print('Got a connection from %s' % str(addr))
         request = conn.recv(1024)
         print('Content = %s' % str(request))
         sensor_readings = read_sensor()
         print(sensor_readings)
         response = web_page()
```

```
        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()

app()
```

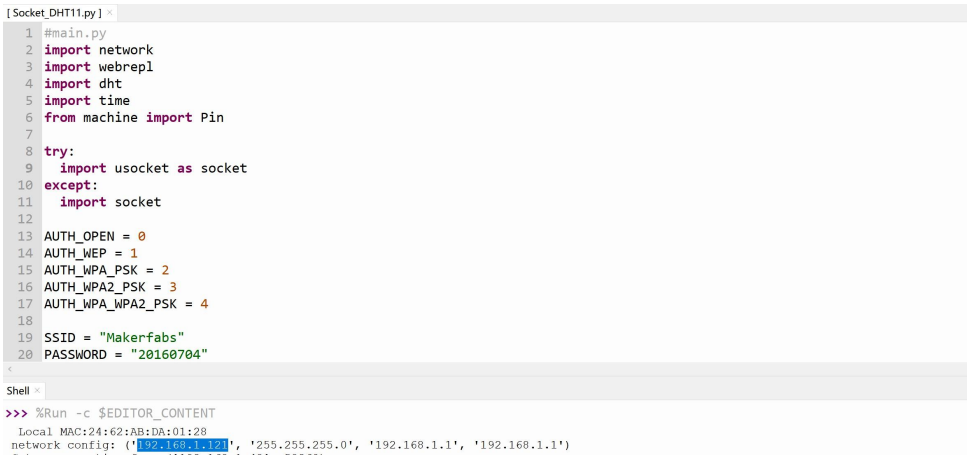
Code download link:

https://www.makerfabs.com/Makepython-esp32-starter-kit.html

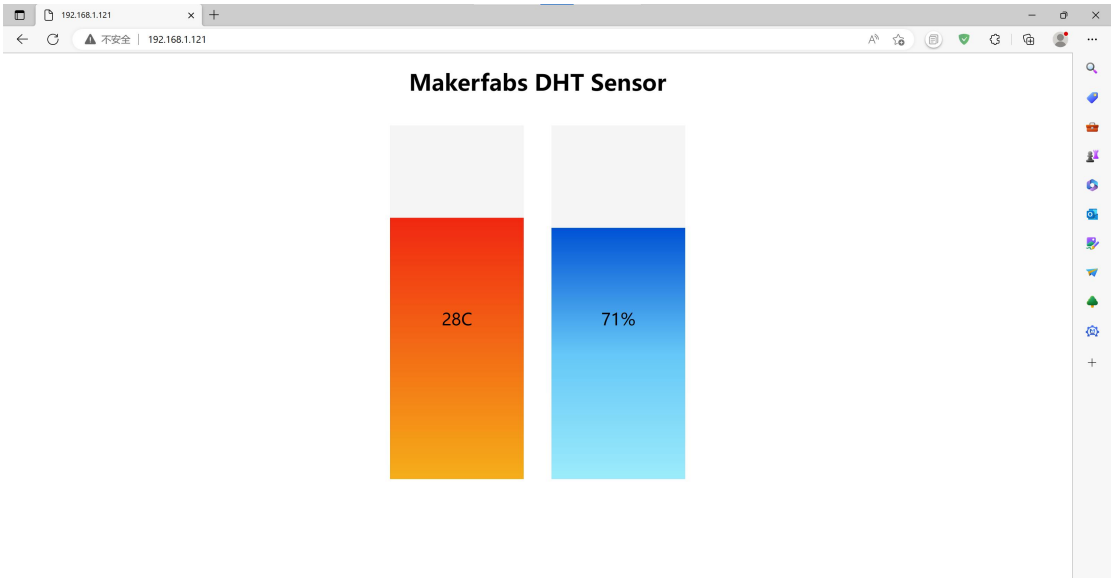
Please refer to the previous examples for code comments and DHT11 wiring. It will not be repeated here.

Results:

Modify **SSID** and **PASSWORD** in the code and restart. If the program runs and connects to the router, the serial port can be seen to print out the IP address:



Open your IP with any computer in the WIFI could access this temperature& humidity monitor now, by entering the IP address in any browser:



***note:** some browsers are not supported, this example was tested using the Chrome browser*

III. Troubleshooting

We've discovered some common problems and error messages that occur with Thonny IDE. Usually restarting your ESP with the on-board RST button fixes your problem. Or pressing the Thonny IDE "Stop" button and repeating your desired action. In case it doesn't work for you, read these next common errors and discover how to solve them

Error #1: You get the following message:

```
>>> Select Serial Port could not open port 'COM4': FileNotFoundError(2, 'The
system cannot find the file specified.', None, 2)
```

Or an equivalent message:

```
>>> could not open port 'COM4': PermissionError(13, 'A device attached to
the system is not functioning.', None, 31)
```

Unplug, and plug back your ESP board. Then, double-check that you've selected the right serial port in the Tools > Serial menu. Then, click the "Connect/disconnect" button to establish a serial communication. You should now be able to upload a new script or re-run new code.

This error might also mean that you have your serial port being used in another program (like a serial terminal or in the Arduino IDE). Double-check that you've closed all the programs that might be establishing a serial communication with your ESP board. Then, unplug and plug back your ESP board. Finally, restart the Thonny IDE – try to select the serial port in the Tools > Serial menu.

Error #2: Trouble uploading a new script.

```
>>> already in download model,please wait.
```

Press the "Stop" button in uPyCraft IDE (1 or 2 times) to make sure any code that was running stops. After that, press the "Download and run" button to upload the new script to your ESP board.

Error #3: After uploading a new script, if you see the following message:

```
>>>
Ready to download this file,please wait!
...
download ok
os.listdir('.')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'os' isn't defined
```

Or this message:

```
>>>
Ready to download this file,please wait!
...
download ok
os.listdir('.')
OSError: [Errno 98]
```

It means the new file was uploaded to your board successfully. You can notice that it printed the "download ok" message. Press the ESP on-board "EN/RST" button to restart your board and re-run the new uploaded script from the beginning.

Error #4: Problem restarting your ESP board, running a new script or opening the serial port:

```
>>>
Brownout detector was triggered
```

The "Brownout detector was triggered" error message means that there's some sort of hardware problem. It's often related to one of the following issues:

- Poor quality USB cable;
- USB cable is too long;
- Board with some defect (bad solder joints);
- Bad computer USB port;
- Or not enough power provided by the computer USB port

Solution: try a different shorter USB cable (with data wires), try a different computer USB port or use a USB hub with an external power supply.

Important: *if you keep having constant problems or weird error messages, we recommend re-flashing your ESP board with the latest version of MicroPython firmware: Flash/Upload MicroPython Firmware to ESP32.(The next chapter will have specific tutorials)*

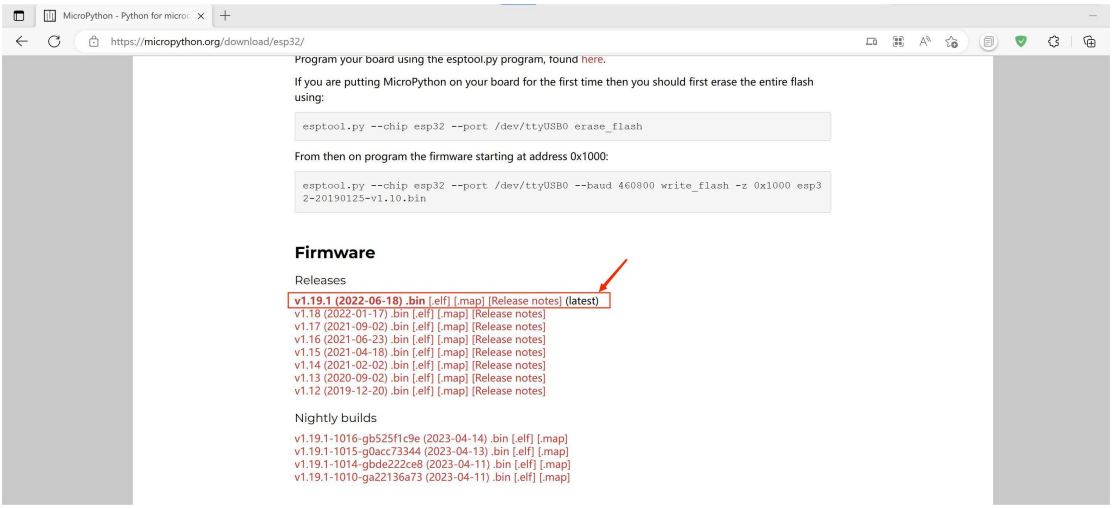
Error #5: When I try to open a serial communication with the ESP32 in Thonny IDE, sometimes it prompts the "Burn Firmware" window asking to re-flash the MicroPython firmware.

Basically, we think this is what's happening: when you're running a script in your board, sometimes it's busy running that script and performing the tasks. So, you need to try opening the COM port multiple times or restart the ESP to catch it available to establish the serial communication with Thonny IDE.
If you're running a script that uses Wi-Fi, deep sleep, or it's doing multiple tasks, I recommend trying 3 or 4 times to establish the communication. If you can't, I recommend re-flash the ESP with MicroPython firmware.

IV. Flash/Upload MicroPython Firmware to ESP32

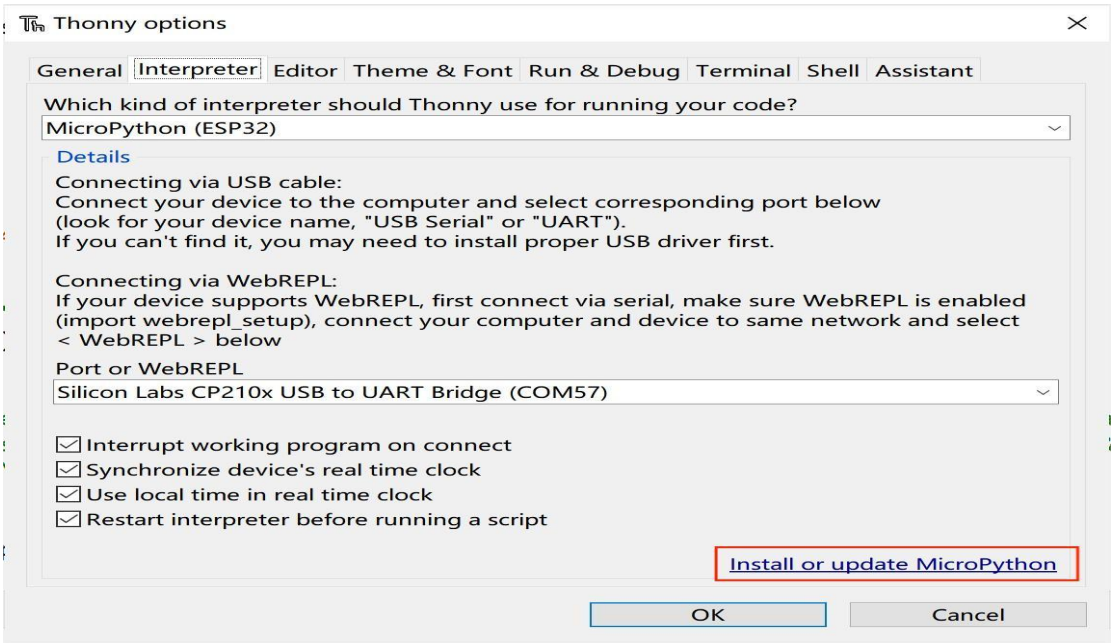
This chapter shows how to flash MicroPython firmware to ESP32 boards using the Thonny IDE software. It works on Windows, Linux, and Mac OS X. Note the MaESP is programmed Micropython by default, that you do not need to do this before the experiments. This is just for Micropython updating.

To download the latest version of MicroPython firmware for the ESP32, go to the https://micropython.org/download/esp32/ and scroll all the way down to the ESP32 section.
You should see a similar web page (see figure below) with the latest link to download the ESP32 .bin file – for example:v1.19.1 (2022-06-18) .bin.

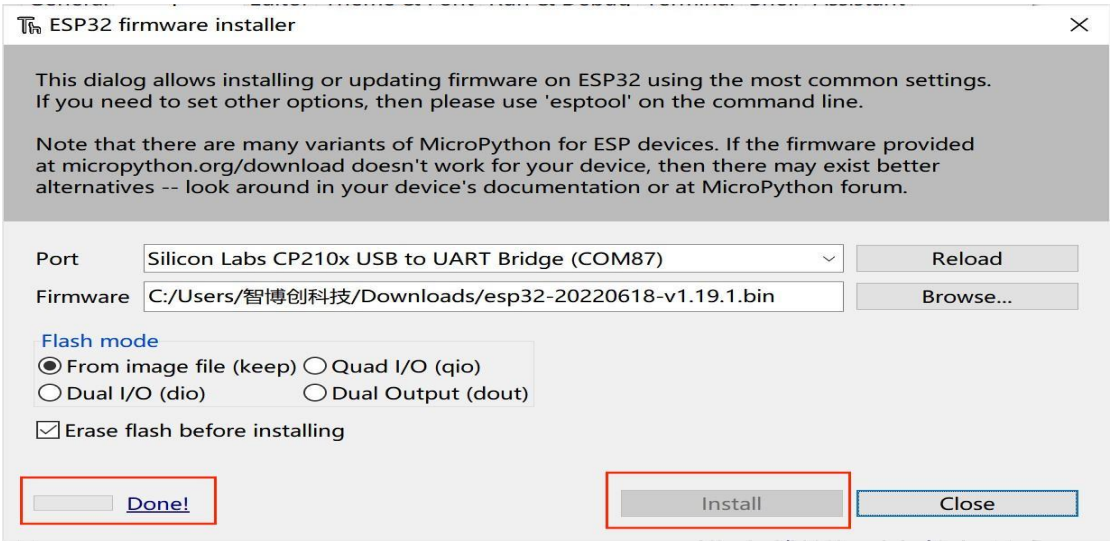


Micropython firmware burning.

Inter the interpreter, Select the correct configuration, and click the **Install or update Micropython,** the page for burning micropython firmware is displayed.



Select the correct serial port and Micropython firmware path. If the firmware burn well, it will displaying the “Done! “

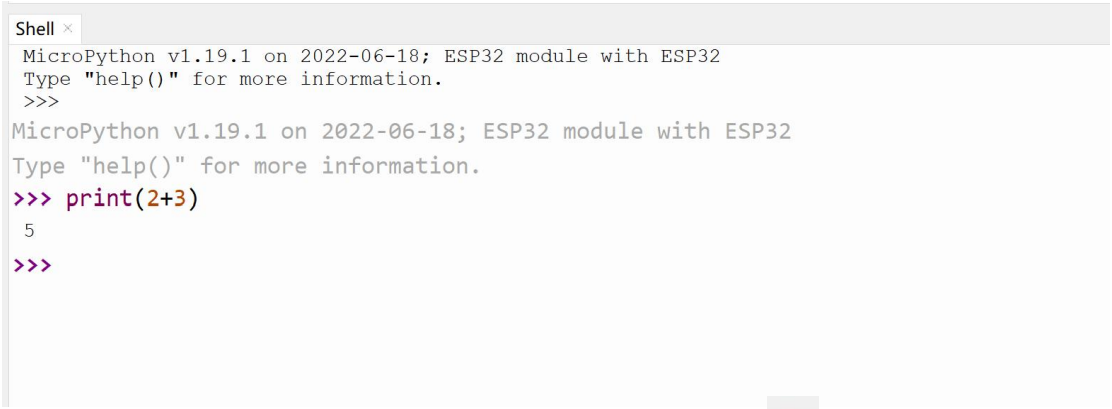


Verify the MicroPython firmware

You will see "**>>>** " appear in the Shell window after a successful connection with your board. You can type the print command to test if it's working. Try to print:

```
>>>print(2+3)
```

And the shell will print the result of (2+3)



Note: **If the shell don't displayed, users can click the stop button:** **to reset the board.**

About Makerfabs

Makerfabs is open hardware facilitator based on Shenzhen, China, We make open hardware projects and help our customers project prototyping& small batch production, includes PCBA/ Molding/ OEM. Contact service@makerfabs.com for more info.