

# Get Started MicroPython with ESP32

Version V2.2

Date 2021/04/22

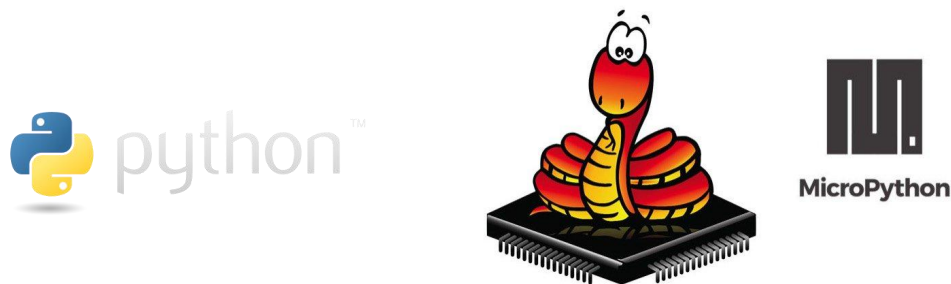
By Gray

## Introduction

Makerfabs MakePython ESP32 Kit intended to help beginners starting MicroPython with the ESP32. This Kits includes some common electronic components and modules, for the beginners to learn the basic knowledge of MicroPython, include hardware and programming.

There totally 12 basic lessons help starters to learn the usage and programming skills, and 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. Previously, Python is mainly for software programming on PC. 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 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 now is easily accessible to beginners and seasoned Python programmers alike.

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

## Product List



**MakePython  
ESP32 x1**



**Ultrasonic ranging  
module x1**



**Temperature and  
humidity sensor x1**



**Buzzer module x1**



**DS18B20 module x1**



**Infrared module x1**



**Potentiometer x1**



**WS2812 module x1**



**Sound sensor x1**



**Vibration sensor x1**



**Photosensitive  
resistance module x1**



**Button x2**



**Pulse sensor x1**



**Jump Wire x45**



**USB cable x1**



**Servo motor x1**



**Bread board x2**



**Resistance 330R x10**

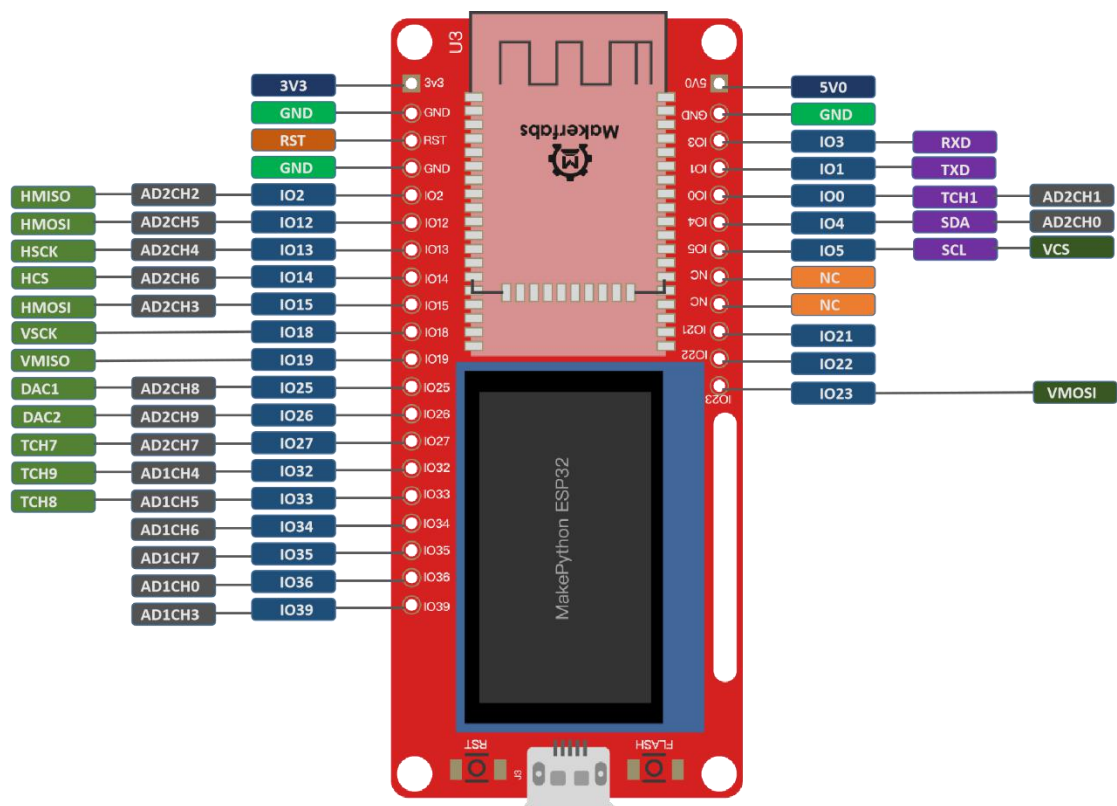


**Red LED x10**



**Green LED x10**

## MakePython ESP32 Pin figure



**MakePython ESP32** is designed by Makerfabs, Shenzhen, in 2019. It is based on ESP32 module and 128\*64 OLED, it features:

- Wi-Fi Protocols: 802.11b/g/n(802.11n up to 150Mbps),A-MPDU and A-MSDU aggregation and 0.4us guard interval support
- Wi-Fi Frequency range: 2.402GHz - 2.483Ghz
- CPU: Two low-power Xtensa® 32-bit LX6, ESP32-D0WDQ6
- CPU clock frequency: 80MHz to 240MHz
- expandable interface
- On-chip sensor: Hall sensor
- On-chip Internal RAM: 520KB
- Integrated crystal: 40MHz crystal
- Integrated SPI flash: 4MB
- Power supply: 5V, integrate 5V-to-3.3V LDO
- Operating current: Average 80 mA
- Minimum current delivered by power supply: 500 mA
- OLED: 1.3inch SSD1306/SSD1315
- Recommended operation temperature range: -40°C ~ +85°C
- Board size: 70\*32.6mm
- Moisture sensitivity level (MSL): Level 3

# CONTENT

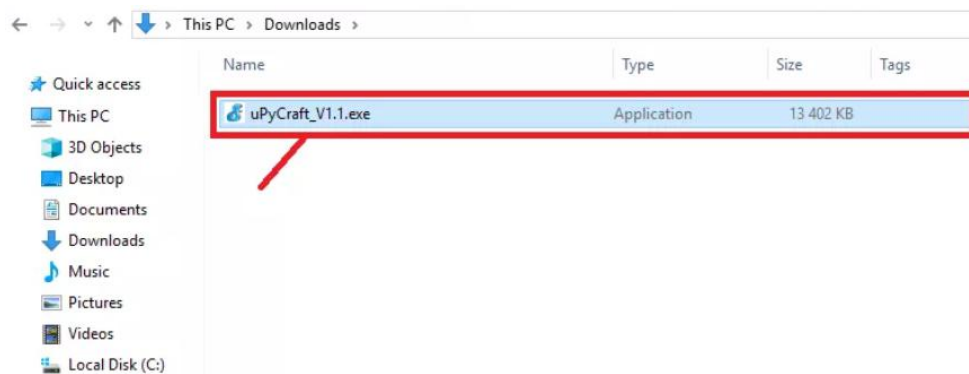
Introduction.....	1
What is MicroPython.....	1
Product List.....	2
MakePython ESP32 Pin figure.....	3
CONTENT.....	4
MicroPython Development Tool.....	5
Installing uPyCraft IDE.....	5
Get Start with uPyCraft IDE.....	6
Establishing a communication with the board.....	7
Creating new file on your board.....	9
Uploading first program: Blinking LED.....	10
MakePython Lessons.....	12
Lesson1: LED Control.....	12
Lesson2: Running LED.....	15
Lesson3: Button.....	17
Lesson4: PWM Control.....	19
Lesson5: Voice Control.....	23
Lesson6: OLED Display.....	26
Lesson7: Temperature Monitor DS18B20.....	28
Lesson8: Digital LED WS2812.....	31
Lesson9: ADC.....	33
Lesson10: Pulse Sensor.....	36
Lesson11: Ultrasonic Ranging.....	38
Lesson12: WiFi.....	41
A. WiFi connection.....	41
B. WebREPL.....	43
C. Socket communication.....	46
About us.....	50

## MicroPython Development Tool

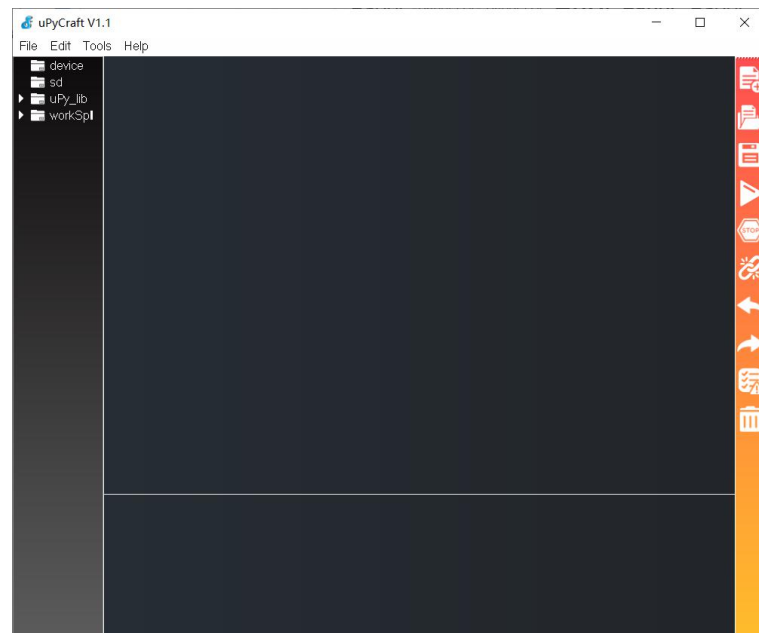
There many programming tools with MicroPython, such as Adafruit MicroPython tool (ampy), pyccham, most of them are identical. For this tutorial, uPyCraft IDE is recommended which is relatively easy for starters.

### Installing uPyCraft IDE

- Click the link to download uPyCraft IDE for Windows:  
<https://randomnerdtutorials.com/uPyCraftWindows>.
- After download the install package, you would see a similar file (uPyCraft\_VX.exe) in the Downloads folder

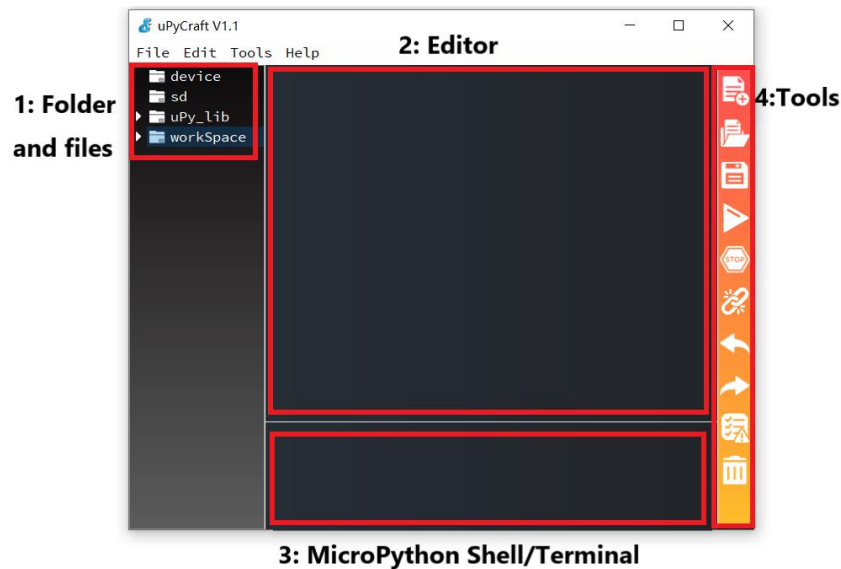


- Double-click the downloaded file to start the uPyCraft IDE:



## Get Start with uPyCraft IDE

Open uPyCraft IDE, take a look at each section of uPyCraft IDE:



- 1->Folder and files
- 2->Editor
- 3->MicroPython Shell/Terminal
- 4->Tools

### 1. Folder and files

This section lists several folders. Click the device folder and it will list all of the files which are currently stored on your MakePython ESP32 board. If connected MakePython ESP32 to uPyCraft IDE via serial, all files stored should load and shown at the folder. By default, the board has only one boot.py file when it has not downloaded any files yet. To be able to run the program normally, it is recommended to create a main.py file for programming.

**device** folder:

- **boot.py**: Runs firstly for setting up configuration options when the device starts up;
  - **main.py**: This is the main script that contains your code. It is executed immediately after the boot.py.
- A. **sd** folder is meant to access files stored on SD cards this is only works with boards like the PyBoard that come with an SD card slot.
- B. **uPy\_lib** lists the library files which built-in the IDE.
- C. **workspace** is one file directory that defines a path for storing the program files on the PC.

When using uPyCraft for the first time, it necessary to define your working directory. Click the workspace folder, a new window pops up, and you can choose the workspace path which to create a new folder or select an existing folder to be your working directory.

*\*Note: how to change the directory of **workspace**, go to **Tools >InitConfig** for initializing the config and click the workspace again to choose a different path.*

## 2. Editor

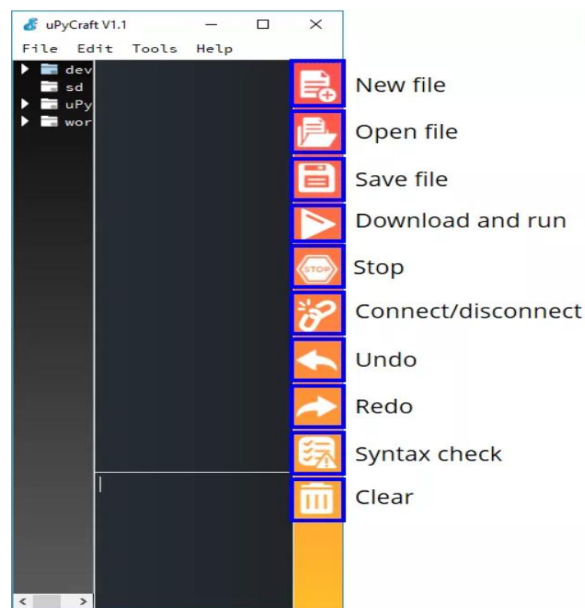
The Editor section is where you edit your codes. You can open more than one file, and the Editor will open a new tab for each file.

## 3. MicroPython Shell/terminal

On the MicroPython Shell, you can type some code to execute immediately by MakePython ESP32 without loading files. The terminal also prompts some information about the state of executing, and shows the errors related about upload, syntax errors, prints messages, etc...

## 4. Tools

The icons for quick tasks. Each button is labeled in the below figure:

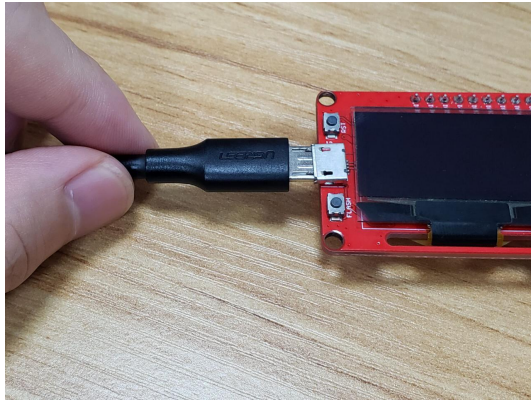


- **New file:** creates a new file on the Editor;
- **Open file:** open a file from your computer;
- **Save file:** saves a file;
- **Download and run:** upload the code to your board and execute the code;
- **Stop:** stop the execution of the code – it's the same as entering CTRL+C on the Shell to stop all scripts from running;
- **Connect/Disconnect:** connect or disconnect to your board via Serial. You must select the serial port first in Tools > Serial;
- **Undo:** undo last change in the code Editor;
- **Redo:** redo last change in the code Editor;
- **Syntax check:** checks the syntax of your code;
- **Clear:** clear the Shell/terminal window messages.

## Establishing a communication with the board

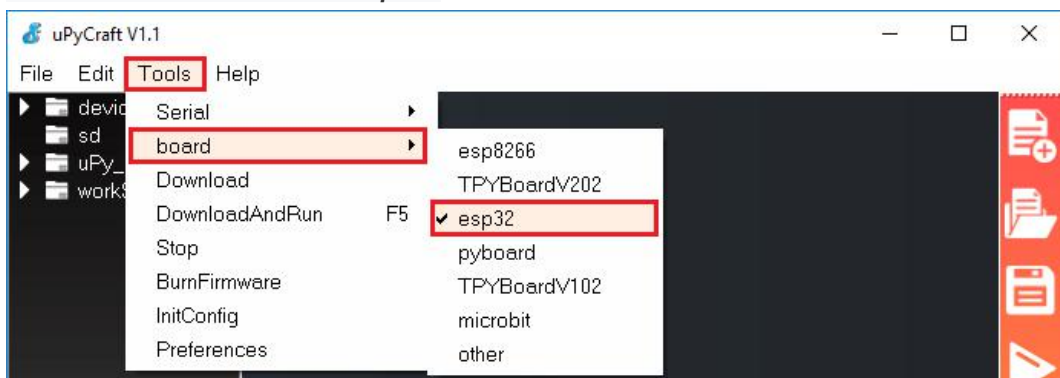
After unpacking the MakePython ESP32, connect it to the PC with a Micro USB cable.





Open uPyCraft IDE and follow the steps:

- A. Go to **Tools > Board** and select **esp32**.



- B. Go to **Tools > Serial** and select the port your MakePython ESP32 connected (the like for downloading the USB driver is:

<https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers>)

- C. Press the Connect button to establish a serial communication with your board.



Connect/disconnect

- D. You will see ">>>" appear in the Shell window after connecting successfully. Type the print command to check the connection.

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

Then the MakePython board calculates and feedback the result:

```
>>> print(2+3)
5
>>>
```

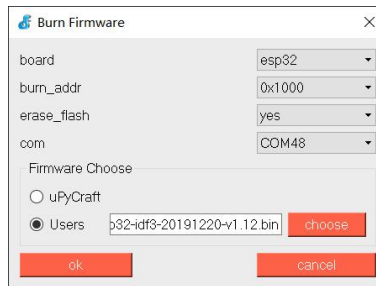
- E. Makerfabs MakePython is programmed with MicroPython before the shipping. But If your MakePython has been re-programmed for some other firmware, you have to re-load the Micropython firmware to the board until the connection is success.

Download the MicroPython firmware for MakePython ESP32 from here :


<https://micropython.org/resources/firmware/esp32-idf3-20191220-v1.12.bin>. Go to

**Tools>BurnFirmware**, select the parameter as the following picture. Then click OK and wait a minute for the firmware loading.



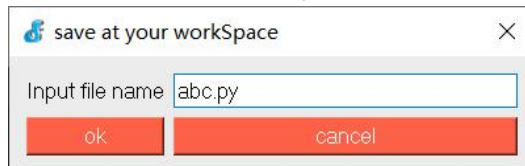


## Creating new file on your board

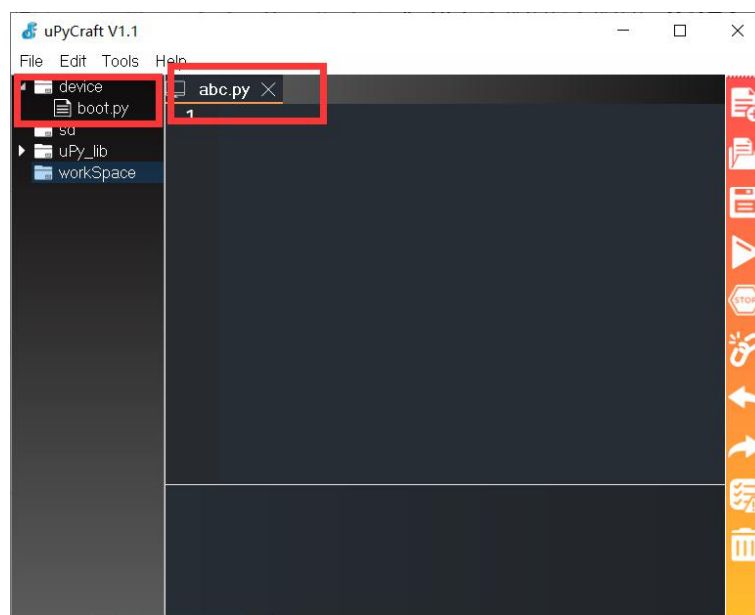
A. Press the “**New file**” button  to create a new file.


B. Press the “**Save file**” button  to save the file in your computer.

C. In the pop-up window, name your file whatever **not main.py** and save it:

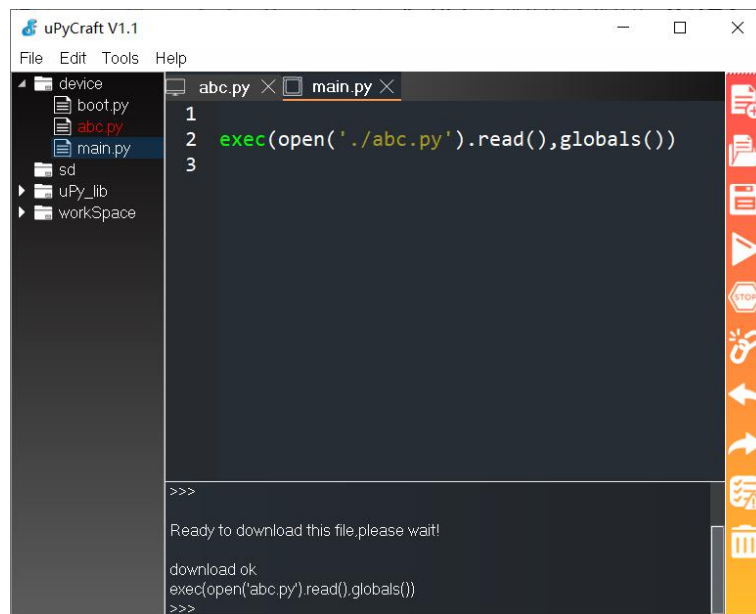


D. You would see as following in the uPyCraft IDE:



E. Click the “**DownloadAndRun**”  button to load the file to MakePython ESP32.

F. The abc.py (although it is empty now) has been stored on the board. The device directory will list the abc.py file. If you want the board to execute the file automatically after power again, right-click and set it to **Default Run**. The uPyCraft IDE would create **main.py** file to the board, which is the key for the file running automatically.





## Uploading first program: Blinking LED

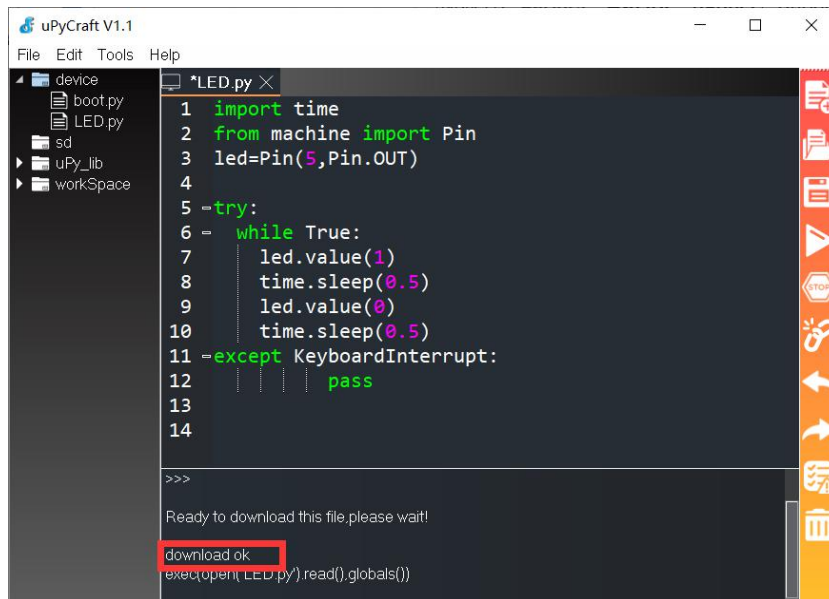
Let's try to upload the file coding for LED blink script into MakePython ESP32 board.

- A. Create a new file, Copy the following code to the file:

```
import time
from machine import Pin
led=Pin(5,Pin.OUT)

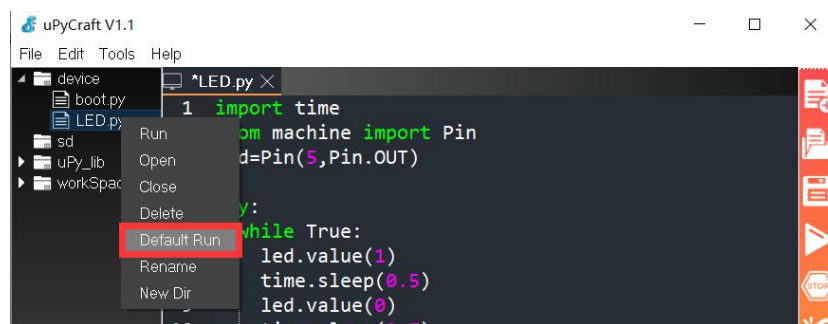
try:
    while True:
        led.value(1)
        time.sleep(0.5)
        led.value(0)
        time.sleep(0.5)
except KeyboardInterrupt:
    pass
```

- B. Click the **“Save”** button to save and name it as **“LED.py”**, then the file will be stored at the workspace directory.
- C. Press the **“Stop”** button  to stop any script running in the board.
- D. Click **“DownloadAndRun”**  to upload the file to MakePython ESP32.
- E. You will see **“download ok”** printing in the Shell window, it means the update succeed. The file would be run when it loaded successfully.

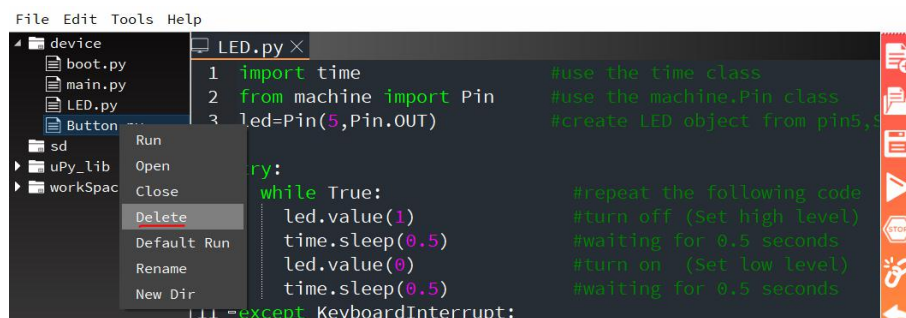


- F. To running the script again, you can right-click the file and select “Run” , the file will be executed again.

If you want to run the program on power, the file should be set to **Default Run**.



\* Note: It is suggested to delete un-necessary files when much files have been in the board. Right click the file and select the “delete” to delete.



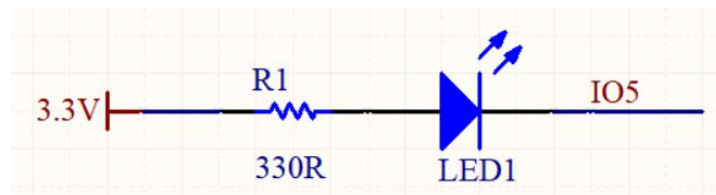
# MakePython Lessons

## Lesson1: LED Control

This is the basic lesson, in which we will learn how to control the pin of MakePython ESP32 to output with MicroPython.

### Material

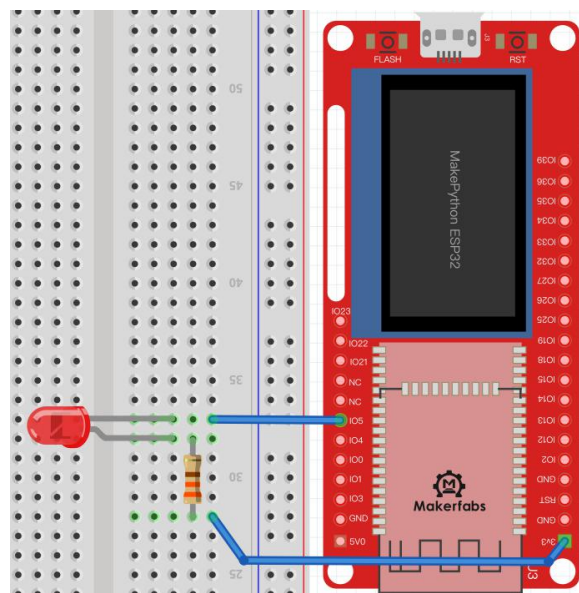
- A. 1\*LED;
- B. Resistor: 1\*330R;
- C. Jump wires;




### Instructions


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

### Wiring



### Create a file

Open the uPyCraft IDE, and click “**New**”  to create a new python file. After programming, click

“**Save**”  to save. During the saving you have to name it as “LED.py”.

### Programming

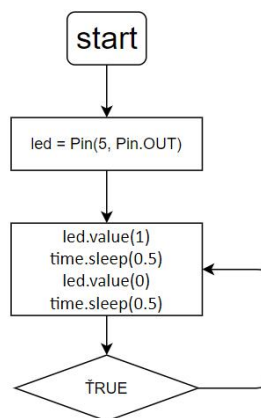
```

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.
- `from machine import Pin`  
import the Pin class from the machine library.
- `while True`  
Used to execute a program in loop under certain conditions.
- `led.value(1)`  
Set the led pin value to high level. There is no voltage difference between the two pins which both are connected to high level, that the LED is off.
- `led.value(0)`  
Set the pin value to low level.
- `time.sleep(0.5)`  
Delay 0.5 seconds, during the time the MCU will be sleep and do nothing.

*Note: Indentation refers to the spaces at the beginning of a code line. the indentation in Python is very important.*

*Python uses indentation to indicate a block of code.*

### Experimental result

Click "DownloadAndRun"  to download the Python file to MakePython ESP32 and run. You will see the LED blink with interval of 0.5 seconds.


The pin connection of the LED can be changed at your wishes. For example, if modify the code to **led=Pin(4,Pin.OUT)** and change the wiring of the LED negative to IO4, the light will flash. Similarly, if change the code to **time.sleep(1)**, the light will flash with interval of 1s.

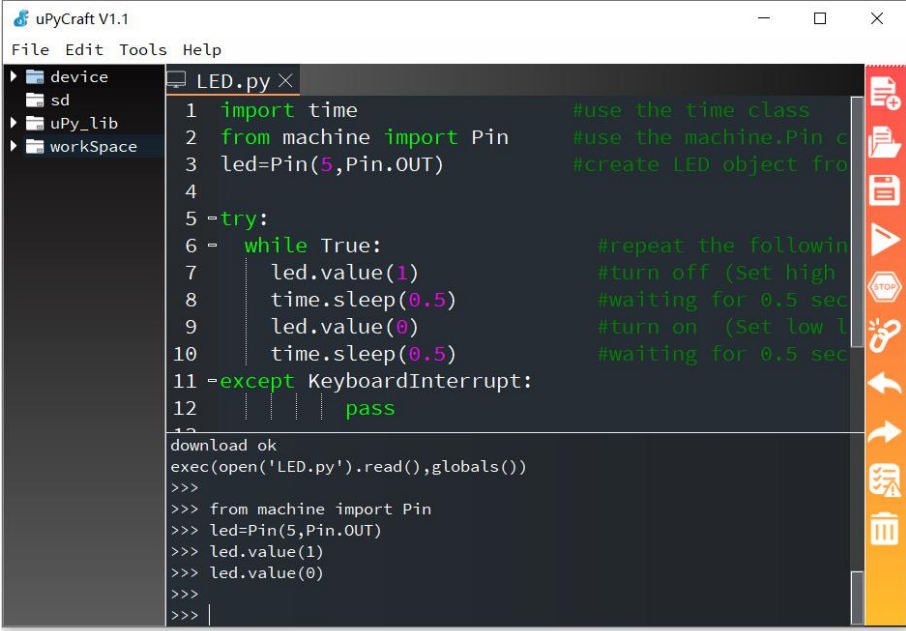
Set the LED.py to **Default Run**, and the program will run automatically when the MakePython ESP32 power on.

### Extension

With MicroPython, you can control the LED light on/off by sending command through serial port without loading the files into MakePython board.

Click "**Stop**" and input the following command into the shell.

*\*Note: If the 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:*



The screenshot shows the uPyCraft V1.1 interface. On the left is a file explorer with folders: device, sd, uPy\_lib, and workSpace. The main area displays a Python script in a file named LED.py. The script imports time and Pin from machine, creates an LED object on pin 5, and enters a while loop that toggles the LED state (high/low) with 0.5-second delays. It also includes a KeyboardInterrupt exception handler. Below the script, the MicroPython Shell shows the execution of the code line by line, with prompts like 'download ok', 'exec(open('LED.py').read(),globals())', and '>>>'.

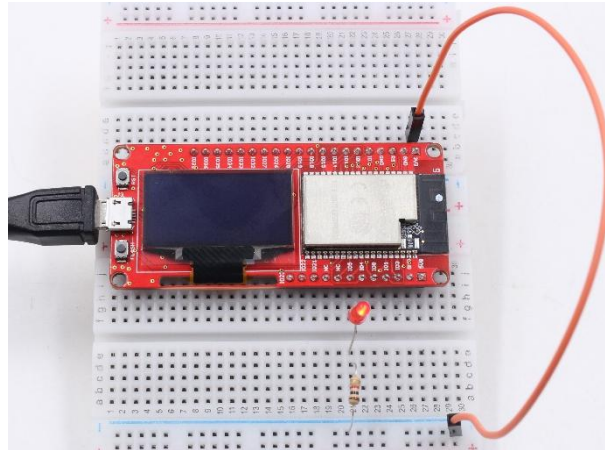
**MicroPython Shell/terminal**

### Command

```
>>>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 will be Off
>>>led.value(0)    #Press enter, led will be on
```

*\* Each time you finish writing the code, you need to press enter and move the cursor to the next line.*

With this method, you can directly control the MakePython ESP32 board hardware directly with commands.



---

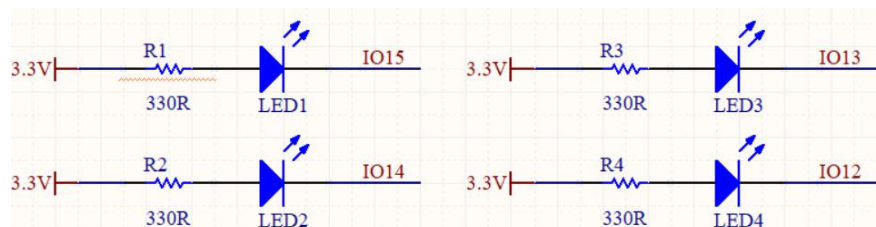
## Lesson2: Running LED

In this lesson, we will control multiple LED for understanding the port, compilation environment and program framework deeply.

### Material

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

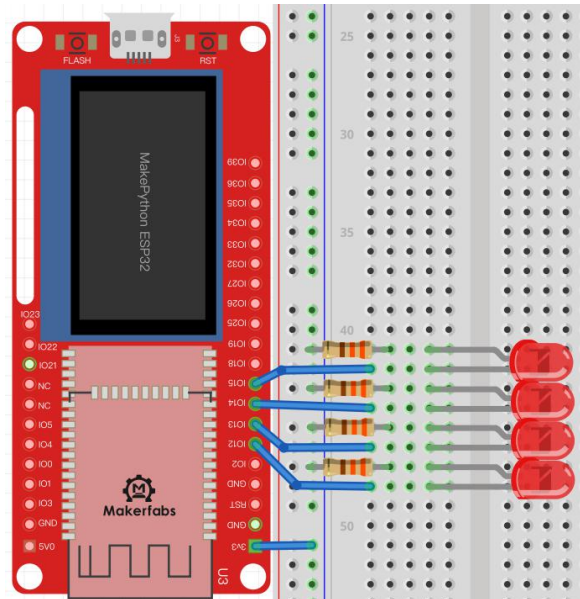
### Schematic



### Wiring

Connect the short LED pin to the MakePython ESP32 GPIO ports (IO15, IO14, IO13 and IO12). Connect the long pin to 3.3V. A 330R resistor must be connected in serial between the LED and 3.3V.





## Programming

Create a new file and add the following code. Then save it and name "Running\_LED.py".

```
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
```

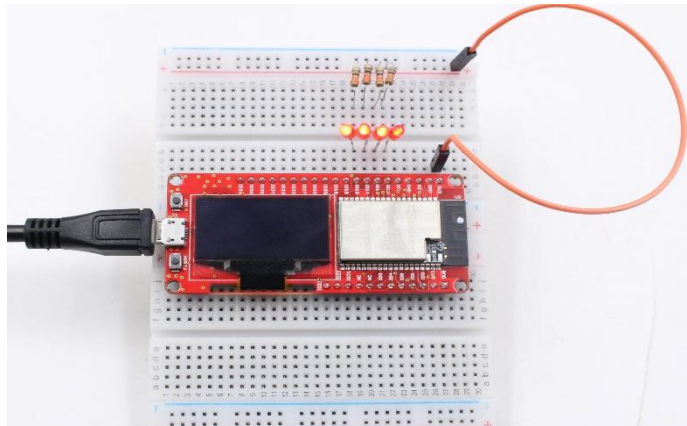
## Grammar

- `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.
- `n = (n+1)%4`  
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

## Experimental result

After loading and running the code, you will see four LED lights on and off in cycles. Set the

Running\_LED.py to **Default Run**, and the program will run automatically when the MakePython ESP32 power on.



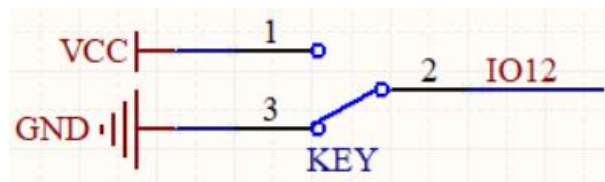
---

## Lesson3: Button

In the lesson, you will learn the usage of GPIO input, the MakePython board check the status of the button connected, and to control the LED light on/off.

### Material

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

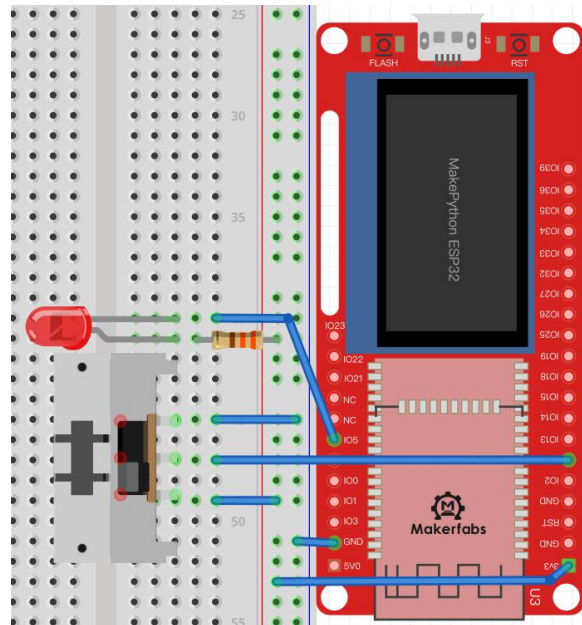


### Instructions

The button module has three pins—VCC, OUT and GND. When the button is pressed, OUT will be connected to VCC that OUT level is high; when the button is released, OUT will be connected to GND that OUT level is low.

### Wiring

Wiring LED is the same as the Lesson1. Connect the long lead of LED to 3V3 via a resistance, and connect the short to the Pin IO5. As the introduction, VCC of the button module is connected to the 3V3, OUT is connected to the Pin IO12, GND is connect to the GND of the MakePython ESP32. Your finished circuit should be like the blow figure.

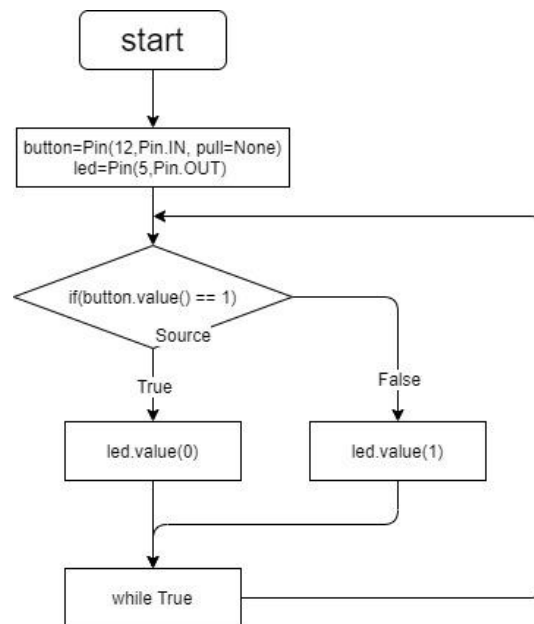


## Programming

Create a new file with the following code, then save and name it **"Button.py"**.

```
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
```

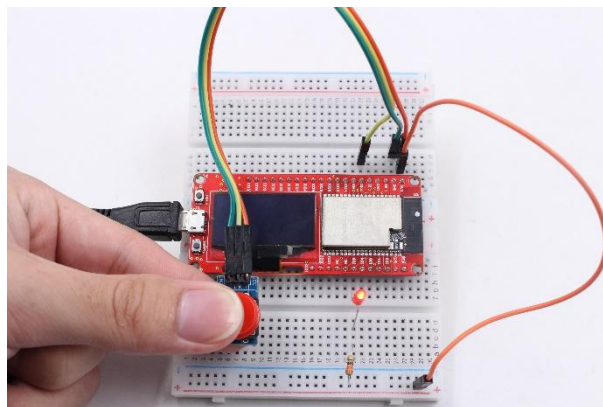
## Grammar explanation



- `button=Pin(12, Pin.IN)`  
Create a button object and set it to input mode.
- `if...else...`  
if statement: If true, execute the statement after the “if”. Otherwise, execute the statement after the “else”.

## Result

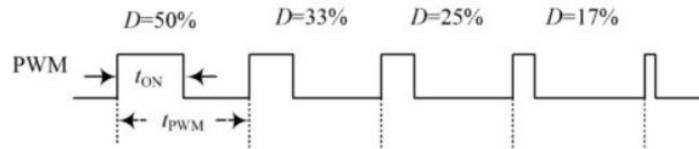
Loading the code, and pressing the button, the LED will be on with button pressed, and off with button released. Set the Button.py to **Default Run**, and the program will run automatically when the MakePython ESP32 power on.




---

## Lesson4: PWM Control

Pulse width modulation (PWM) output “square waves”, with different duties depends on the controller. The pin changes from high to low in one cycle. There are two important parameters: switching frequency and duty. Duty is defined as the percentage of high-level time in one cycle.



This lesson shows how to use the MakePython board to generate a PWM signal, and thus to control LED brightness and steering gear rotation Angle.

### Material

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

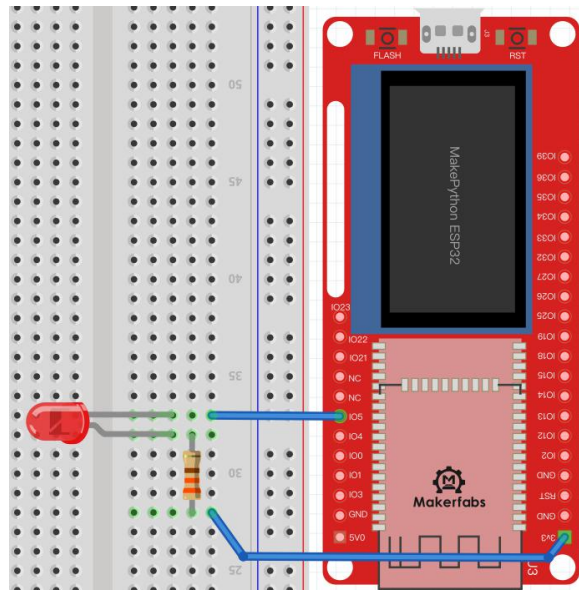
### A. Control the LED brightness

#### Instructions

By adjusting the duty cycle of the PWM, the LED brightness can be adjusted. For example, a PWM cycle is 100ms, which 33ms high and 67ms is low, and the duty cycle is  $33/100=33\%$ .

#### Wiring

Connect the long lead of LED to 3V3 via a resistance, and connect the short to the Pin IO5. As follows:



```

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(0,1024)`  
The range() function create a list and is used to iterate over a sequence of numbers in a loop.
- `for duty_cycle in range(0, 1024)`  
For statement iterates over the items of any sequence in the order that they appear in the sequence.
- `led.duty()`  
The larger the duty\_cycle value of the led, the brighter the led.

#### The experimental results:

Loading the code, the LED brightness changes, from full bright to dark by cycle.

## B. Control the servo

**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.

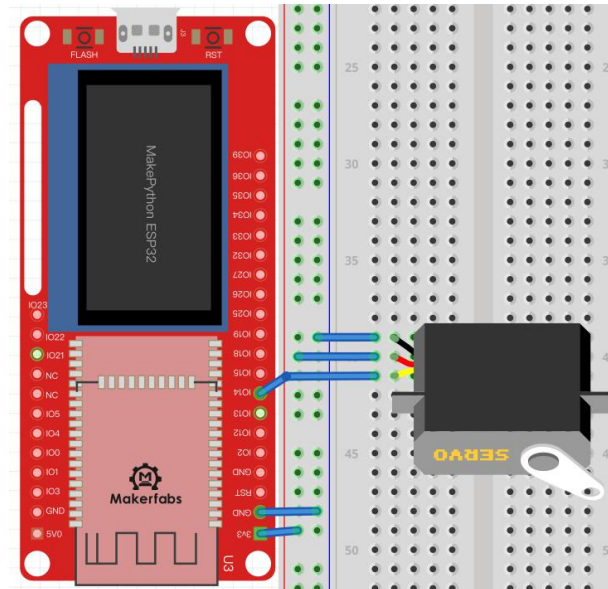


#### Instructions

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 servo motor has three wires: Orange wire is Signal, Red wire: Power, Brown wire: GND. The orange line is connected to IO14, the red line is connected to 3.3v, and the brown line is connected to GND.



## Programming

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
```

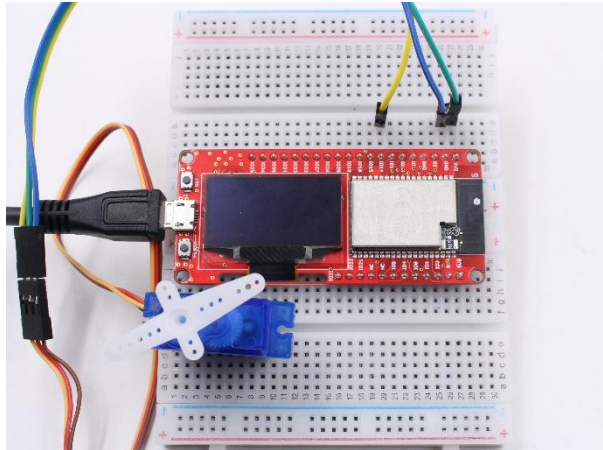
## Grammar explanation

- `servo = machine.PWM(p14,freq=50)`  
Create the PWM object and set the frequency to 50Hz.
- `servo.duty(i)`  
Adjust the duty cycle.

## The experimental results

The servo motor keeps turning.





### More important

You also can input the commands in the terminal to make the servo rotate to any position you want after the programming loaded, which is very convenient in robot application debugging:

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

```
Ready to download this file,please wait!
..
download ok
exec(open('servo-demo.py').read(),globals())
>>> servo.duty(120)
>>>
```

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

## Lesson5: Voice Control

This experiment introduces how to do some experiments about sound with the buzzer and sound sensor module.

### A. Experiment for buzzer module



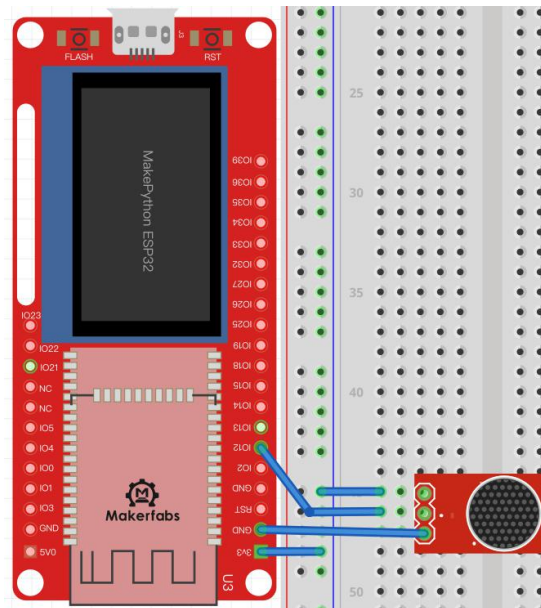
#### Instructions

According to the driving mode, the buzzer is mainly divided into active buzzer and passive buzzer. Active buzzer needs the DC voltage, and passive buzzer needs a specific frequency of vibration signal to drive. To do this experiment requires the active one.

#### Wiring

The buzzer module has three pins (VCC, I/O, GND). VCC is connected to the 3V3 of the MakePython

ESP32, GND is connected to GND, and I/O is connected to IO12:



### Programing

Create a **Buzzer.py** file with following code and comments:

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

### Grammar explanation

- *def ambulanceSiren(pwm)*  
Define a function, 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.

### Experimental result

You can hear the sound that likes an ambulance whistle, and you can also modify **pwm.freq()** to change sounds.

## B. Experiment for sound sensor

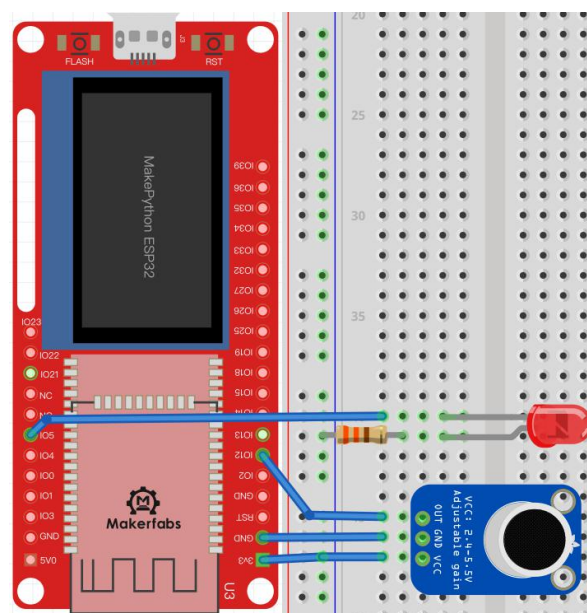


### Instructions

With the sound sensor in the kit, we can make a “sound switch”, that controls an LED On/Off by the sound. The sound intensity of the surrounding environment can be detected by sound sensor module. When the sound of the external environment fails to reach the threshold, the pin “OUT” outputs a high level; when the sound intensity of the environment reaches the threshold, “OUT” outputs a low level. The blue digital potentiometer on the module can be used to adjust the trigger threshold.

### Wiring

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



### Programming

Create a file named **Voice**, and add following code.

```
from machine import Pin
import time

voice=Pin(12,Pin.IN)
led=Pin(5,Pin.OUT)

try:
```

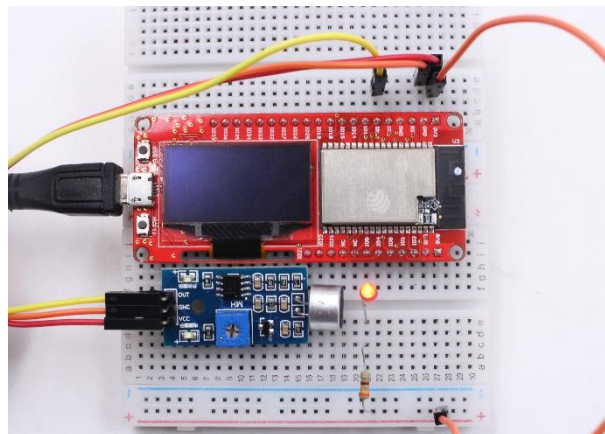
```

while True:
    if (voice.value() == 1):
        led.value(0)
        time.sleep(1)
    else :
        led.value(1)
except KeyboardInterrupt:
    pass

```

### Experimental result

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

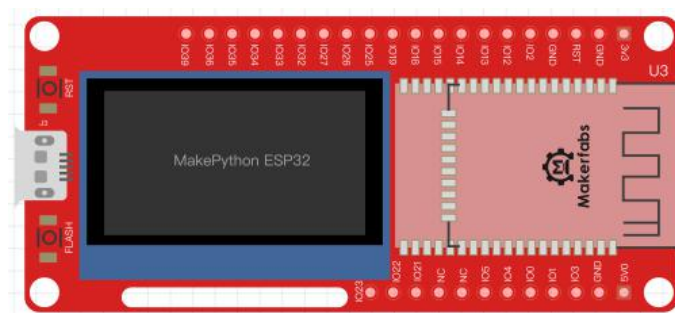


## Lesson6: OLED Display

There is an on-board OLED 1.3" OLED module integrated on the MakePython board, with 128x64 pixels. One pixel of a monochrome screen is a light-emitting diode. OLED screens have two versions that support I2C or SPI communication protocols, and the two versions are completely incompatible due to different protocols. In the lesson, the OLED is configured to be compatible with the I2C protocol.

### Material

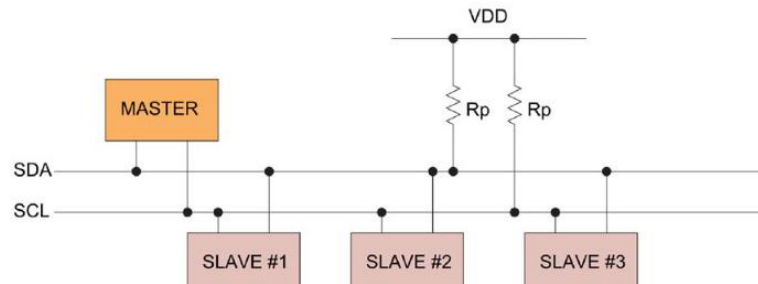
MakePython ESP32:



### I2C communication

I2C is widely used for controller communication 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 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.



### The SSD1306 library

The MakePython board cannot simply program the pin to display text on the OLED, and the OLED screen is driven by the SSD1306 driver chip, so it'll need the library to drive the SSD1306 for display. The library can be obtained from the following link, or searched in the Github.

<https://www.makerfabs.com/makepython-esp32-starter-kit.html>

### Load library function

After downloading the file, copy **ssd1306.py** to the **workSpace** file directory.

Open the **ssd1306.py** file by uPyCraft IDE and click "DownloadAndRun" button to download and run, and the library file can be loaded into the board. Create a new **ssd1306demo.py** file to program. As shown in figure:

```

1 from machine import Pin,I2C
2 import ssd1306
3
4 i2c = I2C(scl=Pin(5), sda=Pin(4), freq=100000) #
5 lcd=ssd1306.SSD1306_I2C(128,64,i2c) #
6
7 =try:
8     lcd.text("Makerfabs",30,10) #
9     lcd.text("time:",30,25) #
10    lcd.text("2019-12-5",30,40) #
11    lcd.show() #
12 -except KeyboardInterrupt:
13     pass
14
15
16
Ready to download this file,please wait!
.....
download ok
exec(open('ssd1306Demo.py').read(),globals())
>>>

```

### Programming

Create a new file named **ssd1306Demo.py** , and write the following code into the file.

```
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

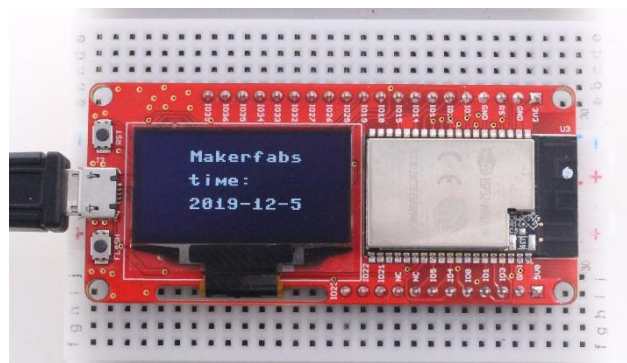
```

### Grammar explanation

- `I2C()`  
Initialize the I2C bus, configure the SCL and SDA pins, and set the frequency at 100000Hz.
- `ssd1306.SSD1306_I2C(128,64,i2c)`  
Use the function in the SSD1306 library to create a LCD object. Then the LCD can be driven by programming with the library functions.
- `lcd.text("Makerfabs",30,10)`  
"Makerfabs" is the text to display, The Numbers "30","10" are the X and Y axes of the screen.
- `lcd.show()`:  
Enable the display to show something.

### The experimental results

Save and click the "DownloadAndRun" button, and you'll see "Makerfabs/time:/2019-12-5". You can modify the text in `LCD.text()` statement to display.




---

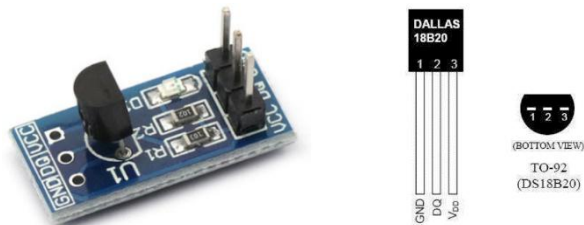
## Lesson7: Temperature Monitor DS18B20

DS18B20 is a low-cost digital temperature sensor. Its measurement range is between  $-55^{\circ}\text{C}$  and  $125^{\circ}\text{C}$ , with an inherent error of  $1^{\circ}\text{C}$  and an accuracy of  $0.5^{\circ}\text{C}$  between  $-10^{\circ}\text{C}$  and  $85^{\circ}\text{C}$ , which can meet the daily temperature measurement needs.

There are two power supply modes: independent power supply and data parasitic power supply. In the

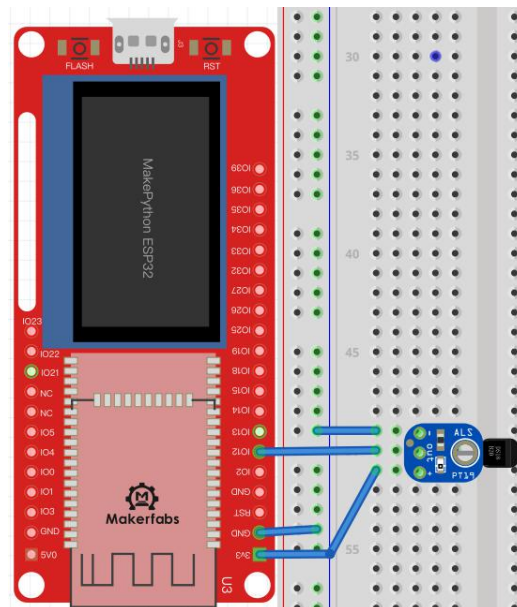


parasitic power supply mode, the power line is grounded, and the power is obtained from the data line, which can save a connection line. DS18B20 uses OneWire (single-wire) communication protocol, which only needs a signal line to complete the reading and writing of data. Here we show you how to read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



## Wiring

DS18B20 has three pins: GND; Data (need to connect the pull resistor a 10k resistor to 3.3v); VDD. The GND pin is connected to GND, VCC pin is connected to 3v3, and the DQ pin is connected to IO12.



## Programming

Create a new file named **DS18B20Demo** with the following code.

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

```
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(1))    # 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

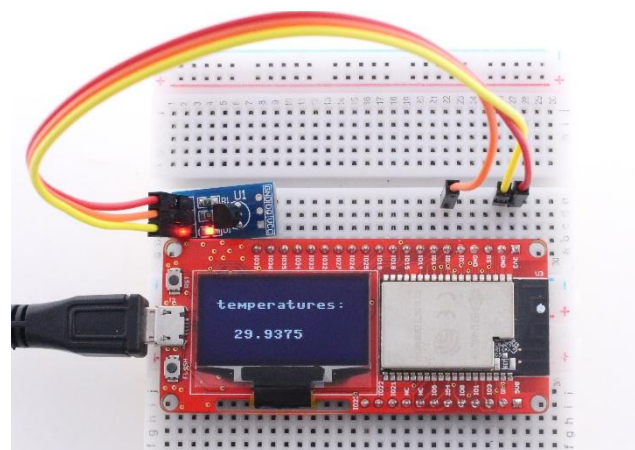
```

### Grammar explanation

- `scan()`  
Scan for devices on the bus.
- `convert_temp()`  
Start temperature reading. The `convert_temp()` method must be called each time you want to sample the temperature.
- `read_temp()`  
Read the collected temperature.
- `fill(n)`  
Empty the screen when `n=0`, and fill the screen when `n` is not 0.

### The experimental results

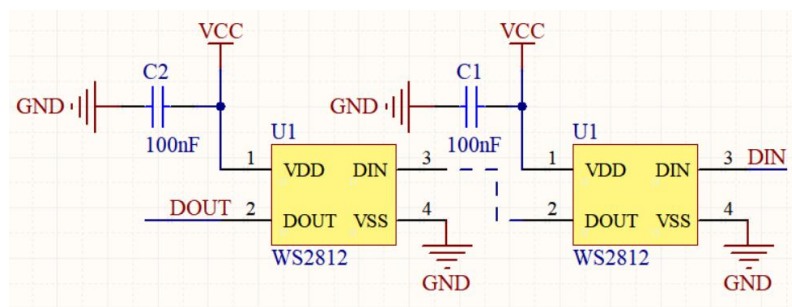
After running, the temperature collected appears on the display screen and refreshed every 750ms.



## Lesson8: Digital LED WS2812

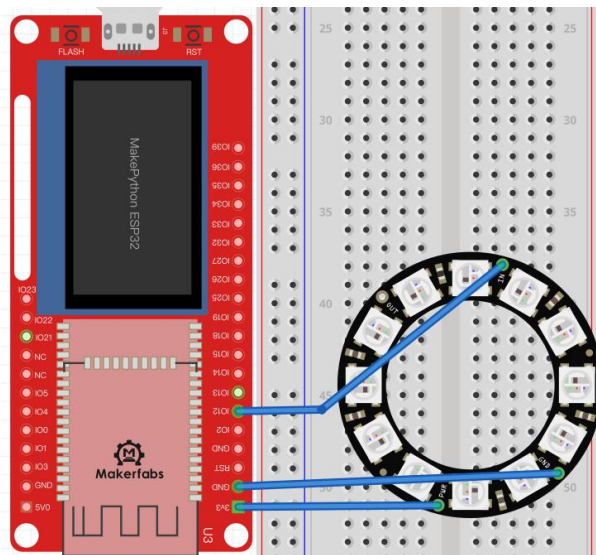
WS2812 is an intelligent control LED light source that the control circuit and RGB chip are integrated in a package of 5050 components. It is easy to control by one wire. WS2812 can be connected each one by one into a light strip or light ring. Besides, the color of WS2812 can be set at will which is composed with the RGB trichromatic lamp.

This experiment will show you how to turn the WS2812 light.



### Wiring

VDD is connected to 3.3V/5V, VSS is connected to GND, DIN is connected to IO12 of the board.



### Programming

Create a new file named **ws2812Demo.py** with following code.

```
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

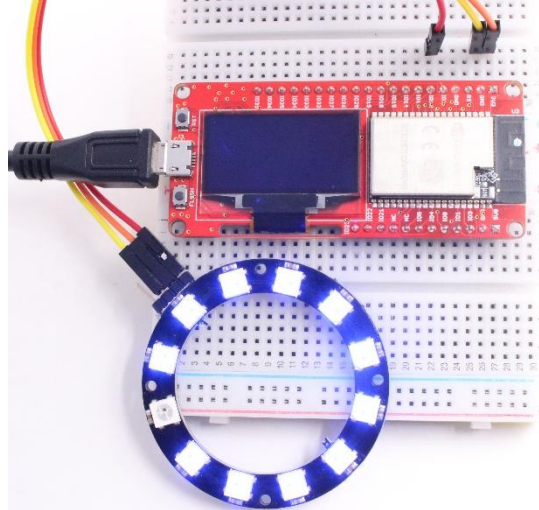
```

### Grammar explanation

- `np = neopixel.NeoPixel(machine.Pin(12), n)`  
Create the NeoPixe object, 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.

### Experimental result

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.



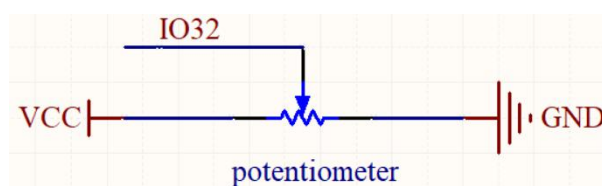
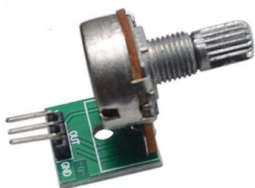
---

## Lesson9: ADC

ADC is an Analog/Digital Converter that converts Analog signals into Digital. In the previous lessons, you have learned how to use digital input or output on the MakePython ESP32. A digital input just has two binary state: logic LOW or logic HIGH. But the analog signal can be anything from completely off to completely on – a range of possible values. The analog signals are used for anything we met in everyday life, such as light intensity, sound waves, battery voltages and so on. Analog signal work through a piece of hardware named Analog/Digital converter. ADC take analog signal and change it to digital one.

MakePython ESP32 have on-board ADCs, it has 18 channels brought out to the GPIO pins. The channels usually used are GPIO32-GPIO39. When using the ADCs with default configuration, the voltages level of input pin must be between 0.0v and 1.0v (as long as above 1.0v, the value converted will be as 4095). Attenuation must be applied in order to increase the range of input voltage level. The attenuation can be set to 11DB in the code so that the range of input voltage level can be risen to 3.6V.

In this chapter, it will show how to accept the analog signal with the MakePython ESP32.

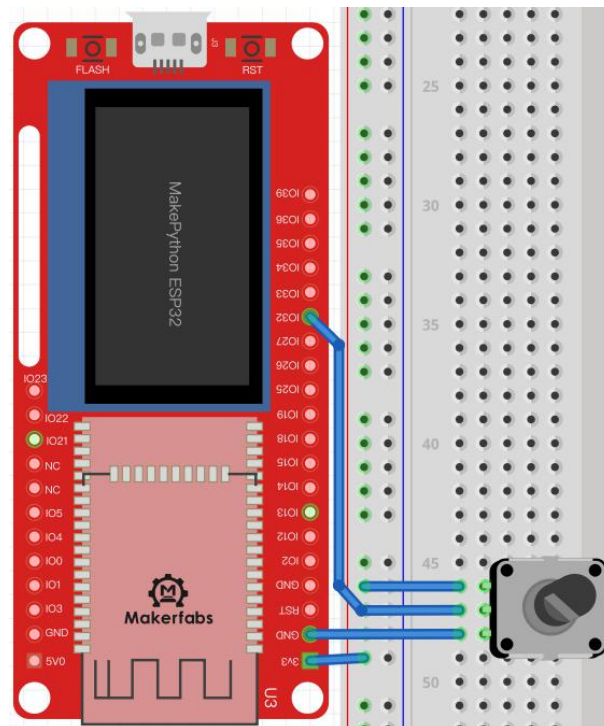


## Instructions

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 of potentiometer on the left and right sides are connected to 3.3V and GND respectively, the middle pin is connected to IO32.



## Programming

Create a new file with following code, save and name it **ssd1306\_adc.py**.

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

```
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
        adc.atten(ADC.ATTN_11DB)  # set 11dB input attenuation (voltage range roughly 0.0v - 3.6v)
        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

```

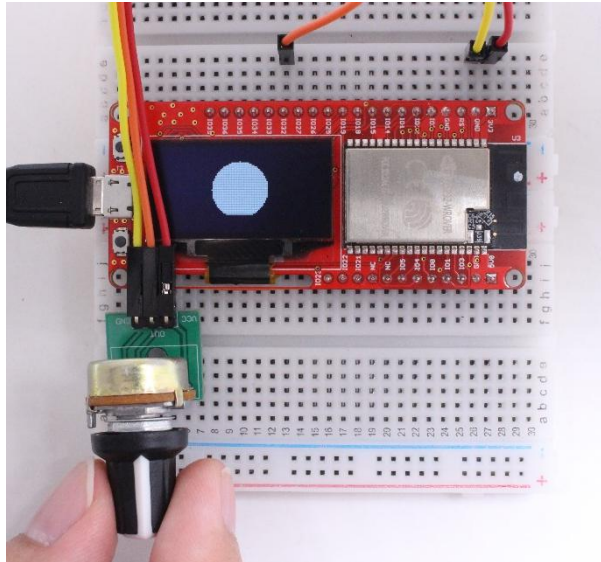
### Grammar explanation

- `adc.read()`  
Read ADC sampled data.
- `adc.atten()`  
Set the amount of attenuation on the analog input. Different attenuation setting result in different ranges of acceptance.
- `adc.width()`  
Set the bit number of the value which ADC used or return.
- `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 with the length w, starting at (x,y).
- `vline(x,y,w,c)`  
Draw a vertical line with height w, starting at (x,y).

### Experimental result

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




---

## Lesson10: Pulse Sensor

The Pulse Sensor is a photoelectric reflex analog sensor for measuring pulse or heart rate. Touch it with the finger, the sensor will output the relevant analog signal. The analog can be changed it by ADC to digital one. Then the heart rate value can be obtained after these digital data are calculated.

In this experiment, it will show you how to use the analog output sensor.



### Instructions

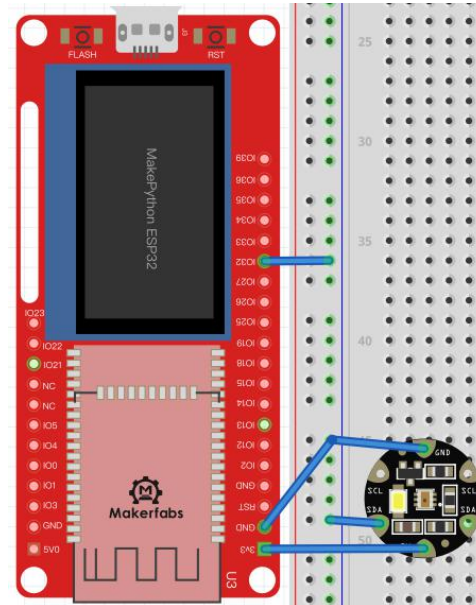
The pulse sensor has three pins, as the figure: the left pin is the s-pin of the signal output, the middle one



is the positive power supply VCC (3.3v ~5V), and the right one is the negative power supply GND.

## Wiring

The s-pin is connected to IO32, the VCC pin is connected to 3V3, and the "-" pin is connected to GND.



## Programming

Create a new file named **pulse.py** with following code.

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

```
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

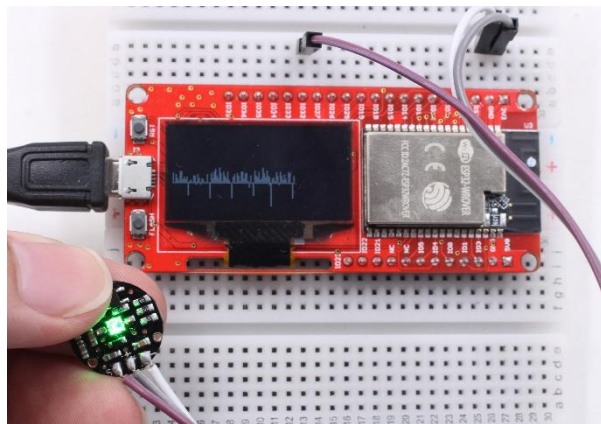
if(counttime>127):
    counttime=0
    oled.fill(0)           #Clear the screen
    oled.show()
except KeyboardInterrupt:
    pass
```

### Grammar explanation

- `counttime`  
Cache x coordinates (x less than 128)
- `line(x,y,x1,y1,c)`  
Draw a Line, the (x,y) is the starting point coordinate, (x1,y1) is the ending point coordinate.

### Experimental result

Touch the logo side with your finger, the screen will display your heart rate beat line. If contact instability or vibration, measurement will be affected.




---

## Lesson11: Ultrasonic Ranging

HC-SR04 is a non-contact distance measurement module, it provides a convenient and simple way to detect the distance with a high degree of precision. This ultrasonic sensor detects objects which from 2cm to 450cm. The module includes ultrasonic transmitter, receiver and control circuit.

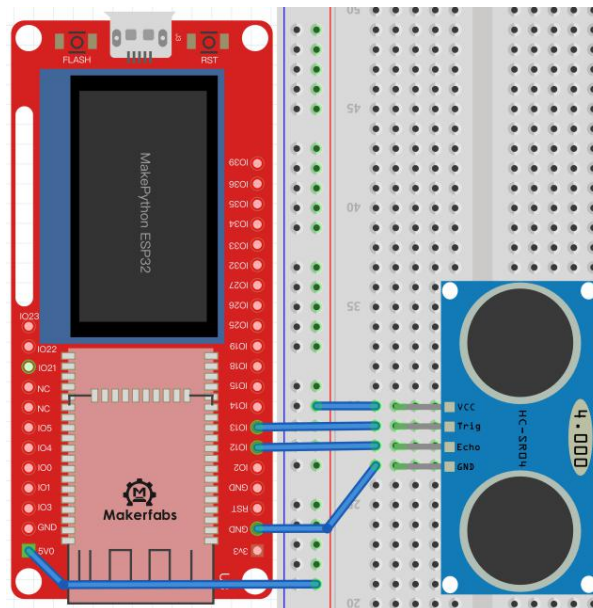


### Ranging principle

The ultrasonic sensor works on the same principles as a radar system. It can convert electrical energy into acoustic waves, the acoustic waves transmit in the air and come back to the receiver when hit the detected object. The running time of the acoustic wave is detected. And distance to the target object is calculated using the runtime of the acoustic waves.

### Wiring to ESP32

The Ultrasonic sensor has four pins. The pin connection of the sensor and MakePython ESP32 is as the below figure, "Trig" pin is connected to IO13, "Echo" pin is connected to IO21.



### Library function download link

<https://www.makerfabs.com/makepython-esp32-starter-kit.html>

Just like the LCD, the sensor also needs special library to supports MakePython ESP32 working. The library can be obtained from the zip of the project (*MakePython ESP32 lesson source code*).

Copy the **hcsr04.py** file to the **workSpace** directory. Open the file by the uPyCraft IDE, and load it to the MakePython ESP32.

## Programming

Create a new file fill with the following codes, save and name it **hc\_sr04.py**.

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

```
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)

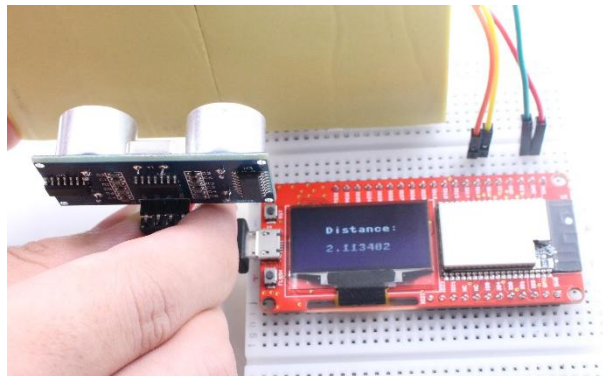
try:
    while True:
        distance = sensor.distance_cm()
        print(distance)
        lcd.fill(0)
        lcd.text("Distance:",30,20)
        lcd.text(str(distance),30,40)
        lcd.show()
except KeyboardInterrupt:
    pass
```

## Grammar explanation:

- `HCSR04()`  
Define a object and initialize the pin config of the ultrasonic sensor and timeout reminder.
- `sensor.distance_cm()`  
Returns the detected distance.

## Experimental result

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

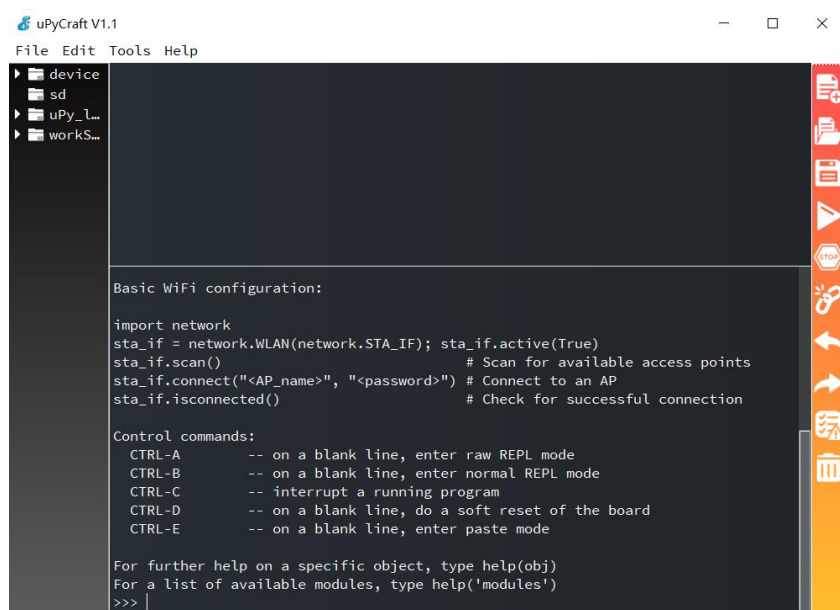


## Lesson12: WiFi

MakePython ESP32 devices have built-in Wifi& Bluetooth, which enables to be accessed via Wifi or home WiFi network connected. In this chapter, it will show how to access MakePython ESP32 with WebREPL(read-evaluate -print-loop) and remote control. Users can remote load the program to MakePython ESP32, without a USB cable connecting. Besides, this chapter also shows how to make the MakePython be a web-server with socket communication.

### A. WiFi connection

MakePython ESP32 can be a site to connect to WIFI when it works in **STA Mode**. Input **help()** in MicroPython Shell/terminal, press enter key, and the printed message will prompt you how to set **STA mode**.



```
uPyCraft V1.1
File Edit Tools Help

Basic WiFi configuration:

import network
sta_if = network.WLAN(network.STA_IF); sta_if.active(True)
sta_if.scan() # Scan for available access points
sta_if.connect("<AP_name>", "<password>") # Connect to an AP
sta_if.isconnected() # Check for successful connection

Control commands:
CTRL-A -- on a blank line, enter raw REPL mode
CTRL-B -- on a blank line, enter normal REPL mode
CTRL-C -- interrupt a running program
CTRL-D -- on a blank line, do a soft reset of the board
CTRL-E -- on a blank line, enter paste mode

For further help on a specific object, type help(obj)
For a list of available modules, type help('modules')
>>> |
```

### Command for Connecting WIFI (STA)

You can input the following command one by one in the MicroPython Shell/terminal to connect WIFI. Remember to modify the WiFi SSID and password.

```
>>> 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)`  
To enable the wlan interface.

- `wlan.scan()`  
Scan the WIFI around and display.
- `wlan.connect('ap_name','password')`  
Change 'ap\_name' and 'password' to your WIFI SSID and password for connecting
- `wlan.isconnected()`  
If the connection is successful, it will return True, otherwise return False. Use it to check the WIFI is connecting.
- `wlan.ifconfig()`  
Printed the information of the connected WIFI.

## Result

```

I (314018) wifi: state: init -> auth (b0)
I (314028) wifi: state: auth -> assoc (0)
I (314038) wifi: state: assoc -> run (10)
I (314048) wifi: connected with Makerfabs, channel 1, 40U, bssid = 20:6b:e7:a7:6a:fa
I (314048) wifi: pm start, type: 1

[0.32ml (314048) network: CONNECTED [0m
[0.32ml (314868) event: sta ip: 192.168.1.119, mask: 255.255.255.0, gw: 192.168.1.1 [0m
[0.32ml (314868) network: GOT_IP [0m

>>> wlan.isconnected()
__class__  __name__  gc      network
uos        bdev      webrepl wlan
>>> wlan.isconnected()
True
>>> wlan.ifconfig()
('192.168.1.119', '255.255.255.0', '192.168.1.1', '192.168.1.1')
>>>

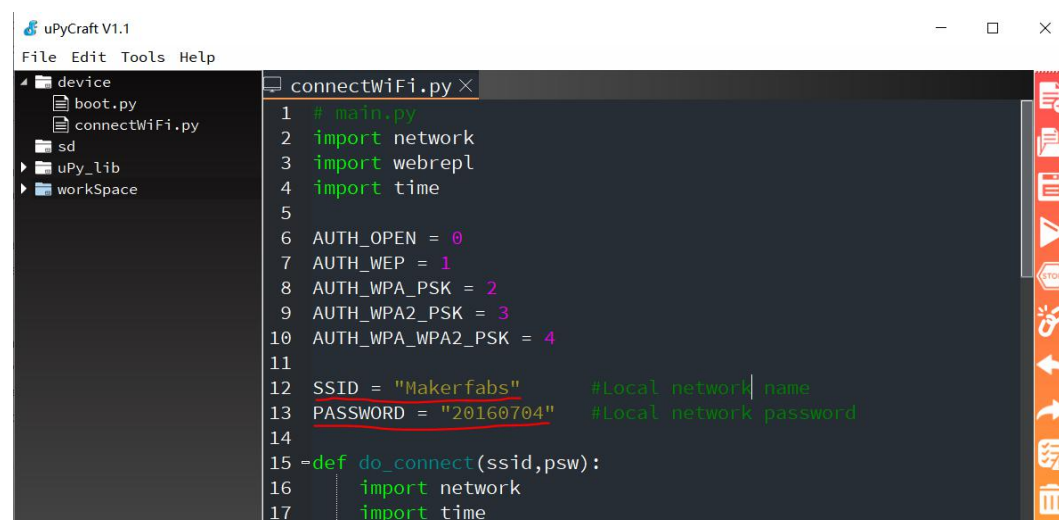
```

## Connect WIFI by programming

There is another way to set MakePython ESP32 connect WIFI, which is the most commonly used. Instead of directly inputting commands, download the program file and execute it, so that MakePython ESP32 can connect to WIFI. The program is available and can be obtain from here:

<https://www.makerfabs.com/makepython-esp32-starter-kit.html>

You can find the file **connectwifi.py** in the downloaded package, copy it to the **workSpace** directory. Open the file by uPyCraft IDE and change the WIFI **SSID** and **PASSWORD** as the follow picture.



```

uPyCraft V1.1
File Edit Tools Help

device
├── boot.py
├── connectWiFi.py
├── sd
├── uPy_lib
└── workSpace

connectWiFi.py
1 # main.py
2 import network
3 import webrepl
4 import time
5
6 AUTH_OPEN = 0
7 AUTH_WEP = 1
8 AUTH_WPA_PSK = 2
9 AUTH_WPA2_PSK = 3
10 AUTH_WPA_WPA2_PSK = 4
11
12 SSID = "Makerfabs" #Local network name
13 PASSWORD = "20160704" #Local network password
14
15 def do_connect(ssid,psw):
16     import network
17     import time

```

After saving and loading to the board, the shell will show **network config** information. For connecting WIFI next power on, press the **Stop** button, and set the connectWiFi.py to "Default Run". Restart MakePython ESP32, the shell will show **network config** information, WIFI connected!



```

uPyCraft V1.1
File Edit Tools Help
device
├── boot.py
├── connectWiFi.py
├── main.py
├── sd
├── uPy_lib
└── workspace

connectWiFi.py
1 import network
2 import webrepl
3 import time
4
5 AUTH_OPEN = 0
6 AUTH_WEP = 1
7 AUTH_WPA_PSK = 2
8 AUTH_WPA2_PSK = 3
9 AUTH_WPA_WPA2_PSK = 4
10
11 SSID = "Makerfabs" #network network_name
12 WPA_PSK = "123456789" #network network_name

I (1606) wifi: new <1,1>, bld<1,0>, ap<255,255>, sta<1,1>, prot.1
I (1606) wifi: state: init -> auth (b0)
I (1606) wifi: state: auth -> assoc (0)
I (1615) wifi: state: assoc -> run (10)
I (1625) wifi: connected with Makerfabs, channel 1, 40U, bssid = 20:6b:e7:a7:6a:fa
I (1625) wifi: pm start, type: 1

[0.32ml (162 ) network: CONNECTED [0m

[0.32ml (2425) event: sta ip: 192.168.1.129, mask: 255.255.255.0, gw: 192.168.1.1 [0m
[0.32ml (2425) network: GOT_IP [0m

network config: ('192.168.1.129', '255.255.255.0', '192.168.1.1', '192.168.1.1')
MicroPython V1.12.0 on 2019-12-20; ESP32 module with ESP32
Type "help()" for more information.
>>>

```

## B. WebREPL

WebREPL is a management platform which is developed by MicroPython to provide the wireless control. It means the WebREPL can do the same as the uPyCraft IDE such as loading file, it also has the same terminal to input commands. But WebREPL is not enabled by default, it has to input some commands to the board by uPyCraft IDE in order to enable.

Input the following commands in the shell of the uPyCraft IDE.

```
>>>import webrepl
>>>import webrepl_setup
```

The shell will show the following prompt, follow the prompt to operate. Enter: "E" (enable), then set the password to enable WebREPL, which the password is about 4-9 characters (in the example that I use "12345"), and input it again for confirming. The WebREPL is enabled successfully, follow the prompt to input "y" to restart the board.

```

>>> import webrepl
>>> import webrepl_setup
WebREPL daemon auto-start status: disabled

Would you like to (E)nable or (D)isable it running on boot?
(Empty line to quit)
> E
To enable WebREPL, you must set password for it
New password (4-9 chars): 12345
Confirm password: 12345
Changes will be activated after reboot
Would you like to reboot now? (y/n) y
>>>


```

The following figure shows that the restart is successful.

```

Started webrepl in normal mode
MicroPython v1.12-35-g10709846f on 2019-12-30; ESP32 module with ESP32
Type "help()" for more information.
>>>

```

*\*Note: after setting the password, the device will generate a **webrepl\_cfg.py** file automatically to save your password. If you didn't refresh it, you couldn't see it. You can try to reconnect the board, click  twice, and the file will appear.*



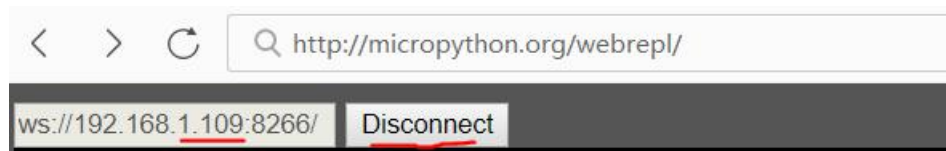
When the MakePython ESP32 restart, it has to follow the previous steps to connect WIFI again (if you have set connectWiFi.py to Default Run and MakePython ESP32 have connected to WIFI, please ignore). After connected WIFI, enter the following commands in the shell to start the WebREPL.

```
>>>import webrepl
>>>webrepl.start()
```

Then you will see the IP address: **ws:// 192.168.1.109:8266**

```
>>> import webrepl
>>> webrepl.start()
WebREPL daemon started on ws://192.168.1.109:8266
Started webrepl in normal mode
>>>
```

Make sure your PC connected the same WIFI with the MakePython ESP32. Open the following link with the browser to enter WebREPL: <http://micropython.org/webrepl/>. Or click the below link to download the local version of WebREPL : <https://codeload.github.com/micropython/webrepl/zip/master>. Input the IP address to connect the board.

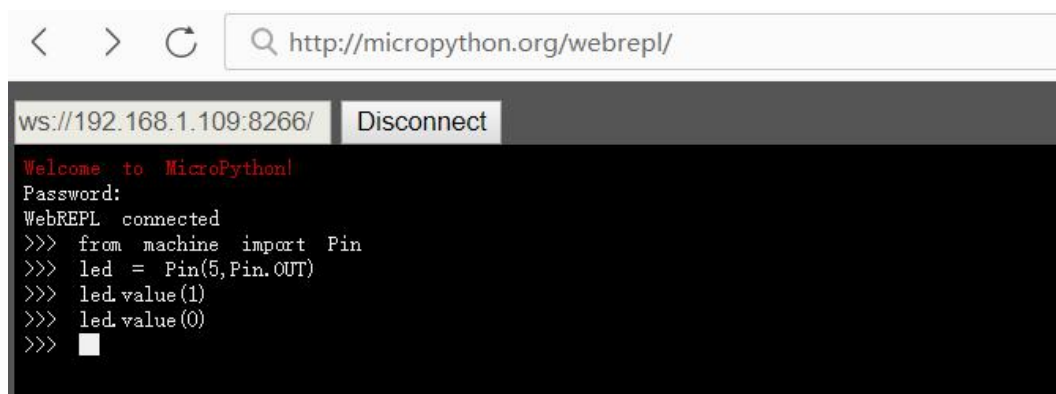


Enter the password you set and press “enter” key to connect to WebREPL.

There are two examples to show how to use and remote control MakePython ESP32 via WebREPL.

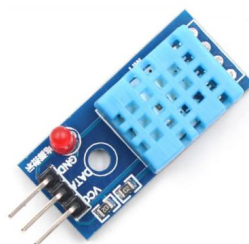
#### Example 1

Remember how to turn on an LED light? Now, wire the LED to MakePython ESP32 as the lesson 1, and type the commands via WebREPL to control the LED On/Off.



#### Example 2:

Use the DHT11 module to measure the temperature and humidity.



## Instructions

DHT11 digital temperature and humidity sensor contains calibrated digital signal output. It applies special digital module acquisition technology and temperature and humidity sensing technology.

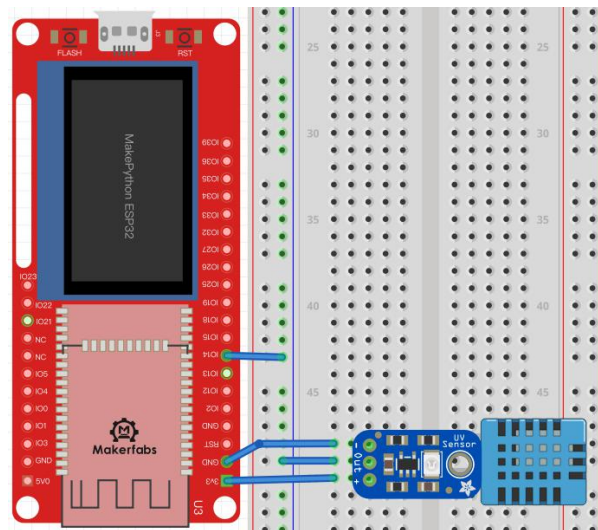
VCC: 3V~ 5.5v;

GND: Ground,

DATA: serial DATA pin;

## Wiring

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



## Programming

Create a new file, add the following code to the file. Save the file and name it **DHT11.py**.

```
from machine import Pin
import dht
import time

p14=Pin(14, Pin.IN)
d=dht.DHT11(p14)

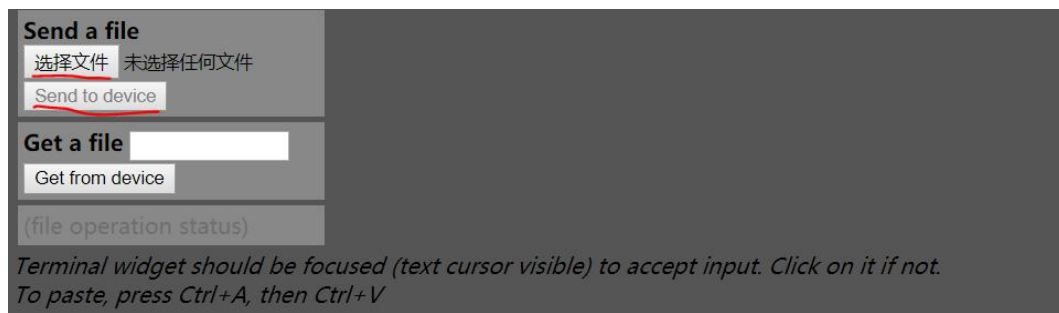
try:
    while True:
        d.measure()                #Measurement of temperature and humidity
        t=d.temperature()          #Read Celsius temperature
        h=d.humidity()             #Read relative humidity
        print('Temperature=', t, 'C', 'Humidity=', h, '%')
        time.sleep(1)              #Delay of 1 second
except KeyboardInterrupt:
    pass
```

### Grammar explanation

- `measure()`  
get the Measurement of temperature and humidity
- `temperature()`  
Read the Celsius temperature
- `humidity()`  
Read the relative humidity

### Load the file by WebREPL

The files can be loaded through WebREPL and show the measurement of temperature and humidity in WebREPL. In the WebREPL, click **select file** to choose the **DHT11.py**, and click **Send to device**:



After pressing the “send to device”, enter the below commands to print the files which has been stored in the MakePython.

```
>>>import os
>>>os.listdir()
```

Input the following command to execute dht11.py. the result of the sensor measuring will be shown.

```
>>>exec(open('DHT11.py').read(),globals())
```

```
>>> exec(open('DHT11.py').read(),globals())
Temperature= 27 C Humidity= 70 %
Temperature= 27 C Humidity= 69 %
Temperature= 27 C Humidity= 71 %
Temperature= 27 C Humidity= 71 %
Temperature= 27 C Humidity= 71 %
>>> █
```

Click “Ctrl+C” key to stop the script from running.

### C. 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 chapter will use MakePython ESP32 as the server and the browser on PC as the client side. Both sides will use TCP protocol to transmit and receive data from each other.

### Example\_1: LED Remote control by socket communication

In this example, the MakePython ESP32 with LED will be as a server, it can be controlled remotely with any PC in the same WIFI:

## Wiring

The LED wiring is the same as Lesson1.

## Programming

Use the uPyCraft IDE to loading the program to MakePython ESP32, the program can be obtained from here. Before loading it, it is necessary to modify the WIFI SSID and password in the code to yours.

>>>>>>As the source code long, please download the source code at:


<https://www.makerfabs.com/makepython-esp32-starter-kit.html>, to check the **Socket\_LED.py**<<<<<<<

## 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

## Experimental result

Press “DownloadAndRun” to load the code, then press “Stop” and set the file to “Default Run”. Press the RST button to restart the MakePython ESP32, the network config will print out of the shell:



```
Local MAC:A4:CF:12:42:6D:6C
connecting to network...Makerfabs
I (795) wifi: new<1,1>, old<1,0>, ap:<255,255>, sta:<1,1>, prof:1

I (1635) wifi: state: init -> auth (b0)
I (1645) wifi: state: auth -> assoc (0)
I (1655) wifi: state: assoc -> run (10)
I (1665) wifi: connected with Makerfabs, channel 1, 40U, bssid = 20:6b:e7:a7:6a:fa
I (1665) wifi: pm start, type: 1

[0.32ml (1665) network: CONNECTED [0m

[0.32ml (2375) event: sta ip: 192.168.1.129, mask: 255.255.255.0, gw: 192.168.1.1 [0m
[0.32ml (2375) network: GOT_IP [0m

network config: ('192.168.1.129', '255.255.255.0', '192.168.1.1', '192.168.1.1')
```

Open the IP address with the browser which on the PC or phone. The browser will show the page as follow, click the “ON” to control the LED on, click “OFF” to turn the LED off.



### Example\_2: Remote monitoring by socket communication

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

#### Programming

>>>>>>As the source code long, please download the source code at:

<https://www.makerfabs.com/makepython-esp32-starter-kit.html>, to check the *Socket\_DHT11.py*<<<<<<<

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

#### Experimental result

Modify **SSID** and **PASSWORD** in the code and load it. Press the “Stop” button and set the file to Default Run. Reset the MakePython ESP32, the information of network config will be printed out of the shell.

```
1 #main.py
2 import network
3 import webrepl
4 import dht
5 import time
6 from machine import Pin
7
8 =try:
9     import usocket as socket
10 -except:
11     import socket
12
13 local MAC: STA: 0x12:42:40:00
connecting to network. Makerfabs
I (803) wifi: new.<1.1>, old.<1.0>, ap:<255,255>, sta:<1.1>, prof:1
I (1853) wifi: state: init -> auth (b0)
I (1853) wifi: state: auth -> assoc (0)
I (1863) wifi: state: assoc -> run (10)
I (1883) wifi: connected with Makerfabs, channel 1, 40U, bssid = 20:6b:e7:a7:6a:fa
I (1883) wifi: pm start, type: 1
[0.32ml (1693) network: CONNECTED [0m]
[0.32ml (2373) event: sta ip: 192.168.1.129, mask: 255.255.255.0, gw: 192.168.1.1 [0m]
[0.32ml (2373) network: GOT_IP [0m]
network config: ('192.168.1.129', '255.255.255.0', '192.168.1.1', '192.168.1.1')
```

Input the IP (such as 192.168.1.129) to the browser on the PC or phone which connected the same WiFi, the page will show the temperature and humidity measurement.  
To get the latest measurement, please refresh the page.



*\*note: some browsers are not supported, please try another.*

## About us

### Description

This MakePython ESP32 Kits includes the basic components and guidance for MicroPython starters. With the 12 experiments in this guide, you will be able to make simple electronic projects with MicroPython and ESP32, and will get the basic knowledge& usage of IoT projects.

### About Makerfabs

Makerfabs is an open hardware facilitator located in Shenzhen, China. We make open hardware projects and provide the help of customers project prototyping& small batch production, includes PCBA/ Molding/ OEM. Contact [service@makerfabs.com](mailto:service@makerfabs.com) for more info.

### Techsupport

For any troubles, please contact the [support@makerfabs.com](mailto:support@makerfabs.com).