Kick — Start to MicroPython

(ESP32 / ESP8266)

Code bundle
Video links

Harish Kondoor

Kick – Start

to

MicroPython

ESP32 / ESP8266

Harish Kondoor

Disclaimer

This eBook has been written for information purposes only. Main aim for the eBook is to educate MicroPython. Content included is at the best and tries to minimise errors and be as accurate as possible.

The author (Mr.Harish Kondoor) shall not be responsible for any errors or omissions.

The author (Mr.Harish Kondoor) is not liable for any damage / loss caused or alleged directly or indirectly by this eBook.

The external links (video/ resource/ website url) provided throughout the eBook can be removed or content can be changed / updated at any time from third-party websites. The author shall not be responsible for that.

The Windows 10 operating system is used for course explanation along with ESP32 Dev as a development board. Other operating systems like Mac and Linux will support it if Thonny IDE and Python 3 is installed to the computer. 80 % of the course also supports ESP8266 development board. Please check the above compatibility before pursuing this course. Author shall not be responsible for any technical issues with your computer and its configuration.

This eBook has been created with the assumption that readers have basic knowledge in electronics.

About the Author

Mr.Harish Kondoor has more than 6 years of experience in the field of STEM education. He is primarily focused on robotics, 3D printing, and IoT. He is the founder of Umydo Solutions based in Bangalore, India. He helped different institutions to set up the STEM labs. The main hobbies are to create DIY projects and learn new technologies.

Table of contents

Module1- Introduction to course

Module2- Setting up software for the course

Module3- Flashing MicroPython firmware to ESP32

Module4- Python 3 syntax, recap using Micropython

Module6- Connect to internet over WiFi

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.

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

Module2- Setting up software for the course

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

- **Python 3**
- **Thonny IDE**

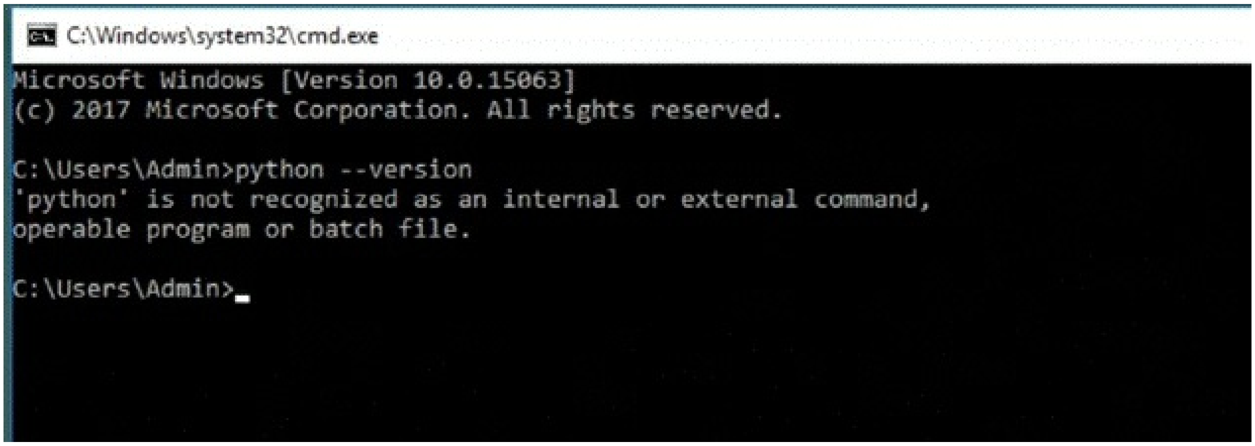
Note: Mentioned software will support in Windows ,Linux and Mac operating systems. This course mainly discusses using the Windows10 operating system.

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** ’



```
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>_
```

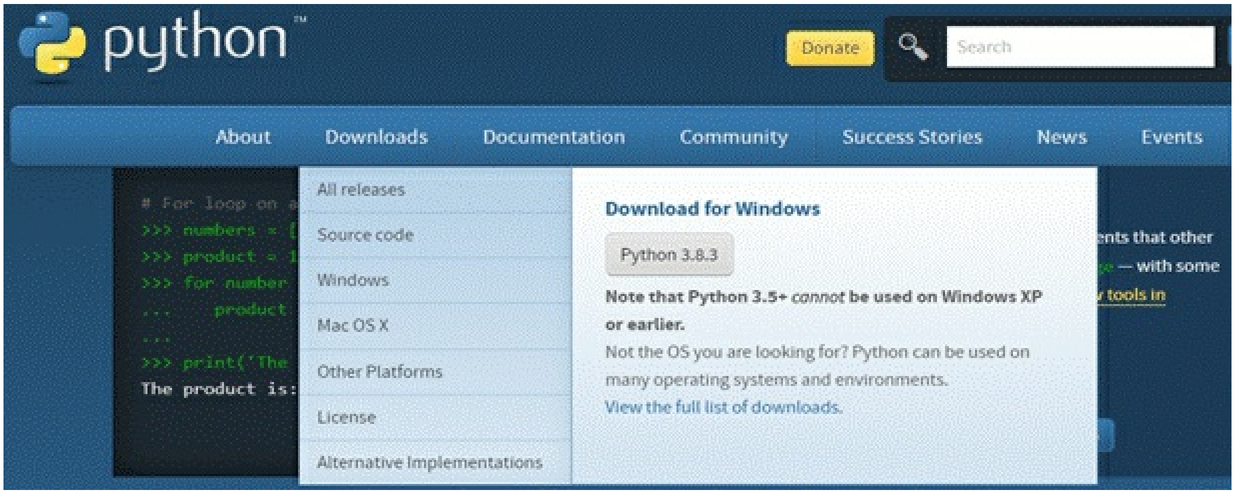
If **‘python’** is not recognized. Then Python have to install to the PC by following steps

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.



```
# For loop on a
>>> numbers = [
>>> product = 1
>>> for number
...     product
...
>>> print('The
The product is:
```

Note that Python 3.5+ *cannot* **be used on Windows XP or earlier.**

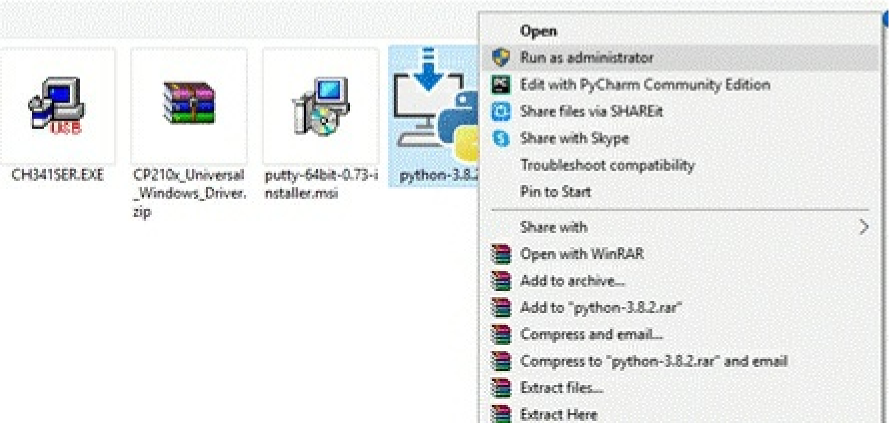
Not the OS you are looking for? Python can be used on many operating systems and environments.

Step4:

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

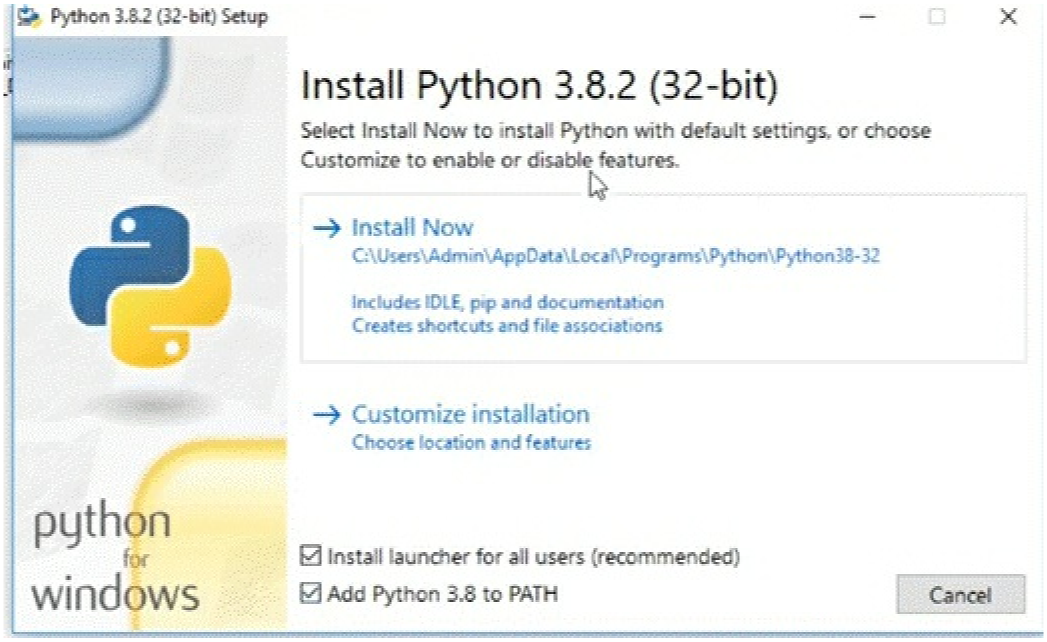
Step5:

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


CH341SER.EXE
CP210x_Universal_Windows_Driver.zip
putty-64bit-0.73-installer.msi
python-3.8.2
Open
Run as administrator
Edit with PyCharm Community Edition
Share files via SHAREit
Share with Skype
Troubleshoot compatibility
Pin to Start
Share with
Open with WinRAR
Add to archive...
Add to "python-3.8.2.rar"
Compress and email...
Compress to "python-3.8.2.rar" and email
Extract files...
Extract Here

Step6:

- Tick on **'Add Python 3.8 to PATH'** then click on **'Install Now'**



Python 3.8.2 (32-bit) Setup

Install Python 3.8.2 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

→ Install Now
C:\Users\Admin\AppData\Local\Programs\Python\Python38-32

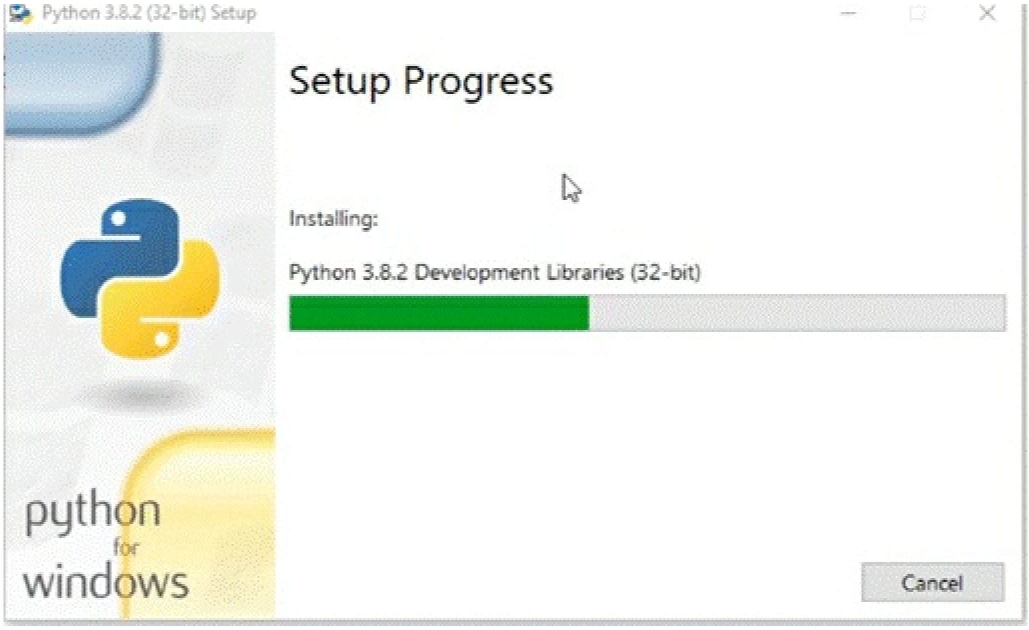
Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.8 to PATH

Step7:

- Wait for a few minutes until installation is completed.



Python 3.8.2 (32-bit) Setup

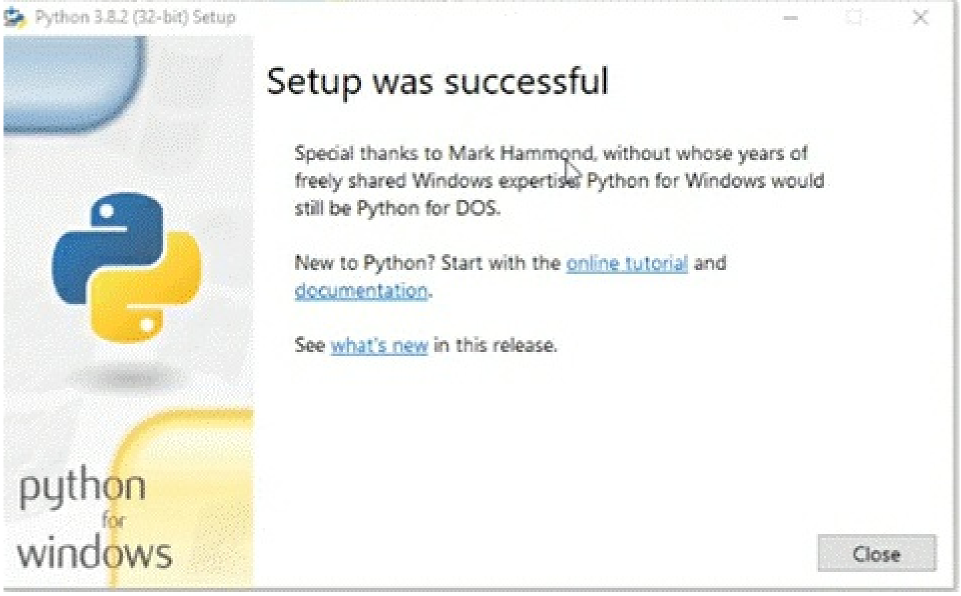
Setup Progress

Installing:

Python 3.8.2 Development Libraries (32-bit)

Step8:

- ➢ Once installation is completed click on **‘Close’** Tab



Setup was successful

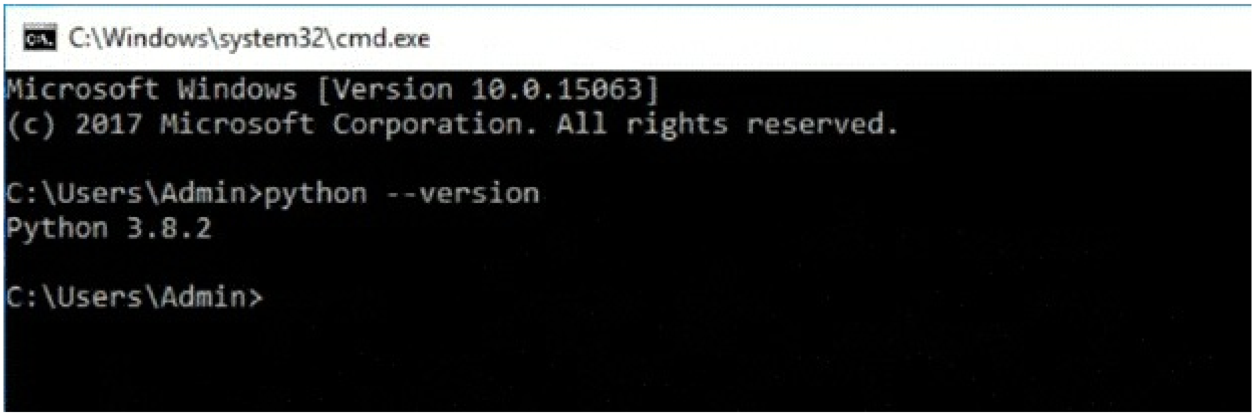
Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

New to Python? Start with the online tutorial and documentation.

See what's new in this release.

Step9:

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



```
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.

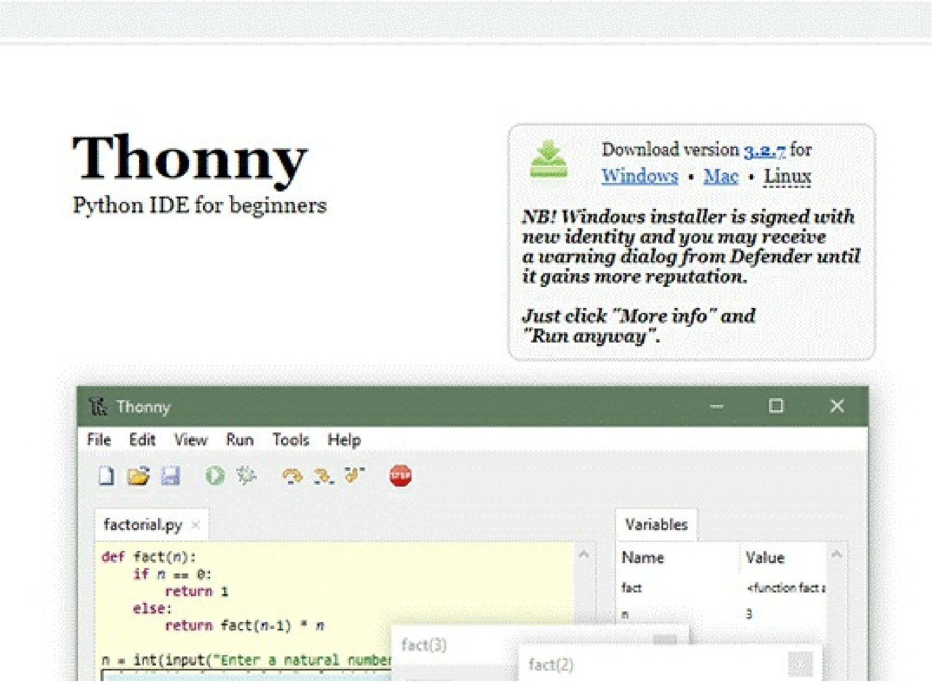
Note: If this point of time python is not recognized in the command prompt , then set the path of python3 properly.

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/



Thonny

Python IDE for beginners

Download version 3.2.7 for Windows • Mac • Linux

NB! Windows installer is signed with new identity and you may receive a warning dialog from Defender until it gains more reputation.

Just click "More info" and "Run anyway".

```
def fact(n):
    if n == 0:
        return 1
    else:
        return fact(n-1) * n

n = int(input("Enter a natural number
```

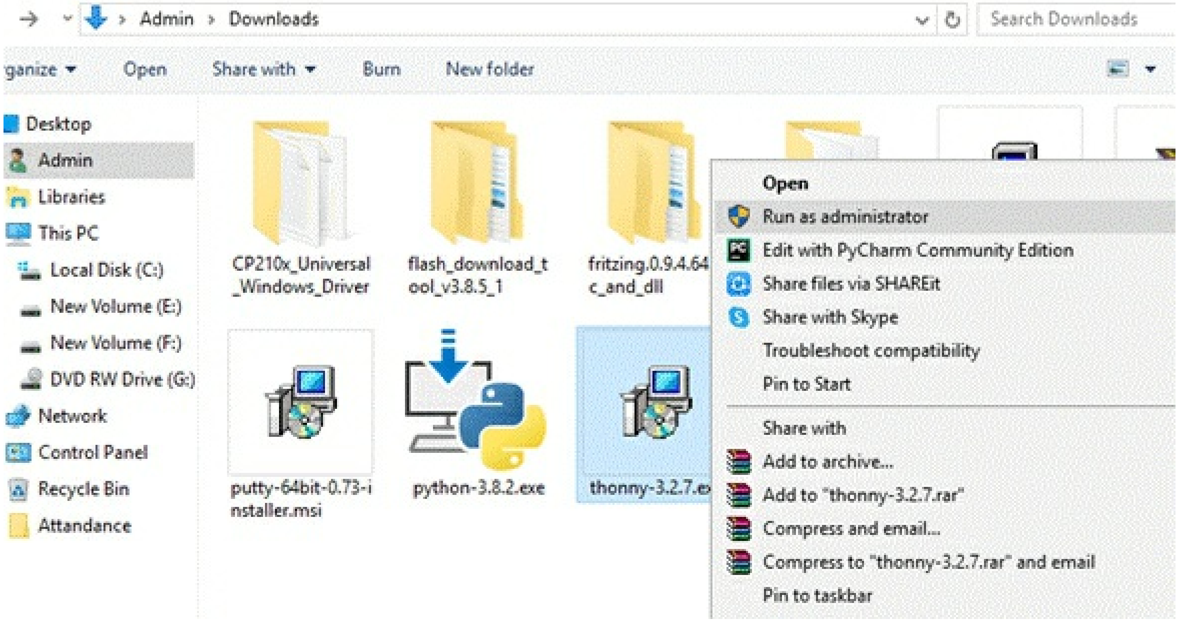
Name	Value
fact	<function fact
n	3

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

Step2:

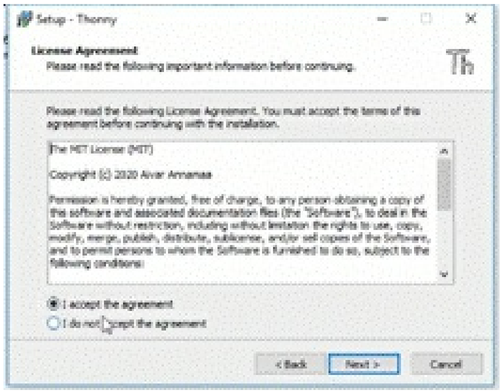
➢ Once the download is completed. Go to **'Downloads'** folder in the PC and find the downloaded **'Thonny-3.2.*.exe'** file.

➢ Right click on the **'thonny-3.2.8.exe'** file and click on **'Run as an administrator'**.



Step3:

- ➢ **Accept the agreement** from installation window and click on **‘Next>’**



License Agreement

Please read the following important information before continuing.

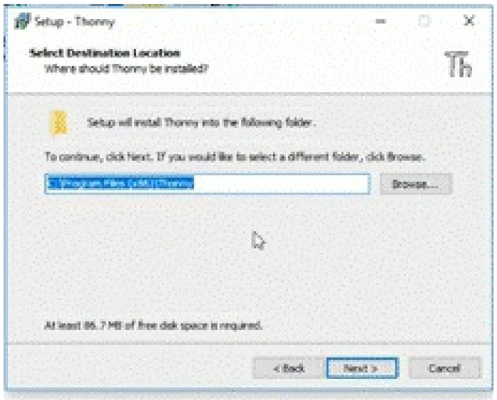
Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.

The MIT License (MIT)

Copyright (c) 2020 Aivar Annamaa

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

● I accept the agreement
○ I do not accept the agreement



Select Destination Location

Where should Thonny be installed?

Setup will install Thonny into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

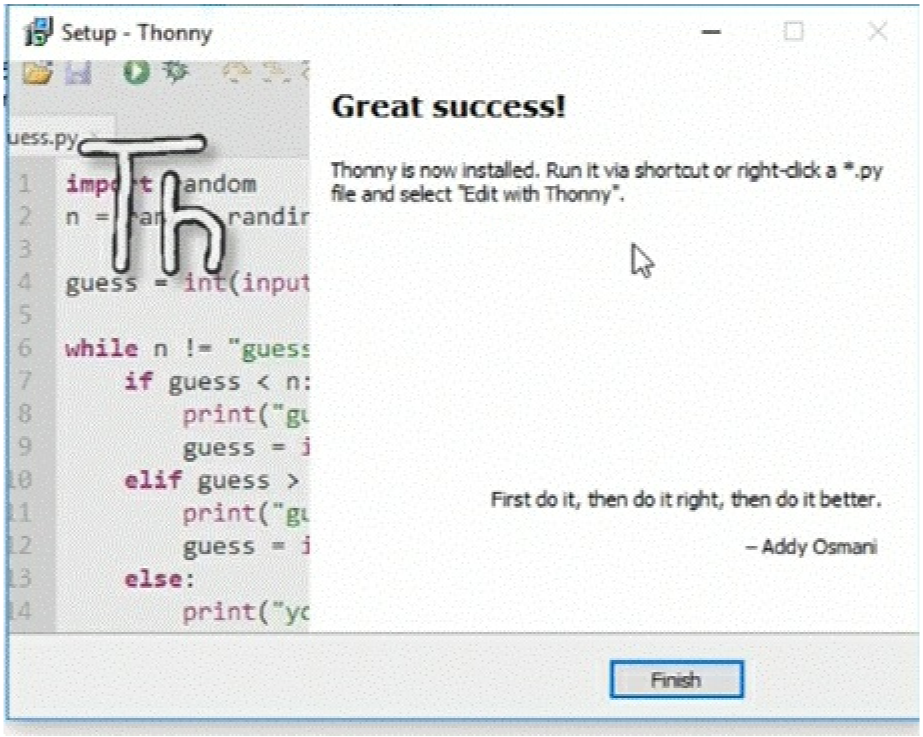
C:\Program Files (x86)\Thonny

At least 86.7 MB of free disk space is required.

- ➢ Browse the the folder you want to install and click on **‘Next >’**
- ➢ Wait until installation completes.

Step4:

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



Great success!

Thonny is now installed. Run it via shortcut or right-click a *.py file and select "Edit with Thonny".

First do it, then do it right, then do it better.

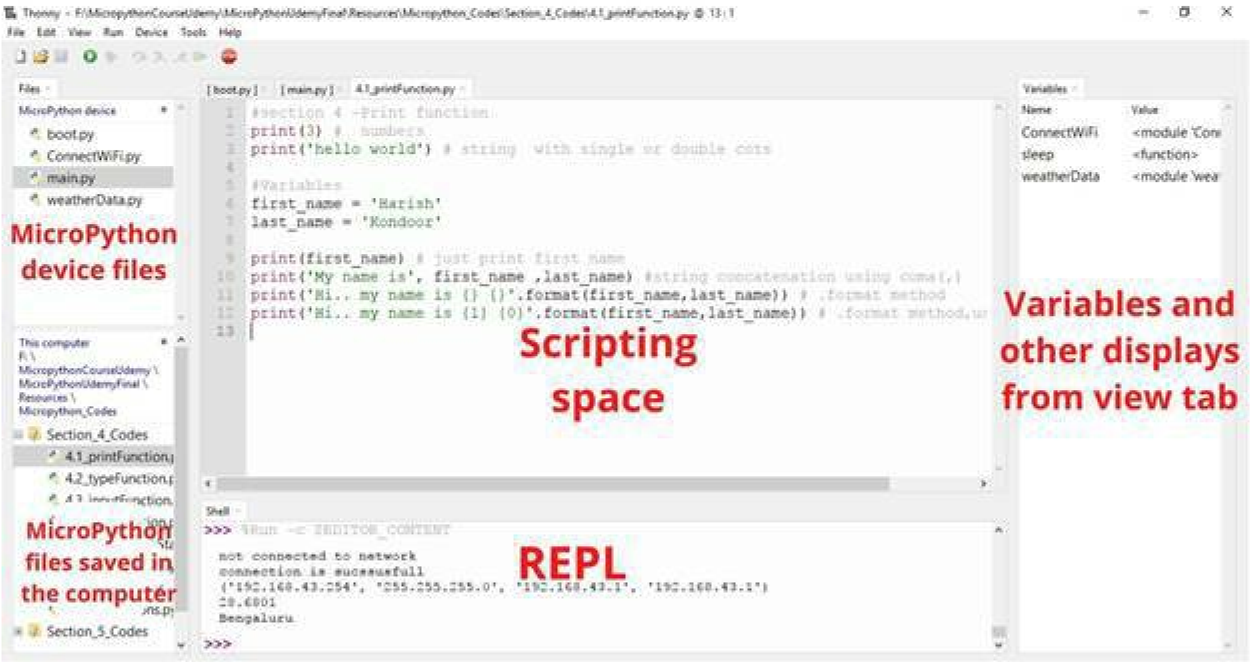
– Addy Osmani

```
import random
n = ran randir

guess = int(input

while n != "guess
    if guess < n:
        print("gu
        guess = i
    elif guess >
        print("gu
        guess = i
    else:
        print("yc
```

Step5:

- ➢ You can find the **Thonny IDE** icon from the desktop or find the application from the **‘Start menu’**. Launch the application.

2.3 Thonny IDE Interface


Thonny - F:\MicropythonCourseUdemy\MicroPythonUdemyFinal\Resources\Micropython_Codes\Section_4_Codes\4.1_printFunction.py @ 13 : 1
File Edit View Run Device Tools Help
Files
MicroPython device
boot.py
ConnectWiFi.py
main.py
weatherData.py
MicroPython device files
This computer
F:\ MicropythonCourseUdemy \ MicroPythonUdemyFinal \ Resources \ Micropython_Codes
Section_4_Codes
4.1_printFunction.py
4.2_typeFunction.py
4.3_inputFunction.py
MicroPython files saved in the computer
Section_5_Codes
[boot.py]
[main.py]
4.1_printFunction.py
Scripting space
Variables
Variables and other displays from view tab
Shell
REPL

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

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

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,u
```

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

 not connected to network
 connection is sucessfull
 ('192.168.43.254', '255.255.255.0', '192.168.43.1', '192.168.43.1')
 28.6801
 Bengaluru
>>>
```

Name	Value
ConnectWiFi	<module 'Conn
sleep	<function>
weatherData	<module 'wea

F5 : Save & Run the current script
Ctrl + N : New file
Ctrl + S : Save the file
Ctrl + D : Soft reboot
Ctrl + C : Keyword interrupt

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 **'espress-if'** 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**

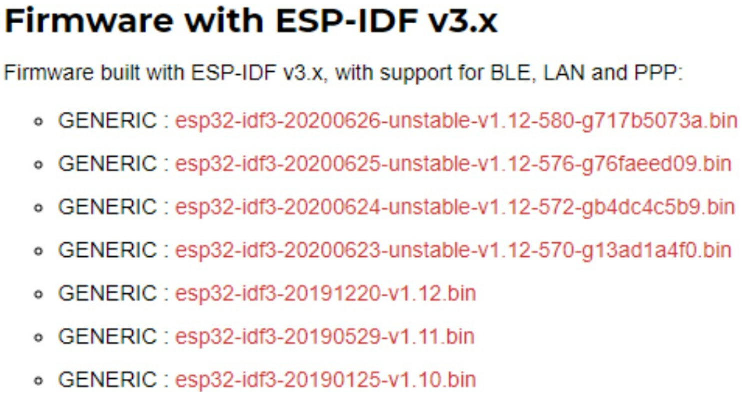
3.1 Downloading MicroPython firmware

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** 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-g76faeed09.bin
- GENERIC : esp32-idf3-20200624-unstable-v1.12-572-gb4dc4c5b9.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

3.2 Flashing Micropython to ESP32 using Thonny IDE

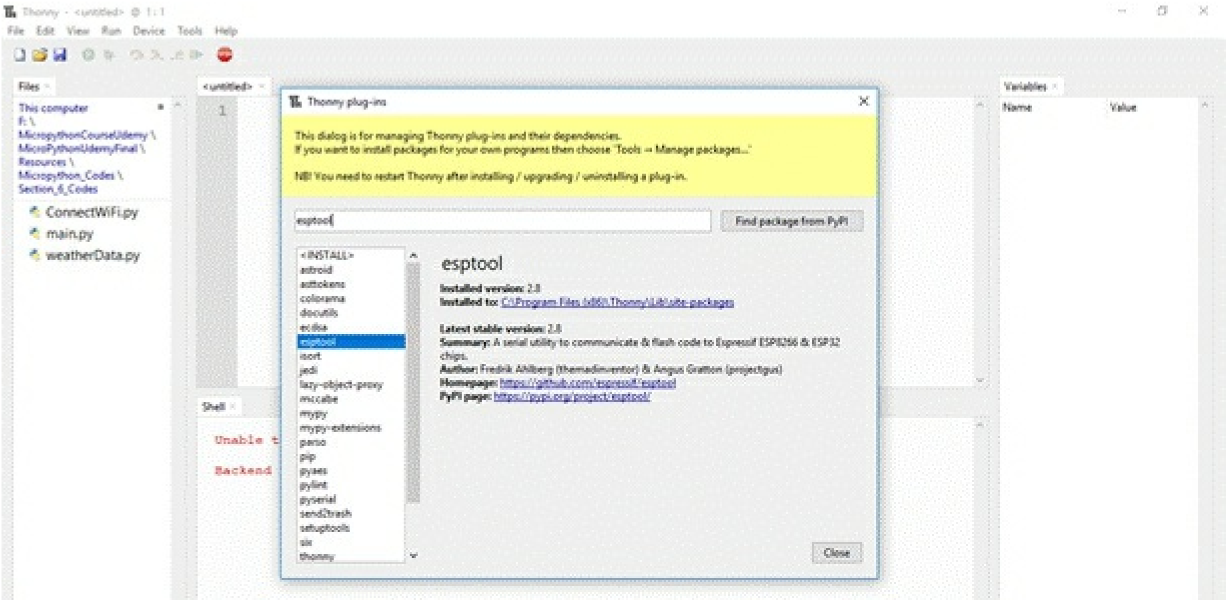
Step1:

➢ 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.



This dialog is for managing Thonny plug-ins and their dependencies.
If you want to install packages for your own programs then choose 'Tools → Manage packages...'

NB! You need to restart Thonny after installing / upgrading / uninstalling a plug-in.

esptool

Installed version: 2.8
Installed to: C:\Program Files (x86)\Thonny\Lib\site-packages

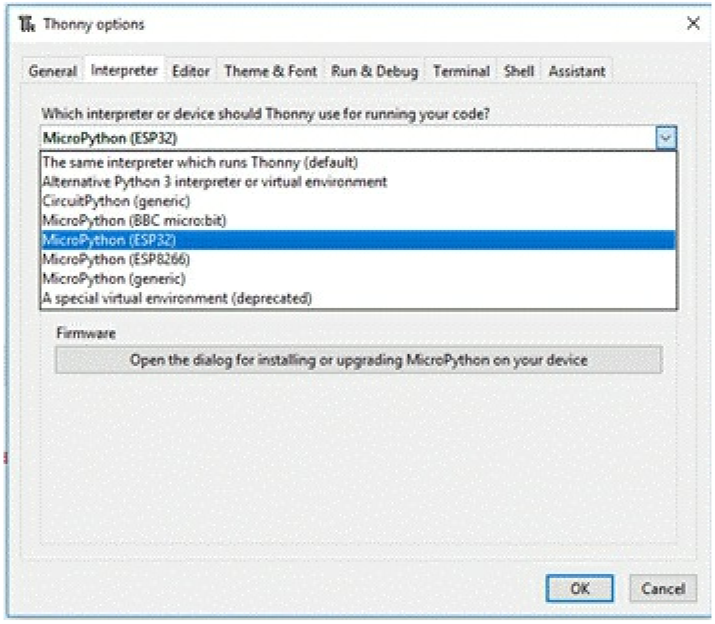
Latest stable version: 2.8
Summary: A serial utility to communicate & flash code to Espressif ESP8266 & ESP32 chips.
Author: Fredrik Ahlberg (themadinventor) & Angus Gratton (projectgus)
Homepage: https://github.com/espressif/esptool
PyPI page: https://pypi.org/project/esptool/

```
Unable t
Backend
```

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

Step3

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


Thonny options
General
Interpreter
Editor
Theme & Font
Run & Debug
Terminal
Shell
Assistant

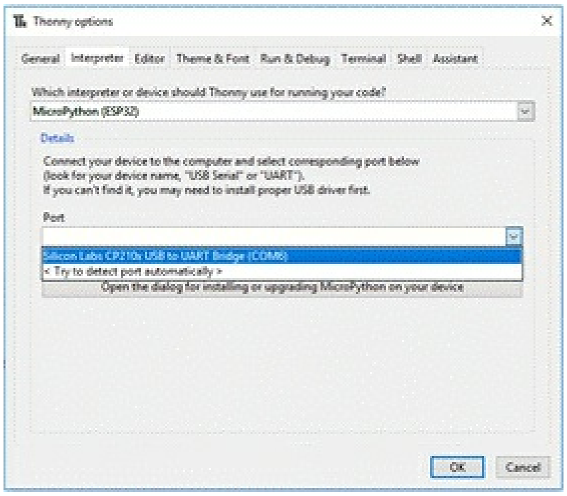
Which interpreter or device should Thonny use for running your code?

MicroPython (ESP32)

The same interpreter which runs Thonny (default)
Alternative Python 3 interpreter or virtual environment
CircuitPython (generic)
MicroPython (BBC micro:bit)
MicroPython (ESP32)
MicroPython (ESP8266)
MicroPython (generic)
A special virtual environment (deprecated)
Firmware
Open the dialog for installing or upgrading MicroPython on your device
OK
Cancel

Step4:

- ➢ From **Port** option select the port (eg:- COM6)


Thonny options
General
Interpreter
Editor
Theme & Font
Run & Debug
Terminal
Shell
Assistant

Which interpreter or device should Thonny use for running your code?

MicroPython (ESP32)

Details

Connect your device to the computer and select corresponding port below
(look for your device name, "USB Serial" or "UART").
If you can't find it, you may need to install proper USB driver first.

Port

Silicon Labs CP210x USB to UART Bridge (COM6)

< Try to detect port automatically >
Open the dialog for installing or upgrading MicroPython on your device
OK
Cancel

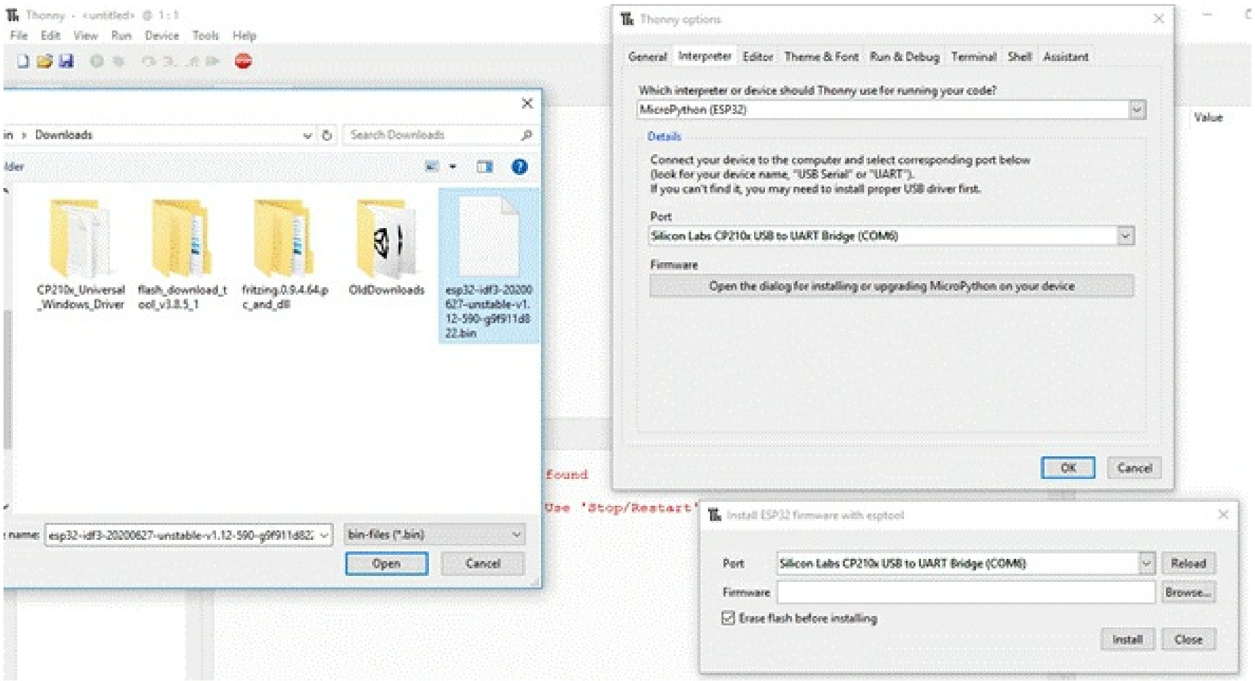
Note: If the port number is not visible, check the quality of the USB cable used and install the Device driver CP210x or CH340 to PC.

Step5:

- ➢ Click on the **firmware option**.

Step6:

- ➢ From the new window select the **Port** (ex:- COM6)
- ➢ Browse to downloaded **.bin** file → click on **'Open'**


Thonny - <untitled> @ 1 : 1
File
Edit
View
Run
Device
Tools
Help
Downloads
Search Downloads
CP210x_Universal_Windows_Driver
flash_download_tool_v3.8.5_1
fritzing.0.9.4.64.pc_and_dll
OldDownloads
esp32-idf3-20200627-unstable-v1.12-590-g9f911d822.bin
Open
Cancel
found
Use 'Stop/Restart'
Thonny options
General
Interpreter
Editor
Theme & Font
Run & Debug
Terminal
Shell
Assistant
Details
Open the dialog for installing or upgrading MicroPython on your device
OK
Cancel
Value
Install ESP32 firmware with esptool
Reload
Browse...
Install
Close

File name: esp32-idf3-20200627-unstable-v1.12-590-g9f911d82:

bin-files (*.bin)

Which interpreter or device should Thonny use for running your code?

MicroPython (ESP32)

Connect your device to the computer and select corresponding port below
(look for your device name, "USB Serial" or "UART").
If you can't find it, you may need to install proper USB driver first.

Port: Silicon Labs CP210x USB to UART Bridge (COM6)

Firmware

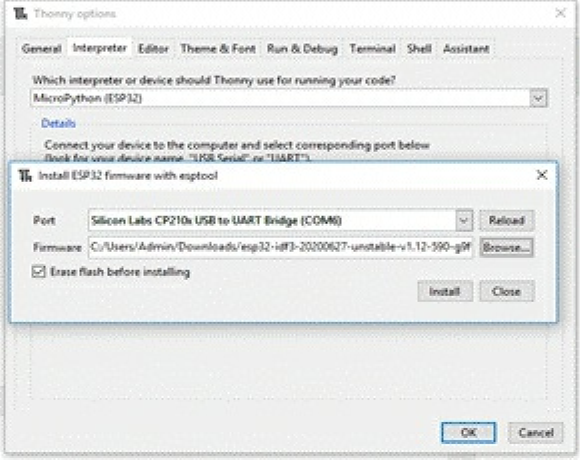
Port: Silicon Labs CP210x USB to UART Bridge (COM6)

Firmware:

☑ Erase flash before installing

Step7:

- Tick the option **‘Erase flash before installing’**

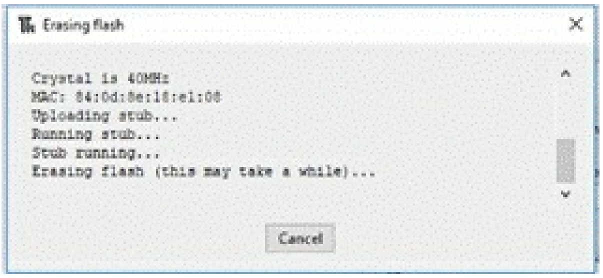


Which interpreter or device should Thonny use for running your code?
MicroPython (ESP32)

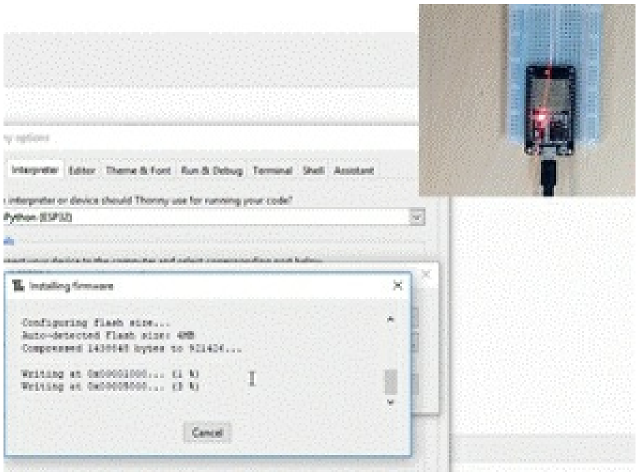
Connect your device to the computer and select corresponding port below
(look for your device name, "USB Serial" or "UART").

Port: Silicon Labs CP210x USB to UART Bridge (COM6)
Firmware: C:/Users/Admin/Downloads/esp32-idf3-20200627-unstable-v1.12-590-g9f
☑ 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 firmware**.



```
Crystal is 40MHz
MAC: 84:0d:8e:18:e1:08
Uploading stub...
Running stub...
Stub running...
Erasing flash (this may take a while)...
```



interpreter or device should Thonny use for running your code?
Python (ESP32)

```
Configuring flash size...
Auto-detected flash size: 4MB
Compressed 1439148 bytes to 921424...

Writing at 0x00001000... (1 %)
Writing at 0x00005000... (3 %)
```

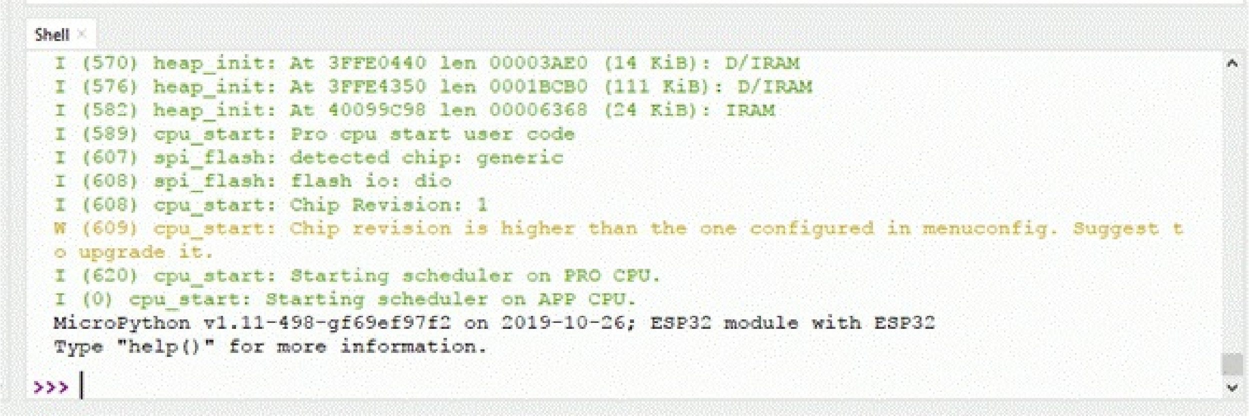
- Once firmware is 100% loaded press on the **EN / RESET** pin in the

ESP32 (**RST** in the case of ESP8266).

- You have successfully flashed MicroPython firmware to ESP32.
- Close the Thonny IDE and disconnect ESP32 from PC.

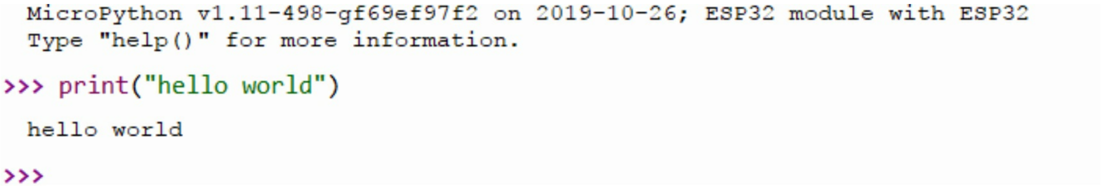
Step8:

- ➢ 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



```
I (570) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM
I (576) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (582) heap_init: At 40099C98 len 00006368 (24 KiB): IRAM
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
W (609) cpu_start: Chip revision is higher than the one configured in menuconfig. Suggest t
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-gf69ef97f2 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-gf69ef97f2 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**

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, 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.

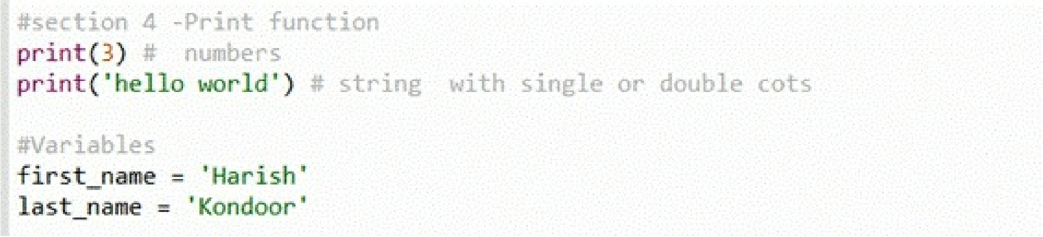
Topics

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

4.1 Print function

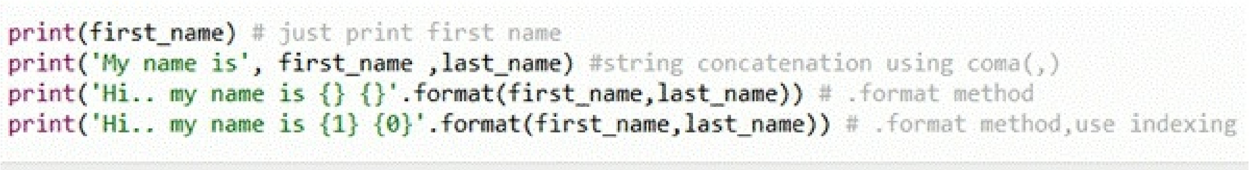
Steps:

- ➢ 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'
```

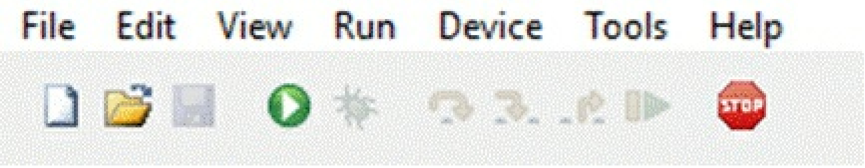
- ➢ For print the number 3.
- ➢ For print the string- hello world , string can be provided in **“ ”** double cots or in **‘ ‘** single cots.
- ➢ Syntax for print in the REPL is **print(X)** where X can be variable, numbers, boolean etc.



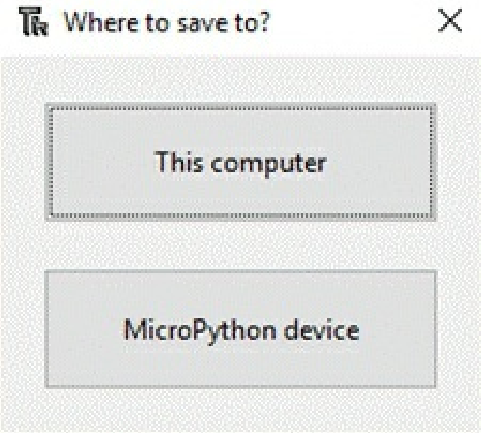
```
print(first_name) # just print first name
print('My name is', first_name ,last_name) #string concatenation using coma(,)
print('Hi.. my name is {} {}'.format(first_name,last_name)) # .format method
print('Hi.. my name is {1} {0}'.format(first_name,last_name)) # .format method,use indexing
```

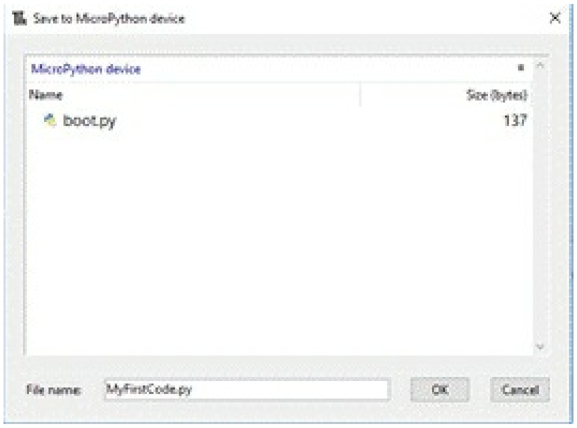
For string concatenation, coma **(,)** or plus **(+)** sign can be used. See the second line from the above example.

- ➢ 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.



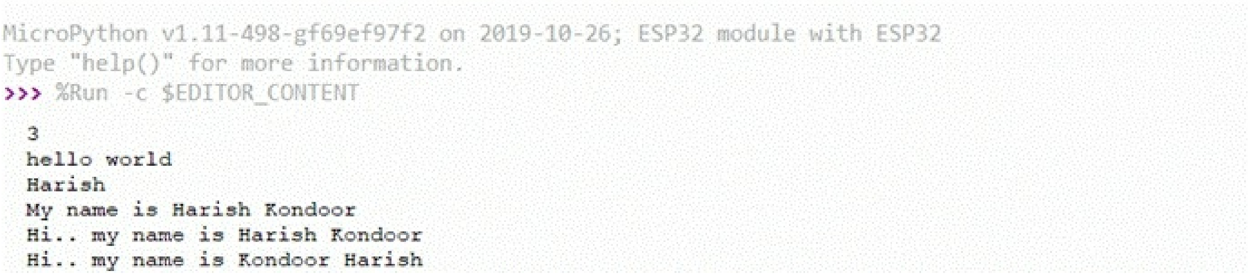
- ➢ If the script is not saved, from the new prompt window, you can choose where to save the script. Is it to the computer or to the **device.**
- ➢ Make sure you are providing a **.py** file extension to the script. (**ex:- MyFirstCode.py**)
- ➢ Then click ‘**OK**’





File name: MyFirstCode.py

- ➢ Output will get in the **REPL** as shown below



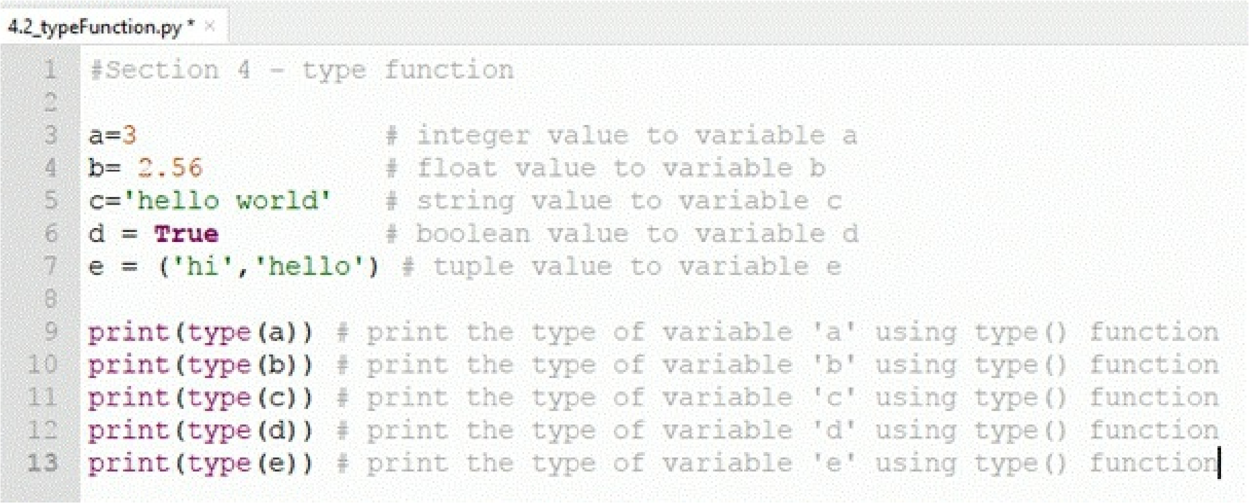
```
MicroPython v1.11-498-gf69ef97f2 on 2019-10-26; ESP32 module with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

  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.

4.2 Type function

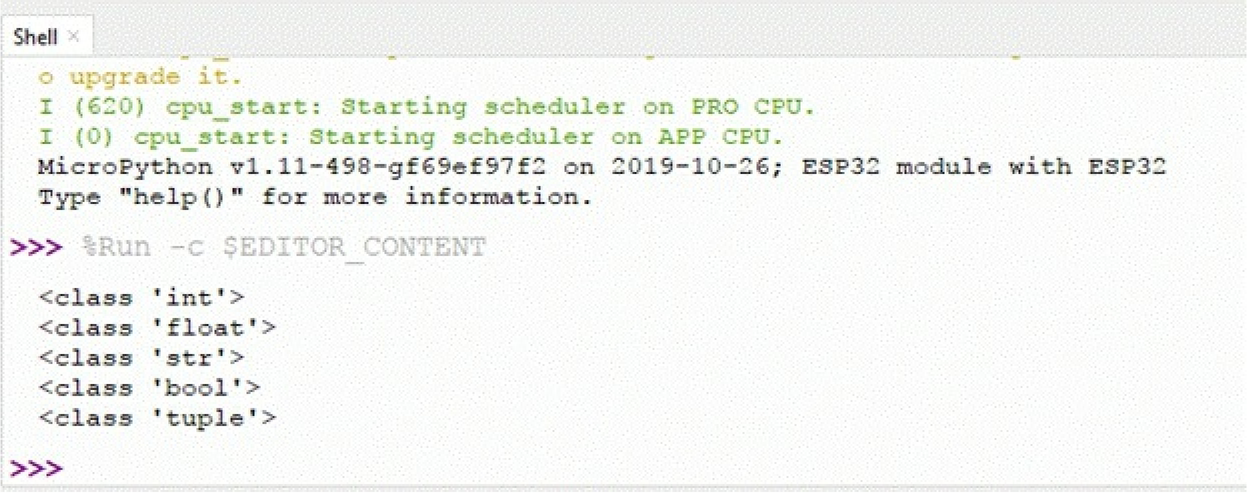
type() function is used to know the type of the variable like, is that an integer, string, boolean etc.

- ➢ **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))**



```
#Section 4 - type function

a=3                  # integer value to variable a
b= 2.56              # float value to variable b
c='hello world'      # string value to variable c
d = True             # boolean value to variable d
e = ('hi','hello') # tuple value to variable e

print(type(a)) # print the type of variable 'a' using type() function
print(type(b)) # print the type of variable 'b' using type() function
print(type(c)) # print the type of variable 'c' using type() function
print(type(d)) # print the type of variable 'd' using type() function
print(type(e)) # print the type of variable 'e' using type() function
```

- ➢ Output in the REPL

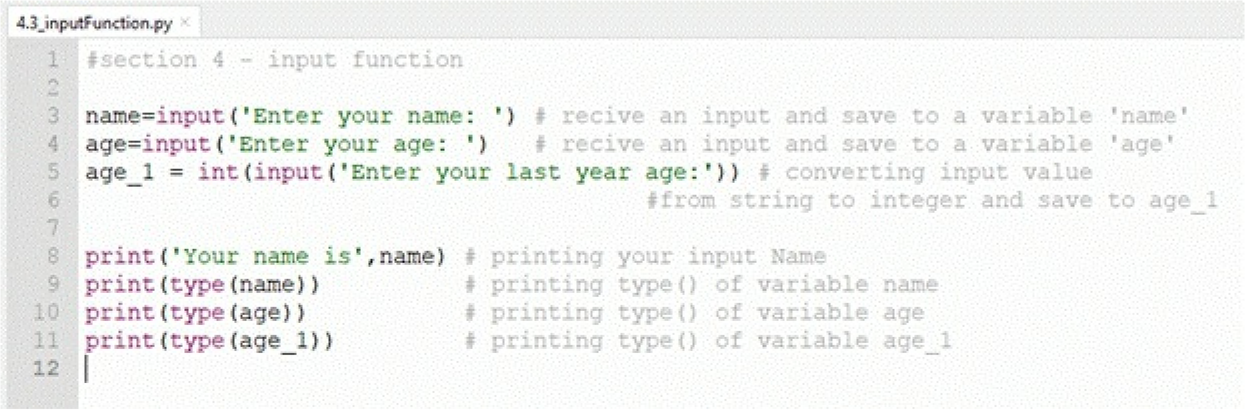


```
  o upgrade it.
  I (620) cpu_start: Starting scheduler on PRO CPU.
  I (0) cpu_start: Starting scheduler on APP CPU.
  MicroPython v1.11-498-gf69ef97f2 on 2019-10-26; ESP32 module with ESP32
  Type "help()" for more information.

>>> %Run -c $EDITOR_CONTENT

  <class 'int'>
  <class 'float'>
  <class 'str'>
  <class 'bool'>
  <class 'tuple'>

>>>
```

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.

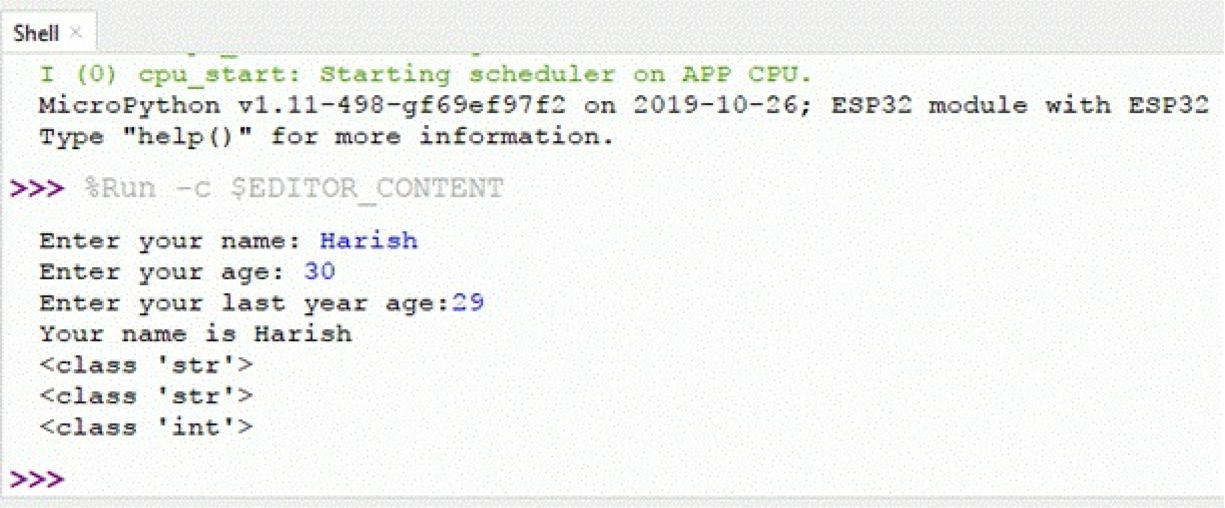
- **Syntax: input(“Ask the input from user”)**



```
 1  #section 4 - input function
 2
 3  name=input('Enter your name: ') # recive an input and save to a variable 'name'
 4  age=input('Enter your age: ')   # recive 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.

- Output in the REPL



```
 I (0) cpu_start: Starting scheduler on APP CPU.
 MicroPython v1.11-498-gf69ef97f2 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'>

>>>
```

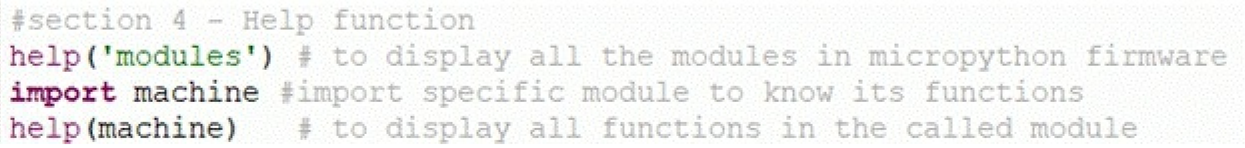
- ➢ Variable **name** is a **string**
- ➢ Variable **age** is a **string**
- ➢ Variable **age_1** converted as an integer using **int()** before **input()** function.

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.

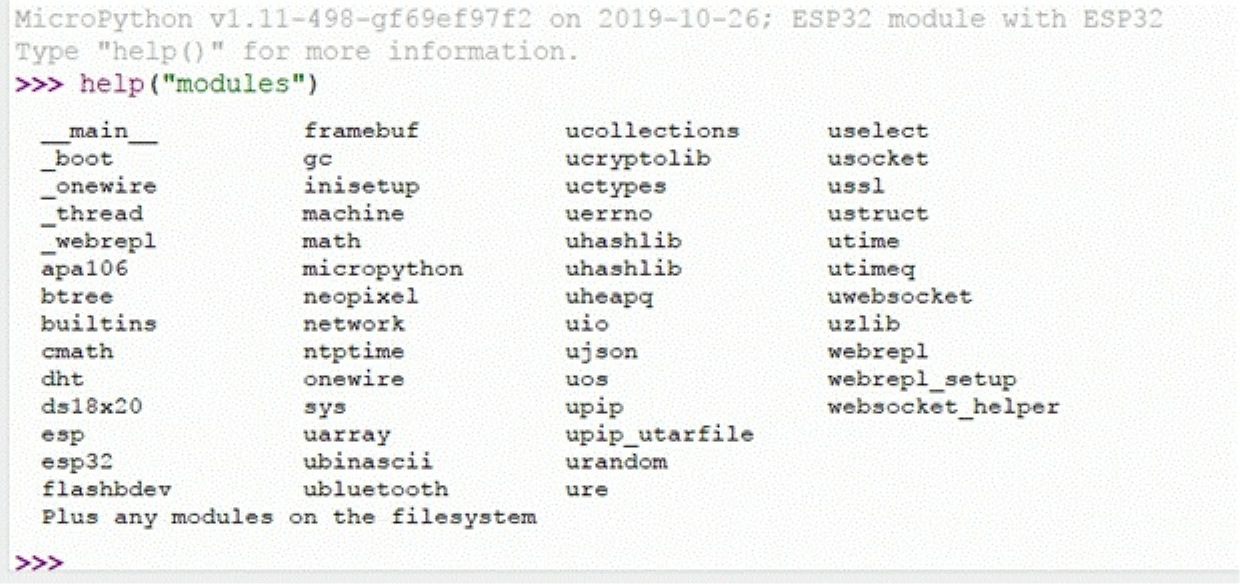
Syntax:

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



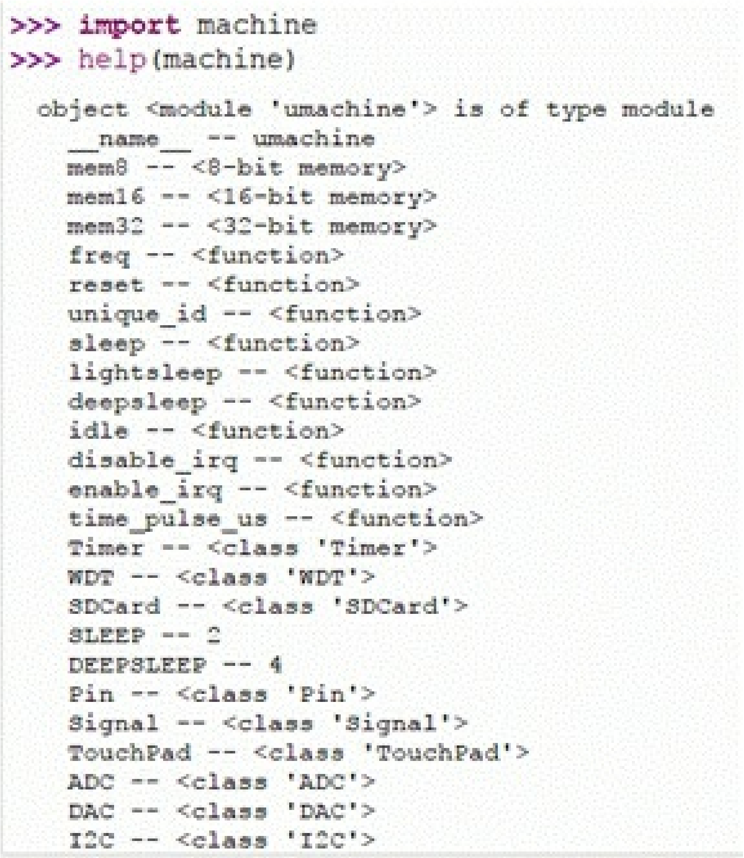
```
#section 4 - Help function
help('modules') # to display all the modules in micropython firmware
import machine #import specific module to know its functions
help(machine)   # to display all functions in the called module
```

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

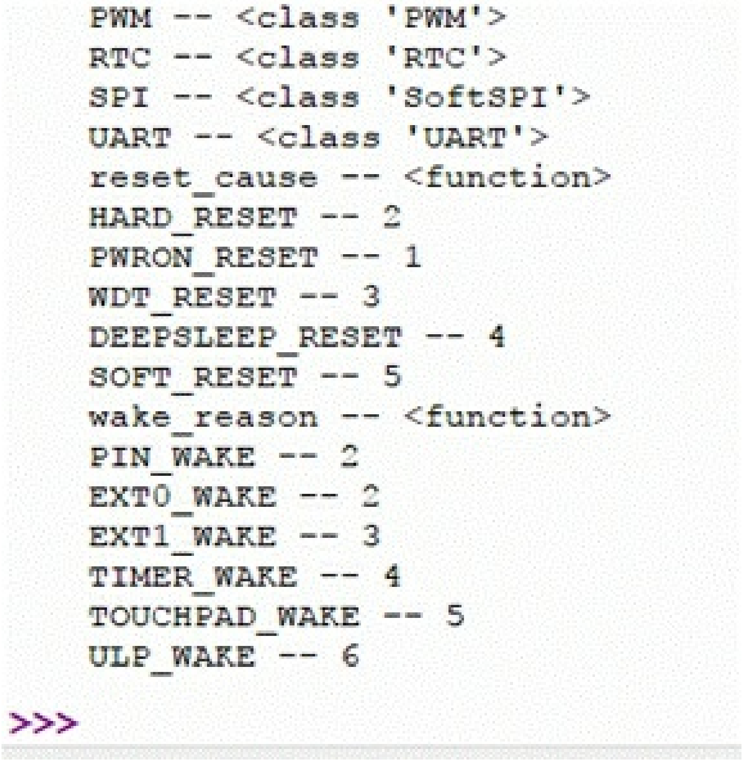


```
MicroPython v1.11-498-gf69ef97f2 on 2019-10-26; ESP32 module with ESP32
Type "help()" for more information.
>>> help("modules")

  __main__          framebuf          ucollections      uselect
  _boot             gc                ucryptolib        usocket
  _onewire          inisetup          uctypes           ussl
  _thread           machine           uerrno            ustruct
  _webrepl          math              uhashlib          utime
  apa106            micropython       uhashlib          utimeq
  btree             neopixel          uheapq            uwebsocket
  builtins          network           uio               uzlib
  cmath             ntptime           ujson             webrepl
  dht               onewire           uos               webrepl_setup
  ds18x20           sys               upip              websocket_helper
  esp               uarray            upip_utarfile
  esp32             ubinascii         urandom
  flashbdev         ubluetooth        ure
  Plus any modules on the filesystem

>>>
```

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'>
    TouchPad -- <class 'TouchPad'>
    ADC -- <class 'ADC'>
    DAC -- <class 'DAC'>
    I2C -- <class 'I2C'>
```



```
    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 .**

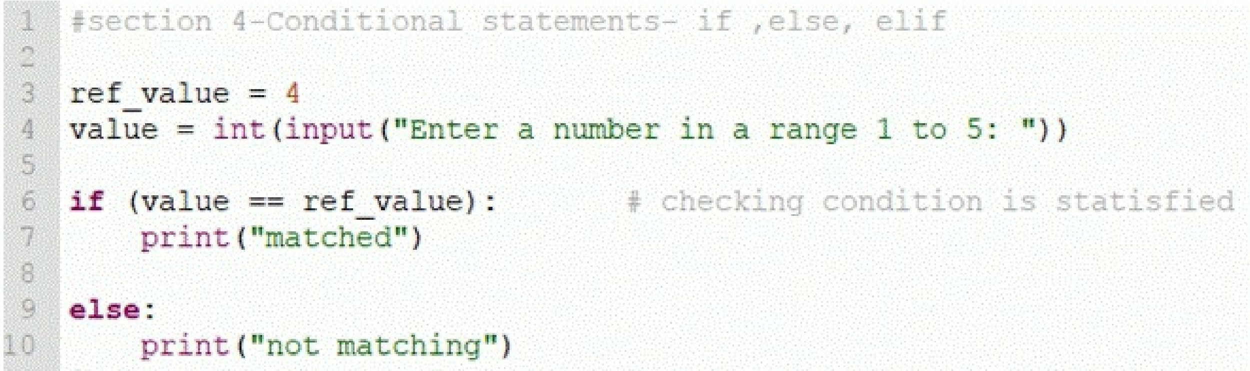
4.5 Conditional statements (if, else, elif)

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
```

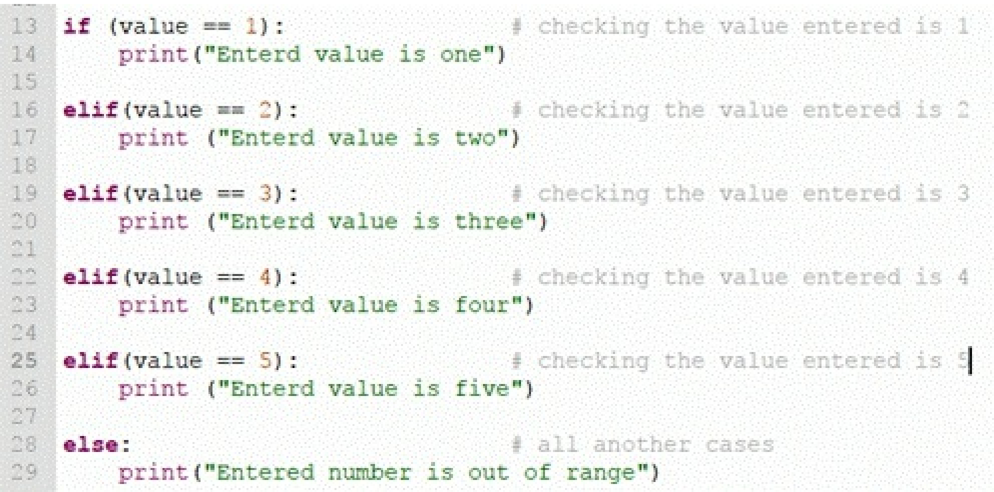


```
 1  #section 4-Conditional statements- if ,else, elif
 2
 3  ref_value = 4
 4  value = int(input("Enter a number in a range 1 to 5: "))
 5
 6  if (value == ref_value):        # checking condition is statisfied
 7      print("matched")
 8
 9  else:
10      print("not matching")
```

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.



```
if (value == 1):                # checking the value entered is 1
    print("Enterd value is one")

elif(value == 2):               # checking the value entered is 2
    print ("Enterd value is two")

elif(value == 3):               # checking the value entered is 3
    print ("Enterd value is three")

elif(value == 4):               # checking the value entered is 4
    print ("Enterd value is four")

elif(value == 5):               # checking the value entered is 5
    print ("Enterd value is five")

else:                           # all another cases
    print("Entered number is out of range")
```

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

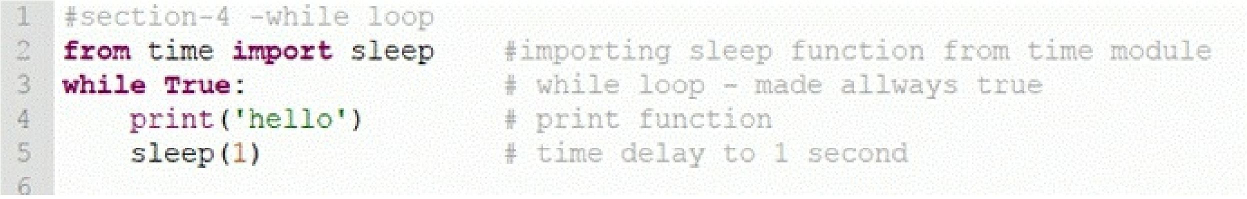
4.6 While loop

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

Syntax:

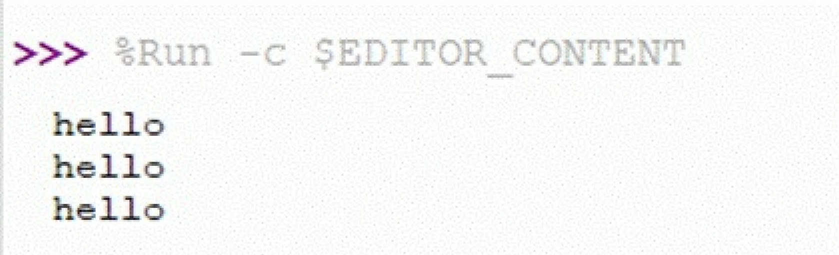
While Condition is satisfied:
Statement1
Statement 2

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 allways 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.
- 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.

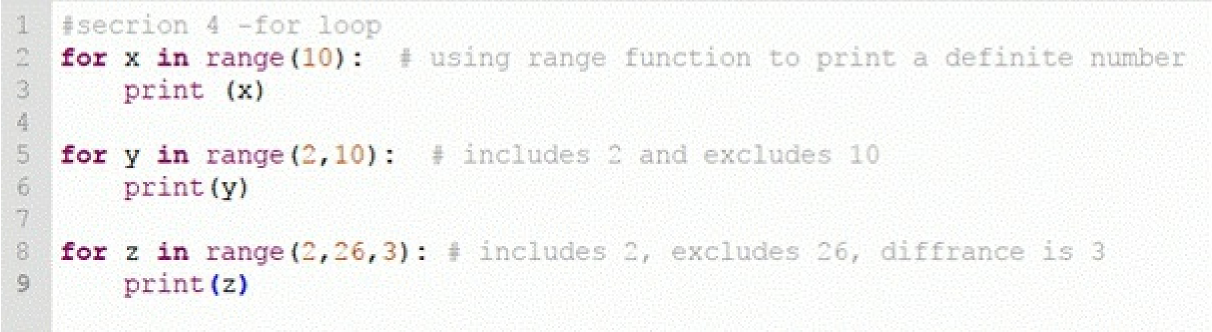
4.7 For loop

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  #secrion 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, diffrance 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

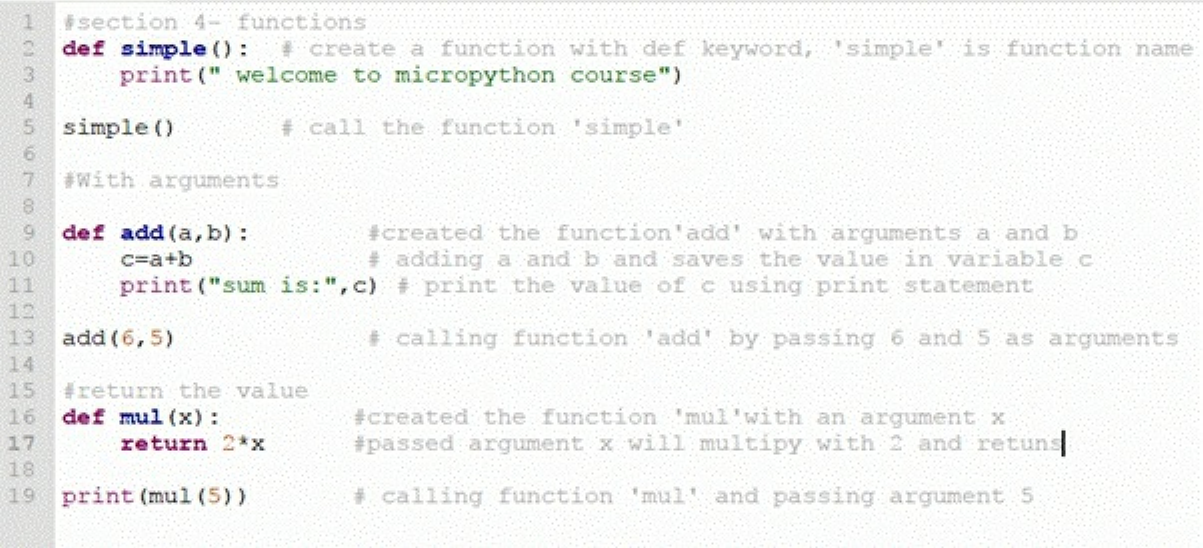
Creating your own function is important in MicroPython programming. A set of codes or tasks can be set in a function. So that you don't need to repeat the same set of codes somewhere else in the script, instead you can call the function just with a single line of code.

Syntax (*Creating a function*):

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

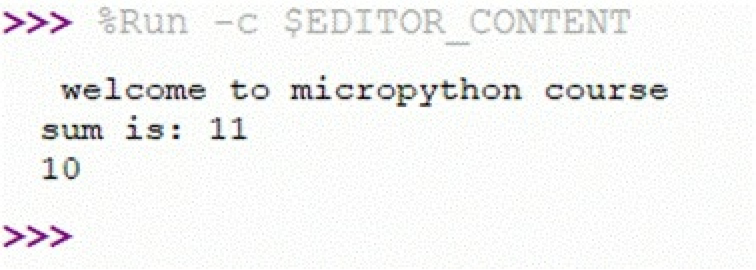
Syntax (*Calling a function*):

```
function_name(arguments)
```



```
#section 4- functions
def simple():  # create a function with def keyword, 'simple' is function name
    print(" welcome to micropython course")

simple()        # call the function 'simple'

#With arguments

def add(a,b):          #created the function'add' with arguments a and b
    c=a+b              # adding a and b and saves the value in variable c
    print("sum is:",c) # print the value of c using print statement

add(6,5)               # calling function 'add' by passing 6 and 5 as arguments

#return the value
def mul(x):           #created the function 'mul'with an argument x
    return 2*x        #passed argument x will multipy with 2 and retuns

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

>>>
```

Module5 - Controlling GPIO pins

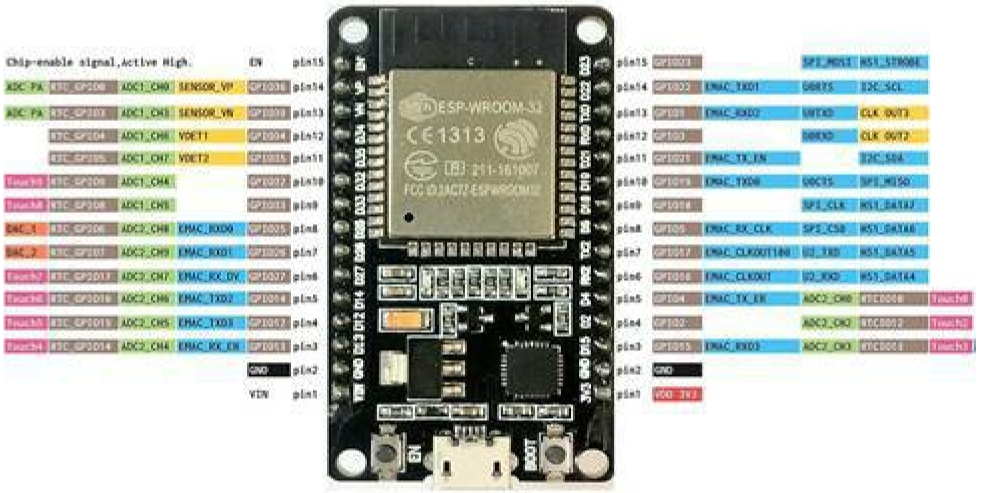
Controlling general purpose input output (GPIO) pins is comparatively comfortable in MicroPython.

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
- MultiThreading

5.1 Pinout diagram of ESP32



- VIN - Input voltage (maximum 10 V).
- 3V3 - Output voltage of 3.3 V (AMS1117 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.

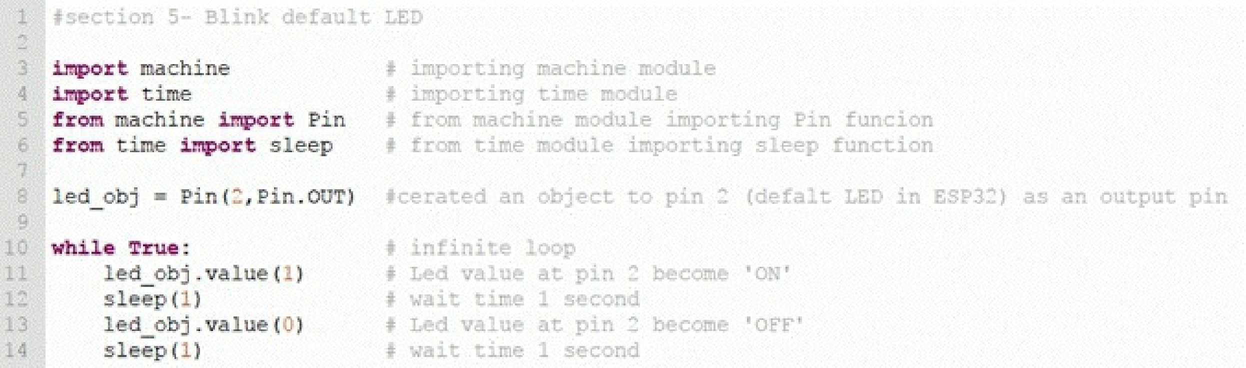
ESP32 chip works with 3.3 volt logic level. Make sure input connections to ESP32 from sensors maintain 3.3V logic level. If sensors give a 5 V logic level, use basic resistors based voltage divider circuits to cut down the voltage to 3.3V .

- 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 port without any fail.
- ESP32 comes with a power LED.
- Inbuilt LED is at GPIO 2.

5.2 Blink an LED

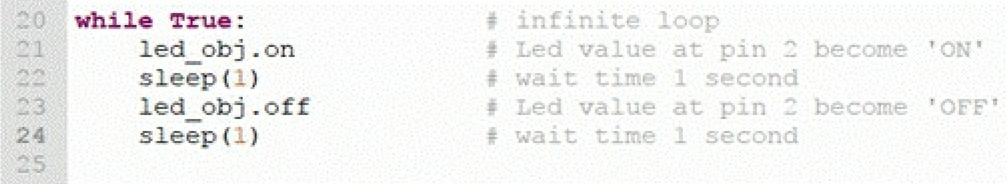
ESP32 comes with an Inbuilt LED at GPIO 2. We can start blinking this LED as our first project in this section.

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.



```
#section 5- Blink default LED

import machine            # importing machine module
import time               # importing time module
from machine import Pin   # from machine module importing Pin funcion
from time import sleep    # from time module importing sleep function

led_obj = Pin(2,Pin.OUT)  #cerated an object to pin 2 (defalt LED in ESP32) as an output pin

while True:               # infinite loop
    led_obj.value(1)      # Led value at pin 2 become 'ON'
    sleep(1)              # wait time 1 second
    led_obj.value(0)      # Led value at pin 2 become 'OFF'
    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.

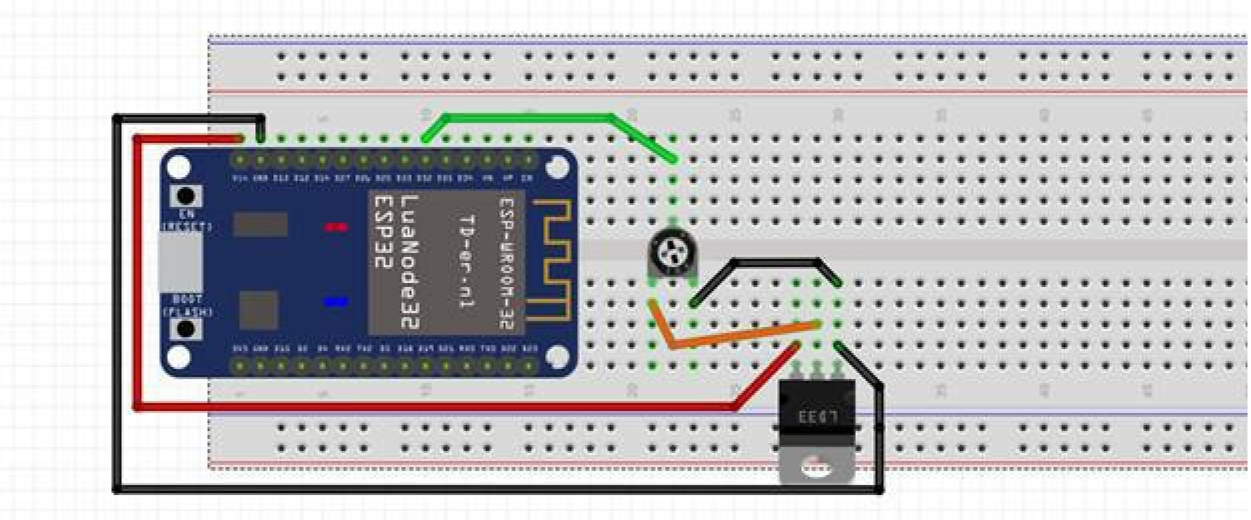


```
while True:               # infinite loop
    led_obj.on            # Led value at pin 2 become 'ON'
    sleep(1)              # wait time 1 second
    led_obj.off           # Led value at pin 2 become 'OFF'
    sleep(1)              # wait time 1 second

```

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.

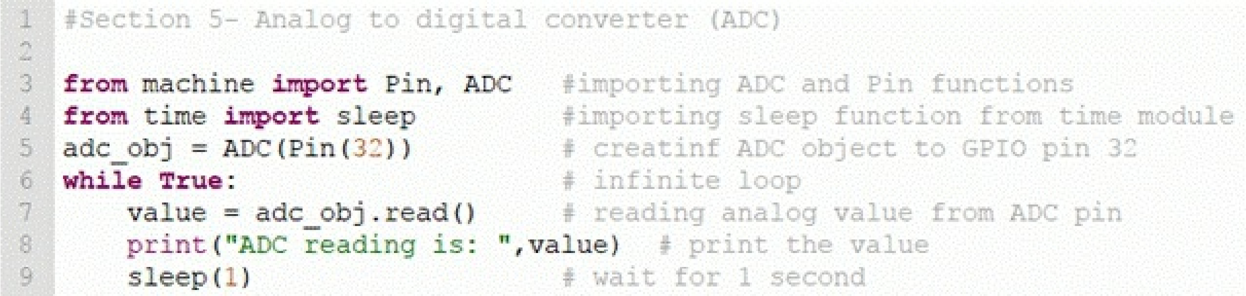


- ➢ LD33 voltage regulator has been used to regulate the voltage to 3.3 V for powering external components.
- ➢ A 10 kilo ohm potentiometer is used for ADC conversion.
- ➢ GPIO32 is used as an ADC pin.

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.

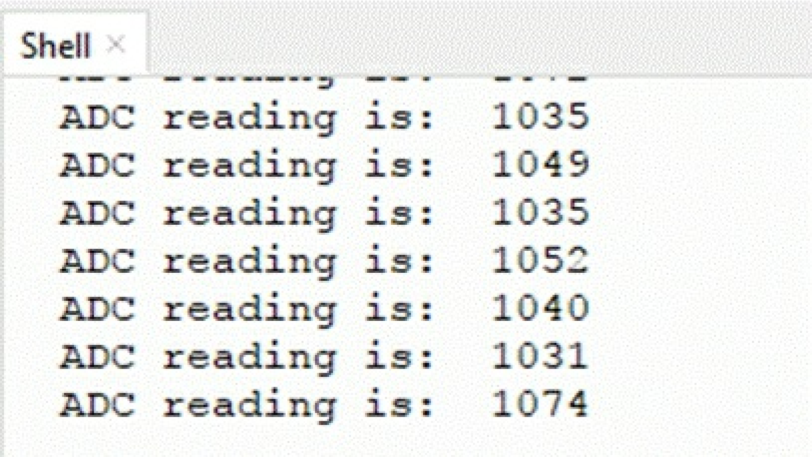
Note: ESP8266 has only one ADC pin.

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



```
#Section 5- Analog to digital converter (ADC)

from machine import Pin, ADC   #importing ADC and Pin functions
from time import sleep         #importing sleep function from time module
adc_obj = ADC(Pin(32))         # creatinf ADC object to GPIO pin 32
while True:                    # infinite loop
    value = adc_obj.read()     # reading analog value from ADC pin
    print("ADC reading is: ",value)  # print the value
    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.



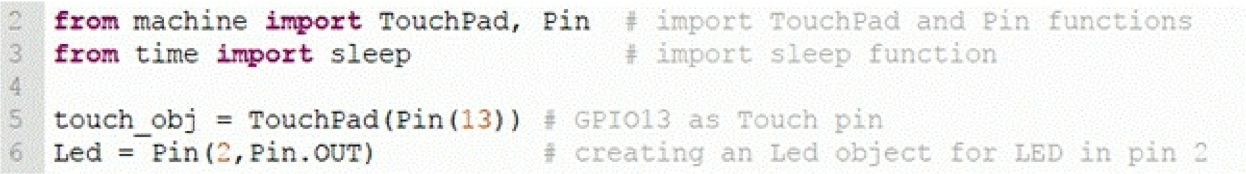
```
ADC reading is:  1035
ADC reading is:  1049
ADC reading is:  1035
ADC reading is:  1052
ADC reading is:  1040
ADC reading is:  1031
ADC reading is:  1074
```

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**.

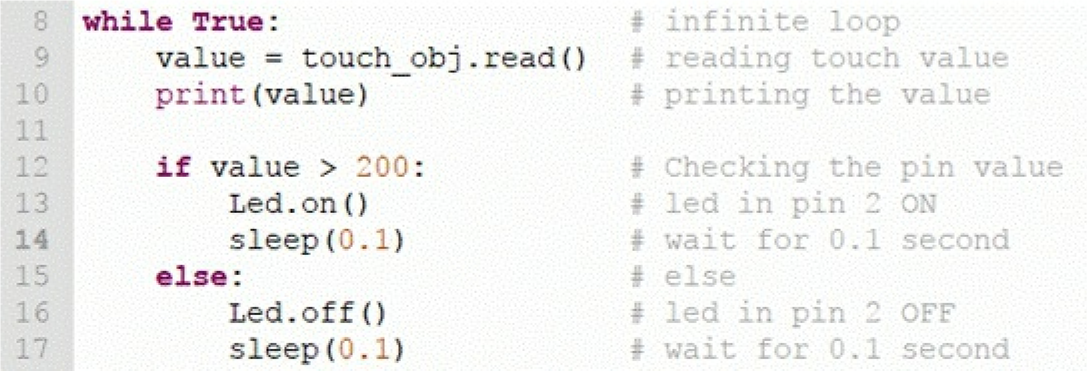
Note:This feature is not available in the ESP8266.

- ➢ 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.



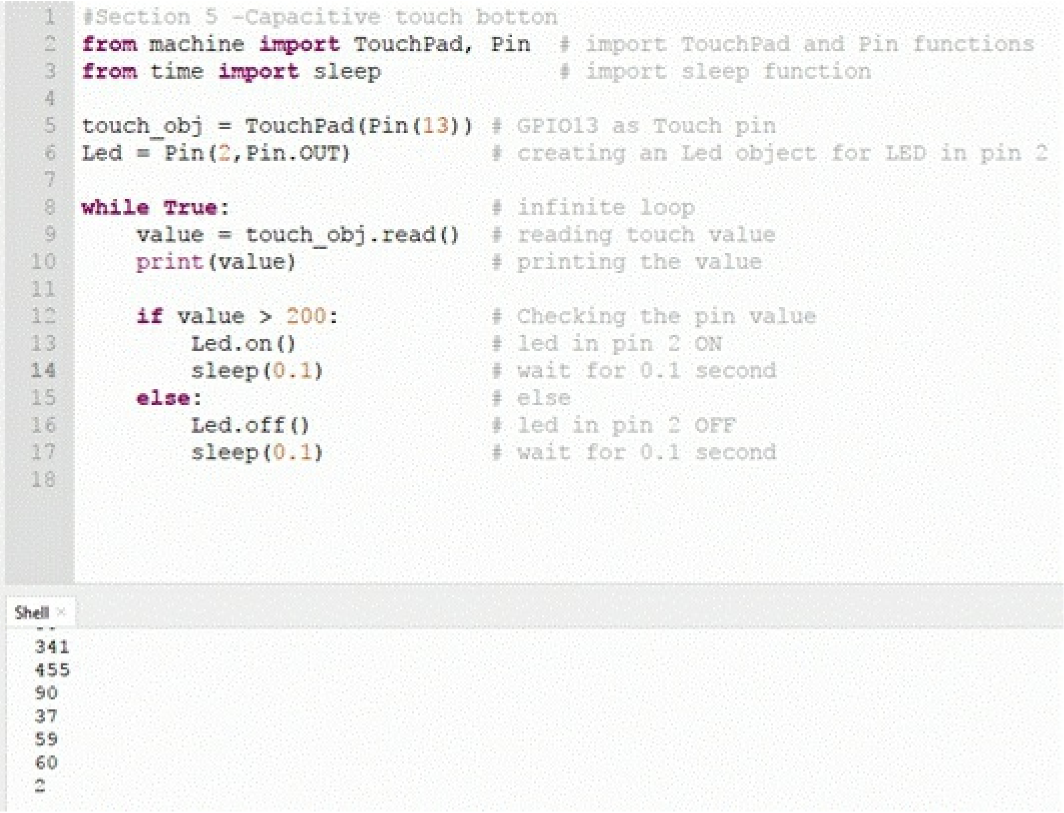
```
2  from machine import TouchPad, Pin  # import TouchPad and Pin functions
3  from time import sleep             # import sleep function
4
5  touch_obj = TouchPad(Pin(13)) # GPIO13 as Touch pin
6  Led = Pin(2,Pin.OUT)          # creating an Led object for LED in pin 2
```

- ➢ 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
```

- 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.



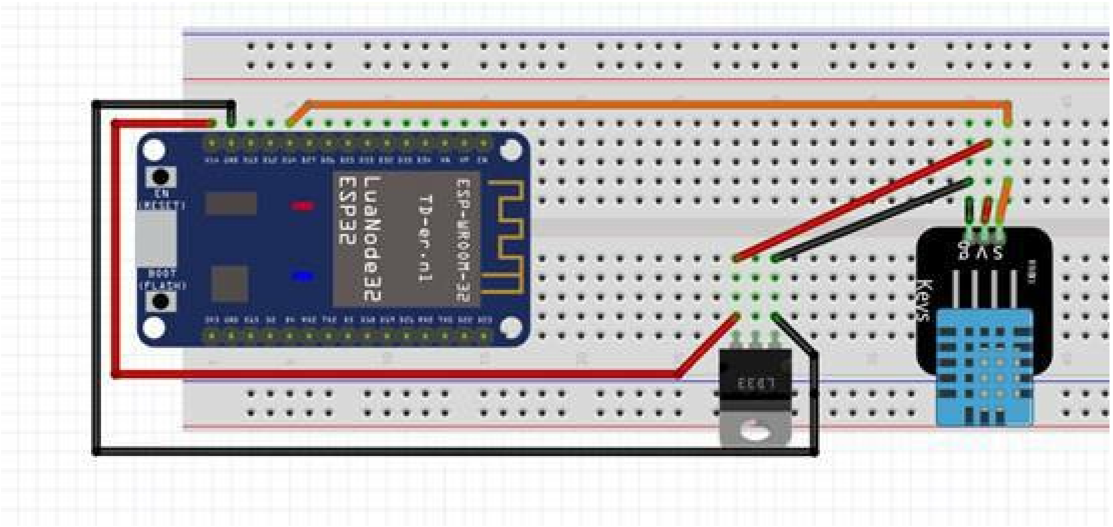
```
 1  #Section 5 -Capacitive touch botton
 2  from machine import TouchPad, Pin  # import TouchPad and Pin functions
 3  from time import sleep             # import sleep function
 4
 5  touch_obj = TouchPad(Pin(13)) # GPIO13 as Touch pin
 6  Led = Pin(2,Pin.OUT)          # creating an Led object for 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
```

```
 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.



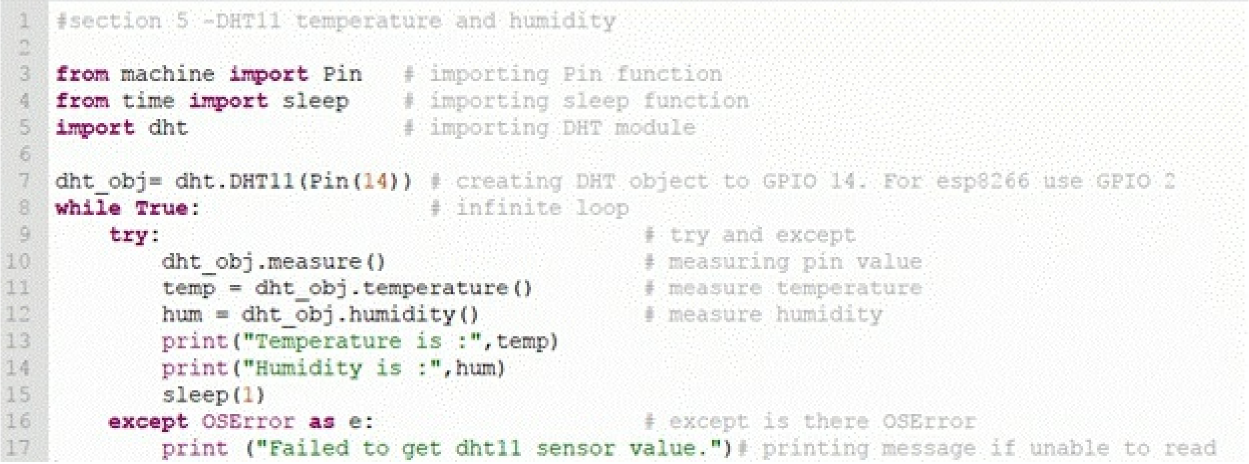
- ➢ Signal pin is connected to GPIO14. Use GPIO2 for ESP8266
- ➢ Ground pin from DHT11 to GND of LD33
- ➢ Power pin from DHT11 to 3.3 V output of LD33

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.

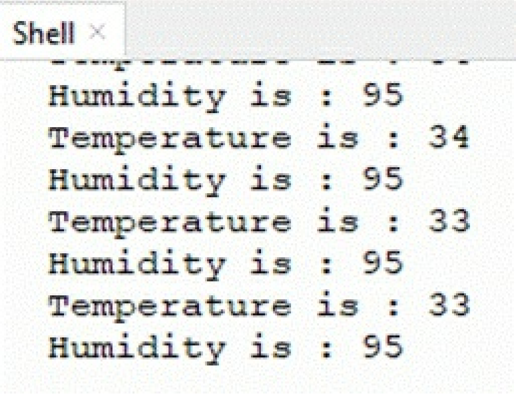
- ➢ Using **temperature()** and **humidity()** functions you can read out temperature and humidity separately and save to different variables and later on print it.

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.



```
#section 5 -DHT11 temperature and humidity

from machine import Pin    # importing Pin function
from time import sleep     # importing sleep function
import dht                 # importing DHT module

dht_obj= dht.DHT11(Pin(14)) # creating DHT object to GPIO 14. For esp8266 use GPIO 2
while True:                 # infinite loop
    try:                                # try and except
        dht_obj.measure()               # measuring pin value
        temp = dht_obj.temperature()    # measure temperature
        hum = dht_obj.humidity()        # measure humidity
        print("Temperature is :",temp)
        print("Humidity is :",hum)
        sleep(1)
    except OSError as e:                # except is there OSError
        print ("Failed to get dht11 sensor value.")# printing message if unable to read
```

- ➢ Output in the REPL

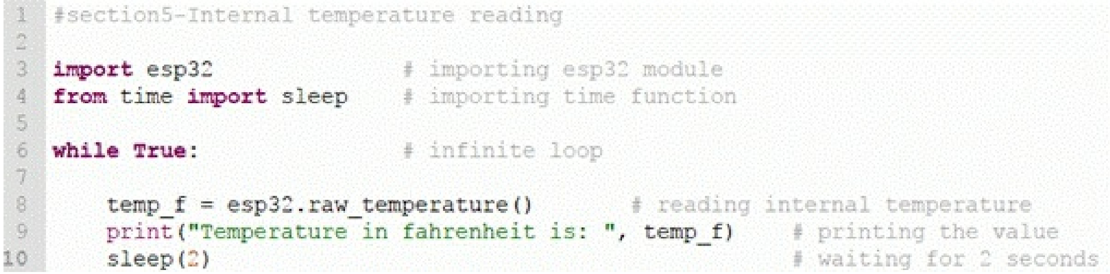


```
Humidity is : 95
Temperature is : 34
Humidity is : 95
Temperature is : 33
Humidity is : 95
Temperature is : 33
Humidity is : 95
```

5.6 ESP32 read internal temperature

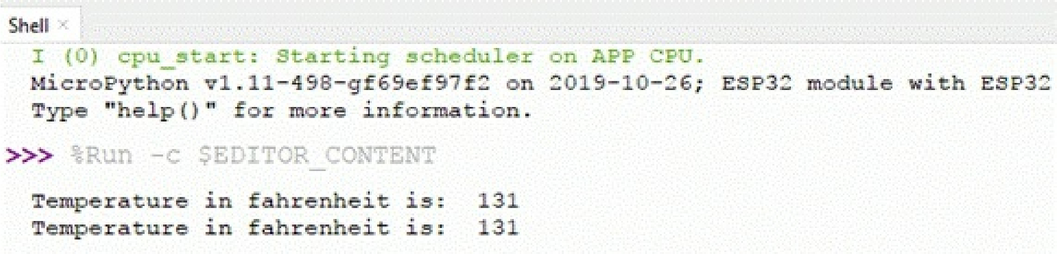
ESP32 comes with an internal temperature sensor. Reading internal temperature in the ESP32 is a good feature.With few lines of code the internal temperature can read. Value will measure in fahrenheit by default. ESP8266 didn't have this feature.

- ➢ 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.



```
#section5-Internal temperature reading

import esp32                # importing esp32 module
from time import sleep      # importing time function

while True:                 # infinite loop

    temp_f = esp32.raw_temperature()        # reading internal temperature
    print("Temperature in fahrenheit is: ", temp_f)     # printing the value
    sleep(2)                                            # waiting for 2 seconds
```

- ➢ Output in the REPL



```
 I (0) cpu_start: Starting scheduler on APP CPU.
 MicroPython v1.11-498-gf69ef97f2 on 2019-10-26; ESP32 module with ESP32
 Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

 Temperature in fahrenheit is:  131
 Temperature in fahrenheit is:  131
```

5.7 ESP32 read internal hall effect sensor

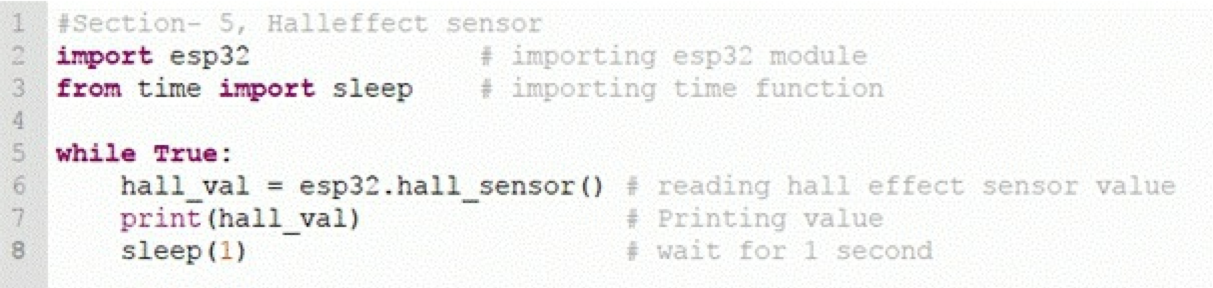
Another ESP32 feature is that it has an internal hall effect sensor.Following method can be used for a magnet based switch, based on your idea it can be integrated to your project.

➢ Import ESP32 module.

➢ Import sleep function from module.

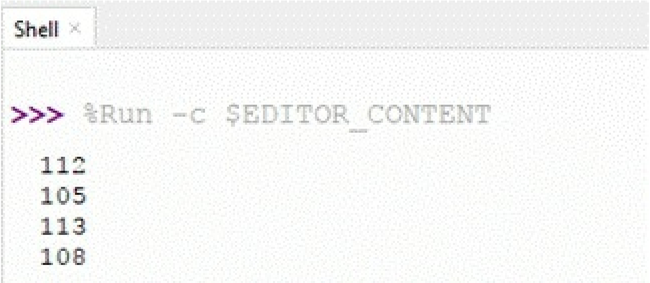
➢ Read directly the temperature by using the command **esp32.hall_sensor()**

➢ While loop is used to display the raw temperature continuously.



```
#Section- 5, Halleffect sensor
import esp32                # importing esp32 module
from time import sleep      # importing time function

while True:
    hall_val = esp32.hall_sensor() # reading hall effect sensor value
    print(hall_val)                # Printing value
    sleep(1)                       # wait for 1 second
```

➢ Output in theREPL



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

  112
  105
  113
  108
```

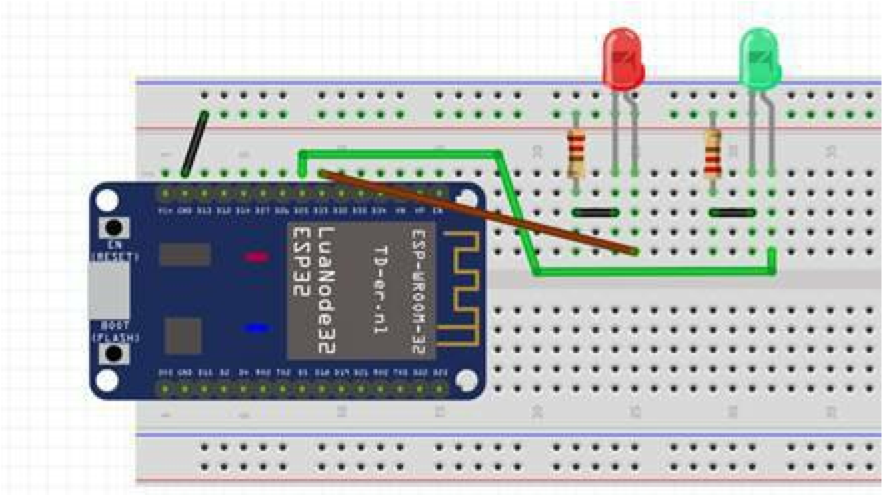
Keep a magnet very close to the ESP32 module (top of the metal shield) then the value will change from above shown value to closer to 25 depending on the strength of the magnet.

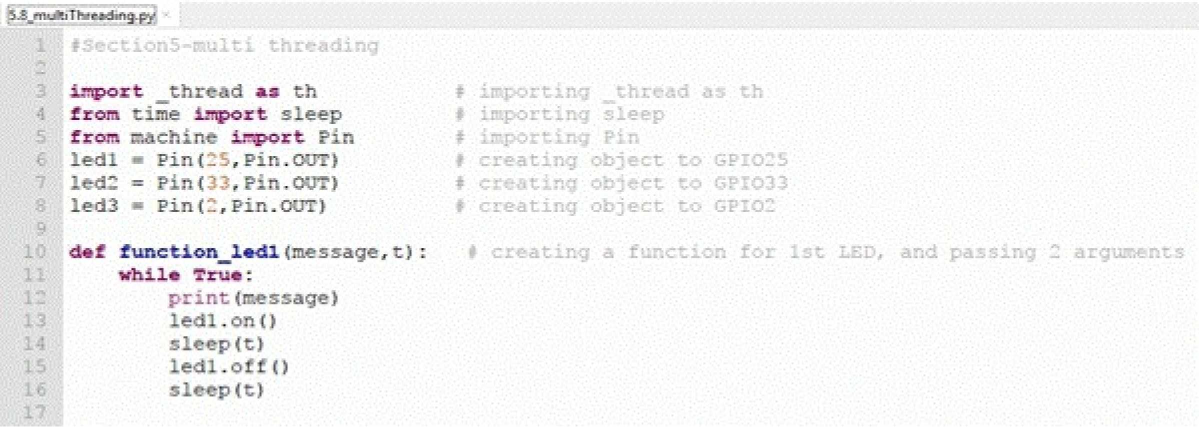
5.8 MultiTreading

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.

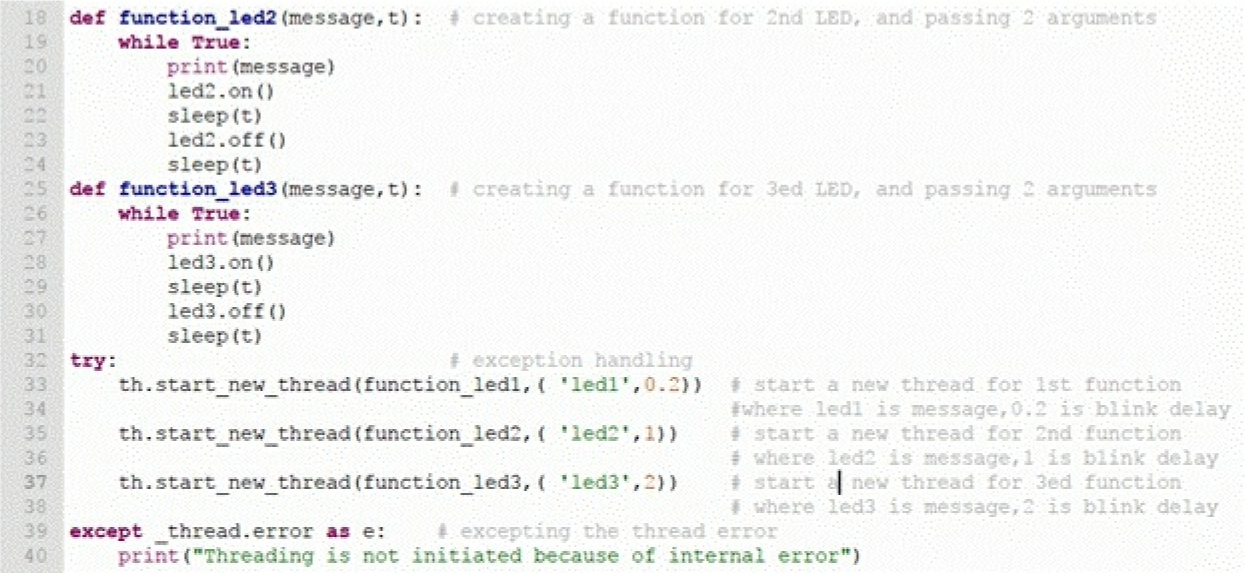
Syntax:

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





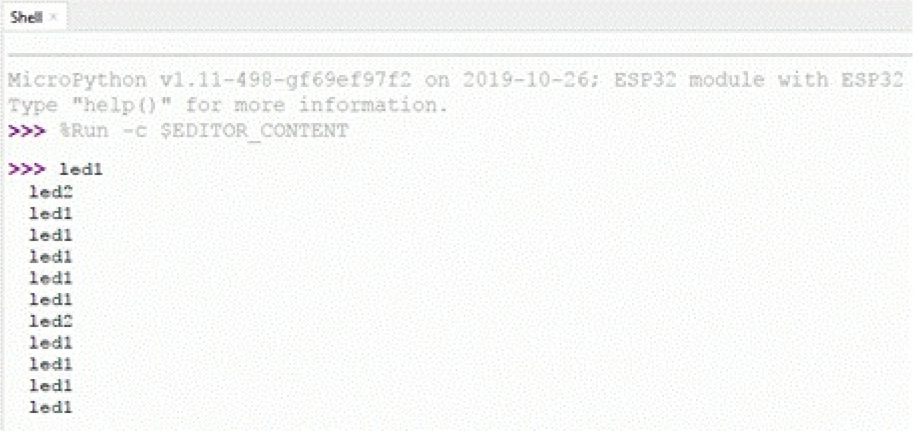
```
#Section5-multi threading

import _thread as th                # importing _thread as th
from time import sleep              # importing sleep
from machine import Pin             # importing Pin
led1 = Pin(25,Pin.OUT)              # creating object to GPIO25
led2 = Pin(33,Pin.OUT)              # creating object to GPIO33
led3 = Pin(2,Pin.OUT)               # creating object to GPIO2

def function_led1(message,t):        # creating a function for 1st LED, and passing 2 arguments
    while True:
        print(message)
        led1.on()
        sleep(t)
        led1.off()
        sleep(t)

```



```
def function_led2(message,t):  # creating a function for 2nd LED, and passing 2 arguments
    while True:
        print(message)
        led2.on()
        sleep(t)
        led2.off()
        sleep(t)
def function_led3(message,t):  # creating a function for 3ed LED, and passing 2 arguments
    while True:
        print(message)
        led3.on()
        sleep(t)
        led3.off()
        sleep(t)
try:                           # exception handling
    th.start_new_thread(function_led1,( 'led1',0.2))  # start a new thread for 1st function
                                                      #where led1 is message,0.2 is blink delay
    th.start_new_thread(function_led2,( 'led2',1))    # start a new thread for 2nd function
                                                      # where led2 is message,1 is blink delay
    th.start_new_thread(function_led3,( 'led3',2))    # start a new thread for 3ed function
                                                      # where led3 is message,2 is blink delay
except _thread.error as e:    # excepting the thread error
    print("Threading is not initiated because of internal error")
```

- ➢ 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)
- ➢ 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.

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



```
MicroPython v1.11-498-gf69ef97f2 on 2019-10-26; ESP32 module with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

>>> led1
  led2
  led1
  led1
  led1
  led1
  led1
  led2
  led1
  led1
  led1
  led1
```

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

Module6- Connect to internet over WiFi

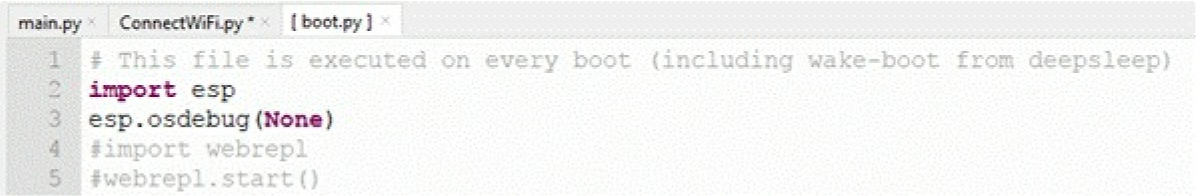
ESP32 and ESP8266 have 2.4 GHz inbuilt WiFi feature. Both chips can work as a **STATION mode** or an **ACCESS POINT mode** or both together. With the WiFi connectivity we can control ESP32 boards from anywhere in the world using appropriate tools like **Blynk App, IFFF** platform or other IoT platforms 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:

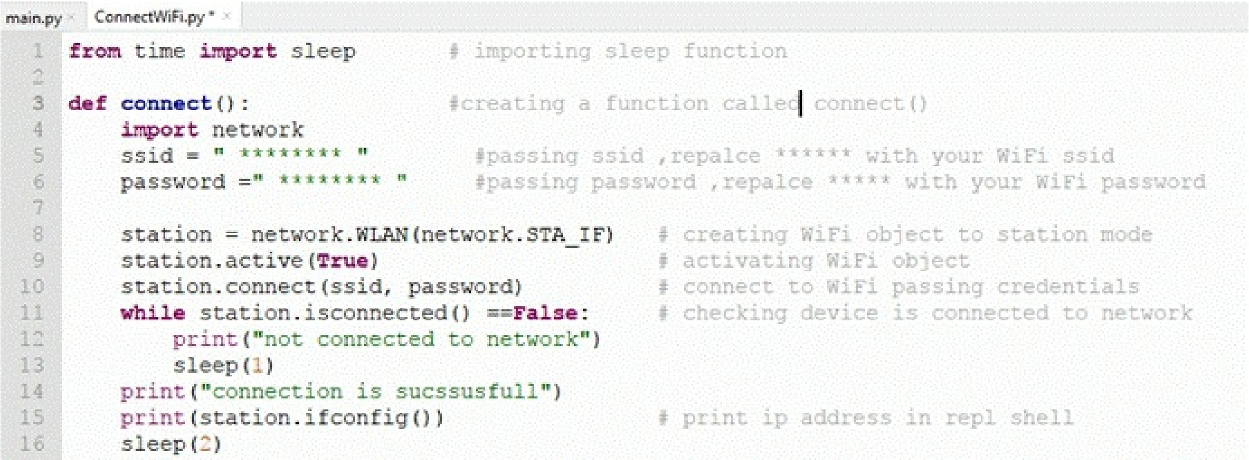
➢ Open **boot.py** file in the ESP32 and uncomment for **import esp** and **esp.osdebug(None)**



```
# This file is executed on every boot (including wake-boot from deepsleep)
import esp
esp.osdebug(None)
#import webrepl
#webrepl.start()
```

Step2:

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



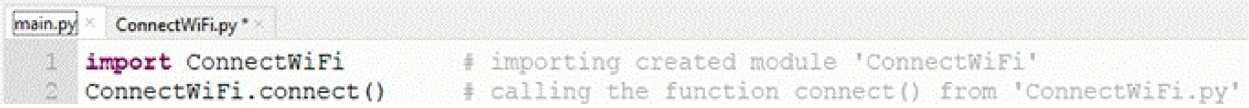
```
from time import sleep          # importing sleep function

def connect():                  #creating a function called connect()
    import network
    ssid = " ******** "           #passing ssid ,repalce ****** with your WiFi ssid
    password =" ******** "        #passing password ,repalce ***** with your WiFi password

    station = network.WLAN(network.STA_IF)    # creating WiFi object to station mode
    station.active(True)                      # activating WiFi object
    station.connect(ssid, password)           # connect to WiFi passing credentials
    while station.isconnected() ==False:      # checking device is connected to network
        print("not connected to network")
        sleep(1)
    print("connection is sucssusfull")
    print(station.ifconfig())                 # print ip address in repl shell
    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
- 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:

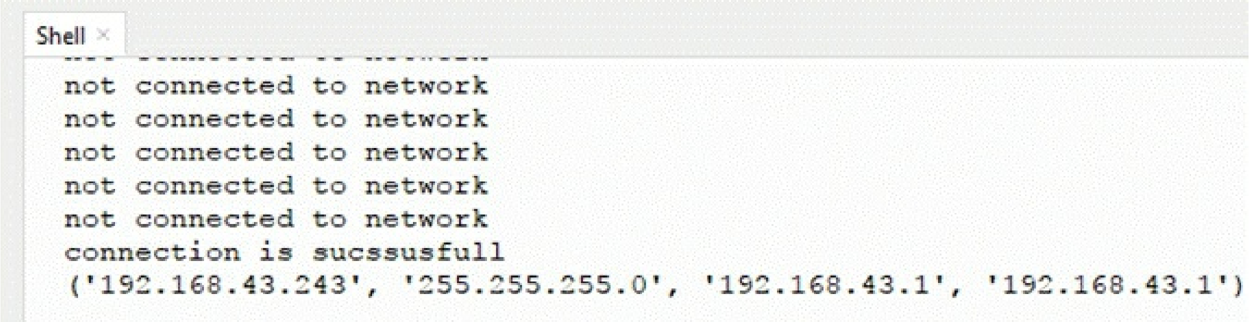
- Create a new file called **main.py** and save to MicroPython device(ESP32)
- Import the previously created file called **ConnectWiFi.py** to **main.py** using the command **import ConnectWiFi**
- Call the function **connect()** from ConnectWiFi using the command **ConnectWiFi.connect()**



```
import ConnectWiFi          # importing created module 'ConnectWiFi'
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.

- 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



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

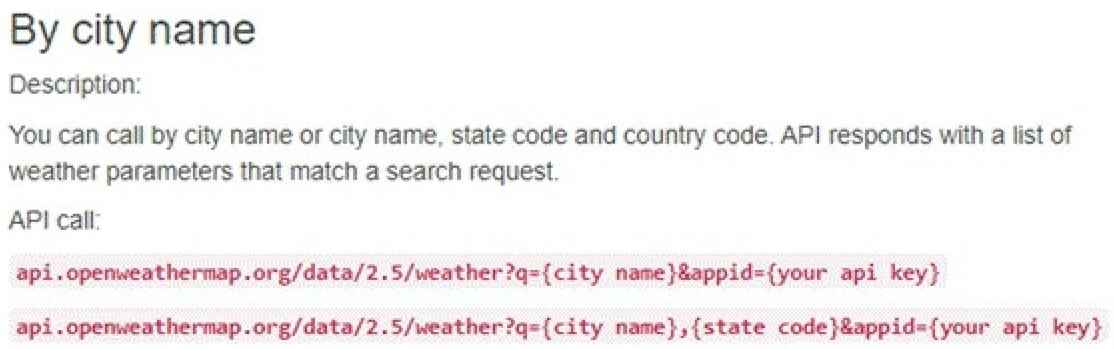
- Where **192.168.43.243** is the local ip address. It may be different in your case.

6.2 Get weather data from OpenWeatherMap using API

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



By city name

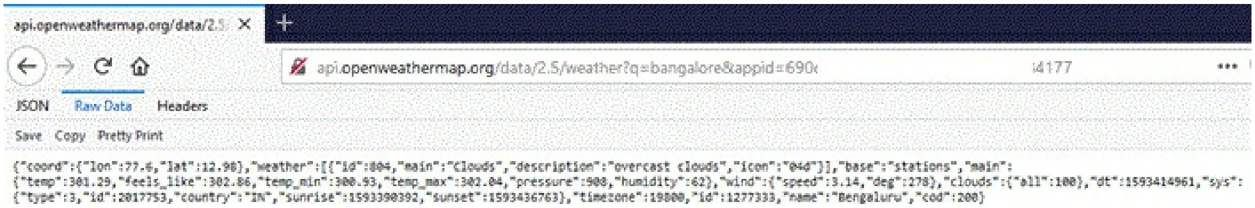
Description:

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.

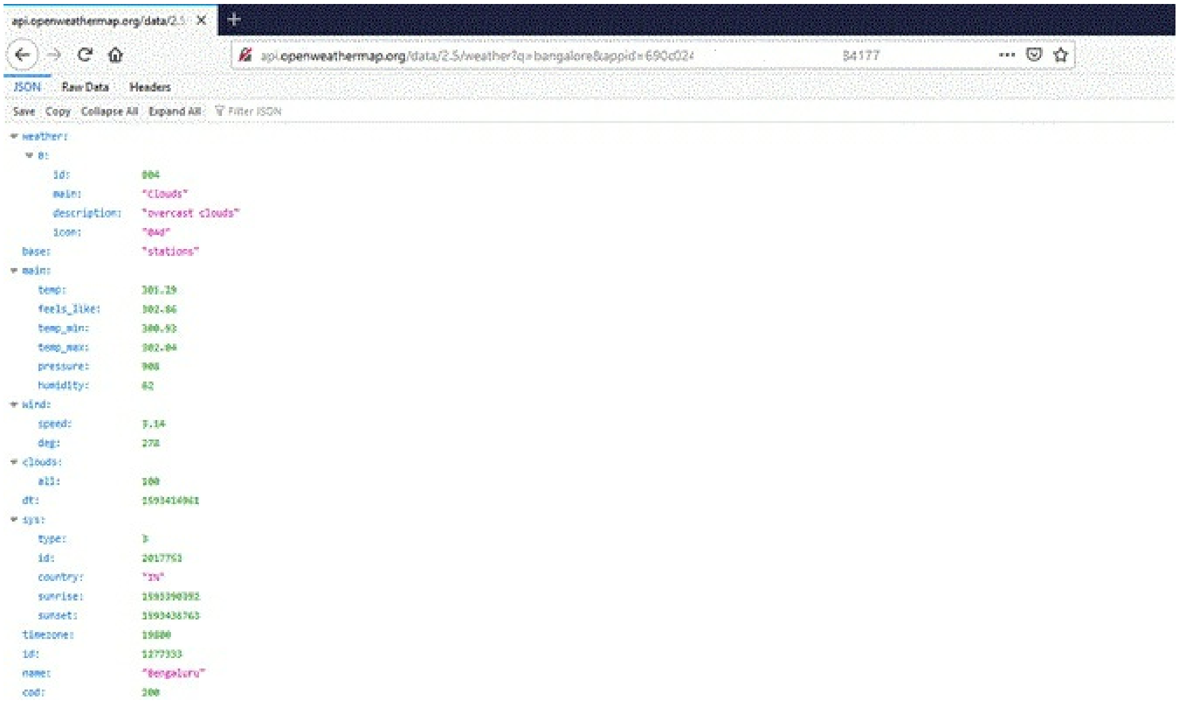
API call:

```
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**


api.openweathermap.org/data/2.5
+
api.openweathermap.org/data/2.5/weather?q=bangalore&appid=690(4177
JSON
Raw Data
Headers
Save
Copy
Pretty Print

```
{"coord":{"lon":77.6,"lat":12.98},"weather":[{"id":804,"main":"Clouds","description":"overcast clouds","icon":"04d"}],"base":"stations","main":
{"temp":301.29,"feels_like":302.86,"temp_min":300.93,"temp_max":302.04,"pressure":908,"humidity":62},"wind":{"speed":3.14,"deg":278},"clouds":{"all":100},"dt":1593414961,"sys":
{"type":3,"id":2017753,"country":"IN","sunrise":1593390392,"sunset":1593436763},"timezone":19800,"id":1277333,"name":"Bengaluru","cod":200}
```


api.openweathermap.org/data/2.5
+
api.openweathermap.org/data/2.5/weather?q=bangalore&appid=690c024 34177
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON

```
▾ weather:
  ▾ 0:
      id:            804
      main:          "Clouds"
      description:   "overcast clouds"
      icon:          "04d"
  base:              "stations"
▾ main:
    temp:            301.29
    feels_like:      302.86
    temp_min:        300.93
    temp_max:        302.04
    pressure:        908
    humidity:        62
▾ wind:
    speed:           3.14
    deg:             278
▾ clouds:
    all:             100
  dt:                1593414961
▾ sys:
    type:            3
    id:              2017753
    country:         "IN"
    sunrise:         1593390392
    sunset:          1593436763
  timezone:          19800
  id:                1277333
  name:              "Bengaluru"
  cod:               200
```

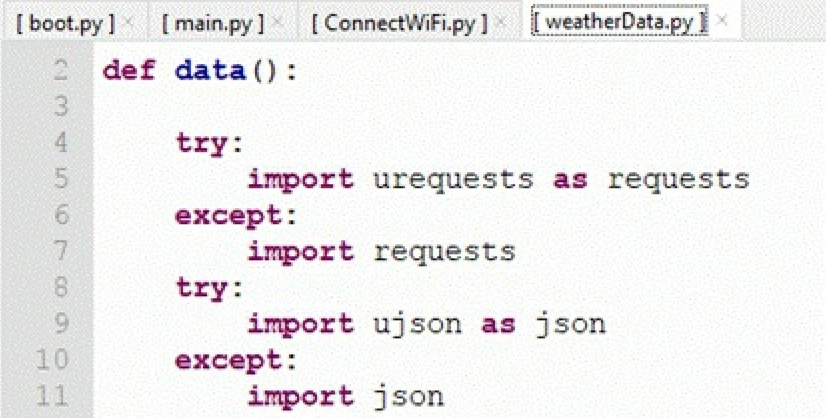
- ➢ Example like temperature, humidity and pressure comes under **‘main’**
- ➢ City name is under **‘name’**

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’)**

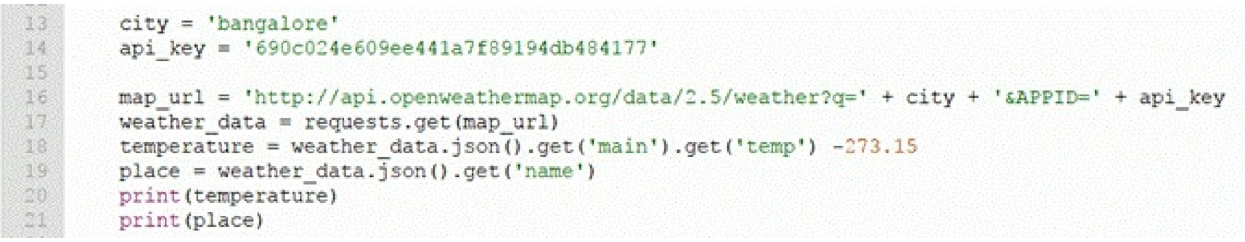
Step3:

- ➢ Follow the code



```
def data():

    try:
        import urequests as requests
    except:
        import requests
    try:
        import ujson as json
    except:
        import json
```

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

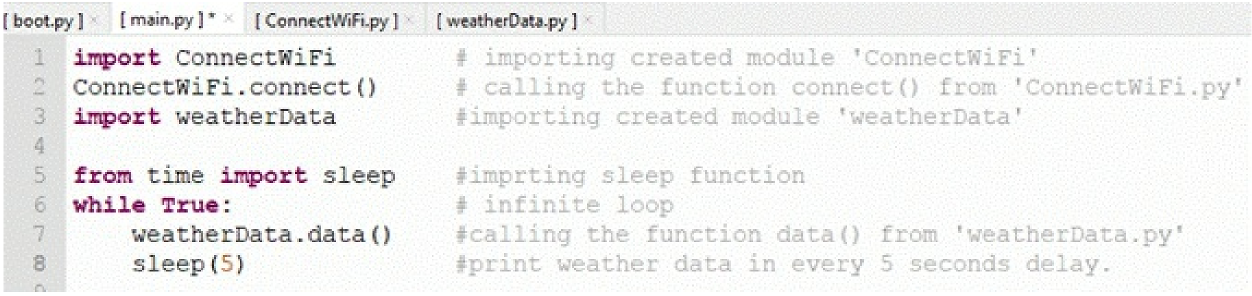


```
    city = 'bangalore'
    api_key = '690c024e609ee441a7f89194db484177'

    map_url = 'http://api.openweathermap.org/data/2.5/weather?q=' + city + '&APPID=' + api_key
    weather_data = requests.get(map_url)
    temperature = weather_data.json().get('main').get('temp') -273.15
    place = weather_data.json().get('name')
    print(temperature)
    print(place)
```

- ➢ 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 **OpenWheatherMap** 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

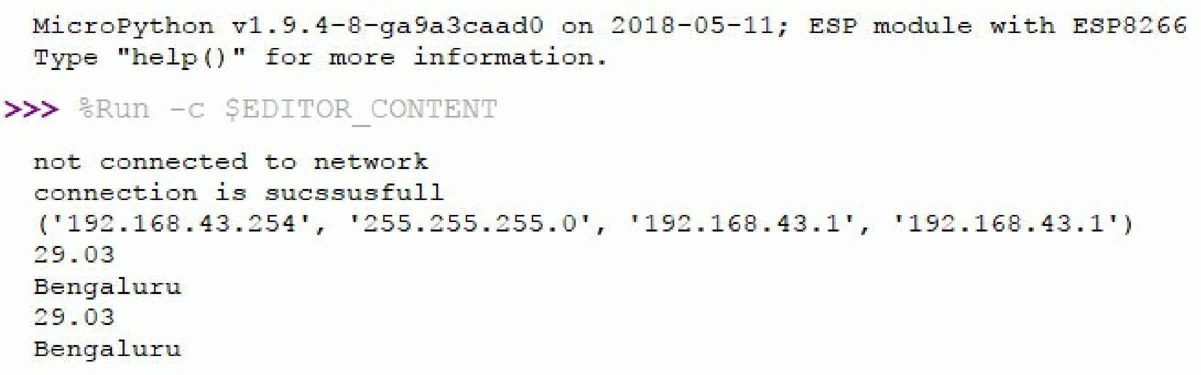
Step4:

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



```
import ConnectWiFi          # importing created module 'ConnectWiFi'
ConnectWiFi.connect()       # calling the function connect() from 'ConnectWiFi.py'
import weatherData          #importing created module 'weatherData'

from time import sleep      #imprting sleep function
while True:                 # infinite loop
    weatherData.data()      #calling the function data() from 'weatherData.py'
    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 v1.9.4-8-ga9a3caad0 on 2018-05-11; ESP module with ESP8266
Type "help()" for more information.

>>> %Run -c $EDITOR_CONTENT

not connected to network
connection is sucssusfull
('192.168.43.254', '255.255.255.0', '192.168.43.1', '192.168.43.1')
29.03
Bengaluru
29.03
Bengaluru
```

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. Expected total cost of the mentioned components is 15 dollars.

Resources

Installing python3	**:**	**https://youtu.be/YN3jyNT2ADU**
Installing Thonny IDE	**:**	**https://youtu.be/XwrAGE2NA2M**
Flashing ESP32	**:**	**https://youtu.be/0mmgo2CHppk**
Help function	**:**	**https://youtu.be/3SLnaMakz6o**
Blink inbuilt LED	**:**	**https://youtu.be/AmIPWqRWnAI**
Auto connect to WiFi	**:**	**https://youtu.be/ak298OC-rv0**
Website	**:**	**https://umydo.com/**