

## Getting Started with Arduino, IOT and Micropython for Beginners

*Prepared by*  
**Mr. Nikhil Mohan Shikare**  
(UG Student of INFT Department)

**Under guidance/mentorship of:**  
Prof. Kanchan Dhuri, Prof. Ajitkumar Khachane and  
Department of Information Technology  
Vidyalankar Institute of Technology, Wadala.

for any **Correction/Suggestion/Feedback**,

Email: - [nikhil.shikare@vit.edu.in](mailto:nikhil.shikare@vit.edu.in) / [nikhilshikare765@gmail.com](mailto:nikhilshikare765@gmail.com)  
Linkdin: - <https://www.linkedin.com/in/nikhil-shikare-18127a216/>

### Professors' Contact Details

**Prof. Kanchan Dhuri:** - [kanchan.dhuri@vit.edu.in](mailto:kanchan.dhuri@vit.edu.in)  
**Prof. Ajitkumar Khachane:** - [ajitkumar.khachane@vit.edu.in](mailto:ajitkumar.khachane@vit.edu.in)

## Table of Content

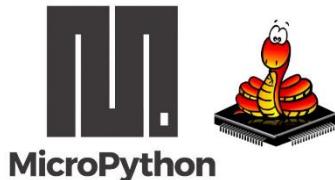
Sr.no	<u>Description</u>	<u>Page no.</u>
1	Commonly Used Component with Their Values	5
2	Arduino-Uno Board Introduction	6
3	Types Of Sensors	7
4	Introduction To Arduino IDE	12
5	Basic Code Structure, Pin Mode, And Serial	14
6	Led Brightness PWM Control Demonstration	15
7	Interfacing Lcd and Types of Lcd	16
8	Different Types of Boards Series	19
9	How To Download Libraries for Different Board Series & GitHub Link's	20
10	Wi-Fi Connectivity For (Wi-Fi-Module- Esp-01) And NodeMcu	22
11	What is Thing Speak? and Setup.	23
12	Interface Sensor, display, and store on cloud Like "Things Speak"	25
13	Communication Protocols (For Board-To-Board and module Comm.)	26
	<-----Additional / Intermediate Lavel Add On ----->	
14	OLED I2C Display and Library's (Adafruit SH110X & SSD1306)	30
15	<i>Confused About <u>UART Rx And Tx Pins</u> Connection?? I.E. PC To Module Vs Board To Board/Module Communication</i>	33
16	What Is "AT" Commands?? And	35
17	Operating WI-FI Module (HTTP Request) Using "AT" Commands	36
18	Making HTTP call using NodeMcu/ESP8266	38
19	Common Errors and Troubleshooting. <b>(BEFORE HANDS-ON, refer this chapter and below chapter)</b>	42
20	Common mistake that Damage Boards/Module. <b>(BEFORE HANDS-ON, refer this chapter and above chapter)</b>	46
21	Voltage Converters modules and IC	47



## Micropython Basic Intro

(We will use esp8266 and esp32 board for practicals and reference.)

Sr.no	<u>Description</u>	<u>Page no.</u>
22	Downloading CP210X Driver for NodeMcu (Refer from chapter 18)	38
23	Exploring Python Variants	51
24	Downloading Micro-python Firmware for NodeMcu	52
25	Flashing NodeMcu using NodeMcu-flasher	53
26	Flashing NodeMcu using Thonny IDE	54
27	REPL and Micropython interaction via Shell	57
28	Installing Micropython external library's	59
29	Installing esp32-cam firmware	60
30	Syntax Comparison - Arduino C And Micropython	62
31	Esp-32 Features and Info	63
32	ESP32 – Various Versions	64
33	ESP32 cam QR scanner – Project 1 (Camera, Wi-Fi, and Socket)	65
34	ESP32 / ESP8266 interfacing DHT Sensor – Project 2 (GPIO)	69
35	Esp32 / ESP8266 implementing MQTT for Above – Project 3 (MQTT)	71
36	Esp32 / ESP8266 interfacing OLED display and Pot – Project 4 (I2C)	74
37	ESP32 / ESP8266 interfacing GSM Module – Project 5 (UART)	80
38	ESP32 / ESP8266 GPS Weather Station – Project 6 (Wi-Fi, API & HTTP)	83
39	Sample projects for reverence	101
40	Online References links	102



**Note:** To keep the document concise, I have included links to the official documentation for further reference. <https://docs.micropython.org/en/latest/>

**Note:** - This document covers most of the detailed information required to start with IoT. Don't worry if the document seems confusing at first; if you follow each step from the respective chapters, most things will work. Screenshots with circles and rectangles highlight important information. For detailed information, reference links to official websites are provided.

While learning, you may face software or hardware errors—it's common. Don't stop; there are multiple resources like [Stack Overflow](#), Arduino forums, Arduino support, YouTube channels, and many other websites to resolve your issues. The key is to learn by trial and error.

Apart from this, I have observed that many prestigious publishers' books on IoT and IOE are more “theoretical” and “exam-oriented”. They include content in question answer format and advanced information which typically expected by candidate to write in exams. However, this guide focuses primarily on hands-on/practical aspects for IoT/IOE projects.

For mini-projects in IOT/IOE subjects, appropriate information/guides about different microcontrollers, sensors, and software resources are often lacking, especially for IT/CS students. Some, students are unaware of the different types of boards, use cases, etc., such as the ESP series, Raspberry Pi series, and Mega, Uno, Nano variants, and dual-core microcontrollers like [ESP32](#) and [Pico w.](#)

This guide aims to minimize these issues and focus on introducing these concepts practically and hands-on. Most of the modules, software, steps, and tasks mentioned in this document have been performed and are provided with step-by-step guides. This guide is written for beginner to intermediate levels, keeping things simple.

Some theoretical knowledge about protocols, hardware/software, and basic electrical and electronic components is required before hands-on activities, so necessary theory is included in each chapter.

All the best! . I hope this guide is useful to you...

- Nikhil Mohan Shikare

Please feel free for any **Correction/Suggestion/Feedback**.

Photos courtesy - **Internet** (Images taken from different online platforms and sources).

**Correction/ Suggestion:** -

Email: - [nikhil.shikare@vit.edu.in](mailto:nikhil.shikare@vit.edu.in) and [nikhilshikare765@gmail.com](mailto:nikhilshikare765@gmail.com)

LinkedIn: - <https://www.linkedin.com/in/nikhil-shikare-18127a216/>

# 1) Commonly Used Component with Their Values

(we will see, most commonly used component values with use cases)

- 1) **Resistors:** - mainly used when connecting LED and pull-up resistor in some sensors.

- For LED: - we use **220-ohm** resistor
- For **pull up resistor** in sensor like DS18B20, we use **4.7K Ohm resistor**
- Resistor **10k pot** for controlling brightness and other uses.



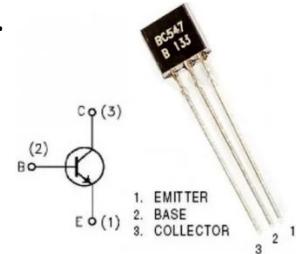
- **2.2k ohm** is also common observed value.

- 2) **Diodes and LED:** - **1N4001** and series used in rectification circuit and LED buzzer as output.



In most of LED  
bigger probe is  
**+VE** and smaller  
one is **-VE**

- 3) **BC547 transistor**- BC547 is usually used for **current amplifier**, **quick switching** and **pulse-width modulation (PWM)** and **control the speed of a motor**.



- 4) **5v dc Relay Module:** - Used for **trigger action**, more used in projects like HOME automation, pump on off. Typically involved AC voltage so be careful and don't Handle when plugged in **AC Power**. (GPIO pins **low voltage to trigger high voltage**)

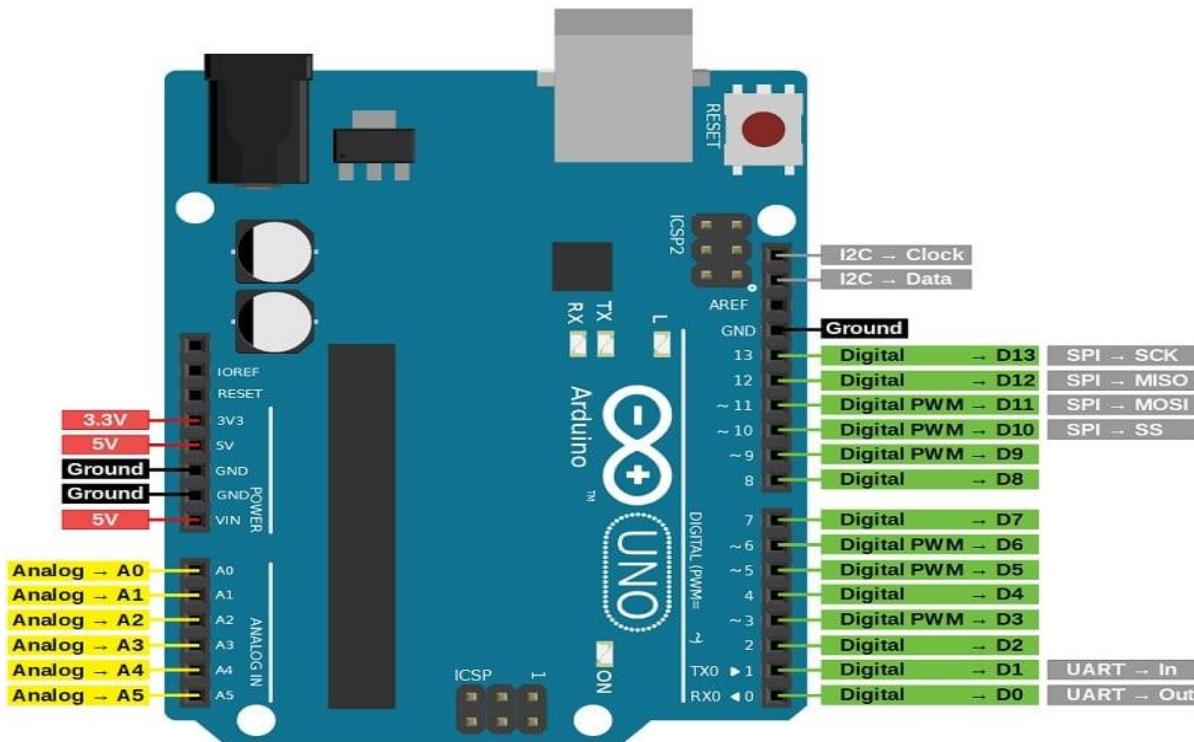


- 5) **Multi-Meter and Clamp-Meter:** - Generally used for **reading voltage, current, checking transistors, CONTINUITY of circuit, reading resistor value etc.**



## 2) Arduino-Uno Board Introduction

- **Why Arduino -Uno – beginner friendly,** as it consists different voltage level (3.3v and 5v), onboard voltage management, wide range of different type of pins and large community to help you when you stuck which make it simpler for beginner to start with it.
- **Pin Configuration: -**

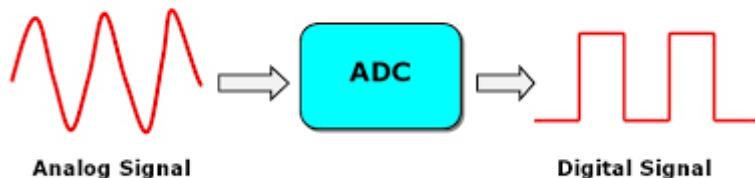


- **Common Thing's: -** You will find **Voltage pins 3.3v/5v, Reset Pin** and (Some have reset Button like RED Button shown), **GPIO and RX/TX Pins** in Almost Every Microcontroller series.
- **Power: -** For Power We Have Barrel Jack Connector (7-12v), USB power (5v) and Other Pin VIN (**Not Recommended**)
- **Specification Table: -**

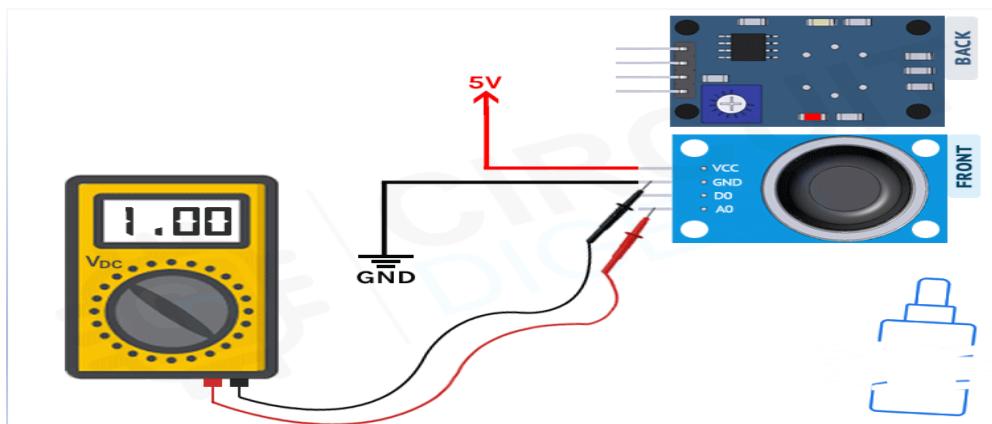
Microcontroller	ATmega328
Clock Speed	16MHz
Operating Voltage	5V
Maximum supply Voltage (not recommended)	20V
Supply Voltage (recommended)	7-12V
Analog Input Pins	6
Digital Input/Output Pins	14
DC Current per Input/Output Pin	40mA
DC Current in 3.3V Pin	50mA
SRAM	2KB
EEPROM	1KB
Flash Memory	32KB of which 0.5KB used by boot loader

### 3) Types of Sensors (Analog and Digital)

- **Analog:** - 1] Produces Output in Form of Voltage (Analog). as mentioned, Arduino have 6 Analog pins A0-A6. Each Read the **Voltage Between 0v to 5v**. **Raw Voltage** Cannot be measured by UNO, instead UNO board have **10bit ADC** (Analog to Digital Converter). So possible values are  $2^{10} = 1024$ . **Including 0** it would be **1023**. (0-1023).



2] This Means **Reading value Taken By Analog Sensor on Uno Board is ranged from 0 – 1023 (not less than 0 not greater than 1023)**. Refer Below Diagram Measurement of Sensor Voltage (Analog) and this voltage Is Converted to digital values by using ADC



3] **But wait**, if you are beginner, it is fine even you don't understand above mechanism, Lets Directly Jump to Analog Read function in Arduino IDE.

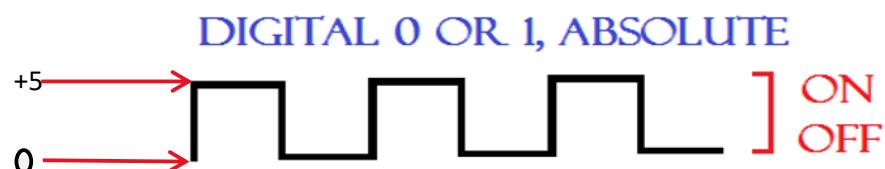
1) `analogRead ()`:-

```
Int val = analogRead(analogPin); // read the input pin
Serial.println(val);           // Print Stored value in variable val
```

2) `analogWrite():-` `analogWrite(pin, value)`

```
analogWrite(PIN, 185); // analogWrite values from 0 to 255 to
respective pin
```

- **Digital:** - 1] Only **Two Values** =+5v Indicate 1(High) and 0v indicate 0(Low) logic level. For Arduino Pin Voltage is +5v max so 5 is used to show high, but in boards like NodeMcu the Voltage used Is 3.3v for represent logic 1(High).



2] Digital Pin use in Arduino: -

1) **digitalRead()**: - Useful When Reading Push button pushed or not , and Reading digital Sensors., Meany more other usages.

```
Int val = digitalRead(inPin); // read the input pin
```

2) **digitalWrite()**: - Useful For Led , Buzzer Activation and Meany More...

```
digitalWrite(PIN_NO, HIGH); // sets the digital pin 13 on
delay(1000); // waits for a second
digitalWrite(PIN_NO, LOW); // sets the digital pin 13 off
delay(1000);
```

- **delay()**: - Pause The Program For Specified Times (Milliseconds)

Syntax:- `delay(1000); // waits for a second`

Before We Procced, we shoud Know the Concept of “**Flashing Chip /Microcontroller’s Memory**” i.e. *Uploading code in Arduino/Microcontroller*

 You

what is the word flash means in flash memory or flashing microcontroller chip like esp-01

< 2 / 2 >



**ChatGPT**

In the context of flash memory and microcontrollers, the term "flash" refers to a type of non-volatile memory technology that retains data even when power is turned off. Flash memory is commonly used in electronic devices for storing firmware, software, and other data that needs to persist through power cycles.

When people talk about "flashing" a microcontroller chip, they are referring to the process of programming or writing data to the flash memory of the microcontroller. This process typically involves loading new firmware or software onto the microcontroller, allowing it to perform a specific function or operate a particular device.

For example, in the case of the ESP-01, which is a popular Wi-Fi module based on the ESP8266 microcontroller, flashing refers to the process of loading new firmware onto the ESP-01's flash memory. This firmware could be custom code that enables the ESP-01 to connect to a specific Wi-Fi network and perform specific tasks.

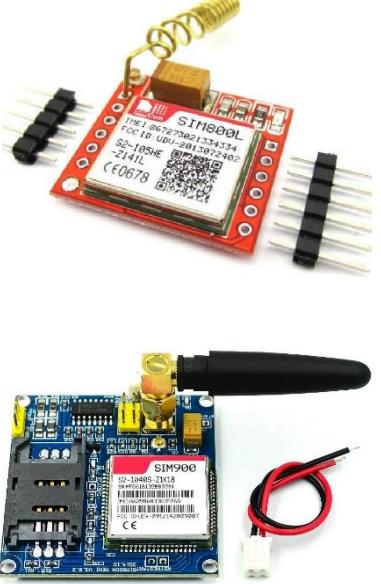
### 3.1) Commonly Used Analog and Digital Sensors

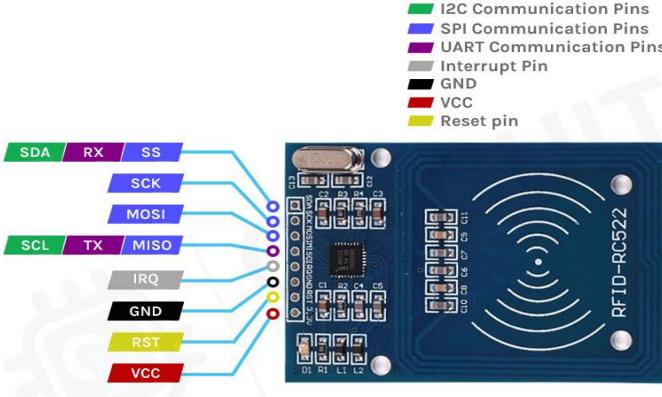
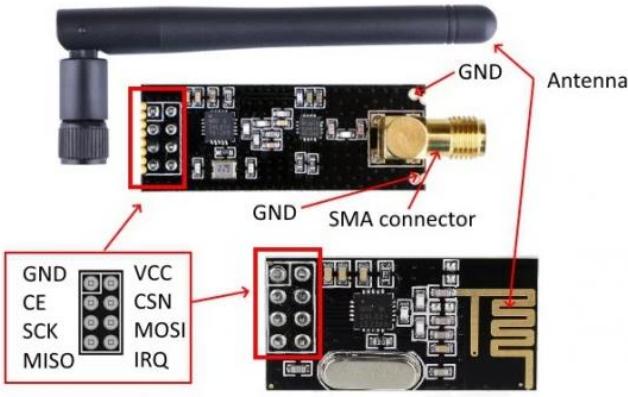


### MQ Series GAS Sensors

 <b>MQ-2</b> Combustible gas,Smoke	 <b>MQ-3</b> Alcohol	 <b>MQ-4</b> Methane,Propane,Butane
 <b>MQ-5</b> Methane,Propane,Butane	 <b>MQ-6</b> liquefied petroleum butane , propane , LPG	 <b>MQ-7</b> Carbon Monoxide
 <b>MQ-8</b> Hydrogen	 <b>MQ-9</b> Carbon monoxide,Methane	 <b>MQ-135</b> Ammonia sulfide,Benzene vapor

### 3.2) Commonly Used Modules With Protocols

<b>Module Photo</b>	<b>Name</b>	<b>Communication Protocol with Arduino/ Microcontroller</b>
	<b>Optical Fingerprint Sensor</b>	<b>UART (RX-TX)</b>
	<b>GSM/GPRS Modules</b>	<b>UART (RX-TX)</b>
	<b>GPS Module</b>	<b>UART (RX-TX)</b>

 <p><b>0.96 inch OLED Color:White</b></p>	<b>OLED Display</b>	<b>SPI / I2C</b>																											
 <p>Legend:</p> <ul style="list-style-type: none"> <li>I2C Communication Pins</li> <li>SPI Communication Pins</li> <li>UART Communication Pins</li> <li>Interrupt Pin</li> <li>GND</li> <li>VCC</li> <li>Reset pin</li> </ul> <table border="1"> <tr> <td>SDA</td> <td>RX</td> <td>SS</td> </tr> <tr> <td>SCK</td> <td></td> <td></td> </tr> <tr> <td>MOSI</td> <td></td> <td></td> </tr> <tr> <td>SCL</td> <td>TX</td> <td>MISO</td> </tr> <tr> <td></td> <td></td> <td></td> </tr> <tr> <td>IRQ</td> <td></td> <td></td> </tr> <tr> <td>GND</td> <td></td> <td></td> </tr> <tr> <td>RST</td> <td></td> <td></td> </tr> <tr> <td>VCC</td> <td></td> <td></td> </tr> </table>	SDA	RX	SS	SCK			MOSI			SCL	TX	MISO				IRQ			GND			RST			VCC			<b>RFID Module</b>	<b>UART / I2C / SPI</b>
SDA	RX	SS																											
SCK																													
MOSI																													
SCL	TX	MISO																											
IRQ																													
GND																													
RST																													
VCC																													
 <p>Legend:</p> <table border="1"> <tr> <td>GND</td> <td>VCC</td> <td>CSN</td> <td>MOSI</td> <td>IRQ</td> </tr> </table>	GND	VCC	CSN	MOSI	IRQ	<b>nRF24L01 2.4GHz transceiver Module  (2.4 GHz ISM Band)</b>	<b>SPI</b>																						
GND	VCC	CSN	MOSI	IRQ																									

For correction/query's: - [nikhil\\_shikare@vit.edu.in](mailto:nikhil_shikare@vit.edu.in) / <https://www.linkedin.com/in/nikhil-shikare-18127a216/>  
[nikhilshikare765@gmail.com](mailto:nikhilshikare765@gmail.com)

**Note: - Refer More details on Common Communication Protocols in Chapter 13**

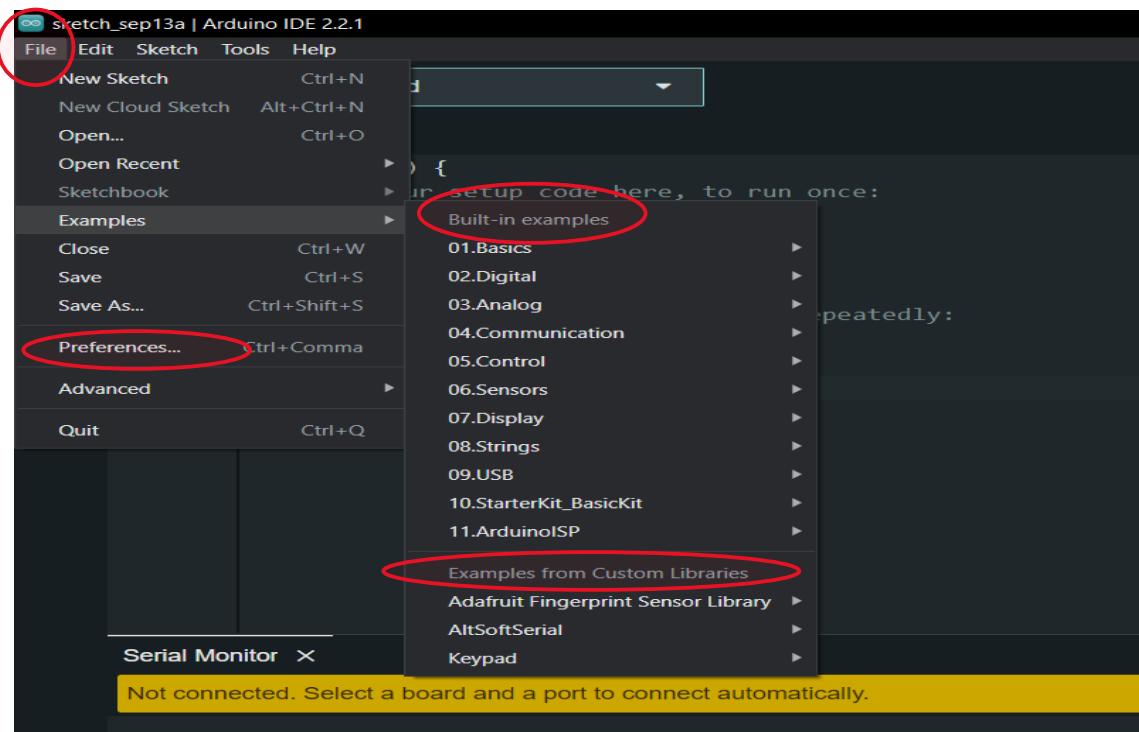
**1) UART: - universal asynchronous receiver / transmitter**

**2) I2C: - Inter-Integrated Circuit**

**3) SPI: - serial peripheral interface**

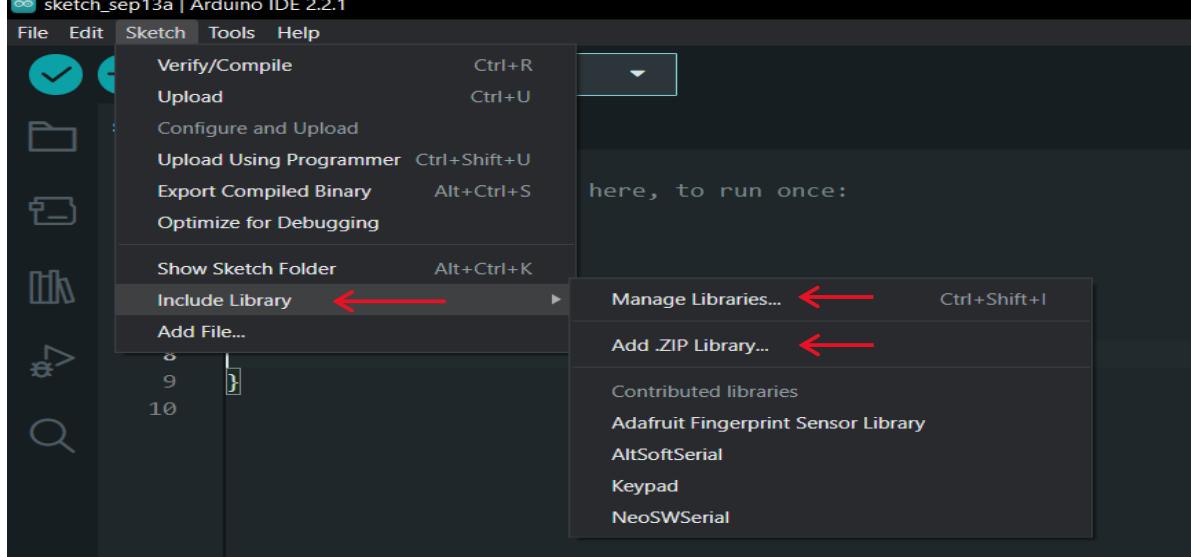
## 4) Intro To Arduino IDE

- Open-Source Arduino IDE Provide Environment for **Coding Debugging, uploading** code on board, it also has **inbuilt and Third-Party Library's** Support. We Can Code It in language similar to **C/C++**. Let's take a Look of **Most used options from IDE**
- Download And Install from Official Website: - <https://www.arduino.cc/en/software>
- **File > Example and file>Preferences** Option:
  - 1) Example:** - Contains Inbuilt example Code and Add on library's Section.
  - 2) Preferences:** - Under preference's Most used option is "**Arduino Board manager URL**", When You Want To install **Add on boards Support Library** like ESP8266, raspberry Series. We need to pest add on URL'S in above option. (Refer **Chapter 9**)
  - 3) Screenshot:** -

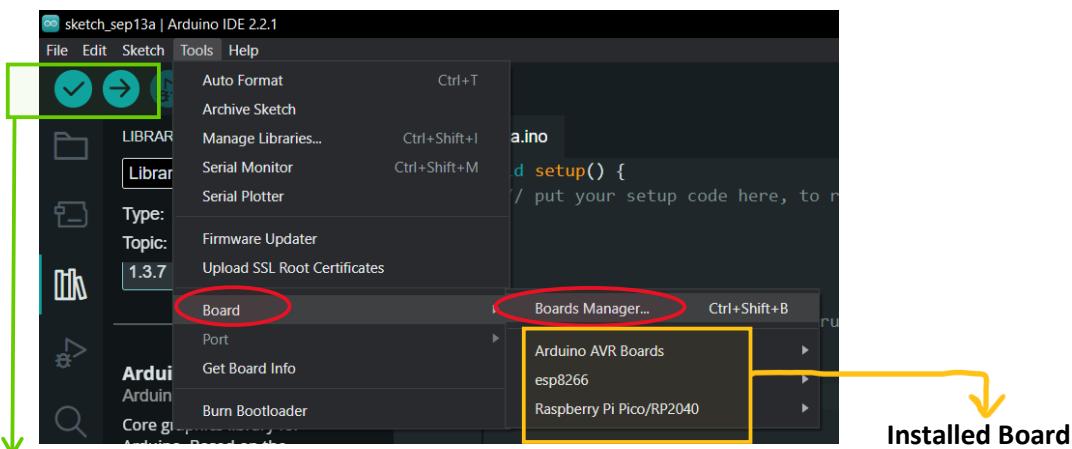


- **Sketch> include Library:** - Used to **Install Library and any 3<sup>rd</sup> party/.zip Library**. You Can Also View Newly Installed Library.

**1) Screenshot:** -



- Tools:** - Most used Option From tool Is Tool>Board. We can view **already installed boards** and can install **new board through board manager**. For some Boards We may need to pest URL'S (index.json) First in file>preferences > “**Arduino Board manager URL**”



- Verify And Upload button:** - Used To compile And Upload the Code to Board
  - Serial Monitor:** - **Mainly Used for Debugging** of Code, Printing Values/data To Serial Monitor When Board Is Connected to Your Pc Via USB.
- Green Box Option** Consist NL (New Line), CR (Carriage Return) and Both Primarily used to send along with Serial input, Similar to “**\n**” and “**\r**” in programming.
- Red box Option** Consist of Baud Rate options i.e., the speed of communications for Serial Communication by **UART** Protocol. **Most common is 9600bps** (bits per second)



## Do you know? - Tinkercad

- If you are beginner and if you don't have any experience about hands on IOT/IOE, I recommend to practice on online platform **Tinker cad** with basic component like, LED, resistor with Arduino,
- It also includes component like, force sensor, distance/ultrasonic sensor, LCD display etc. and meany more.



Tinker cad is an intuitive online platform that enables users to design and simulate 3D models and Arduino-based circuits for Internet of Things (IoT) projects. With a user-friendly interface, it caters to both beginners and educators, providing tools for creating prototypes and experimenting with electronic components virtually. Tinkercad's versatile features make it a valuable resource for learning and prototyping in the fields of 3D modelling and IoT development.

## 5) Basic Code Structure, Pin Mode, And Serial

- **Code Structure:** -

**1) void setup ():** - Put your **code which run once** i.e mainly pinmode define , connection initialization code , Serial.begin(bps) used start serial. Mainly code which run once

**2) void loop ():** - Main code here, to **run repeatedly**.

```
on_sep13aino
1 #define LED 13 //connect LED to digital pin13
2 |
3 void setup() {
4 // initialize the digital pin13 as an output.
5 pinMode(LED, OUTPUT);
6 }
7
8 void loop() {
9 digitalWrite(LED, HIGH); // set the LED on
10 delay(1000); // for 1000ms / 1s
11 digitalWrite(LED, LOW); // set the LED off
12 delay(1000);
13 }
```

**3) #define <variable\_name> <value>** used to define pin, i.e., in above code line 1 tells “**on Arduino uno PIN 13, the LED is connected**”. also used to declare global values, can be used throughout our program.   **Example:** - **#define LED 13 or #define N 10**

**4) pinMode(PIN,I/O):** - For Which Purpose We are Using pin i.e. for taking input or Providing output. In above example we are using **LED pin as OUTPUT** Similarly we have **INPUT, INPUT\_PULLUP, INPUT\_PULLDOWN** options but most used are **OUTPUT** and **INPUT**.

**5) Serial:** - used To Initiate Serial Communication **Serial.begin(9600)** i.e **start Hardware serial** at 9600bps.

- **Serial.print(Value):** - Print Value on Serial Monitor via hardware Serial
- **Serial.println(Values to Print):** - Print on new line in Serial Monitor via hardware Serial
- ❖ **Serial.readString():** - It returns the characters read as a String object
- ❖ **Serial.readStringUntil():** - Reads characters from the serial buffer into a string until a specified character is encountered or a timeout occurs. It returns the characters read as a String object, excluding the specified termination character.

## 6) Led Brightness PWM Control Demonstration

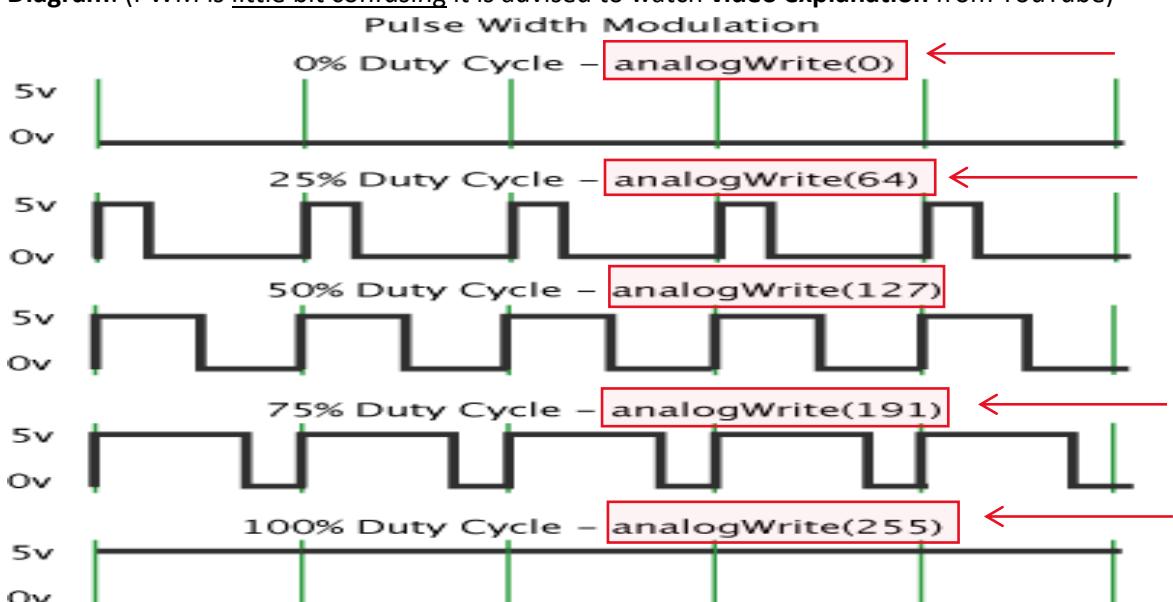
[Now use Above Knowledge in this project]

- What is **PWM**: - Stands for **Pulse Width Modulation**. Simple example to understand PWM is, When we want to control RPM (rotating speed) of motor using 9v battery, you may need variable resistor (potentiometer pot). By adjusting resistance, the RPM is increased or decreased. But main problem is “where is my power which is not used by motor???” , **unfortunately it has been converted into heat by resistance.** **wastage** of power, it plays crucial role in battery powered projects.

So, how to overcome this, simple **control the timing for which the pin is HIGH** i.e if time required for motor to reach at full RPM is 1 second then, only power the motor for 0.5 second and again wait for 0.5 seconds for getting half RPM.in this way the motor RPM can be controlled by varying **on/off Time**. i.e., **controlling the WIDTH of PULSE**.

On Arduino **UNO PWM** pin Is Denoted By **(~)** symbol.

- Diagram:** (PWM is little bit confusing it is advised to watch **video explanation** from YouTube)



- LED Brightness control using PWM:** -(Try Yourself before viewing code)

**Steps**

- Do basic connection
- Declare led pin variable
- Start serial
- Read Analog value from Analog pin
- Use map function to map values in range (0-255)
- Finally write on PWM pin

```
#define led 11
void setup()
{
  pinMode(led, OUTPUT);
  Serial.begin(9600);
}
void loop()
{
  int val = analogRead(A0);
  val = map(val,0,1023,0,255);
  Serial.println(val);
  analogWrite(led,val);
  delay(1);
}
```

**Serial Monitor**

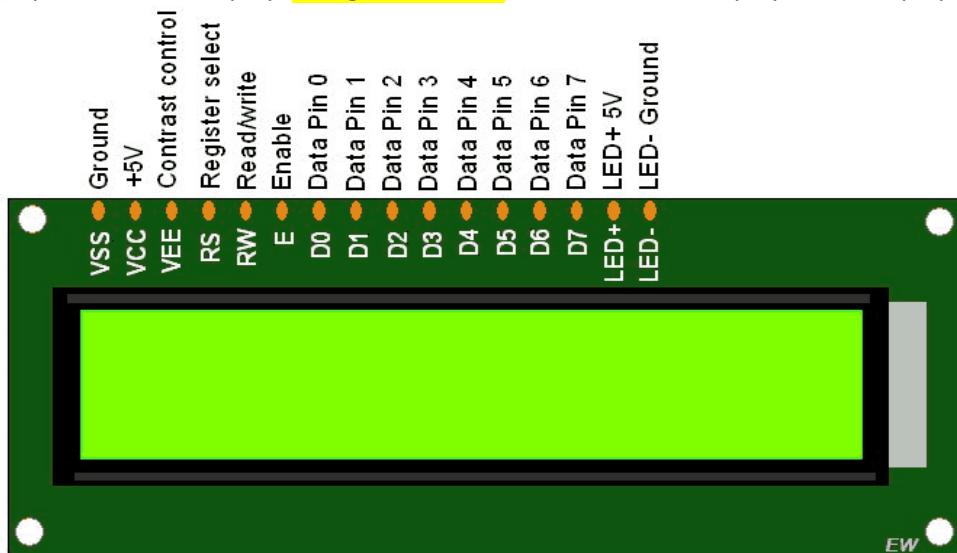
```
29
29
29
29
29
29
29
29
29
29
```

## 7) Interfacing Lcd and Types of Lcd

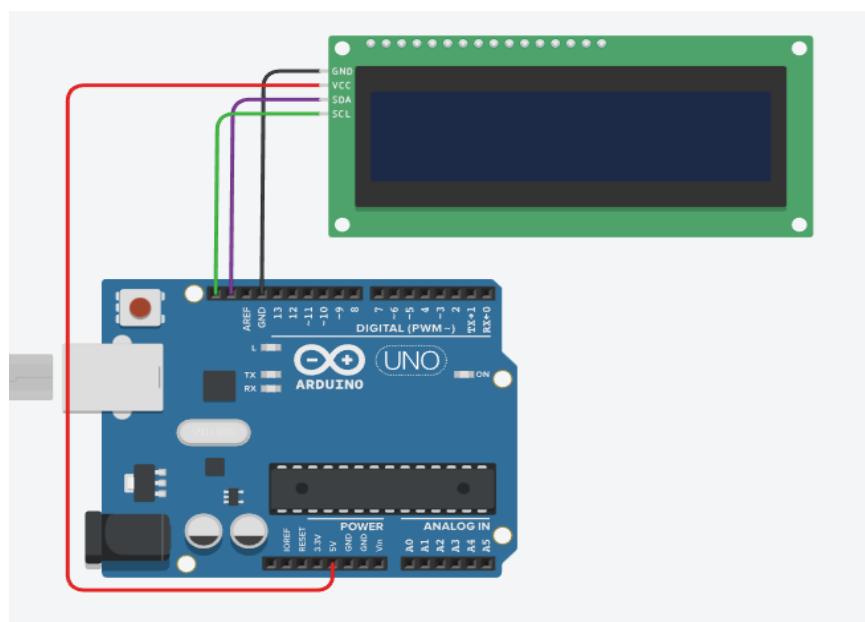
[most used LCD displays]

- LCD 16x2 8-bit and 4-bit: -

**Note:** - 1) if you have enough **GPIO** pins then only go for this type of display, the number of pins in **parallel mode** are high compared to I2C display so, it is little bit complicated for beginners to interface with Arduino. Also require **10k Ohm pot** for controlling contrast.  
 2) Connection and Documentation: - <https://docs.arduino.cc/learn/electronics/lcd-displays> (if you have this display, **Using I2C module** we can use this display as I2C display)

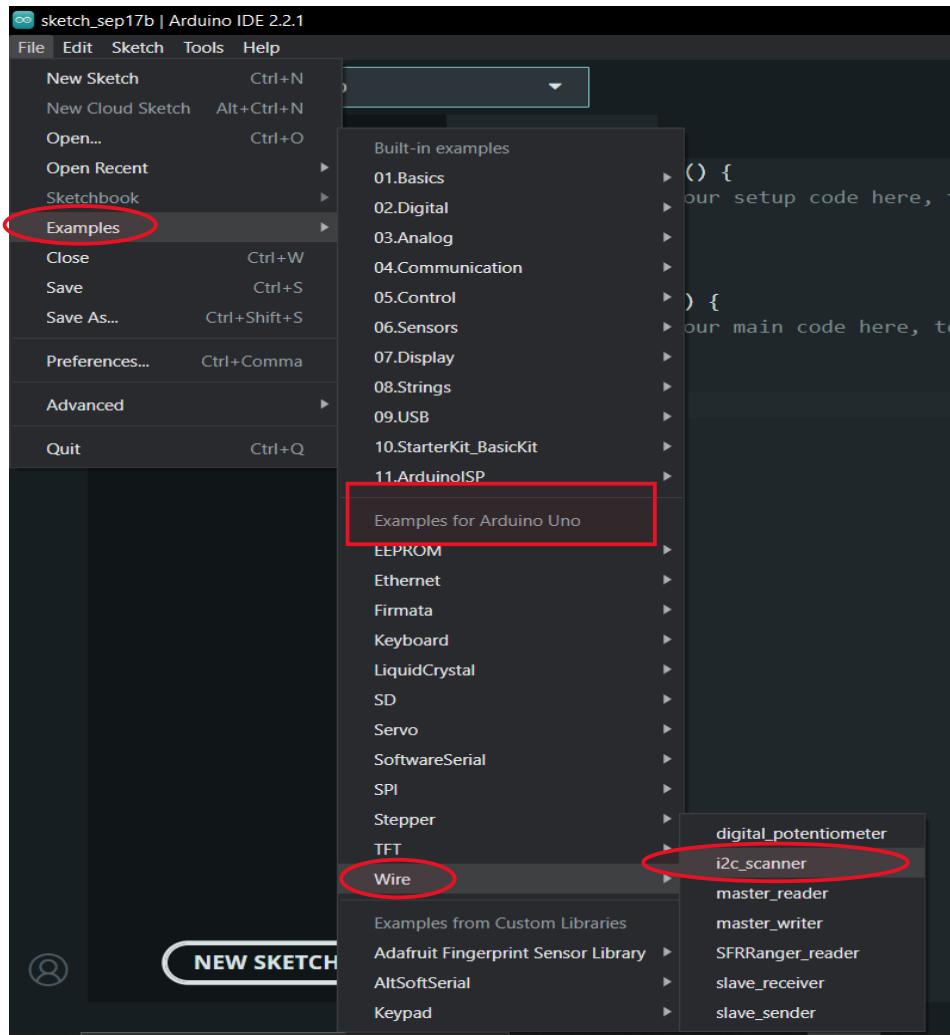


- I2C LCD:** - (we will learn more about communication protocol further)
  - Require Only 4 pins: - VCC (Power), GND, SDA, SCL (I2C pin location depend on board)
  - Uno have one peripheral block for I2C protocol which uses 7 bits addressing, so possible address range is **0x00 to 0x127**, means **theoretically we can connect up to 128 devices** using SDA/SCL pins. each device must have different I2C address.
  - But **Why I2C**, **Reduce the number of wires** to only four, also multiple devices on same line.

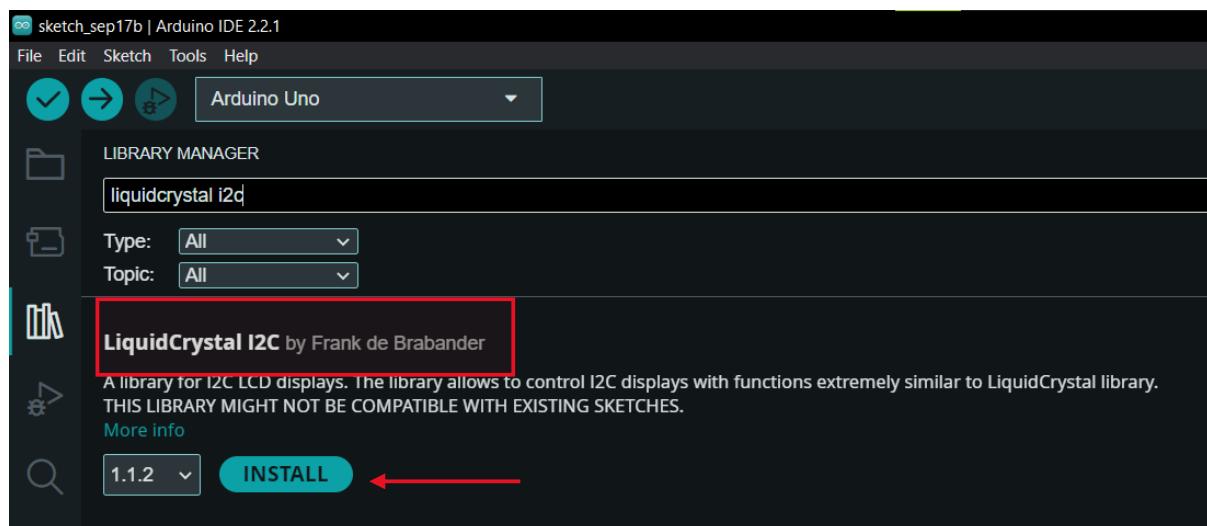


**Steps To Interface I2C 16x2 lcd.**

- Connect SDA/SCL pins as shown in fig. and basic connection like power, gnd.
- For I2C hex address, run below sketch it will display I2C LCD address start with **0x**



- Now in , **Sketch>Include Library>Manage Library** Type/Search for liquidcrystal I2C by **Frank de Brabander**. (you may use different Library depend on LCD Type).



- Now, we will see some basic function to start with LCD. Note that this Function is same for LCD in parallel Mode (1<sup>st</sup> fig) and I2C LCD (2<sup>nd</sup> fig). Also, on Arduino UNO pin A0 is SDA and pin A5 is SCL.

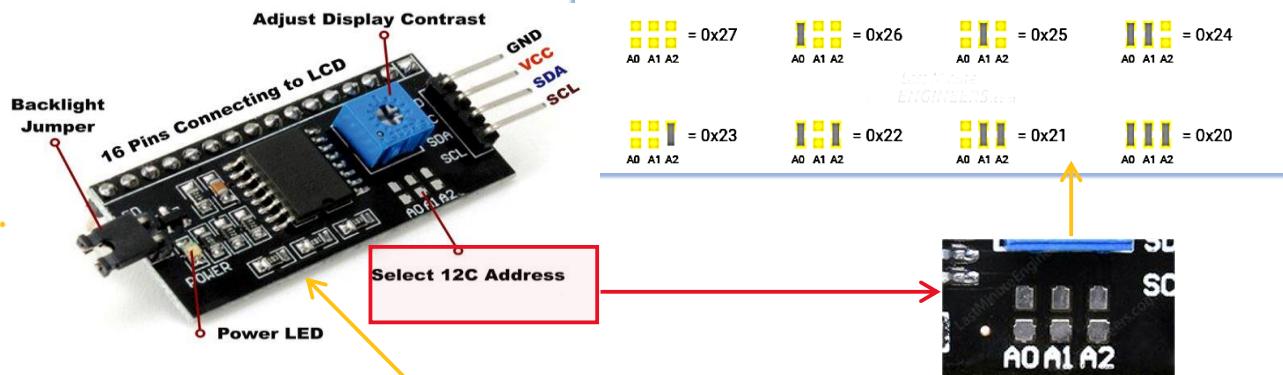
- lcd.clear(); : - Clear Lcd Display
- lcd.print("Hello world!"); :- Print On LCD
- lcd.setCursor(COLUMN,ROW); :- Set Cursor On Specified position
- For I2C lcd Display additional Functions: -
  - lcd.init(); and lcd.backlight(); add in void setup() Function
- Note: -The above functions are basic. for additional functions like blink, scroll do refer documentation

*Now Using Above Knowledge, try to print hello world on Lcd. Search on internet for connection / basic code. try to remember above function*

- Your Task: - 1) Find Different Types of display, OLED, LCD, their size and interfacing method/Protocols (I2C, SPI, parallel). and use cases of this displays
- 2) Also Try to Print Custom Character. (Hint: - use createChar() func. Refer documentation)
- Example Code for I2C: - (Except init and backlight all other function are same as parallel mode 4/8-bit LCD)

```
#include <LiquidCrystal_I2C.h> // Add I2C Lcd library
LiquidCrystal_I2C lcd(0x3F,16,2); // LCD address to 0x3F for a 16 chars and 2 line display
void setup() {
  lcd.init();
  lcd.clear();
  lcd.backlight(); // Make sure backlight is on
  lcd.setCursor(2,0); //Set cursor to character 2 on line 0
  lcd.print("Hello world!");
  lcd.setCursor(2,1); //Move cursor to character 2 on line 1
  lcd.print("LCD Tutorial");
}
```

- Note: - if you have different I2C devices with same address, then we can change I2C LCD Address by short-circuiting. Any two combination in on I2C Module.

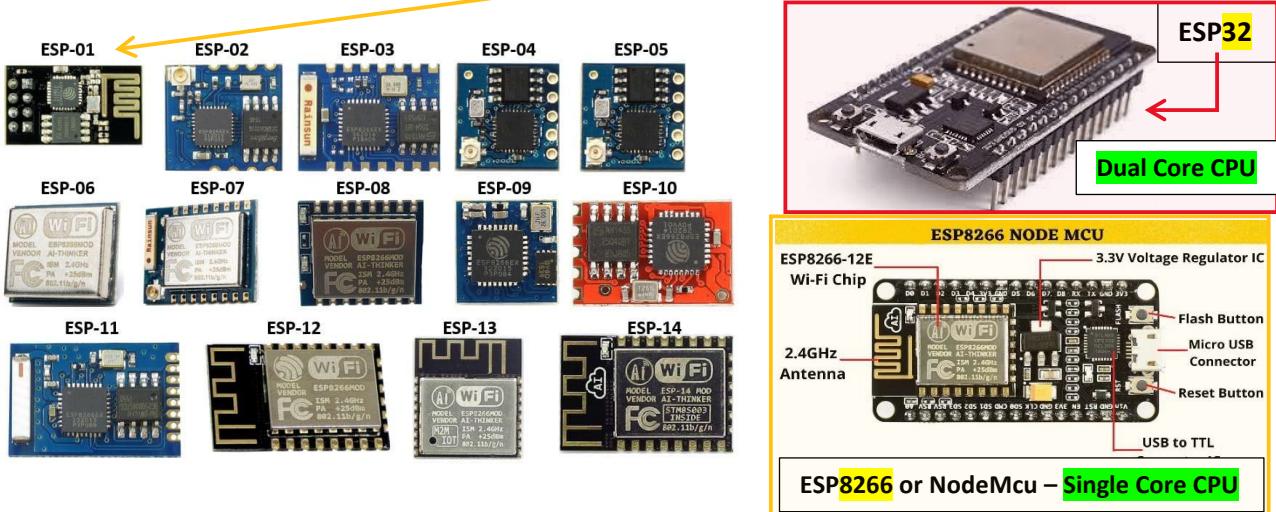


Note That depending upon I2C Module Manufacturer I2C Address May Change. Example  
**1) NXP's PCF8574 chip 2) Texas Instruments' PCF8574 chip**

## 8) Different Types of Boards Series

[Need To install respective board manager] (we will see most common series)

- **ESP8266 and ESP Series:** - Note (ESP-01 also called as Wi-Fi module, it has dual mode)



- **Raspberry Pi Based:** -

Raspberry pi 2/3/4, board have high computing power. high level OS like Linux can be installed. High level Programming Language Like Python is used to control GPIO pins, used to deploy ML Models, Making Smart Drone, Robotics and Stuff Required Higher Processing power.



Pico and Pico W Series (RP2040 MCU based series similar to Arduino nano, Esp Series etc. but with add on Features like dual core, Wi-Fi, inbuilt temperature sensor in board like Pico w)



## 9) Download Libraries for Different Board Series

Note: - for **some boards** we may require to add "additional board manager URL", before installing from Arduino board manager, in this we will see, most popular board manager series installation like **ESP8266** and **raspberry Pico** based boards.

- For **ESP8266 Based Chipset module**: (Yes Same Board's you have seen in chapter 8) Steps: -

- 1) Type **Esp8266 Json link** and open **GitHub page** which appears first /second or open direct GitHub link: - <https://github.com/esp8266/Arduino>
- 2) Scroll down And Copy index.json link :-  
[https://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](https://arduino.esp8266.com/stable/package_esp8266com_index.json)

### Installing with Boards Manager

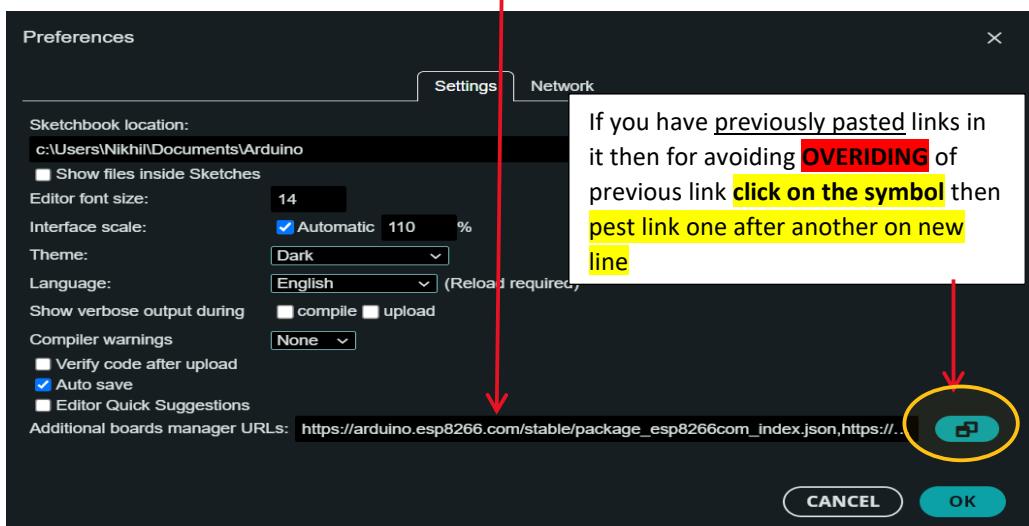
Starting with 1.6.4, Arduino allows installation of third-party platform packages using Boards Manager. We have packages available for Windows, Mac OS, and Linux (32 and 64 bit).

- Download and install Arduino IDE 1.x or 2.x
- Start Arduino and open the **Preferences** window
- Enter [https://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](https://arduino.esp8266.com/stable/package_esp8266com_index.json) into the **File>Preferences>Additional Boards Manager URLs** field of the Arduino IDE. You can add multiple URLs, separating them with commas.
- Open Boards Manager from Tools > Board menu and install esp8266 platform (and don't forget to select your ESP8266 board from Tools > Board menu after installation).

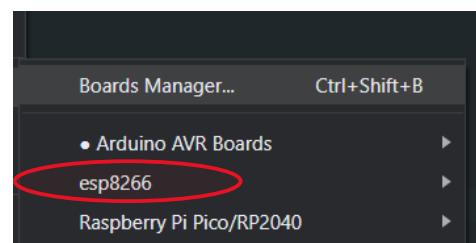
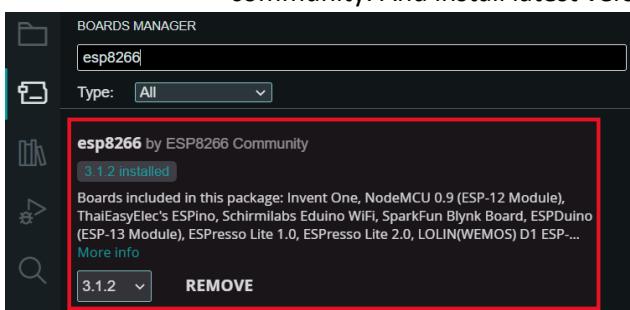
Latest release **release v3.1.2**

Boards manager link: [https://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](https://arduino.esp8266.com/stable/package_esp8266com_index.json)

- 3) In Arduino IDE **file>preferences** pest the **copied** link in Arduino board manager.



- 4) Finally, Under **Tools>Board>Boards manager** search for **esp8266 by ESP8266** community. And install latest version – **1<sup>st</sup> pic**. After Successful installation **2<sup>nd</sup> pic**.

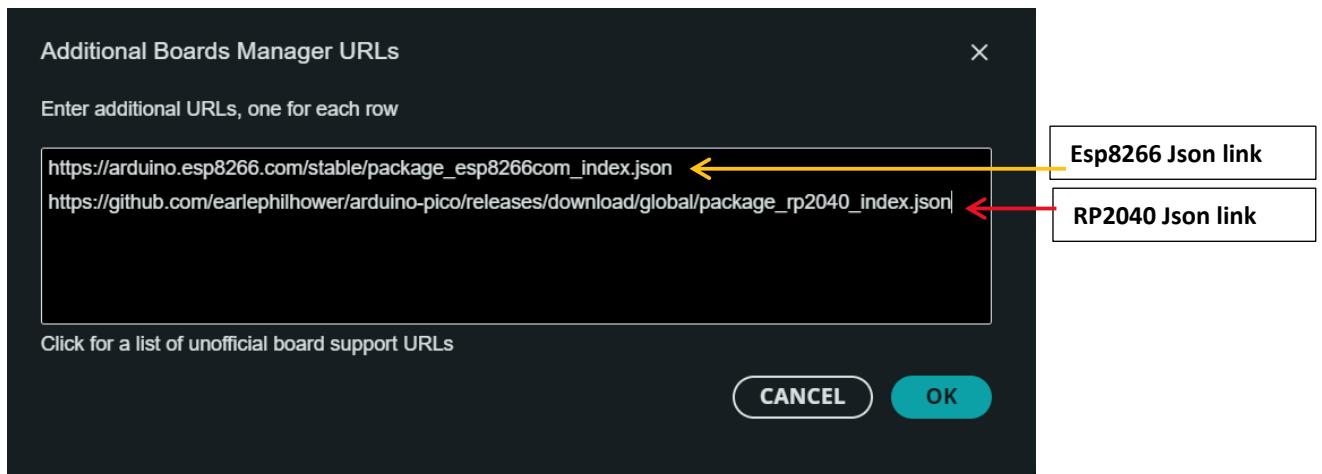


- For **Raspberry pi RP2040** based board's: - (Yes, for boards which you seen in 8.2) Steps: -

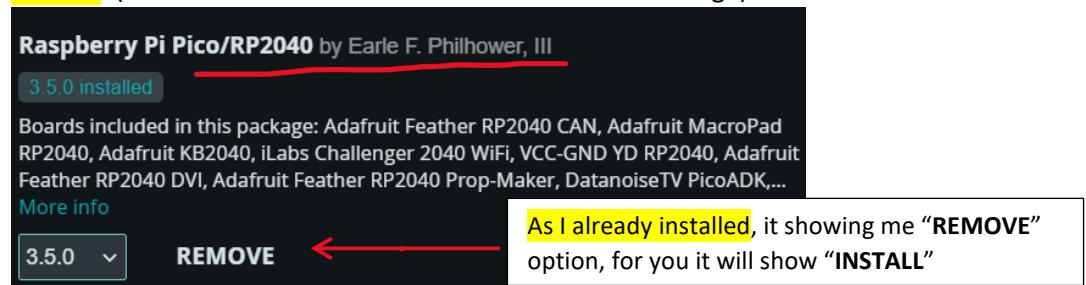
(Note: - Steps are similar like previous ESP8266 based board)

- 1) Similar to previous steps, in browser search for raspberry Pico board manager and open GitHub link: - <https://github.com/earlephilhower/arduino-pico>
- 2) Scroll down and **Copy index.json** link you can also view the supported board series like raspberry Pico , Pico w etc. OR directly copy below Json Link: - [https://github.com/earlephilhower/arduino-pico/releases/download/global/package\\_rp2040\\_index.json](https://github.com/earlephilhower/arduino-pico/releases/download/global/package_rp2040_index.json)
- 3) In Arduino IDE file>preferences pest the **copied** link in Arduino board manager: -
 

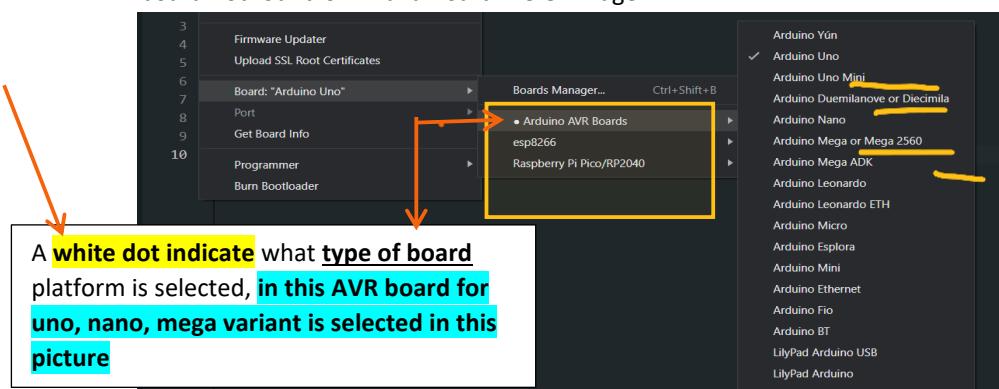
**Note** as I said, I already have esp8266 index .json link, for **avoiding overriding of previous index.json link** maximize the box and pest one after another on new line



- 4) Finally, Under Tools>Board>Boards manager search **for pico rp2040 and install latest version**. (Please install same author name as shown in image)



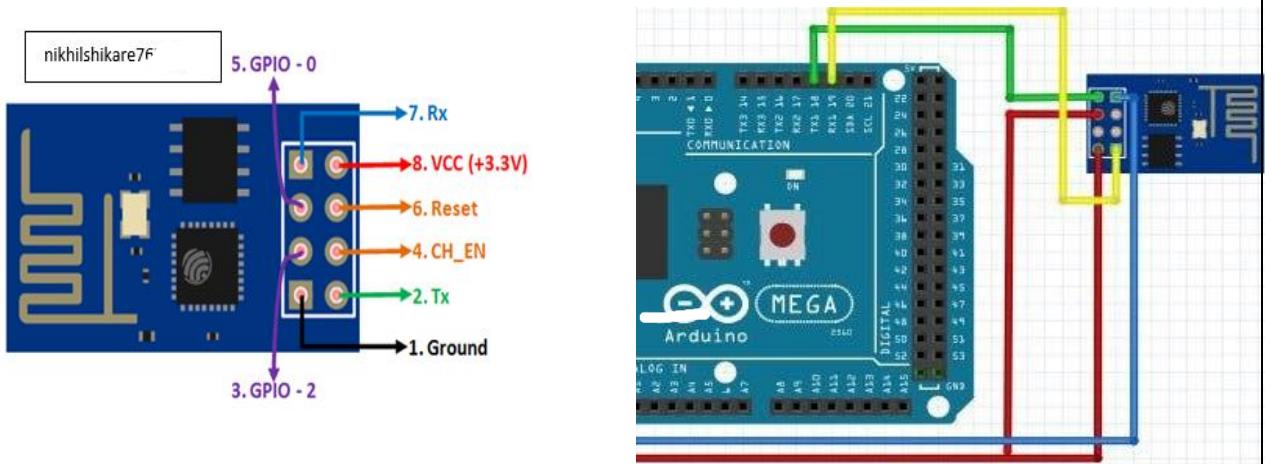
- 5) You can verify installed board under tools>board. Also, you need to **select type of board** first before uploading code. Meany beginners make mistake that they want to upload code for esp8266 / NodeMcu, but **by default** Arduino AVR bord is selected so it will throw error as board not found or invalid Board. Refer image.



## 10) Wi-Fi Connectivity For (Wi-Fi Esp-01 And NodeMcu)

[**Note**] as Code for both esp-01(wi-fi module) and NodeMcu the program code is big and taking lot of space for both it is taking 6 pages (3+3), so I have added this code in upcoming chapters, i.e., **refer chapter 17 and 18**]

- For Esp-01 we Require “**AT**” command for operation like, **connecting WIFI, check baud rate, list all AP etc.** it also includes advance command. kindly refer the official documentation link: - [https://docs.espressif.com/projects/esp-at/en/latest/esp32/AT\\_Command\\_Set/Wi-Fi\\_AT\\_Commands.html](https://docs.espressif.com/projects/esp-at/en/latest/esp32/AT_Command_Set/Wi-Fi_AT_Commands.html)
- Refer Diagram for connection. and **use 3.3v only. Connecting to 5v will damage** the esp-01 (wi-fi module) board.  
And Use **SoftwareSerial Library** for **Rx, Tx** Pins
- Both **NodeMCU and ESP-01 are Wi-Fi modules** based on the ESP8266 chipset, and they operate in the **2.4 GHz frequency band**. The 2.4 GHz frequency band is a part of **the ISM (Industrial, Scientific, and Medical) bands and is widely used for various wireless communication technologies, including Wi-Fi.**



- Now for NodeMcu for making different methods calls like POST, GET, search on internet/YouTube, we have lot of **step-by-step guides**.
- For making HTTP, URL calls, POST and GET request again go for online guide. For both NodeMcu and esp-01 Wi-Fi module there are **different approach for making **HTTP Request****

**Note: -**

**Refer chapter 17 and 18,** I have added code for **Wi-Fi connectivity** and Making **HTTP GET request** in letter parts due to Size for Both NodeMcu And ESP-01 code.

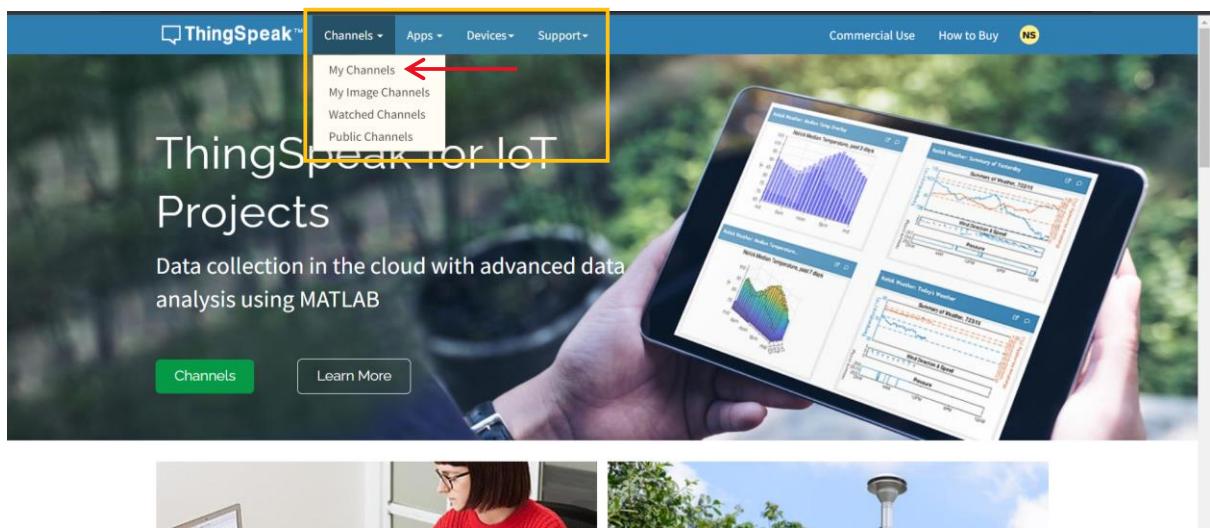
## 11) What is Thing Speak? Setup.

- Quick Intro: - **ThingSpeak** is online platform service specially made for **IOT** devices. It can store sensors data, retrieve data, Trigger another HTTP request and Meany other things. It offers free and paid service depending on use Cases. In this we are going to start with free plan which is more than sufficient for beginners.

Official YouTube Link for quick start: - <https://youtu.be/ZayVoLxeRq4?si=1B5LiAfZDxNhpnlE>

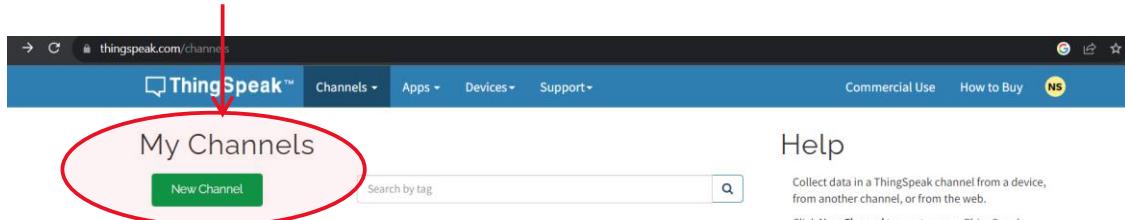
**Step to setup** ThingSpeak channel to store and display the insights of the sensor data, like temperature, humidity sensor which we are sending via **HTTP GET URL**. **STEPS:** -

- 1) **Signup / login** to your ThingSpeak using email. Note: - for **1<sup>st</sup> time they will send you a verification email. (Collage/Organisation email is recommended)**. After login homepage will look like this.



- 2) Click on My Channel you will see **list of channels if you have** and create button.

**Click on New Channel**



- 3) Fill the detail like **Name**, description, and **field**. we can add up to 8 fields. Other **filled are optional**. **Leave it blank**. Then **click on Save Channel** button.

### New Channel

### Help

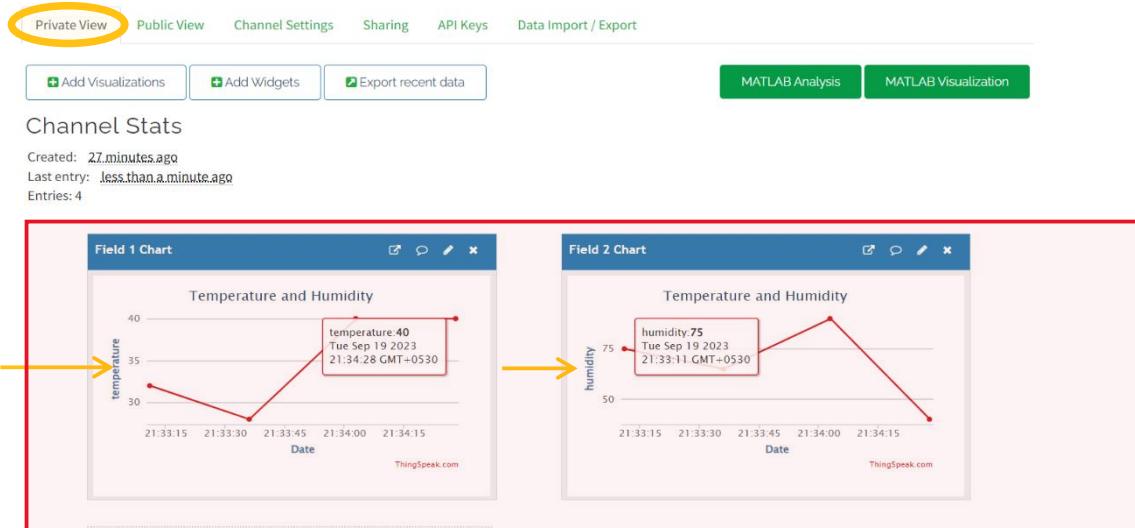
Name	Temperature and Humidity	Channels: eight field: status data visualize it
Description	Just for testing	Chann
Field 1	temperature	• Pers ch cha cha
Field 2	humidity	• Cha
Field 3		• Des

- 4) Now after creating a channel, we will see below window with bunch of other options, just go through once, for us **most important and most used options** are explained below.

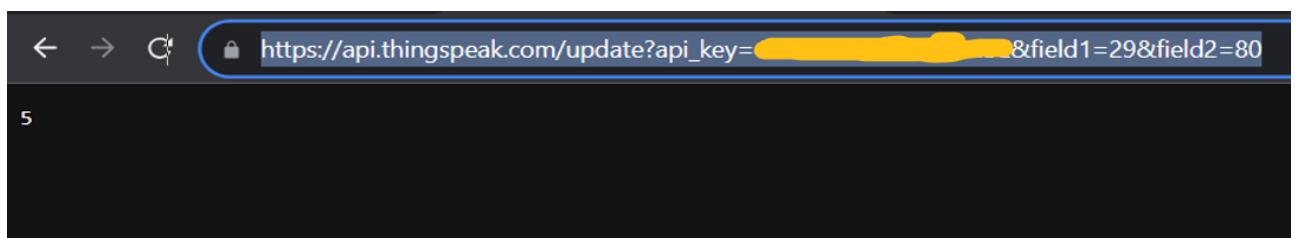
- **Private View** tab which shows graphical representation of data. we can modify it with Gauge, Numerical display etc.
  - **Channel setting** reopen the previous image window and modify the channel setting
  - **API keys** contain **read/write** API keys which we are calling in **HTTP URL** from **Microcontroller**
- 5) In API keys, **copy write Api** and **post** in browser for **testing**. I.e add manual value in **field1** and **field2**. Note that **REMOVE "GET"** from URL when posting in browser also use **&operator** to add field2 in URL while calling, similar way **multiple fields** is added using & operator

~~GET https://api.thingspeak.com/update?api\_key=<YOUR\_API\_KEY>&field1=<value>&field2=<value>....&fieldn=<value>~~

- 6) Now after making 3-4 **test https calls to URL from browser** check where the reading is updated in graph. If updated then your ThingSpeak part setup is **Successful**.



- 7) Making 3-4 **test URL calls** from browser by **random test value in fields** and removing "GET" from start.



(When calling URL **from microcontrollers** we need to **add "GET" in front** of URL in **some cases**)

## 12) Read Sensor Data and Upload to Thing Speak

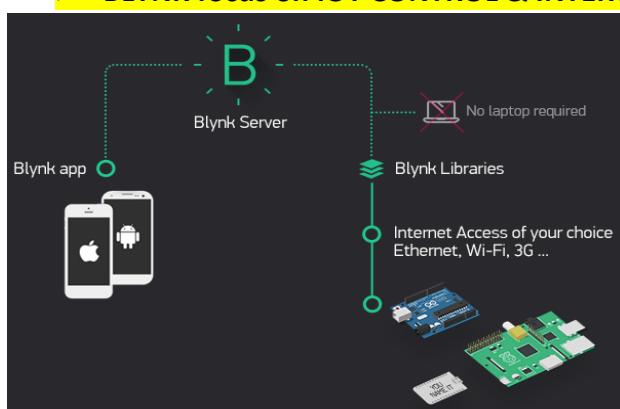
**Note:** - In this, I have provided you approach/steps you need to follow. Based on previous chapters/topic we covered till now.

- 1) Get a sensor it can be Analog, digital etc. From internet find the **specification of sensor** i.e., operating voltage, current from internet or manufacturers datasheet and **library if applied**.  
Example: -DHT11 temp. and humidity sensor or MQ Series gas sensor
- 2) Interface with Arduino, do essential connection first (i.e., VCC/GND). Then add other pins. And then try to read value from sensor and print on **Serial Monitor**
- 3) After printing on serial, if you want to add Display like LCD do add this. Here our layer1 – **Physical layer completed**
- 4) Connect to Wi-Fi for NodeMcu you will have built-in Wi-Fi chip. For Arduino uno we **need to add External Wi-Fi module** (Also Known as ESP-01 – for Photo go to chapter – 8 and 1<sup>st</sup> photo in ESP series), which we **operate using AT commands**
- 5) Setup **thing speak** channel. **Test in browser** where it is writing data in **thing speak** graph.
- 6) **Make HTTP URL call** from microcontrollers **along with parameters** like reading taken from sensor. i.e., for DHT11 we have temp. and humidity reading. In this we completed **layer-2 Communication/Network layer**.
- 7) **Finally** check whether data from Arduino is updating on cloud or not. Further **we may retrieve use this data** for analysis or mobile / web application. In this way **Third layer is completed**

→ **NOTE:** - ThingSpeak, focus on **Data Logging and Visualization, Integration with MATLAB, Customizable Charts, and Support for MATLAB Analytics**

### → Your task is... Blynk

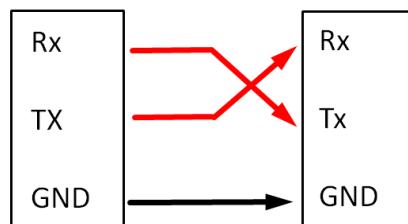
- Perform **same operation using different cloud** service like **BLYNK**
- BLYNK is one of most popular IOT cloud, we can download **BLYNK library** from **Arduino Library manager**. Refer YouTube / Documentation for getting started
- **BLYNK focus on IOT CONTROL & INTERACTIONS, UI, WIDGETS and TRIGGERS**



## 13) Communication Protocols

- Set of rules for **communication and exchanging the information between sensors and microcontrollers**. Example: - Arduino uno board to other board communication. There are several communication protocols and can be categorized under **simplex, half duplex and full duplex**, and depending on baud rate, pins we have use cases.in this we will see some of most popular protocols used in IOT/Arduino and they support almost all type of microcontrollers series.

### 1) Serial Communication (UART):



Serial communication is one of the most basic and widely used communication protocols in Arduino. It involves sending and receiving data over a physical wire (TX and RX pins) or via USB. You can also use it to communicate between two Arduino boards and module.

**Note** that we have **hardware serial** and **virtual/software serial**. In **Arduino UNO** pin **0** and **1** is hardware serial and can be accessed using `Serial.begin()` / `Serial` object.

For **Arduino Mega** board we have **4 hardware Serial** and can be accessed using `Serial.begin()`, `Serial1.begin()`, `Serial2.begin()`, `Serial3.begin()` etc.

But What if we **run out of** HARDWARE serial port, then **use Software / Virtual Serial**, In SoftwareSerial we declare any 2 Digital pin as Rx/Tx pins and create object using **SoftwareSerial** library: - code: -

```

#include <SoftwareSerial.h>
      Rx , Tx
SoftwareSerial mySerial(2, 3); // RX (receive) on pin 2, TX (transmit) on pi

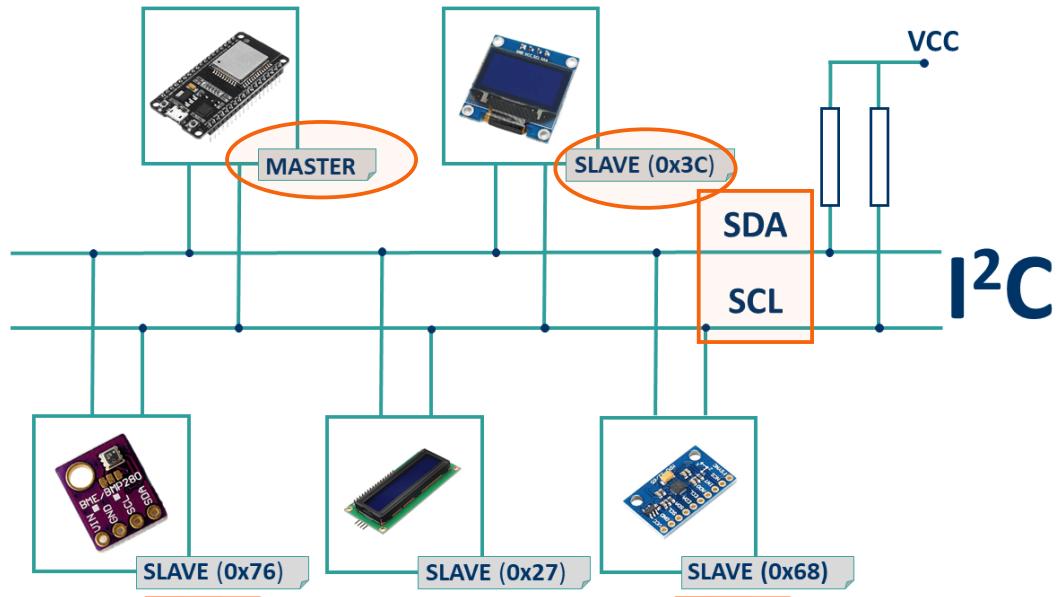
void setup() {
  Serial.begin(9600);    // Initialize the hardware serial port
  mySerial.begin(9600);  // Initialize the software serial port
}
  
```

**Note:** - Software Serial is **not efficient as hardware serial**, they might not be supporting High baud rate. Apart from SoftwareSerial library, we have two more library for virtual/Software serial: - (Click on name for official documentation with code if not open, then search online)

- `#include<SoftwareSerial.h>`
- `#include<AltSoftSerial.h>` // Altsoftserial have Reserved pins, it is designed to work on specific pins, for **Arduino UNO** the pins are **-8 (RX) , 9 (TX) . Pins varies BOARD by BOARD**
- `#include<NeoSWSerial.h>`

**Note** that each virtual/software Serial library has **its own advantage and disadvantage's** like for **AltSoftSerial** we have some **RESERVED pins**, and disable some **PWM** pins but greater accuracy over SoftwareSerial Library. Similarly, we have NeoSWSerial, again depending on use cases we have to use this library

2) **I2C (Inter-Integrated Circuit):** - Uses **Master-Slave Configuration**. for Arduino uno one **peripheral block** is reserved for I2C protocol which uses 7 bits addressing ( $2^7 = 128$ ). So possible range address is 0x00 to 0x127. Used for connecting multiple devices to the same bus, connecting I2C LCD, OLED display, sensors etc. it has **SDA/SCL pins**, and common pin which is VCC and GND. **Theoretically**, in UNO we can connect **up-to 128** devices but in real we don't connect that much. (Invented in **1982 by Philips Semiconductor [now NXP Semiconductors]**)



**Note:** - each device must have **unique I2C address** for Communication:-

```
cpp
// Example code for using I2C with Arduino
#include <Wire.h> ←

void setup() {
    Wire.begin(); // Initialize I2C communication
}

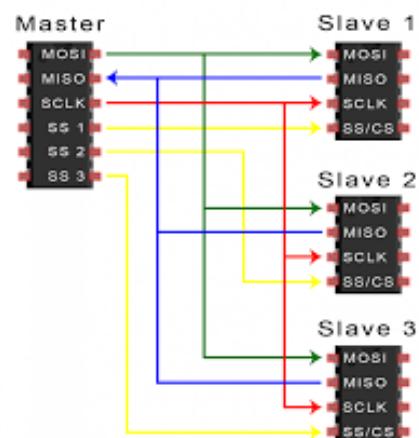
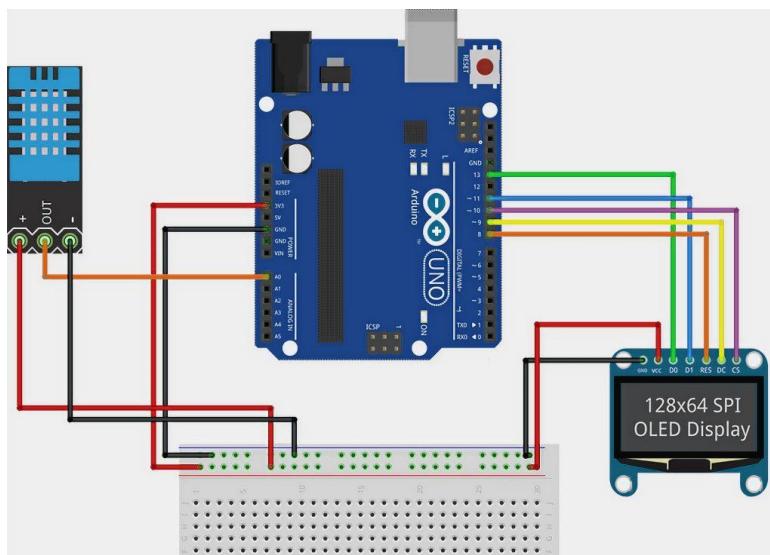
void loop() {
    // Read data from an I2C device
    Wire.requestFrom(DEVICE_ADDRESS, NUM_BYTES);
    while (Wire.available()) {
        int data = Wire.read();
        // Process data
    }
    delay(1000);
}
```

3) **SPI (Serial Peripheral Interface):** - synchronous serial communication protocol used for **high-speed data** transfer between devices. It is most commonly used for interfacing with peripherals such as displays, SD cards, and sensor modules **that require high data rates**.

```
// Example code for using SPI with Arduino
#include <SPI.h> ←

void setup() {
    SPI.begin(); ←
}

void loop() {
    digitalWrite(SS, LOW); // Enable SPI communication
    // Send data and receive data
    digitalWrite(SS, HIGH); // Disable SPI communication
    delay(1000);
}
```



#### 4) One-Wire Protocol:-

it is used in some digital sensors like, temperature , humidity etc.

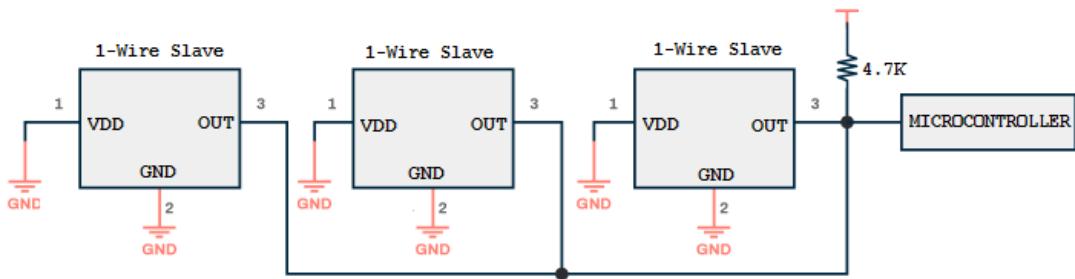
- **Single Wire:** The communication typically occurs over a single wire and ground.
- **Time-Division Multiplexing:** Communication involves time-division multiplexing, where bits are represented by different time durations of high and low voltage levels on the single wire.
- **Parasitic Power:** In some implementations, the sensor **may draw power from the data line (parasitic power)**. In others, there may be a **separate power** line.
- **Checksums:** To ensure data integrity, a checksum or similar method may be used
- **Lets, find out how it is actually happen in case of DHT11**

DHT11 sensor uses a proprietary one-wire protocol for communication with the microcontroller (such as an Arduino). This protocol involves the sensor sending a start signal followed by the data signal, and the microcontroller reading the data.

basic overview of how the DHT11 communication protocol works:

- a. **Start Signal:** The microcontroller initiates communication by pulling the data line low for a specific period (around **18 milliseconds**). This low signal serves as the start signal.

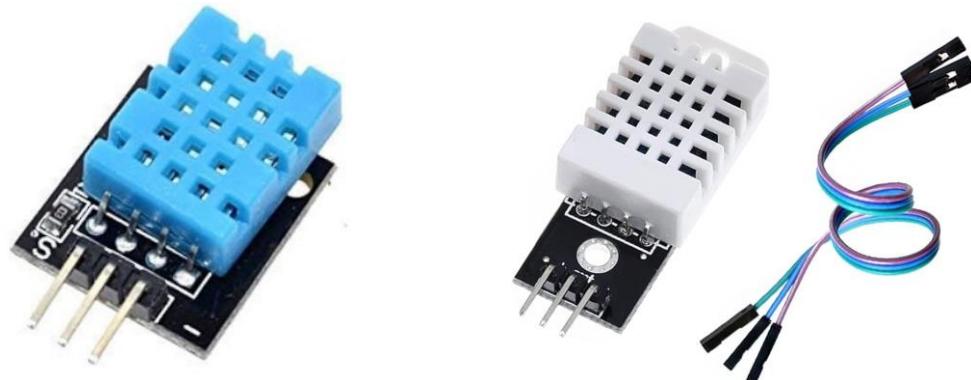
- b. **Response Signal:** After the start signal, the DHT11 responds by pulling the data line low for about 20-40 microseconds.
- c. **Data Transmission:** Following the response, the DHT11 sends the data by modulating the duration of subsequent low and high signals on the data line. Each bit is represented by the duration of the high signal after a low signal. The data consists of humidity and temperature information.
- d. **Checksum:** The DHT11 sends a checksum byte as the last byte of data to ensure data integrity. The microcontroller can use this checksum to verify the received data.



- List of sensors which uses, "one-wire-protocol":-
- a. **DS18B20 (Temperature Sensor):** The DS18B20 is a digital temperature sensor that uses the one-wire protocol.(2<sup>nd</sup> image is waterproof version..)



- b. **DHT11 and DHT22 (Humidity and Temperature Sensors):** While these sensors use a proprietary one-wire-like protocol, they are not strictly adhering to the Dallas Semiconductor's One-Wire protocol. They are often referred to as "single-bus" sensors. (Image of DHT11 and DHT22 sensors)



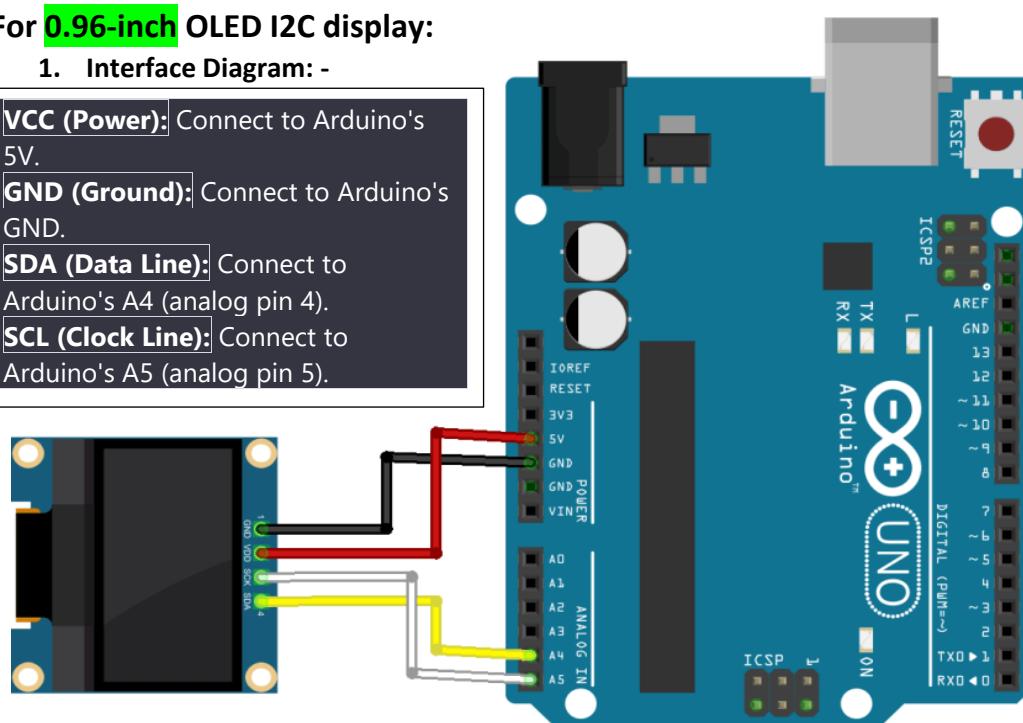
## 14) OLED I2C Display and Library's

**Note:** - (for 1.3-inch OLED use Adafruit SH110X & for 0.96-inch use Adafruit SSD1306)

### A. For 0.96-inch OLED I2C display:

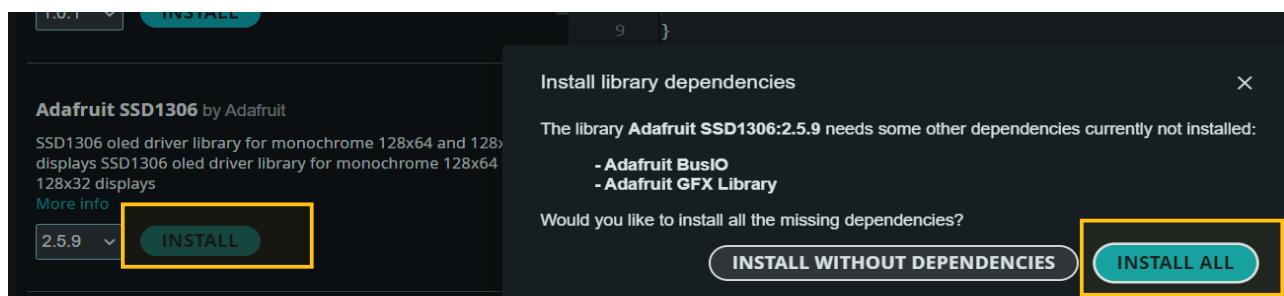
#### 1. Interface Diagram: -

1. **VCC (Power):** Connect to Arduino's 5V.
2. **GND (Ground):** Connect to Arduino's GND.
3. **SDA (Data Line):** Connect to Arduino's A4 (analog pin 4).
4. **SCL (Clock Line):** Connect to Arduino's A5 (analog pin 5).

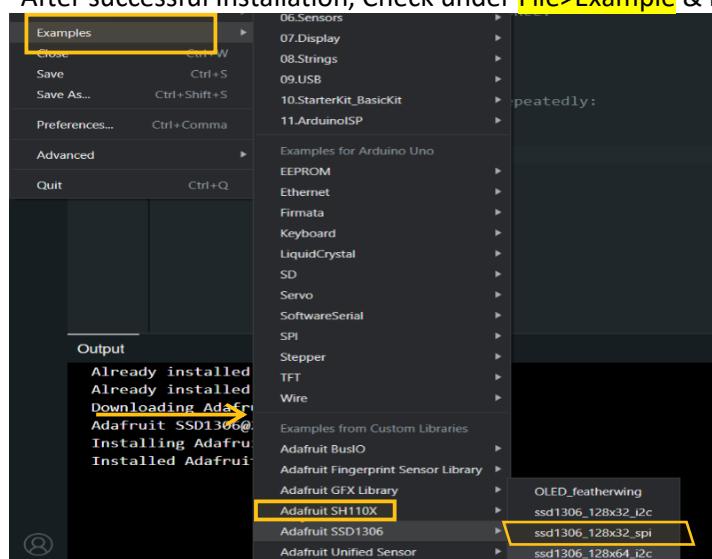


#### 2. Install Library's: -

- Under Sketch > include library > Manage Library's search for **SSD1306 by Adafruit**



- Click on install, and Click on **INSTALL ALL DEPENDENCIES**
- After successful Installation, Check under File>Example & run demo exm.



- if you Don't Know **I2C address of Your OLED**, run **I2C Address Scanner** code from **File>Examples>Wire>I2C Scanner**, or just get from internet/ChatGPT
- Once Example test Code runs successfully, for Custom Print Refer Below Code to Start With.

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

void setup() {
    delay(250);
    // Initialize with the I2C addr 0x3C or 0x3D (for the 128x64)
    if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
        Serial.println(F("SSD1306 allocation failed"));
        for(;;);
    }

    display.clearDisplay();
    display.setTextSize(1); // Set test size 1
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0,0); // Use Cursor For X and y axis
    display.println(F("Hello, world!")); //Print Hello world
    display.display(); //Update/Show on Display
}

void loop() {
    // Nothing to do here
}
```

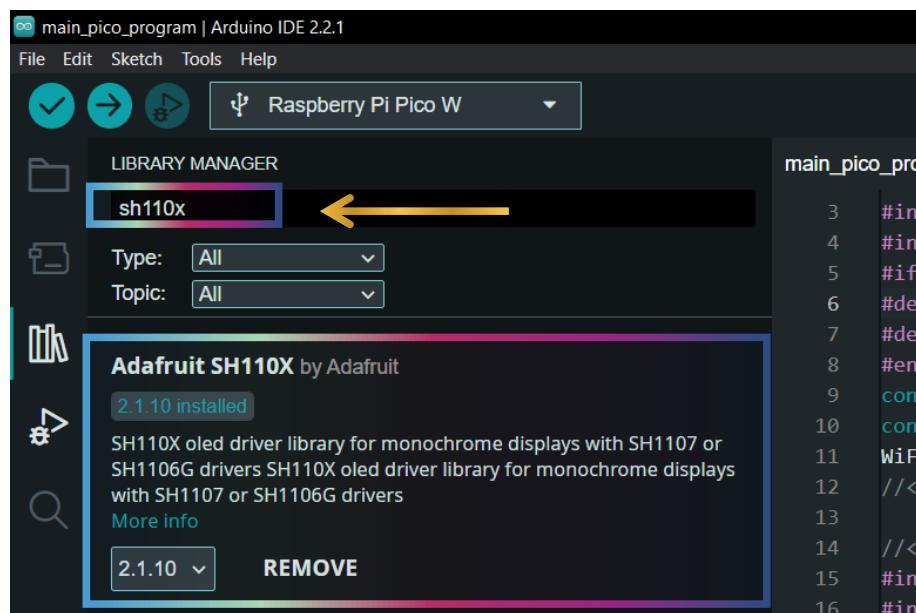
- **Remember Key Functions:** -
  - 1) `ClearDisplay()` :-> Clear Display
  - 2) `setTextSize(1)` :-> Set Size of display , valid Options Are 1 , 2 and 3
  - 3) `setTextColor()` :-> Text colour , some OLED support's White &Yellow
  - 4) `setCursor()` :-> Set Cursor on X and Y position
  - 5) `println()` :-> Print On display & `display()` :-> Update print Buffer

- **Refer Link For** Advanced options like Draw Circle, line, rectangle print Mochrome Images and Play Animations on OLED -->  
[https://adafruit.github.io/Adafruit\\_SSD1306/html/class\\_adafruit\\_ssd1306.html](https://adafruit.github.io/Adafruit_SSD1306/html/class_adafruit_ssd1306.html)
- **Key Function is same for 1.3-inch OLED Display.**

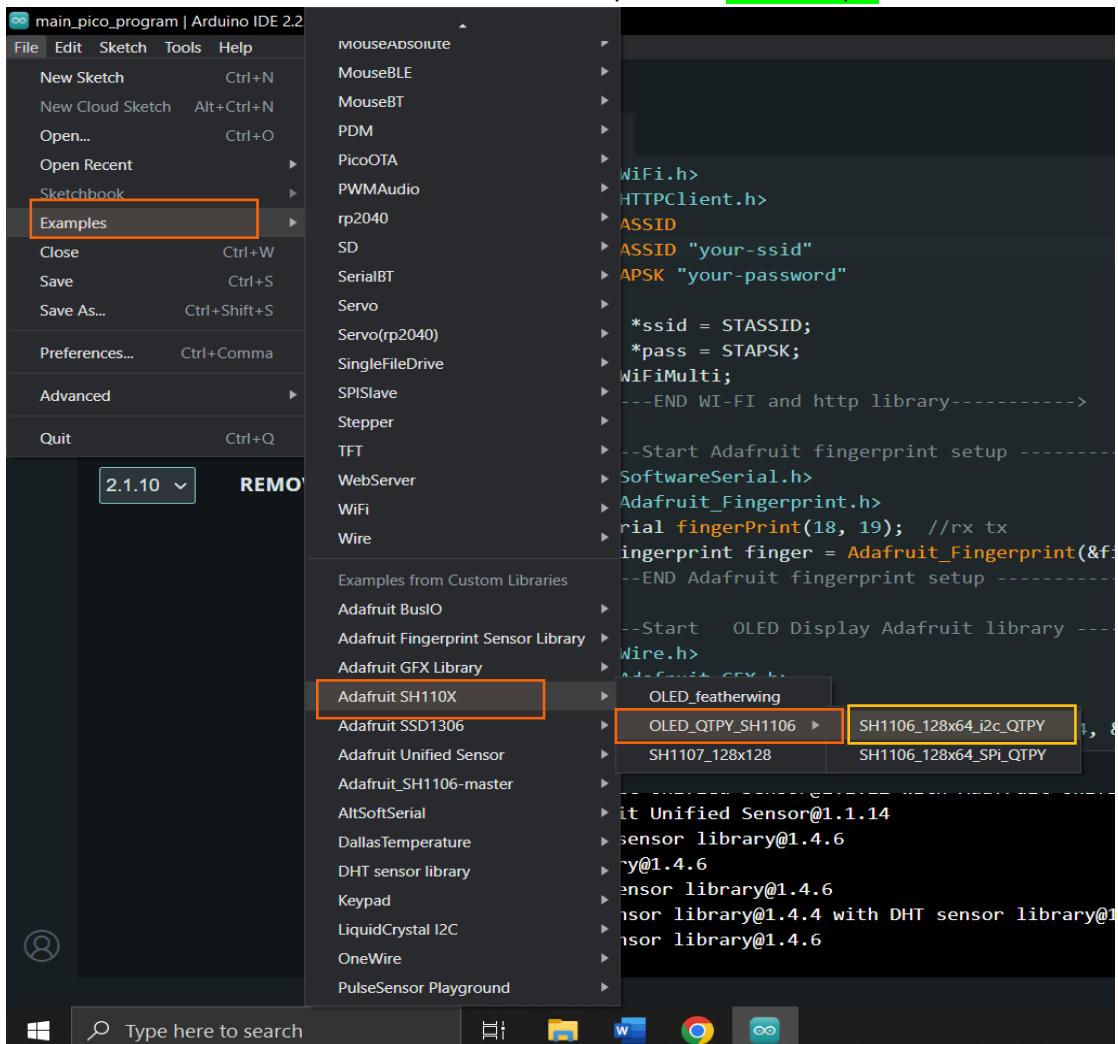
There Are Multiple Libraries Are Available for OLED like **Adafruit SSD1306 & SH110X Library**, **U8g2 Library**, **u8glib Library**, **MicroPython OLED Library** and **list goes on**. But Each Library has it's own use cases, simplicity and platform support. **Just for comparison** We have **Adafruit SSD1306 library** And **U8glib library**. The Adafruit is easy to learn and it's beginner friendly, while u8glib is versatile, compact but have **greater learning curve** and not much beginner friendly but, **THAT DOSENT MEANS u8glib is bad** as I said each has pros and cons such as Adafruit is much heavier than u8glib **so u8glib is used in meany MEMORAY CONSTRATINETD IOT Applications.**

**B. For 1.3-inch OLED Display: - (Except Library's all function's is same as 0.96 OLED)**

1. **Connections:** - Connections Are *Same* as Above / 0.96-inch OLED
2. **Install library's:** - Under **Sketch > include library > Manage Library's** search for **SH110X by Adafruit**



**3. After Successful Installation: - Run Example From `file>Example`**



## 15) PC To Module Vs Board to Module Communication

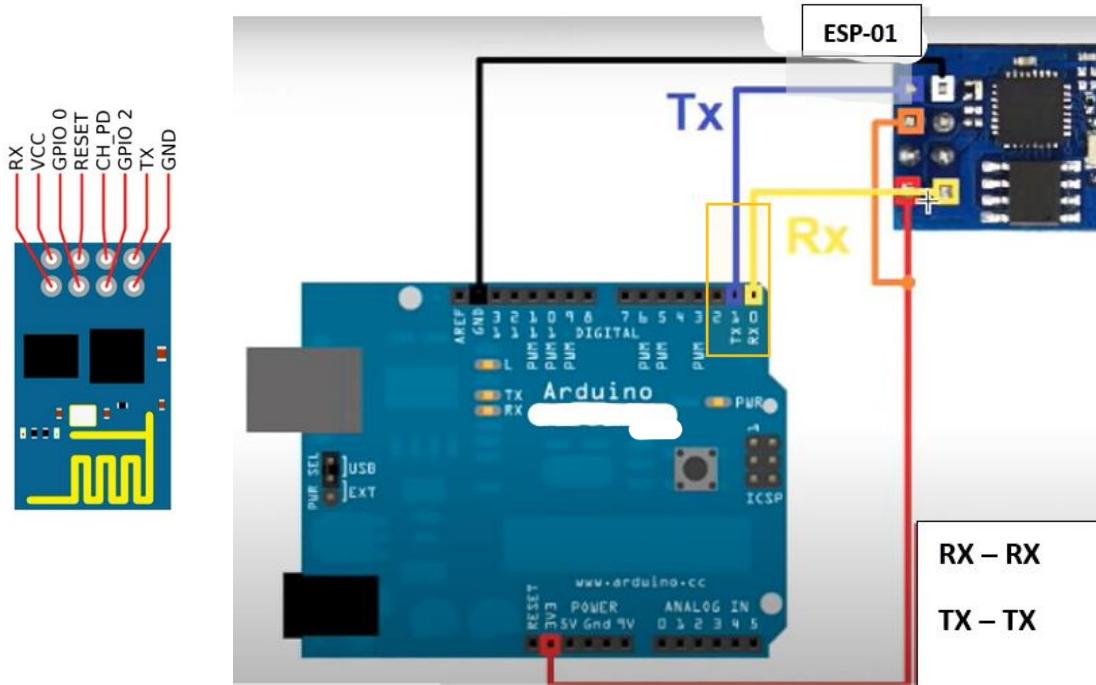
(**UART** Communication with **PC & Module** using **Arduino** as a **Intermediator** VS **Communication between Arduino and Module**)

But wait, why we want to **communicate with our module via Serial monitor??**, Because first we need to test **AT commands** manually, i.e. **in case of Wi-Fi module it used to Connect with Wi-Fi, check Wi-Fi list ,connection status, Make HTTP GET and POST request, sleep commands etc.** similarly **in case of Bluetooth module**, we may test AT commands so, instead of Uploading “**test**” Commands to Arduino Again and Again, better manually test commands via sending through Serial monitor and finally **once we sure we upload SET of Commands** in Microcontroller

**Note:** - Another Most Important use is when **flashing chip / Uploading new firmware**

### a) Pc to Module Communication using Arduino as Intermediator: -

- 1) in **UART Communication RX is connected to TX and Tx Connected to Rx** in sender and receiver.
- 2) But when we want to send **TEST Commands** from **Serial Monitor from Computer** to Module Like **esp-01 WI-FI module or Bluetooth module** Which operate using **AT Commands**, then we need an **intermediator in-between**.
- 3) For an intermediary we have **Microcontroller like Arduino OR USB to TTL Adapter**.
- 4) If We are using Arduino as intermediary then Connect **RX to RX and TX to TX**, now we might get confused that **UART work on cross RX-TX connection, Right???**
- 5) The Reason is as we are using Arduino to **SEND COMMAND THROUGH SERIAL MONITOR** and Our **ARDUINO IS ALREADY CONNECTED** in **RX -TX and TX -RX manner**.



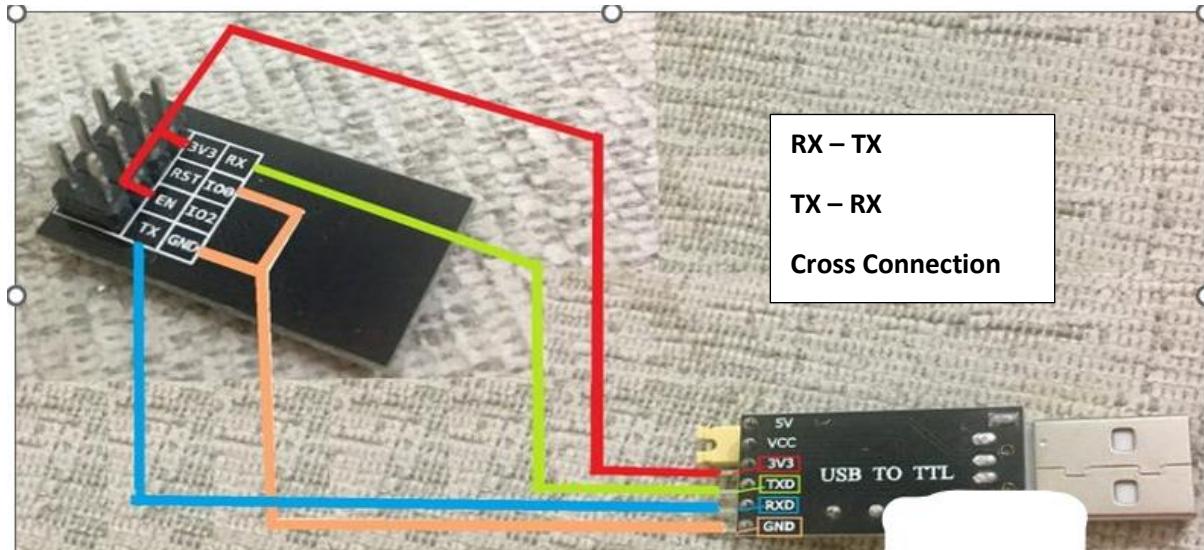
### b) Pc to Module Communication using USB to TTL: -

In This We are connecting in **RX to TX and TX to Rx manner, OPPOSITE of ARDUINO UNO, when Using it as Intermediator**. But why we use **USB to TTL converter**, Again Performing same **task like Testing AT Commands, Firmware update, flashing/Programming chip**, with more convenient use and as an **option to Arduino as Intermediator** we use **USB to TTL**.

They are straight forward; Just **CROSS connect Sender and Receiver Rx-TX**.

Refer Below Image

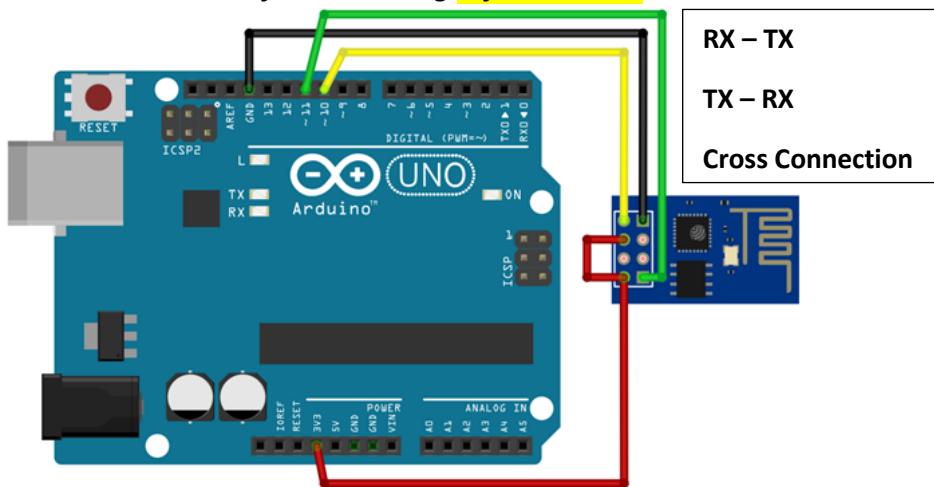




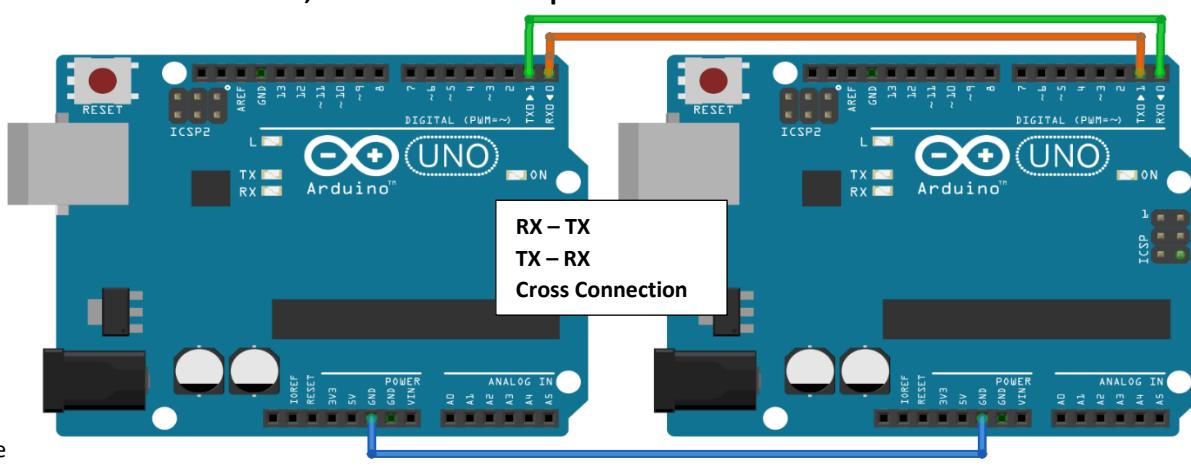
### c) Communication between Arduino and Module: -

Once You Done with Manually Testing AT Commands, it's Time to feed commands in Microcontroller like Arduino or in case If you want to Communicate with Other Microcontroller Board, WE **CROSS CONNECT RX-TX and Terminals** i.e., Connection for UART Communication (Refer Further Lessons for http call by using wi-fi (esp-01) module).

- UART Communication with **wi-fi module using Software Serial**: -



- **UART Communication Between Two Boards via Hardware Serial.** We can also Use **Software/Virtual Serial** using Software Serial Library by using any two digital pins as Rx and Tx, as shown in above picture: -



## 16) What Is “AT” Commands??

- Why we have AT commands for Bluetooth, Wi-Fi, GSM and other modules. Is there any history of AT commands??
  - ➔ AT commands, which stand for "**Attention**" commands, originated from the Hayes command set developed for controlling modems. like perform tasks like dialling, hanging up, or changing settings These commands allowed communication with the modem by sending **specific character sequences prefixed with "AT"** (**Attention**). Over time, these commands became a standard for modem control.
  - ➔ As technology progressed, these commands evolved and **became a foundation** for controlling various communication devices, such as **GSM/3G/4G modules, Bluetooth, and Wi-Fi modules in the IoT domain**. The AT command structure provided a familiar and standardized way to configure and control these diverse communication modules, enabling interoperability and ease of integration across different platforms and devices.
  - ➔ Today, AT commands persist in many IoT modules due to their **ease of use, simplicity, and compatibility across a wide array of devices**, making them a convenient choice for **device configuration and communication** in the IoT landscape.
- IOT modules Which Operated Using AT Commands & Official Link: -
  - ❖ Wi-Fi Modules AT Commands: - [https://docs.espressif.com/projects/esp-at/en/latest/esp32/AT\\_Command\\_Set/Wi-Fi\\_AT\\_Commands.html](https://docs.espressif.com/projects/esp-at/en/latest/esp32/AT_Command_Set/Wi-Fi_AT_Commands.html)
  - ❖ Bluetooth Modules AT Commands: - [https://docs.espressif.com/projects/esp-at/en/latest/esp32/AT\\_Command\\_Set/BT\\_AT\\_Commands.html](https://docs.espressif.com/projects/esp-at/en/latest/esp32/AT_Command_Set/BT_AT_Commands.html)
  - ❖ GSM Modules AT Commands: - <https://www.engineersgarage.com/at-commands-gsm-at-command-set/>
  - ❖ List Of All Other AT commands like, BLE , TCP , MQTT , HTTP: - [https://docs.espressif.com/projects/esp-at/en/latest/esp32/AT\\_Command\\_Set/index.html](https://docs.espressif.com/projects/esp-at/en/latest/esp32/AT_Command_Set/index.html)

The image shows a screenshot of a web-based documentation page for the AT Command Set. On the left, there is a vertical sidebar titled "AT Binary Lists" with a list of command categories. On the right, the main content area has a large red circular border around its title and list of commands.

**AT Binary Lists**

- AT Command Set
  - Basic AT Commands
  - Wi-Fi AT Commands
  - TCP-IP AT Commands
  - Bluetooth® Low Energy AT Commands
  - Bluetooth® AT Commands
  - MQTT AT Commands
  - HTTP AT Commands
  - FileSystem AT Commands
  - WebSocket AT Commands
  - Ethernet AT Commands
  - Signaling Test AT Commands
  - Web server AT Commands
  - Driver AT Commands
  - User AT Commands
  - AT Command Types
  - AT Commands with Configuration Saved in the Flash
  - AT Messages

**AT Command Set**

[中文]

Here is a list of AT commands.

- Basic AT Commands
- Wi-Fi AT Commands
- TCP-IP AT Commands
- Bluetooth® Low Energy AT Commands
- Bluetooth® AT Commands
- MQTT AT Commands
- HTTP AT Commands
- FileSystem AT Commands
- WebSocket AT Commands
- Ethernet AT Commands
- Signaling Test AT Commands
- Web server AT Commands
- Driver AT Commands
- User AT Commands

## 17) Operating WI-FI Module Using “AT” Commands...

(Covers [HTTP Request call Demo](#) and [Connecting to Wi-Fi Network](#))

[ For **Explanation of Individual AT Command**, Please Refer Different **AT Commands Documentation links** from above section. Specially ***take look at WI-FI, TCP-IP & HTTP AT commands***]

**Code and connection:** -

```
//-----Include SoftSerial/Virtual Serial Library ----->
#include <SoftwareSerial.h>
//Code by ® Nikhil Mohan Shikare"

//----- WI-FI /ESP-01 module pin rx-tx----->
#define RX 4
#define TX 3

String AP = "YOUR_WIFI_NAME";           // AP NAME
String PASS = "YOUR_WIFI_PASS";         // AP PASSWORD
String API = "WRITE_API_THINGSPEAK";    // Write API KEY Of THingspeak
                                         // API looks Similar like -->
PC3C54YMXFg53QP2D6
String HOST = "api.thingspeak.com"; //Host name
String PORT = "80";

//UART Communcation
SoftwareSerial esp8266(RX, TX); //Create Instance Of SoftSerial

void setup() {
  // put your setup code here, to run once:
  delay(2000);
  Serial.begin(9600);      // Start Hardware Serial at 9600 bps
  //Code by ® Ni-khil Mohan Shikare"
  esp8266.begin(115200); //Start Virtual/Software Serial At 115200 bps
  Serial.println("Serial Started..");
  //Serial.println("Code by Nikhill mo_han Shi-kare");

  //-----Code To Connect WI-FI Start From Here----->
  sendCommand("AT");
  sendCommand("AT+CWMODE=1");

  sendCommand("AT+CWJAP=\"" + AP + "\",\"" + PASS + "\"");
  delay(800);
  //-----Code To Connect WI-FI END From Here----->
}

void loop() {
  // put your main code here, to run repeatedly:

  int random_value = 25; // Replace With Your Sensors value
```

```

// We have one Parameters Field1. You may have UPTO 8 Parameter on
// "thingspeak"
//NOTE--> Depending On Sensor. Just Modify Below "getData" String
//<-----Thingspeak START Hrom Here----->

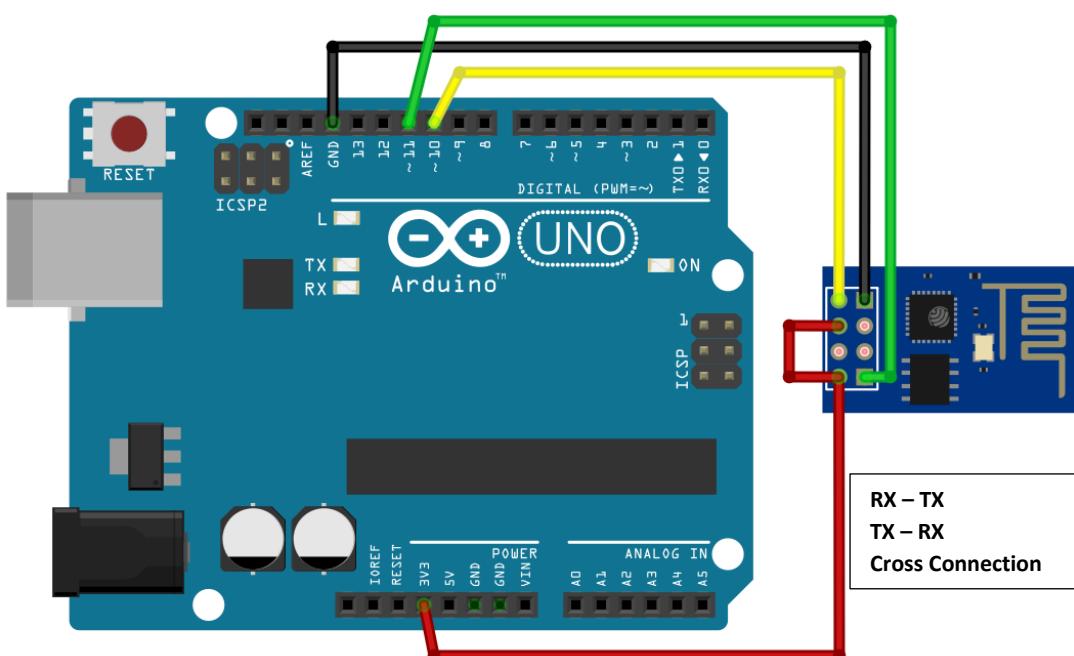
String getData = "GET /update?api_key=" + API + "&field1=" +
(String)random_value;
sendCommand("AT+CIPMUX=1");
delay(200);
sendCommand("AT+CIPSTART=0,\\"TCP\\",\"" + HOST + "\",," + PORT);
delay(200);
sendCommand("AT+CIPSEND=0," + String(getData.length() + 4));
esp8266.println(getData);
delay(100);
sendCommand("AT+CIPCLOSE=1");
//<-----Thingspeak END Hrom Here----->

//Wait for 5 seconds before making Next HTTP Call
delay(5000);
}

void sendCommand(String command) {
Serial.println("---At command => ");
Serial.print(command);

//Send "AT" Commands to Softserial of Esp-01/Wifi module
esp8266.println(command);
delay(300);
Serial.println("<----Response From ESP-01/WIFI module---->");
Serial.println(esp8266.readStringUntil("\n"));
delay(300);
}
}

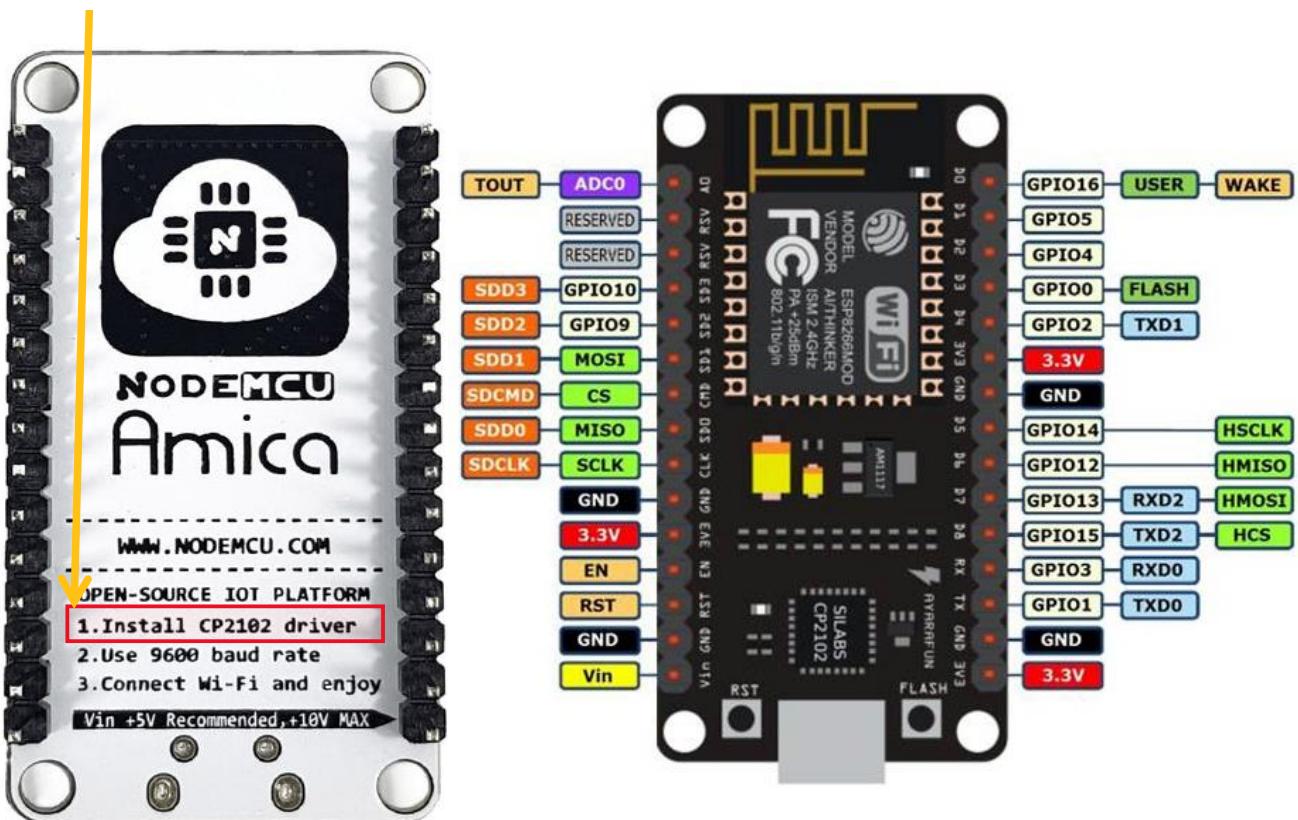
```



## 18) Making HTTP call using NodeMcu/ESP8266

- The NodeMCU is an open-source development board that features the **ESP8266 Wi-Fi module**. The ESP8266 is a low-cost, low-power, and **integrated Wi-Fi chip** that became popular in the maker community for Internet of Things (IoT) projects.
- Unlike Arduino uno NodeMcu have built-in WI-FI Chip on it. Which make extremely easy for beginners to connect with WI-FI and make HTTP calls. Unlike Arduino UNO we required separate WI-FI (esp-01) module for ([https://www.nodemcu.com/index\\_cn.html](https://www.nodemcu.com/index_cn.html))

- 1) Install Driver for NodeMcu Printed on backside. Search on Internet open 1<sup>st</sup> / 2<sup>nd</sup> link. **Download According to OPERATING SYSTEM** i.e., Windows/MAC/LINUX OR Open Link: - <https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads>



[silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads](https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads)

**SILICON LABS** Products Applications Ecosystems Resources Company English

Home // Developers // USB to UART Bridge VCP Drivers

### Download and Install VCP Drivers

Downloads for Windows, Macintosh, Linux and Android below.

\*Note: The Linux 3.x.x and 4.x.x version of the driver is maintained in the current Linux 3.x.x and 4.x.x tree at [www.kernel.org](http://www.kernel.org).

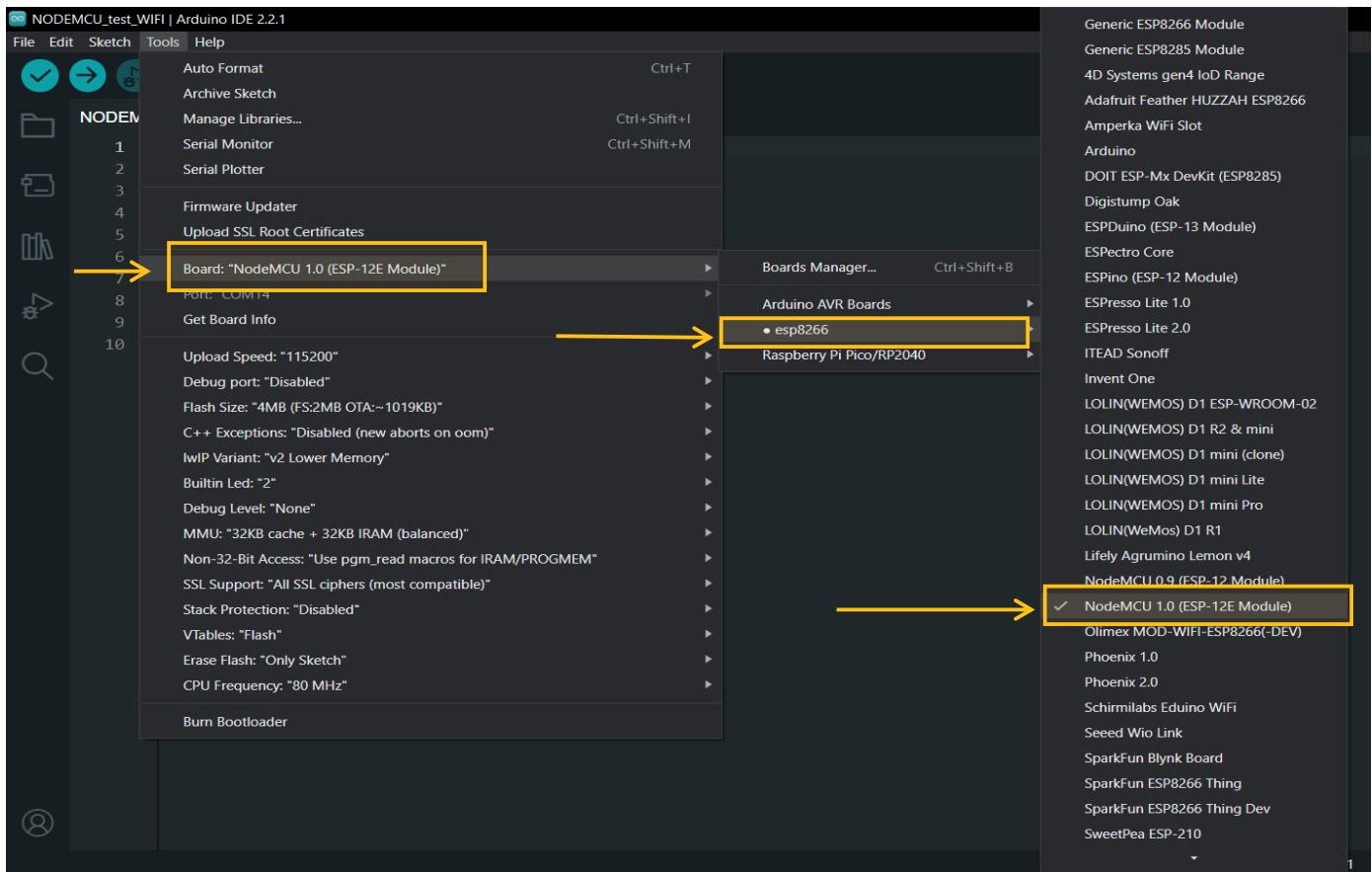
#### Software Downloads

Software (11)	Software • 11
<a href="#">CP210x Universal Windows Driver</a>	v11.3.0 6/24/2023
<a href="#">CP210x VCP Mac OSX Driver</a>	v6.0.2 10/26/2021
<a href="#">CP210x VCP Windows</a>	v6.7 9/3/2020
<a href="#">CP210x Windows Drivers</a>	v6.7.6 9/3/2020
<a href="#">CP210x Windows Drivers with Serial Enumerator</a>	v6.7.6

**Legacy OS Software Versions**  
Driver Package download links and support information

**Serial Enumeration Driver**  
What is the serial enumeration driver and why would I need it?

- 2) But Why Device drivers, unlike Arduino NodeMcu needs **device drivers to get DETECTED by OS on Communication Port**
- 3) After Installation, under **tools>boards>esp8266** select NodeMCU. For board manager Installation refer chapter 9 first part.



- 4) Using **USB cable Upload Below Code in NodeMcu for WIFI Connection and HTTP calls:** - **Make Sure to change parameter** with your sensor value.

```
#include <SoftwareSerial.h>
#include <ESP8266WiFi.h>
//Code by © Nikhil Mohan Shikare"
#include <ESP8266HTTPClient.h>
#include <WiFiClient.h>

WiFiClient client;
HTTPClient http;

String url;
String AP = "YOUR_WIFI_NAME";           // AP NAME
String PASS = "YOUR_WIFI_PASS";         // AP PASSWORD
String API = "WRITE_API_THINGSPEAK";    // Write API KEY Of Thingspeak
                                         // API looks Similar like --> PC3M6ETC54YMXFg53QP2D6

int InitWifi() {
  if (WiFi.status() == WL_CONNECTED)
    return 1;
```

```

WiFi.begin(AP,PASS);

delay(1000);
while (WiFi.status() != WL_CONNECTED) {
    delay(400);
    Serial.println("...");
}

Serial.println("Connected To Wifi");
Serial.println(WiFi.localIP());
Serial.println("Code by Nikhil Mohan Shikare");
if (WiFi.status() != WL_CONNECTED)
    return 0;
return 1;
}

String send_API_request(int value1 , int value2) {

url = "http://api.thingspeak.com/update?api_key=";
url = url + API;
url = url + "&field1=";
url = url + (String)value1;
url = url + "&field2=";
url = url + (String)value2;

Serial.println(url);
if (WiFi.status() == WL_CONNECTED) {
    http.begin(client, url);
    delay(100);
    int http_code = http.GET();
    Serial.println("http Code");
    //Code by @ niKhIl Mohan Shikar_e"
    Serial.println(http_code);
    if (http_code > 0) {
        String response = http.getString(); //Response from server
        http.end();
        //Serial.println(response);
        return response;
    }
    http.end();
    delay(1500);
    return "No Response";
}
return "No WI-FI";
}

void setup() {
Serial.begin(9600); // Start Hardware Serial At 9600 bps
InitWifi(); // Our Function To Connect Wifi
//Code by @ Nikhil Mohan Shikare"
delay(1000);
}

```

```

}

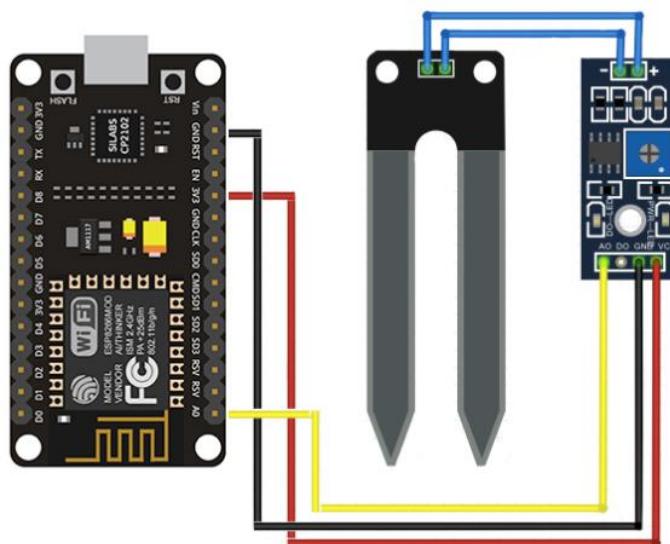
void loop() {

    int value1 = 47;//Replace with Your sensor values
    int value2 = 86;//Replace with Your sensor values
    //Like temprature, gas or any other sensor reading

    Serial.println(send_API_request(value1,value2));
    delay(5000); // Wait for 5 seconds before making next HTTP call
}

```

➤ **Demo Image for Reference** Which reading data and uploading to cloud: -



**For correction/suggestion/improvements:** - [nikhil.shikare@vit.edu.in](mailto:nikhil.shikare@vit.edu.in) / <https://www.linkedin.com/in/nikhil-shikare-18127a216/>

[nikhilshikare765@gmail.com](mailto:nikhilshikare765@gmail.com)

## 19) Common Errors and Troubleshooting

(For verification, I am adding ChatGPT screenshots)

- In case of I2C communication, I have observed That, **board is freezes** on incorrect connection. Let's find out why? (Carefully refer below Screenshots).



**You**

In some cases , i have observed that when you Incorrectly connect I2C cable in Arduino board . i.e. ~~sda~~ is pin A4 and scl is A5. When we make mistake in connection. The Arduino Uno board is "freezes " **Is this observation is correct**



**ChatGPT**

**Yes, your observation is correct.** In Arduino boards, including the Arduino Uno, when you incorrectly connect the I2C (Inter-Integrated Circuit) cables, such as swapping the SDA (data line) and SCL (clock line) pins, it can lead to communication issues and cause the board to appear frozen.

The I2C communication relies on a specific protocol, and swapping SDA and SCL disrupts the proper communication between the Arduino and the connected devices. As a result, the connected devices may not respond as expected, **leading to a perceived freeze in the Arduino board.**

- In case of **UART**. We don't see freezing issues, but we might face issue such as receiving garbage values or unable to receive any message.



**You**

In case of UART communication we generally don't see such behavior. Unlike I2C , which freezes on swapping pins , but **UART** on swapping just don't show response and also it not freezes arduino. **Is this observation is correct ?**



**ChatGPT**

**Yes, your observation is correct.** The behavior you described is a result of the fundamental differences between UART (Universal Asynchronous Receiver-Transmitter) and I2C communication protocols.

In UART communication:

- Asynchronous Nature:** UART is asynchronous, meaning **there is no shared clock signal** between the communicating devices. Instead, both the **sender and receiver agree on a specific baud rate.** Each byte is framed by start and stop bits, allowing devices to synchronize.
- No Strict Timing Requirements:** UART is more forgiving in terms of timing. Swapping TX (transmit) and RX (receive) lines **typically doesn't cause the system to freeze.** Instead, it may lead to data not being received or transmitted properly, resulting in communication errors.
- No Fixed Master-Slave Relationship:** UART communication doesn't have a master-slave relationship like I2C. Devices can communicate in a point-to-point or multi-point configuration, and the communication flow is often determined by the software.

Due to these characteristics, **swapping TX and RX lines in UART communication might result in missed or incorrectly received data, but it generally won't cause the Arduino to freeze or hang.** The **Arduino will continue executing the code, and you may not observe a complete system halt.**

- 3) There is one more **interesting issue I observed**, when using SoftwareSerial/Virtual Serial UART **communication protocol** in our project.
- So, in our IoT subject, we have a mini project on IoT. For our project, I was **using 3 serials**: 1 **hardware serial at 9600 bps** and **2 software serial instances at 57000 and 115200 bps**.
- So, when Arduino start executing code, only hardware serial (which is accessed using Serial object i.e., `Serial.Begin(9600)`) and 1 software Serial is working properly.
- I have **one software serial for fingerprint sensor module** and **one for WI-FI (esp-01) module**. So, fingerprint serial and hardware serial working properly. But ESP serial doesn't print anything. I **cross checked connections**, everything was correct, **then what was issue???**
- After, spending lot of time on debugging, I found that this was **Software/virtual serial issue**. As mentioned in chapter 13(Communications protocol) 1<sup>st</sup> part UART communication, that **VIRTUAL/SOFTWARE Serial IS NOT as EFFICIENT as HARDWARE SERIAL, if FINGERPRINT sensor is working then ESP serial don't work and IF we comment out fingerprint serial then only ESP serial working...**
- So, after searching, I found that it was happening due to **HIGHER BAUD RATE USED BY Multiple SOFTWARE SERIAL Objects**, On Arduino UNO. i.e., if **using multiple software serial at higher baud rate** like 57000, 115200 then it causes this issue.
- So, to overcome this, I change the baud rate of ESP serial from **115200 to 9600**, but it started printing **garbage** values as ESP-01 module as default baud rate is 115200 so Mismatch in sender and receiver BAUD rate in UART protocol print garbage values.
- **To overcome this, the options was:** -
- Change ESP-01 default baud rate to 9600 by using **AT commands**.
  - Use board like **NodeMcu and Arduino**, **board to board communication** using UART protocol at 9600, i.e., **using Arduino for sensor interface and NodeMcu for WIFI connectivity and HTTP calls**.
  - I used second Option.
- Let's take a look of my code that causing this issue. Note The ESP Baud rate

```

41 //-----Start hardware & Virtual instance Of SoftwareSerial----->
42 Serial.begin(9600); ←
43 ESP8266.begin(115200); ←
44 lcd.begin(16, 2);
45 finger.begin(57600); ←
46 delay(1500);

```

- So, to fix this, I have to minimize baud rate of ESP, from 115200 to 9600.

```

41 //-----Start hardware & Virtual instance Of SoftwareSerial----->
42 Serial.begin(9600); ←
43 ESP8266.begin(9600); ←
44 lcd.begin(16, 2);
45 finger.begin(57600); ←
46 delay(1500);

```

- When I asked about my **observations, conclusions**, and challenges to ChatGPT, to find if any technical mistake or anything was wrong, then **ChatGPT response** is ->

Your conclusion is that the inefficiency of software serial, particularly when used with higher baud rates, leads to issues when multiple instances are employed simultaneously. Specifically, you noted that if you comment out the software serial instance for the fingerprint sensor, the ESP-01 Wi-Fi module's software serial starts working.

It's a valid conclusion, and your observation aligns with common challenges when working with SoftwareSerial on Arduino Uno, especially at higher baud rates and with multiple instances. The limited processing power and interrupt capabilities of the Arduino Uno can lead to timing and communication issues in such scenarios. The recommendation to use hardware serial ports or optimize code is relevant in addressing these challenges.

for correction or suggestions: [nikhil.shikare.765@gmail.com](mailto:nikhil.shikare.765@gmail.com) / [nikhil.shikare@vit.edu.in](mailto:nikhil.shikare@vit.edu.in) / <https://www.linkedin.com/in/nikhil-shikare-18127a216/>

4) Some Most Common issues are mentioned below with their solutions,

➤ **General Hardware and Wiring Issues:**

1. **Loose Connections:** Ensure all wires and components are securely connected. Loose connections can lead to unreliable behaviour.
2. **Power Supply:** Verify that the power supply meets the requirements of your components. Insufficient power can cause erratic behaviour.
3. **Incorrect Wiring:** Double-check your wiring against the datasheets and documentation for your components. Incorrect connections can lead to unexpected behaviour.
4. **Polarity and Orientation:** Check the polarity of components, such as diodes and electrolytic capacitors, and the orientation of integrated circuits (ICs).
5. **Grounding Issues:** Ensure a common ground is established between all components. Inconsistent grounding can cause signal integrity issues.

➤ **Software Issues:**

6. **Code Errors:** Carefully review your code for syntax errors, logical mistakes, or incorrect configurations. Use debugging tools and print statements to identify issues.
7. **Memory Constraints:** Be mindful of the limited memory on microcontrollers. Optimize your code, especially when dealing with large datasets or libraries.
8. **Library Compatibility:** Ensure that libraries are compatible with your hardware and software versions. Using outdated or incompatible libraries can lead to issues.
9. **Variable Types:** Pay attention to variable types and data sizes, especially when working with different platforms. Mismatched data types can cause unexpected behavior.
10. **Infinite Loops:** Avoid infinite loops without breaks or delays. Such loops can make it challenging to upload new code and may lead to unresponsive behaviour.

➤ **Communication Protocols:**

11. **I2C Address Conflicts:** Ensure that the I2C addresses of your devices do not conflict. Multiple devices on the same bus with the same address can cause communication issues.

12. **SPI Wiring and Modes:** Double-check the wiring and SPI modes (CPOL and CPHA) when using SPI. Inconsistent configurations can lead to communication problems.
13. **UART Baud Rate Mismatch:** Ensure that the baud rates of devices communicating over UART are set to the same value. Mismatched baud rates can result in garbled data.
14. **Interrupt Conflicts:** Be cautious with interrupt usage, especially when using multiple devices that rely on interrupts. Conflicts can lead to unexpected behavior.
15. **Timing Considerations:** Pay attention to timing requirements in communication protocols. Inaccurate timing can result in data corruption or missed communication.
  
- 5) Another Some Most Common issues are mentioned below with their **solutions**,
  1. **Check Connections:** Ensure all wires and components are connected correctly. Loose or incorrect connections can lead to errors.
  2. **Power Supply Issues:** Verify that the power supply meets the requirements and is stable. Insufficient power can cause erratic behaviour. i.e. (**Starving for voltage/current**)
  3. **Code Errors:** Review your code for syntax errors, logical issues, or incorrect pin assignments. The Arduino IDE usually provides error messages to help identify issues.
  4. **Serial Monitor:** Use the Serial **Monitor to debug**. Print relevant information to the Serial Monitor to understand how your code is executing and identify potential issues.
  5. **Update Libraries:** Ensure that your libraries are up-to-date. Incompatibility issues can arise if you are using outdated libraries.
  6. **Memory Constraints:** Arduino boards have limited memory. If your code is too large, it may lead to unexpected behavior or failure. Optimize your code or consider upgrading to a more capable board.
  7. **Breadboard Problems:** If you are using a breadboard, check for faulty connections or damaged components.
  8. **Sensor Issues:** Verify that your sensors are functioning correctly. Check for loose connections, damaged wires, or faulty sensors.
  9. **Compatibility:** Ensure that your components (sensors, shields, etc.) are compatible with your board and library versions.
  10. **Firmware Updates:** Keep the firmware on your IoT devices up-to-date. New firmware releases may address known issues and improve stability.
  11. **Driver Issues:** Make sure you have the necessary drivers installed for your board. Incorrect or outdated drivers can cause communication problems.
  12. **Environmental Factors:** Consider external factors like interference, noise, or environmental conditions that might affect your hardware.

For correction/suggestion: - [nikhil.shikare@vit.edu.in](mailto:nikhil.shikare@vit.edu.in) / <https://www.linkedin.com/in/nikhil-shikare-18127a216/>

[nikhilshikare765@gmail.com](mailto:nikhilshikare765@gmail.com)

## 20) Common mistake that Damage Boards/Module

### ➤ Overloading GPIO Pins:

1. **Insufficient Current Handling:** Arduino Uno GPIO pins are designed to provide a limited amount of current, typically around 20mA per pin and up to 200mA in total for all pins combined. Connecting high-current devices such as motors or high-wattage LEDs directly to the GPIO pins without using appropriate driver modules can lead to excessive current draw.
2. **Risk of Damage:** Overloading the GPIO pins with high-current devices can result in overheating, voltage drops, and potentially damage the Arduino Uno. This can lead to the failure of individual pins or, in extreme cases, the entire microcontroller.
3. **Use of Motor Driver or LED Driver:** To safely drive motors, high-power LEDs, or other high-current components, use intermediary modules like motor driver shields or LED driver circuits. These modules are designed to handle the higher current requirements and provide protection to the microcontroller.
4. **Respect Pin Specifications:** Always check the datasheet or specifications of the components you are connecting to ensure they operate within the current limits of the Arduino Uno GPIO pins. If the component requires more current, use external power sources and appropriate driver circuits.
5. **Protective Measures:** Consider incorporating protective measures such as current-limiting resistors or external power supplies to ensure safe operation of both the Arduino Uno and connected components.

### ➤ Wrong voltage level and connecting VCC/Power pin first,

- When working with **voltage sensitive modules** like esp-01 (WI-FI) module (**yes same module which we used with Arduino in chapter 17 last diagram**), **It operates at 3.3v**, connecting with 5v will cause permanent damage Modules, **during projects I have observed many peoples blow up them ( WI-FI(ESP-01) module)** by accidentally **connecting** in +5v pin. Instead of 3.3v. Not only with this. So, take care while connecting voltage/current sensitive module
- One more common **mistake even experienced people do is that is, 1<sup>st</sup> they connect VCC and GPIO pins of module to board and at LAST they connect GND/ground.** **Naver do that, voltage will find ground through GPIO pins and cause potential damage to module as large current will flow directly from power/VCC pin to GPIO pins as ground.** i.e., Without a ground reference, any stray or induced current may **find its way through the GPIO pins as an unintended path to ground**. This can lead to potential damage, especially if the **GPIO pins are not designed to handle significant current flow**.



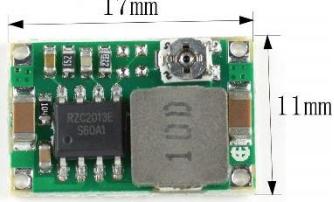
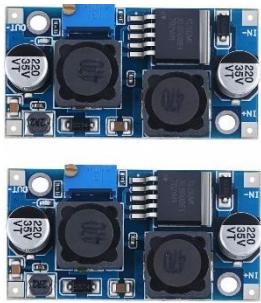
Similarly, at a time of **removing module from board/Microcontroller**, **First remove VCC, then remove GPIO and ground**. To avoid this better power off board supply, and then attach / remove wires/pins.

- Some modules, come with **reverse polarity protection diode**. but not all component have this safety features. So, carefully crosscheck connections, especially **POWER connection (VCC and GND)**

## 21) Voltage Converters modules and IC

- Why we required voltage converter module and IC??
  - For converting high voltage into small voltage and vice versa
  - When we want to power stuff which is not possible to power by Arduino voltage, we use modules, also in battery powered projects, **battery output voltage is not constant**, so when dealing with this situation we require appropriate converters. i.e., they provide **stable output voltage** for voltage sensitive modules like esp-01 when powered using battery's
  - One more scenario is, there are some modules which we can't power using **Arduino standard output voltage** level like GSM module which have **operating voltage in 3.4 to 4.4v.**
  
- Main types of voltage converters: -
  - **Boost Converter:** **Increases** the output voltage. useful when input voltage is low and module operation voltage is high.
  - **Buck Converter:** **Decreases** the output voltage. useful when input voltage is high and module operation voltage is low.
  - **Buck-Boost Converter:** Can step up (boost) or step down (buck) the input voltage.
  
- Some of **most used modules** and **IC** are as follow: - (refer current specification from manufacturer datasheet). and we can **change output current/voltage by rotating pot.**

Module photo	Operating range	Converter Type
 <p><b>MT3608 2A Max DC-DC Step Up Power Module Booster</b></p>	<ul style="list-style-type: none"> <li>➤ Input voltage: <b>2V-24V DC</b></li> <li>➤ Maximum output voltage: <b>5V-28V DC</b></li> </ul>	<b>Boost</b>
 <p><b>XL6009 DC-DC Step-up Module</b></p>	<ul style="list-style-type: none"> <li>➤ Input voltage: <b>3V – 32V DC</b></li> <li>➤ Maximum output voltage: <b>5V-35V DC</b></li> </ul>	<b>Boost</b>

 <p><b>Lm2587 step-up voltage module dc-dc power module boost module</b></p>	<ul style="list-style-type: none"> <li>➤ Input voltage range: 3.5V-30V</li> <li>➤ Output voltage range: 4.0V-30V</li> </ul>	<b>Boost</b>
 <p><b>LM2596 DC-DC Buck Converter Step Down Module</b></p>	<ul style="list-style-type: none"> <li>➤ Input: DC 3V to 40V (input voltage must be higher than the output voltage to 1.5v above cannot boost)</li> <li>➤ Output: DC 1.5V to 35V</li> </ul>	<b>Buck</b>
 <p><b>Buck Module Adjustable 3V 5V 16V Power MP2307 Chip</b></p>	<ul style="list-style-type: none"> <li>➤ Input Voltage: DC 4.75V-23V</li> <li>➤ Output range 1.0V-17V</li> </ul>	<b>Buck</b>
 <p><b>Buck Boost Converter XL6009</b></p>	<ul style="list-style-type: none"> <li>➤ Input voltage: - 5V-32V</li> <li>➤ Output Voltage: - 1.25V-35V</li> </ul>	<b>Buck Boost</b>
 <p><b>REES52 Buck Boost Converter Display, Buck-Boost Board</b></p> <p>Current adjustment (CC) Voltage adjustment (CV) Output indicator Constant current indicator Short press: Switch input/output voltage Hold press: Set power-on default ON/OFF Short press: Switch ON/OFF Hold press: Set power-on default ON/OFF</p>	<ul style="list-style-type: none"> <li>➤ Input voltage: - DC 5.5-30V</li> <li>➤ Output voltage: - DC 0.5-30V</li> </ul>	<b>Buck Boost</b>

➤ **Linear voltage regulator**: - (useful for constant output voltage)

**1) AMS1117 3.3V Power Supply Module**:-

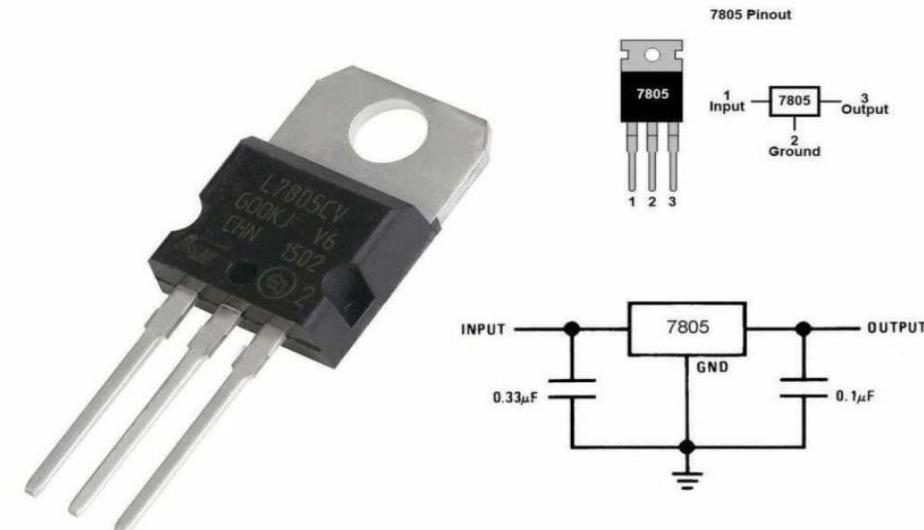
- Arduino boards have a **3.3V output pin** that is **regulated by the AMS1117**. This 3.3V output can be used to power sensors, peripherals, or other devices that require a 3.3V power supply. For example, **WIFI (ESP-01) module** which operate at 3.3v
- Input voltage range of **4.75V to 12V**.
- Output: **3.3V, 800mA** (load current cannot exceed 800ma).
- Onboard power indicator to indicate power is turned on: Red LED



**2) LM7805 IC**:-

- The **LM7805** is a popular **linear voltage regulator** IC that provides **stable 5V** output voltage.
- Input voltage range is **7V to 25V**.
- The LM7805 provides a fixed output voltage of 5V.

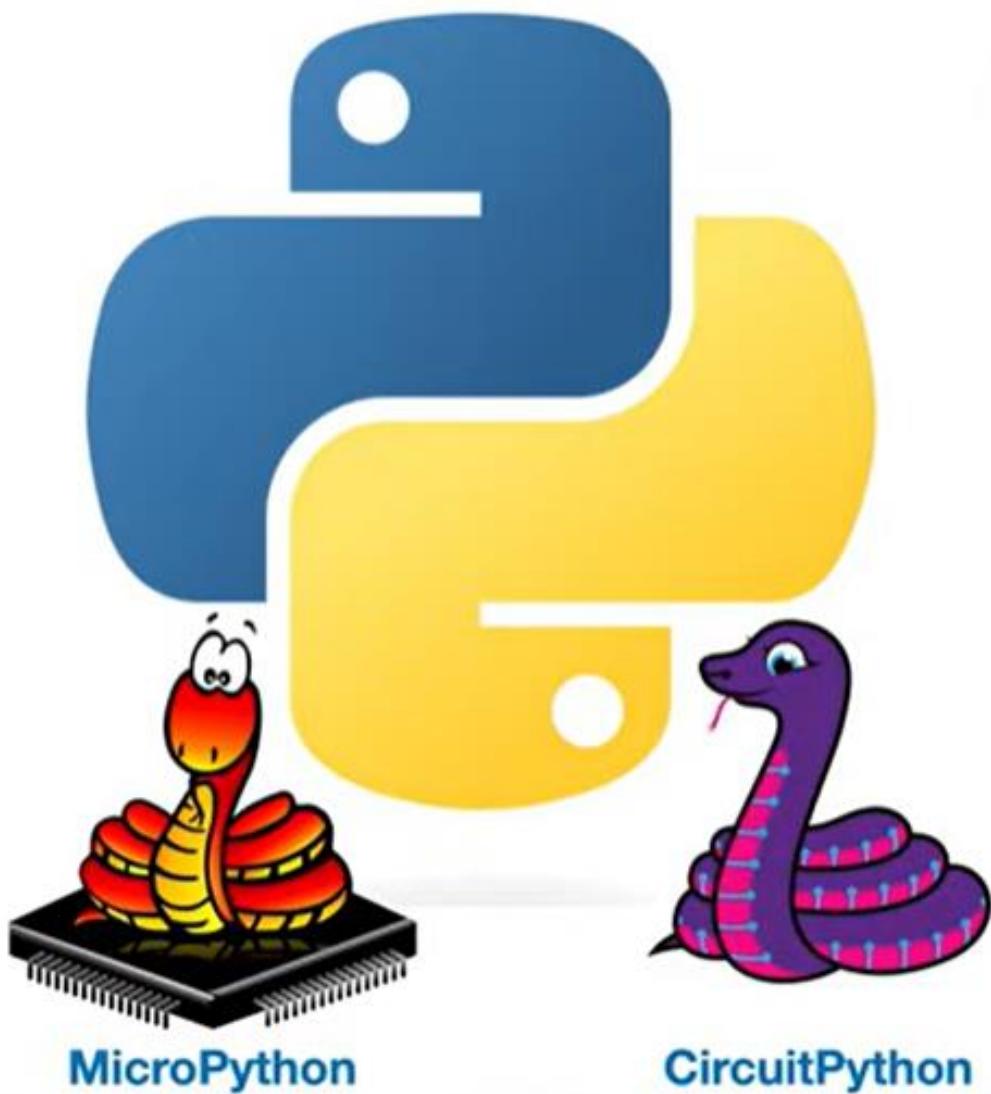
**LM-7805 IC 5V, 1.5A voltage regulator**



Feel free for any correction / suggestions in this document

[nikhil.shikare@vit.edu.in](mailto:nikhil.shikare@vit.edu.in) / <https://www.linkedin.com/in/nikhil-shikare-18127a216/>

# Python



**Note:** - CircuitPython and MicroPython are **both variants of Python** designed for embedded systems, microcontrollers, and IoT applications. They are derived from standard Python (CPython or Cython) and offer simplified syntax and libraries tailored for embedded development both languages are ideal for beginners and experienced developers alike. **CircuitPython is a subset of MicroPython**, sharing similar language syntax and supporting many of the same modules and libraries. Developed by Adafruit Industries, CircuitPython is designed to be beginner-friendly and easy to use. For more information on CircuitPython, please refer to the official documentation. I am covering **MicroPython in this Section.**

For CircuitPython refer online Documentation: - <https://circuitpython.org/>

<https://circuitpython.org/downloads>

<https://docs.circuitpython.org/en/latest/README.html>

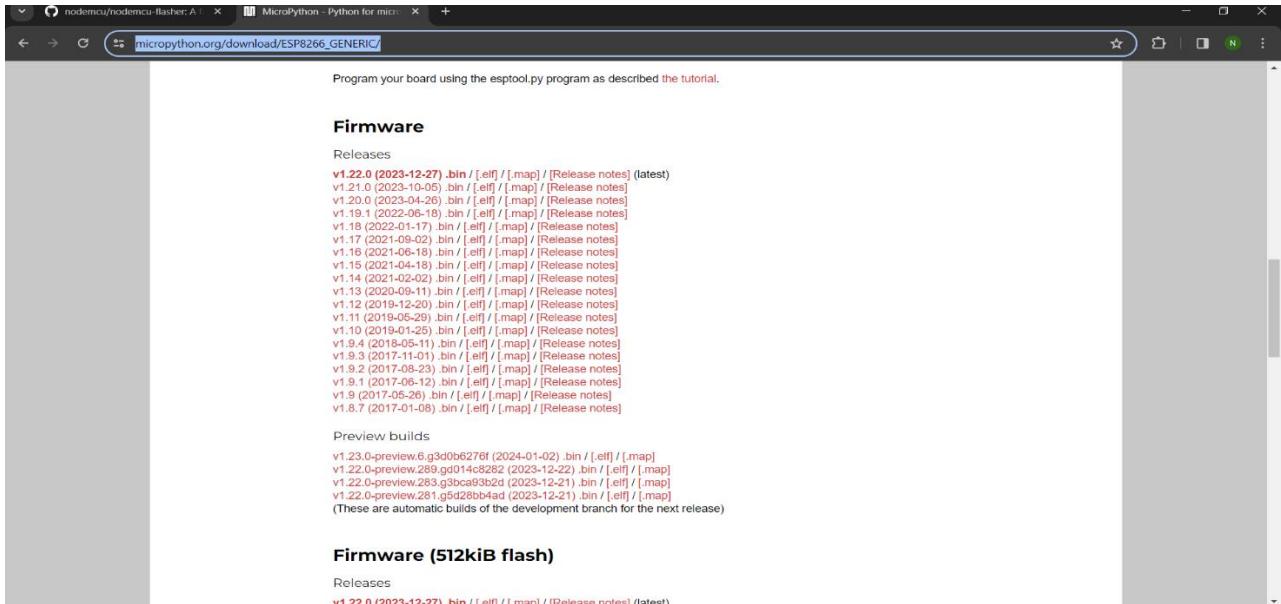
## 23) Exploring Python Variants

There are indeed many different implementations and versions of Python, each catering to specific needs and environments. Here's a more streamlined summary of some of the most important and widely used ones:

1. **CPython:**  **python** <https://www.python.org/>
  - The standard and most widely used implementation of Python, **written in C**.
2. **PyPy:**  **pypy** <https://www.py.py.org/>
  - Known for its performance improvements through **Just-In-Time (JIT) compilation**.
3. **Jython:**  **Jython** <https://www.jython.org/>
  - Runs on the **Java platform**, allowing for seamless integration with Java libraries.
4. **IronPython:**  **IronPython** <https://ironpython.net/>
  - Targets the **.NET framework**, enabling the use of **.NET libraries** in Python.
5. **MicroPython:**  **MicroPython** <https://micropython.org/>
  - Designed for **microcontrollers** and embedded systems, ideal for **IoT projects**.
6. **CircuitPython:**  **circuit python** <https://circuitpython.org/>
  - A variant of **MicroPython tailored by Adafruit** for ease of use in electronics projects.
7. **Cython:**  **Cython** <https://cython.org/>
  - Aims to improve performance by **compiling Python code into C/C++**.
8. **Stackless Python:**  **Stackless Python** <http://www.stackless.com/>
  - Focuses on **concurrency with support for microthreads**, enhancing performance in specific use cases.
9. **Brython**(Browser's Python)-  **BRYTHON** <https://brython.info/>
  - Translates **Python code to JavaScript**, allowing Python to run in web browsers.
10. **Pyston:**  **Pyston** <https://www.pyston.org/>
  - An implementation that uses **JIT techniques** to improve performance, built on CPython.
11. **Nuitka:**  **Nuitka** <https://nuitka.net/>
  - **Compiles Python code into C/C++ executables**, aiming for a blend of Python's simplicity and C's performance.
12. **Transcrypt:**  **Transcrypt**  
*Python in the browser* <https://www.transcrypt.org/>
  - Converts **Python code into JavaScript**, useful for **web development**.
13. **RPython:**  **rpython 0.2.1** <https://rpython.readthedocs.io/en/latest/>
  - A restricted subset of Python used for **writing interpreters**, notably for implementing PyPy
14. **Anaconda Python:** A distribution of Python tailored for **data science and machine learning**, including many popular libraries and tools.  <https://www.anaconda.com/>

## 24) Downloading Micro-python Firmware for NodeMcu

- Visit the Official MicroPython Website <https://micropython.org/download/>
- Choose the Appropriate Board: - For this guide, we will use the NodeMCU. Download the firmware from the following link: MicroPython for ESP8266. You will see different firmware options like firmware 512KiB, 1MiB, Firmware OTA Compatible, etc.
- Download the Suitable Firmware: - (refer below image)



- Flashing MicroPython Firmware on NodeMCU: -  
(There are different ways to flash MicroPython firmware): -

### 1. Using Python esptool.py

- First, install Python if you haven't already.
- Then, install esptool using the following command:

```
pip3 install esptool
```

Refer to the steps on the official website for detailed instructions: [MicroPython ESP8266 Tutorial](https://docs.micropython.org/en/latest/esp8266/tutorial/intro.html) <https://docs.micropython.org/en/latest/esp8266/tutorial/intro.html>

### 2. Using NodeMCU Py-Flasher

- Download and install NodeMCU Py-Flasher from this link: [NodeMCU Py-Flasher](https://github.com/nodemcu/nodemcu-flasher)
- <https://github.com/nodemcu/nodemcu-flasher>
- Refer to the upcoming chapter for detailed steps.

### 3. Using Thonny IDE

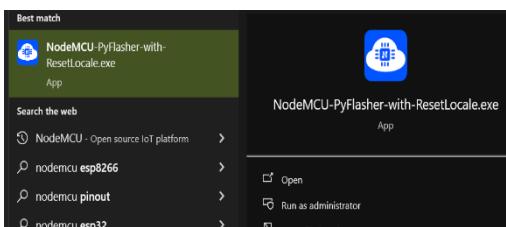
- Thonny IDE can be used for REPL and uploading MicroPython code.
- Download Thonny IDE from this link: [Thonny IDE](https://thonny.org/) <https://thonny.org/>

### 4. Other Methods

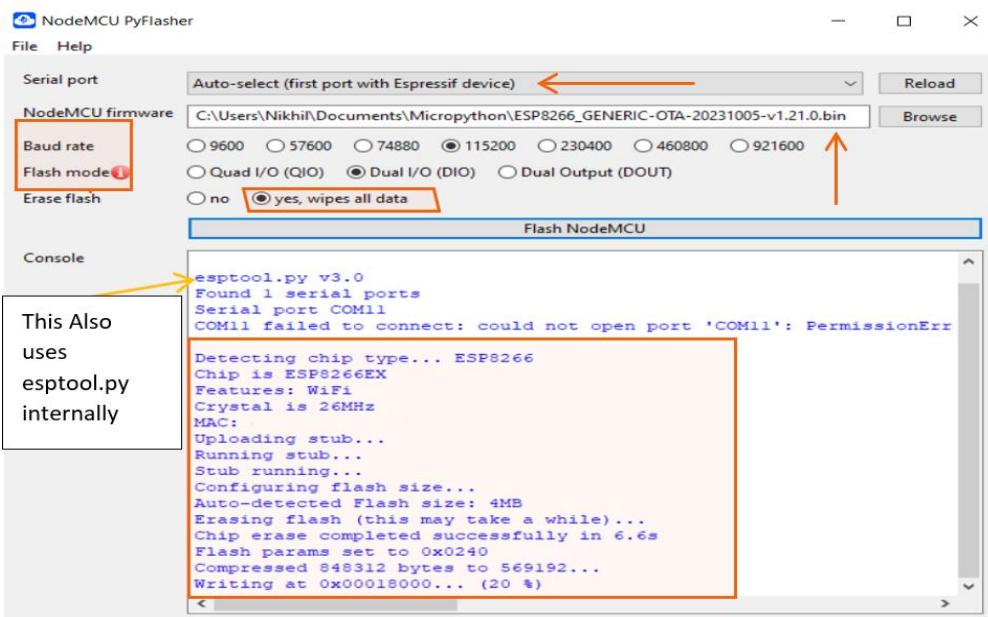
- There are various other methods for flashing MicroPython firmware. We will explore some of these methods in this guide.

## 25) Flashing NodeMcu using NodeMcu-Flasher

a) Installation and Initial Setup (After installing NodeMCU-Flasher, you will see the following options):



b) Selecting Firmware File (browse for the ".bin" firmware file on your computer that you downloaded from the MicroPython website.)



c) Selecting Serial Port (Select the serial port. If the default option does not work, change it)

d) Browsing for Firmware File

e) Erase Flash Option (Make sure to set "Erase flash" to "yes, wipe all data.")

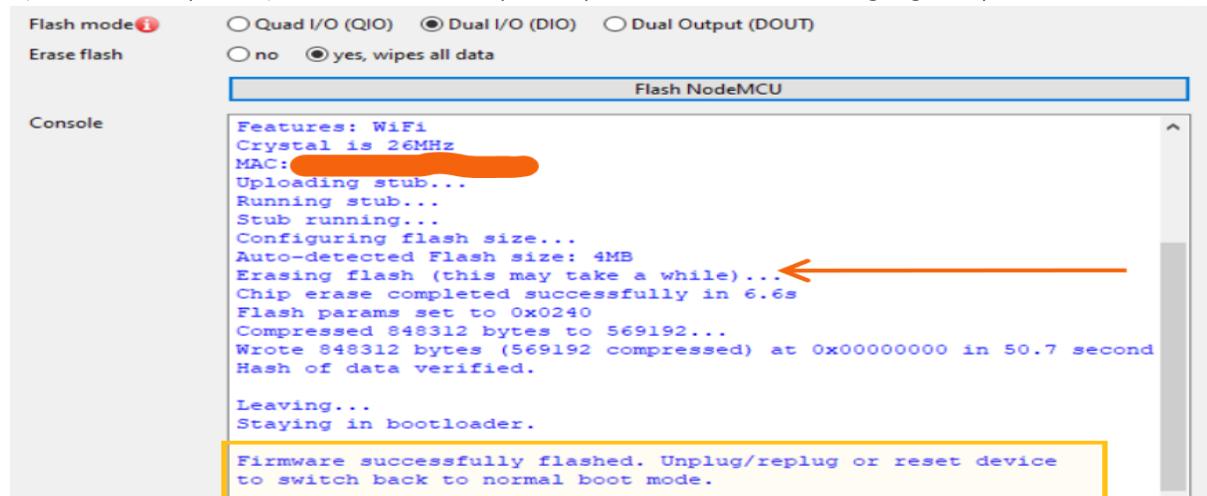
f) Baud Rate and Flash Mode

Usually, the default settings for the baud rate and flash mode work fine, so leave them as they are.

g) Flashing the Firmware

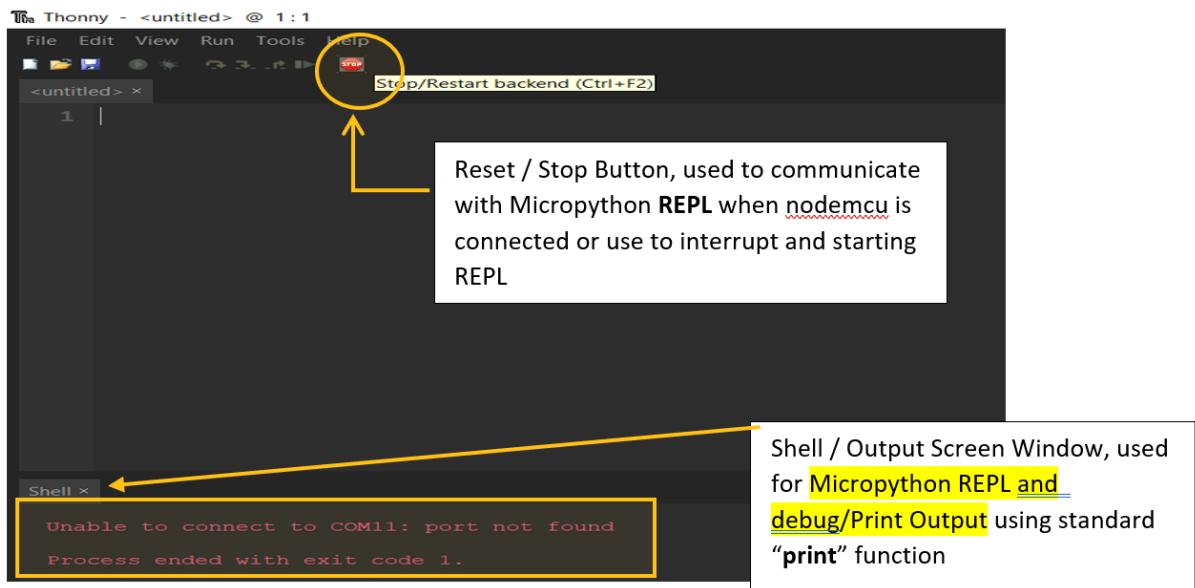
Ensure that while inserting the NodeMCU into the USB port, you **“press and hold the FLASH button”**. Otherwise, it will just say **“connecting”** and then **“failed to connect.”**

h) Successful Update (After a successful update, you will see the following logs output on the screen)



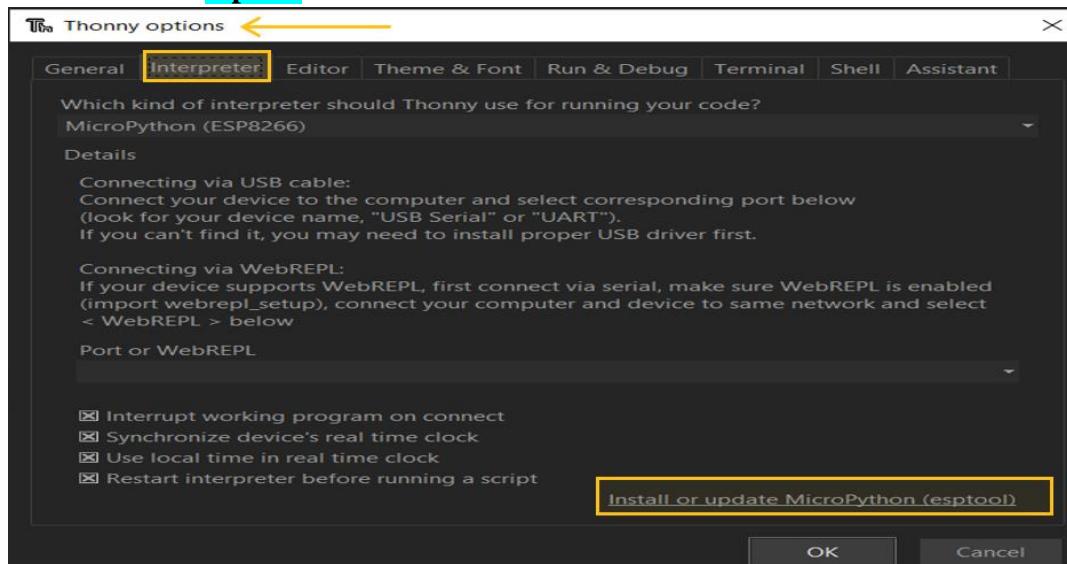
## 26) Flashing NodeMcu using Thonny IDE

- a) Installation and Setup (After downloading Thonny IDE, you will see the following options)



- b) Setting Up the IDE: -

- 1) Go to Tools > Options.
- 2) Select Interpreter.
- 3) Click on "Install or update MicroPython (esptool)". Internally Thonny IDE uses the same "esptool" to flash the NodeMCU.



- c) Flashing Process: -

- 1) In Thonny IDE, you don't need to specify the MicroPython ".bin" firmware file. It will show the latest firmware version in the "Select version" option. However, this may be a disadvantage as you cannot flash different versions like 512KiB, 1MiB, OTA updates, or old versions of firmware as you can with NodeMCU Py-Flasher. (This may change in future versions.)
- 2) Keep "Erase all flash" marked. Change the flash size to "detect."
- 3) Change the flash mode to "dio" which is the same as NodeMCU Py-Flasher.

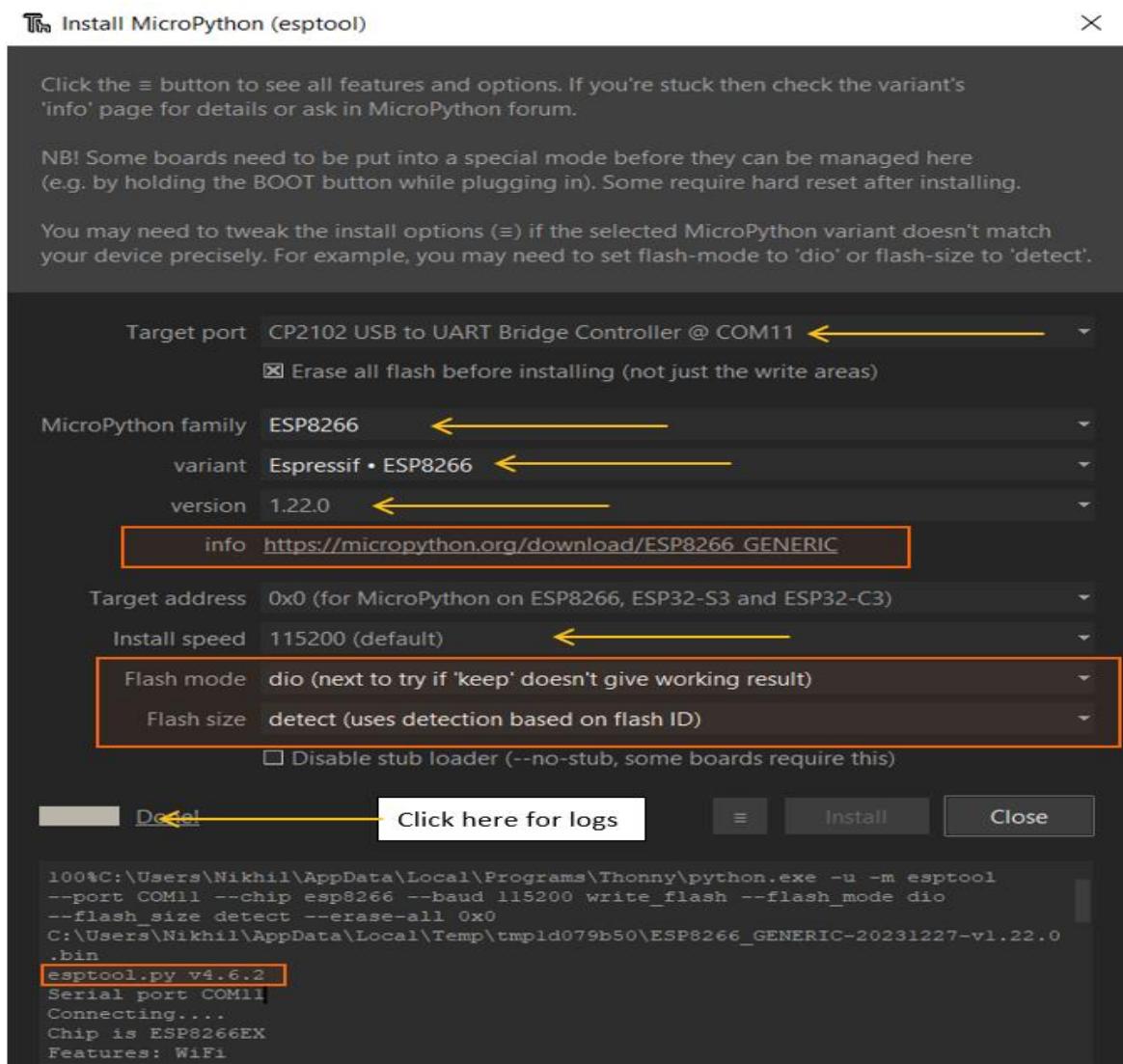
- d) Troubleshooting Installation Errors: -

- 1) If you encounter an installation error or the onboard LED starts blinking rapidly and prints garbage values on the Serial monitor (Shell) at a very high speed, disconnect the board and reconnect the USB while pressing the FLASH button

on the NodeMCU. Retry by changing default options like "Installation Speed" or "Flash mode."

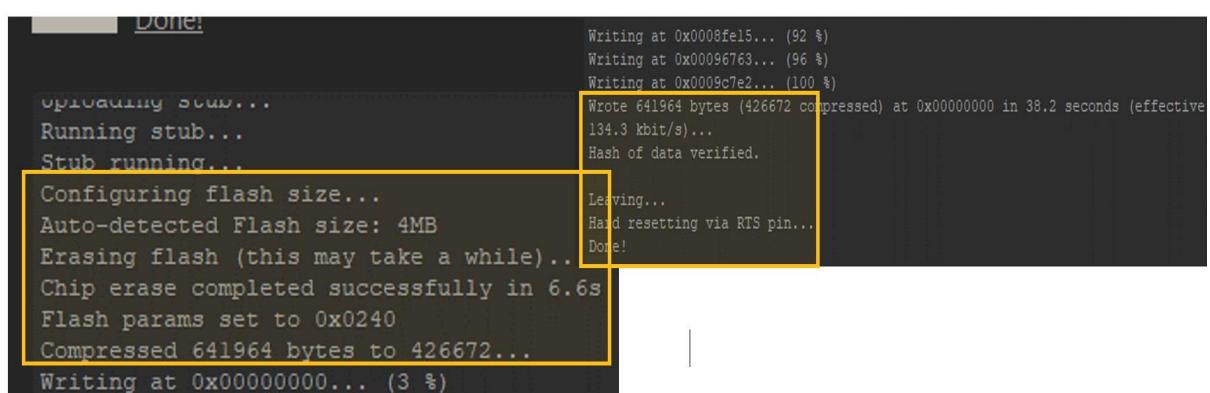
- 2) If the firmware is not flashing, press and hold the "FLASH" button on the NodeMCU and release it after inserting the USB.

e) Refer below screenshot: -



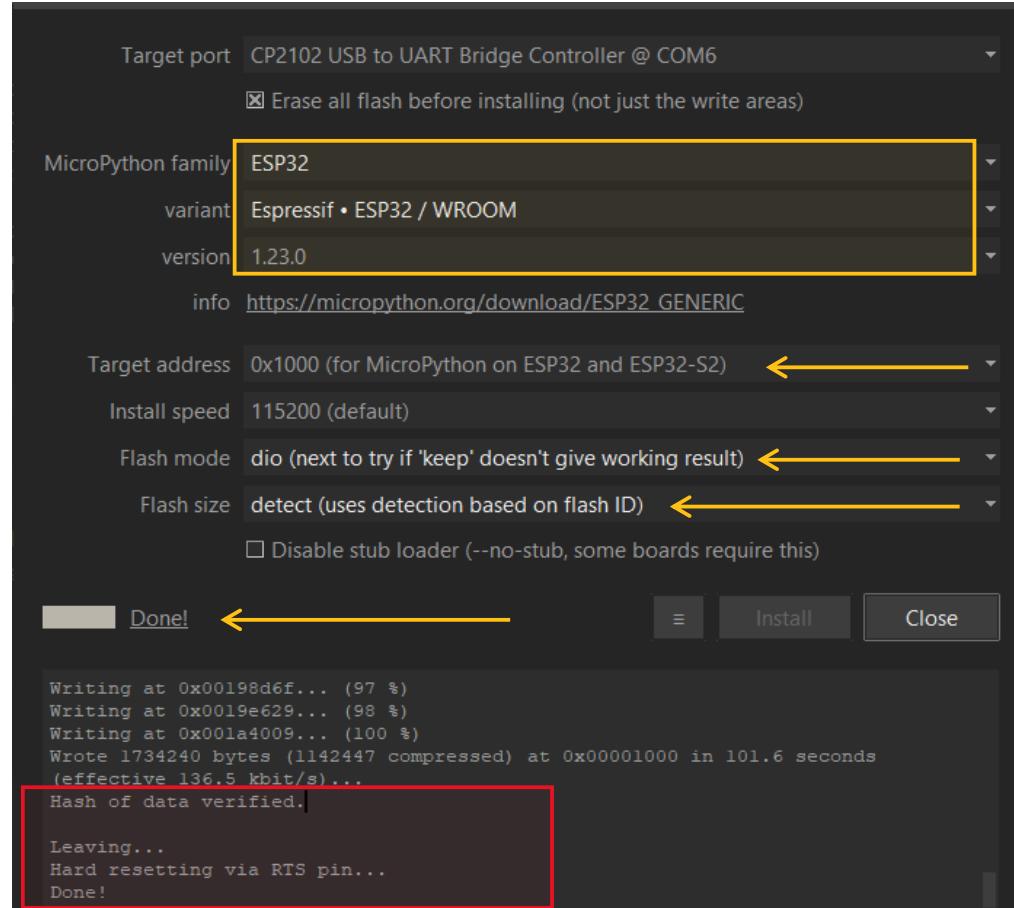
f) Verifying Successful Installation: -

(After a successful installation, you will see common logs in most / all of the above tools. This is because most of them internally use the common Python tool "esptool.py" for flashing.)



**Note:** - Procedure is Almost same for ESP32 MCU , Some minor changes are shown below.

➤ Flashing with Thonny-IDE :-



➤ After successful Flashing , Interaction via MicroPython REPL :-

```

MicroPython v1.23.0 on 2024-06-02; Generic ESP32 module with ESP32
Type "help()" for more information.

MicroPython v1.23.0 on 2024-06-02; Generic ESP32 module with ESP32
Type "help()" for more information.
>>> help("modules")
__main__           bluetooth      heapq          select
 asyncio           btree          inisetup       socket
 boot              builtins       io             ssl
 espnow             cmath          json           struct
 onewire            collections   machine        sys
 thread             cryptolib     micropython   time
 webrepl            deflate        mip/_init_    tls
 aioespnow          dht           neopixel      uasyncio
 apa106             ds18x20      network       uctypes
 array              errno          ntptime       umqtt/robust
 asyncio/_init__    esp            onewire      umqtt/simple
 asyncio/core      esp32          os             upysh
 asyncio/event     espnow         platform      urequests
 asyncio/funcs     flashbdev    random        vfs
 asyncio/lock      framebuffer  re             webrepl
 asyncio/stream    gc             requests/_init_ websocket
 binascii            hashlib       re
 Plus any modules on the filesystem
>>> |

```

Some important library are marked above with arrows.

## 27) REPL and Micropython interaction via Shell

### a) Connecting to MicroPython REPL: -

After a successful installation, you can connect to the MicroPython Read-Eval-Print Loop (REPL) by pressing the "Start/Stop" button, which is a red button in Thonny IDE.

```

Shell x

Unable to connect to COM1: port not found
Process ended with exit code 1.

MicroPython v1.22.0 on 2023-12-27; ESP module with ESP8266
Type "help()" for more information.

MPY: soft reboot
MicroPython v1.22.0 on 2023-12-27; ESP module with ESP8266
Type "help()" for more information.
>>>

```

### b) Executing MicroPython Commands: -

Start executing MicroPython commands, similar to using the Python IDLE. The main advantage is that you can test commands or code **without repeatedly uploading** code to the NodeMCU board. (In the Arduino IDE, you have to flash the code to test even small changes.)

### c) Handling Keyboard Interrupt: -

Press "**Ctrl+C**" for a keyboard interrupt. For example, if you are running an infinite while loop in your code (e.g., in main.py or in the shell), it won't allow communication with the REPL or uploading new code to the NodeMCU. In this case, using the **"red Start/Stop"** button or "Ctrl+C" for a keyboard interrupt can help re-establish communication.

```

MPY: soft reboot
MicroPython v1.22.0 on 2023-12-27; ESP module with ESP8266
Type "help()" for more information.
>>> print("Hii Nikhil Mohan Shikare")
Hii Nikhil Mohan Shikare
>>> import network , os
>>> os.listdir("/")
['boot.py']
>>> for i in range(3):
    print(i)

0
1
2
>>> with open("boot.py" , "r") as file:
    file.read()

'# This file is executed on every boot (including wake-boot from deepsleep)\n#import esp\n#RT(0)\nimport gc\n#import webrepl\n#webrepl.start()\nngc.collect()\n'
>>> |

```

## Before we proceed, we shood know some micro python tools and concepts

➤ **MIP:** -

mip is the MicroPython package installer (**mip stands for "MicroPython package installer"**). It can install MicroPython packages from various sources, including micropython-lib, GitHub, or arbitrary URLs. mip can be used on-device on network-capable boards and internally by tools such as mpremote. For more information on using mip, refer to the Package management section.

- **micropython-lib** -

<https://docs.micropython.org/en/latest/reference/glossary.html#term-micropython-lib>

- **mpremote** -

<https://docs.micropython.org/en/latest/reference/glossary.html#term-mpremote>

- **Package management** -

<https://docs.micropython.org/en/latest/reference/packages.html#packages>

➤ **.mpy file:** -

The .mpy file is the output of the cross-compiler. It is a compiled form of a .py file that contains MicroPython bytecode instead of Python source code.

- **cross-compiler** -

<https://docs.micropython.org/en/latest/reference/glossary.html#term-cross-compiler>

- **.py file** - <https://docs.micropython.org/en/latest/reference/glossary.html#term-.py-file>

- **bytecode** - <https://docs.micropython.org/en/latest/reference/glossary.html#term-bytecode>

➤ **Extending MicroPython Using C MicroPython Wrapper:** -

- 1) To extend MicroPython with custom functionality written in C, you can use the MicroPython C wrapper concept. This allows you to write C functions and expose them as callable functions in MicroPython. This is **useful for performance-critical code, interfacing with hardware, or using existing C libraries**
- 2) **MicroPython C Wrapper:** MicroPython C wrapper enables the integration of C functions into the MicroPython environment. By using this approach, you can leverage the performance benefits of C while maintaining the ease of use and flexibility of Python
- 3) The process typically involves writing C code, defining the wrapper, and compiling it into a format that MicroPython can use.
- 4) The steps involve: - Writing C Function, Wrapper Definition, Compilation, Loading and Using in MicroPython
- 5) For more details, please refer official docs: -
 

<https://docs.micropython.org/en/latest/develop/extendingmicropython.html>

## 28) Installing external Micropython library's

➤ **Checking Available Modules:** -

Check available modules in the MicroPython firmware using - `help ("modules")`

➤ **Using MicroPython Package Managers:** -

Similar to standard **Python's "PIP," MicroPython has "mip" and "upip"** package manager and installer modules.

➤ **Installing Modules:** -

To install modules like "base64" and "binascii," you can use either "upip" or "mip." However, "upip" has been replaced by "mip," which offers several advantages.

➤ Example using mip MicroPython Package Manager:

```
">>> mip.install("binascii")
```

Installing binascii (latest) from <https://micropython.org/pi/v2> to /lib

Exists: /lib/binascii.mpy

Done

```
>>> mip.install("base64")
```

Installing base64 (latest) from <https://micropython.org/pi/v2> to /lib

Exists: /lib/binascii.mpy

Exists: /lib/struct.mpy

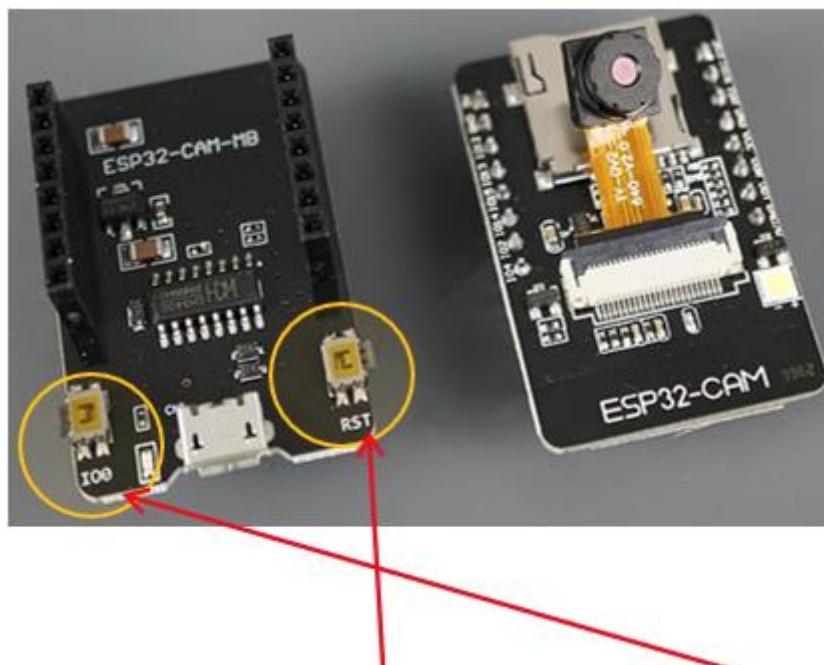
Exists: /lib/base64.mpy

Done"

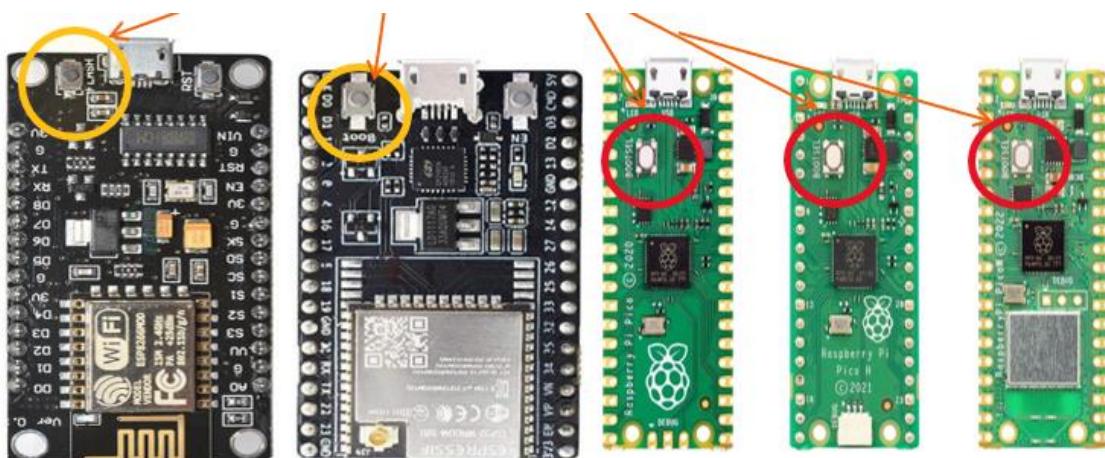
```
>>> mip.install("binascii")
Installing binascii (latest) from https://micropython.org/pi/v2 to /lib
Exists: /lib/binascii.mpy
Done
>>> mip.install("base64")
Installing base64 (latest) from https://micropython.org/pi/v2 to /lib
Exists: /lib/binascii.mpy
Exists: /lib/struct.mpy
Exists: /lib/base64.mpy
Done
```

## 29) Installing ESP32-CAM Firmware

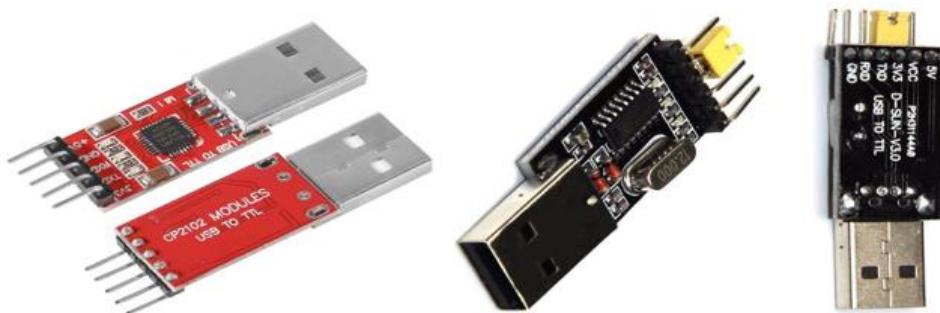
- 1) The procedure is similar to NodeMCU firmware flashing. You can use any of the tools mentioned above with slight changes for ESP32-CAM.
- 2) On the MicroPython official site, you might not find firmware specifically for the ESP32-CAM. However, there are multiple GitHub repositories that provide customized ESP32 firmware for the **ESP32-CAM** with additional support for the camera.
- 3) The recommended tool for flashing the ESP32-CAM MCU is `esptool.py`, a CLI-based Python tool. Please refer to [Chapter 2](#) for instructions on how to use this tool.
- 4) For flashing the ESP32-CAM, you will need a **USB to TTL converter or a USB converter board**. Refer to the image for an example.



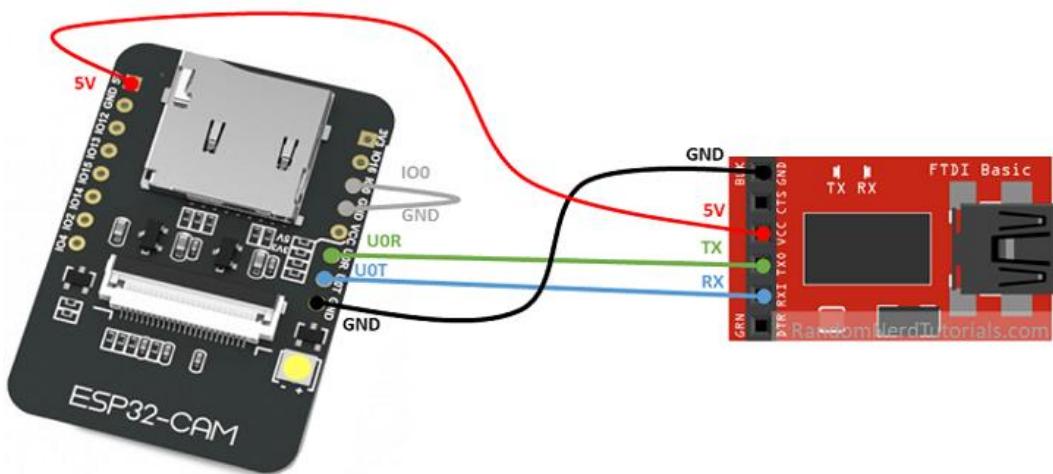
- 5) Make sure the converter has both a **"RST" restart button and an "IO0" button**. To flash the ESP32-CAM, you will need to hold the "IO0" button and insert or plug in the USB to the computer to enter boot mode (**tell esp32cam board that we are going to flash it. (boot mode)**). Some boards only have a reset button, which is fine if you are using the Arduino IDE.
- 6) Another option is to use hardware serial I/O, such as UART RX and TX pins, which works for most types of boards, including ESP-01 or ESP series boards, as well as for flashing MCUs like NodeMCU/ESP32-CAM. Some boards, like the NodeMCU , raspberry pico w, already have a "flash" button to simplify this process.



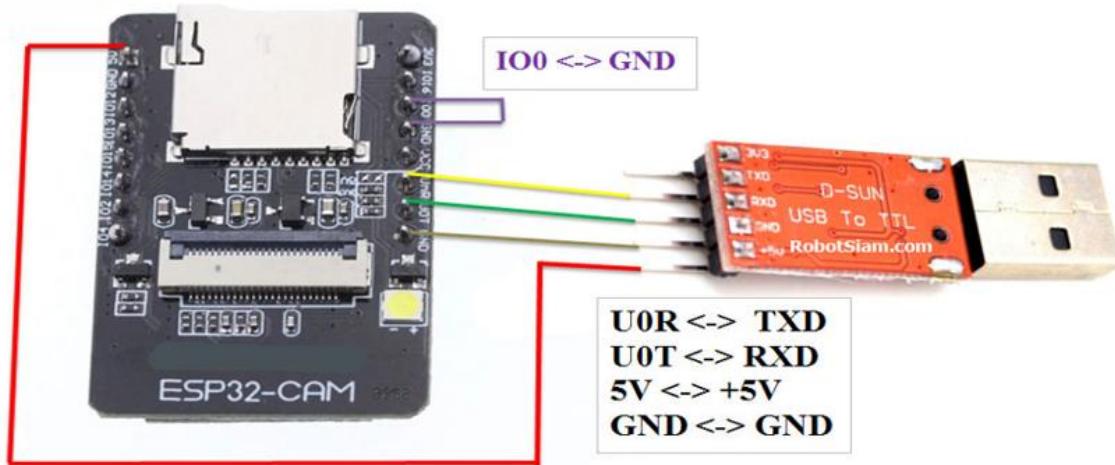
- 7) Use USB to TTL converters, which also have the advantage of a "power shifter" that allows you to connect boards with different power levels to the PC.



- 8) Connect the ESP32-CAM as shown in the image. **Before plugging it into the PC, make sure to short IO0 to ground to enter boot mode.** After inserting the USB, remove this connection after 3-4 seconds. When the program is uploaded, make sure to **remove** this connection before restarting; otherwise, it will remain in boot mode.



i.e. with different TTL converter Connection are remain same,



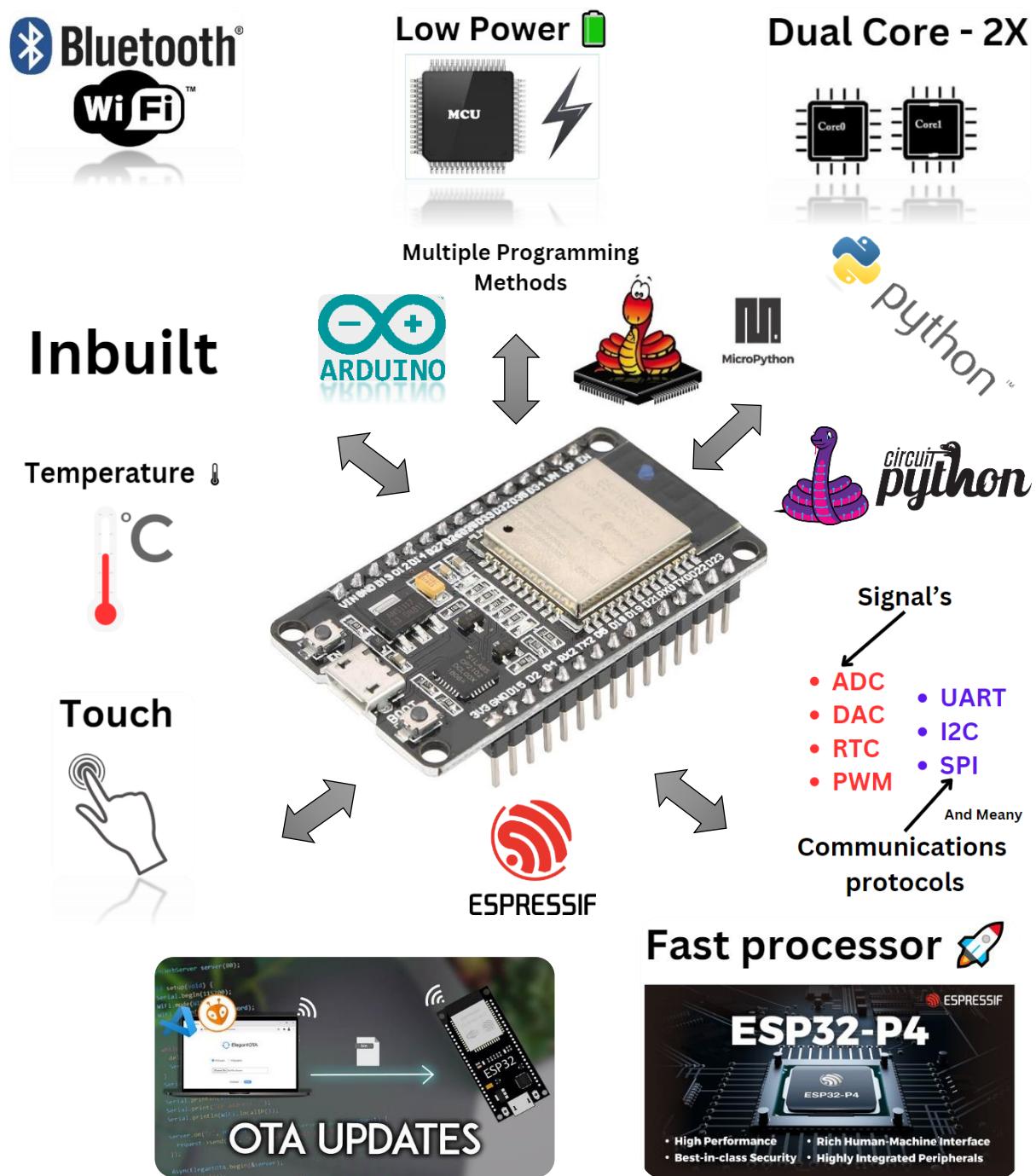
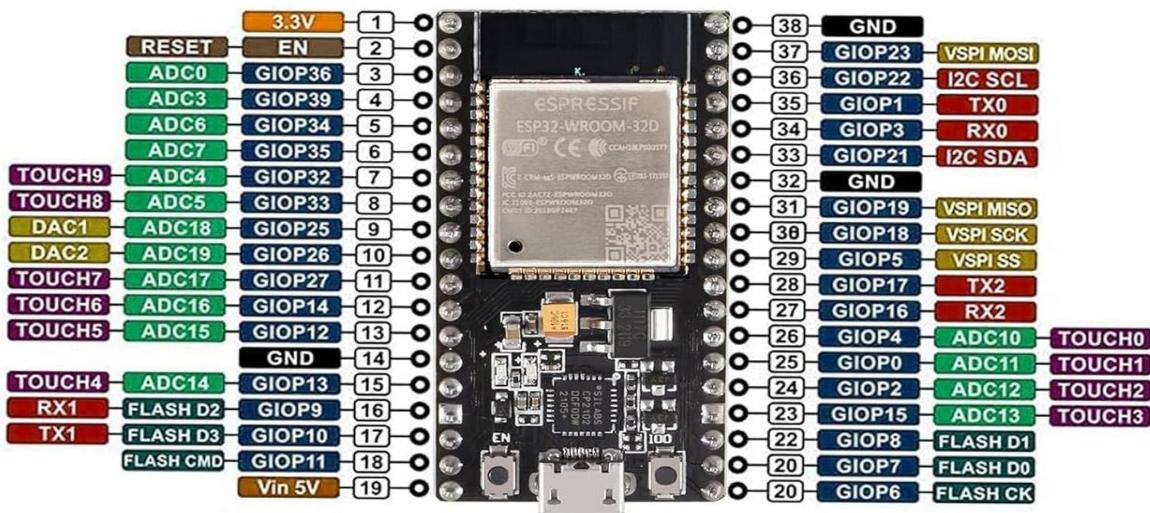
- 9) Download the firmware from GitHub and flash it using “`esptool.py`” `python` tool.  
Firmware GitHub URL: [MicroPython-ESP32-CAM/firmware at main · e135193/MicroPython-ESP32-CAM](https://github.com/e135193/MicroPython-ESP32-CAM)

## 30) Syntax Comparison - Arduino C And Micropython

(some basic syntax comparison table between common operations in Arduino C and MicroPython referred from chatgpt, please **study this table** carefully, before proceeding...)

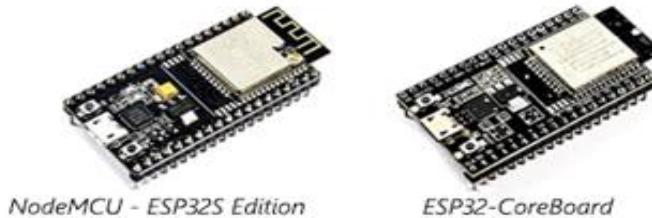
Comparaison	Arduino C	Micropython
Include Libraries	#include <library_name.h>	import <module>
Setup	void setup() { /* initialization code */ }	def setup(): /* initialization code */
Loop	void loop() { /* main code */ }	while True: /* main code */
Print to serial	Serial.begin(9600);  Serial.print("text");	from machine import UART uart = UART(0, 9600)  print("text")
Digital pin write	pinMode (13, OUTPUT);  digitalWrite(13, HIGH);	from machine import Pin pin = Pin(13, Pin.OUT)  pin.value(1)
Digital pin read	pinMode(2, INPUT);  int state = digitalRead(2);	pin = Pin(2, Pin.IN)  state = pin.value()
Analog pin read	int value = analogRead(A0);	from machine import ADC adc = ADC(Pin(36)) value = adc.read()
Delay	delay(1000);	import time time.sleep(1)
PWM Output	analogWrite(9, 128);	from machine import PWM pwm = PWM(Pin(9)) pwm.duty(512)
I2C Initialization	Wire.begin();	from machine import I2C i2c = I2C(0)
I2C Read/Write	Wire.requestFrom(0x48, 2); Wire.read();	i2c.readfrom(0x48, 2) i2c.writeto(0x48, b'\x00')
SPI Initialization	SPI.begin();	from machine import SPI spi = SPI(1)
SPI Read/Write	SPI.transfer(val);	spi.read(1) spi.write(b'\x00')
Software Serial	#include <SoftwareSerial.h> SoftwareSerial mySerial(10, 11);	from machine import UART uart = UART(1, baudrate=9600, tx=Pin(10), rx=Pin(11))
Interrupts	attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);	pin.irq(trigger=Pin.IRQ_RISING, handler=callback_function)
Using an external library	#include <LibraryName.h>	import library_name

### **31) Esp-32 Features and Info.**



## 32) ESP32 – Various Versions

- There are various new versions of ESP32 launched by **Espressif**. Some of them include: (**Also, Refer Chapter - 8**)



- It is not possible to cover all board's here. Explore more about this boards.



### 33) ESP32 cam QR scanner

#### Overview of approach

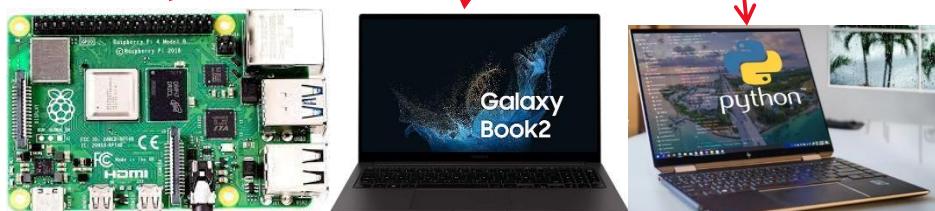
**Note:** The following MicroPython programs are self-explanatory.

1. We have used the “socket” concept and programmed two sockets: **UDP** and **TCP**.
2. Make sure to use the **UDP** socket. While **TCP** is also available, it is not recommended for this purpose. It overloads and overheats CPU of esp32 cam MCU, if multiple simultaneous sender socket connections are there.
3. The sender code (**esp32 cam side**) is written in MicroPython using sockets, and the **receiver server** is coded in standard Python which will run either on laptop or raspberry pi with some OS like installed in it. The server decodes images and processes data using a QR code library received from the MicroPython sender.
4. If possible, we will scan and decode QR data on the ESP32-CAM itself using ".mpy" files.

#### ➤ Receiver (Server) Side Code: - (UDP socket receiver code)

The receiver (server) side code is responsible for receiving the **frame buffer from the ESP32-CAM** and processing it on a Raspberry Pi or laptop using standard Python with a QR code library.

**Code: -**



```
import cv2
import numpy as np
from pyzbar.pyzbar import decode
import socket
import struct
def process_image(image_data):
    try:
        image_np = np.frombuffer(image_data, dtype=np.uint8)
        image = cv2.imdecode(image_np, cv2.IMREAD_GRAYSCALE)
        if image is not None and image.size > 0:
```

```

cv2.imshow("ESP32-CAM Stream", image)
cv2.waitKey(1) # Needed to update the display (1 ms delay)
codes = decode(image)
if len(codes) > 0:
    for code in codes:
        print("Decoded QR code:", code.data.decode('utf-8'))
except Exception as e:
    print(f"Error processing image: {e}")

def start_server(host, port):
    # Create a UDP socket
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_socket.bind((host, port)) # Bind the socket to the host and port
    print(f"Server listening on {host}:{port}")

try:
    while True:
        # Receive data from a client
        data, client_address = server_socket.recvfrom(65507) # Maximum
        UDP packet size
        #print(f"Received data from {client_address}")
        # Extract the size of the image data (as a 4-byte integer, big-
        endian)
        size = struct.unpack("!I", data[:4])[0]

        image_data = data[4:4 + size] # Extract the image data
        #print(image_data)
        process_image(image_data) # Process the received image data

except KeyboardInterrupt:
    print("UDP Server stopped by user.")
finally:
    server_socket.close()
    cv2.destroyAllWindows()

# Start the server
start_server("0.0.0.0", 8765)

```

➤ **Sender (Esp32 cam) Side Micropython Code:** - (UDP socket sender code)

The sender (esp32 cam) MicroPython side, it sends frame buffer for processing to server



**Code: -**

```

import camera
import network
import socket
import time
import machine
import struct

def connect_to_wifi(ssid, password):
    wifi = network.WLAN(network.STA_IF)
    wifi.active(True)
    try:
        wifi.connect(ssid, password)
        while not wifi.isconnected():
            pass
        print("Connected to Wi-Fi")
        return True
    except OSError as e:
        print("Error connecting to Wi-Fi:", e)
        return False

wifi_ssid = "Mohan samsung"
wifi_password = "mohan12345"

if not connect_to_wifi(wifi_ssid, wifi_password):
    pass # Add Error handling , if failed to connect wifi..

# Set up the server address (Raspberry Pi's IP address) and port
server_ip = "192.168.79.153" # Replace with your Raspberry Pi's IP address
server_port = 8765 # Replace with your desired port number
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # Create a UDP socket

# Initialize the camera
try:
    camera.init(0, format=camera.JPEG, framesize=camera.FRAME_HVGA,
    fb_location=camera.PSRAM)
    camera.saturation(1)
    camera.brightness(1)

```

```

camera.contrast(1)
camera.quality(15)
print(f"Camera init..")
except Exception as e:
    print(f"An error occurred: {e}")
    time.sleep(2)
    print("Hold on..Attempting to Restart...")
    print("Restarting MCU...in less than 7 Sec...")
    time.sleep(5)
    machine.reset()

print("going to loop..")
while True:
    buf = camera.capture() # Capture an image
    #buf = "Hii from esp32 cam..."
    size = len(buf)
    try:
        #print(size)
        # Send the size as a 4-byte integer (big-endian) followed by the image
data
        sock.sendto(struct.pack("!I", size) + buf, (server_ip, server_port))
    except Exception as e:
        print(f"Failed.. to send data -> {e}")
    buf = None
    time.sleep(0.1)

# Close the socket when done
sock.close()
print("End of program..")

```

For any correction/suggestion/feedback: - Email: - [nikhil.shikare@vit.edu.in](mailto:nikhil.shikare@vit.edu.in)



Linkdin : - <https://www.linkedin.com/in/nikhil-shikare-18127a216/>



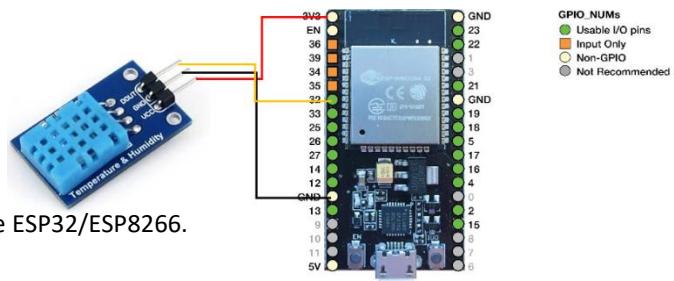
## 34) ESP32 / ESP8266 interfacing DHT Sensor (GPIO)

➤ Components Needed:-

- 1) ESP32 or ESP8266.
- 2) DHT11 / DHT22.
- 3) Jumper wires.

➤ Connection:-

- 1) VCC: Connect to the 3.3V pin on the ESP32/ESP8266.
- 2) GND: GND pin.
- 3) DATA: Connect to GPIO 32 (Your Preferred any GPIO).



➤ Code:-

- 1) Note: - we don't need to install any external module from mip or github. As there is already "dht" module is present on MicroPython Firmware (Refer below screenshot)

```
MicroPython v1.23.0 on 2024-06-02; Generic ESP32 module with ESP32
Type "help()" for more information.

>>> help("modules")
_main_
 asyncio
_boot
 espnow
 onewire
_thread
_webrepl
 aioespnow
 apa106
 array
 asyncio/_init_
 asyncio/core
 asyncio/event
 asyncio/funcs
 asyncio/lock
 asyncio/stream
 binascii
 bluetooth
 btree
 builtins
 cmath
 collections
 cryptolib
 deflate
 ds18x20
 errno
 esp
 esp32
 espnow
 flashbdev
 framebuffer
 gc
 hashlib
 heapq
 inisetup
 io
 json
 machine
 math
 micropython
 mip/_init_
 neopixel
 network
 ntptime
 onewire
 os
 platform
 random
 re
 requests/_init_
 select
 socket
 ssl
 struct
 sys
 time
 tls
 uasyncio
 uctypes
 umqtt/robust
 umqtt/simple
 upysh
 urequests
 vfs
 webrepl
 webrepl_setup
 websocket

Plus any modules on the filesystem

>>> import dht
>>> help(dht)
object <module 'dht' from 'dht.py'> is of type module
DHTBase -- <class 'DHTBase'>
__version__ -- 0.1.0
__name__ -- dht
__file__ -- dht.py
sys -- <module 'sys'>
DHT11 -- <class 'DHT11'>
dht.readinto -- <function>
DHT22 -- <class 'DHT22'>
```

Contains Both DHT11 and DHT22 Classes. In same module.

Document by: - Nikhil Mohan Shikare.  
Email: - nikhil.shikare@vit.edu.in

2) Program Code:-

```
from machine import Pin
import dht
import time

# Initialize DHT11 sensor
dht_pin = Pin(32) # Change to your GPIO pin
sensor = dht.DHT11(dht_pin)
# sensor = dht.DHT22(dht_pin) #For DHT22

while True:
    try:
        sensor.measure() # Read sensor values
```

```

temperature = sensor.temperature()
humidity = sensor.humidity()

# Print values to console
print('Temperature: {}°C'.format(temperature))
print('Humidity: {}%'.format(humidity))

except OSError as e:
    print('Failed to read sensor.')

time.sleep(2) # Wait for 2 seconds before reading again

```

- **Output:** - DHT22 have high accuracy over DHT11

```

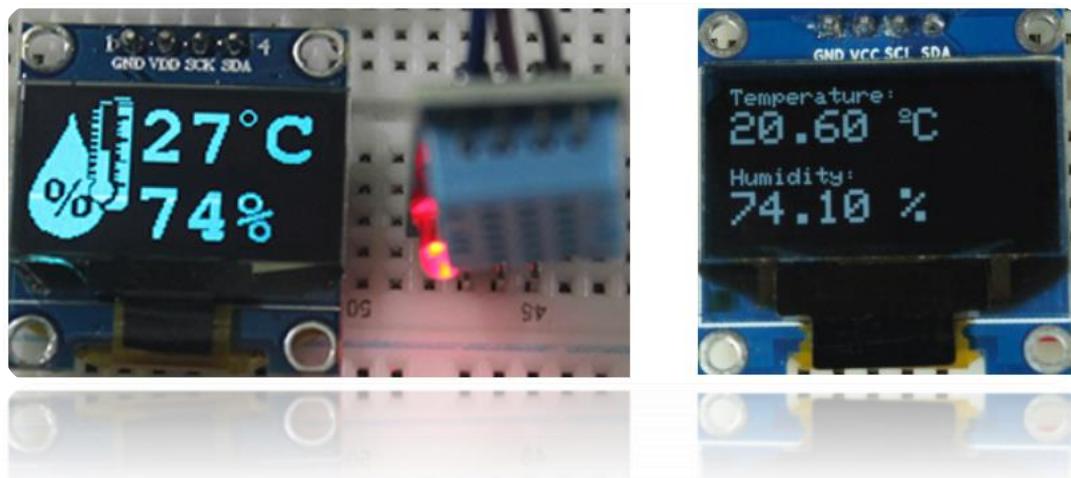
Temperature: 32.17 °C
Humidity: 69.71 %
Temperature: 32.00 °C
Humidity: 69.71 %
Temperature: 31.52 °C
Humidity: 69.65 %
Temperature: 31.92 °C
Humidity: 70.95 %
Temperature: 32.10 °C
Humidity: 69.91 %

```

- **Challenge/Assignment Task:** -

- 1) Now you have studied DHT11/22 Sensor interface and printing of values in **Micropython** Console using ESP32/8266.
- 2) Refer **Chapter Number 33**, Learn Basic of OLED display , Like Displaying Images using Framebuffer , printing various shapes like Circle and Rectangle on varios OLED Display Driver
- 3) After this, Interface DHT11/22 along with OLED ,**take reading and show on OLED display.**
- 4) Refresh OLED display after some intervals , also if possible **try to create moving animations on it, which show realtime bar graph animations of Temperature and Humidity** , It is covered in OLED chapter 33.

- **Some Reference Images From Internet :-**



## 35) Esp32 / ESP8266 implementing MQTT for Above

(Note: - There are Lot of Advanced concept on MQTT, but we will see basic.)

➤ **Components Needed:** -

- 1) ESP32 or ESP8266.
- 2) DHT 11 / 22
- 3) Jumper Wires.

➤ **Protocol: - MQTT (Message Queuing Telemetry Transport)-**

- 1) **MQTT:** -lightweight messaging protocol for small sensors and mobile devices optimized for high-latency or unreliable networks
- 2) **Publisher:** Sends messages to a topic
- 3) **Subscriber:** receives messages from a topic it is subscribed
- 4) **Broker:** A server that receives messages from publishers and routes them to subscribers.
- 5) **Message:** The data transmitted by publishers to topics, which subscribers receive

➤ **Quality of Service (QoS):** Defines the guarantee of message delivery:

- 1) **QoS 0:** At most once (fire and forget).
- 2) **QoS 1:** At least once (acknowledged delivery).
- 3) **QoS 2:** Exactly once (assured delivery)

➤ **Steps To set Broker:** - (There are lot of steps for Broker setup, the concept setups are same but little bit vary based on OS of installation, server and for testing we can use pre-configured mobile app or browser extensions.)

- 1) **Go to** - <https://cedalo.com/blog/how-to-install-mosquitto-mqtt-broker-on-windows/>
- 2) Follow each instruction for installing and configuring – “mosquitto-mqtt-broker”
- 3) After successful installation, using CMD check for “mosquitto-mqtt-broker service is running”
- 4) Screenshot: - (For Windows.)

```
C:\Windows\system32>sc start mosquitto ←
SERVICE_NAME: mosquitto
    TYPE               : 10  WIN32_OWN_PROCESS
    STATE              : 2   START_PENDING
                           (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
    WIN32_EXIT_CODE    : 0   (0x0)
    SERVICE_EXIT_CODE : 0   (0x0)
    CHECKPOINT        : 0x0
    WAIT_HINT         : 0x7d0
    PID                : 3880
    FLAGS              :

C:\Windows\system32>sc query mosquitto ←
SERVICE_NAME: mosquitto
    TYPE               : 10  WIN32_OWN_PROCESS
    STATE              : 4   RUNNING
                           (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
    WIN32_EXIT_CODE    : 0   (0x0)
    SERVICE_EXIT_CODE : 0   (0x0)
    CHECKPOINT        : 0x0
    WAIT_HINT         : 0x0

C:\Windows\system32>netstat -an | findstr 1883 ←
TCP    0.0.0.0:1883          0.0.0.0:0              LISTENING
```

- “Mosquitto-mqtt” Subscriber Command: - (**1<sup>st</sup> for remote Sever and 2<sup>nd</sup> for Local setup**)

- 1) >mosquitto\_sub -h your\_broker\_ip\_address -i mosq\_sub1 -t "home/DHT" -d
- 2) > mosquitto\_sub -h localhost -i mosq\_sub1 -t "home/DHT" -d

```
c:\Windows\system32>mosquitto_sub -h your_broker_ip_address -i mosq_sub1 -t "home/DHT" -d
```

- Upload code on ESP32: - (we already have preinstalled module named as umqtt. Also Make sure to **change Network and MQTT credentials**)

```
import time
import dht
from machine import Pin, SoftI2C
from umqtt.simple import MQTTClient
import sh1106
import network

# WiFi Configuration
ssid = "mohan Samsung" #Your WIFI username
password = "mohanshikare12345" # Your WIFI pass

def connect_wifi(): # Connect to WiFi
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(ssid, password)

    while not wlan.isconnected():
        print("Connecting to WiFi...")
        time.sleep(1)

    print("Connected to WiFi")
    print("Network config:", wlan.ifconfig())

# Configuration for DHT sensor
dht_pin = Pin(4, Pin.IN)
sensor = dht.DHT11(dht_pin)

# Configuration for OLED display
i2c = SoftI2C(scl=Pin(22), sda=Pin(21))
oled = sh1106.SH1106_I2C(128, 64, i2c)

# Configuration for MQTT
mqtt_server = "your_broker_ip_address"
client_id = "ESP32_DHT11"
topic_pub = "home/DHT"

# Initialize MQTT client
client = MQTTClient(client_id, mqtt_server)

# Function to connect to MQTT broker
def connect_mqtt():
    client.connect()
    print("Connected to MQTT broker at", mqtt_server)
```

```

# Function to read DHT sensor data
def read_dht_sensor():
    try:
        sensor.measure()
        temp = sensor.temperature()
        hum = sensor.humidity()
        return temp, hum
    except OSError as e:
        print("Failed to read from DHT sensor:", e)
        return None, None

connect_wifi() # Connect to WiFi
connect_mqtt() # Connect to MQTT broker

while True: # Main loop
    temp, hum = read_dht_sensor()
    if temp is not None and hum is not None:
        msg = "Temperature: {:.1f} C, Humidity: {:.1f}%".format(temp, hum)
        client.publish(topic_pub, msg)
        print("Published:", msg)

        # Display values on OLED
        oled.fill(0)
        oled.text('Temp: {:.1f} C'.format(temp), 0, 0)
        oled.text('Humidity: {:.1f}%'.format(hum), 0, 10)
        oled.show()

    time.sleep(2) # Wait for 2 seconds before the next reading

```

➤ **Output Screenshots: - (Receiver/Subscriber Side -Read logs)**

```

C:\Windows\system32>mosquitto_sub -h localhost -i mosq_sub1 -t "home/DHT" -d
Client mosq_sub1 sending CONNECT
Client mosq_sub1 received CONNACK (0)
Client mosq_sub1 sending SUBSCRIBE (Mid: 1, Topic: home/DHT, QoS: 0, Options: 0x00)
Client mosq_sub1 received SUBACK
Subscribed (mid: 1): 0
Client mosq_sub1 sending PINGREQ
Client mosq_sub1 received PINGRESP
Client mosq_sub1 sending PINGREQ
Client mosq_sub1 received PINGRESP
Client mosq_sub1 received PUBLISH (d0, q0, r0, m0, 'home/DHT', ... (37 bytes))
Temperature: 31.7 C, Humidity: 71.1 % ←
Client mosq_sub1 received PUBLISH (d0, q0, r0, m0, 'home/DHT', ... (37 bytes))
Temperature: 31.6 C, Humidity: 69.2 %
Client mosq_sub1 received PUBLISH (d0, q0, r0, m0, 'home/DHT', ... (37 bytes))
Temperature: 32.3 C, Humidity: 71.0 %
Client mosq_sub1 received PUBLISH (d0, q0, r0, m0, 'home/DHT', ... (37 bytes))
Temperature: 31.9 C, Humidity: 69.8 %
Client mosq_sub1 received PUBLISH (d0, q0, r0, m0, 'home/DHT', ... (37 bytes))
Temperature: 31.9 C, Humidity: 68.4 %
Client mosq_sub1 sending PINGREQ

```

➤ **ESP32 Logs on Thonney IDE Micropython Console: -**

```

0
Connected to MQTT broker at 192.168.43.253
Published: Temperature: 31.7 C, Humidity: 71.1 %
Published: Temperature: 31.6 C, Humidity: 69.2 %
Published: Temperature: 32.3 C, Humidity: 71.0 %
Published: Temperature: 31.9 C, Humidity: 69.8 %
Published: Temperature: 31.9 C, Humidity: 68.4 %

```

## 36) Esp32 / ESP8266 interfacing OLED display and Pot

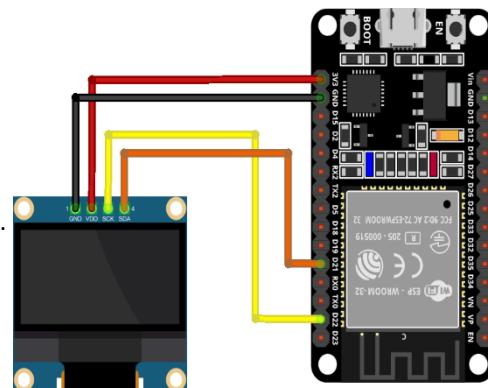
(Note: - Refer / Study **I<sup>2</sup>C protocol from chapter 13** – Communications Protocols)

### ➤ Components Needed: -

- 1) ESP32 or ESP8266
- 2) I<sup>2</sup>C OLED display
- 3) Jumper wires, 10 K pot.

### ➤ Connection: -

- 1) VCC: Connect to the 3.3V pin on the ESP32/ESP8266.
- 2) GND: GND pin
- 3) SCL (Clock): GPIO 22 (ESP32) / GPIO 5 (ESP8266)
- 4) SDA (Data): GPIO 21 (ESP32) / GPIO 4 (ESP8266)



### ➤ I<sup>2</sup>C Address Scanner Code: -

Code: - Scans and Detect Connected I<sup>2</sup>C Component

```
from machine import Pin, SoftI2C

# Initialize SoftI2C interface
i2c = SoftI2C(scl=Pin(22), sda=Pin(21)) # For ESP32
# i2c = SoftI2C(scl=Pin(5), sda=Pin(4)) # For ESP8266

# Scan for devices
devices = i2c.scan()

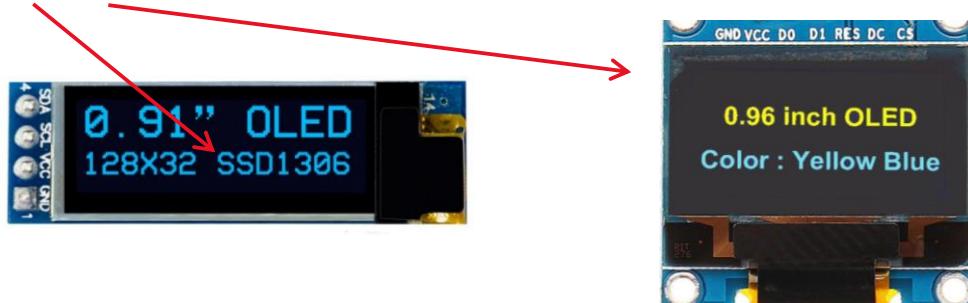
if devices:
    for device in devices:
        print("I2C device found at address:", hex(device))
else:
    print("No I2C devices found")
```

### ➤ Common Functions: - Before Proceeding try to remember common function for OLED.

- 1) **oled.fill(0)** : - Clears the display by filling it with zeros (off).
- 2) **oled.text('Hello', x, y)** : - Displays text at x and y coordinates
- 3) **oled.show()** : - Updates the physical display with the current buffer contents.
- 4) **oled.pixel(x, y, color)**: - Sets a pixel at the specified x and y coordinates of colour
- 5) **oled.rect(x, y, width, height, color)** and **oled.fill\_rect(x, y, width, height, color)**
- 6) **oled.line(x1, y1, x2, y2, color)** and **oled.scroll(dx, dy)**
- 7) **framebuf.FrameBuffer(buf, 128, 64, framebuf.MONO\_HLSB)** : - Display Images

Note: - This function is **Same For SSD1306 and SH110X Driver Display**

### ➤ **SSD1306 Driver** OLED Display: - “ssd1306” module is preinstalled on Micropython Firmware.



- **Code:** - Don't Worry, in some firmware if you don't find it preinstalled, then just install with "Micropython Package Installer" using "mip". Let's see How.

1) Connect Board with Internet: -



```
import network
import time

# Wi-Fi credentials
ssid = 'Mohan samsung' #your_SSID
password = 'mohanshikare12345'#your_PASSWORD

def connect_wifi():
    wlan = network.WLAN(network.STA_IF)      # Set up the station (client) mode
    wlan.active(True)
    wlan.connect(ssid, password) # Connect to the Wi-Fi network

    while not wlan.isconnected(): # Wait for connection
        print('Connecting to network...')
        time.sleep(1)

    print('Network connected!')
    print('IP address:', wlan.ifconfig()[0])

connect_wifi()
```

2) Import mip and Download "ssd1306": -

(It is Equivalent of saying "**pip install XYZ**" here instead of "pip" we have "mip").

```
>>> import os
>>> os.listdir("/")
['boot.py', 'lib']
>>> help("modules")
In this I don't have "ssd1306"
preinstalled
```

__main__	bluetooth		heapq	select
_asyncio	btree		inisetup	socket
_boot	builtins		io	ssl
_espnow	cmath		json	struct
_onewire	collections		machine	sys
_thread	cryptolib		math	time
_webrepl	deflate		micropython	tls
aioespnow	dht		mip/_init_	uasyncio
apal06	ds18x20		neopixel	uctypes
array	errno		network	umqtt/robust
asyncio/_init_	esp		ntptime	umqtt/simple
asyncio/core	esp32		onewire	upsh
asyncio/event	espnow		os	urequests
asyncio/funcs	flashbdev		platform	vfs
asyncio/lock	framebuf		random	webrepl
asyncio/stream	gc		re	webrepl_setup
binascii	hashlib		requests/_init_	websocket

Plus any modules on the filesystem

Install using mip and verify by importing

```
>>> import mip
>>> mip.install("ssd1306") ←
Installing ssd1306 (latest) from https://micropython.org/pi/v2 to /lib
Copying: /lib/ssd1306.mpy ←
Done
```

It installed **precompiled** byte code module file, its **binary file** having **".mpy"** extension (fast , less power as it is machine byte code)

If precompiled byte code module not available then it installs **plain ".py" Format**. Letter it interpreted by MCU (slow, more power req.)

```
>>> os.listdir("/lib")
['sh1106.py', 'ssd1306.mpy']
>>> import ssd1306
>>> |
```

```

from machine import Pin, SoftI2C
import ssd1306

# Initialize I2C interface
i2c = SoftI2C(scl=Pin(22), sda=Pin(21)) # For ESP32
# i2c = SoftI2C(scl=Pin(5), sda=Pin(4)) # For ESP8266

# Initialize the OLED display
oled_width = 128
oled_height = 64
oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c, addr=0x3c)

oled.fill(0) # Clear the display

oled.text('Hello, World!', 0, 0) # Display text
oled.text('MicroPython', 0, 10)
oled.text('OLED Display', 0, 20)

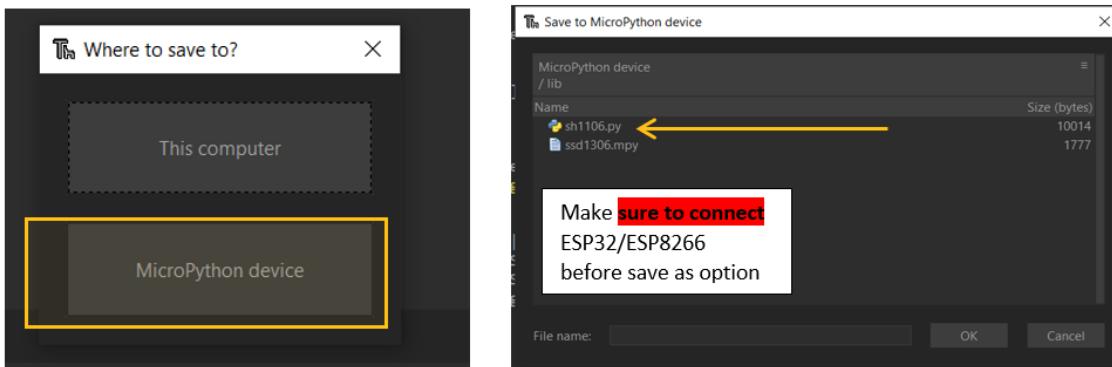
oled.show() # Show the display

```

➤ **Code for SH110X Driver Display:** - Procedure is almost same, but we have to install "sh1106" module from 3<sup>rd</sup> party GitHub repository (Download and extract repo. Content from link)

1) **Repo link:** - <https://github.com/robert-hh/SH1106>

2) **Open "sh1106" file in Thonny-IDE:** - Connect **ESP** then click **"save as"** > **"Micropython device"**



```

MPY: soft reboot
MicroPython v1.23.0 on 2024-06-02; Generic ESP32 module with ESP32
Type "help()" for more information.
>>> import os
>>> os.listdir("/lib")
['sh1106.py', 'ssd1306.mpy']
>>> import sh1106
>>>

```

Verify Installation.  
 By **importing** Module

3) **Code for 1.3-inch SH110X Driver OLED:** -

```

from machine import Pin, SoftI2C
import sh1106

i2c = SoftI2C(scl=Pin(22), sda=Pin(21)) # For ESP32
# i2c = SoftI2C(scl=Pin(5), sda=Pin(4)) # For ESP8266

# Initialize the OLED display
oled_width = 128
oled_height = 64
reset_pin = Pin(16) # If reset pin is used, otherwise set to None
oled = sh1106.SH1106_I2C(oled_width, oled_height, i2c, reset_pin, addr=0x3c)

oled.fill(0) # Clear the display
oled.rect(1, 1, 127, 63, 1) # x, y, width, height, color (1 for white)
oled.fill_rect(70, 10, 40, 20, 1) # x, y, width, height, color (1 for white)
oled.line(0, 0, 123, 61, 1) # x1, y1, x2, y2, color (1 for white)
oled.vline(60, 0, 64, 1) # x, y, height, color - vertical line
#oled.hline(0, 30, 128, 1) # x, y, width, color -horizontal line

# Draw a circle function
def draw_circle(oled, x0, y0, radius, color):
    f = 1 - radius
    ddF_x = 1
    ddF_y = -2 * radius
    x = 0
    y = radius

    oled.pixel(x0, y0 + radius, color)
    oled.pixel(x0, y0 - radius, color)
    oled.pixel(x0 + radius, y0, color)
    oled.pixel(x0 - radius, y0, color)

    while x < y:
        if f >= 0:
            y -= 1
            ddF_y += 2
            f += ddF_y
        x += 1
        ddF_x += 2
        f += ddF_x

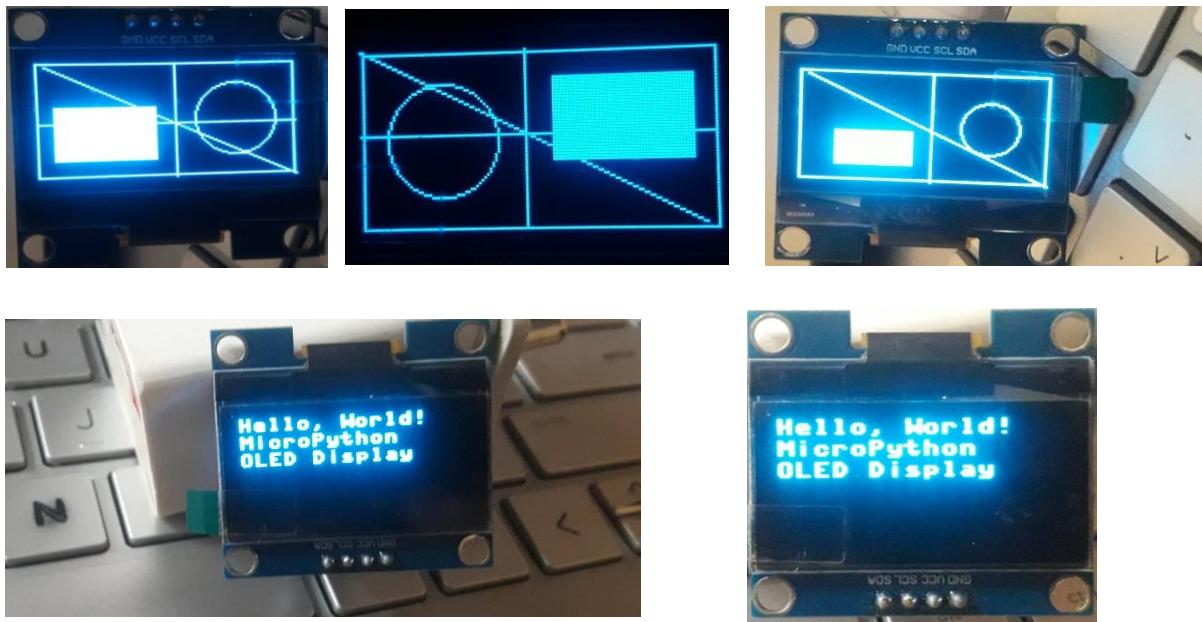
        oled.pixel(x0 + x, y0 + y, color)
        oled.pixel(x0 - x, y0 + y, color)
        oled.pixel(x0 + x, y0 - y, color)
        oled.pixel(x0 - x, y0 - y, color)
        oled.pixel(x0 + y, y0 + x, color)
        oled.pixel(x0 - y, y0 + x, color)
        oled.pixel(x0 + y, y0 - x, color)
        oled.pixel(x0 - y, y0 - x, color)

    draw_circle(oled, 30, 32, 15, 1) # x, y, radius, color (1 for white)

    oled.show() # Show the display

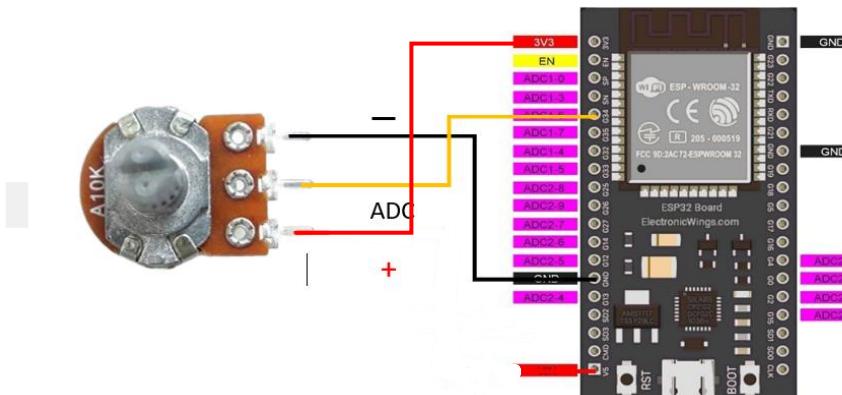
```

#### 4) Output Photos: -



### 36.1) Esp32 / ESP8266 interfacing OLED display and Pot

- Connect additional **10 k Pot** to above circuit: - (Connect ADC pin to GPIO -34, VCC – 3.3V)



- **Don't Remove** previous Circuit OLED, in this we have added **10k Pot.**, Use breadboard for **3.3v**
- **Code:** -

```
from machine import Pin, ADC, SoftI2C
import sh1106
import time

# Initialize SoftI2C interface
i2c = SoftI2C(scl=Pin(22), sda=Pin(21)) # For ESP32
# i2c = SoftI2C(scl=Pin(5), sda=Pin(4)) # For ESP8266

# Initialize the OLED display
oled_width = 128
oled_height = 64
reset_pin = Pin(16) # If reset pin is used, otherwise set to None
oled = sh1106.SH1106_I2C(oled_width, oled_height, i2c, reset_pin, addr=0x3c)
```

```

# Initialize ADC for potentiometer
pot = ADC(Pin(34)) # ADC pin connected to potentiometer (ESP32)
# pot = ADC(Pin(0)) # For ESP8266, use appropriate ADC pin

# Function to map potentiometer value to 0-100 range
def map_value(value, in_min, in_max, out_min, out_max):
    return (value - in_min) * (out_max - out_min) // (in_max - in_min) + out_min

while True:
    pot_value = pot.read() # Read potentiometer value
    # Map potentiometer value to 0-100 range
    scaled_value = map_value(pot_value, 0, 4095, 0, 100)
    # Calculate width for bar graph
    bar_width = scaled_value * oled_width // 100

    oled.fill(0) # Clear the display
    oled.text('Pot Value: {}'.format(pot_value), 0, 0)
    oled.fill_rect(0, 20, bar_width, 10, 1) # Draw horizontal bar graph

    oled.show() # Show the display

    time.sleep(1) # Wait for 1 second

```

➤ **Output Photos: -**



## 37) ESP32 / ESP8266 interfacing GSM Module (UART)

(Note: - Refer / Study **UART protocol from chapter 13** – Communications Protocols)

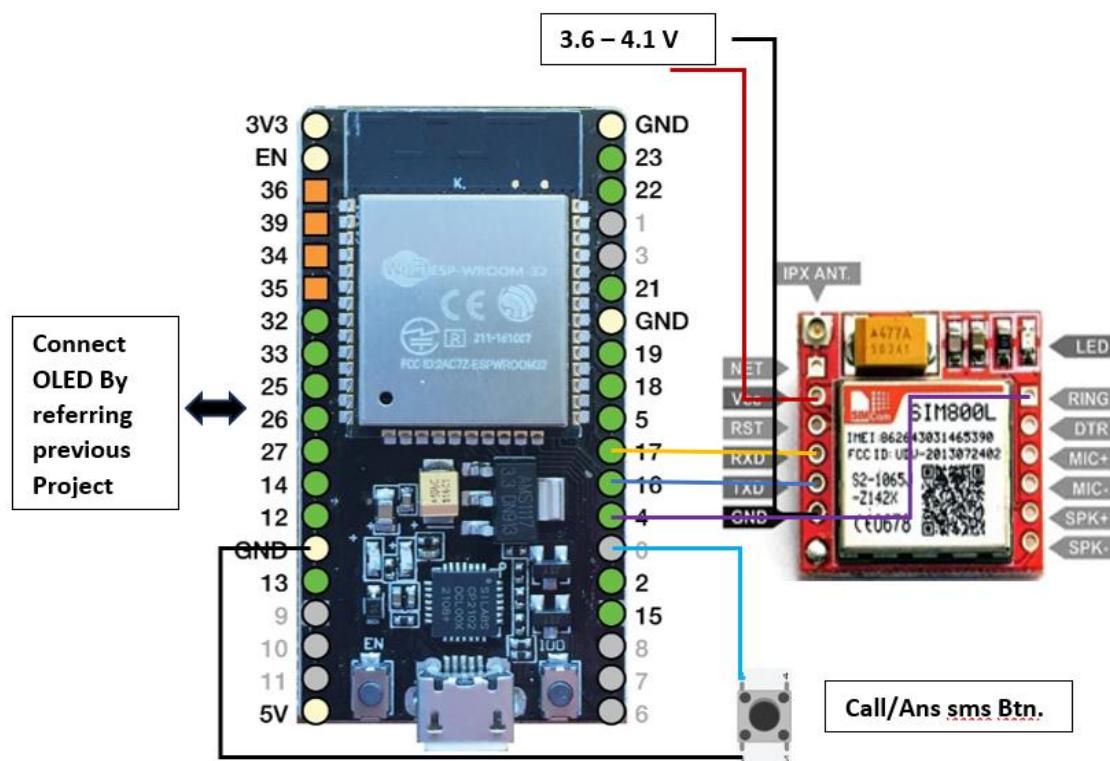
➤ **Components Needed:** -

- 1) ESP32 or ESP8266
- 2) GSM module & OLED display
- 3) Jumper wires, push Button

➤ **Connection:** -

- 1) Connect OLED.
- 2) GSM - RX - 17
- 3) GSM - TX - 16
- 4) GSM RING - 4
- 5) Button – 0
- 6) Power for GSM – 3.6 to 4.1 v.

**(Not possible to power by ESP as the voltage is 3.3 vs or 5v)**



➤ **Some gsm AT Commands:** -

- 1) **Check Module:** AT - Check if the module is responsive. Example: AT
- 2) **Signal Strength:** AT+CSQ - Check signal quality. Example: AT+CSQ
- 3) **Network Registration:** AT+CREG? - Check network registration status. Example: AT+CREG?
- 4) **SIM Card Info:** AT+CIMI - Get SIM card IMSI. Example: AT+CIMI
- 5) **SIM Provider Info:** AT+CSPN? - Get SIM provider name. Example: AT+CSPN?
- 6) **Send SMS:** AT+CMGS - Send an SMS. Example: AT+CMGS="+1234567890"
- 7) **Make Call:** ATD - Dial a phone number. Example: ATD+1234567890;
- 8) **Hang Up Call:** ATH - Hang up the call. Example: ATH
- 9) **Check Operator:** AT+COPS? - Check current operator. Example: AT+COPS?

➤ **Code:** -

```

import time , sh1106
from machine import Pin, SoftI2C, UART

# OLED Configuration
i2c = SoftI2C(scl=Pin(22), sda=Pin(21)) # Adjust pins for ESP32/ESP8266
oled = sh1106.SH1106_I2C(128, 64, i2c)
# oled.rotate(True) # Rotate display

# Button Configuration and Ring pin
button = Pin(0, Pin.IN, Pin.PULL_UP) # Adjust pin as per your connection
ring_pin = Pin(2, Pin.IN) # RING pin of SIM800L

# GSM Module Configuration
uart = UART(1, baudrate=9600, tx=Pin(17), rx=Pin(16)) # pins for TX/RX

# Phone number and message
phone_number = "1234567890"
message = """Hello from ESP32! And
Nikhil Mohan Shikare.."""

# Function to send AT command
def send_at(command, timeout=1000):
    uart.write(command + '\r\n')
    time.sleep_ms(timeout)
    response = uart.read()
    return response.decode('utf-8') if response else ''

def init_gsm(): # Initialize GSM
    print(send_at("AT")) # Check if GSM is ready
    print(send_at("AT+CMGF=1")) # Set SMS text mode
    print(send_at("AT+CNMI=1,2,0,0,0")) # Set SMS to show immediately

def send_sms(number, text):
    send_at("AT+CMGF=1")
    send_at(f'AT+CMGS="{number}"')
    uart.write(text + '\r\n')
    uart.write(b'\x1A') # ASCII code for CTRL+Z
    print("Message sent")

def make_call(number): # Make a call
    send_at(f"ATD{number};")
    print("Call initiated")

def answer_call(): # Answer a call
    send_at("ATA")
    print("Call answered")

# Display message on OLED
def display_message(line1, line2, line3):
    oled.fill(0)
    oled.text(line1, 0, 0)
    oled.text(line2, 0, 10)
    oled.text(line3, 0, 20)

```

```

# Display call icon or any other graphical elements if needed
oled.show()

init_gsm() # Initialize GSM

while True: # Main loop
    if ring_pin.value() == 0: # Incoming call detected (RING pin is low)
        display_message("Incoming Call", "Press button", "to pick up")
        while button.value() == 1: # Wait for button press
            pass
        answer_call()
        display_message("Call Answered", "", "")
        time.sleep(5) # Keep display for a while
    else:
        display_message("Press button", "to call/send SMS", "")
        if button.value() == 0: # Button pressed
            send_sms(phone_number, message)
            make_call(phone_number)
            display_message("Message sent", "and Call initiated", "")
            time.sleep(5) # Keep display for a while

    time.sleep(1) # Check every second

```

➤ **Output Photo:** -



➤ **Challenge/Assignment Task:**

- 1) Extend this project by **adding keypad** to type phone number, and feed different SMS template in it.
- 2) Additionally add GPS module (refer next **chapter 37**) to send locations in SMS on pressing button
- 3) Use Multithreading, Asyncio, take advantage of **ESP32 powerful Dual core processor** and onboard modules like, **hashlib, threading, os, esp for speed/power freqnky.** Options.

<code>array</code>	<code>array</code>	<code>array</code>
<code>asyncio/_init_</code>	<code>esp</code>	<code>network</code>
<code>asyncio/core</code>	<code>esp32</code>	<code>ntptime</code>
<code>asyncio/event</code>	<code>espnw</code>	<code>onewire</code>
<code>asyncio/funcs</code>	<code>flashbdev</code>	<code>os</code>
<code>asyncio/lock</code>	<code>framebuf</code>	<code>platform</code>
<code>asyncio/stream</code>	<code>gc</code>	<code>random</code>
<code>binascii</code>	<code>hashlib</code>	<code>re</code>
		<code>requests/_</code>

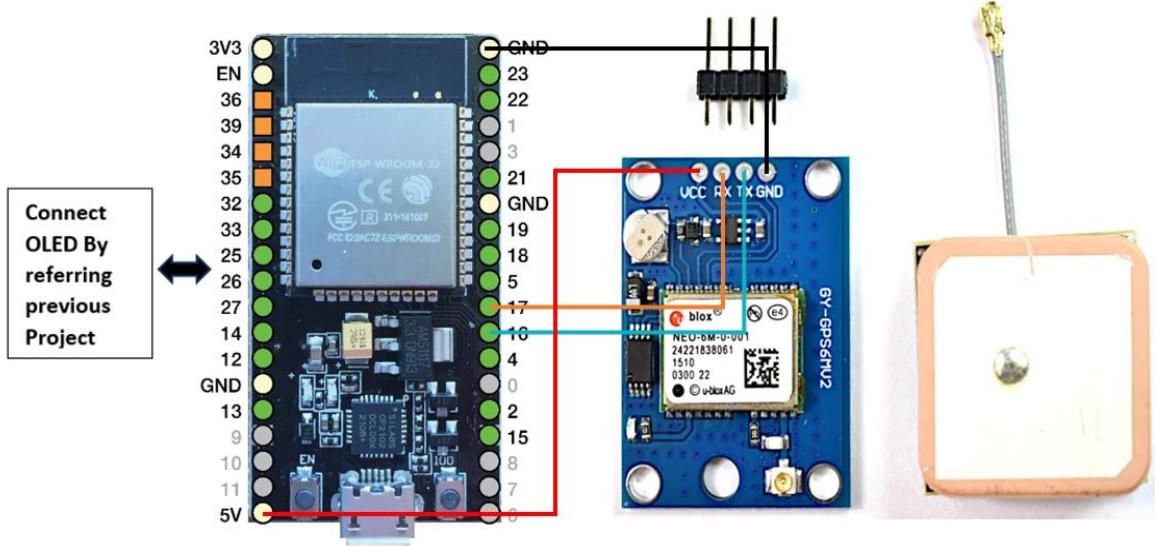
## 38) ESP32 / ESP8266 GPS Weather Station (API & HTTP)

➤ **Components Needed:** -

- 1) ESP32 or ESP8266
- 2) GPS neo 6M/7M/8M module & OLED display
- 3) Jumper wires.

➤ **Connection:** -

- 1) Connect OLED.
- 2) GPS - RX - 17
- 3) GPS - TX - 16
- 4) Power for GPS – 2.7V~5.0V – power with ESP's – **5V**.
- 5) GPS – GND.



➤ **Some GPS AT Commands:** -

- 1) **Check GPS:** AT - Check if the module is responsive. Example: AT
- 2) **Get GPS Position:** AT+CGPSINFO: -  
**Output:**  
+CGPSINFO: <lat>, <N/S>, <long>, <E/W>, <date>, <time>, <alt>, <speed>
- 3) **Turn on GPS:** AT+CGPSPWR=1 - **Parameter:** 1 (to turn on), 0 (to turn off)  
**Output:** - AT+CGPSPWR=1 : OK
- 4) **Get GPS Status:** AT+CGPSSTATUS? - **Parameter:** None  
**Output:** +CGPSSTATUS: <status>
- 5) **Get Latitude and Longitude:** AT+CGPSINF=32 - **Parameter:** 32  
**Output:** +CGPSINF: <mode>, <lat>, <long>, <date>, <time>

- Key GPS Data Parameters: -
  - 1) **Latitude and Longitude:** - Example: 19.123456, N, 72.987654, E
  - 2) **Altitude:** - Example: 0.0 (in meters)
  - 3) **Speed:** - Example: 0.0 (in km/h or knots)
  - 4) **Date and Time:** - Example: 230621, 130202.0 (DDMMYY, HHMMSS.S)
  - 5) **Direction/Heading:** - Example: 0.0 (in degrees from North)
  - 6) **Fix Status:** Example: Location 3D Fix (no fix, 2D fix, 3D fix).
  - 7) **Satellites Used:** - Example: 8 (number of satellites)
  
- some common **NMEA sentence** formats: - (For Format Examples, Refer Screenshot in **Output**)
  - 1) **GPGLL** - Geographic Position, Latitude/Longitude: -
 

**Format:** \$GPGLL,latitude,latitude direction,longitude,longitude direction,UTC time,Status,Mode\*Checksum
  - 2) **GPRMC** - Recommended Minimum Specific GPS/Transit Data: -
 

**Format:** \$GPRMC,UTC time,Status,latitude,latitude direction,longitude,longitude direction,Speed over ground,Course over ground,Date,Magnetic variation,Magnetic variation direction,Mode\*Checksum
  - 3) **GPVTG** - Course Over Ground and Ground Speed: -
 

**Format:** \$GPVTG,True track made good,True,T,Magnetic track made good,Magnetic,N,Speed over ground in knots,N,Speed over ground in kilometers per hour,K,Mode\*Checksum
  - 4) **GPGGA** - Global Positioning System Fix Data: -
 

**Format:** \$GPGGA,UTC time,latitude,latitude direction,longitude,longitude direction,Fix quality,Number of satellites,Horizontal dilution of position,Altitude,Altitude units,Height of geoid above WGS84 ellipsoid,Height units,Time since last DGPS update,DGPS station ID\*Checksum
  - 5) **GPGSA** - GPS DOP and Active Satellites: -
 

**Format:** \$GPGSA,Mode,M,Satellite used(1-12),PDOP,HDOP,VDOP\*Checksum
  - 6) **GPZDA** - Date and Time: -
 

**Format:** \$GPZDA,UTC time,Day,Month,Year,Local time zone hours,Local time zone minutes\*Checksum
  - 7) **GPGSV** - GPS Satellites in View: -
 

**Format:** \$GPGSV,Number of sentences,Sentence number,Number of satellites in view,Satellite PRN number,Elevation,Azimuth,SNR (C/N0) (1-4 of these fields per sentence)\*Checksum

➤ Typical Use of GPS: -

- 1) **Navigational Applications:** Latitude, longitude, altitude, speed, and direction are critical for navigation.
- 2) **Tracking Systems:** Use GPS data to track the movement and speed of vehicles or assets.
- 3) **Geofencing:** Use location data to define virtual boundaries for triggering events when a device enters or leaves a specific area.
- 4) **Weather Stations:** Combine GPS data with weather APIs to provide localized weather information based on the current GPS coordinates.

➤ Code: - (Normal code to display **text data** on **OLED Display**): -

```
import machine
import sh1106
import framebuffer
import network
import time
import urequests
from machine import UART, Pin

# Network and API
SSID = 'Mohan Samsung'
PASS = 'mohanShikare123459'
# Replace with your OpenweathermapAPIKEY
API_KEY = '74427e9646edc6980be'

# I2C Configuration using SoftI2C
i2c = machine.SoftI2C(scl=machine.Pin(22), sda=machine.Pin(21))
oled = sh1106.SH1106_I2C(128, 64, i2c)
oled.rotate(True)

# Configure UART for GPS
uart = UART(1, baudrate=9600, tx=Pin(17), rx=Pin(16))

# Dictionary to store GPS data
gps_data = {
    'latitude': None,
    'longitude': None,
    'speed': None,
    'time_utc': None,
    'date': None
}
weather_data = {
    'temp': None,
    'pressure': None,
    'humidity': None,
    'wind_speed': None
}

class Network_API:
    wlan = None # Store the WLAN object

    @staticmethod
```

```

def connect_to_wifi(ssid, password):
    Network_API.wlan = network.WLAN(network.STA_IF)
    Network_API.wlan.active(True)
    Network_API.wlan.connect(ssid, password)

    # Define the WIFI_BITMAP
    WIFI_BITMAP = bytearray([
        0x01, 0xfe, 0x00, 0x0e, 0x01, 0xc0, 0x10, 0x00, 0x20, 0x60, 0x00,
        0x18, 0x80, 0x00, 0x04, 0x01,
        0xce, 0x00, 0x06, 0x01, 0x80, 0x08, 0x00, 0x40, 0x00, 0x00, 0x00,
        0x00, 0x78, 0x00, 0x01, 0x86,
        0x00, 0x00, 0x00, 0x00, 0x30, 0x00, 0x00, 0x48, 0x00, 0x00,
        0x48, 0x00, 0x00, 0x30, 0x00
    ])

    # Create a framebuffer for the image
    fb = framebuf.FrameBuffer(WIFI_BITMAP, 22, 16, framebuf.MONO_HLSB)

    # Wait for the connection to establish
    oled.fill(0)
    oled.text("Connecting WI-FI..", 0, 0)
    oled.blit(fb, 100, 25)
    oled.show()

    while not Network_API.wlan.isconnected():
        print("Connecting..")
        time.sleep(1)

        oled.text("Connected..", 0, 20)
        oled.text(f"IP -> {Network_API.wlan.ifconfig()[0]}", 0, 50)
        print(Network_API.wlan.ifconfig())
        oled.show()
        time.sleep(2)

@staticmethod
def get_API_Data(api_key , latitude , longitude):
    print("In API CALL FUNCTION..")
    if not Network_API.wlan.isconnected():
        return
    global weather_data
    try:
        url =
f'https://api.openweathermap.org/data/2.5/weather?lat={latitude}&lon={longitude}&appid={api_key}'
        response = urequests.get(url)
        data = response.json()

        weather_data['temp'] = (data['main']['temp'])-273.15
        weather_data['pressure'] = data['main']['pressure']
        weather_data['humidity'] = data['main']['humidity']
        weather_data['wind_speed'] = data['wind']['speed']

        print(weather_data) # Print the weather data to see updates

```

```

        except Exception as e:
            print(f"Error fetching weather data: {e}")

class GPS_Data:
    @staticmethod
    def read_gps():
        global gps_data
        try:
            line = uart.readline()
            if line is None:
                return

            line = line.decode('utf-8').strip()
            if line.startswith('$GPGLL'):
                GPS_Data.parse_gpgll(line)
            elif line.startswith('$GPRMC'):
                GPS_Data.parse_gprmc(line)
            elif line.startswith('$GPVTG'):
                GPS_Data.parse_gpvtg(line)

            print(gps_data) # Print the dictionary to see updates
        except Exception as e:
            print(f"Error reading GPS: {e}")

    @staticmethod
    def parse_gpgll(line):
        global gps_data
        parts = line.split(',')
        if len(parts) < 7:
            print("Incomplete GPGLL data")
            return

        latitude = GPS_Data.convert_to_degrees(parts[1], parts[2])
        longitude = GPS_Data.convert_to_degrees(parts[3], parts[4])
        time_utc = parts[5]

        if latitude is not None and longitude is not None:
            gps_data['latitude'] = latitude
            gps_data['longitude'] = longitude
        if time_utc:
            gps_data['time_utc'] = time_utc

    @staticmethod
    def parse_gprmc(line):
        global gps_data
        parts = line.split(',')
        if len(parts) < 12:
            print("Incomplete GPRMC data")
            return

        time_utc = parts[1]
        latitude = GPS_Data.convert_to_degrees(parts[3], parts[4])
        longitude = GPS_Data.convert_to_degrees(parts[5], parts[6])

```

```

speed = parts[7]
date = parts[9]

if time_utc:
    gps_data['time_utc'] = time_utc
if latitude is not None:
    gps_data['latitude'] = latitude
if longitude is not None:
    gps_data['longitude'] = longitude
if speed:
    gps_data['speed'] = speed
if date:
    gps_data['date'] = date

@staticmethod
def parse_gpvtg(line):
    global gps_data
    parts = line.split(',')
    if len(parts) < 9:
        print("Incomplete GPVTG data")
        return

    speed = parts[7] # Speed in knots

    if speed:
        gps_data['speed'] = speed

@staticmethod
def convert_to_degrees(value, direction):
    try:
        if not value or not direction:
            return None

        # Determine if it's latitude or longitude based on the direction
        if direction in ['N', 'S']:
            degrees = float(value[:2])
            minutes = float(value[2:])
        elif direction in ['E', 'W']:
            degrees = float(value[:3])
            minutes = float(value[3:])

        result = degrees + (minutes / 60)

        if direction in ['S', 'W']:
            result = -result

        return round(result, 6)
    except ValueError as e:
        print(f"Error converting to degrees: {e}")
        return None

def format_time(time_utc):
    if time_utc and len(time_utc) >= 6:

```

```

        hours = time_utc[:2]
        minutes = time_utc[2:4]
        seconds = time_utc[4:6]
        return f"{hours}:{minutes}:{seconds} UTC"
    return "N/A"

def format_date(date):
    if date and len(date) == 6:
        day = date[:2]
        month = date[2:4]
        year = date[4:6]
        return f"{day}-{month}-{year}"
    return "N/A"

def show_data(screen_counter):
    oled.fill(0)
    # Persist one screen for 5 seconds
    if screen_counter < 5: # for GPS data
        oled.text(f"Lat: {gps_data['latitude']}", 0, 15)
        oled.text(f"Long: {gps_data['longitude']}", 0, 25)
        oled.text(f"Speed: {gps_data['speed']} KN", 0, 35)
        if gps_data['date']:
            oled.text(f"Date: {format_date(gps_data['date'])}", 0, 45)
        else:
            oled.text("Date: N/A", 0, 45)
        if gps_data['time_utc']:
            oled.text(f"Time: {format_time(gps_data['time_utc'])}", 0, 55)
        else:
            oled.text("Time: N/A", 0, 55)
        oled.show()
    elif screen_counter >= 5 and screen_counter <= 11: # for weather data
        oled.text(f"Temp: {weather_data['temp']}C", 0, 20)
        oled.text(f"pres: {weather_data['pressure']} hPa", 0, 30)
        oled.text(f"humid: {weather_data['humidity']} %", 0, 40)
        oled.text(f"wind: {weather_data['wind_speed']} m/s", 0, 50)
        oled.show()
    else:
        screen_counter = 0
    return screen_counter

# Connect to WiFi
Network_API.connect_to_wifi(SSID, PASS)

# Main loop
screen_counter = 0
counter=0
while True:
    GPS_Data.read_gps()
    if counter > 10:
        if gps_data['latitude'] is not None and gps_data['longitude'] is not
None:
            Network_API.get_API_Data(API_KEY, gps_data['latitude'],
gps_data['longitude'])

```

```

        counter = 0
screen_counter = show_data(screen_counter)
screen_counter += 1
counter+=1
time.sleep(2.5)

```

➤ **Output Photo:** -

- 1) Common **NMEA sentence formats** and their Values: -

Common **NMEA sentence formats** and their Values:

```

>>> import time
>>> import urequests
>>> from machine import UART, Pin
>>> uart = UART(1, baudrate=9600, tx=Pin(17), rx=Pin(16))
>>> while(True):
    line = uart.readline()
    print(line)
    time.sleep(2)

b'$GPRMC,050547.00,V,,,,,,240624,,,N*78\r\n'
b'$GPVTG,,,,,,N*30\r\n'
b'$GPGGA,050547.00,,,,0.00,99.99,,,,,*65\r\n'
b'$GPGSA,A,1,,,,,,99.99,99.99,99.99*30\r\n'
b'$GPGSV,1,1,02,05,,,39,12,,,41*72\r\n'
b'$GPGLL,,,,050547.00,V,N*49\r\n'
b'$GPRMC,050548.00,V,,,,,,240624,,,N*77\r\n'
b'$GPVTG11.27864,N,07303.02406,E,050740.00,A,A*6F\r\n'
b'$GPRMC,050741.00,A,1911.27853,N,07303.02389,E,0.184,,240624,,,A*78\r\n'
b'$GPVTG,,T,,M,0.184,N,0.340,K,A*29\r\n'
b'$GPGGA,050741.00,1911.27853,N,07303.02389,E,1,04,4.61,19.6,M,-65.5,*4D\r\n'
b'$GPGSA,A,3,05,18,25,12,,,,6.56,4.61,4.66*08\r\n'
b'$GPGSV,2,1,06,05,81,085,40,12,07303.02290,E,050748.00,A,A*6F\r\n'
b'$GPRMC,050749.00,A,1911.27854,N,07303.02284,E,0.089,,240624,,,A*77\r\n'
b'$GPVTG,,T,,M,0.089,N,0.165,K,A*20\r\n'
b'$GPGGA,050749.00,1911.27854,N,07303.02284,E,1,04,4.60,21.7,M,-65.5,M,*45\r\n'
b'$GPGSV,2,2,06,20,53,037,,25,58,291,35*79\r\n'
b'$GPGLL,1911.27854,N,07303.02253,E,050750.00,A,A*6B\r\n'
b'050756.00,A,A*6C\r\n'
b'$GPRMC,050757.00,A,1911.27909,N,07303.02228,E,0.351,,240624,,,A*71\r\n'
b'$GPVTG,,T,,M,0.351,N,0.651,K,A*26\r\n'
b'$GPGGA,050757.00,1911.27909,N,07303.02228,E,1,0408.00,A,A*69\r\n'
b'$GPRMC,050809.00,A,1911.27830,N,07303.02120,E,0.043,,240624,,,A*75\r\n'
b'$GPVTG,,T,,M,0.043,N,0.080,K,A*2C\r\n'
b'$GPGGA,050809.00,1911.27830,N,07303.02120,E,1,04,4.57,24.0,M,-65.5,M,*47\r\n'
b'$GPGSA,A,3,05,18,25,12,,,,6.50,4.57,4.63*0E\r\n'
b'$GPGSV,2,1,06,05,81,085,40,12,07303.02103,E,050818.00,A,A*62\r\n'
b'$GPRMC,050819.00,A,1911.27745,N,07303.02097,E,0.105,,240624,,,A*77\r\n'

```

There are multiple formats of **NMEA sentences, which start with the '\$' character.** We can see satellite data received in the form of NMEA sentences. For more information, refer to the beginning of the Theory section of this chapter. (NMS...)

- 2) Output On OLED Screen Text Format: -



### GPS Data After Processing by Above Code and Fetching from OpenWeather API

```
{
  'time_utc': '072101.00', 'date': '240624', 'latitude': 19.18818, 'longitude': 73.05016, 'speed': '1.732'}
  {'time_utc': '072101.00', 'date': '240624', 'latitude': 19.18818, 'longitude': 73.05016, 'speed': '1.732'}
In API CALL FUNCTION..
{'pressure': 1001, 'temp': 31.05002, 'humidity': 73, 'wind_speed': 4.46}
{'time_utc': '072101.00', 'date': '240624', 'latitude': 19.18818, 'longitude': 73.05016, 'speed': '1.732'}
{'time_utc': '072106.00', 'date': '240624', 'latitude': 19.18819, 'longitude': 73.05016, 'speed': '1.732'}
{'time_utc': '072107.00', 'date': '240624', 'latitude': 19.18819, 'longitude': 73.05016, 'speed': '0.063'}
```

Initially Receiving NULL Values from GPS (For 1<sup>st</sup> 25 to 30 seconds. Depends on Signal Strength)



After Some Time, It Starts Receiving Correct Values.



After adding images and symbols, it looks a bit more graphical.

➤ **Code:** - (Code to display **Image and Text data** on **OLED Display**): -

```
import machine
import sh1106
import framebuf
import network
import time
import urequests
```

```

from machine import UART, Pin

# Network and API
SSID = 'Mohan Samsung'
PASS = 'mohanShikare1252345'
API_KEY = '7444ejshbjh5456486edc6b7aef980be' # Replace with your OpenweatherAPIKEY

# I2C Configuration using SoftI2C
i2c = machine.SoftI2C(scl=machine.Pin(22), sda=machine.Pin(21))
oled = sh1106.SH1106_I2C(128, 64, i2c)
oled.rotate(True)
uart = UART(1, baudrate=9600, tx=Pin(17), rx=Pin(16))
gps_data = {
    'latitude': "NA",
    'longitude': "NA",
    'speed': "NA",
    'time_utc': "NA",
    'date': "NA"
}
weather_data = {
    'temp': "NA",
    'pressure': "NA",
    'humidity': "NA",
    'wind_speed': "NA"
}

class Network_API:
    wlan = None # Store the WLAN object

    @staticmethod
    def connect_to_wifi(ssid, password):
        Network_API.wlan = network.WLAN(network.STA_IF)
        Network_API.wlan.active(True)
        Network_API.wlan.connect(ssid, password)

        # Define the WIFI_BITMAP
        WIFI_BITMAP = bytearray([
            0x01, 0xfe, 0x00, 0x0e, 0x01, 0xc0, 0x10, 0x00, 0x20, 0x60, 0x00, 0x18, 0x80,
            0x00, 0x04, 0x01,
            0xce, 0x00, 0x06, 0x01, 0x80, 0x08, 0x00, 0x40, 0x00, 0x00, 0x00, 0x00, 0x78,
            0x00, 0x01, 0x86,
            0x00, 0x00, 0x00, 0x00, 0x30, 0x00, 0x00, 0x48, 0x00, 0x00, 0x48, 0x00
        ])

        # Create a framebuffer for the image
        fb = framebuf.FrameBuffer(WIFI_BITMAP, 22, 16, framebuf.MONO_HLSB)

        # Wait for the connection to establish
        oled.fill(0)
        oled.text("Connecting WI-FI..", 0, 0)
        oled.blit(fb, 100, 25)
        oled.show()

```

```

        while not Network_API.wlan.isconnected():
            print("Connecting..")
            time.sleep(1)

            oled.text("Connected..", 0, 20)
            oled.text(f"IP -> {Network_API.wlan.ifconfig()[0]}", 0, 50)
            print(Network_API.wlan.ifconfig())
            oled.show()
            time.sleep(2)

    @staticmethod
    def get_API_Data(api_key , latitude , longitude):
        print("In API CALL FUNCTION..")
        if not Network_API.wlan.isconnected():
            return
        global weather_data
        try:
            url =
f'https://api.openweathermap.org/data/2.5/weather?lat={latitude}&lon={longitude}&appid={api
_key}'
            response = urequests.get(url)
            data = response.json()

            weather_data['temp'] = (data['main']['temp'])-273.15
            weather_data['pressure'] = data['main']['pressure']
            weather_data['humidity'] = data['main']['humidity']
            weather_data['wind_speed'] = data['wind']['speed']
            print(weather_data) # Print the weather data to see updates
        except Exception as e:
            print(f"Error fetching weather data: {e}")

class GPS_Data:
    @staticmethod
    def read_gps():
        global gps_data
        try:
            line = uart.readline()
            if line is None:
                return
            line = line.decode('utf-8').strip()
            if line.startswith('$GPGLL'):
                GPS_Data.parse_gpgll(line)
            elif line.startswith('$GPRMC'):
                GPS_Data.parse_gprmc(line)
            elif line.startswith('$GPVTG'):
                GPS_Data.parse_gpvtg(line)
            print(gps_data) # Print the dictionary to see updates
        except Exception as e:
            print(f"Error reading GPS: {e}")

    @staticmethod
    def parse_gpgll(line):

```

```

global gps_data
parts = line.split(',')
if len(parts) < 7:
    print("Incomplete GPGLL data")
    return
latitude = GPS_Data.convert_to_degrees(parts[1], parts[2])
longitude = GPS_Data.convert_to_degrees(parts[3], parts[4])
time_utc = parts[5]

if latitude is not None and longitude is not None:
    gps_data['latitude'] = latitude
    gps_data['longitude'] = longitude
if time_utc:
    gps_data['time_utc'] = time_utc

@staticmethod
def parse_gprmc(line):
    global gps_data
    parts = line.split(',')
    if len(parts) < 12:
        print("Incomplete GPRMC data")
        return

    time_utc = parts[1]
    latitude = GPS_Data.convert_to_degrees(parts[3], parts[4])
    longitude = GPS_Data.convert_to_degrees(parts[5], parts[6])
    speed = parts[7]
    date = parts[9]

    if time_utc:
        gps_data['time_utc'] = time_utc
    if latitude is not None:
        gps_data['latitude'] = latitude
    if longitude is not None:
        gps_data['longitude'] = longitude
    if speed:
        gps_data['speed'] = speed
    if date:
        gps_data['date'] = date

@staticmethod
def parse_gpvtg(line):
    global gps_data
    parts = line.split(',')
    if len(parts) < 9:
        print("Incomplete GPVTG data")
        return

    speed = parts[7] # Speed in knots

    if speed:
        gps_data['speed'] = speed

```

```

@staticmethod
def convert_to_degrees(value, direction):
    try:
        if not value or not direction:
            return None

        # Determine if it's latitude or longitude based on the direction
        if direction in ['N', 'S']:
            degrees = float(value[:2])
            minutes = float(value[2:])
        elif direction in ['E', 'W']:
            degrees = float(value[:3])
            minutes = float(value[3:])

        result = degrees + (minutes / 60)

        if direction in ['S', 'W']:
            result = -result

        return round(result, 6)
    except ValueError as e:
        print(f"Error converting to degrees: {e}")
        return None

def format_time(time_utc):
    if time_utc and len(time_utc) >= 6:
        hours = time_utc[:2]
        minutes = time_utc[2:4]
        seconds = time_utc[4:6]
        return f"{hours}:{minutes}:{seconds}"
    return "N/A"

def format_date(date):
    if date and len(date) == 6:
        day = date[:2]
        month = date[2:4]
        year = date[4:6]
        return f"{day}-{month}-{year}"
    return "N/A"

def draw_rect_line():
    oled.fill(0)
    oled.rect(0, 0, 127, 63, 1)
    oled.line(0, 31, 127, 31, 1)

def show_data(screen_counter):
    draw_rect_line()
    # Persist one screen for 5 seconds
    if screen_counter < 5: # for GPS data
        #<-----BitMap Images ByteArray----- Start----->
        # Define Latitude Long
        LAT_LONG_BITMAP = bytearray([

```

```

        0x00, 0x00, 0x00, 0x00, 0x00, 0x3e, 0x00, 0x00, 0x01, 0xff, 0xc0, 0x00, 0x03, 0xeb,
0xf0, 0x00,
        0xe, 0x49, 0xb8, 0x00, 0xc, 0xc8, 0x9c, 0x00, 0x18, 0x88, 0x8e, 0x00, 0x30, 0x88,
0x46, 0x00,
        0x3f, 0xff, 0xff, 0x00, 0x61, 0x08, 0x03, 0x00, 0x61, 0x08, 0x01, 0x00, 0x61, 0x08,
0x60, 0x00,
        0x61, 0x91, 0xf8, 0x00, 0x7f, 0xe1, 0x8c, 0x00, 0x61, 0x03, 0x06, 0x00, 0x61, 0x03,
0x06, 0x00,
        0x61, 0x03, 0x06, 0x00, 0x3f, 0xc3, 0x04, 0x00, 0x30, 0x81, 0xdc, 0x00, 0x18, 0x81,
0xf8, 0x00,
        0xc, 0xc0, 0xf8, 0x00, 0x06, 0x40, 0xf0, 0x00, 0x03, 0xf0, 0x70, 0x00, 0x00, 0xf8,
0x60, 0x00,
        0x00, 0x1c, 0x00, 0x00
    ])

#define Ground Speed Bitmap
GROUND_SPEED_BITMAP = bytarray([
    0x00, 0x00, 0x00, 0x00, 0x00, 0x3e, 0x00, 0x00, 0x01, 0xc1, 0xc0, 0x00, 0x02, 0x0c,
0x60, 0x00,
    0x04, 0xeb, 0x90, 0x00, 0x09, 0x88, 0xc8, 0x00, 0x12, 0x80, 0xa4, 0x00, 0x24, 0x00,
0x12, 0x00,
    0x28, 0x00, 0xa, 0x00, 0x4c, 0x08, 0x18, 0x00, 0x50, 0x18, 0x05, 0x00, 0x50, 0x14,
0x05, 0x00,
    0x50, 0x14, 0x05, 0x00, 0x10, 0x22, 0x05, 0x00, 0x18, 0x22, 0xd, 0x00, 0x40, 0xa,
0x05, 0x00,
    0x50, 0x14, 0x05, 0x00, 0x50, 0x2a, 0x05, 0x00, 0x54, 0x14, 0x15, 0x00, 0x28, 0x00,
0xa, 0x00,
    0x00, 0x00, 0x02, 0x00, 0x00,
0x00, 0x00,
    0x00, 0x00, 0x00, 0x00
])

#define Date Bitmap
DATE_BITMAP = bytarray([
    0x02, 0x00, 0x40, 0x00, 0x06, 0x00, 0x60, 0x00, 0x16, 0x90, 0xe8, 0x00, 0x7e, 0xff,
0xee, 0x00,
    0x7f, 0xff, 0xfe, 0x00, 0x7f, 0xff, 0xfe, 0x00, 0x7f, 0xff, 0xfe, 0x00, 0x7f, 0xff,
0xfe, 0x00,
    0x7f, 0xff, 0xfe, 0x00, 0x7f, 0xff, 0xfe, 0x00, 0x60, 0x00, 0x06, 0x00, 0x67, 0x38,
0xc0, 0x00,
    0x67, 0x38, 0x3c, 0x00, 0x60, 0x00, 0x7e, 0x00, 0x60, 0x00, 0xc3, 0x00, 0x67, 0x18,
0xc3, 0x00,
    0x67, 0x38, 0xc3, 0x00, 0x60, 0x00, 0xc3, 0x00, 0x60, 0x00, 0x67, 0x00, 0x7f, 0xff,
0x7e, 0x00,
    0x7f, 0xff, 0xbc, 0x00, 0x3f, 0xff, 0x9c, 0x00, 0x00, 0x00, 0x18, 0x00, 0x00, 0x00,
0x34, 0x00,
    0x00, 0x00, 0x3c, 0x00
])

#define Time Bitmap
TIME_BITMAP = bytarray([
    0x00, 0xf8, 0x00, 0x00, 0x03, 0x06, 0x00, 0x00, 0x0c, 0x01, 0x00, 0x00, 0x08, 0x00,
0x80, 0x00,

```

```

        0x10, 0x00, 0x40, 0x00, 0x10, 0x00, 0x40, 0x00, 0x20, 0x20, 0x20, 0x00, 0x20, 0x50,
0x20, 0x00,
        0x20, 0x80, 0x20, 0x00, 0x20, 0x70, 0x20, 0x00, 0x20, 0x00, 0x40, 0x00, 0x10, 0x00,
0x00, 0x00,
        0x10, 0x00, 0x00, 0x00, 0x08, 0x01, 0xdc, 0x00, 0x04, 0x03, 0x02, 0x00, 0x04, 0x06,
0x21, 0x00,
        0x02, 0x04, 0x21, 0x00, 0x01, 0x04, 0x20, 0x80, 0x01, 0x04, 0x3c, 0x80, 0x00, 0x84,
0x00, 0x80,
        0x00, 0x84, 0x00, 0x80, 0x00, 0x54, 0x01, 0x00, 0x00, 0x72, 0x03, 0x00, 0x00, 0x21,
0x86, 0x00,
        0x00, 0x20, 0x78, 0x00
    ])

#<-----BitMap Images ByteArray----->
image_list = [LAT_LONG_BITMAP, GROUND_SPEED_BITMAP, DATE_BITMAP, TIME_BITMAP]
temp_ctr=5
for i in image_list:
    fb = framebuffer.FrameBuffer(i, 25, 25, framebuffer.MONO_HLSB)
    oled.blit(fb, temp_ctr, 4)
    temp_ctr+=30

#<-----Show Data ----->

if counter%2==0:
    oled.text(f'L:{str(gps_data['latitude'])[:2]}', 2, 35)
else:
    oled.text(f'G:{str(gps_data['longitude'])[:2]}', 2, 35)

oled.text(f'{str(gps_data['speed'])[:4]}K ", 2, 50)
if gps_data['date']:
    oled.text(f'{format_date(gps_data['date'])}', 46, 35)
else:
    oled.text("N/A", 46, 35)
if gps_data['time_utc']:
    oled.text(f'{format_time(gps_data['time_utc'])}', 50, 50)
else:
    oled.text("N/A", 50, 50)

elif screen_counter>=5 and screen_counter <= 11: # for weather data
#<-----BitMap Images ByteArray----->
    # Define the Humidity
    HUMIDITY_BITMAP = bytearray([
        0x00, 0x00, 0x00, 0x00, 0x60, 0x00, 0x00, 0x00, 0xf0, 0x00, 0x00, 0x01,
0xf8, 0x00, 0x00,
        0x01, 0xfc, 0x00, 0x00, 0x03, 0xfc, 0x00, 0x00, 0x07, 0xfe, 0x00, 0x00, 0x0f, 0xff,
0x00, 0x00,
        0x0f, 0xff, 0x00, 0x00, 0x1f, 0xff, 0x80, 0x00, 0x1f, 0xff, 0xc0, 0x00, 0x3f, 0xff,
0x00, 0x00,
        0x3f, 0xfc, 0x00, 0x00, 0x7f, 0xf9, 0xfc, 0x00, 0x7f, 0xf3, 0xfe, 0x00, 0x77, 0xf3,
0x77, 0x00,
        0x77, 0xf6, 0x67, 0x00, 0x33, 0xe7, 0xc7, 0x00, 0x33, 0xe7, 0x8f, 0x00, 0x38, 0xe7,
0x17, 0x00,
    ])

```

```

        0x1c, 0x77, 0x33, 0x00, 0x0f, 0xf3, 0xfe, 0x00, 0x07, 0xfb, 0xfe, 0x00, 0x01, 0xf8,
0xf8, 0x00,
        0x00, 0x00, 0x00, 0x00
    ])

#define Temperature Bitmap
TEMP_BITMAP = bytearray([
    0x00, 0x18, 0x00, 0x00, 0x00, 0x7c, 0x00, 0x00, 0x00, 0x66, 0x00, 0x00, 0x00, 0x42,
0x78, 0x00,
    0x00, 0x46, 0x00, 0x00, 0x00, 0x46, 0x00, 0x00, 0x00, 0x42, 0x78, 0x00, 0x00, 0x46,
0x00, 0x00,
    0x00, 0x42, 0x00, 0x00, 0x00, 0x5a, 0x78, 0x00, 0x00, 0x5a, 0x00, 0x00, 0x00, 0x5a,
0x00, 0x00,
    0x00, 0x5a, 0x78, 0x00, 0xda, 0x00, 0x00, 0x01, 0x9b, 0x80, 0x00, 0x03, 0x3d,
0x80, 0x00,
    0x03, 0x7e, 0xc0, 0x00, 0x02, 0xff, 0x40, 0x00, 0x02, 0xff, 0x40, 0x00, 0x02, 0xff,
0x40, 0x00,
    0x03, 0x7e, 0xc0, 0x00, 0x01, 0x3d, 0x80, 0x00, 0x01, 0xc3, 0x00, 0x00, 0x00, 0x7e,
0x00, 0x00,
    0x00, 0x00, 0x00, 0x00
])

#define Pressure Bitmap
PRESSURE_BITMAP = bytearray([
    0x00, 0xfe, 0x00, 0x00, 0x03, 0x81, 0xc0, 0x00, 0x0c, 0xff, 0x20, 0x00, 0x1b, 0x10,
0xd0, 0x00,
    0x35, 0x91, 0x68, 0x00, 0x28, 0x91, 0x34, 0x00, 0x58, 0x42, 0x1e, 0x00, 0x56, 0x00,
0x6a, 0x00,
    0xf2, 0x01, 0xca, 0x00, 0xa0, 0x1b, 0x07, 0x00, 0xac, 0x2c, 0x75, 0x00, 0xac, 0x7c,
0x35, 0x00,
    0xa3, 0xd8, 0x07, 0x00, 0xf1, 0x80, 0xa, 0x00, 0x50, 0x80, 0xa, 0x00, 0x58, 0x00,
0x16, 0x00,
    0x28, 0x00, 0x34, 0x00, 0x16, 0x00, 0x68, 0x00, 0x1b, 0x81, 0xd0, 0x00, 0x04, 0xff,
0x60, 0x00,
    0x03, 0x81, 0xc0, 0x00, 0x01, 0xff, 0x80, 0x00, 0x01, 0x81, 0x80, 0x00, 0x00, 0xff,
0x00, 0x00,
    0x00, 0x42, 0x00, 0x00
])

#define Wind Speed Bitmap
WIND_SPEED_BITMAP = bytearray([
    0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0x00, 0x00, 0x01, 0xfe, 0x00, 0x00, 0x03, 0xff,
0x00, 0x00,
    0x03, 0xff, 0x80, 0x00, 0x07, 0xff, 0xf8, 0x00, 0x07, 0xff, 0xfc, 0x00, 0x07, 0xff,
0xfe, 0x00,
    0x1f, 0xe7, 0xff, 0x00, 0x3f, 0xc1, 0xff, 0x00, 0x7f, 0xbd, 0xff, 0x00, 0x7f, 0xa6,
0xff, 0x00,
    0x7f, 0x82, 0xff, 0x00, 0x70, 0x06, 0xff, 0x00, 0x77, 0xfd, 0xff, 0x00, 0x70, 0x00,
0x7e, 0x00,
    0x20, 0x00, 0x1c, 0x00, 0x0f, 0xff, 0xe8, 0x00, 0x00, 0x00, 0x20, 0x00, 0x1f, 0xc0,
0x10, 0x00,
    0x1f, 0xe0, 0x10, 0x00, 0x00, 0x26, 0x10, 0x00, 0x03, 0x62, 0x20, 0x00, 0x01, 0xc3,
0xe0, 0x00,
])

```

```

    0x00, 0x00, 0x00, 0x00
])
#<-----BitMap Images ByteArray----- END----->
image_list = [TEMP_BITMAP,HUMIDITY_BITMAP,PRESSURE_BITMAP,WIND_SPEED_BITMAP]
temp_ctr=5
for i in image_list:
    fb = framebuffer.FrameBuffer(i, 25, 25, framebuffer.MONO_HLSB)
    oled.blit(fb, temp_ctr, 4)
    temp_ctr+=30
#----- Show Data -----
oled.text(f"{str(weather_data['temp'])[:5]} C",3,35)
oled.text(f"{weather_data['pressure']} hPa",3,50)
oled.text(f"{weather_data['humidity']} %",70,35)
oled.text(f"{weather_data['wind_speed']}m/s",70,50)

else:
    screen_counter = 0
oled.show()
return screen_counter

# Connect to WiFi
Network_API.connect_to_wifi(SSID, PASS)
screen_counter = 0
counter=0
while True: # Main loop
    GPS_Data.read_gps()
    if counter > 10:
        if gps_data['latitude'] is not None and gps_data['longitude'] is not None:
            Network_API.get_API_Data(API_KEY, gps_data['latitude'], gps_data['longitude'])
            counter = 0
    screen_counter = show_data(screen_counter)
    screen_counter += 1
    counter+=1
    time.sleep(2.5)

```

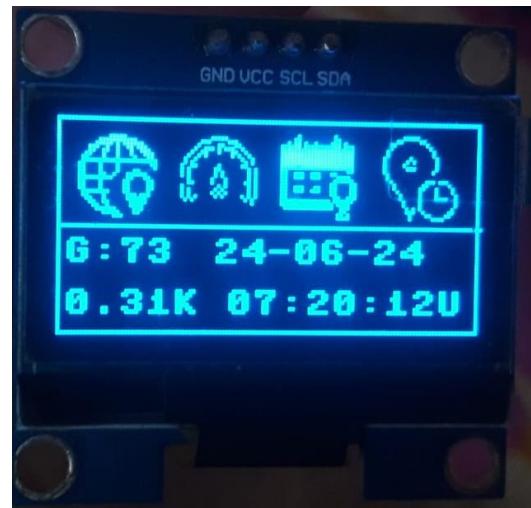
- **Output Photo:** - (After adding **images and symbols**, it looks a bit more **graphical**)
  - 1) Output On **OLED Screen** Image Format: -

Initially **Receiving NULL Values** from GPS (For 1<sup>st</sup> 25 to 30 seconds. Depends on Signal Strength)



2) After getting all values from GPS and OpenWeather API

**GPS Latitude(L) And UTC Date, Time, Speed on Ground in KNOTS Longitude(G)**



3) OpenWeather API Data: - (GPS Location Temperature, Humidity, Pressure, Wind Speed Data)



For any correction/suggestion/feedback: - Email: - [nikhil.shikare@vit.edu.in](mailto:nikhil.shikare@vit.edu.in)



Linkdin : - <https://www.linkedin.com/in/nikhil-shikare-18127a216/>



## 39) Sample Projects for Reference

### Simple Project Photo -1



### Simple Project Photo - 2



for any **Correction/Suggestion/Feedback**,

**Prof. Kanchan Dhuri:** - [kanchan.dhuri@vit.edu.in](mailto:kanchan.dhuri@vit.edu.in)

**Prof. Ajitkumar Khachane:** - [ajitkumar.khachane@vit.edu.in](mailto:ajitkumar.khachane@vit.edu.in)

## 40) Online References links

**➤ Information taken from the following websites...**

- 1) <https://chat.openai.com/> -
- 2) <https://arduino.stackexchange.com/>
- 3) <https://forum.arduino.cc/>
- 4) <https://docs.arduino.cc/>
- 5) <https://store.arduino.cc/>
- 6) <https://store.micropython.org/>
- 7) <https://forum.micropython.org/>
- 8) <https://docs.micropython.org/en/latest/>
- 9) <https://micropython.org/>
- 10) <https://projecthub.arduino.cc/>
- 11) <https://circuitpython.org/>
- 12) <https://www.arduino.cc/>
- 13) <https://www.arduinolibraries.info/libraries>
- 14) <https://thingspeak.com/>
- 15) <https://www.silabs.com/>
- 16) <https://www.adafruit.com/> - Adafruit Industries
- 17) <https://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use/arduino-libraries>

**➤ Images were taken from the following websites.**

- 18) <https://www.raspberrypi.org/>
- 19) <https://github.com/>
- 20) <https://www.tinkercad.com/>
- 21) <https://www.arduino.cc/>
- 22) <https://forum.arduino.cc/>
- 23) <https://www.silabs.com/>
- 24) <https://robu.in/arduino-pin-configuration/>
- 25) [https://en.m.wikipedia.org/wiki/File:Logo\\_tinkercad-wordmark.svg](https://en.m.wikipedia.org/wiki/File:Logo_tinkercad-wordmark.svg)
- 26) [https://www.researchgate.net/figure/ARDUINO-UNO-SPECIFICATIONS\\_tbl1\\_304412762](https://www.researchgate.net/figure/ARDUINO-UNO-SPECIFICATIONS_tbl1_304412762)
- 27) <https://binaryupdates.com/adc-in-lpc2148-arm7-microcontroller/>
- 28) [Arduino MQ5 Gas Sensor Tutorial - How MQ5 Gas Sensor Works and Interfacing MQ5 Gas Sensor with Arduino \(circuitdigest.com\)](https://circuitdigest.com/arduino/mq5-gas-sensor-tutorial-how-mq5-gas-sensor-works-and-interfacing-mq5-gas-sensor-with-arduino)
- 29) <https://www.aliexpress.com/item/16pcs-lot-Raspberry-Pi-3-Sensor-Module-Sensor-kit-for-Raspberry-Pi-3B-for-DIY-Extension/32966927875.html>
- 30) <https://www.flipkart.com/ds-robotics-9pcs-lot-gas-detection-sensor-module-mq-2-mq-3-mq-4-mq-5-mq-6-mq-7-mq-8-mq-9-mq-135>
- 31) <https://www.amazon.in/Optical-Fingerprint-Reader-Sensor-Module/dp/B08WKK1QMS>
- 32) <https://lastminuteengineers.com/i2c-lcd-arduino-tutorial/>
- 33) <https://www.espressif.com/en>
- 34) <https://thingspeak.com/>
- 35) <https://www.alldatasheet.com/>
- 36) <https://blynk.io/>