WhiteHat Jr
Live online Coding for Kids

Topic	MONITORING SYSTEM- 2	
Class Description	**Students will be introduced to how to interface the BMP180 pressure sensor with ESP32 and they will learn to publish and subscription technique used in sending and receiving messages on the cloud server**	
Class	**PRO C249**	
Class time	**45 mins**	
Goal	● Introduction to Publish & Subscribe ● Send data to the server ● Receive data from the server	
Resources Required	● Teacher Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Smartphone ● Student Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen	
Class structure	**Warm-Up** **Teacher-Led Activity** **Student-Led Activity** **Wrap-Up**	**10 mins** **20 mins** **20 mins** **05 mins**

WARM-UP SESSION - 10 mins

Teacher Action	**Student Action**

Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today? **Following are the WARM-UP session deliverables:** ● Greet the student. ● Revision of previous class activities. ● Quizzes.	**ESR**: Hi, thanks! Yes, I am excited about it! Click on the slide show tab and present the slides
WARM-UP QUIZ Click on In-Class Quiz	
Activity Details **Following are the session deliverables:** ● Appreciate the student. ● Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.	
TEACHER-LED ACTIVITY - 10 mins	
Teacher Initiates Screen Share	
ACTIVITY ● **Introduction to libraries** ● **Publish & Subscribe**	
Teacher Action	**Student Action**
In the last class we designed a cloud server where we set gauges and toggle switches. But we haven't seen how this really works. Today we will send real-time sensor data on the server and receive the real-time data from the cloud server and with that, we can control our end devices like Led's So let's learn how to interface sensors and end devices	

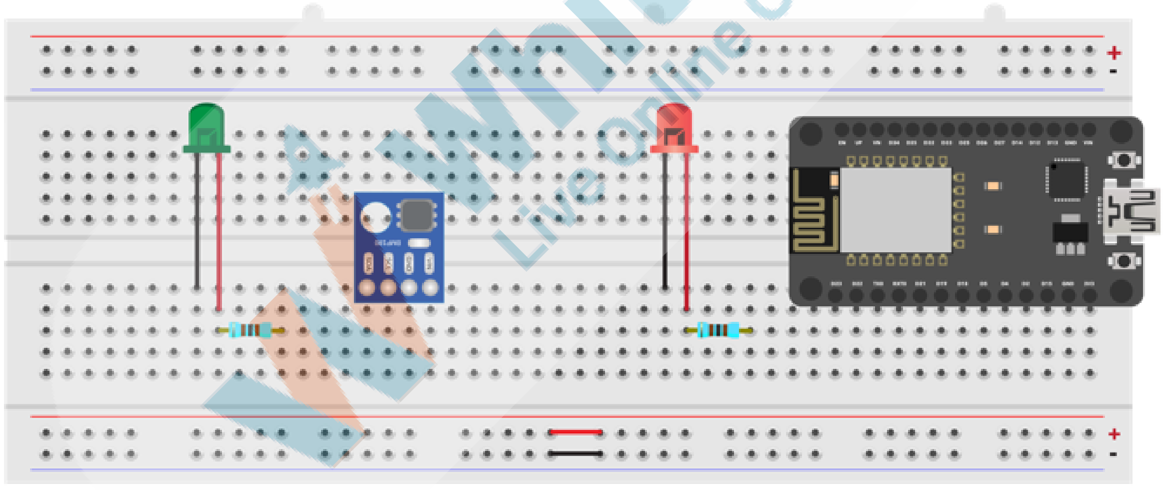
with the cloud server.

Note: The teacher will guide the student in making connections: This will be the same connection that we did in the last class.

Step -1:Gather the material from the IoT kit:

- 1 x ESP32
- 1 x USB Cable
- 1 x Breadboard
- 4 x Jumper wires
- 1 x BMP180
- 2 x LED
- 2 x Resistors

Mount the components



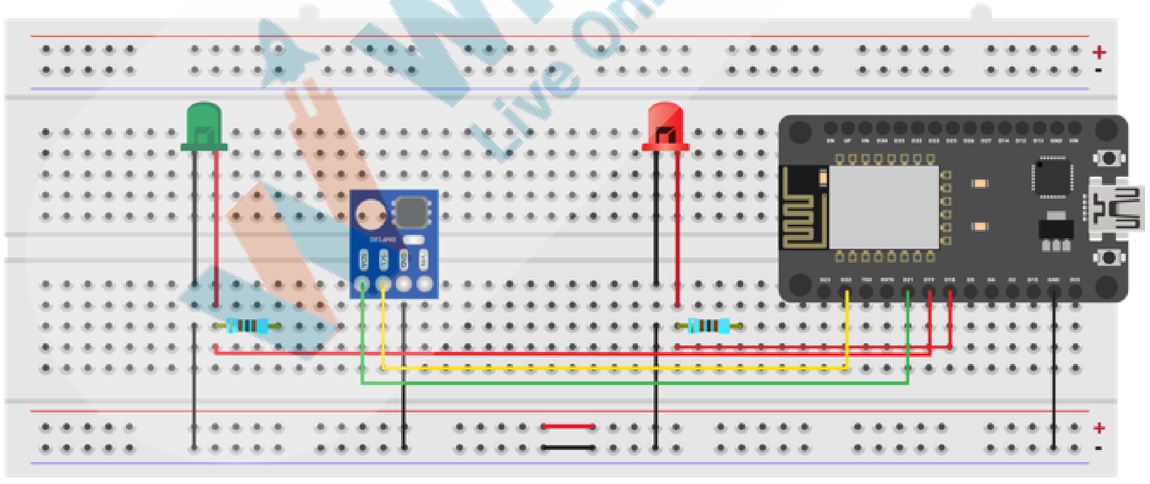
Step -2: Let's do connections:

Note : Follow the same pins number as mentioned below to connect the sensor with ESP32

BMP180 Pins	**Wiring Connections**
VCC	Connect with **3V3 PIN** of the ESP32
GND	Connect with **GND** of the ESP32
SCL	Connect with GPIO **PIN 22**
SDA	Connect with GPIO **PIN 21**

LED Connections: Connect positive leg of **LED** with the resistor and negative with **GND(0V).**
Another end of the resistor with **GPIO pin 18**.

Do the same connection for the second **LED**, but this time connects with **GPIO pin 19**



🟩 SDA - D21 🟨 SCL - D22 🟥 RED LED - D18 🟩 GREEN LED - D19

Let’s write a program

The top of the program has the includes. We will write all supporting header files and libraries

include keyword is used to import libraries in embedded language as we used to import in python language

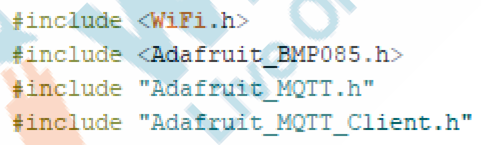
WiFi.h: WiFi library will be able to answer all HTTP request

Adafruit BMP085.h Adafruit **BMP085** is used to connect the BMP180 sensor with the Adafruit dashboard**.**

Adafruit supports a protocol called **MQTT**, or **message queue telemetry transport**, for communication with devices. To send and receive feed data,

Adafruit_MQTT header file tells about sending and receiving packets.

Adafruit_MQTT_Client allows you to publish to a feed.



```
#include <WiFi.h>
#include <Adafruit_BMP085.h>
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"
```

After uploading libraries the next step is to connect with ESP32 with the WiFi. For that, we need to use SSID(Wi-Fi credentials i.e WiFi name and WiFi Password)

- **WLAN_SSID, WLAN_PASS** are the variables that are used to save WiFi credentials. Set the **SSID** and **password**

Note: Teacher/Student should use their actual WIFi Credentials.

```
#define WLAN_SSID       "WR3005N3-757E"
#define WLAN_PASS       "70029949"
```

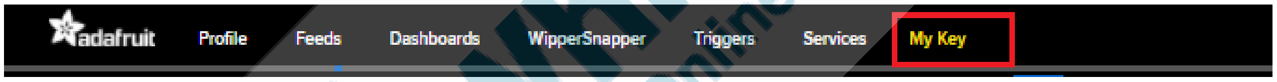
Adafruit Setup and Authentication

Set the Adafruiit **AIO_SERVER , AIO_SERVERPORT, AIO_USERNAME, AIO_KEY**

Note: Red highlighted one need to change as per teacher and student adafruit credentials.

```
#define AIO_SERVER      "io.adafruit.com"
#define AIO_SERVERPORT  1883                   // use 8883 for SSL
#define AIO_USERNAME    "Tamtap"
#define AIO_KEY         "aio_ozIw67GZ2KKiwjN7Uvz5HDQeiuY0"
```

To get **AIO_USERNAME** & **AIO_KEY go to adafruit,** Click on **My Key**

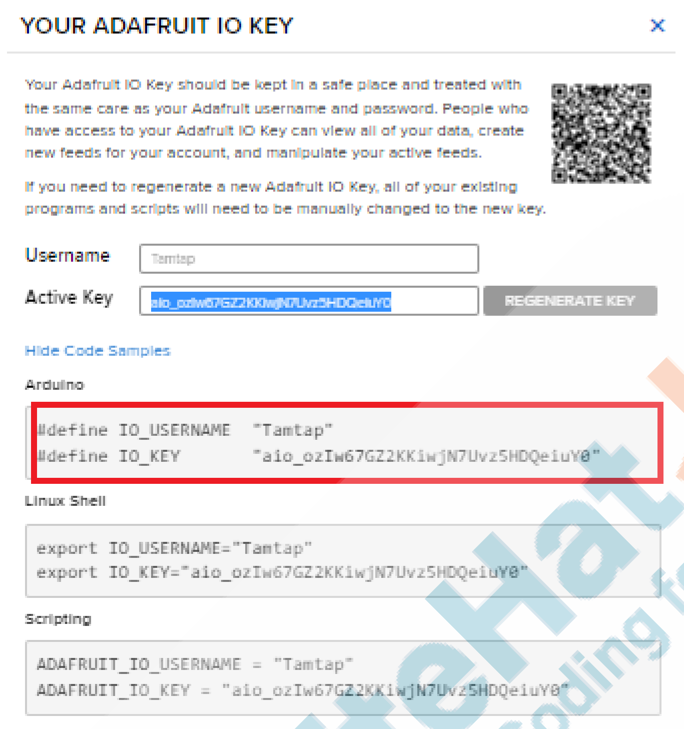


After clicking on **My Key**, the following window will appear.

Key - This is a long, unique identifier that is used to authenticate devices using an Adafruit account.

Note down **IO_USERNAME** & **IO_KEY,** paste the same in the program where **AIO_USERNAME** & **AIO_KEY is mentioned.**

Note: *A very important step to connect the Adafruit server with ESP32*


YOUR ADAFRUIT IO KEY

Your Adafruit IO Key should be kept in a safe place and treated with the same care as your Adafruit username and password. People who have access to your Adafruit IO Key can view all of your data, create new feeds for your account, and manipulate your active feeds.

If you need to regenerate a new Adafruit IO Key, all of your existing programs and scripts will need to be manually changed to the new key.

Username: Tamtap

Active Key: aio_ozIw67GZ2KKiwjN7Uvz5HDQeiuY0

Arduino

```
#define IO_USERNAME  "Tamtap"
#define IO_KEY       "aio_ozIw67GZ2KKiwjN7Uvz5HDQeiuY0"
```

Linux Shell

```
export IO_USERNAME="Tamtap"
export IO_KEY="aio_ozIw67GZ2KKiwjN7Uvz5HDQeiuY0"
```

Scripting

```
ADAFRUIT_IO_USERNAME = "Tamtap"
ADAFRUIT_IO_KEY = "aio_ozIw67GZ2KKiwjN7Uvz5HDQeiuY0"
```

Publish & Subscribe

Publish - push data from device to server for example. Sensor Data

Subscribe- push data from server to device for example. LED Control

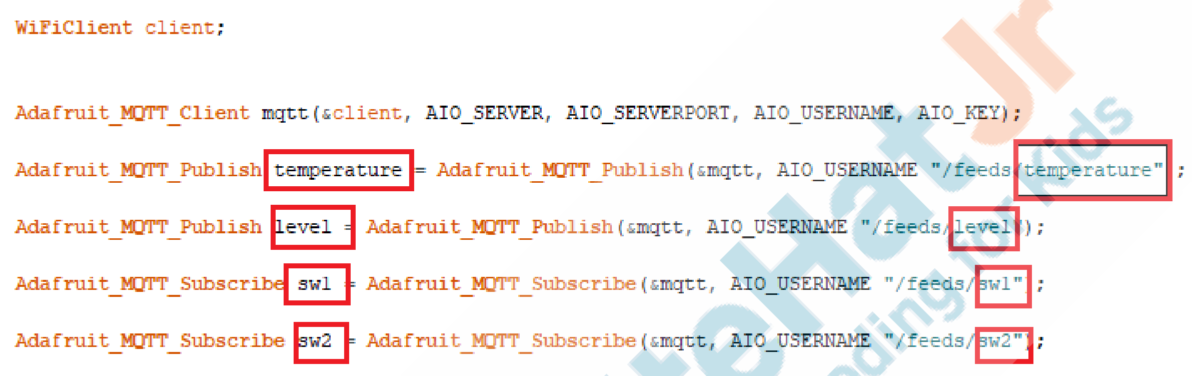
Setup the MQTT client class by passing is Adafruit server set up details like **AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY**

BMP180 which is used to sense temperature and pressure will use **Adafruit_MQTT_Publish** publish data, and to control LED from the server we use **Adafruit_MQTT_Subscribe**

AIO_USERNAME/feeds/temperature is feed name

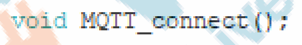
Note: Below highlighted needs to be changed as per the feed name, which student/teacher has entered in very first step i.e Enter Feed Name

In the below screenshot, its name as temperature for Room Temperature, level for Pressure, sw1 for Room AC, sw2 for Room Light



```
WiFiClient client;


Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);

Adafruit_MQTT_Publish temperature = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/temperature");

Adafruit_MQTT_Publish level = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/level");

Adafruit_MQTT_Subscribe sw1 = Adafruit_MQTT_Subscribe(&mqtt, AIO_USERNAME "/feeds/sw1");

Adafruit_MQTT_Subscribe sw2 = Adafruit_MQTT_Subscribe(&mqtt, AIO_USERNAME "/feeds/sw2");
```

MQTT_ connect will establish a connection og ESp32 with cloud server



```
void MQTT_connect();
```

Note: Teacher start writing code from here

Define **GPIO pins**

- **Define LED's pin** along with data type **int**
- Define Variable **p** with datatype **float**
- Define **String** variable to store value
- Define **bmp** object for **Adafruit_BMP085**

```
const int led1 = 18;
const int led2 = 19;

float p;

String stringOne, stringTwo;

Adafruit_BMP085 bmp;
```

Initialize the setup()

- **Serial. begin(9600)** is used for data exchange speed. speed parameters. This tells the Arduino to get ready to exchange messages with the Serial Monitor at a data rate of 9600 bits per second. That's 9600 binary ones or zeros per second and is commonly called a baud rate.

- Set a **delay** of **10** ms

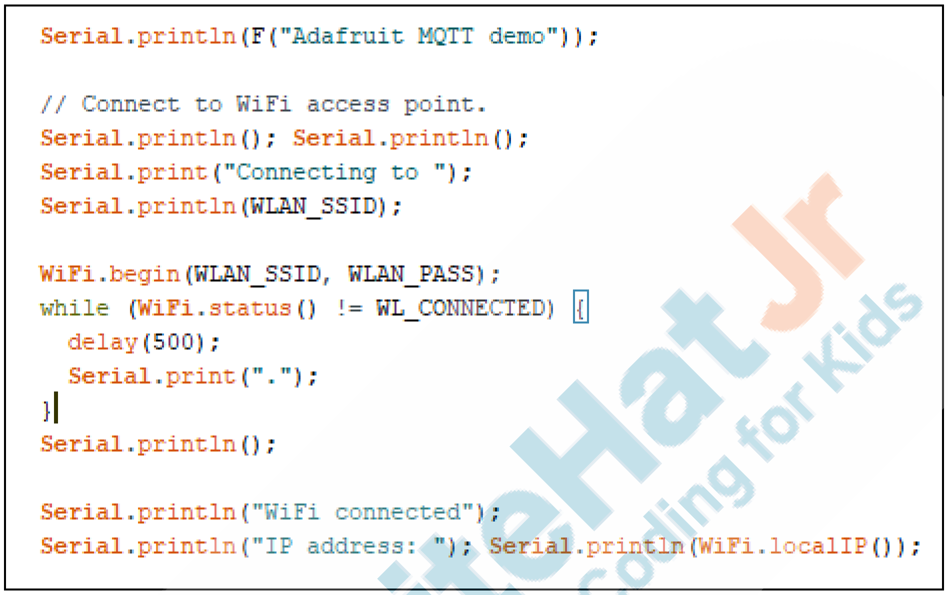
- **PinMode()** configures the specified pin to behave either as an input or an output. As we want to act as output we are writing **OUTPUT** here. Set the pinMode for both **Led1, Led2**

- **digitalWrite()** function helps to change the state of LED from **HIGH to LOW** or vice versa

```
void setup() {
  Serial.begin(9600);
  delay(10);

  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);

  digitalWrite(led1, LOW);
  digitalWrite(led2, LOW);
```

Serial. println is used to print the statement

This section will show details while connecting to the Internet. Basically, it will show when it gets connected with Wi-FI



```
Serial.println(F("Adafruit MQTT demo"));

// Connect to WiFi access point.
Serial.println(); Serial.println();
Serial.print("Connecting to ");
Serial.println(WLAN_SSID);

WiFi.begin(WLAN_SSID, WLAN_PASS);
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
Serial.println();

Serial.println("WiFi connected");
Serial.println("IP address: "); Serial.println(WiFi.localIP());
```

Basically, our purpose is to publish the sensor data and control the LEDs using an Adafruit server.

To do this we must be aware of two terms publish and subscription

- Setup **MQTT** subscriptions for **led1, led2** i.e. **sw1,sw2. sw1 and sw2** are the feed names that we entered at the very first step of naming the feed.
- **bmp. begin()** is used to start the process.
- Print (**“Could not found”,** if the sensor fail to start the process

```
  mqtt.subscribe(&sw1);
  mqtt.subscribe(&sw2);


  if (!bmp.begin()) {
    Serial.println("BMP180 Sensor not found ! ! !");
    while (1) {}
  }
}
```

The unsigned uint32_t data type works on 32-bit numbers.

To execute the main process write the **void loop()**

- The **readPressure()** function will read the pressure around us
- The **readTemperature ()** function will read the pressure around us
- Store the sensor's pressure value into variable **p**
- Set a delay of **100** ms
- **MQTT_connect** function instructs the library to start connecting to the server. This will ensure that the connection to the MQTT server is alive.

```
p = bmp.readPressure();
q = bmp.readTemperature();
Serial.println(p);
Serial.println(q);
delay(100);

MQTT_connect();
```

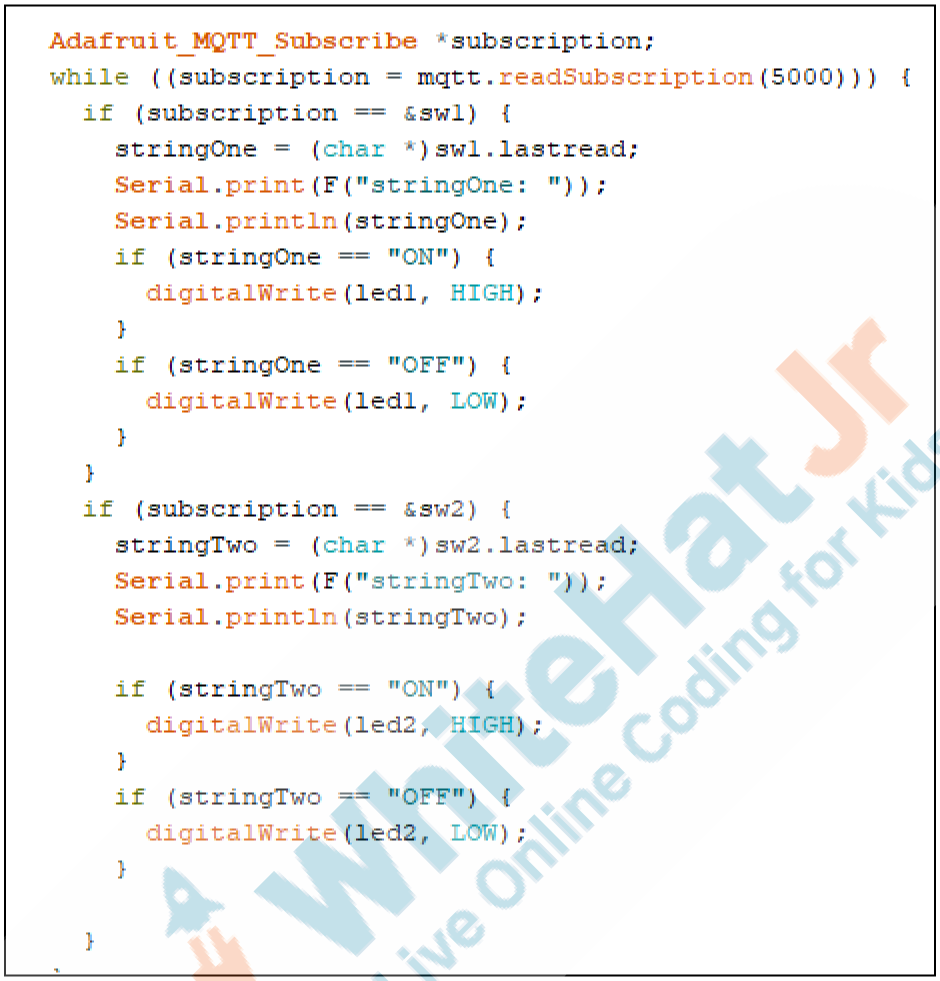
Teacher Stops Screen Share

So now it's your turn.
Please share your screen with me.

We have one more class challenge for you.

Can you solve it? Let's try. I will guide you through it.	
STUDENT-LED ACTIVITY - 20 mins	
● Ask the student to press the ESC key to come back to the panel. ● Guide the student to start Screen Share. ● The teacher gets into Full Screen.	
Student Initiates Screen Share	
ACTIVITY ● Student Activity description (in bullet points).	
Teacher Action	Student Action
So after the connection check, the server waits for subscriptions to come in. Now it's time to send data from server to LED's We know to send data from server to end devices(LEDs) we need to use the **Subscribe** function. Create the **Adafruit_MQTT_Subscribe** object **subscription**, using object subscription we can interact and subscribe. We'll use this to determine which subscription was received. **readSubscription()** will sit and listen for up to 'time' for a message. It will either get a message before the timeout and reply with a pointer to the subscription **or** it will timeout and return **0**. In this case, it will wait up to 5 seconds for a subscription message. Compare the latest feed to the **sw1** feed. If they match, we can read the last message using **lastread**	

If the read times out, the while loop will fail Now it's time to prints out the received data but also compares the data to determine whether the string received is **ON** or **OFF** Since adafruit.io publishes the data as a string, we are using **string. stringOne, stringTwo** will save LED's ON or OFF **digitalWrite()** will change the state of LED using **HIGH** or **LOW** If it is **ON** then make it **HIGH,** otherwise make it **LOW**	



```
Adafruit_MQTT_Subscribe *subscription;
while ((subscription = mqtt.readSubscription(5000))) {
  if (subscription == &sw1) {
    stringOne = (char *)sw1.lastread;
    Serial.print(F("stringOne: "));
    Serial.println(stringOne);
    if (stringOne == "ON") {
      digitalWrite(led1, HIGH);
    }
    if (stringOne == "OFF") {
      digitalWrite(led1, LOW);
    }
  }
  if (subscription == &sw2) {
    stringTwo = (char *)sw2.lastread;
    Serial.print(F("stringTwo: "));
    Serial.println(stringTwo);

    if (stringTwo == "ON") {
      digitalWrite(led2, HIGH);
    }
    if (stringTwo == "OFF") {
      digitalWrite(led2, LOW);
    }

  }
```

Now, it's time to publish the **BMP180** data, We need to publish Pressure and Temperature,

The publication is much easier than subscribing, we need to just call the **publish** function of the feed object.

BMP180 can measure temperature and pressure

Publish the temperature and subtract that temperature from temperature_sens_read and then convert it into Fahrenheit.

Convert raw temperature in Fahrenheit to Celsius degrees

F= 32+1.8C

We can check for success or failure of publication by using the **Serial.print** function.

```
if (! temperature.publish(((temprature_sens_read() - 32 ) / 1.8))) {
} else {
  //Serial.println(F("Teamp OK!"));
}

if (! level.publish(p)) {
  //Serial.println(F(pressure Level Failed"));
} else {
  //Serial.println(F("pressure Level OK!"));
}
```

Function to connect and reconnect as necessary to the MQTT server alive.

It should be called in the loop function and it will take care of connecting.

If gets disconnected then try to connect it again.

ret is a variable which will save temporary return value in datatype **int**

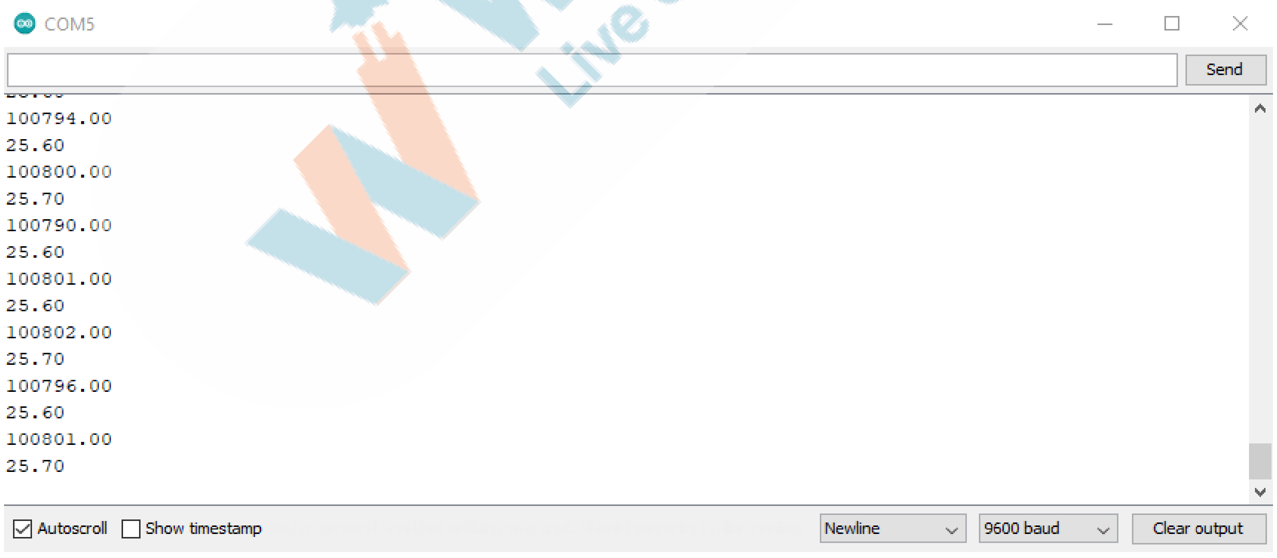
```
void MQTT_connect() {
  int8_t ret;
  if (mqtt.connected()) {
    return;
  }

  uint8_t retries = 3;
  while ((ret = mqtt.connect()) != 0) { // connect will return 0 for connected
    mqtt.disconnect();
    delay(5000);  // wait 5 seconds
    retries--;
    if (retries == 0) {
      while (1);
    }
  }
}
```

Output:
Compile and upload the program to ESP32 board using Arduino IDE

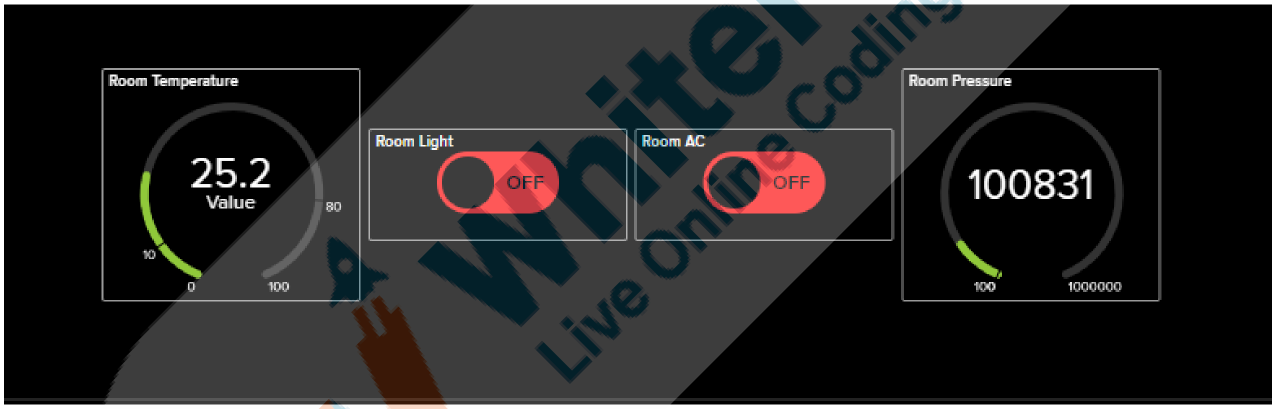
- Verify the program by clicking the Tick option
- Upload the program by clicking the arrow option

Note: If the port is not selected, insert the USB cable in Computer's port and select the port

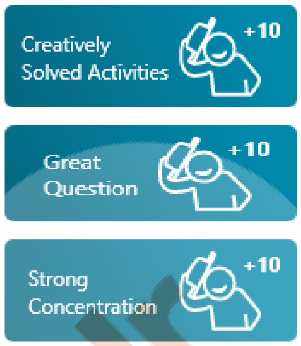
- Go to Tools and select Serial Monitor



```
100794.00
25.60
100800.00
25.70
100790.00
25.60
100801.00
25.60
100802.00
25.70
100796.00
25.60
100801.00
25.70
```

Error Message: If an error message comes like no such file or directory then use the below method to resolve this error Go to **Tools** ● Click on **Manage Libraries** ● Write the component name which needs to install ● Click on Install	
Go to Tools and select **Serial Monitor** ● See the **Pressure** and **Temperature** value ● Open your Adafruit server and check the live values of Pressure and Temperature. Click the Toggle buttons to turn ON and OFF your LED's	
 Room Temperature 25.2 Value 10 0 100 80 Room Light OFF Room AC OFF Room Pressure 100831 100 1000000	
So, Today we made an arrangement where we saw the exact readings of the pressure and temperature around us using a **BMP180** sensor and even we learned how to publish and subscribe data That's fun!	
Teacher Guides Student to Stop Screen Share	
WRAP-UP SESSION - 05 mins	

Teacher Starts Slideshow Slide 14-18	
Activity details **Following are the WRAP-UP session deliverables:** ● Appreciate the student. ● Revise the current class activities. ● Discuss the quizzes.	
WRAP-UP QUIZ Click on In-Class Quiz	
Continue WRAP-UP Session **Slide 19-24**	
Activity Details **Following are the session deliverables:** ● Explain the facts and trivia ● Next class challenge ● Project for the day ● Additional Activity (Optional)	
FEEDBACK ● **Appreciate and compliment the student for trying to learn a difficult concept.** ● **Get to know how they are feeling after the session.** ● **Review and check their understanding.**	
Teacher Action	**Student Action**
You get “hats-off” for your excellent work!	*Make sure you have given at least 2 hats-off during the class for:*

In the next class, we will learn about Relays	 Creatively Solved Activities +10 Great Question +10 Strong Concentration +10
PROJECT OVERVIEW DISCUSSION Refer the document below in Activity Links Sections	
Teacher Clicks	

ACTIVITY LINKS		
Activity Name	**Description**	**Links**
Teacher Activity	Boilerplate Code	https://github.com/procodingclass/PRO-C249-Teacher-Boilerplate
Teacher Activity 1	Reference Code	https://github.com/procodingclass/PRO-C249-Reference-Code
Teacher Reference 1	In-Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/39cf1517-72df-4bea-b146-6a7fba8bde6b.docx
Student Activity 1	Boilerplate Code	https://github.com/procodingclass/PRO-C249-Student-Boilerplate