User Guide

ESP32 IoT Starter Development Kit

Part No: SL0017

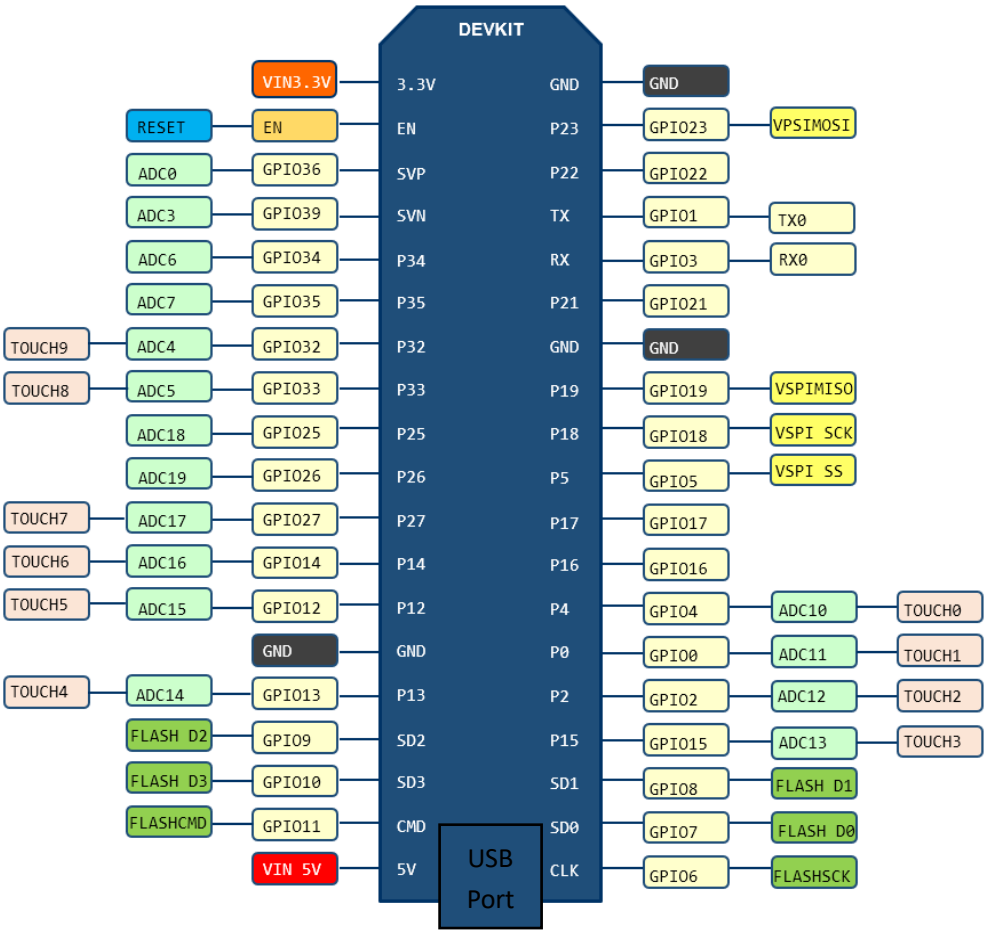
Contents

The ESP32 is loaded with lots of new features and combines Wi-Fi and Bluetooth wireless capabilities and dual-core. The ESP32 peripherals include:

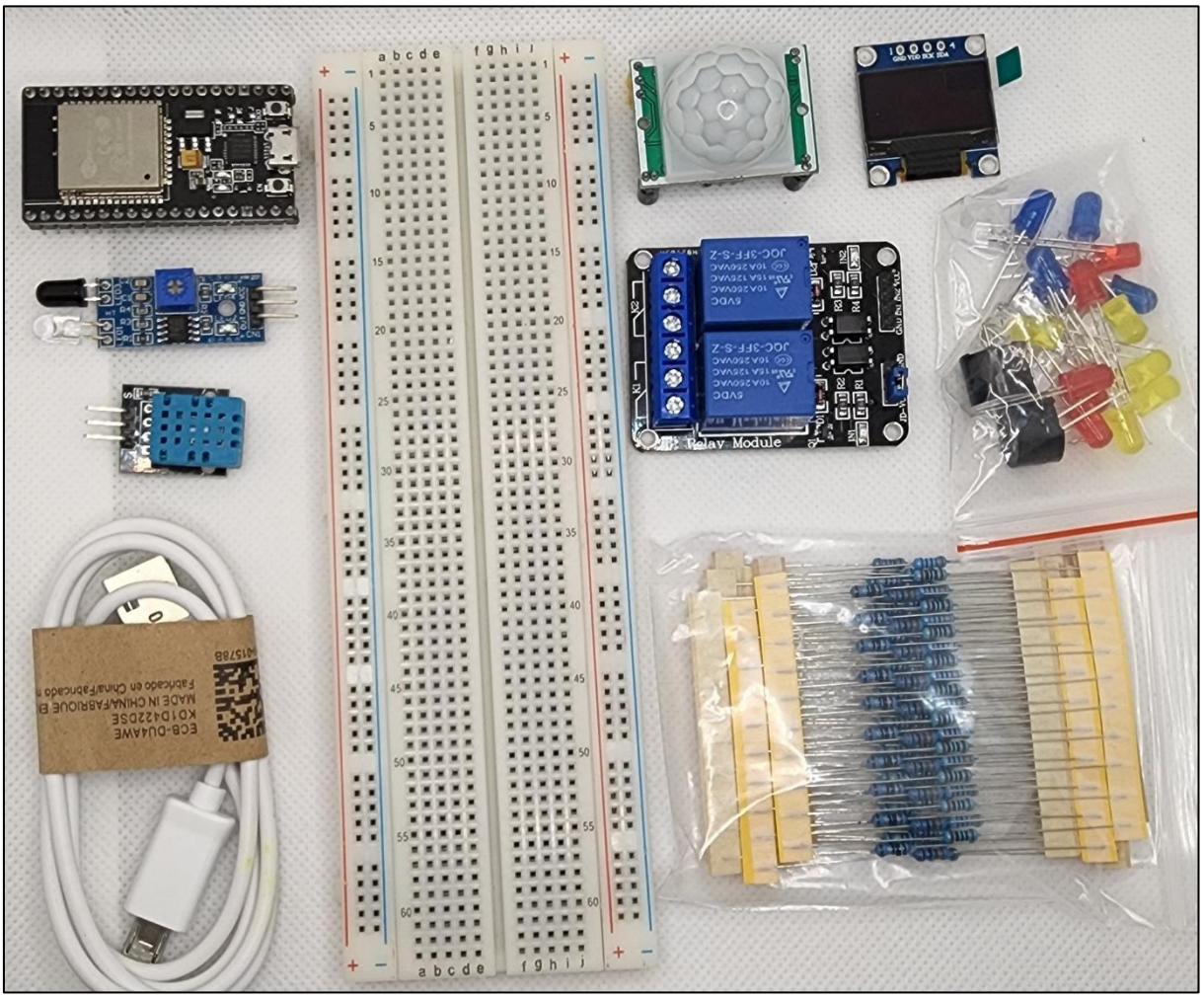
- 18 Analog-to-Digital Converter (ADC) channels
- 3 SPI interfaces
- 3 UART interfaces
- 2 I2C interfaces
- 16 PWM output channels
- 2 Digital-to-Analog Converters (DAC)
- 2 I2S interfaces
- 10 Capacitive sensing GPIOs

The ADC (analog to digital converter) and DAC (digital to analog converter) features are assigned to specific static pins. However, you can decide which pins are UART, I2C, SPI, PWM, etc. – you just need to assign them in the code. This is possible due to the ESP32 chip's multiplexing feature.

Although you can define the pins properties on the software, there are pins assigned by default as shown in the following figure.



Kit Inclusions



The kit comprises of the following components –

1. ESP32 DEVKIT V1 Board
2. DHT11 Temperature and Humidity Sensor
3. IR Sensor
4. PIR Sensor
5. OLED 128x64 Display
6. Two Channel Relay
7. Pack of LEDs
8. Pack of Resistors (different values)
9. 840-point bread board
10. USB Cable (Micro to Type-A)
11. Jumper Cables [Not in picture]
12. Dupoint cables [Not in picture]

There's an add-on for the Arduino IDE that allows you to program the ESP32 using the Arduino IDE and its programming language. In this tutorial we'll show you how to install the ESP32 board in Arduino IDE whether you're using Windows, Mac OS X or Linux.

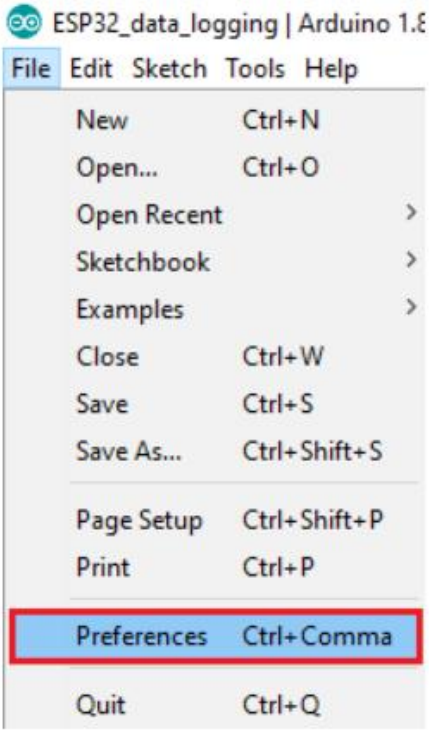
Installing Arduino IDE

Before starting this installation process, please make sure you have installed the latest version of Arduino IDE on your computer. If not, please uninstall and reinstall. Otherwise, it may not work. Install the latest Arduino IDE software from arduino.cc/en/Main/Software , and continue this tutorial.

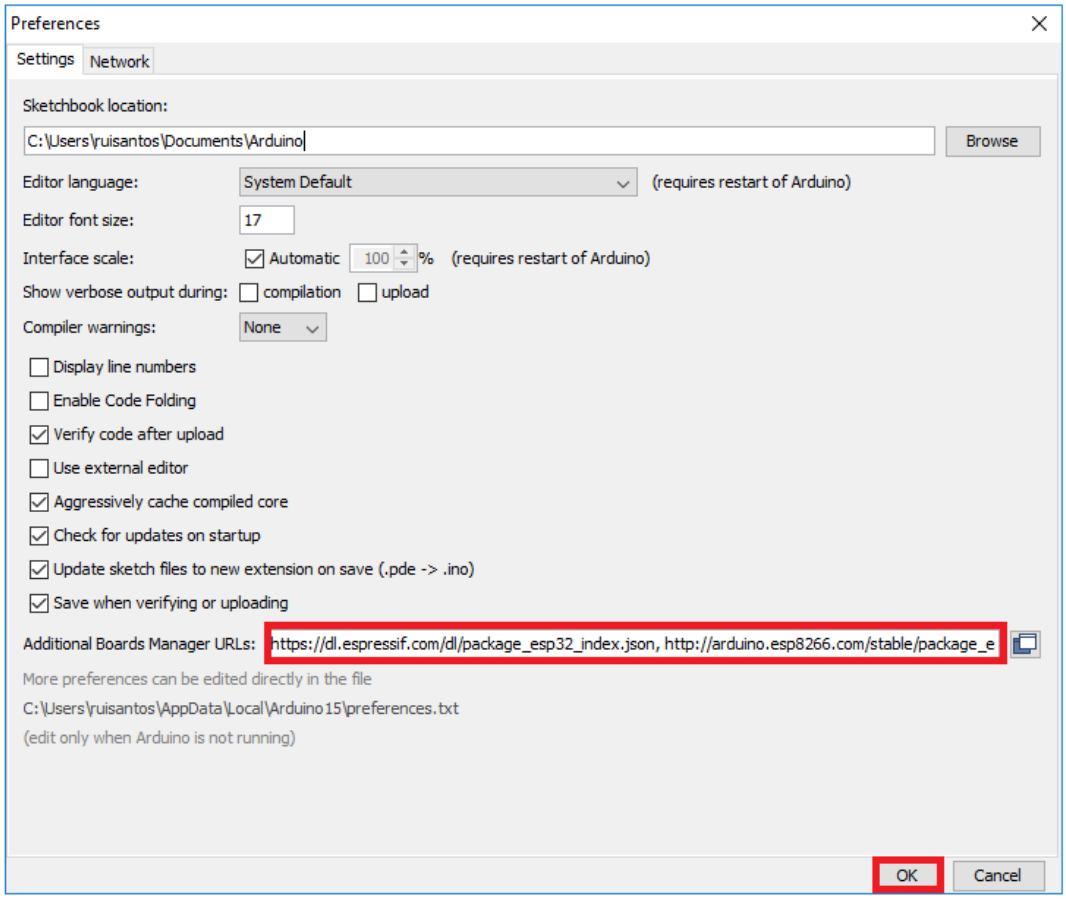
Installing ESP32 Add-on in Arduino IDE

To install the ESP32 board in your Arduino IDE, follow these next instructions:

1. In your Arduino IDE, go to File> Preferences



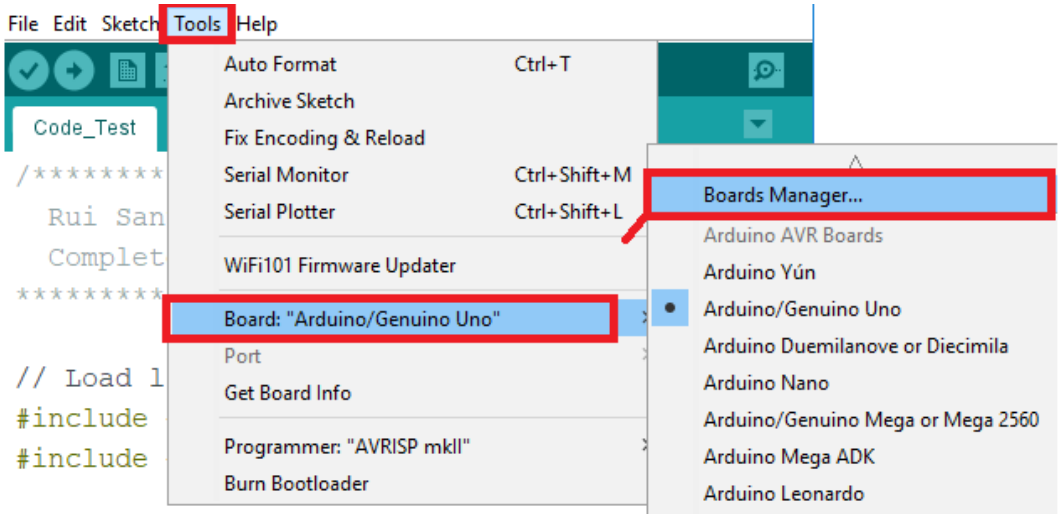
2. Enter **https://dl.espressif.com/dl/package_esp32_index.json** into the "Additional Board Manager URLs" field as shown in the figure below. Then, click the "OK" button:



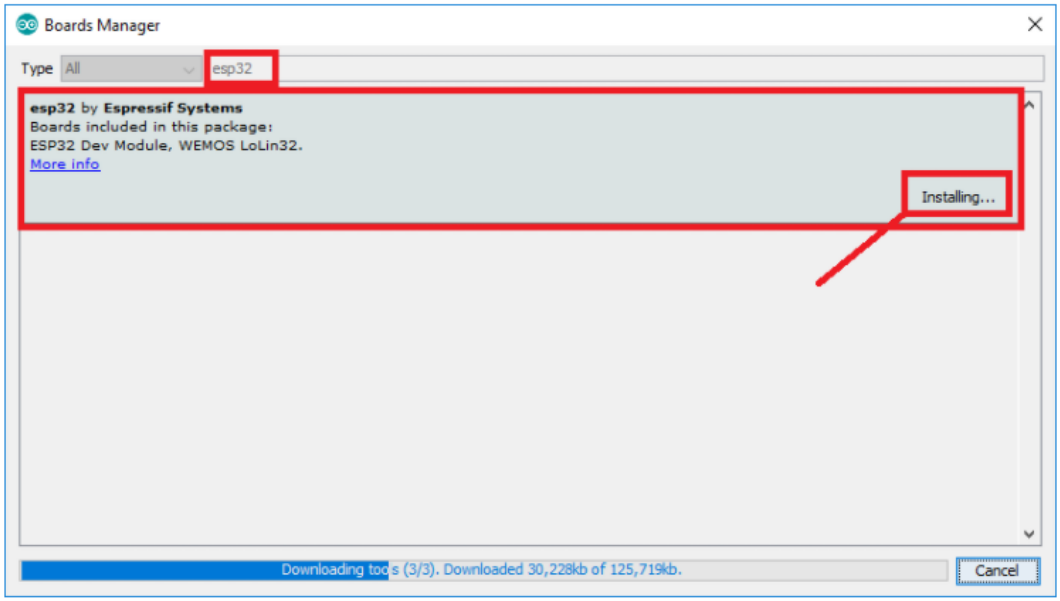
Note: if you already have the ESP8266 boards URL, you can separate the URLs with a comma as follows:

https://dl.espressif.com/dl/package_esp32_index.json,
http://arduino.esp8266.com/stable/package_esp8266com_index.json

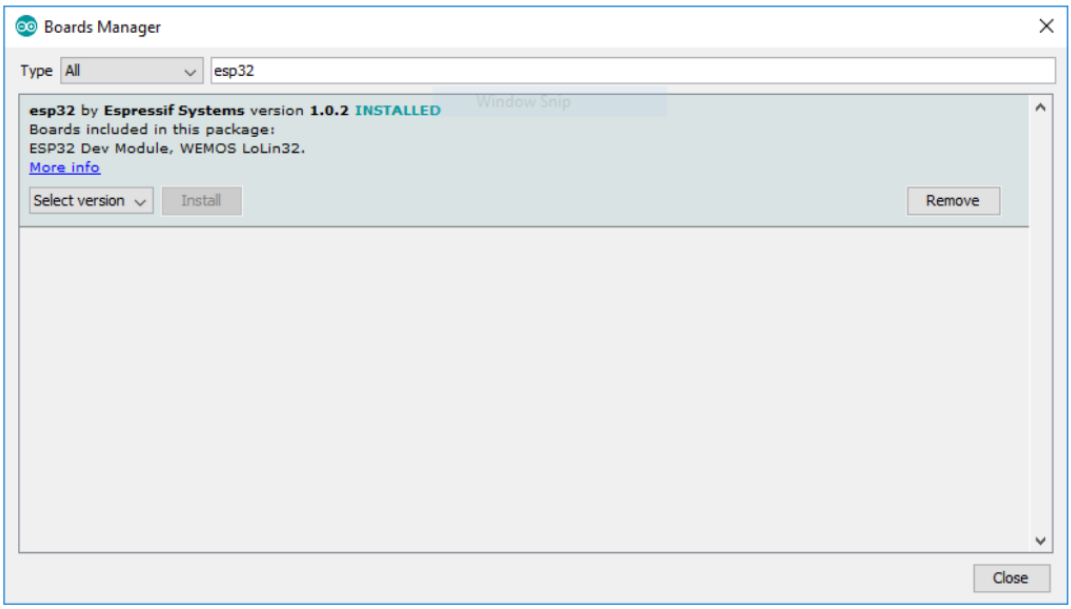
3. Open the Boards Manager. Go to Tools > Board > Boards Manager…



4. Search for ESP32 and press install button for the “ESP32 by Espressif Systems“:



5. It should be installed shortly.

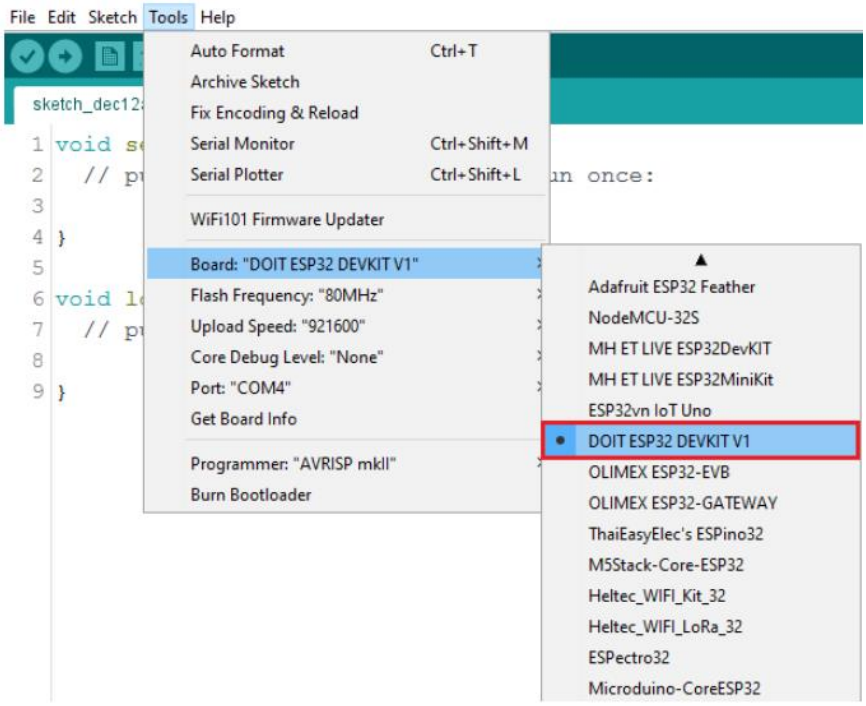


Testing the Installation

Plug the ESP32 board to your computer using the USB Cable included in the kit.

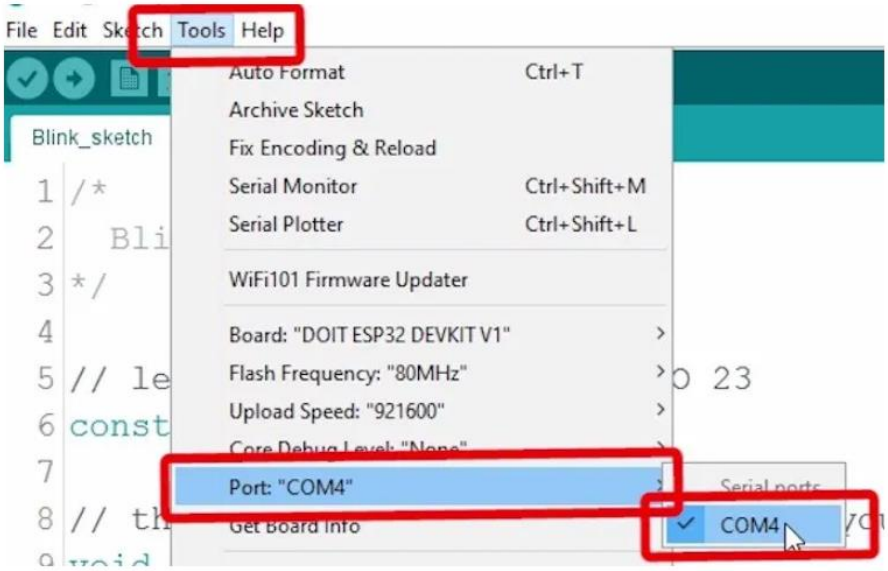
With your Arduino IDE open, follow these steps:

1. Select your Board in Tools > Board menu (in my case it's the DOIT ESP32 DEVKIT V1)



2. Select the Port as below

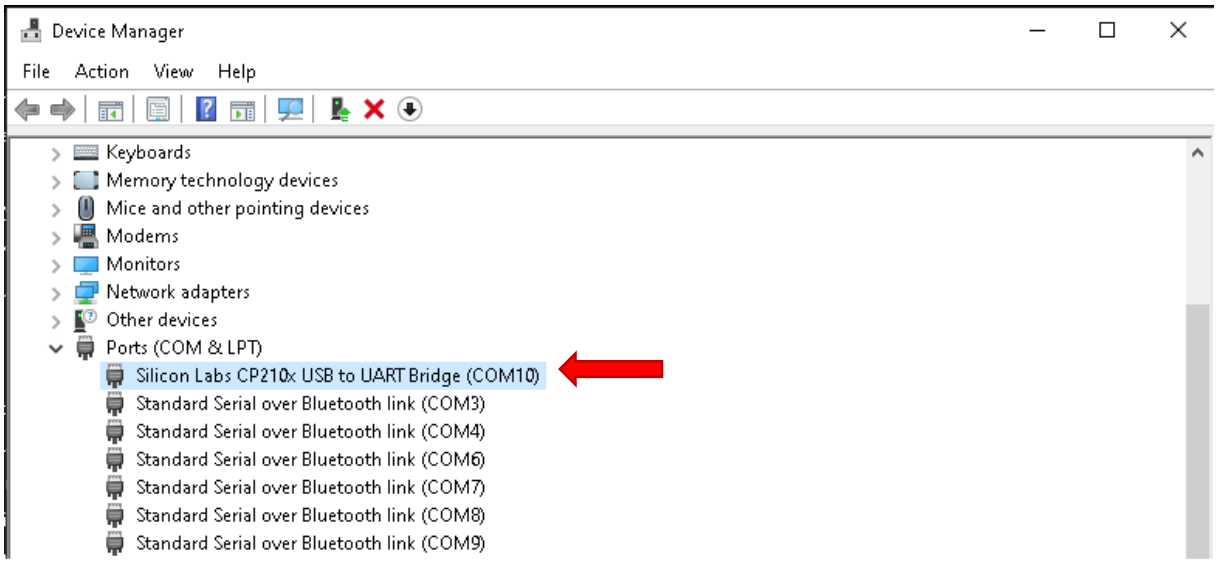
Note: If you don't see the COM Port in your Arduino IDE, you need to install the CP210x USB to UART Bridge VCP Drivers, from the following or similar :
https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers



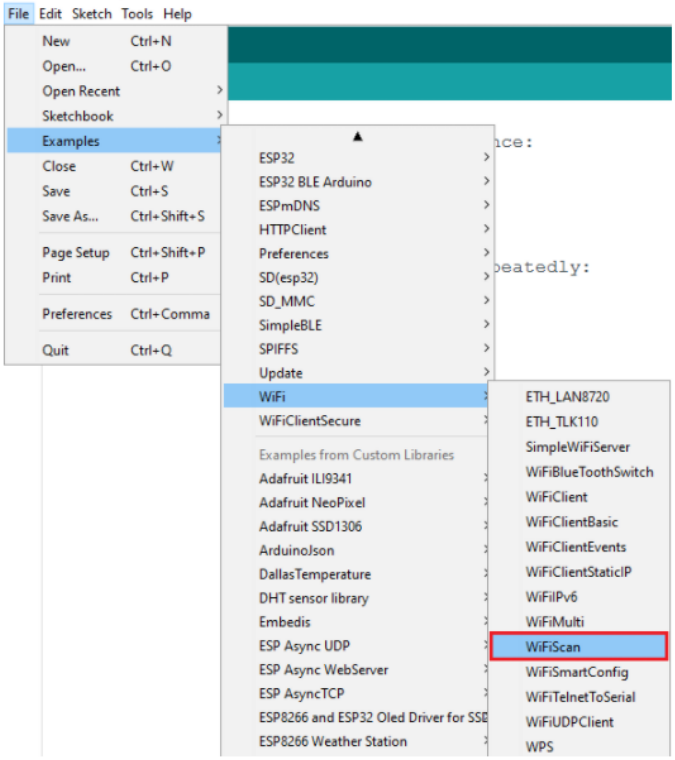
To see the COM port allocated to ESP32 in your Windows PC, please go to 'Device Manager' (you can find that also by typing *device manager* in the start tab, Ctrl + Esc)

and see the COM port.

In this screenshot below, COM10 is allocated, so we need to select that in the Arduino IDE. For iOS-based systems please refer the product user manual for your computer.



3. Open the following example under File > Examples > WiFi (ESP32) > WiFiScan

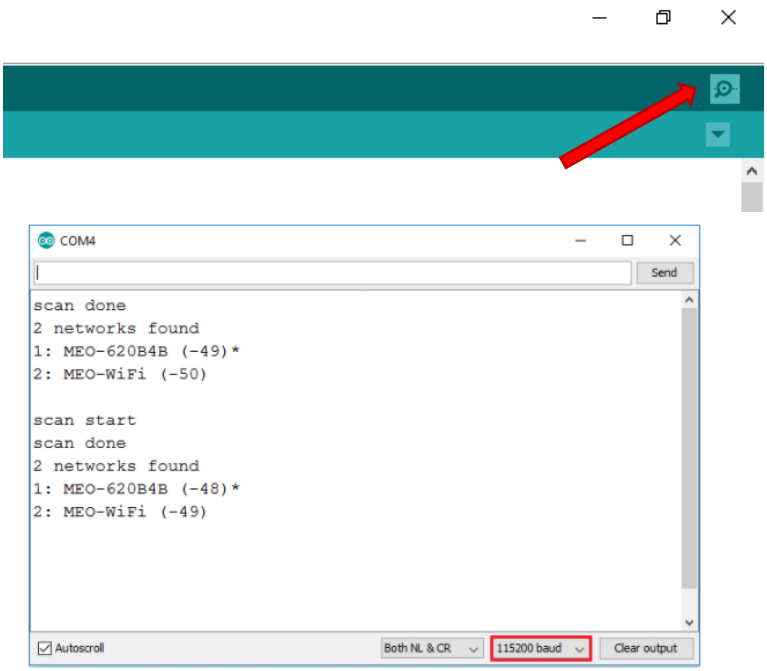


4. A new sketch opens in your Arduino IDE:

5. Press the Upload button in the Arduino IDE. Wait a few seconds while the code compiles and uploads to your board. **Please note - A new code needs to be uploaded each time a new project is undertaken. That is to say, that if the circuit diagram is changed then the code for that particular circuit must also be uploaded.**

If everything went as expected, you should see a “Done uploading.” message.

6. Open the Arduino IDE Serial Monitor at a baud rate of 115200 and Press the ESP32 on-board Enable button and you should see the networks available near your ESP32

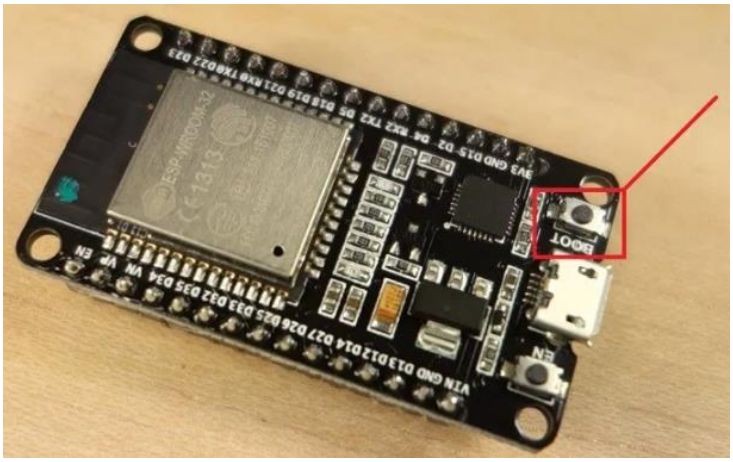


Troubleshooting the installation

If you try to upload a new sketch to your ESP32 and you get this error message “A fatal error occurred: Failed to connect to ESP32: Timed out… Connecting…“. It means that your ESP32 is not in flashing/uploading mode.

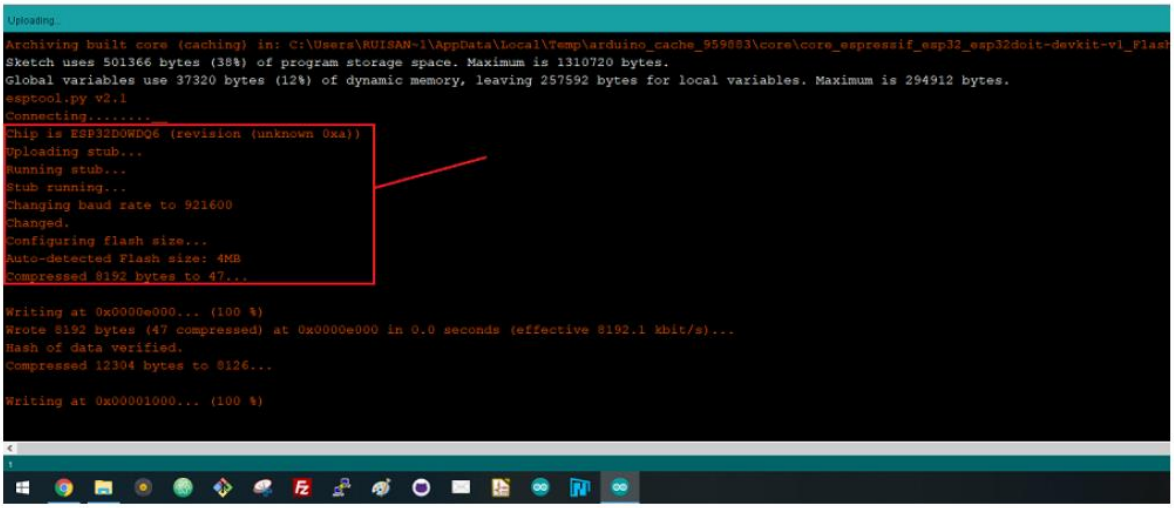
Having the right board name and COM port selected, please follow these steps:

- Hold-down the “**BOOT**” button in your ESP32 board



Press the “Upload” button in the Arduino IDE to upload your sketch: 

After you see the “Connecting….” message in your Arduino IDE, release the finger from the “BOOT” button:



- After that, you should see the “Done uploading” message
- Your ESP32 should have the new sketch running.
- Press the “ENABLE” button to restart the ESP32 and run the new uploaded sketch.
- You’ll also have to repeat that button sequence every time you want to upload a new sketch.

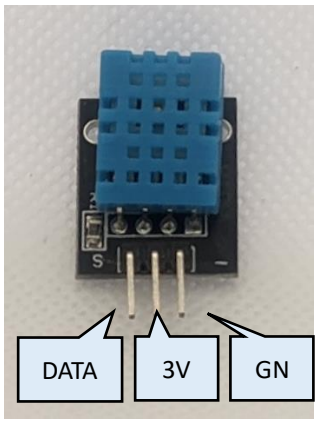
Project 1: DHT11 temperature and humidity sensor module

This tutorial introduces how to use DHT11 temperature and humidity sensor with ESP32 using Arduino IDE. We will quickly introduce these sensors, pinouts, wiring diagrams, and finally the Arduino sketches.

About the DHT 11 Sensor –

Measure both temperature and humidity with this fully digital operated, so no analogue-to-digital calibration is required. Features resistive-type humidity measurement.

- Temperature Range: 0 ºC - 50 ºC +/- 2 ºC
- Humidity Range: 20 – 80% +/- 5%
- Sample Rate: 1Hz



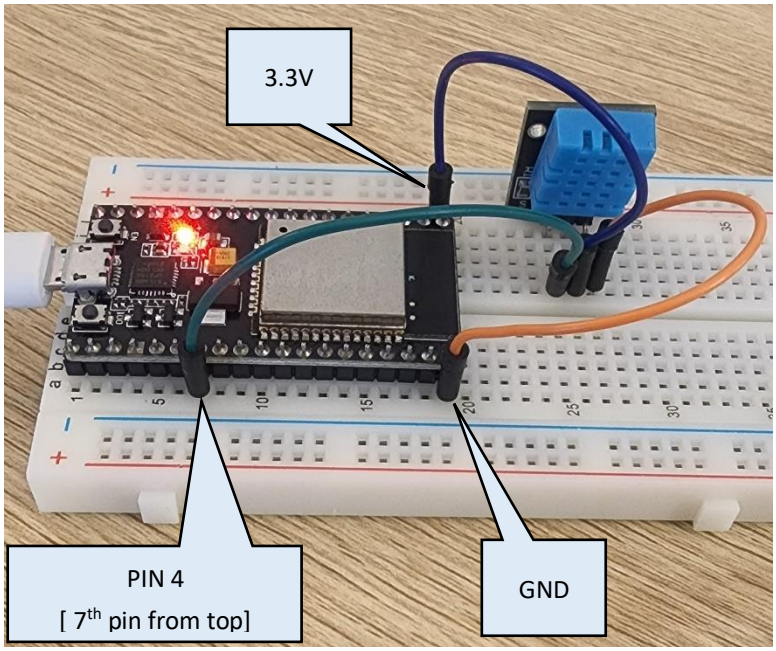
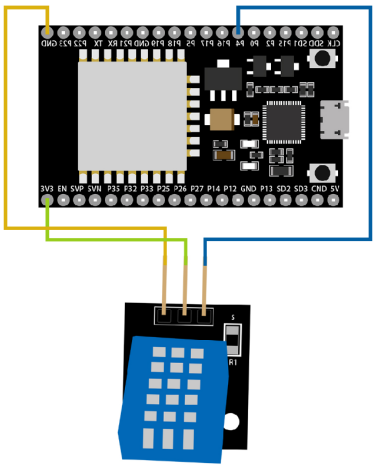


Diagram:

Connect the DHT11 sensor to the ESP32 development board as shown in the figure below.



ESP32 Dev Board	DHT11 Sensor
GND	GND
PIN4(GPIO4)	S (Data)
3.3V	Centre Pin

In this example, we connect the DHT data pin to GPIO 4. However, you can use any other appropriate numeric pin and please change the code accordingly in that case.

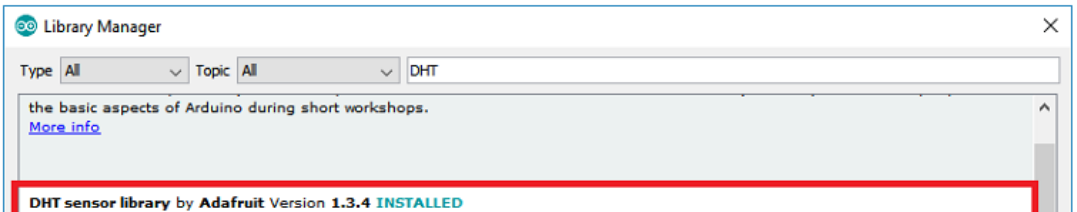
Installing Libraries ：

To read from the DHT sensor, we'll use the DHT library from Adafruit. To use this library you also need to install the Adafruit Unified Sensor library. Follow the next steps to install those libraries.

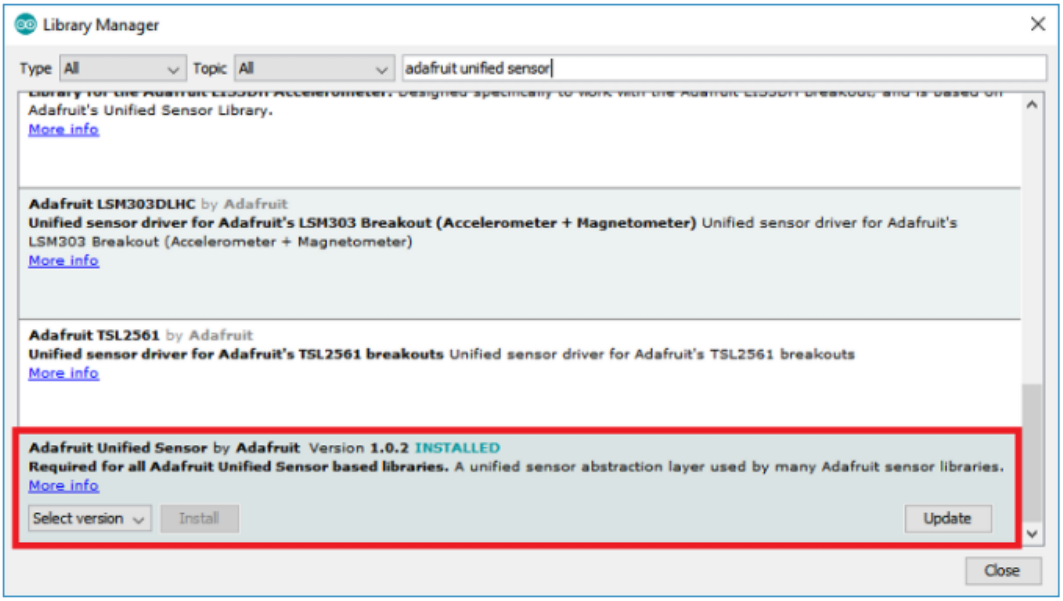
Open your Arduino IDE and go to Sketch > Include Library > Manage Libraries. The Library Manager should open.

Search for "DHT" on the Search box and install the DHT library from Adafruit.

Please note: The version might be different as this user guide was written earlier



After installing the DHT library from Adafruit, type “Adafruit Unified Sensor” in the search box. Scroll all the way down to find the library and install it.



After installing the libraries, restart your Arduino IDE.

The code and how it works:

In this code section, we call the library file DHT.h and define the pin to which the sensor is connected, in this case it is pin 4.

```
#include "DHT.h"

#define DHTPIN 4        // Digital pin connected to the DHT sensor
// DHT must be disconnected during program upload.
```

Create a DHT object called dht on the pin and with the sensor type you’ve specified previously.

```
#define DHTTYPE DHT11     // DHT 11
DHT dht(DHTPIN, DHTTYPE);
```

In the setup (), we initialize the Serial debugging at a baud rate of 9600, and print a message in the serial monitor

```
void setup() {
  Serial.begin(9600);
  Serial.println(F("DHTxx test!"));
  dht.begin();
}
```

The loop() starts with a 2000 ms (2 seconds) delay, because the DHT22 maximum sampling period is 2 seconds. So, we can only get readings every two seconds.

```
void loop() {
  // Wait a few seconds between measurements.
  delay(2000);
```

The temperature and humidity are returned in float format. We create float variables h, t, and f to save the humidity, temperature in Celsius and temperature in Fahrenheit, respectively.
Getting the humidity and temperature is as easy as using the readHumidity() and readTemperature() methods on the dht object

```
  float h = dht.readHumidity();

  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();

  // Read temperature as Fahrenheit (isFahrenheit = true)
  float f = dht.readTemperature(true);
```

There's also an if statement that checks if the sensor returned valid temperature and humidity readings.

```
// Check if any reads failed and exit early (to try again).
if (isnan(h) || isnan(t) || isnan(f)) {
  Serial.println(F("Failed to read from DHT sensor!"));
  return;
}
```

After getting the humidity and temperature, the library has a method that computes the heat index. You can get the heat index both in Celsius and Fahrenheit as shown

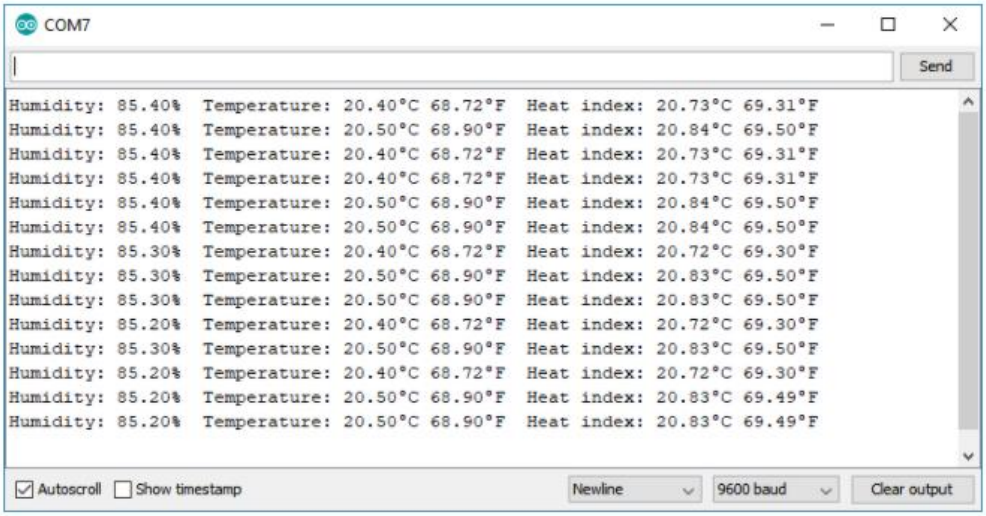
```
// Compute heat index in Fahrenheit (the default)
 float hif = dht.computeHeatIndex(f, h);
 // Compute heat index in Celsius (isFahreheit = false)
 float hic = dht.computeHeatIndex(t, h, false);
```

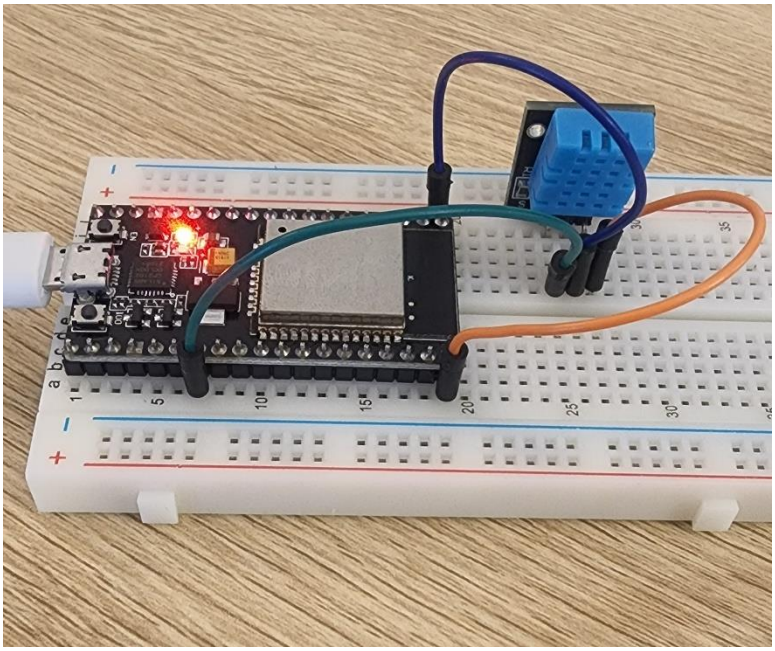
Finally, print all the readings on the Serial Monitor

```
  Serial.print(F("Humidity: "));
  Serial.print(h);
  Serial.print(F("%   Temperature: "));
  Serial.print(t);
  Serial.print(F("°C "));
  Serial.print(f);
  Serial.print(F("°F   Heat index: "));
  Serial.print(hic);
  Serial.print(F("°C "));
  Serial.print(hif);
  Serial.println(F("°F"));
}
```

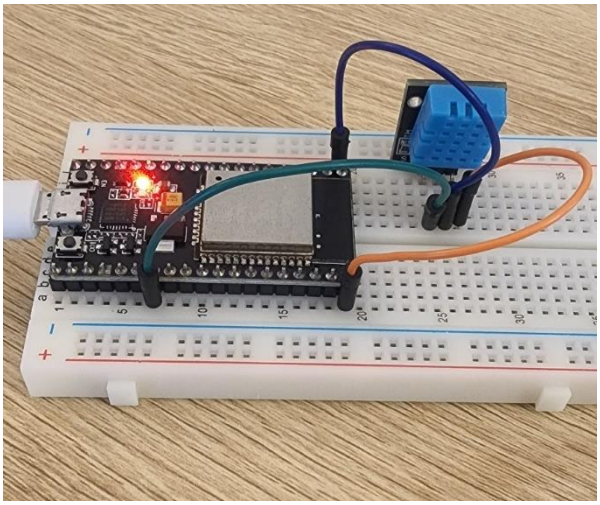
Demonstration

Upload the code to your ESP32 board. Make sure you have the right board and COM port selected in your Arduino IDE settings. After uploading the code, open the Serial Monitor at a baud rate of 9600. You should get the latest temperature and humidity readings in the Serial Monitor every two seconds.





Using the DHT11 temperature and humidity sensor that we used in the previous project, we will now create a webserver with ESP32 using Arduino IDE.



The ESP32 connects to an existing WiFi network (one created by your wireless router) is called Station (STA).

In this mode, the ESP32 gets the IP address from the router, and sets up a webserver that delivers webpages to all connected devices under the existing WiFi network.

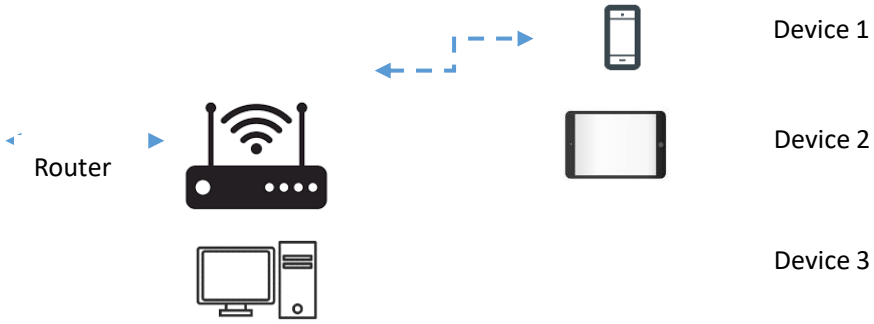
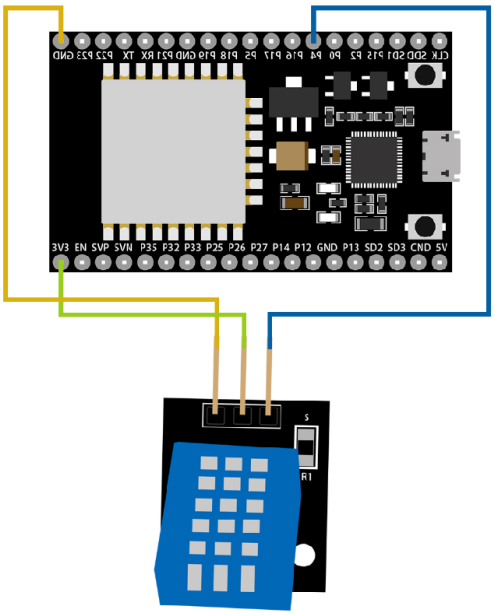


Diagram:

Connect the DHT11 sensor to the ESP32 development board as shown in the figure below, just as the previous project.



Installing Libraries :

Install the libraries for the DHT11 sensor just as explained in the project 1,
please refer page number 13.

The code and how it works:

In this code section, we call the library file DHT.h and define the pin to which the sensor is connected, in this case it is pin 4.

```
#include <WiFi.h>
#include <WebServer.h>
#include "DHT.h"

#define DHTTYPE DHT11      // DHT 11
```

This is the most important step in the code, where the Wi-Fi hotspots SSID (name) and password need to entered exactly, else the ESP32 board would not be able to connect and setup the webserver, therefore.

Please go to your computer's network and connection settings menu to validate the SSID and password, or check the WiFi router for default settings, or check with your internet provider.

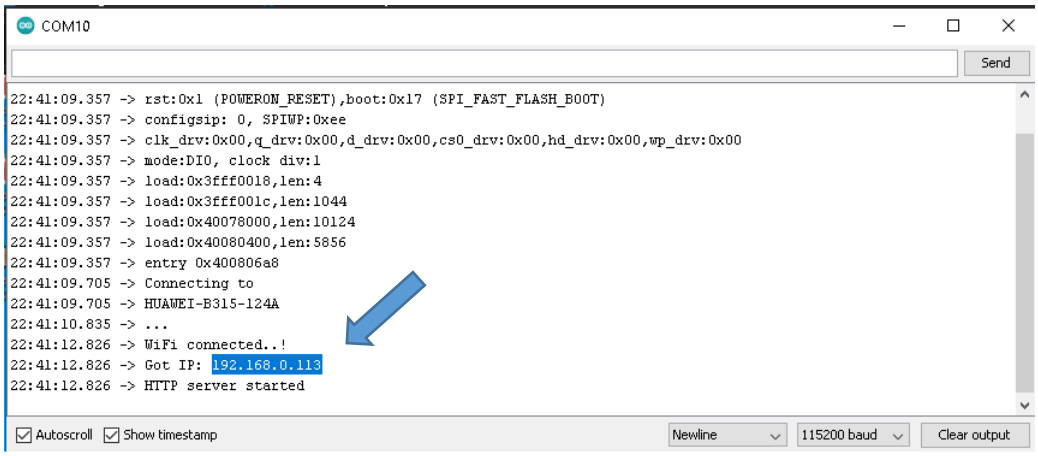
```
/*Put your SSID & Password*/
const char* ssid = "HUAWEI-B315-124A";    // Enter SSID here
const char* password = "3E76L4BR217";    //Enter Password here

WebServer server(80);

// DHT Sensor
uint8_t DHTPin = 4;

// Initialize DHT sensor.
DHT dht(DHTPin, DHTTYPE);

double Temperature;
double Humidity;

void setup() {
  Serial.begin(115200);
  delay(100);

  pinMode(DHTPin, INPUT);

  dht.begin();

  Serial.println("Connecting to ");
  Serial.println(ssid);

  //connect to your local wi-fi network
  WiFi.begin(ssid, password);
```

```
  //check wi-fi is connected to wi-fi network
  while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected..!");
  Serial.print("Got IP: ");    Serial.println(WiFi.localIP());

  server.on("/", handle_OnConnect);
  server.onNotFound(handle_NotFound);

  server.begin();
  Serial.println("HTTP server started");

}
void loop() {

   server.handleClient();

}

void handle_OnConnect() {

  Temperature = dht.readTemperature(); // Gets the values of the temperature
   Humidity = dht.readHumidity(); // Gets the values of the humidity
   server.send(200, "text/html", SendHTML(Temperature,Humidity));
}

void handle_NotFound(){
   server.send(404, "text/plain", "Not found");
}

// Below is the code to display in HTML the values
String SendHTML(float Temperaturestat,float Humiditystat){
   String ptr = "<!DOCTYPE html> <html>\n";
   ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";
   ptr +="<title>ESP32 Weather Report</title>\n";
   ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";
   ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;}\n";
   ptr +="p {font-size: 24px;color: #444444;margin-bottom: 10px;}\n";
   ptr +="</style>\n";
   ptr +="</head>\n";
   ptr +="<body>\n";
   ptr +="<div id=\"webpage\">\n";
   ptr +="<h1>ESP32 Weather Report</h1>\n";
```

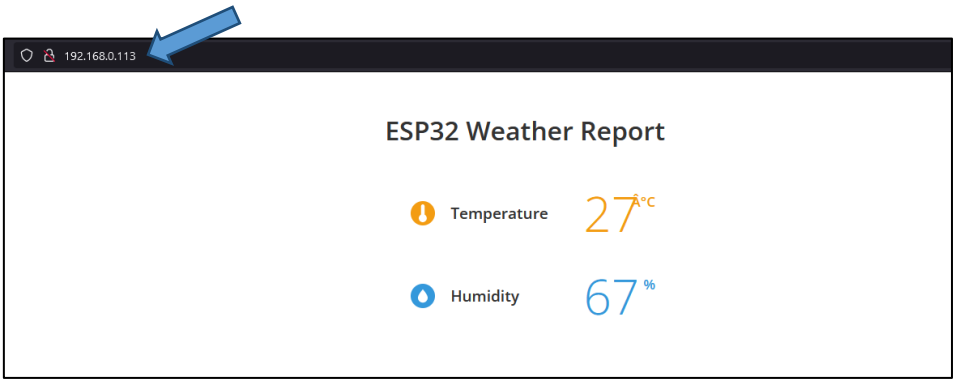
```
  ptr +="<p>Temperature: ";
  ptr +=(int)Temperaturestat;
  ptr +="°C</p>";
  ptr +="<p>Humidity: ";
  ptr +=(int)Humiditystat;
  ptr +="%</p>";
  ptr +="</div>\n";
  ptr +="</body>\n";
  ptr +="</html>\n";
  return ptr;
}
```

Demonstration

Upload the code to your ESP32 board and open the Arduino IDE Serial Monitor.
And there you will see the IP Address assigned to the ESP32 board;

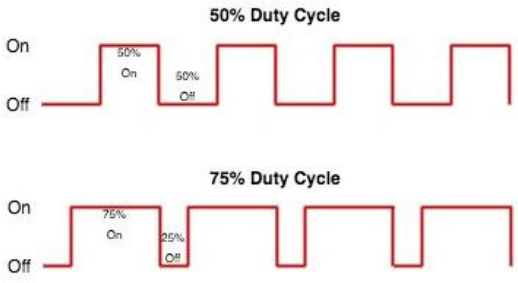


Copy this IP address and paste it directly in your web browser address bar to see the temperature and humidity values. Please note that both the ESP32 board and the device accessing the IP address must be on the same WiFi network.



Project 3: Controlling LED by PWM

What is PWM ?



Pulse width modulation, is a type of a digital signal and is used in basic as well as advanced circuit designs.

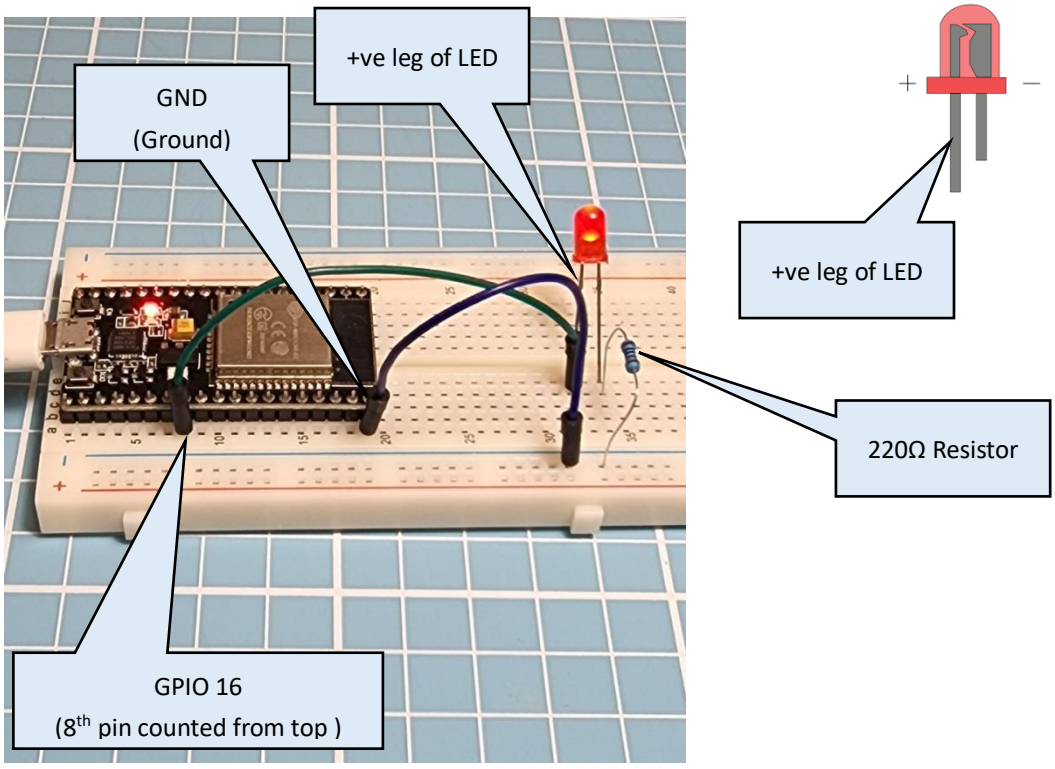
We need to understand the concept of 'Duty cycle' firstly, and simply put – it is the 'on' time, and is measured in percentage. The percentage duty cycle specifically describes the percentage of time a digital signal is on over an interval or period of time. This period is the inverse of the frequency of the waveform.

In the image, the first graph shows a 50% duty cycle. Meaning that it is 'on' 50% of the time, and 50% is 'off' therefore. Similarly, the second graph shows a 75% duty cycle and is 'on'.

The frequency of the square wave does need to be sufficiently high enough when controlling LEDs to get the proper dimming effect.

Diagram:

The LED has different sized legs, and the longer one is the positive (+ve) and should be connected to the IO pin 16 on the ESP32 dev board as shown below, and via a 220 Ω resistor (that is included in the kit, please see page 4 for kit inclusions)

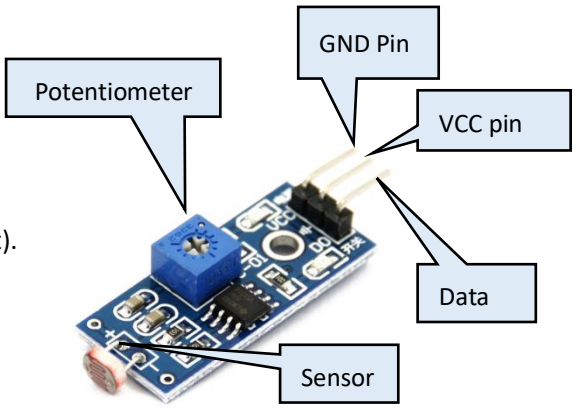


Code :

```
const int ledPin = 16;    // 16 corresponds to GPIO16

// setting PWM properties
const int freq = 5000;
const int ledChannel = 0;
const int resolution = 8;

void setup(){
   // configure LED PWM functionalitites
   ledcSetup(ledChannel, freq, resolution);

   // attach the channel to the GPIO to be controlled
   ledcAttachPin(ledPin, ledChannel);
}

// In this part of the code, the dutycycle of the LED is changed from 0 to 255, since we defined the resolution as 8 bit in the code above. 2^8 = 256.

void loop(){
   // increase the LED brightness
   for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle+=15){
      // changing the LED brightness with PWM
      ledcWrite(ledChannel, dutyCycle);
      delay(15);
   }

   // decrease the LED brightness subsequently in steps of 15.
   for(int dutyCycle = 255; dutyCycle >= 0; dutyCycle-=15){
      // changing the LED brightness with PWM
      ledcWrite(ledChannel, dutyCycle);
      delay(15);
   }
}
```

Project 4: LDR controlled darkness indicator

The Digital LDR Module is used to detect the presence of light / measuring the intensity of light. The output of the module goes high in the presence of light and it becomes low in the absence of light. The sensitivity of signal detection can be adjusted using the potentiometer.

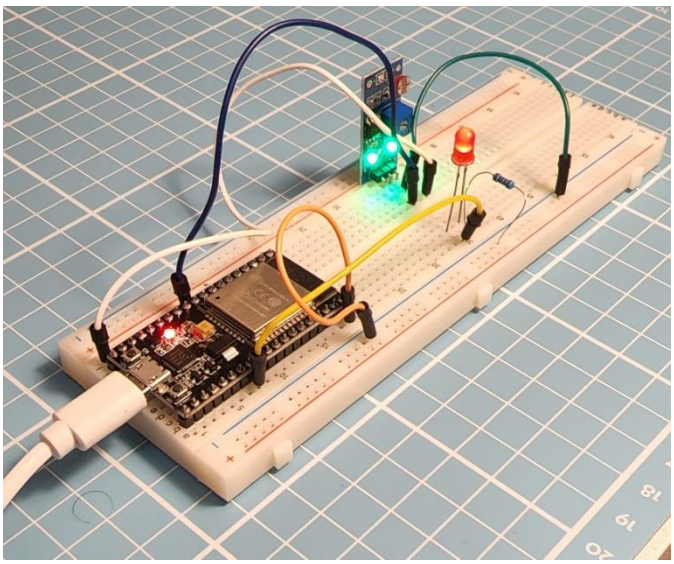


Specifications of LDR sensor -

- LM393 based design.
- Can detect ambient brightness and light intensity.
- Adjustable sensitivity (via blue digital potentiometer adjustment).
- Output Digital – 0V to 5V, Adjustable trigger level from preset.
- LEDs indicating output and power.
- Operating Voltage: 3.3V to 5V DC.

Circuit diagram

There are two circuits connected to the ESP32 board. The first one checks and reports the light intensity back to the ESP32 board; and if the light intensity crosses the defined threshold, then activates the second circuit to light up the LED

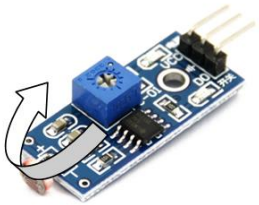


Pin connections		
ESP32	LDR Sensor	LED
GND	GND	-ve
GPIO 26	VCC	
3.3V	Centre Pin	
GPIO 16		+ve (Long)

Code

```
//constants for the pins where sensors are plugged into.
const int sensorPin = 26;
const int ledPin =16;

//Set up some global variables for the light level an initial value.

int lightVal;
// light reading
void setup()
{
// We'll set up the LED pin to be an output.
pinMode(ledPin, OUTPUT);
Serial.begin(115200);
}

void loop()
{Serial.print("Light Value:");
  Serial.print(lightVal);
  Serial.print("\n");
  delay(100);

lightVal = analogRead(sensorPin); // read the current light levels
//if lightVal is less than our initial reading withing a threshold then it is dark.
if(lightVal    < 900)    // Please find the threshold in your room by varying the potentiometer resistance
{
digitalWrite (ledPin, HIGH); // turn on light
}
//otherwise, it is bright
else
{
digitalWrite (ledPin, LOW); // turn off light
}
}
```

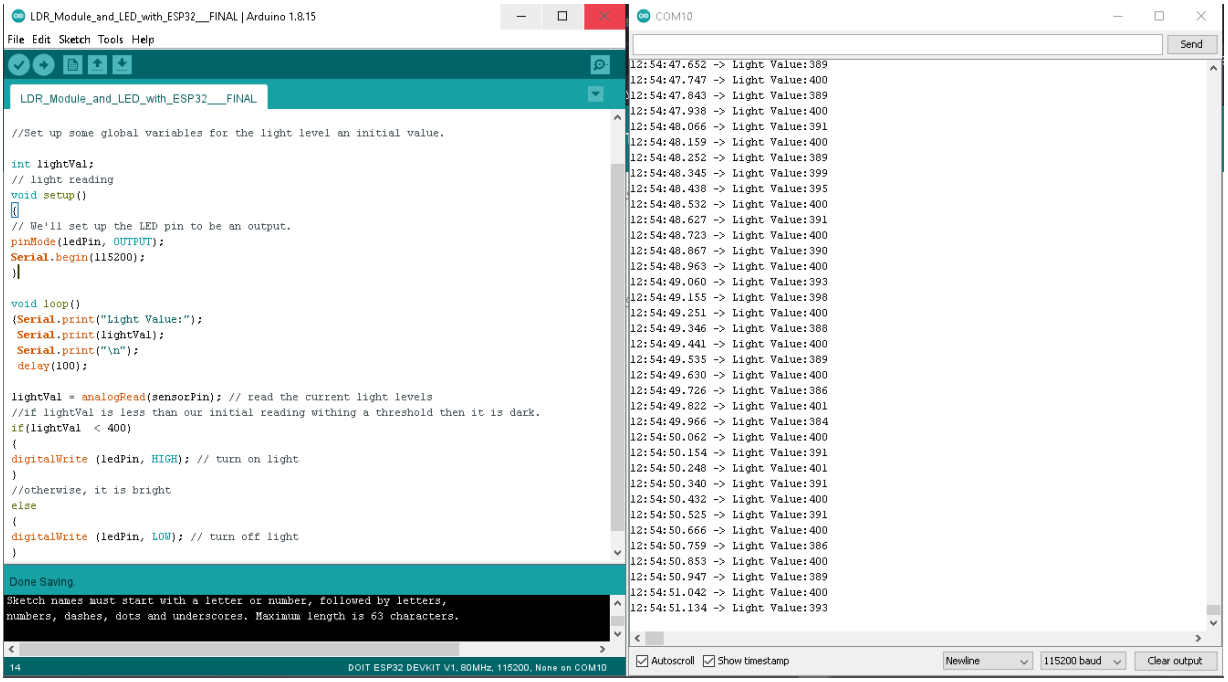
Demonstration



After uploading the code onto the ESP32 board, please adjust the sensitivity of the potentiometer (pot), and open the serial monitor while doing that. Assuming that the pot is on the breadboard, as shown in the image before, and you are facing it. Close the pot, in the anti-clockwise direction. And then slowly start moving the pot in the clockwise direction. Basis the intensity of light on the sensor, you would see a number such as 400 or thereabouts, as in this case. Moving the threshold other way(s), would give a very high value like 4095 and is not what we are after.

The values (400) is the threshold intensity; and please make changes to the code accordingly.

To see the code in action, please vary the light intensity in your room by using additional light source such as a torch, etc. The idea is to vary the light intensity on the sensor.

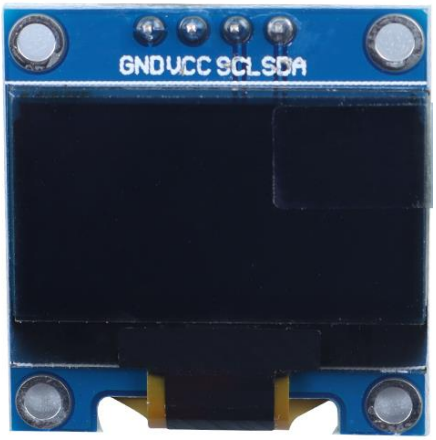


Project 5: Using OLED Display

In this project, we introduce how to use ESP32 to control OLED display. You can modify the code according to your own idea to make OLED display what you want to display. The ESP board will use Arduino IDE for programming.

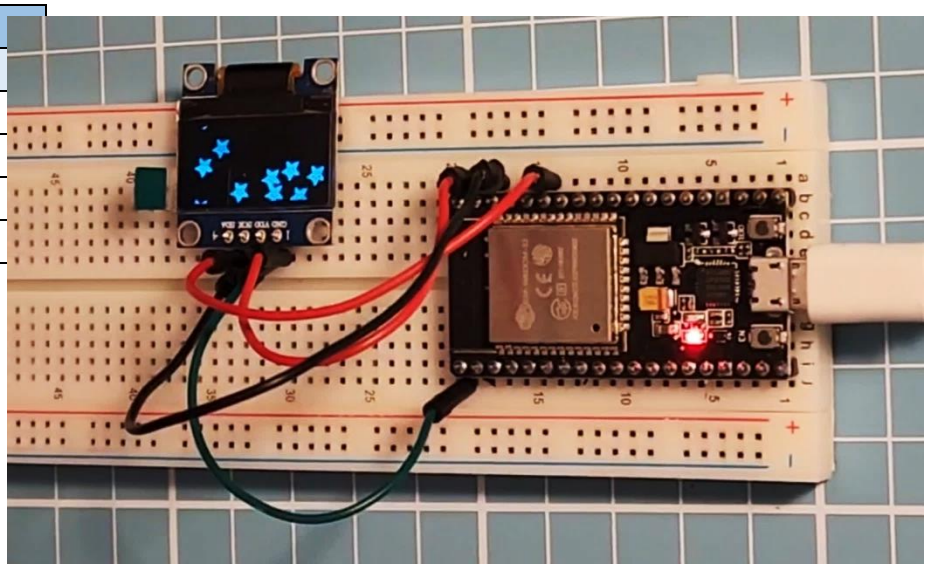
Specifications of OLED Display

- Based pm SSD1306
- 128 x 64 pixels (Width x Height)
- SCL(or SCK) – Signal clock pin
- SDA – Serial data pin
- Voltage : 3.3VDC



Circuit diagram

Pin Connections	
OLED	ESP32
GND	GND
3.3V	VCC
SCK	GPIO22
SDA	GPIO21

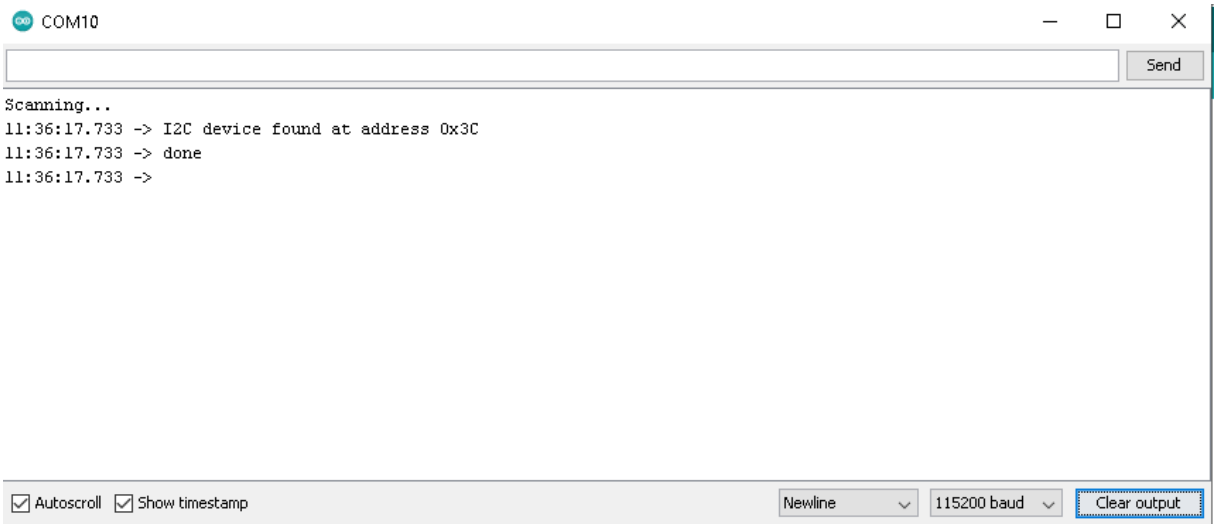


Code

Before running the main code, please run the code for – Screen address finder.

In this code, we are confirming the address of our OLED Screen. And then using this to update the main cod with correct address value.

The code for the screen address finder is as below. Run this code and open the serial monitor, and in my case thr OLED screen is at 0x3C address, as below. The same address must be in all the codes.



Code for confirming the OLED Address:

```
#include <Wire.h>
   void setup() {
   Wire.begin();
   Serial.begin(115200);
   Serial.println("\nI2C Scanner");
}

void loop() {
   byte error, address;
   int nDevices;
   Serial.println("Scanning...");
   nDevices = 0;
   for(address = 1; address < 127; address++ ) {
      Wire.beginTransmission(address);
      error = Wire.endTransmission();
      if (error == 0) {
         Serial.print("I2C device found at address 0x");
         if (address<16) {
            Serial.print("0");
         }
         Serial.println(address,HEX);
         nDevices++;
      }
      else if (error==4) {
         Serial.print("Unknow error at address 0x");
         if (address<16) {
            Serial.print("0");
```

```
            }
            Serial.println(address,HEX);
        }
    }
    if (nDevices == 0) {
        Serial.println("No I2C devices found\n");
    }
    else {
        Serial.println("done\n");
    }
    delay(5000);
}
```

Code for Hello World :

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

void setup() {
    Serial.begin(115200);

    if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)){ // Please cross check the address using screen address finder code.
        Serial.println(F("SSD1306 allocation failed"));
        for(;;);
    }
    delay(2000);
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(0, 10);
    // Display static text
    display.println("Hello, world!");
    display.display();
}
void loop() {    }
```

The address needs to be updated here, from the screen address finder code.

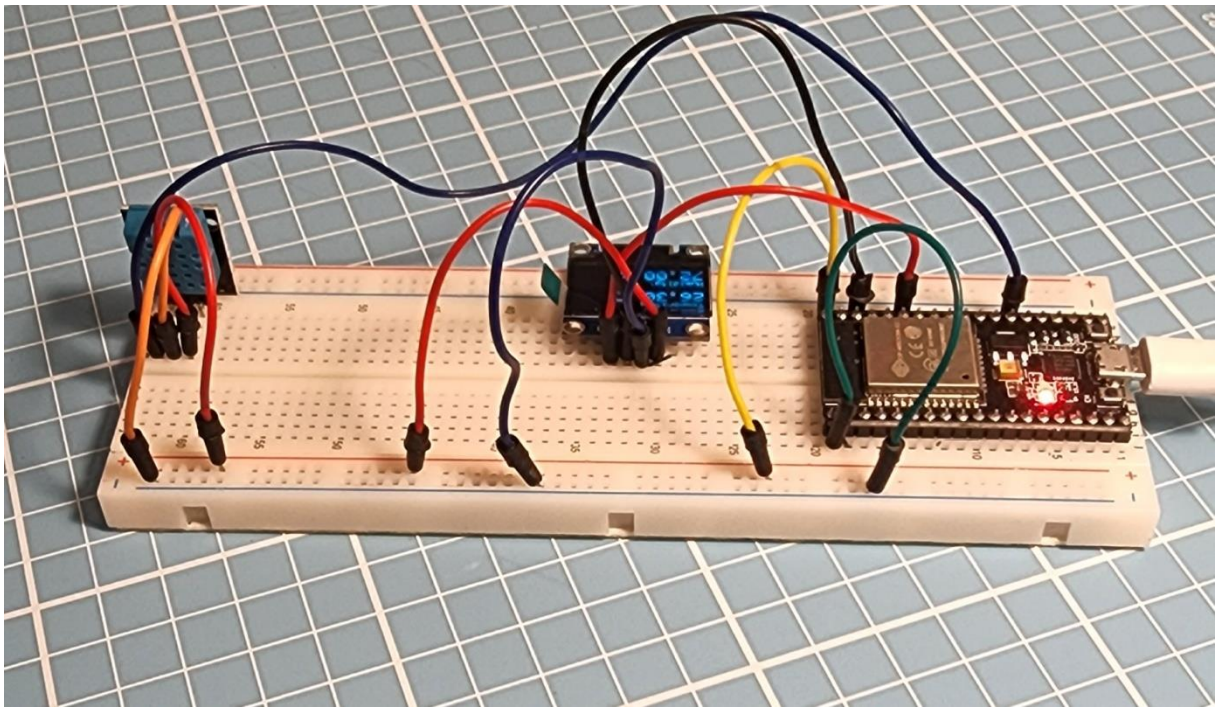
Project 6: Display Temp & Humidity on OLED Display

In this project, we take the OLED display and use it to display the temperature and humidity from the DHT11 sensor.

Circuit diagram

To build this circuit with relative ease, we will use the power rails on the breadboard. One rail is connected to 3.3V and the other is GND.

Pin Connections		
OLED	DHT11	ESP32
GND	GND	GND
3.3V	VCC	VCC
SCK		GPIO22
SDA		GPIO21
	DATA	GPIO4



Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

#define DHTPIN 4          // Digital pin connected to the DHT sensor

#define DHTTYPE DHT11      // DHT 11

DHT dht(DHTPIN, DHTTYPE);

void setup() {
   Serial.begin(115200);

   dht.begin();

   if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
      Serial.println(F("SSD1306 allocation failed"));
      for(;;);
   }
   delay(2000);
   display.clearDisplay();
   display.setTextColor(WHITE);
}

void loop() {
   delay(2500);

   //read temperature and humidity
   float h = dht.readHumidity();

   // Read temperature as Celsius (the default)
   float t = dht.readTemperature();

   // Check if any reads failed and exit early (to try again).
   if (isnan(h) || isnan(t) ){
      Serial.println(F("Failed to read from DHT sensor!"));
      return;
   }
```

```
  // clear display
  display.clearDisplay();

  // display temperature
  display.setTextSize(1);
  display.setCursor(0,0);
  display.print("Temperature: ");
  display.setTextSize(2);
  display.setCursor(0,10);
  display.print(t);
  display.print(" ");
  display.setTextSize(1);
  display.cp437(true);
  display.write(167);
  display.setTextSize(2);
  display.print("C");

  // display humidity
  display.setTextSize(1);
  display.setCursor(0, 35);
  display.print("Humidity: ");
  display.setTextSize(2);
  display.setCursor(0, 45);
  display.print(h);
  display.print(" %");

  display.display();
}
```

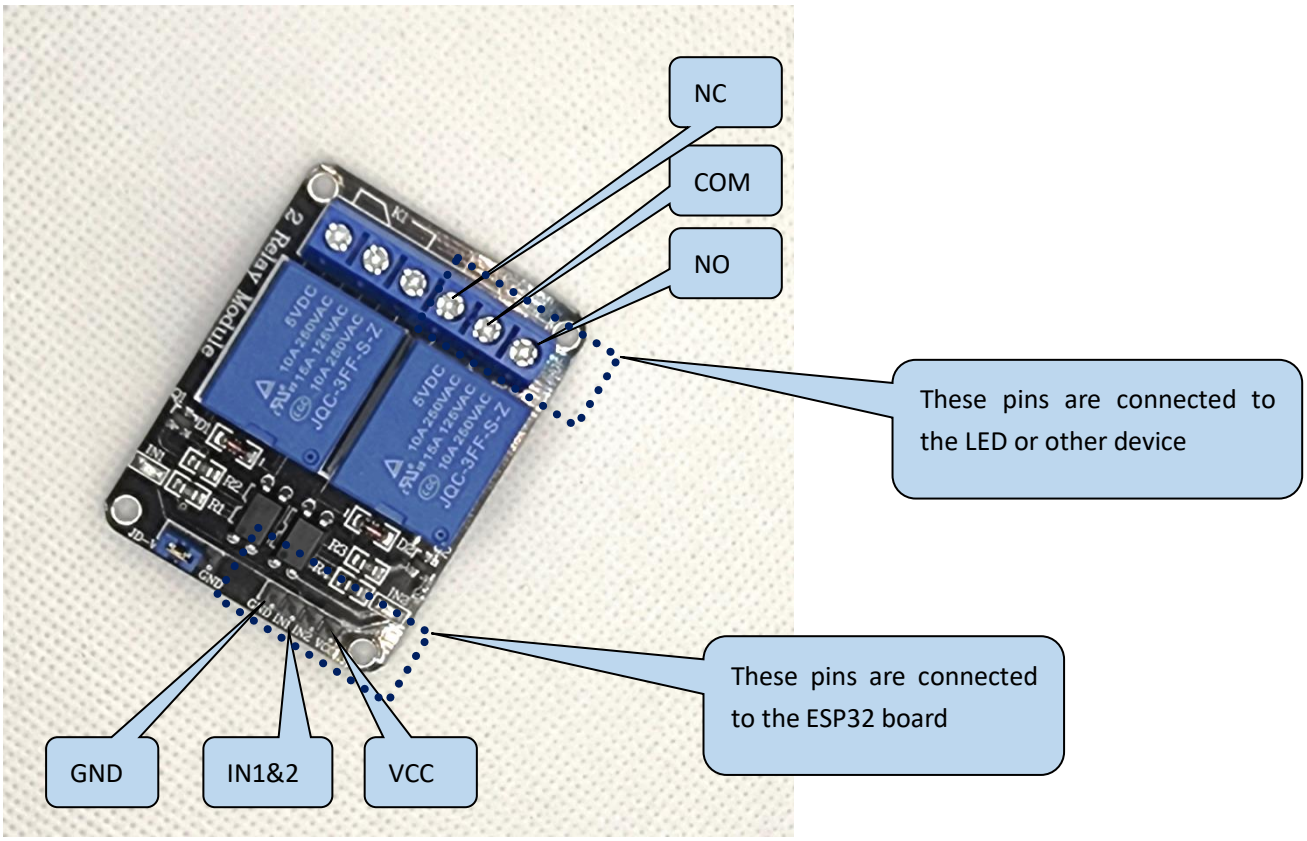
About Relays

A relay is an electrically operated switch and like any other switch, it that can be turned on or off, letting the current go through or not. It can be controlled with low voltages, like the 3.3V provided by the ESP32 GPIOs and allows us to control high voltages like 12V, 24V or mains voltage

There are different relay modules with a different number of channels. You can find relay modules with one, two, four, eight and even sixteen channels. The number of channels determines the number of outputs we'll be able to control.

There are relay modules whose electromagnet can be powered by 5V and with 3.3V. Both can be used with the ESP32 – you can either use the VIN pin (that provides 5V) or the 3.3V pin.

Additionally, some come with built-in optocoupler that add an extra "layer" of protection, optically isolating the ESP32 from the relay circuit.



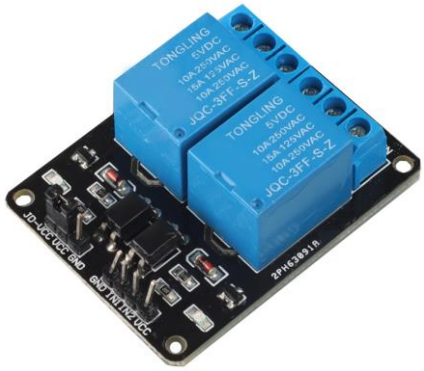
Mains Voltage Connections

The relay module shown in the previous photo has two connectors, each with three sockets: common (COM),

Normally Closed (NC), and Normally Open (NO).

- COM: connect the current you want to control (mains voltage).
- NC (Normally Closed): the normally closed configuration is used when you want the relay to be closed by default. The NC are COM pins are connected, meaning the current is flowing unless you send a signal from the ESP32 to the relay module to open the circuit and stop the current flow.
- NO (Normally Open): the normally open configuration works the other way around: there is no connection between the NO and COM pins, so the circuit is broken unless you send a signal from the ESP32 to close the circuit.

Control Pins



The low-voltage side has a set of four pins and a set of three pins. The first set consists of VCC and GND to power up the module, and input 1 (IN1) and input 2 (IN2) to control the bottom and top relays, respectively.

If your relay module only has one channel, you'll have just one IN pin. If you have four channels, you'll have four IN pins, and so on.

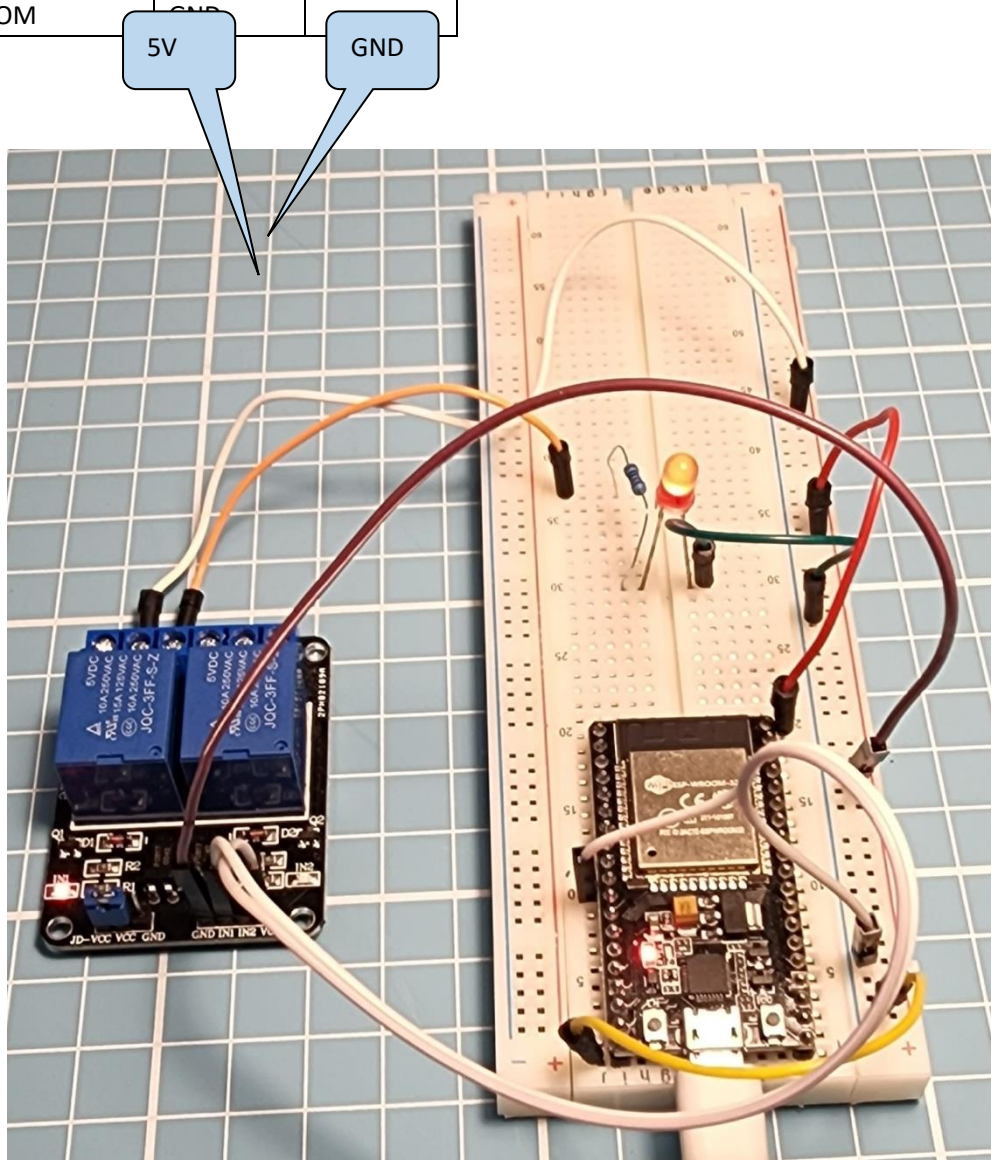
The signal you send to the IN pins, determines whether the relay is active or not. The relay is triggered when the input goes below about 2V. This means that you'll have the following scenarios:

- Normally Closed configuration (NC):
- HIGH signal – current is flowing
- LOW signal – current is not flowing
- Normally Open configuration (NO):
- HIGH signal – current is not flowing
- LOW signal – current in flowing
- You should use a normally closed configuration when the current should be flowing most of the times, and you only want to stop it occasionally.
- Use a normally open configuration when you want the current to flow .

Circuit diagram and connections

To build this circuit with relative ease, we will use the power rails on the breadboard. One rail is connected to 3.3V and the other is GND.

Pin Connections		
Relay	ESP32	LED
GND	GND	
IN1	GPIO26	
VCC	5V	
	5V	+ve
NO		-ve
COM	GND	



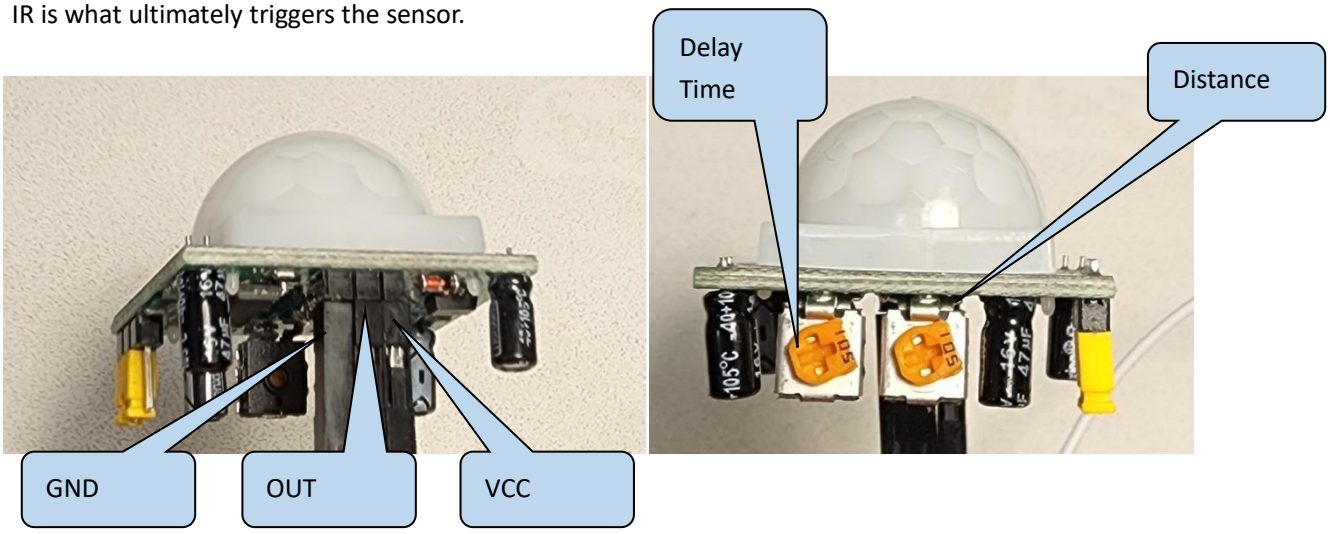
Code

```
const int relay = 26;

void setup() {

Serial.begin(115200);

pinMode(relay, OUTPUT);}

void loop() {
// Normally Open configuration, send LOW signal to let current flow
// (if you're usong Normally Closed configuration send HIGH signal)

digitalWrite(relay, LOW);
Serial.println("Current Flowing");
delay(5000);
// Normally Open configuration, send HIGH signal stop current flow
// (if you're usong Normally Closed configuration send LOW signal)
digitalWrite(relay, HIGH);
Serial.println("Current not Flowing");
delay(5000);
}
```

About PIR Sensor

Passive infrared sensor (PIR) - is a passive motion sensor that means it can only detect something around it and it cannot transmit anything. Whenever there is a motion around the sensor, it will detect the heat of the human body and produces a high output logic 1 at the output of the sensor. has the receiver element divided into two sections. As an object moves in front of one part of the field, it creates a change in the amount of infra-red radiation entering that part of the sensor. A 'warm' object creates an increase in infrared, while a 'cold' object creates a shadow in the infra-red light already entering the sensor from its surroundings, for instance as a vehicle enters a driveway.

The difference between the amount of infra-red light entering one section of the element and the 'default' level of warmth sensed by the other is calculated by an onboard processor. And this difference in incoming IR is what ultimately triggers the sensor.



Operating Voltage : 3.3V
Level output voltage: High 3V / Low 0V
Pin outs of the PIR sensor are as above

The distance and delay time are adjustable potentiometers and their values can be changed using a Phillips screwdriver. Adjust the potentiometers so that you can get the sensitivity right for where you want to use it, This could take some trial and error to find the sweet spot.

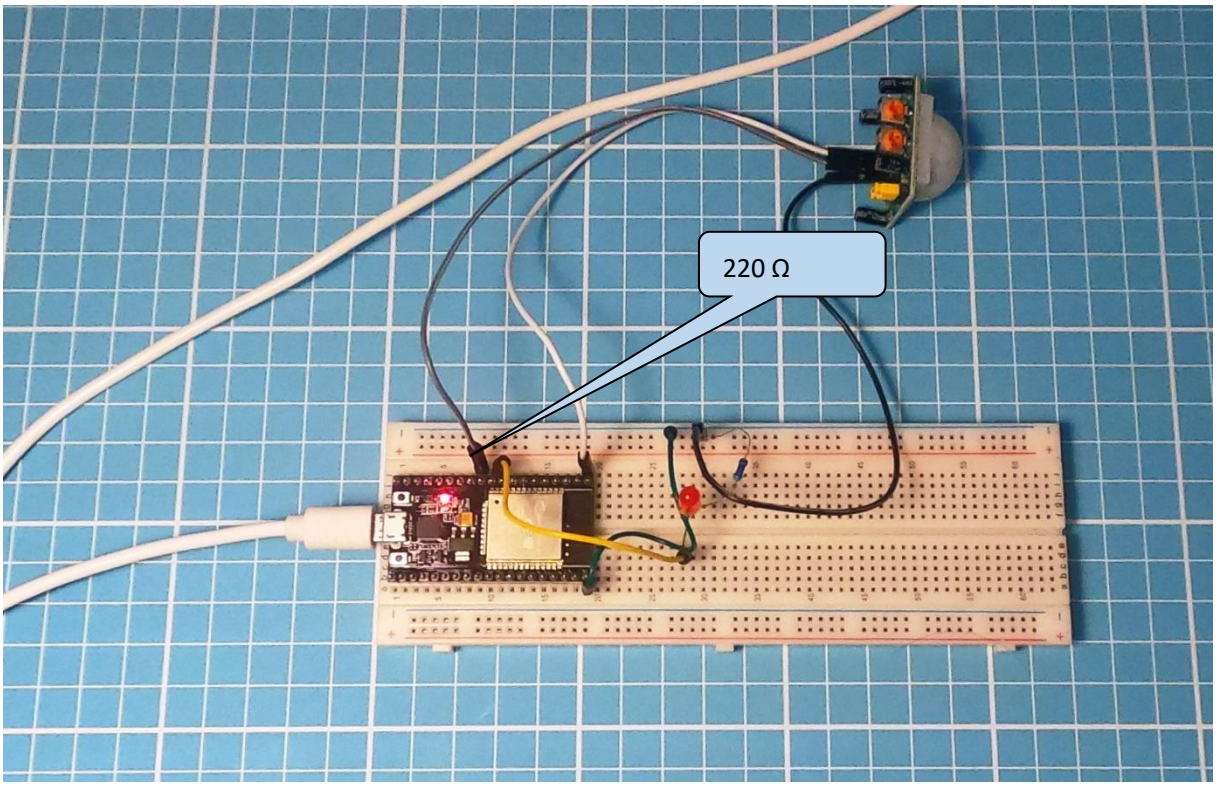
If you are having issues with it always detecting motion, or not detecting motion, try a simple sketch such as in the code -

```
void loop(){       digitalWrite(led_green, digitalRead(pir_sense));       delay(100);}
```

Circuit diagram & Connections

The table below shows the pin connection between the circuit components. Please remember to use the 220Ω for connecting the LED, to limit the current flow and prevent it's burning out.

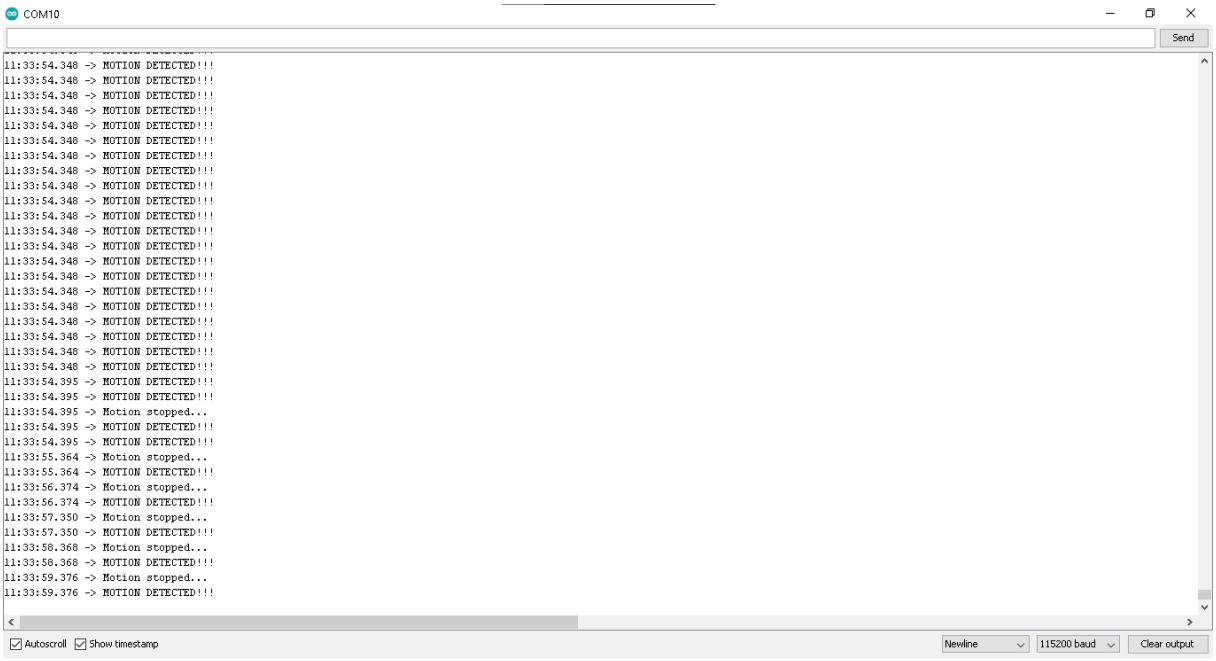
Pin Connections		
ESP32	PIR	LED
GND	GND	-ve (Shorter leg)
3.3V	VCC	
GPIO26		+ve (Longer leg)
GPIO27	OUT	



Code

```
#define timeSeconds 10

// Set GPIOs for LED and PIR Motion Sensor
const int led = 26;
const int motionSensor = 27;

// Timer: Auxiliary variables
unsigned long now = millis();
unsigned long lastTrigger = 0;
boolean startTimer = false;

// Checks if motion was detected, sets LED HIGH and starts a timer
void IRAM_ATTR detectsMovement() {
  Serial.println("MOTION DETECTED!!!");
  digitalWrite(led, HIGH);
  startTimer = true;
  lastTrigger = millis();
}

void setup() {
  // Serial port for debugging purposes
  Serial.begin(115200);

  // PIR Motion Sensor mode INPUT_PULLUP
  pinMode(motionSensor, INPUT_PULLUP);
  // Set motionSensor pin as interrupt, assign interrupt function and set RISING mode
  attachInterrupt(digitalPinToInterrupt(motionSensor), detectsMovement, RISING);

  // Set LED to LOW
  pinMode(led, OUTPUT);
  digitalWrite(led, LOW);
}

void loop() {
  // Current time
  now = millis();
  // Turn off the LED after the number of seconds defined in the timeSeconds variable
  if(startTimer && (now - lastTrigger > (timeSeconds*1000))) {
    Serial.println("Motion stopped...");
    digitalWrite(led, LOW);
    startTimer = false;
  }
}
```

Open the serial monitor to see the code in action. As soon as object moves across the PIR scanned zone, the motion is detected.


COM10
Send
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.348 -> MOTION DETECTED!!!
11:33:54.395 -> MOTION DETECTED!!!
11:33:54.395 -> MOTION DETECTED!!!
11:33:54.395 -> Motion stopped...
11:33:54.395 -> MOTION DETECTED!!!
11:33:54.395 -> MOTION DETECTED!!!
11:33:55.364 -> Motion stopped...
11:33:55.364 -> MOTION DETECTED!!!
11:33:56.374 -> Motion stopped...
11:33:56.374 -> MOTION DETECTED!!!
11:33:57.350 -> Motion stopped...
11:33:57.350 -> MOTION DETECTED!!!
11:33:58.368 -> Motion stopped...
11:33:58.368 -> MOTION DETECTED!!!
11:33:59.376 -> Motion stopped...
11:33:59.376 -> MOTION DETECTED!!!
Autoscroll
Show timestamp
Newline
115200 baud
Clear output

Our aim is to design an IoT motion detection project that does three things whenever motion is detected. This includes turning the LED ON which is connected with ESP32 and PIR sensor, updating the web page with motion detection at exact time and date, and lastly sending an email notification alerting that motion has been detected at the occurred time.

Circuit diagram & Connections

The circuit and connections would be as per the previous project, i.e. project 8

What is IFTTT

IFTTT means 'If this, then that.' It is an open-source service that gives the user the freedom to program a response to an event according to their likes.

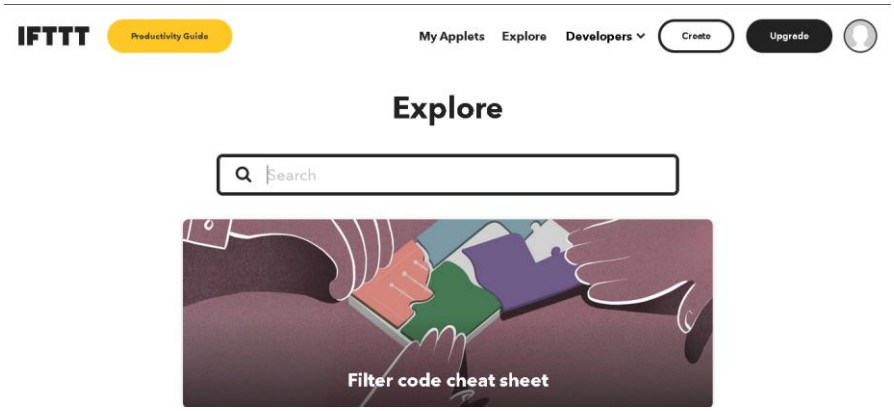
Configuring & Connecting IFTTT Web service

We can create an applet which are chains of conditional statements by a combination of several app services and add triggering parameters. For our project, we will be using this service, to send email alerts whenever motion is detected.

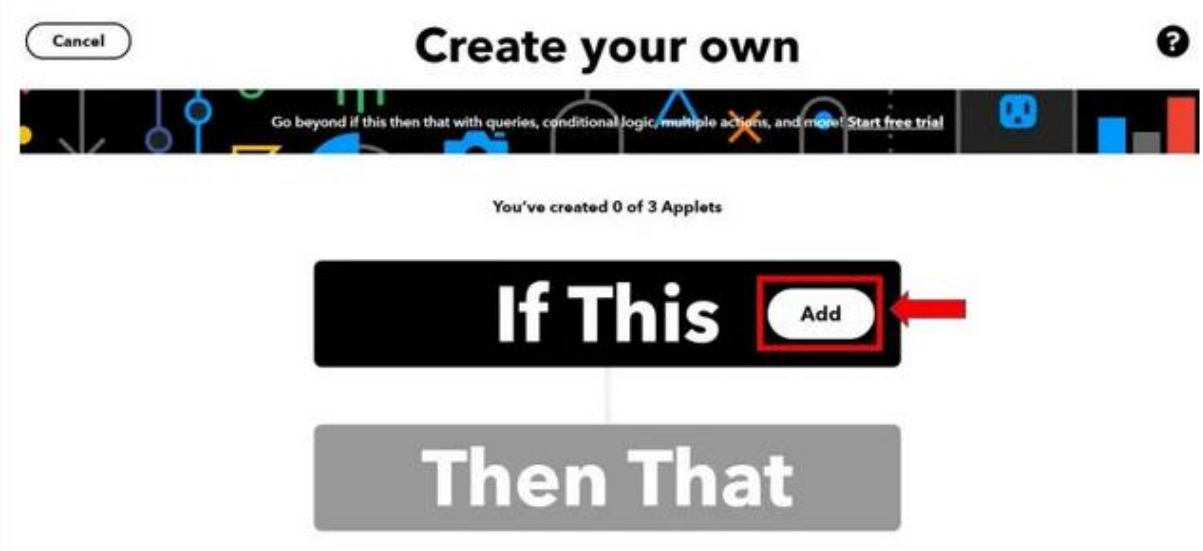
1. *Create an Account & Signup*
   First go to the following website: https://ifttt.com/
   The following window will appear. Click on the 'Get Started' button.

2. *Create an Applet*

After you have created your account, we will be directed to the page where we will create our applet. Click on 'Create.'

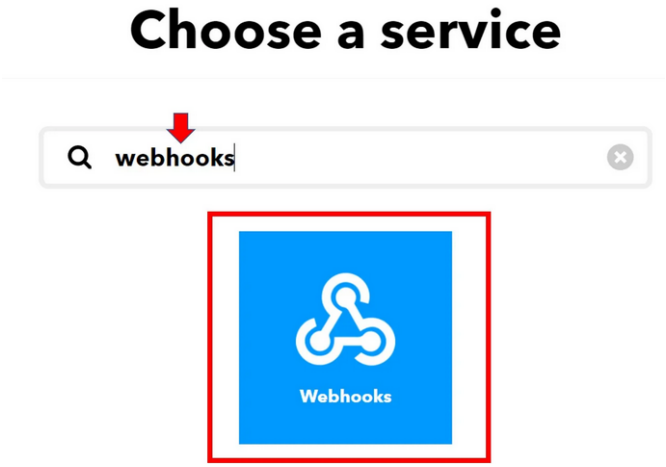


3. *Click on the 'Add' button in the If this section* :



4. *Select – Webhooks*

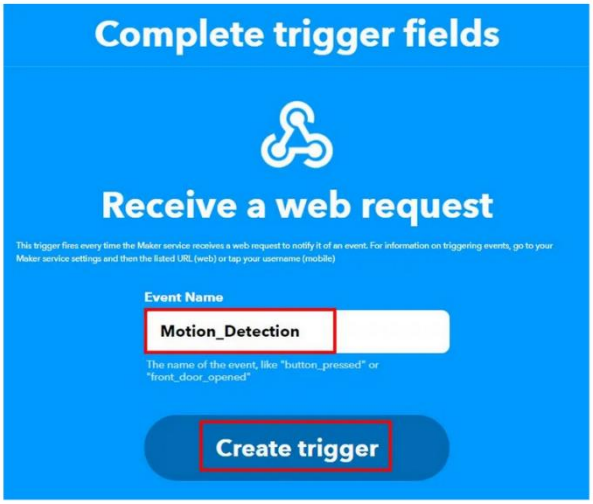
Another page will open in which we will have to choose our service. There is a lot of options to choose from. Write down ‘webhooks’ in the search option and its icon will appear:



5. *Select a trigger ‘Receive a web request’*

choose the trigger as: ‘Receive a web request’ by clicking on it. Whenever webhooks will receive a web request, some action will take place. This we will define in the ‘THAT’ section.

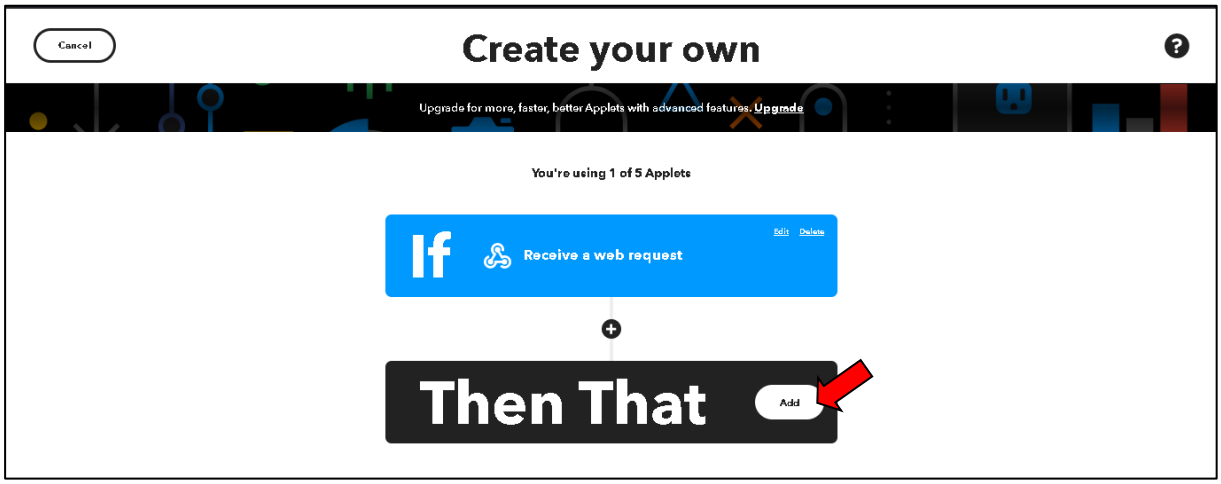
After clicking the Receive a web request, the following window will open up. We will write down Motion_Detection as the event name for the web request. You can use any other name of your choice. Click 'Create Trigger' button.

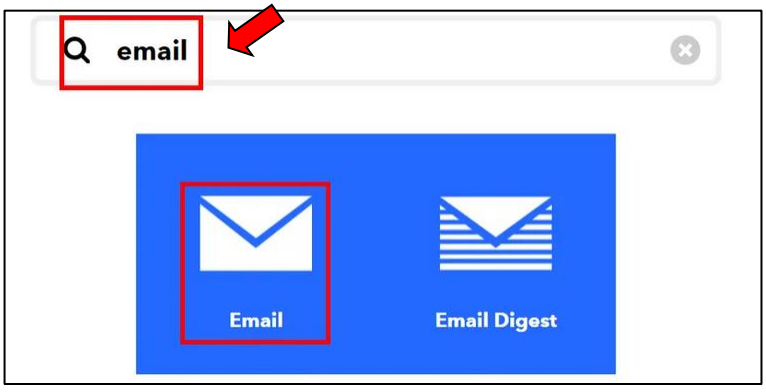


After the trigger is created, we are taken back to the web page where we first added the service for the 'IF THIS' section. Now we will click the ADD button for the 'THEN THAT' section.

6. *Choose and add a service*

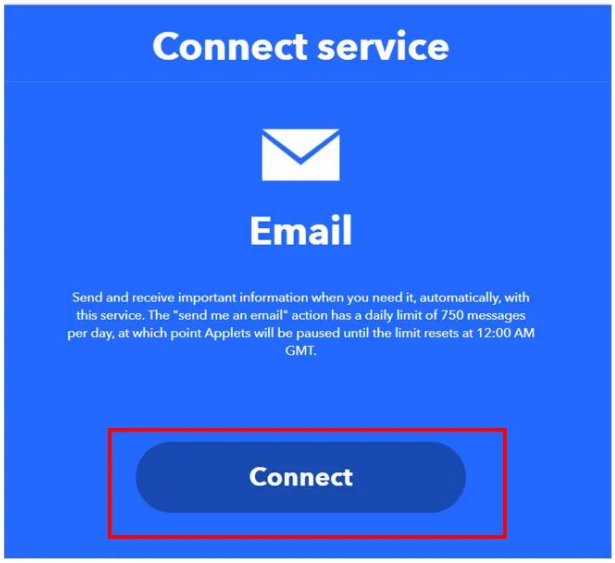
   Now we will choose the service. We have to choose what will happen if a web request is received. We will type 'email' in the search option and click on its icon. This is because we want to receive email notification whenever a web request is received.





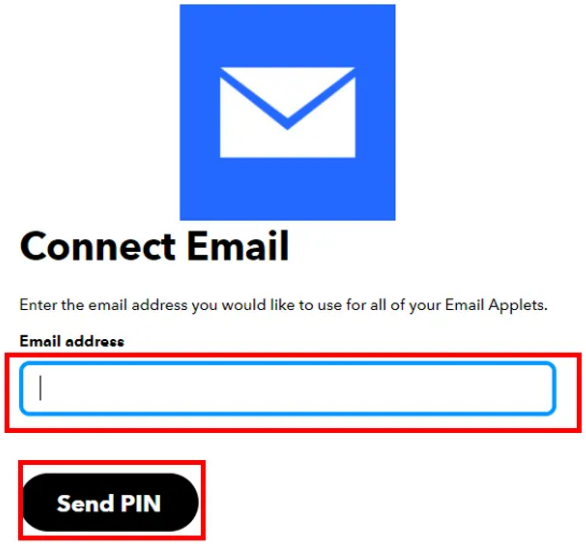
7. *Connect the service*

Click on the 'Connect' button as shown below.

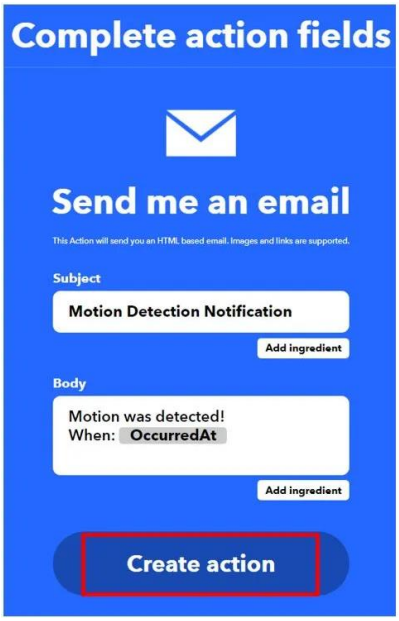


Next, write down your email address and click 'Send Pin' as shown below:

After you successfully enter the PIN, a new window will open up.

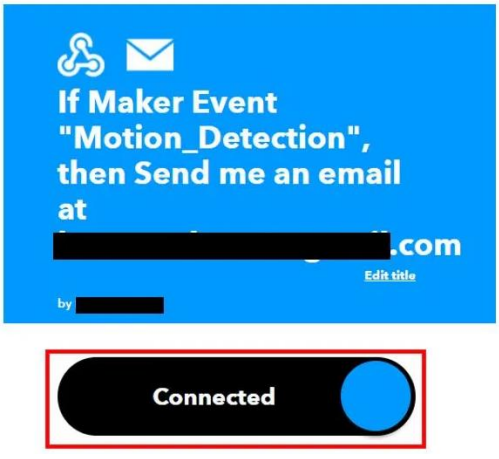


Complete the action fields by specifying the subject and body of the email. Afterwards, click ‘Create Action.’



After we have created the action, we will be guided towards the initial web page of IFTTT. Click ‘Continue’ to proceed.

After this click the Finish button. Make sure to turn ON the notifications when the applet is running. You have successfully created the applet as shown below.



8. *Obtaining private key*

Before we proceed further with our project, we want to access our private key. This is important as it will be required while programming our ESP32 board

You would see a screen after clocking, the display the key as below -

Code

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>

//Check your WiFi hotspot for these settings
const char* ssid = "WiFi-AE1FF6";
const char* password = "185319107";

const char *host = "maker.ifttt.com";
const char *privateKey = "x264Ponc0KUNa0HW21ZOt";

WebServer server(80);
void send_event(const char *event);

int led_pin = 26;
int sensor_pin = 27;
String Message;

const char MAIN_page[] PROGMEM = R"=====(
<!doctype html>
<html>
<head>
<title>IoT Motion detector</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <h1 style="text-align:center; color:red;font-size: 2.5rem;">IoT Motion Detector</h1>
   <style>
   canvas{
      -moz-user-select: none;
      -webkit-user-select: none;
      -ms-user-select: none;
   }
   #data_table {
      font-family: New Times Roman;
      border-collapse: collapse;
      width: 100%;
      text-align: center;
      font-size: 0.8rem;
   }
   #data_table td, #data_table th {
      border: 3px solid #ddd;
      padding: 15px;
   }
   #data_table tr:nth-child(even){background-color: #f7dada;}
   #data_table tr:hover {background-color: #f7dada;}
   #data_table th {
      padding-top: 20px;
      padding-bottom: 20px;
      text-align: center;
```

Please check your WiFi Spot for these setting and update the code accordingly

```
        background-color: #e00909;
        color: white;
    }
    </style>
</head>
<body>
<div>
    <table id="data_table">
        <tr><th>Time</th><th>Activity</th></tr>
    </table>
</div>
<br>
<br>
<script>
var Avalues = [];
var dateStamp = [];

setInterval(function() {
    getData();
}, 3000);
function getData() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
    var date = new Date();
    var txt = this.responseText;
    var obj = JSON.parse(txt);
            Avalues.push(obj.Activity);
            dateStamp.push(date);

        var table = document.getElementById("data_table");
        var row = table.insertRow(1);
        var cell1 = row.insertCell(0);
        var cell2 = row.insertCell(1);
        cell1.innerHTML = date;
        cell2.innerHTML = obj.Activity;
        }
    };
    xhttp.open("GET", "read_data", true);
    xhttp.send();
}
</script>
</body>
</html>
)=====";
void handleRoot() {
  String s = MAIN_page;
  server.send(200, "text/html", s);
}
```

```
void read_data() {
    int state = digitalRead(sensor_pin);
    delay(500);
    Serial.print(state);
        if(state == HIGH){
        digitalWrite (led_pin, HIGH);
        delay(1000);
        digitalWrite (led_pin, LOW);
        Message = "Motion Detected";
        String data = "{\"Activity\":\""+ String(Message) +"\"}";
        server.send(200, "text/plane", data);
        send_event("Motion_Detection");
        Serial.println("Motion detected!");
        }
}
void setup() {
  Serial.begin(115200);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print("Connecting...");
  }
  Serial.println("");
  Serial.println("Successfully connected to WiFi.");
  Serial.println("IP address is : ");
  Serial.println(WiFi.localIP());

  server.on("/", handleRoot);
  server.on("/read_data", read_data);
  server.begin();

  pinMode(sensor_pin, INPUT);
  pinMode(led_pin, OUTPUT);
  digitalWrite (led_pin, LOW);
}
void loop(){
    server.handleClient();
}
void send_event(const char *event)
{
    Serial.print("Connecting to ");
    Serial.println(host);

    WiFiClient client;
    const int httpPort = 80;
    if (!client.connect(host, httpPort)) {
        Serial.println("Connection failed");
        return;
```

```
    }

    String url = "/trigger/";
    url += event;
    url += "/with/key/";
    url += privateKey;
    Serial.print("Requesting URL: ");
    Serial.println(url);
    client.print(String("GET ") + url + " HTTP/1.1\r\n" +
                 "Host: " + host + "\r\n" +
                 "Connection: close\r\n\r\n");
    while(client.connected())
    {
      if(client.available())
      {
        String line = client.readStringUntil('\r');
        Serial.print(line);
      } else {
        delay(50);
      };
    }
    Serial.println();
    Serial.println("Closing Connection");
    client.stop();
}
```

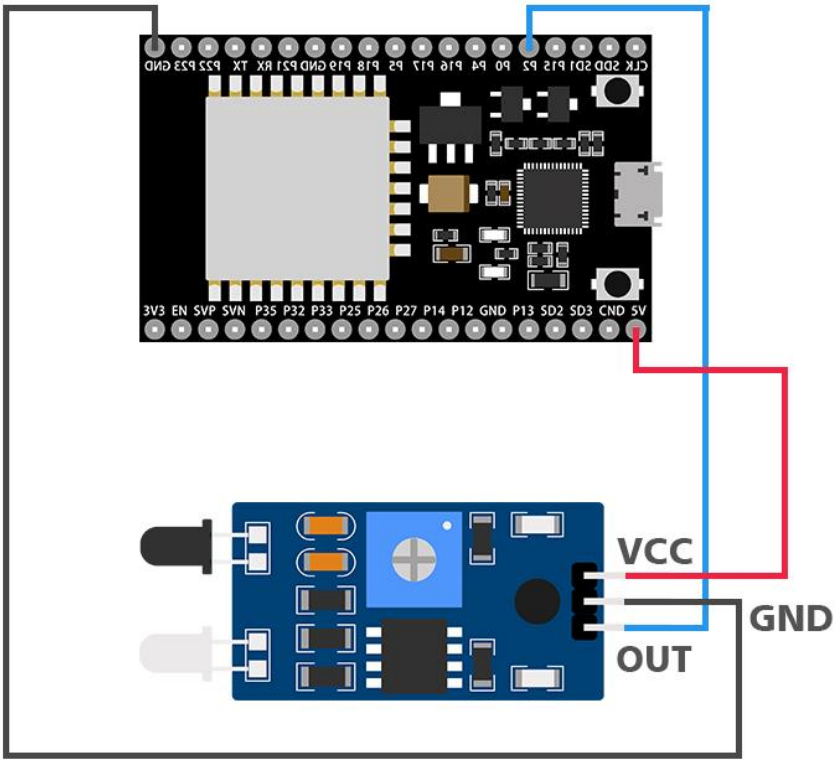
Project 10: IR motion detection

This IR sensor can be used to detect objects or obstacles ahead using reflected infrared light.
The sensor has 2 main parts, namely IR transmitter and IR receiver. The infrared transmitter emits infrared light. When it hits an object, the infrared light gets reflected back.

When the infrared receiver receives the reflected infrared light, the output will be "low". When the infrared receiver does not receive the reflected infrared light, the output will be "high".

There are 2 LED indicators in the sensor. Power indicator light and output indicator light. If the module is powered by current, the power indicator LED will light up. If there is an object in front of the sensor or infrared receiver to receive infrared light reflection, the output indicator LED will light up.

Connections:

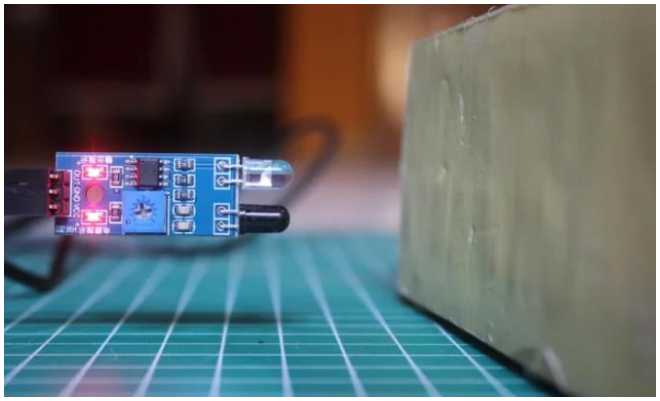


Code:

```
int pinIR = 2;
void setup(){
   Serial.begin(115200);
   pinMode(pinIR, INPUT);
   Serial.println("Detect IR Sensor");
   delay(1000);
}
void loop(){
   int IRstate = digitalRead(pinIR);

   if(IRstate == LOW){
      Serial.println("Detected");

   }
   else if(IRstate == HIGH){
        Serial.println("Not Detected");
   }
   delay(1000);
}
```

Demonstration:

Bring an object closer to the sensor and open the serial monitor to see the code in action.



If you place an object in front of the sensor, the serial monitor will say "Detected".

if there is no object in front of the sensor, the monitor serial will say "Not Detected".