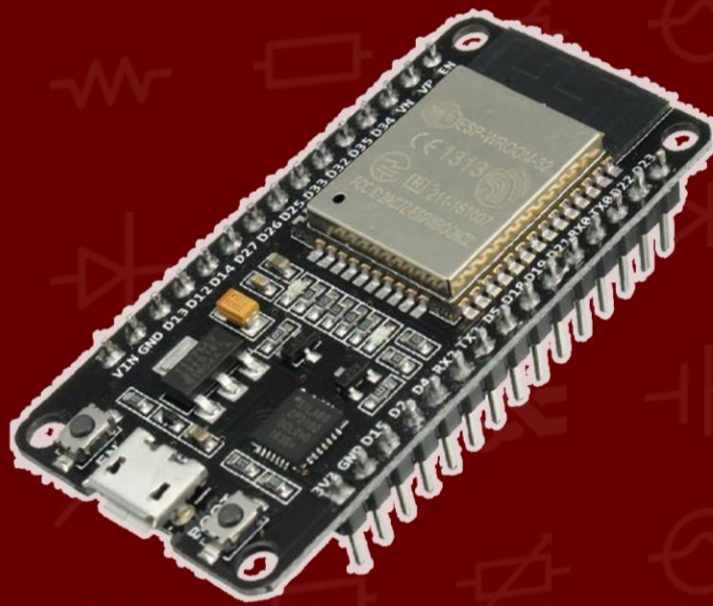


ESP32 WEB SERVER WITH ARDUINO IDE

ESP32 WEB SERVER WITH ARDUINO IDE



STEP-BY-STEP PROJECT GUIDE

Rui Santos & Sara Santos

ESP32 Web Server with Arduino IDE

Hello and thank you for downloading this project eBook!

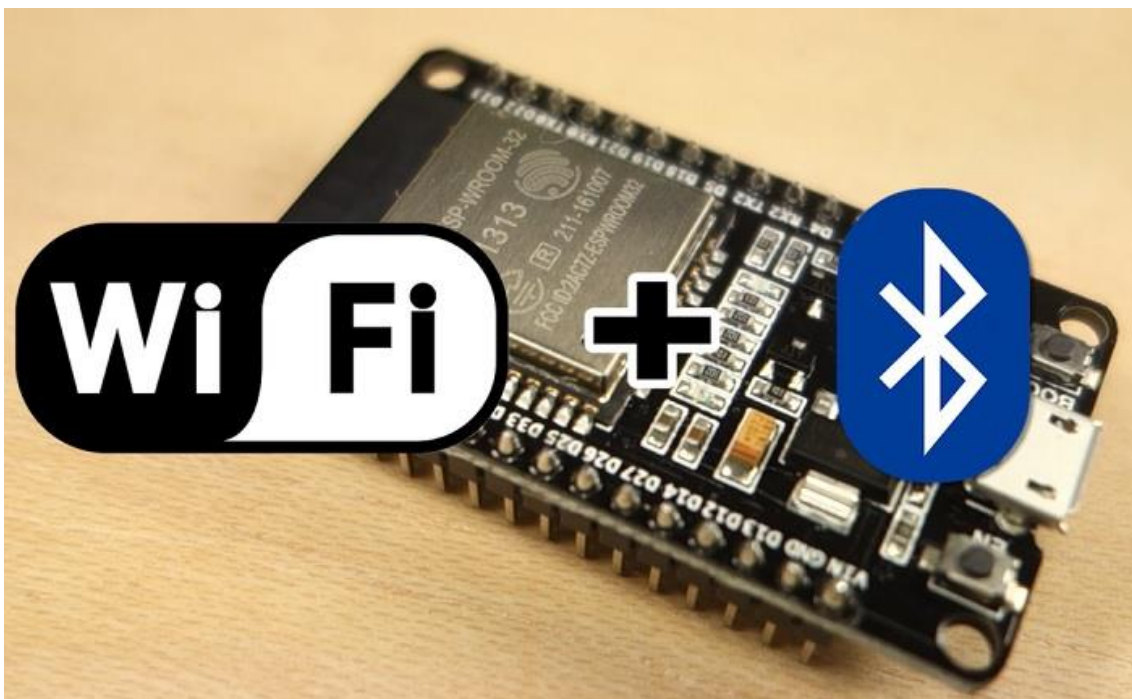
This quick eBook will help you build getting started and building a web server with the ESP32 using Arduino IDE.

If you want to learn more about the ESP32, make sure you take a look at our course:

- [Learn ESP32 with Arduino IDE](#)

Introducing the ESP32 Board

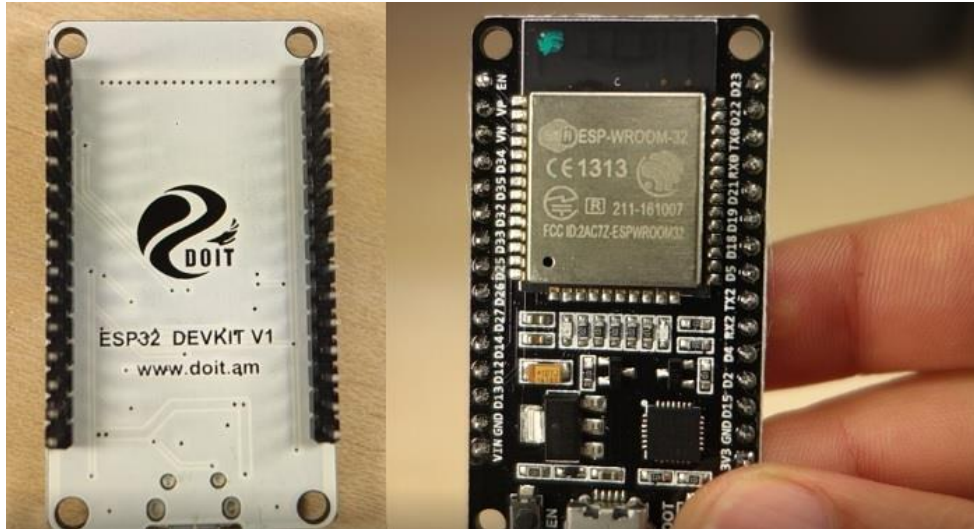
The ESP32 is the ESP8266 successor. It is loaded with lots of new features. It now combines Wi-Fi and Bluetooth wireless capabilities.



There are a lot of ESP32 development boards. I encourage you to visit the [ESP32.net website](https://www.ESP32.net) where each ESP32 chip and development board are listed. You can compare their differences and features.



In this ebook we'll be using the [ESP32 DEVKIT V1 DOIT board](#), but any other ESP32 with the ESP-WROOM-32 chip will work just fine.



Here's just a few examples of boards that are very similar and compatible with the project in this ebook.

DOIT DEVKIT V1



ESP32 DevKit



ESP-32S NodeMCU



ESP32 Thing



WEMOS LOLIN32



"WeMos" OLED



HUZZAH32



Others

(...)

To compare several ESP32 development boards, read the following article:

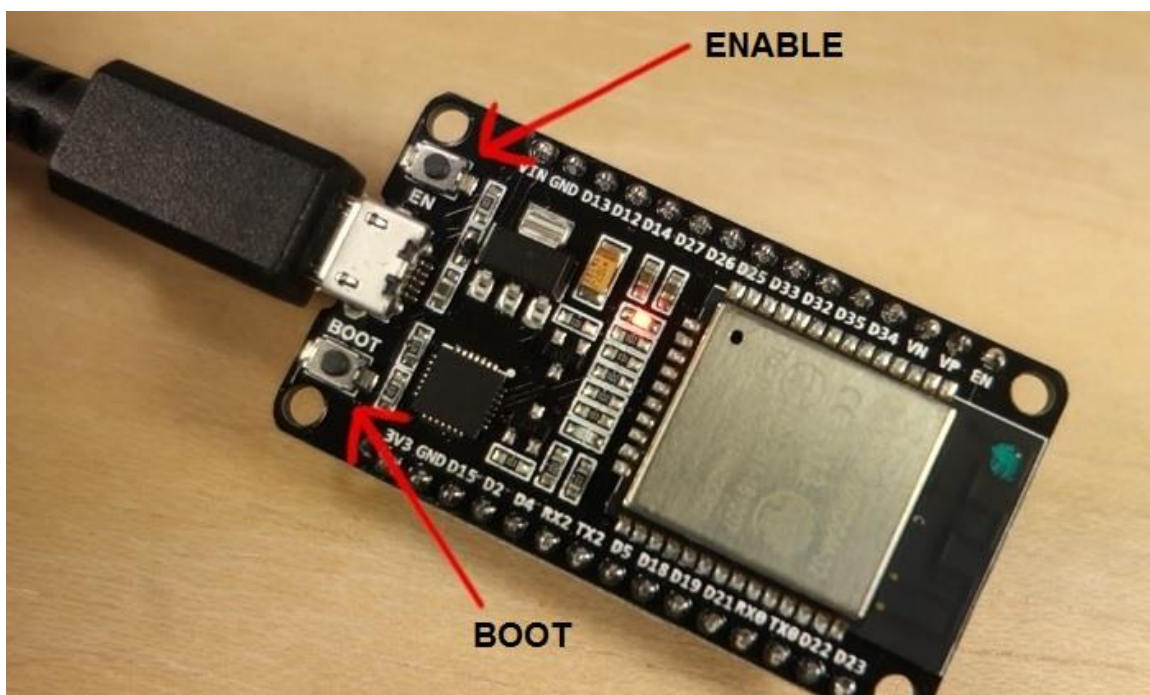
- [Best ESP32 development boards review and comparison](#)

Features

The ESP32 comes with the ESP-WROOM-32 chip. It has a 3.3V voltage regulator that drops the input voltage to power the ESP32 chip. And it also comes with a CP2102 chip that allows you to plug the ESP32 to your computer to program it without the need for an FTDI programmer.



The board has two on-board buttons: the ENABLE and the BOOT button.



If you press the ENABLE button, it reboots the ESP32. If you hold down the BOOT button and then press the enable, the ESP32 reboots in programming mode.

If you don't know where to get the ESP32, you can check this page on [Maker Advisor](#).

Specifications

When it comes to the ESP32 chip specifications, you'll find that:

- The ESP32 is dual core, this means it has 2 processors.
- It has Wi-Fi and bluetooth built-in.
- It runs 32 bit programs.
- The clock frequency can go up to 240MHz and it has a 512 kB RAM.
- It also has wide variety of peripherals available, like: capacitive touch, ADCs, DACs, UART, SPI, I2C and much more.

Specifications - ESP32 DEVKIT V1 DOIT	
Number of cores	2 (Dual core)
Wi-Fi	2.4 GHz up to 150 Mbit/s
Bluetooth	BLE (Bluetooth Low Energy) and legacy Bluetooth
Architecture	32 bits
Clock frequency	Up to 240 MHz
RAM	512 KB
Pins	30
Peripherals	Capacitive touch, ADCs (analog-to-digital converter), DACs (digital-to-analog converter), I ² C (Inter-Integrated Circuit), UART (universal asynchronous receiver/transmitter), CAN 2.0 (Controller Area Network), SPI (Serial Peripheral Interface), I ² S (Integrated Inter-IC Sound), RMII (Reduced Media-Independent Interface), PWM (pulse width modulation), and more.

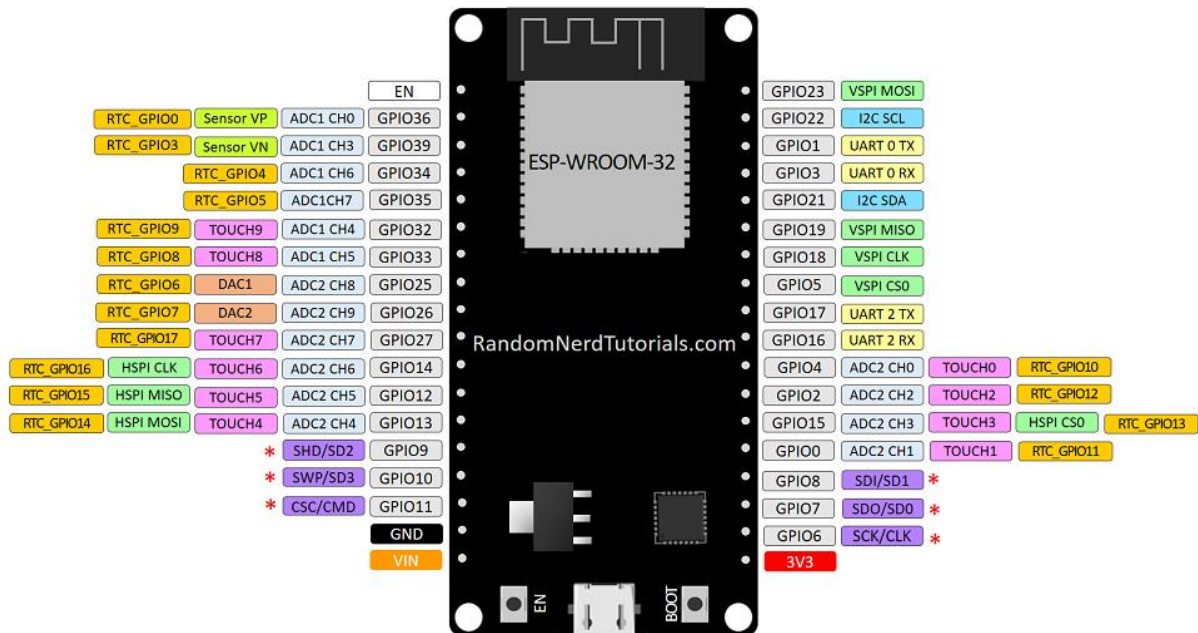
ESP32 Pinout

The following figures clearly describe the board GPIOs and their functionalities. We recommend printing this pinout for a future reference. You can download the pinout in *.pdf* or *.png* files:

- [Printable version](#)
- [Image version 30 pins](#)
- [Image version 36 pins](#)

ESP32 DEVKIT V1 – DOIT

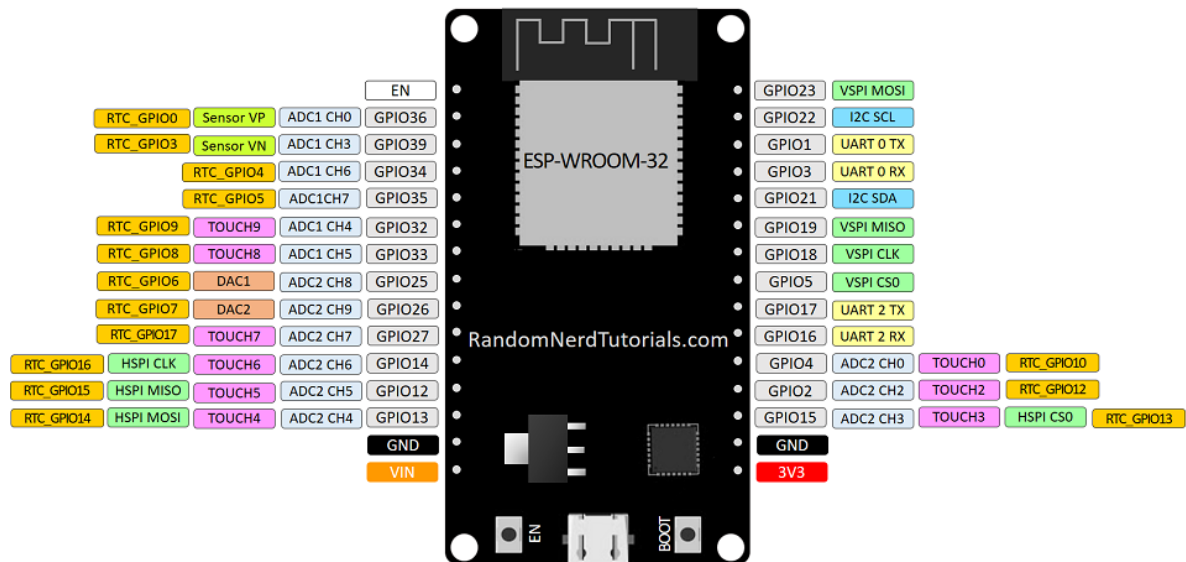
version with 36 GPIOs



* Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and CSC/CMD, namely, GPIO6 to GPIO11 are connected to the integrated SPI flash integrated on ESP-WROOM-32 and are not recommended for other uses.

ESP32 DEVKIT V1 – DOIT

version with 30 GPIOs



Learn how to use the ESP32 GPIOs:

- [ESP32 Pinout Reference: Which GPIO pins should you use?](#)

Installing ESP32 in Arduino IDE

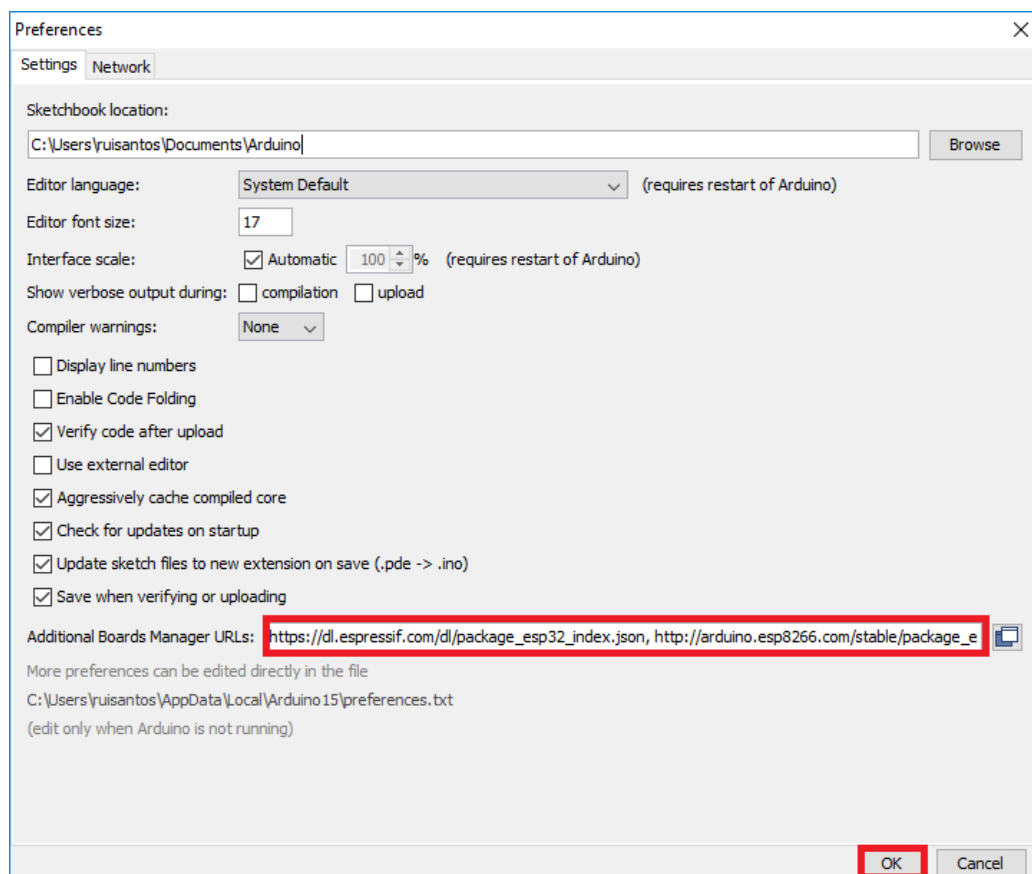
Important: before starting this installation procedure, make sure you have the latest version of the Arduino IDE installed in your computer. If you don't, uninstall it and install it again. Otherwise, it may not work.

The ESP32 is currently being integrated with the Arduino IDE just like it was done for the ESP8266. This add-on for the Arduino IDE allows you to program the ESP32 using the Arduino IDE and its programming language. You can find the latest Windows instructions at the official [GitHub repository](#).

1. Installing the ESP32 Board

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

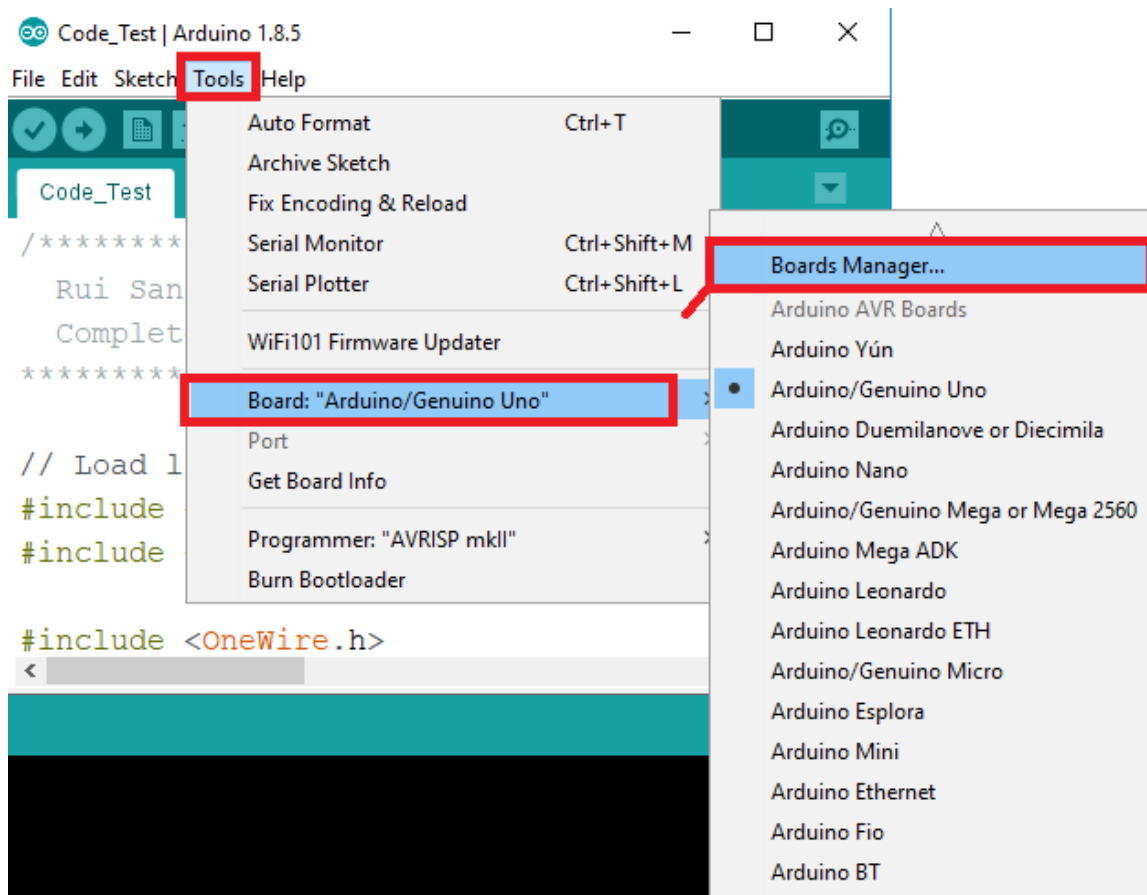
- 1) Open the preferences window from the Arduino IDE. Go to **File ▶ Preferences**
- 2) Enter **https://dl.espressif.com/dl/package_esp32_index.json** into the “**Additional Board Manager URLs**” field as shown in the figure below. Then, click the “**OK**” button.



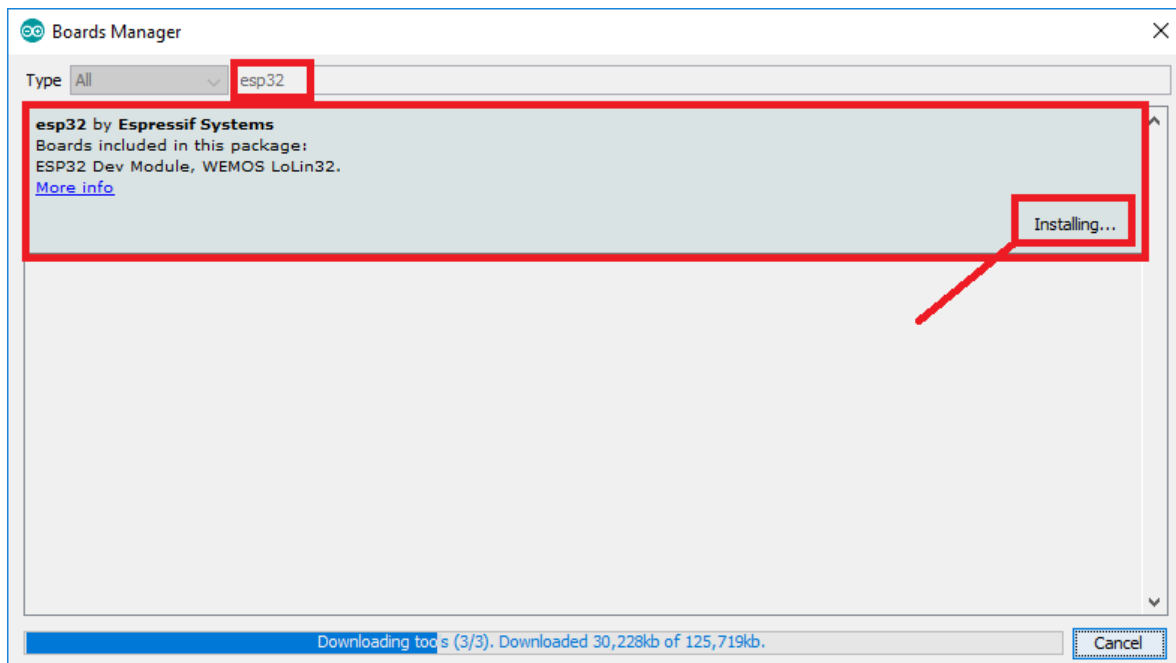
Note: if you already have the ESP8266 boards URL, you can separate the URLs with a comma as follows:

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

3) Open boards manager. Go to **Tools** ▶ **Board** ▶ **Boards Manager...**



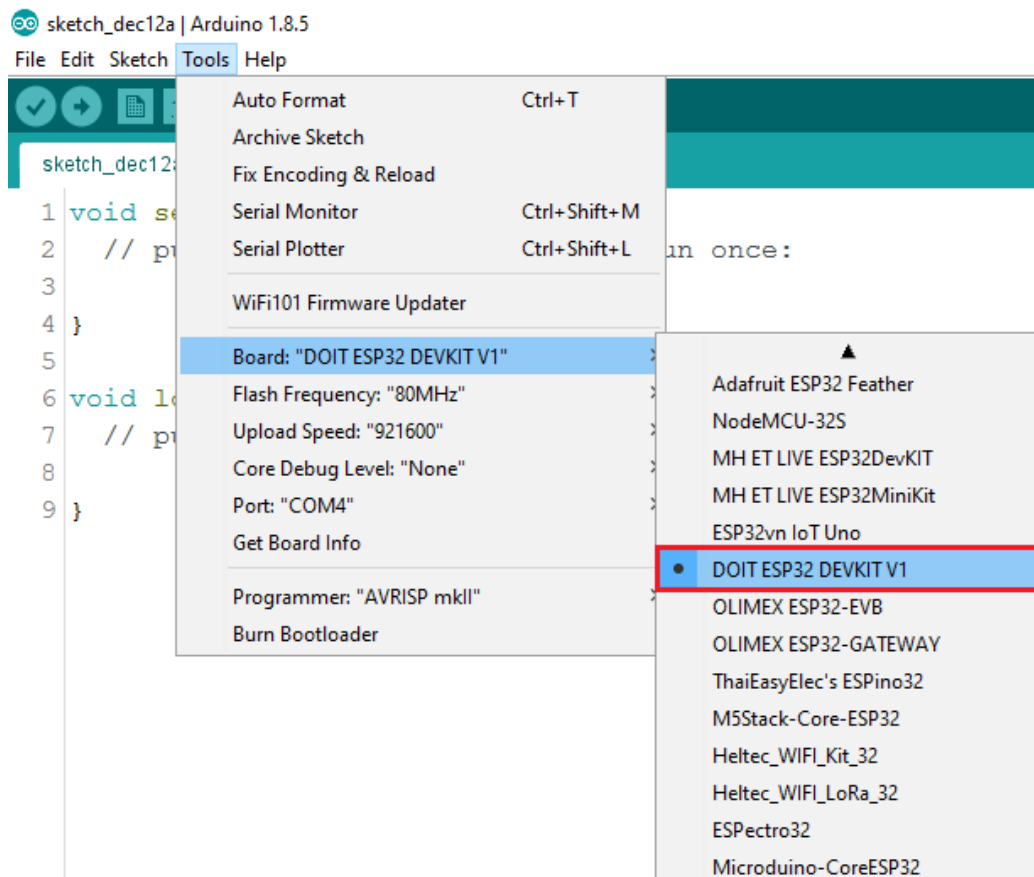
4) Search for ESP32 and press install button for the "ESP32 by Espressif Systems":



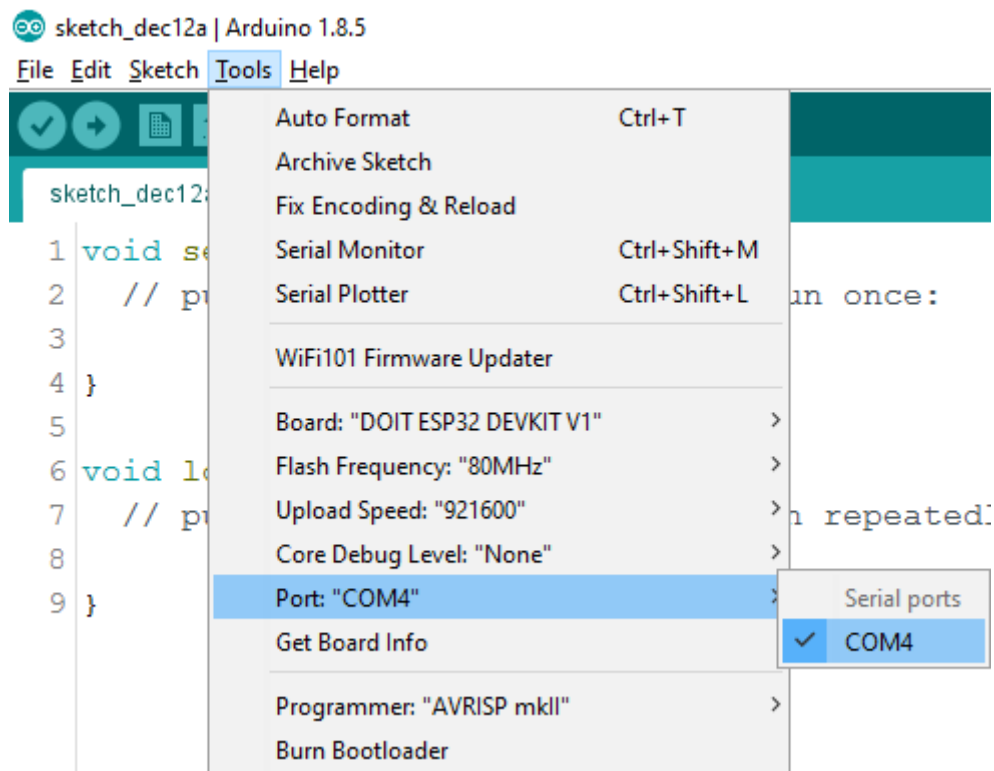
Testing the Installation

Plug your [ESP32 DOIT DEVKIT V1 Board](#) to your computer. Then, follow these steps:

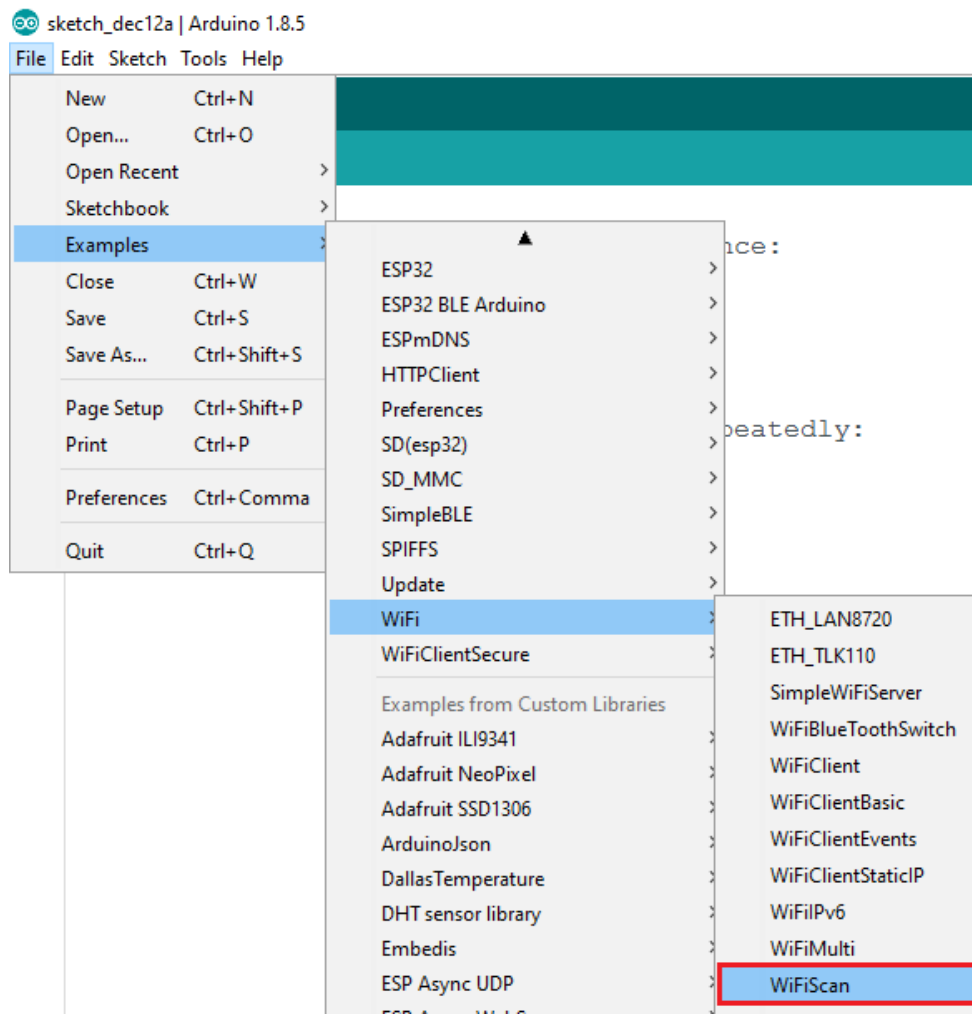
- 1) Open Arduino IDE
- 2) Select your **Board** in **Tools** ▶ **Board** menu (in our case it's the **DOIT ESP32 DEVKIT V1**)



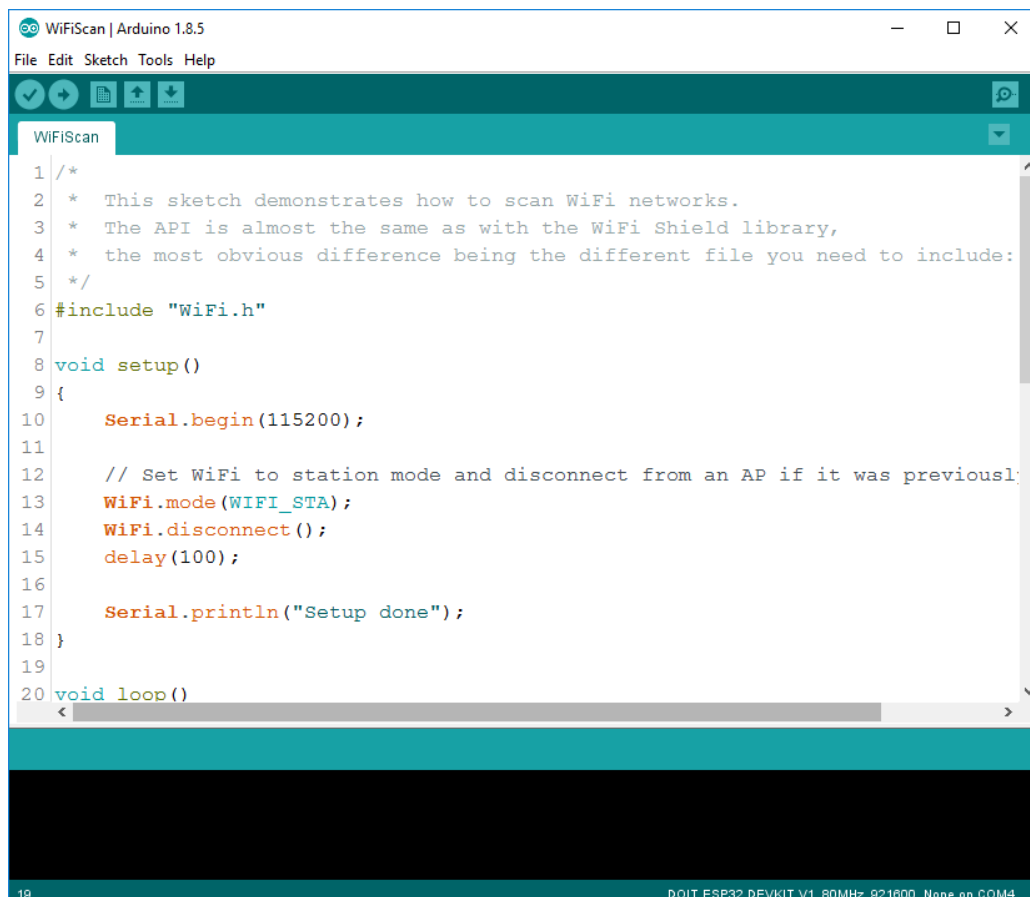
3) Select the Port (if you don't see the COM Port in your Arduino IDE, you need to install the [ESP32 CP210x USB to UART Bridge VCP Drivers](#)):



4) Open the following example under **File** ▶ **Examples** ▶ **WiFi (ESP32)** ▶ **WiFi Scan**



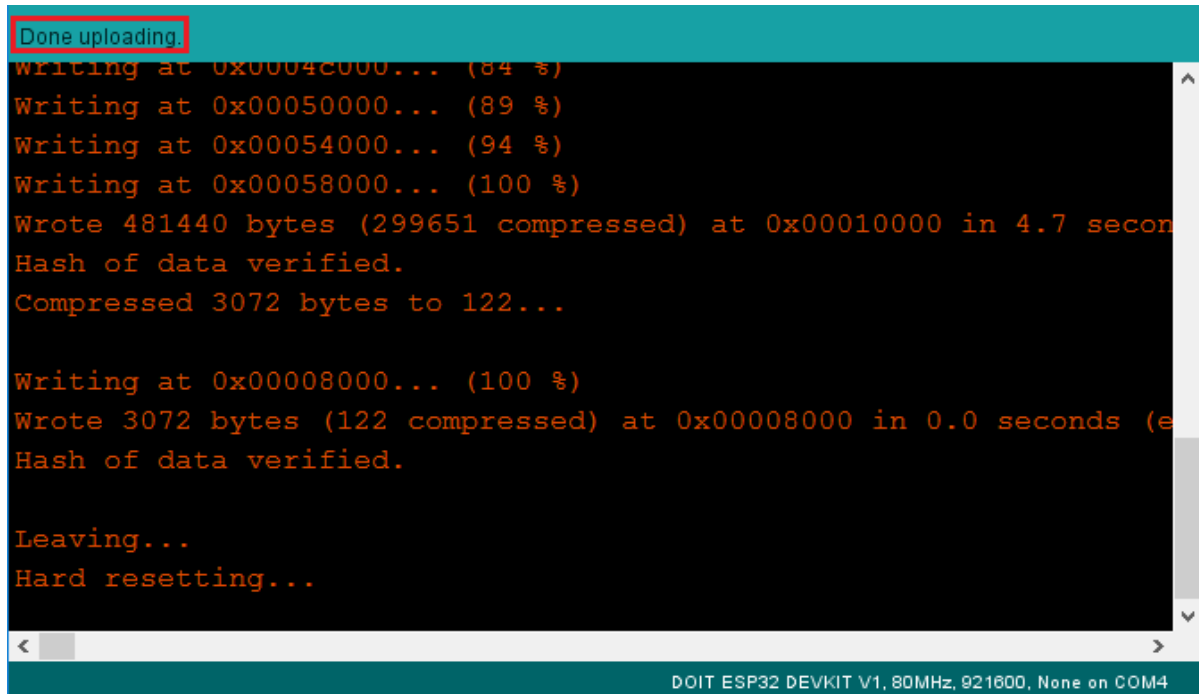
5) A new sketch opens:



6) Press the Upload button in the Arduino IDE. Wait a few seconds while the code compiles and uploads to your board.



7) If everything went as expected, you should see a **"Done uploading."** message.

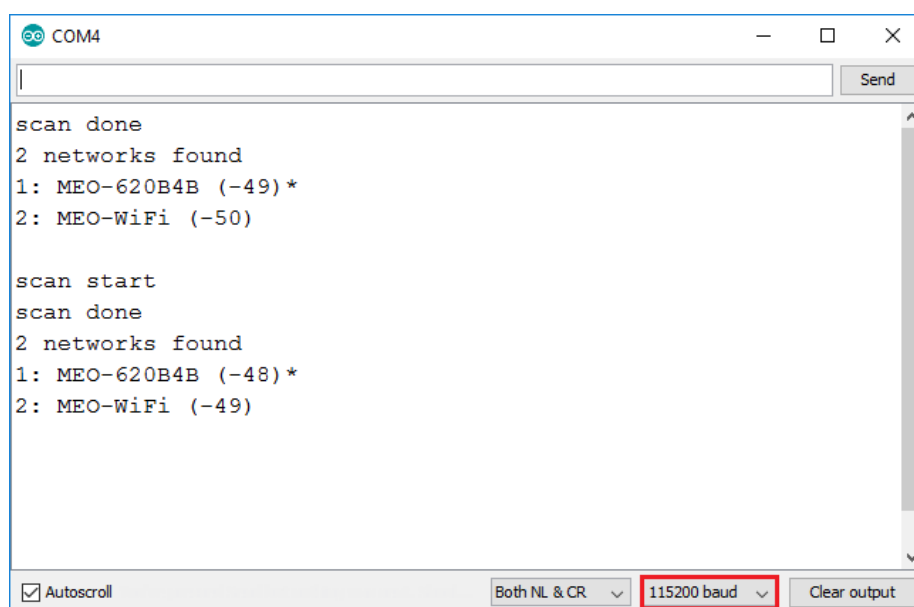
A screenshot of the Arduino IDE Serial Monitor window. The title bar says "COM4". The text area shows the upload progress: "Done uploading." (highlighted with a red box), "writing at 0x00040000... (84 %)", "Writing at 0x00050000... (89 %)", "Writing at 0x00054000... (94 %)", "Writing at 0x00058000... (100 %)", "Wrote 481440 bytes (299651 compressed) at 0x00010000 in 4.7 seconds", "Hash of data verified.", "Compressed 3072 bytes to 122...", "Writing at 0x00008000... (100 %)", "Wrote 3072 bytes (122 compressed) at 0x00008000 in 0.0 seconds (e", "Hash of data verified.", "Leaving...", "Hard resetting...". The status bar at the bottom says "DOIT ESP32 DEVKIT V1, 80MHz, 921600, None on COM4".

```
Done uploading.  
writing at 0x00040000... (84 %)  
Writing at 0x00050000... (89 %)  
Writing at 0x00054000... (94 %)  
Writing at 0x00058000... (100 %)  
Wrote 481440 bytes (299651 compressed) at 0x00010000 in 4.7 seconds  
Hash of data verified.  
Compressed 3072 bytes to 122...  
  
Writing at 0x00008000... (100 %)  
Wrote 3072 bytes (122 compressed) at 0x00008000 in 0.0 seconds (e  
Hash of data verified.  
  
Leaving...  
Hard resetting...
```

8) Open the Arduino IDE Serial Monitor at a baud rate of 115200:



9) Press the ESP32 on-board Enable button and you should see the networks available near your ESP32:

A screenshot of the Arduino IDE Serial Monitor window. The title bar says "COM4". The text area shows the network scan results: "scan done", "2 networks found", "1: MEO-620B4B (-49) *", "2: MEO-WiFi (-50)", "scan start", "scan done", "2 networks found", "1: MEO-620B4B (-48) *", "2: MEO-WiFi (-49)". The status bar at the bottom has "Autoscroll" checked, "Both NL & CR" selected, "115200 baud" (highlighted with a red box), and "Clear output".

```
scan done  
2 networks found  
1: MEO-620B4B (-49) *  
2: MEO-WiFi (-50)  
  
scan start  
scan done  
2 networks found  
1: MEO-620B4B (-48) *  
2: MEO-WiFi (-49)
```

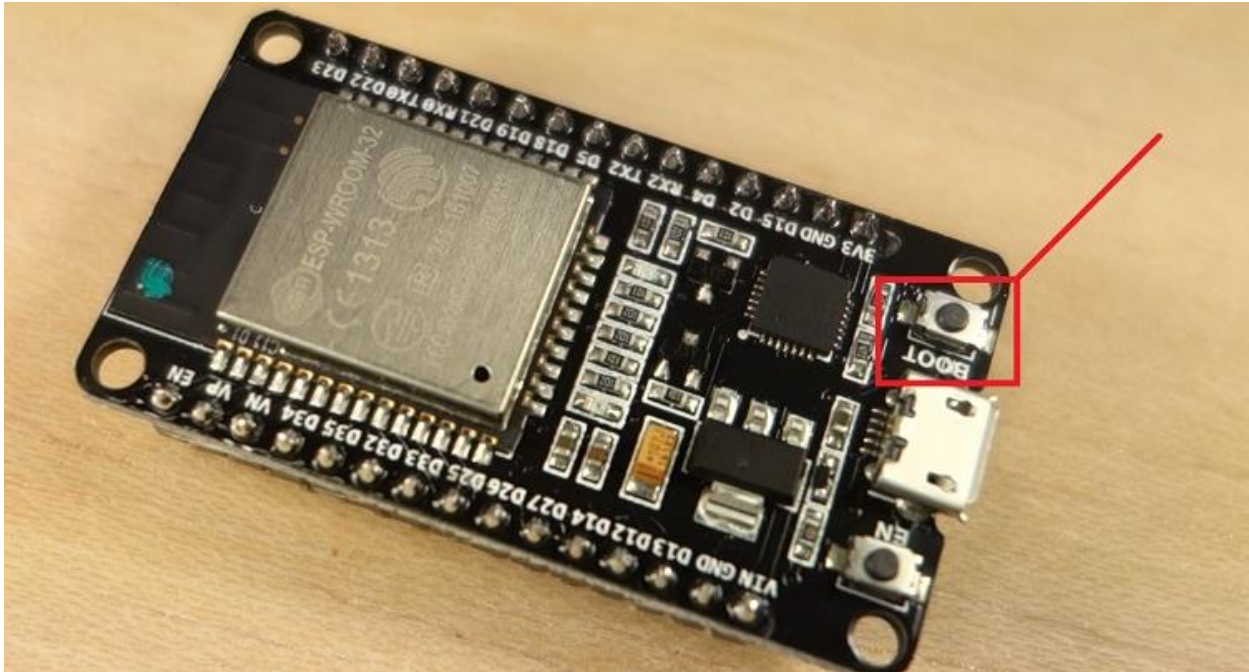
This is a very basic tutorial that illustrates how to prepare your Arduino IDE for the ESP32 on your computer.

Troubleshooting Tip #1:

“Failed to connect to ESP32: Timed out... Connecting...”

When you try to upload a new sketch to your ESP32 and it fails to connect to your board, it means that your ESP32 is not in flashing/uploading mode. Having the right board name and COM port selected, follow these steps:

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



- Press the “**Upload**” button in the Arduino IDE to upload a new sketch:



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

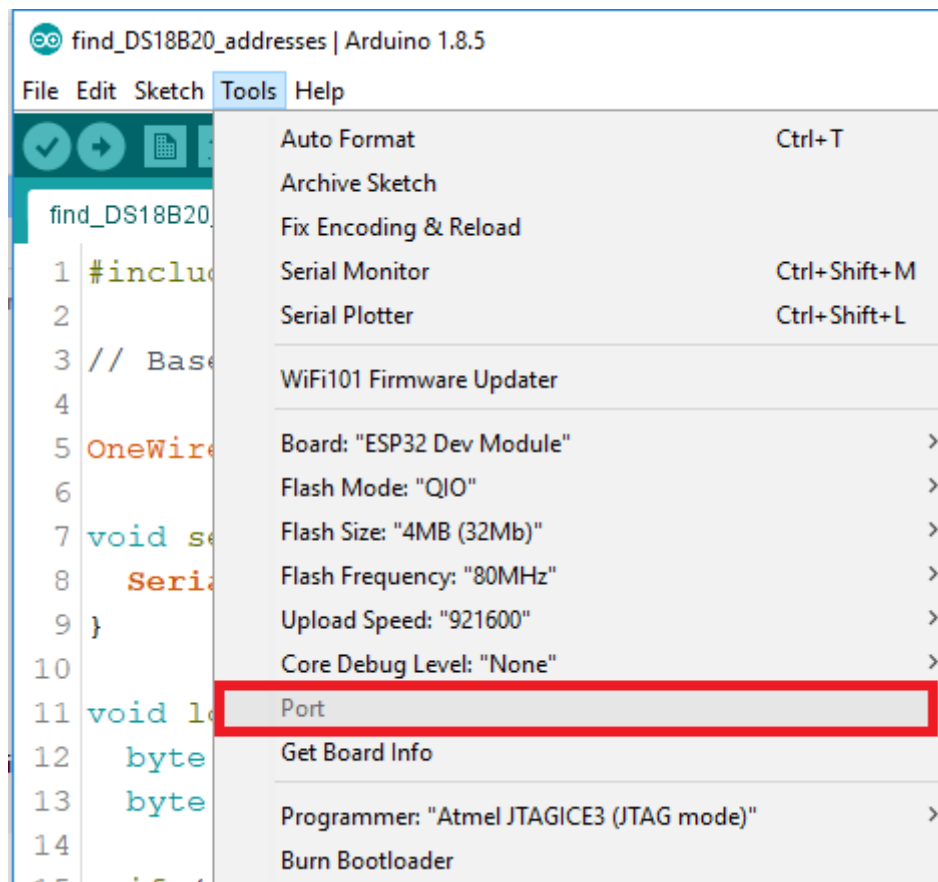
```
Uploading...
Archiving built core (caching) in: C:\Users\RUISAN~1\AppData\Local\Temp\arduino_cache_959883\c
Sketch uses 501366 bytes (38%) of program storage space. Maximum is 1310720 bytes.
Global variables use 37320 bytes (12%) of dynamic memory, leaving 257592 bytes for local varia
esptool.py v2.1
Connecting.....
Chip is ESP32D0WDQ6 (revision (unknown 0xa))
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 921600
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 8192 bytes to 47...
```

After that, you should see the “**Done uploading**” message.

Troubleshooting Tip #2:

COM Port not found/not available

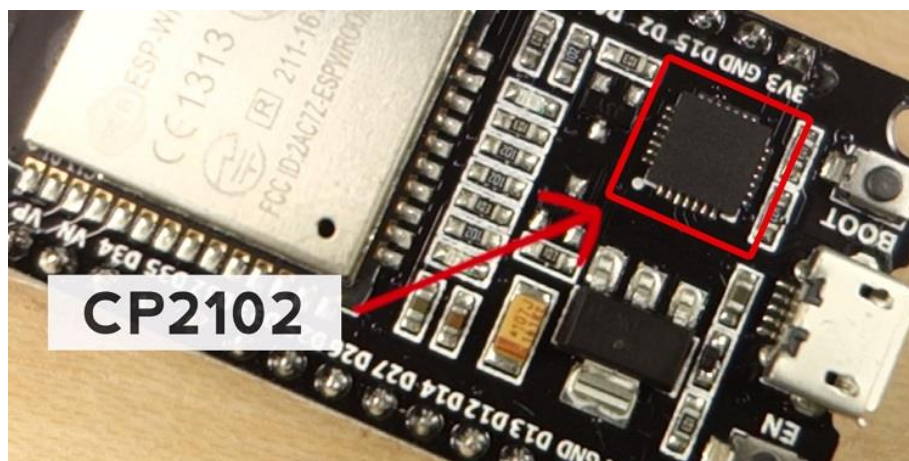
If you plug your ESP32 board to your computer, but you can't find the ESP32 Port available in your Arduino IDE (it's grayed out):



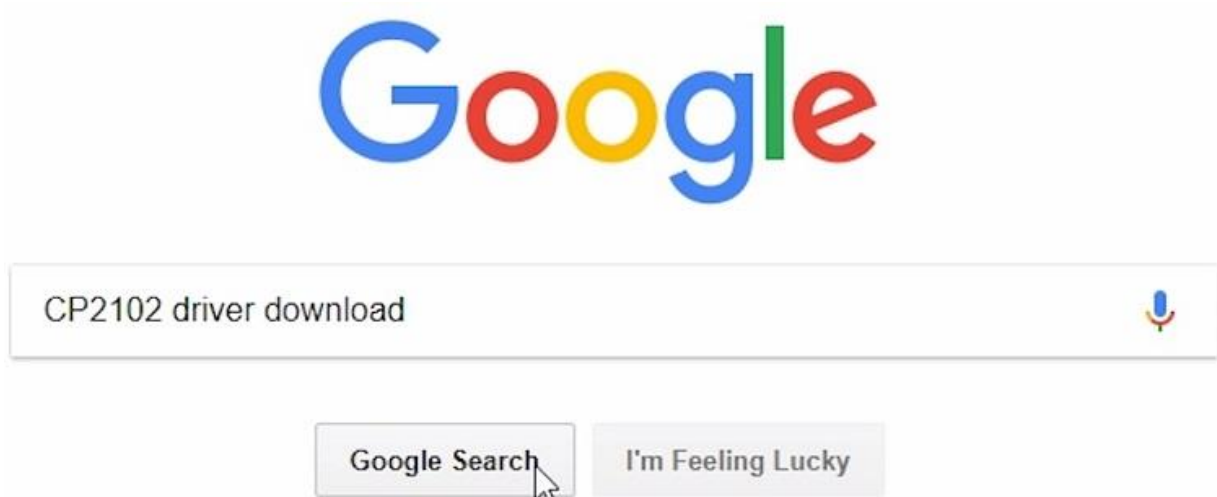
It might be one of these two problems: **1. USB drivers missing** or **2. USB cable without data wires**.

1. If you don't see your ESP's COM port available, this often means you don't have the USB drivers installed. Take a closer look at the chip next to the voltage regulator on board and check its name.

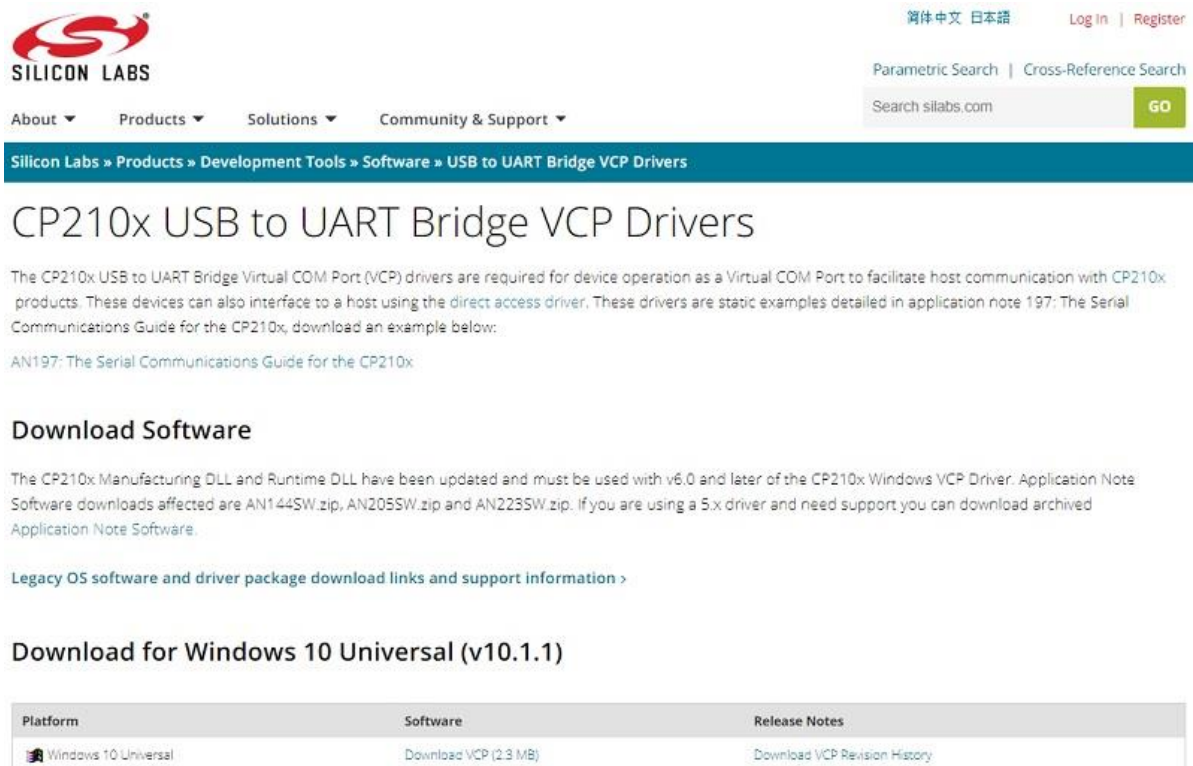
The [ESP32 DEVKIT V1 DOIT](#) board uses the CP2102 chip.



Go to Google and search for your particular chip to find the drivers and install them in your operating system.



You can download the CP2102 drivers on the [Silicon Labs](https://www.siliconlabs.com) website.



The screenshot shows the Silicon Labs website. The header includes the Silicon Labs logo, navigation links (About, Products, Solutions, Community & Support), and user links (Log In, Register). A search bar is also present. The main content area is titled 'CP210x USB to UART Bridge VCP Drivers'. It contains a paragraph explaining that these drivers are required for device operation as a Virtual COM Port. Below this, there is a section for 'Download Software' which mentions that the CP210x Manufacturing DLL and Runtime DLL have been updated and must be used with v6.0 and later of the CP210x Windows VCP Driver. It also provides links for 'Legacy OS software and driver package download links and support information' and 'Download for Windows 10 Universal (v10.1.1)'. At the bottom, there is a table with three columns: Platform, Software, and Release Notes. The table has one row for 'Windows 10 Universal' with links to 'Download VCP (2.3 MB)' and 'Download VCP Revision History'.

Platform	Software	Release Notes
Windows 10 Universal	Download VCP (2.3 MB)	Download VCP Revision History

After they are installed, restart the Arduino IDE and you should see the COM port in the Tools menu.

2. If you have the drivers installed, but you can't see your device, double-check that you're using a USB cable with data wires. USB cables from powerbanks often don't have data wires (they are charge only). So, your computer will never establish a serial communication with your ESP32. Using a proper USB cable should solve your problem.

How to Identify Your ESP32 Board

To make it easier to follow along, regardless of the ESP32 board you're using, we've created this section to show you which changes you need to do to make your ESP32 work.

Visiting the Product Page

To identify your board, you can go to the product page where you purchased your ESP32. I've ordered mine from [Banggood](#) and here's the product page.



Please note that some vendors will add all sorts of keywords to their product name, so you might be thinking that you're ordering an ESP-32S NodeMCU board and you actually bought an ESP32 DOT IT board. Even though they are very similar, they are different.

ESP32 Board Name

After reading your ESP32 product page, you should know the name of your board. But if you still have doubts, you can take a look at the back of the board.

The name is usually printed with silk screen. In my case, you can clearly see this is the ESP32 DEVKIT V1 DOIT board.



If you have an ESP32 NodeMCU, the following figure shows how it looks like (it says NodeMCU ESP-32S).



ESP32 Pinout

Knowing the name of your board is very important so that you can search for its pinout. Now you can go to Google and search for your development board pinout. Search for your ESP32 board name and add the “pinout” keyword in the end.



ESP32 DEVKIT V1 DOIT pinout



Then, find one image that has the same pinout as your ESP32.

I also recommend visiting the ESP32.net website as it provides an extensive list with names and figures for all known ESP32 development boards.

Blink – Example Sketch

Let’s take a look at an example sketch to show you what you need to worry about, if you want to build a simple circuit that blinks an LED with the ESP32. Copy the following code to the Arduino IDE:

SOURCE CODE

https://github.com/RuiSantosdotme/ESP32-Course/blob/master/code/Blink_LED/Blink_LED.ino

```
/*
  Blink
*/

// ledPin refers to ESP32 GPIO 23
const int ledPin = 23;

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin ledPin as an output.
  pinMode(ledPin, OUTPUT);
}

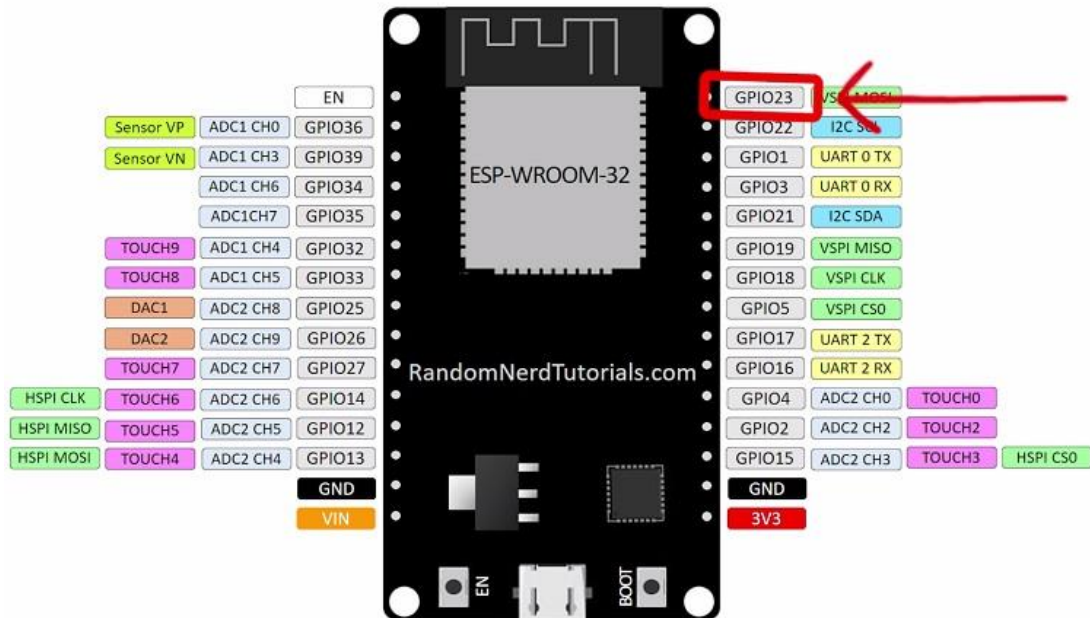
// the loop function runs over and over again forever
void loop() {
  digitalWrite(ledPin, HIGH); // turn the LED on (HIGH is the voltage
level)
  delay(1000);                // wait for a second
  digitalWrite(ledPin, LOW);  // turn the LED off by making the voltage
LOW
  delay(1000);                // wait for a second
}
```


As you can see, you need to connect an LED to pin 23 which refers to GPIO 23:

```
const int ledPin = 23;
```

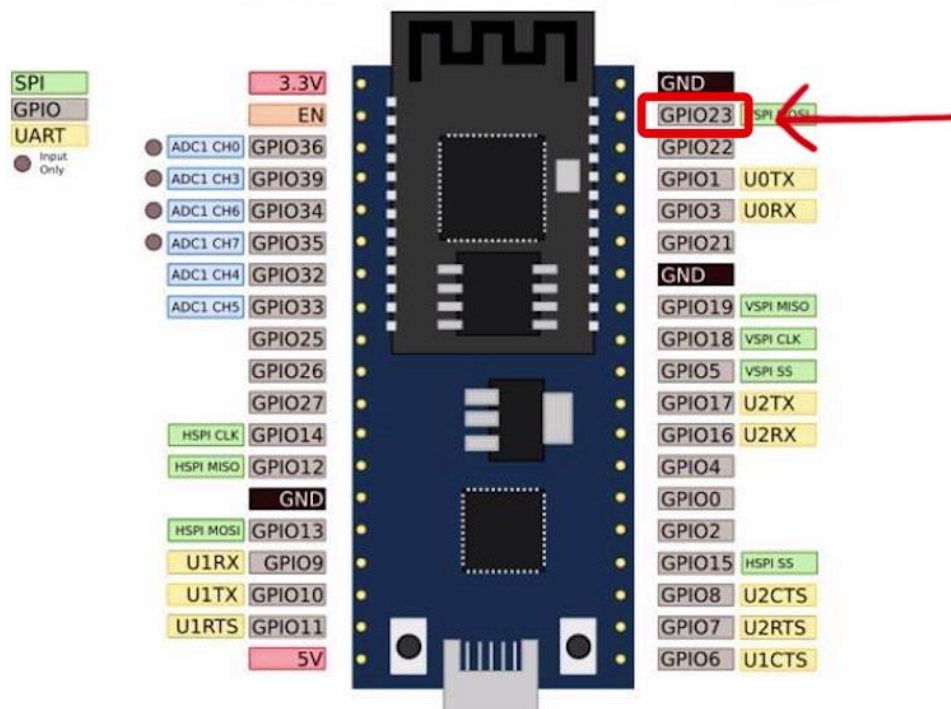
If you're using the ESP32 DEVKIT V1 DOIT board, you need to connect your LED to the first pin on the top right corner.

ESP32 DEVKIT V1 - DOIT



But if you're using the NodeMCU ESP-32S board, GPIO 23 is located in the 2nd pin of the top right corner, as shown in the figure below.

NodeMCU ESP-32S



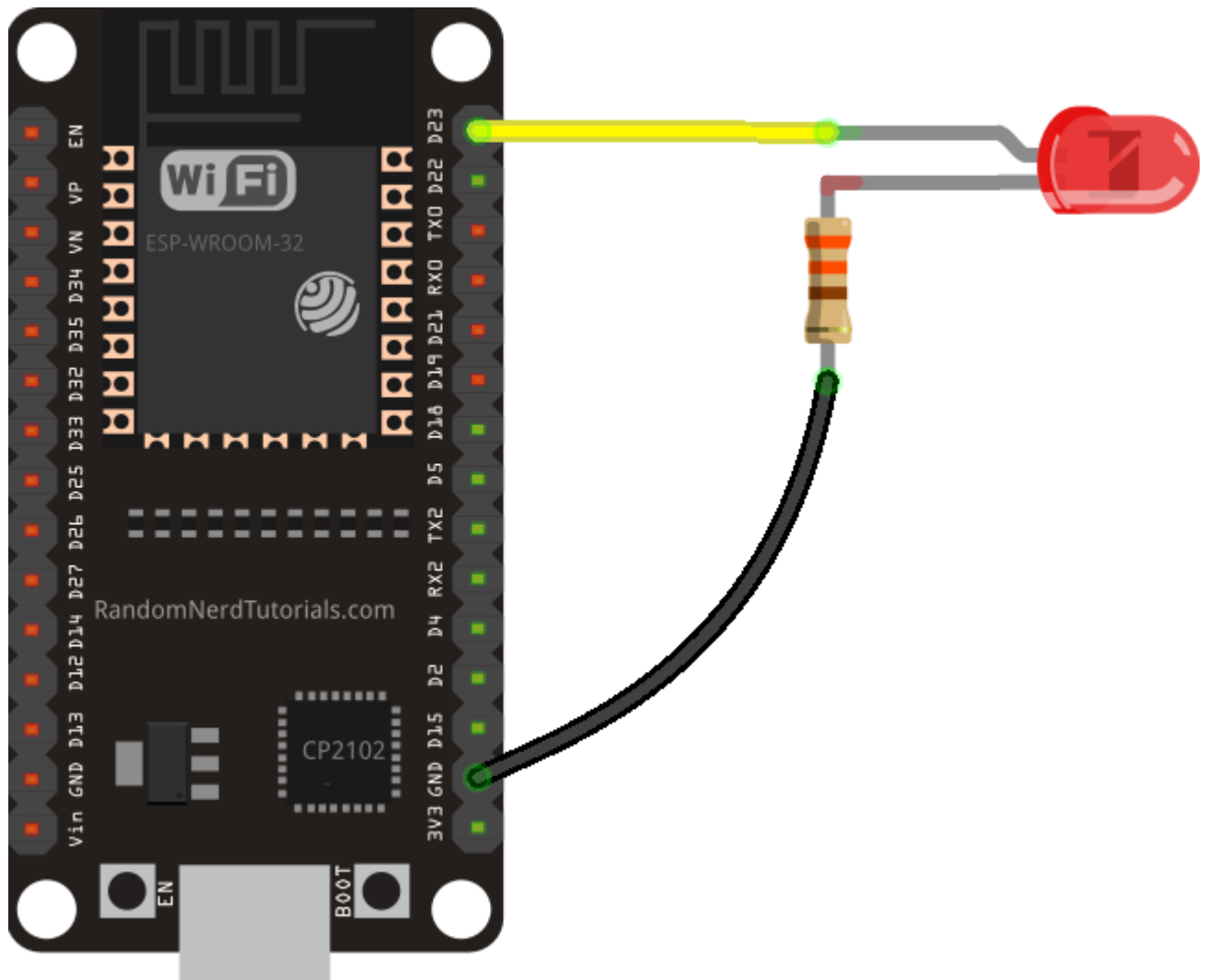
Important: always check the pinout for your specific board, before building any circuit.

Schematic

Here's a list of parts you need to assemble the circuit:

- [ESP32 DOIT DEVKIT V1 Board](#)
- [5mm LED](#)
- [330 Ohm resistor](#)
- [Jumper wires](#)
- [Breadboard](#) (optional)

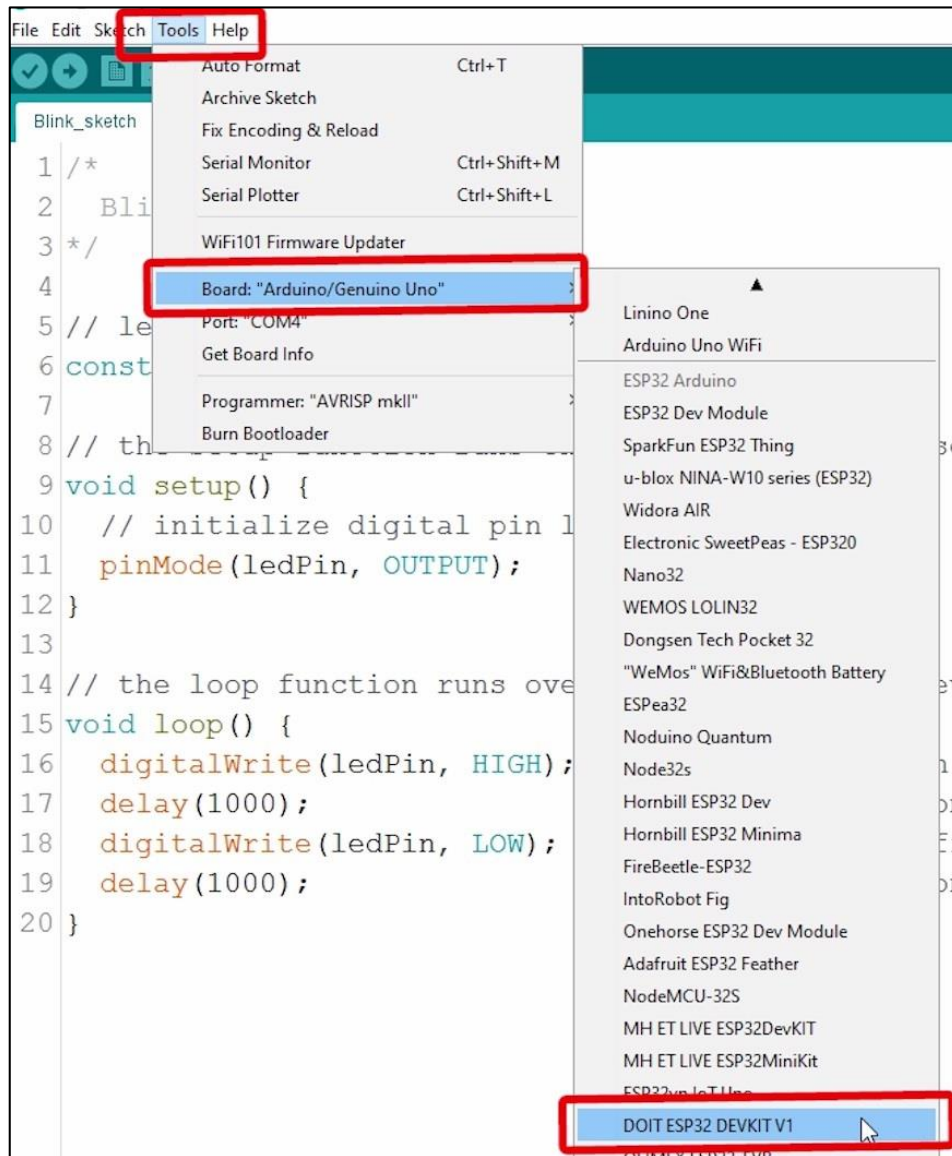
Follow the next schematic to wire the LED to the ESP32 DEVKit DOIT board (also check where GND is located in your board).



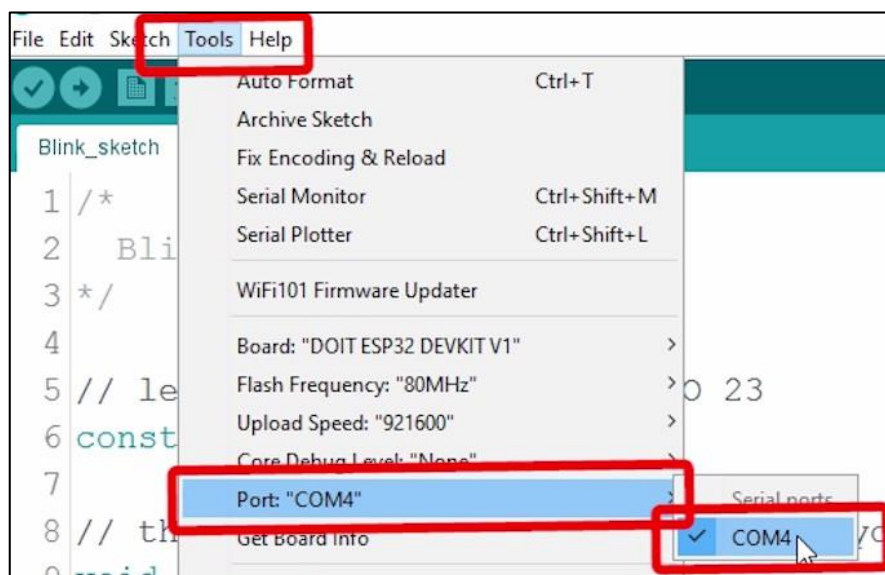
(This schematic uses the ESP32 DEVKIT V1 module version with 30 GPIOs – if you're using another model, please check the pinout for the board you're using.)

Preparing the Arduino IDE

After connecting an LED to the ESP32 board GPIO 23, you need to go to **Tools** ► **Board**, scroll down to the ESP32 section and select the name of your ESP32 board that you found earlier. In my case, it's the **DOIT ESP32 DEVKIT V1** board.



While having the ESP32 plugged to your computer. Go to **Tools** ► **Port** and select a COM port available.

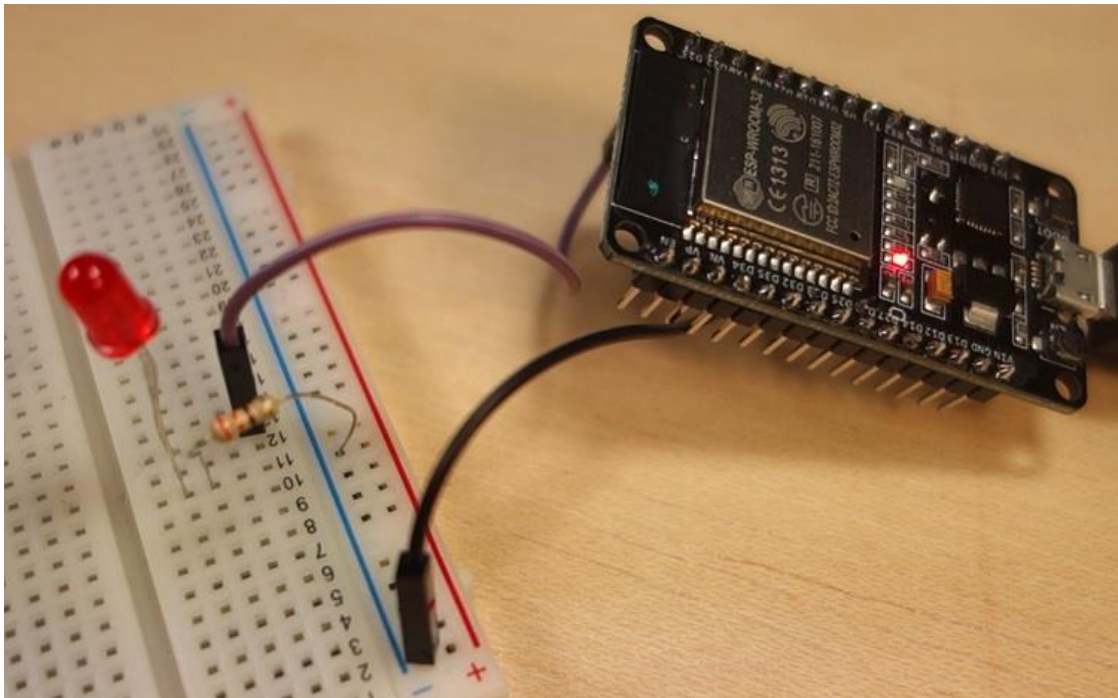


Uploading the Sketch

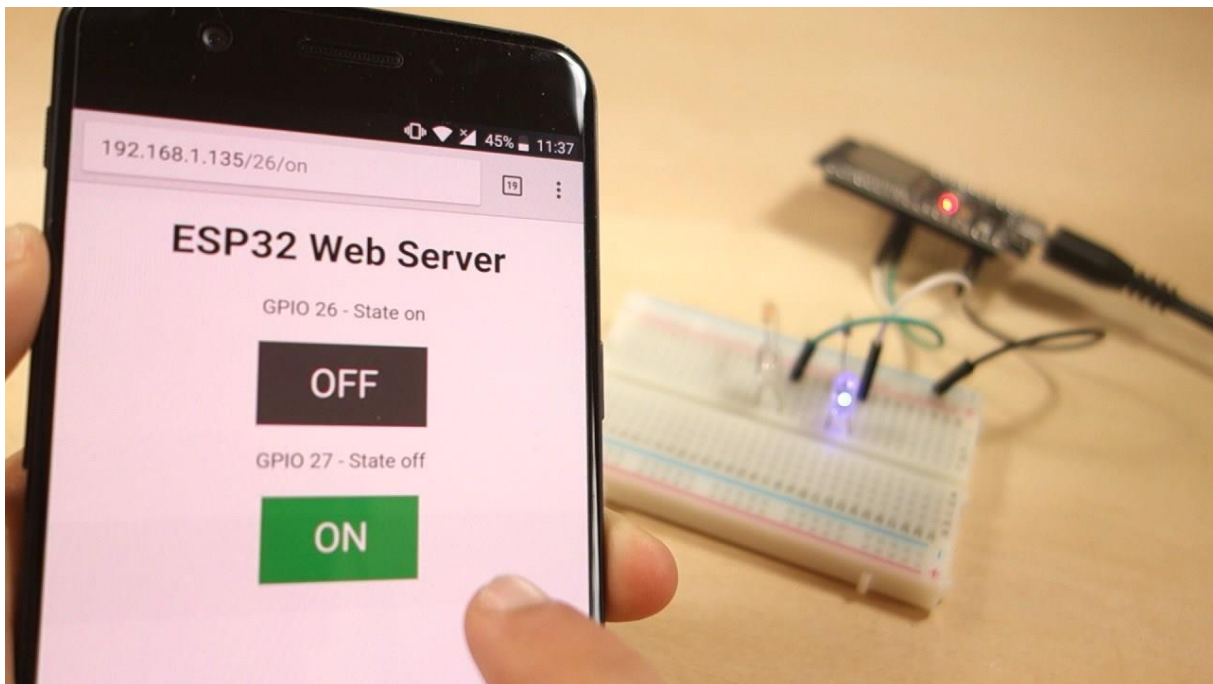
Go back to the Arduino IDE, press the Arduino IDE upload button and wait a few seconds while it compiles and uploads your sketch.



The LED attached to GPIO 23 should be blinking every other second.



ESP32 Web Server



Before going straight to building the web server, it is important to outline what our web server will do, so that it is easier to follow the steps later on.

- The web server you'll build controls two LEDs connected to the ESP32 GPIOs 26, and 27.
- You can access the ESP32 web server by typing the ESP32 IP address on a browser in the local network.
- By clicking the buttons on your web server you can instantly change the state of each LED.

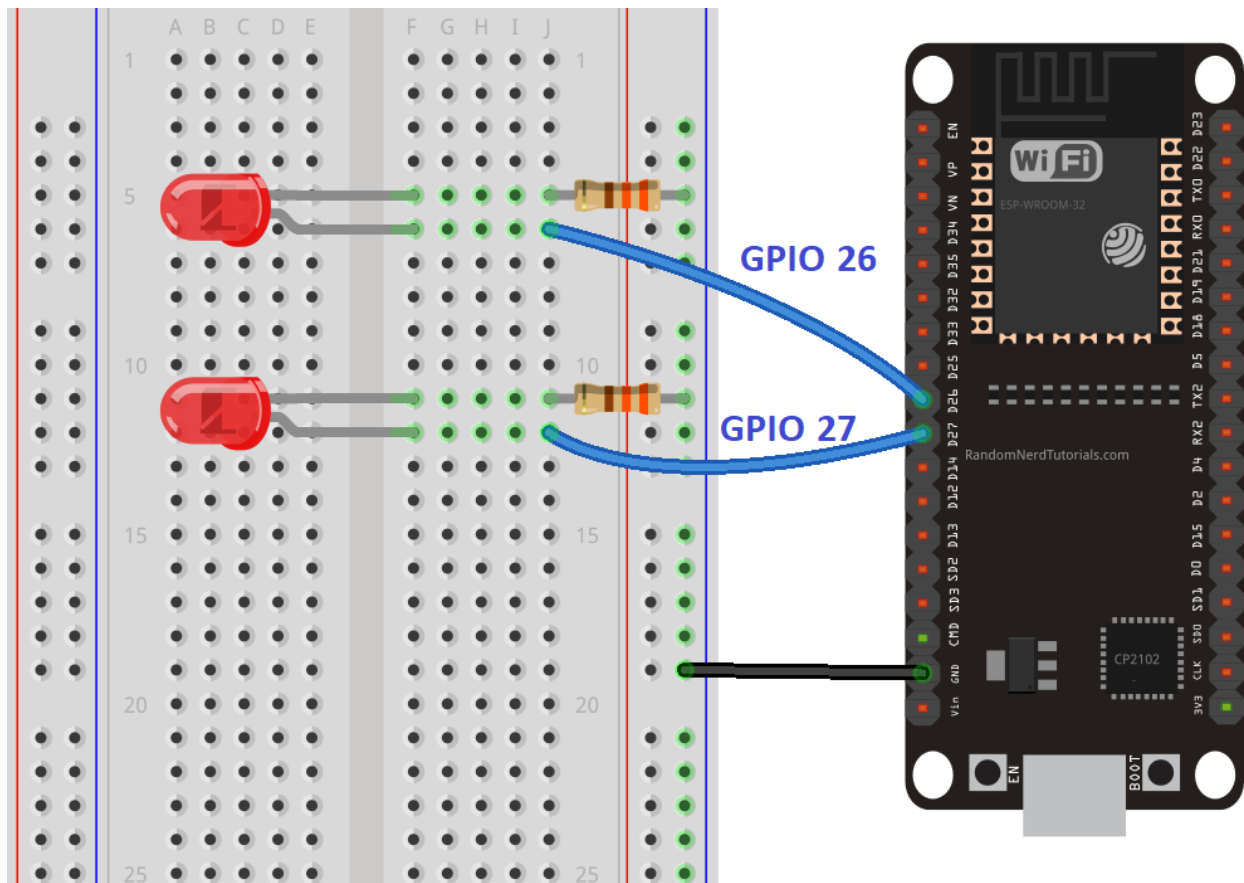
This is just a simple example to illustrate how to build a web server that controls outputs, the idea is to replace those LEDs with a [relay](#), or any other electronic components you want.

Schematic

Start by building the circuit. Connect two LEDs to your ESP32 as shown in the following schematic diagram – with one LED connected to GPIO 26, and another to GPIO 27.

Here's a list of parts you need to assemble the circuit:

- [ESP32 DOIT DEVKIT V1 Board](#)
- [2x 5mm LED](#)
- [2x 330 Ohm resistor](#)
- [Breadboard](#)
- [Jumper wires](#)



(This schematic uses the ESP32 DEVKIT V1 module version with 36 GPIOs – if you're using another model, please check the pinout for the board you're using.)

Building the Web Server

After wiring the circuit, the next step is uploading the code to your ESP32. Copy the code below to your Arduino IDE, but don't upload it yet. You need to make some changes to make it work for you.

SOURCE CODE

https://github.com/RuiSantosdotme/ESP32-Course/blob/master/code/WiFi_Web_Server_Outputs/WiFi_Web_Server_Outputs.ino

```

/*****
  Rui Santos
  Complete project details at http://randomnerdtutorials.com
*****/

// Load Wi-Fi library
#include <WiFi.h>

// Replace with your network credentials
const char* ssid    = "";
const char* password = "";

// Set web server port number to 80
WiFiServer server(80);

// Variable to store the HTTP request
String header;

```

```

// Auxiliar variables to store the current output state
String output26State = "off";
String output27State = "off";

// Assign output variables to GPIO pins
const int output26 = 26;
const int output27 = 27;

void setup() {
    Serial.begin(115200);
    // Initialize the output variables as outputs
    pinMode(output26, OUTPUT);
    pinMode(output27, OUTPUT);
    // Set outputs to LOW
    digitalWrite(output26, LOW);
    digitalWrite(output27, LOW);

    // Connect to Wi-Fi network with SSID and password
    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    // Print local IP address and start web server
    Serial.println("");
    Serial.println("WiFi connected.");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
    server.begin();
}

void loop() {
    WiFiClient client = server.available(); // Listen for incoming clients

    if (client) { // If a new client connects,
        Serial.println("New Client."); // print a message out in the
serial port
        String currentLine = ""; // make a String to hold
incoming data from the client
        while (client.connected()) { // loop while the client's
connected
            if (client.available()) { // if there's bytes to read
from the client,
                char c = client.read(); // read a byte, then
                Serial.write(c); // print it out the serial
monitor
                header += c;
                if (c == '\n') { // if the byte is a newline
character
                    // if the current line is blank, you got two newline characters
in a row.
                    // that's the end of the client HTTP request, so send a response:
                    if (currentLine.length() == 0) {
                        // HTTP headers always start with a response code (e.g.
HTTP/1.1 200 OK)
                        // and a content-type so the client knows what's coming, then a
blank line:
                        client.println("HTTP/1.1 200 OK");
                        client.println("Content-type:text/html");
                        client.println("Connection: close");
                        client.println();

                        // turns the GPIOs on and off

```

```

    if (header.indexOf("GET /26/on") >= 0) {
        Serial.println("GPIO 26 on");
        output26State = "on";
        digitalWrite(output26, HIGH);
    } else if (header.indexOf("GET /26/off") >= 0) {
        Serial.println("GPIO 26 off");
        output26State = "off";
        digitalWrite(output26, LOW);
    } else if (header.indexOf("GET /27/on") >= 0) {
        Serial.println("GPIO 27 on");
        output27State = "on";
        digitalWrite(output27, HIGH);
    } else if (header.indexOf("GET /27/off") >= 0) {
        Serial.println("GPIO 27 off");
        output27State = "off";
        digitalWrite(output27, LOW);
    }

    // Display the HTML web page
    client.println("<!DOCTYPE html><html>");
    client.println("<head><meta name=\"viewport\"");
content="\width=device-width, initial-scale=1\">");
    client.println("<link rel=\"icon\" href=\"data:,\>");
    // CSS to style the on/off buttons
    // Feel free to change the background-color and font-size
attributes to fit your preferences
    client.println("<style>html { font-family: Helvetica; display:
inline-block; margin: 0px auto; text-align: center;}");
    client.println(".button { background-color: #4CAF50; border:
none; color: white; padding: 16px 40px;");
    client.println("text-decoration: none; font-size: 30px; margin:
2px; cursor: pointer;}");
    client.println(".button2 {background-color:
#555555;}</style></head>");

    // Web Page Heading
    client.println("<body><h1>ESP32 Web Server</h1>");

    // Display current state, and ON/OFF buttons for GPIO 26
    client.println("<p>GPIO 26 - State " + output26State + "</p>");
    // If the output26State is off, it displays the ON
button
    if (output26State=="off") {
        client.println("<p><a href=\"/26/on\"><button
class=\"button\">ON</button></a></p>");
    } else {
        client.println("<p><a href=\"/26/off\"><button class=\"button
button2\">OFF</button></a></p>");
    }

    // Display current state, and ON/OFF buttons for GPIO 27
    client.println("<p>GPIO 27 - State " + output27State + "</p>");
    // If the output27State is off, it displays the ON
button
    if (output27State=="off") {
        client.println("<p><a href=\"/27/on\"><button
class=\"button\">ON</button></a></p>");
    } else {
        client.println("<p><a href=\"/27/off\"><button class=\"button
button2\">OFF</button></a></p>");
    }
    client.println("</body></html>");

    // The HTTP response ends with another blank line
    client.println();
    // Break out of the while loop

```



```

        break;
    } else { // if you got a newline, then clear currentLine
        currentLine = "";
    }
    } else if (c != '\r') { // if you got anything else but a carriage
return character,
        currentLine += c;        // add it to the end of the currentLine
    }
}
}
// Clear the header variable
header = "";
// Close the connection
client.stop();
Serial.println("Client disconnected.");
Serial.println("");
}
}

```

Setting Your Network Credentials

You need to modify the following lines with your network credentials: SSID and password. The code is well commented on where you should make the changes.

```

// Replace with your network credentials
const char* ssid      = "";
const char* password = "";

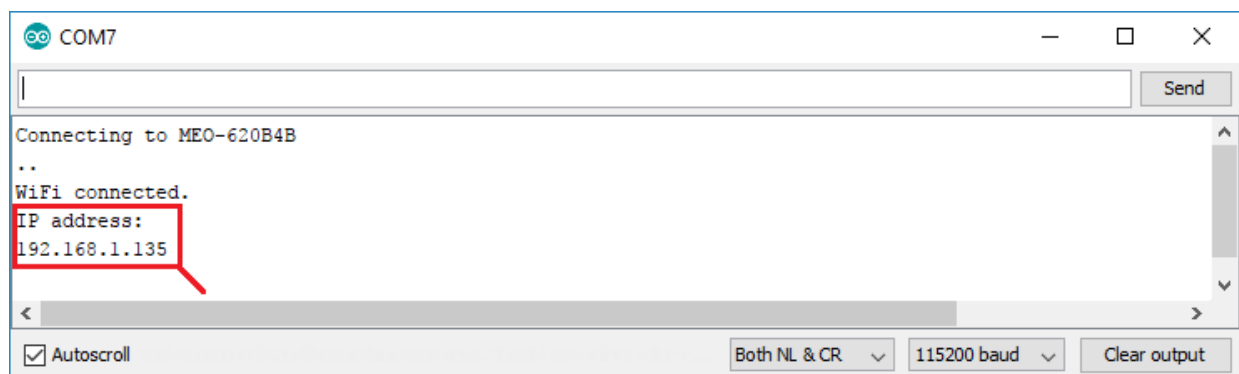
```

Finding the ESP32 IP Address

Now, you can upload the code, and it will work straight away. Don't forget to check if you have the right board and COM port selected, otherwise you'll get an error when trying to upload. Open the Serial Monitor at a baud rate of 115200.



The ESP32 connects to Wi-Fi, and outputs the ESP IP address on the Serial Monitor. Copy that IP address, because you need it to access the ESP32 web server.



Note: if nothing shows up in the Serial Monitor, press the ESP32 "EN" button (enable button next to the micro USB port).

Accessing the Web Server

Open your browser, paste the ESP32 IP address, and you'll see the following page.



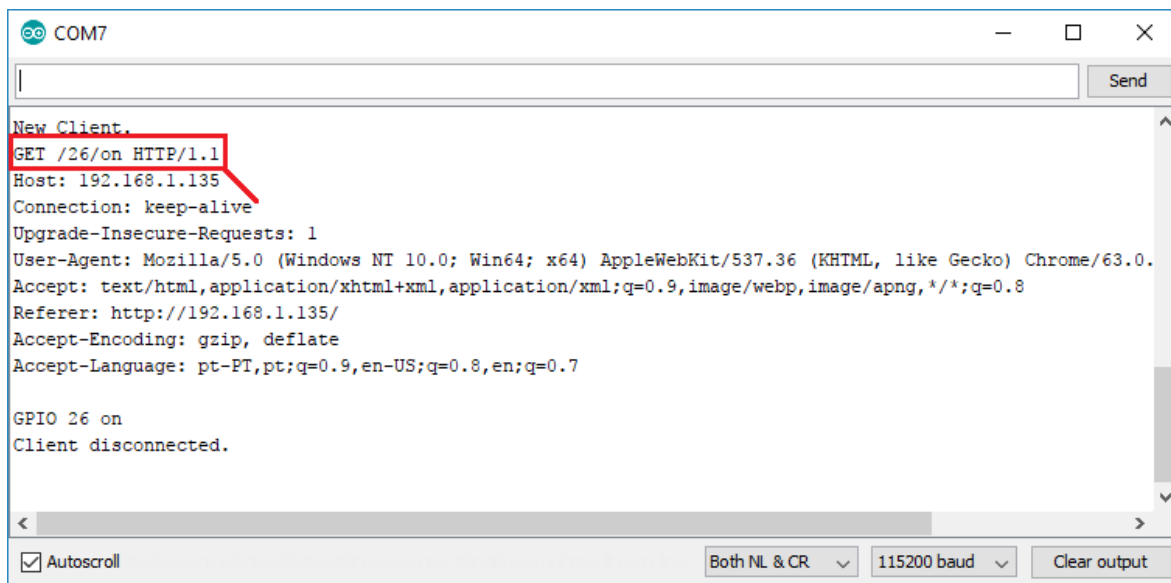
If you take a look at the Serial Monitor, you can see what's going on on the background. The ESP32 receives an HTTP request from a new client (in this case, your browser).

You can also see other information about the HTTP request, the HTTP header fields that define the operating parameters of an HTTP transaction.



Testing the Web Server

Let's test the web server. Click the button to turn GPIO 26 ON. You can see on the Serial Monitor that the ESP32 receives a request on the **/26/on** URL.



The image shows a Serial Monitor window titled 'COM7'. It displays an incoming HTTP GET request from a new client. The request details are as follows:

```
New Client.  
GET /26/on HTTP/1.1  
Host: 192.168.1.135  
Connection: keep-alive  
Upgrade-Insecure-Requests: 1  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8  
Referer: http://192.168.1.135/  
Accept-Encoding: gzip, deflate  
Accept-Language: pt-PT,pt;q=0.9,en-US;q=0.8,en;q=0.7  
  
GPIO 26 on  
Client disconnected.
```

At the bottom of the window, there are controls for 'Autoscroll' (checked), 'Both NL & CR' (selected), '115200 baud' (selected), and a 'Clear output' button.

When the ESP receives that request, it turns the LED attached to GPIO 26 ON, and its state is also updated on the web page.



Test the button for GPIO 27 and see that it works similarly.

How the code Works

Now, let's take a closer look at the code to see how it works, so that you are able to modify it to fulfill your needs.

The first thing you need to do is to include the WiFi library. This is the same library used to create a web server with the Arduino using the Ethernet shield.

```
// Load Wi-Fi library  
#include <WiFi.h>
```

As mentioned previously, you need to insert your ssid and password in the following lines inside the double quotes.

```
const char* ssid      = "";  
const char* password = "";
```

Then, you set your web server to port 80.

```
WiFiServer server(80);
```

The following line creates a variable to store the header of the HTTP request:

```
String header;
```

Next, you create auxiliary variables to store the current state of your outputs. If you want to add more outputs and save its state, you need to create more variables.

```
// Auxiliar variables to store the current output state  
String output26State = "off";  
String output27State = "off";
```

You also need to assign a GPIO to each of your outputs. Here we are using GPIO 26 and GPIO 27. You can use any other suitable GPIOs.

```
// Assign output variables to GPIO pins  
const int output26 = 26;  
const int output27 = 27;
```

setup()

Now, let's go into the **setup()**. The **setup()** function only runs once when your ESP first boots.

First, we start a serial communication at a baud rate of 115200 for debugging purposes.

```
Serial.begin(115200);
```

You also define your GPIOs as OUTPUTs and set them to LOW.

```
// Initialize the output variables as outputs  
pinMode(output26, OUTPUT);  
pinMode(output27, OUTPUT);  
// Set outputs to LOW  
digitalWrite(output26, LOW);  
digitalWrite(output27, LOW);
```

The following lines begin the Wi-Fi connection with **WiFi.begin(ssid, password)**, wait for a successful connection and print the ESP IP address in the Serial Monitor.

```
Serial.print("Connecting to ");  
Serial.println(ssid);  
WiFi.begin(ssid, password);  
while (WiFi.status() != WL_CONNECTED) {  
    delay(500);
```



```

    Serial.print(".");
}
// Print local IP address and start web server
Serial.println("");
Serial.println("WiFi connected.");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
server.begin();

```

loop()

In the `loop()` we program what happens when a new client establishes a connection with the web server.

The ESP is always listening for incoming clients with this line:

```
WiFiClient client = server.available();
```

When a request is received from a client, we'll save the incoming data. The while loop that follows will be running as long as the client stays connected. We don't recommend changing the following part of the code unless you know exactly what you are doing.

```

if (client) { // If a new client connects,
    Serial.println("New Client."); // print a message out in the serial port
    String currentLine = ""; // make a String to hold incoming data
    while (client.connected()) { // loop while the client's connected
        if (client.available()) { // if there's bytes to read from client
            char c = client.read(); // read a byte, then
            Serial.write(c); // print it out the serial monitor
            header += c;
            if (c == '\n') { // if the byte is a newline character
                // if line is blank, you got two newline characters in a row.
                // that's the end of the client HTTP request, so send a response:
                if (currentLine.length() == 0) {
                    // HTTP headers start with a response code (e.g. HTTP/1.1 200 OK)
                    // and a content-type so the client knows what's coming
                    client.println("HTTP/1.1 200 OK");
                    client.println("Content-type:text/html");
                    client.println("Connection: close");
                    client.println();

```

The next section of `if` and `else` statements checks which button was pressed in your web page, and controls the outputs accordingly. As we've seen previously, we make a request on different URLs depending on the button we press.

```

// turns the GPIOs on and off
if (header.indexOf("GET /26/on") >= 0) {
    Serial.println("GPIO 26 on");
    output26State = "on";
    digitalWrite(output26, HIGH);
} else if (header.indexOf("GET /26/off") >= 0) {
    Serial.println("GPIO 26 off");
    output26State = "off";
    digitalWrite(output26, LOW);
} else if (header.indexOf("GET /27/on") >= 0) {
    Serial.println("GPIO 27 on");
    output27State = "on";
    digitalWrite(output27, HIGH);
} else if (header.indexOf("GET /27/off") >= 0) {

```

```
Serial.println("GPIO 27 off");
output27State = "off";
digitalWrite(output27, LOW);
}
```

For example, if you've pressed the GPIO 26 ON button, the ESP receives a request on the **/26/ON** URL, and we receive that information on the HTTP header. So, we can check if the header contains the expression GET **/26/on**. If it contains, it will print a message on the Serial Monitor, it will change the output26statevariable to ON, and turns the LED on.

This works similarly for the other buttons. So, if you want to add more outputs, you should modify this part of the code to include them.

Displaying the HTML web page

The next thing you need to do, is creating the web page. The ESP32 will be sending a response to your browser with some HTML code to build the web page.

Note: in Unit 3 and Unit 4, you'll learn about HTML and CSS basics, so that you can easily modify the web page to fulfill your needs.

The web page is sent to the client using this expressing `client.println()`. You should enter what you want to send to the client as an argument.

The first thing we should send is always the following line that indicates that we are sending HTML.

```
<!DOCTYPE HTML><html>
```

Then, the following line makes the web page responsive in any web browser.

```
client.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">");
```

And the following is used to prevent requests on the favicon. – You don't need to worry about this line.

```
client.println("<link rel=\"icon\" href=\"data:,\">");
```

Styling the Web Page

Next, we have some CSS text to style the buttons and the web page appearance. We choose the Helvetica font, define the content to be displayed as a block and aligned at the center.

```
client.println("<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}");
```

We style our buttons with the #4CAF50 color, without border, text in white color, and with this padding: 16px 40px. We also set the text-decoration to none, define the font size, the margin, and the cursor to a pointer.

```
client.println(".button { background-color: #4CAF50; border: none; color: white; padding: 16px 40px;");
```

```
client.println("text-decoration: none; font-size: 30px; margin: 2px;
cursor: pointer;});
```

We also define the style for a second button, with all the properties of the button we've defined earlier, but with a different color. This will be the style for the off button.

```
client.println(".button2{background-color:#555555;}</style></head>");
```

Setting the Web Page First Heading

In the next line you can set the first heading of your web page. Here we have "ESP32 Web Server", but you can change this text to whatever you like.

```
// Web Page Heading
client.println("<body><h1>ESP32 Web Server</h1>");
```

Displaying the Buttons and Corresponding State

Then, you write a paragraph to display the GPIO 26 current state. As you can see we use the output26Statevariable, so that the state updates instantly when this variable changes.

```
client.println("<p>GPIO 26 - State " + output26State + "</p>");
```

Then, we display the on or the off button, depending on the current state of the GPIO. If the current state of the GPIO is off, we show the ON button, if not, we display the OFF button.

```
if (output27State=="off") {
  client.println("<p><a href=\"/27/on\"><button
class=\"button\">ON</button></a></p>");
} else {
  client.println("<p><a href=\"/27/off\"><button class=\"button
button2\">OFF</button></a></p>");
}
```

We use the same procedure for GPIO 27.

Closing the Connection

Finally, when the response ends, we clear the header variable, and stop the connection with the client with `client.stop()`.

```
// Clear the header variable
header = "";
// Close the connection
client.stop();
```

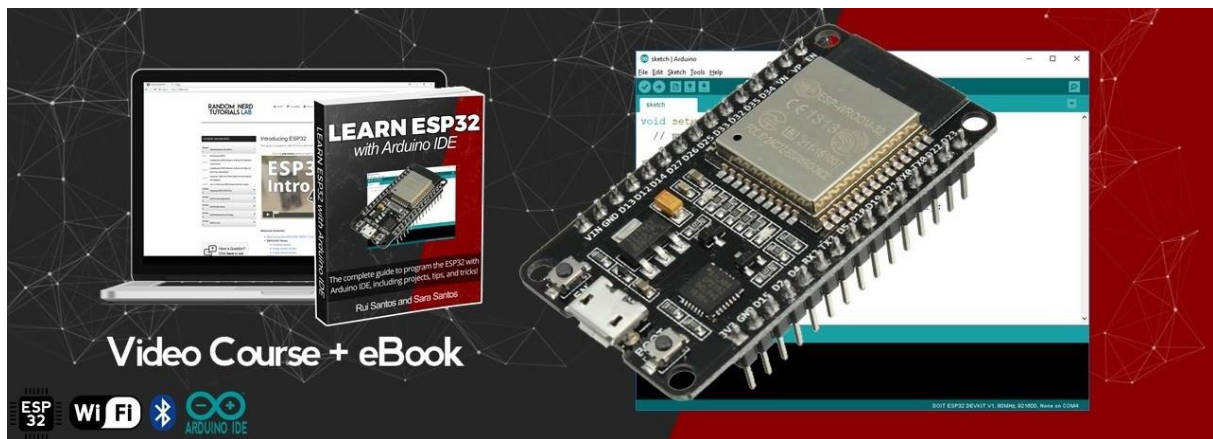
Wrapping Up

Now that you know how the code works, you can modify the code to add more outputs, or modify your web page. To modify your web page you may need to know some HTML and CSS basics.

Thanks for reading this mini project ebook. If you liked this ebook you'll surely like our "[Learn ESP32 with Arduino IDE](#)" course.

Learn ESP32 with Arduino IDE Course

[Learn ESP32 with Arduino IDE](#) is a practical course where you'll learn how to take the most out of the ESP32 using the Arduino IDE. This is our complete guide to program the ESP32 with Arduino IDE, including projects, tips, and tricks!



[GET ACCESS TO THE COURSE HERE »»](#)

What's inside the course?

The course contains 8 Modules to take the most out of the ESP32. We'll start by introducing the ESP32 main features and explore its GPIOs. We'll also cover a variety of subjects related with IoT like Web Servers, Bluetooth Low Energy (BLE), LoRa, and MQTT. Each subject contains practical examples with schematics and code.

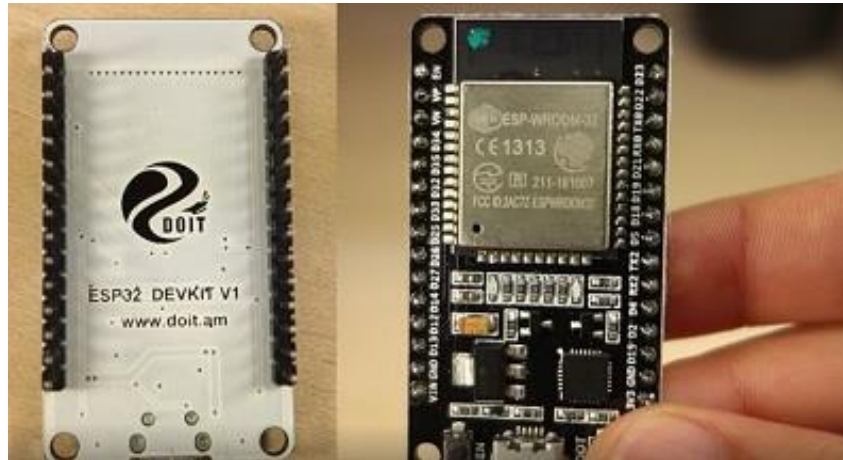
Here's what you'll have access with this course:

- Course dashboard with video, code, schematics, and transcripts
- All 8 Modules (downloadable eBook in PDF format with 510 pages)
- Module #8 includes 4 advanced ESP32 Projects
- Watch and Download the Video Course
- Source Code + Schematics
- Unlimited Updates
- Exclusive access to a private Forum to ask questions
- Exclusive access to our Facebook group community

Course Modules

This course contains 7 Modules and 4 Projects. Scroll down to take a look at the Modules and Projects covered in the course.

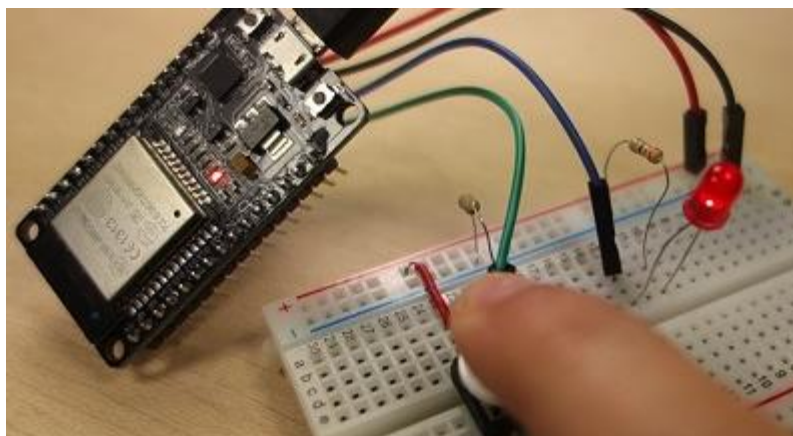
Module #1: Getting Started with ESP32



This first Module is an introduction to the ESP32 board. We'll explore its features, and show you how to use your board with this course. You'll also prepare your Arduino IDE to upload code to the ESP32.

- Unit 1: Introducing ESP32
- Unit 2: Installing the ESP32 Board in Arduino IDE (Windows, Mac OS X, and Linux)
- Unit 3: How To Use Your ESP32 Board with this Course
- Unit 4: Make the ESP32 Breadboard Friendly

Module #2: Exploring the ESP32 GPIO Pins



In this Module we'll explore the ESP32 GPIO functions. We'll show you how to control digital outputs, create PWM signals, and read digital and analog inputs. We'll also take a look at the ESP32 touch capacitive pins and the built-in hall effect sensor.

- Unit 1: ESP32 Digital Inputs and Outputs
- Unit 2: ESP32 Touch Sensor
- Unit 3: ESP32 Pulse-Width Modulation (PWM)
- Unit 4: ESP32 Reading Analog Inputs
- Unit 5: ESP32 Hall Effect Sensor
- Unit 6: ESP32 with PIR Motion Sensor - Interrupts and Timers
- Unit 7: ESP32 Flash Memory - Store Permanent Data (Write and Read)
- Unit 8: Other ESP32 Sketch Examples

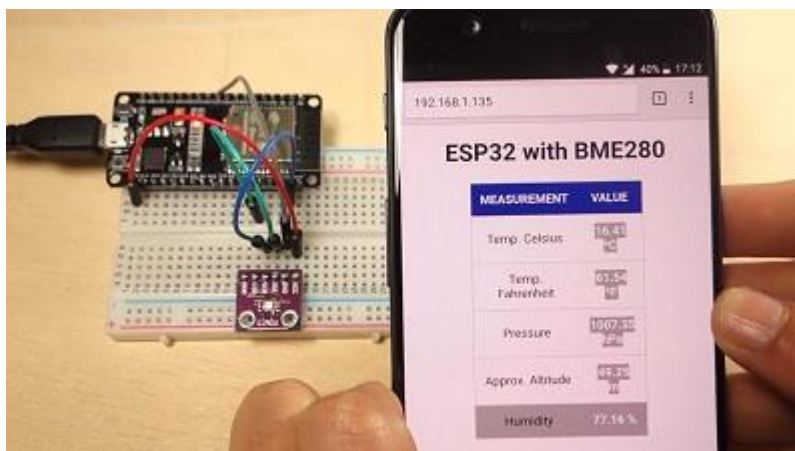
Module #3: ESP32 Deep Sleep Mode



Using deep sleep in your ESP32 is a great way to save power in battery-powered applications. In this Module we'll show you how to put your ESP32 into deep sleep mode and the different ways to wake it up. Units in this Module:

- Unit 1: ESP32 Deep Sleep Mode
- Unit 2: Deep Sleep - Timer Wake Up
- Unit 3: Deep Sleep - Touch Wake Up
- Unit 4: Deep Sleep - External Wake Up

Module #4: Building Web Servers with the ESP32



This Module explains how to build several web servers with the ESP32. After explaining some theoretical concepts, you'll learn how to build a web server to display sensor readings, to control outputs, and much more. You'll also learn how you can edit your web server interface using HTML and CSS.

- Unit 1: ESP32 Web Server - Introduction
- Unit 2: ESP32 Web Server - Control Outputs
- Unit 3: ESP32 Web Server - HTML and CSS Basics (Part 1/2)
- Unit 4: ESP32 Web Server - HTML in Arduino IDE (Part 2/2)
- Unit 5: ESP32 Web Server – Control Outputs (Relay)
- Unit 6: Making Your ESP32 Web Server Password Protected
- Unit 7: Accessing the ESP32 Web Server From Anywhere
- Unit 8: ESP32 Web Server – Display Sensor Readings
- Unit 9: ESP32 Control Servo Motor Remotely (Web Server)

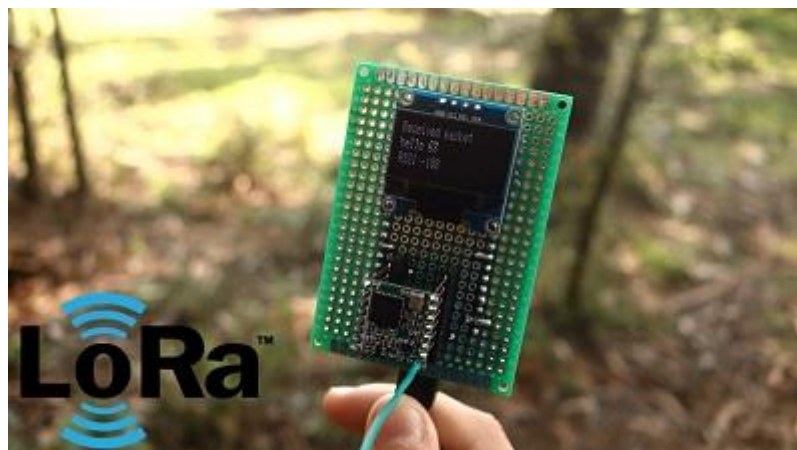
Module #5: ESP32 Bluetooth Low Energy



The ESP32 comes not only with Wi-Fi, but it also has Bluetooth and Bluetooth Low Energy built-in. Learn how to use the ESP32 Bluetooth functionalities to scan nearby devices and exchange information (BLE client and server). Units in this Module:

- Unit 1: ESP32 Bluetooth Low Energy (BLE) - Introduction
- Unit 2: Bluetooth Low Energy - Notify and Scan
- Unit 3: ESP32 BLE Server and Client (Part 1/2)
- Unit 4: ESP32 BLE Server and Client (Part 2/2)

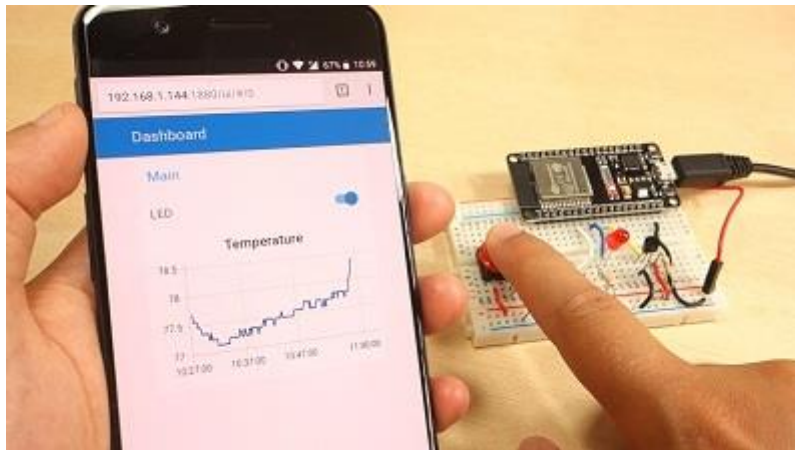
Module #6: LoRa Technology with the ESP32



LoRa is a long range wireless technology. In this Module you'll explore what's LoRa and how you can use it with the ESP32 to extend the communication range between IoT devices. Units in this Module:

- Unit 1: ESP32 with LoRa - Introduction
- Unit 2: ESP32 LoRa Sender and Receiver
- Unit 3: Further Reading about LoRa Gateways
- Unit 4: LoRa - Where to Go Next?

Module #7: ESP32 with MQTT



MQTT stands for Message Queuing Telemetry Transport. It is a lightweight publish and subscribe system perfect for Internet of Things applications. In this module you'll learn how to use MQTT to establish a communication between two ESP32 boards, and how you can control the ESP32 using Node-RED. Units in this Module:

- Unit 1: ESP32 with MQTT – Introduction
- Unit 2: Installing Mosquitto MQTT Broker on a Raspberry Pi
- Unit 3: MQTT Project – MQTT Client ESP32 #1
- Unit 4: MQTT Project – MQTT Client ESP32 #2
- Unit 5: Installing Node-RED and Node-RED Dashboard on a Raspberry Pi
- Unit 6: Connect ESP32 to Node-RED using MQTT

Project #1: ESP32 Wi-Fi Multisensor – Temperature, Humidity, Motion, Luminosity, and Relay Control



In this project you'll build an ESP32 Wi-Fi Multisensor. This device consists of a PIR motion sensor, a light dependent resistor (LDR), a DHT22 temperature and humidity sensor, a relay, and a status RGB LED. You'll also build a web server that allows you to control the ESP32 multisensor using different modes. Units in this project:

- Unit 1: ESP32 Wi-Fi Multisensor - Temperature, Humidity, Motion, Luminosity, and Relay Control
- Unit 2: ESP32 Wi-Fi Multisensor - How the Code Works?

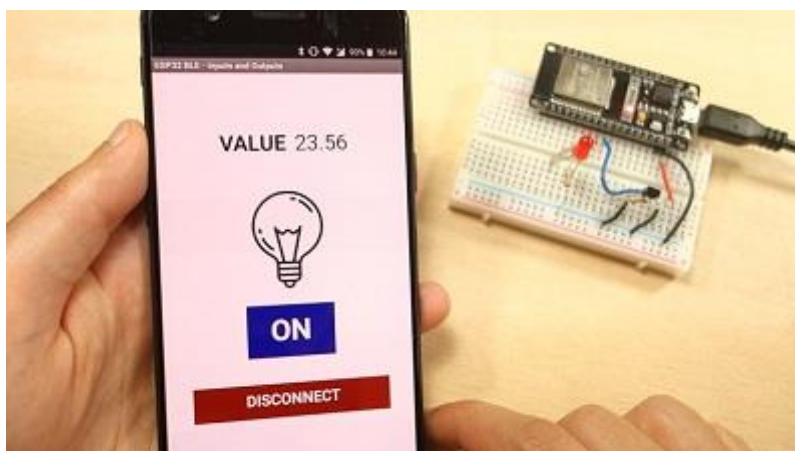
Project #2: Remote Controlled Wi-Fi Car Robot



In this project we'll show you step by step how to create an ESP32 Wi-Fi remote controlled car robot. Units in this project:

- Unit 1: Remote Controlled Wi-Fi Car Robot - Part 1/2
- Unit 2: Remote Controlled Wi-Fi Car Robot - Part 2/2
- Unit 3: Assembling the Smart Robot Car Chassis Kit
- Unit 4: Extra - Access Point (AP) For Wi-Fi Car Robot

Project #3: Bluetooth Low Energy (BLE) Android Application with MIT App Inventor – Control Outputs and Display Sensor Readings



In this project you're going to create an Android application to interact with the ESP32 using Bluetooth Low Energy (BLE). Units in this project:

- Unit 1: ESP32 BLE Android Application – Control Outputs and Display Sensor Readings
- Unit 2: Bluetooth Low Energy (BLE) Android Application with MIT App Inventor 2 – How the App Works?

Project #4: LoRa Long Range Sensor Monitoring – Reporting Sensor Readings from Outside: Soil Moisture and Temperature



In this project you're going to build an off-the-grid monitoring system that sends soil moisture and temperature readings to an indoor receiver. To establish a communication between the sender and the receiver we'll be using LoRa communication protocol. Units in this project:

- Unit 1: LoRa Long Range Sensor Monitoring and Data Logging
- Unit 2: ESP32 LoRa Sender
- Unit 3: ESP32 LoRa Receiver
- Unit 4: ESP32 LoRa Sender Solar Powered
- Unit 5: Final Tests, Demonstration, and Data Analysis

Get Access to “ [Learn ESP32 with Arduino IDE](https://randomnerdtutorials.com/learn-esp32-with-arduino-ide) ”
randomnerdtutorials.com/learn-esp32-with-arduino-ide