

Blinking an LED with Arduino IDE

In this section, you're going to design a simple circuit to blink an LED with the ESP8266 using Arduino IDE.

Why do we always blink an LED first? That's a great question! If you can blink an LED you can pretty much say that you can turn any electronic device on or off.

Writing Your Arduino Sketch

The sketch for blinking an LED is very simple. You can find it in the link below:

SOURCE CODE

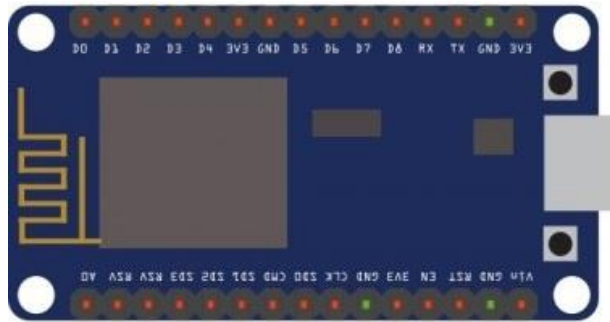
```
int pin = 2;

void setup() {
  // initialize GPIO 2 as an output
  pinMode(pin, OUTPUT);
}

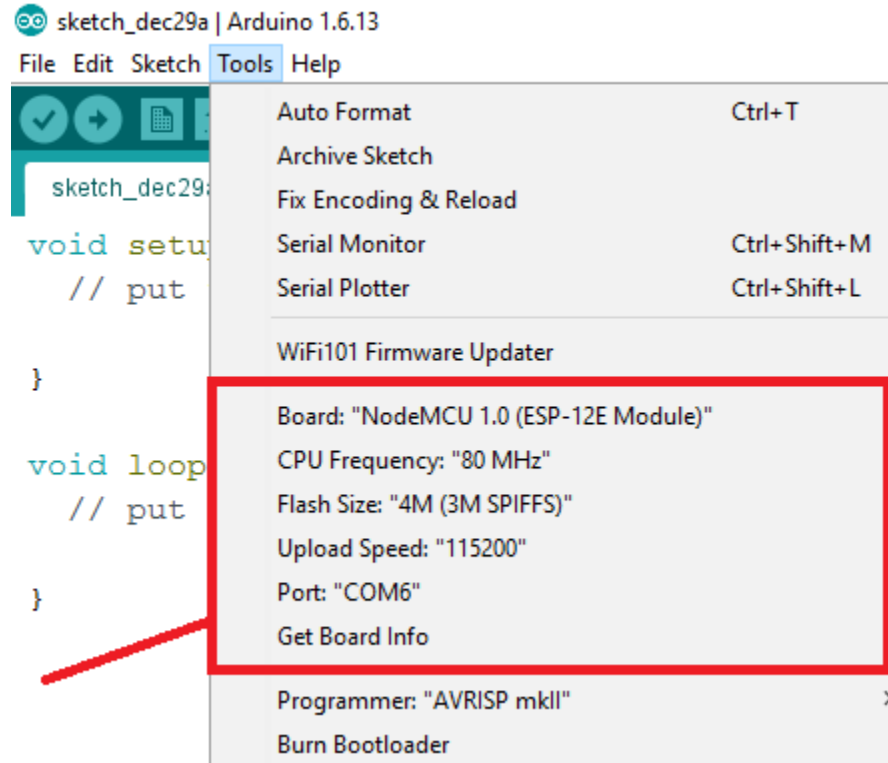
// the loop function runs over and over again forever
void loop() {
  digitalWrite(pin, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);             // wait for a second
  digitalWrite(pin, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);             // wait for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

Uploading Code to ESP8266

Upload code to your ESP-12E NodeMCU Kit is very simple, since it has built-in programmer. You plug your board to your computer and you don't need to make any additional connections.

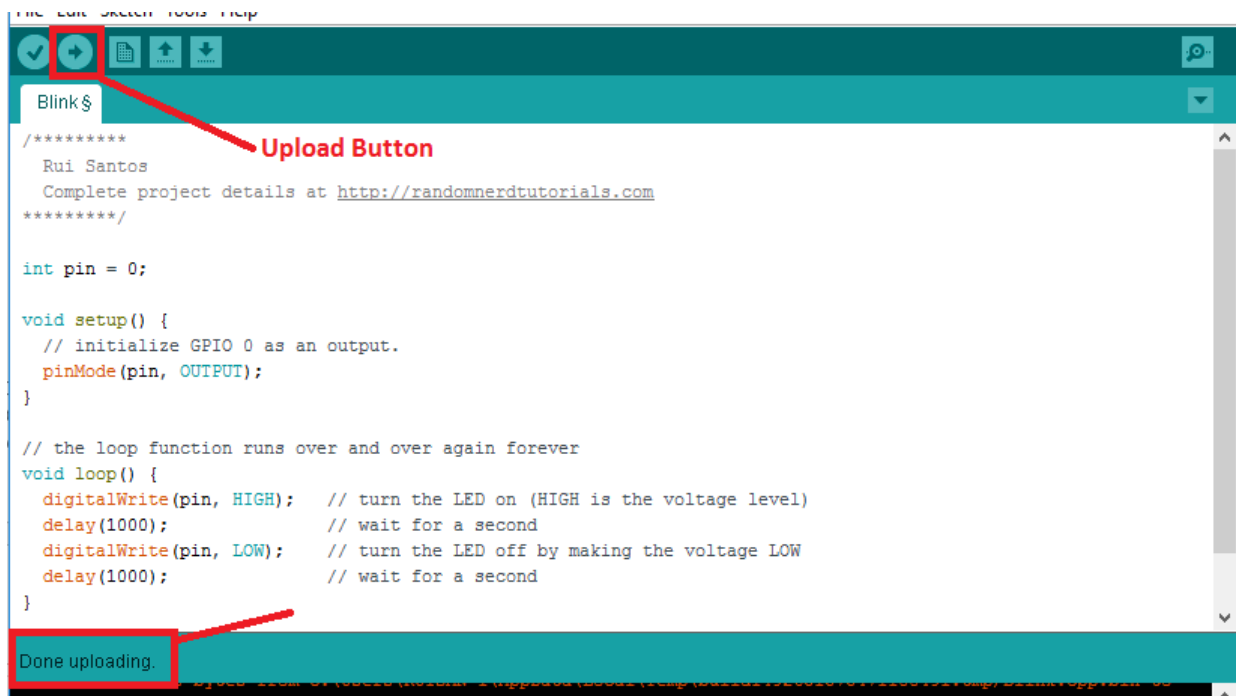


Look at the **Tools** menu, select Board “**NodeMCU 1.0 (ESP-12E Module)**” and all the configurations, by default, should look like this:



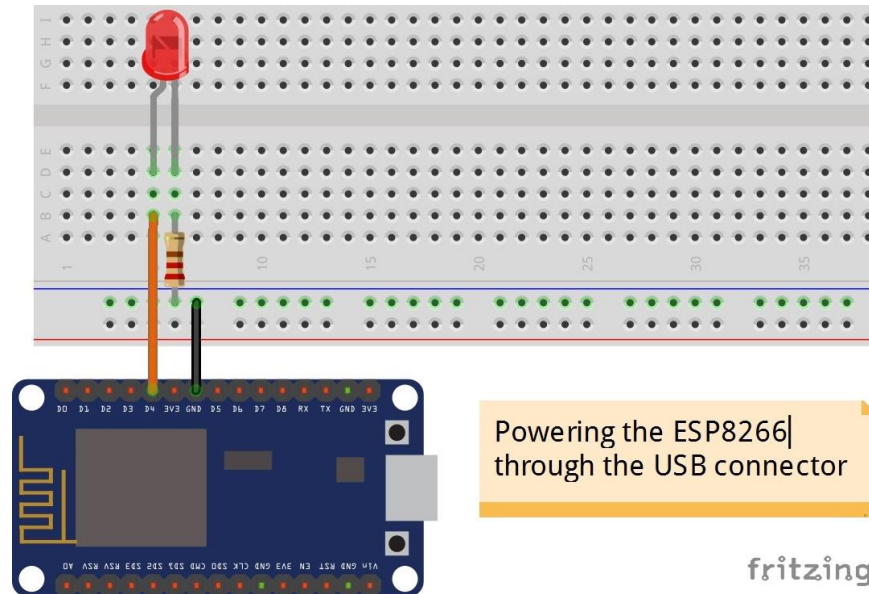
Important: your COM port is very likely to be different from the preceding screenshot (Port: “COM6”). That’s fine, because it doesn’t interfere with anything. On the other hand, all the other configurations should look exactly like mine.

After checking the configurations, click the **“Upload Button”** in the Arduino IDE and wait a few seconds until you see the message **“Done uploading.”** in the bottom left corner.

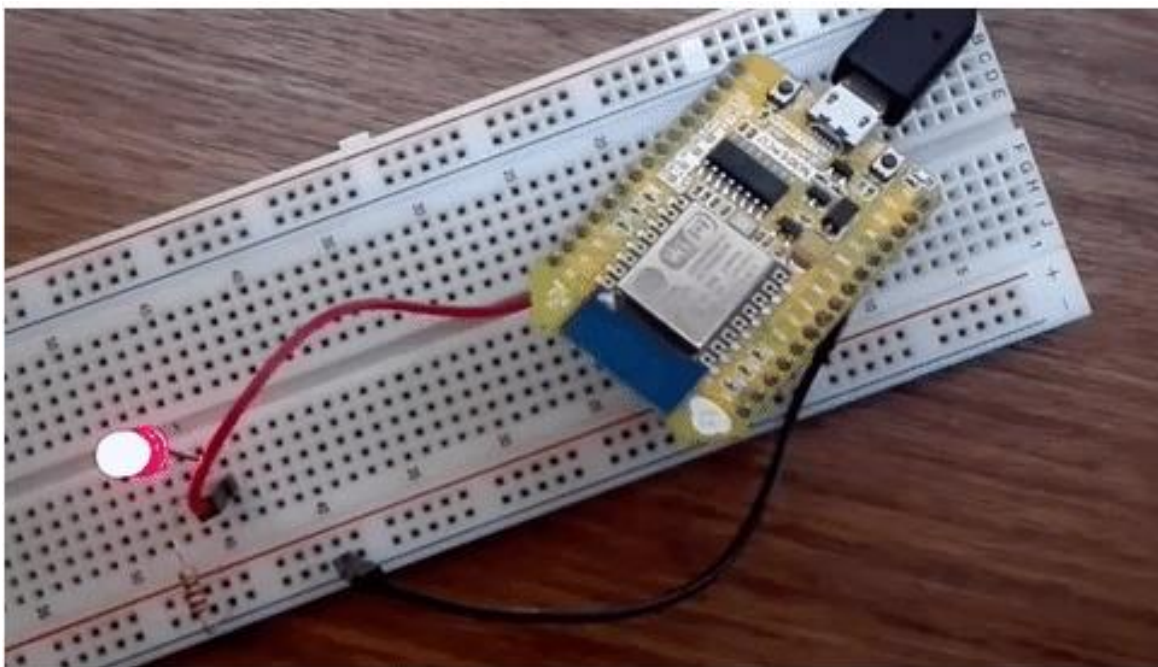


Final ESP-12E circuit

Connect an LED and a 220 Ohm resistor to your ESP8266 D4 (GPIO 2).

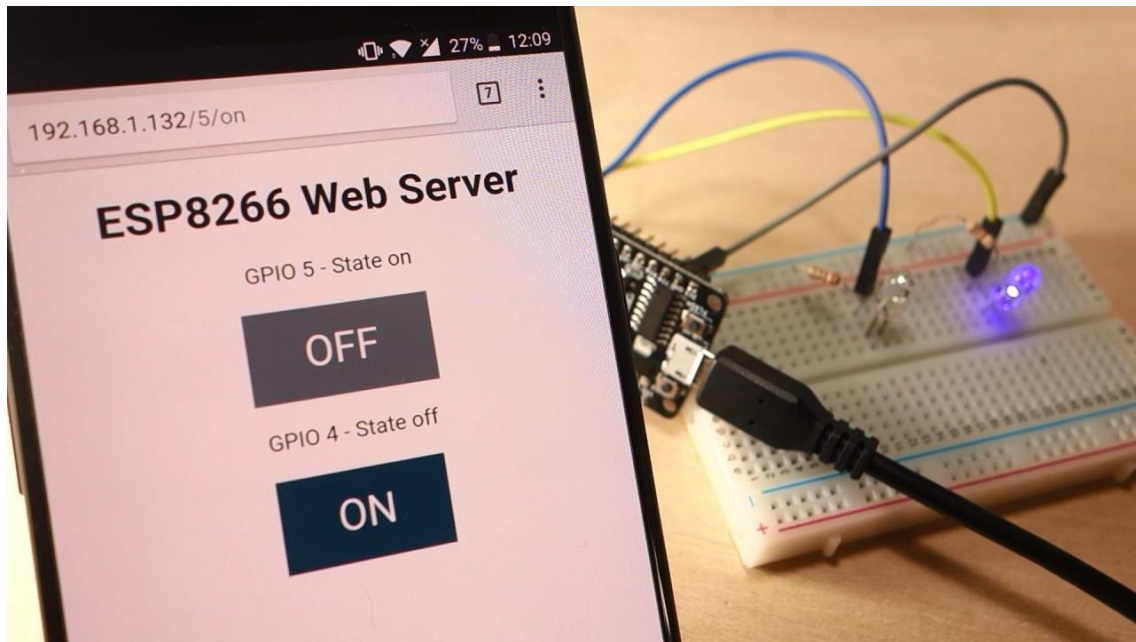


Restart your ESP8266. Congratulations, you've made it! Your LED should be blinking every 1 second!



ESP8266 Web Server

This tutorial is a step-by-step guide that shows you how to build a standalone ESP8266 Web Server that controls two outputs (two LEDs). You can then replace those LEDs with any other electronics appliances.



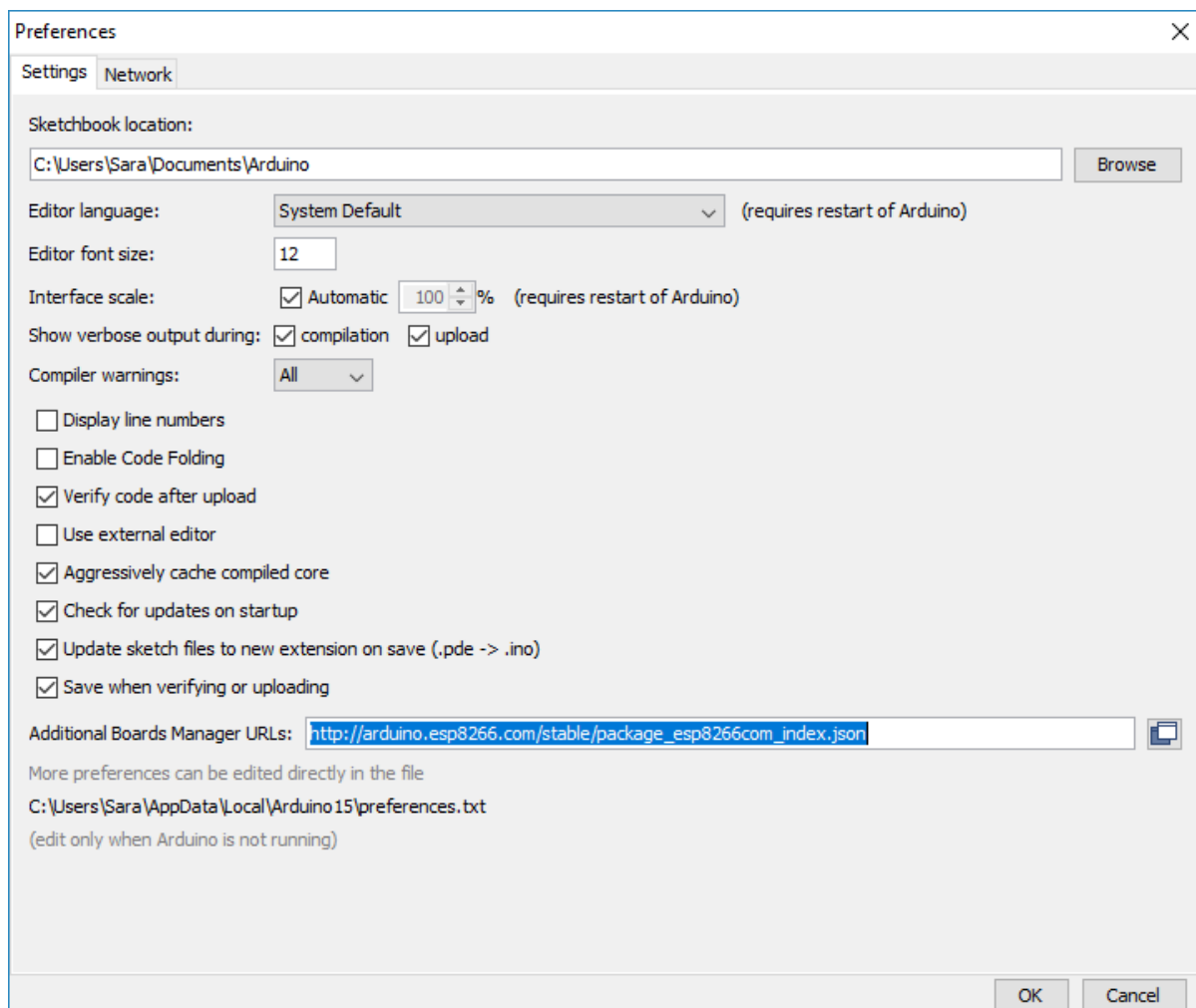
This ESP8266 Web Server is mobile responsive and it can be accessed with any device that has a browser in your local network. The code for this project is done using Arduino IDE.

Prepare the Arduino IDE

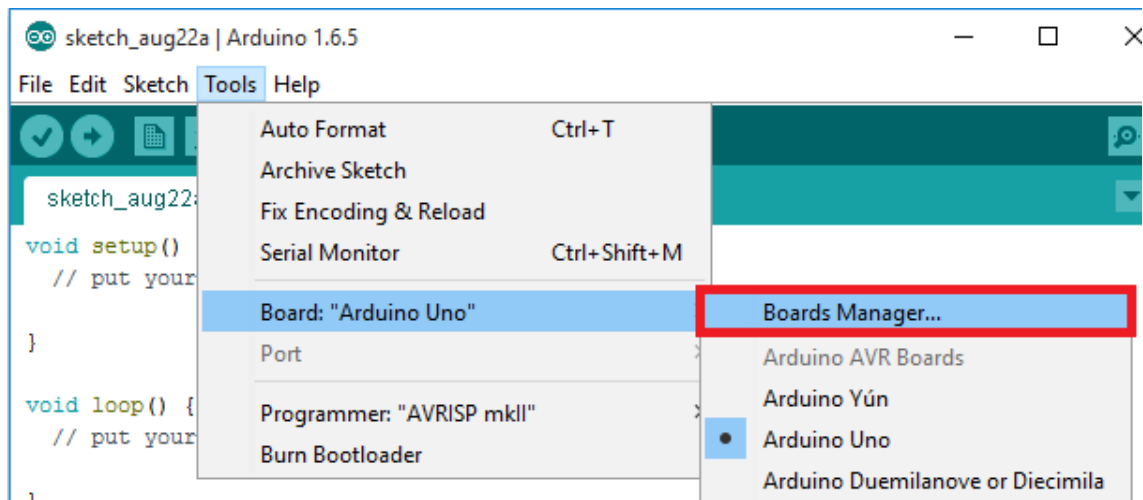
1. Download and [install the Arduino IDE](#) on your operating system (some older versions won't work).
2. Then, you need to install the ESP8266 add-on for the Arduino IDE. For that, go to **File ► Preferences**.
3. Enter the following URL in the **Additional Board Manager** field:

```
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```

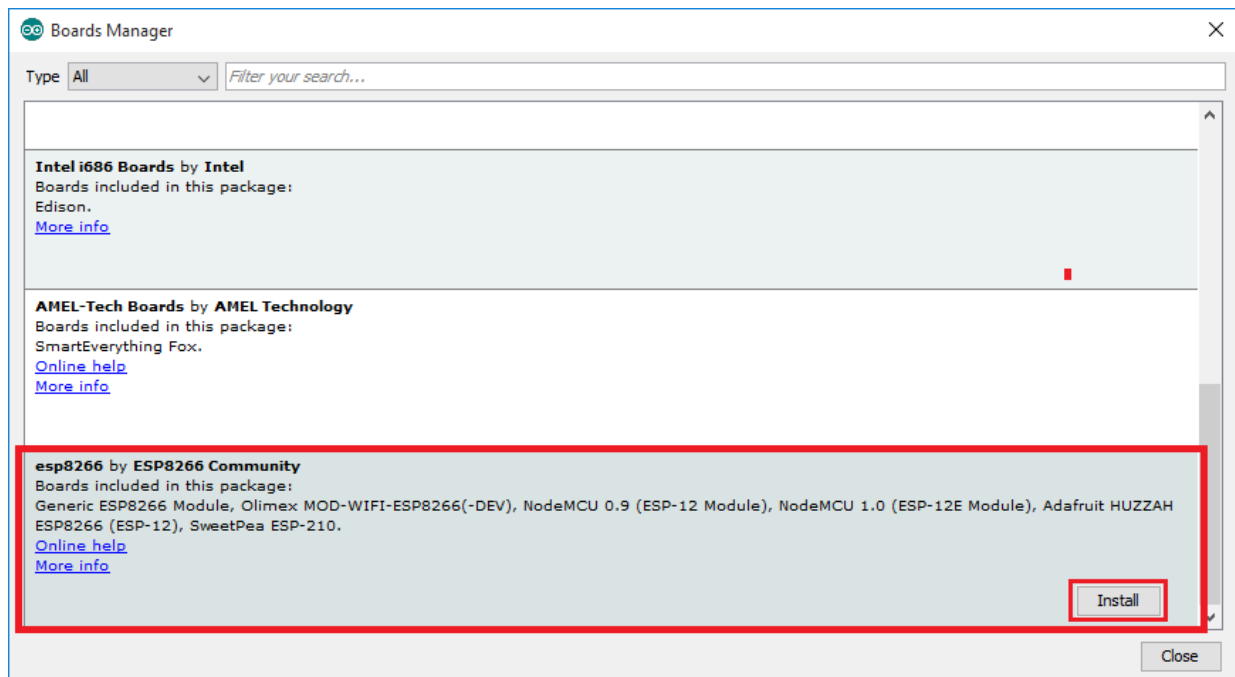
Then, click the **OK** button.



4. Go to **Tools** ▶ **Board** ▶ **Boards Manager...**



5. Scroll down, select the ESP8266 board menu and install “esp8266 platform”, as shown in the figure below.



6. Go to **Tools** ▶ **Board** and select your ESP8266 board. Then, re-open your Arduino IDE.

Code

Copy the code below to your Arduino IDE, but don't upload it yet. You need to make some changes to make it work.

SOURCE CODE

```
// Load Wi-Fi library
#include <ESP8266WiFi.h>

// Replace with your network credentials
const char* ssid      = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_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 output5State = "off";
String output4State = "off";

// Assign output variables to GPIO pins
const int output5 = 5;
const int output4 = 4;

void setup() {
```



```

Serial.begin(115200);
// Initialize the output variables as outputs
pinMode(output5, OUTPUT);
pinMode(output4, OUTPUT);
// Set outputs to LOW
digitalWrite(output5, LOW);
digitalWrite(output4, 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
        while (client.connected()) {       // loop while the client's connected
            if (client.available()) {       // if there's bytes to read
                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 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 /5/on") >= 0) {
    Serial.println("GPIO 5 on");
    output5State = "on";
    digitalWrite(output5, HIGH);
} else if (header.indexOf("GET /5/off") >= 0) {
    Serial.println("GPIO 5 off");
    output5State = "off";
    digitalWrite(output5, LOW);
} else if (header.indexOf("GET /4/on") >= 0) {
    Serial.println("GPIO 4 on");
    output4State = "on";
    digitalWrite(output4, HIGH);
} else if (header.indexOf("GET /4/off") >= 0) {
    Serial.println("GPIO 4 off");
    output4State = "off";
    digitalWrite(output4, 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
client.println("<style>html { font-family: Helvetica; display:");
inline-block; margin: 0px auto; text-align: center;});");
client.println(".button { background-color: #195B6A; border:");
none; color: white; padding: 16px 40px;");
client.println("text-decoration: none; font-size: 30px; margin:");
2px; cursor: pointer;});");
client.println(".button2 {background-color:");
#77878A;}</style></head>");

```

```

// Web Page Heading
client.println("<body><h1>ESP8266 Web Server</h1>");

// Display current state, and ON/OFF buttons for GPIO 5
client.println("<p>GPIO 5 - State " + output5State + "</p>");
// If the output5State is off, it displays the ON button
if (output5State=="off") {
    client.println("<p><a href=\"/5/on\"><button
class=\"button\">ON</button></a></p>");
} else {
    client.println("<p><a href=\"/5/off\"><button class=\"button
button2\">OFF</button></a></p>");
}

// Display current state, and ON/OFF buttons for GPIO 4
client.println("<p>GPIO 4 - State " + output4State + "</p>");
// If the output4State is off, it displays the ON button
if (output4State=="off") {
    client.println("<p><a href=\"/4/on\"><button
class=\"button\">ON</button></a></p>");
} else {
    client.println("<p><a href=\"/4/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("");
}
}
```

You need to modify the following two variables with your network credentials, so that your ESP8266 can establish a connection with your router.

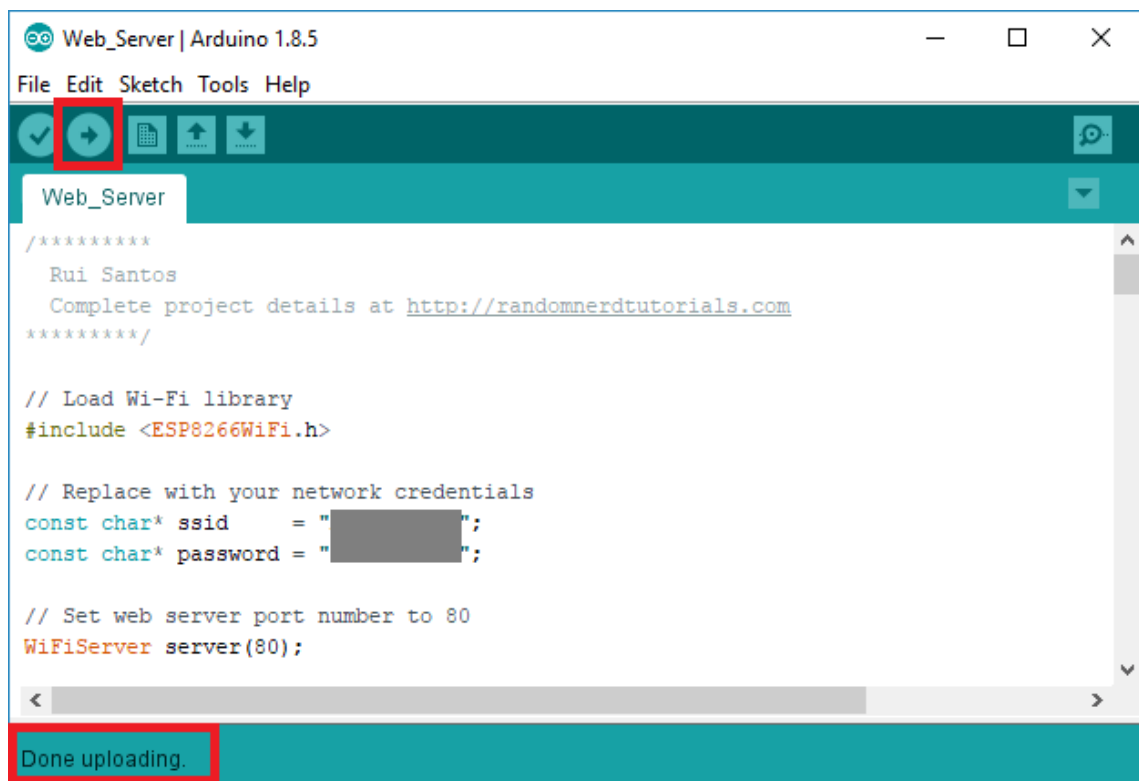
```
// Replace with your network credentials
const char* ssid      = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Uploading the Sketch

Uploading the Sketch to the ESP-12E

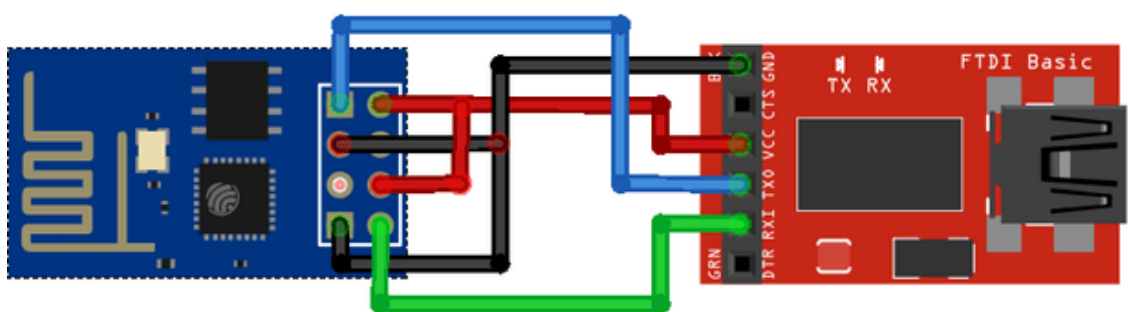
If you're using an ESP-12E NodeMCU Kit, uploading the sketch is very simple, since it has built-in programmer. Plug your board to your computer. Make sure you have the right board and COM port selected.

Then, click the "Upload Button" in the Arduino IDE and wait a few seconds until you see the message "Done uploading." in the bottom left corner.



Uploading Sketch to the ESP-01

Uploading code to the ESP-01 requires establishing a serial communication between your ESP8266 and a FTDI Programmer as shown in the schematic diagram below.



The following table shows the connections you need to make between the ESP8266 and the FTDI programmer.

ESP8266	FTDI programmer
RX	TX
TX	RX
CH_PD	3.3V
GPIO 0	GND
VCC	3.3V
GND	GND

If you have a brand new FTDI Programmer and you need to install your FTDI drivers on Windows PC, visit this website for the [official drivers](#). Alternatively, you can contact the seller that sold you the FTDI Programmer.

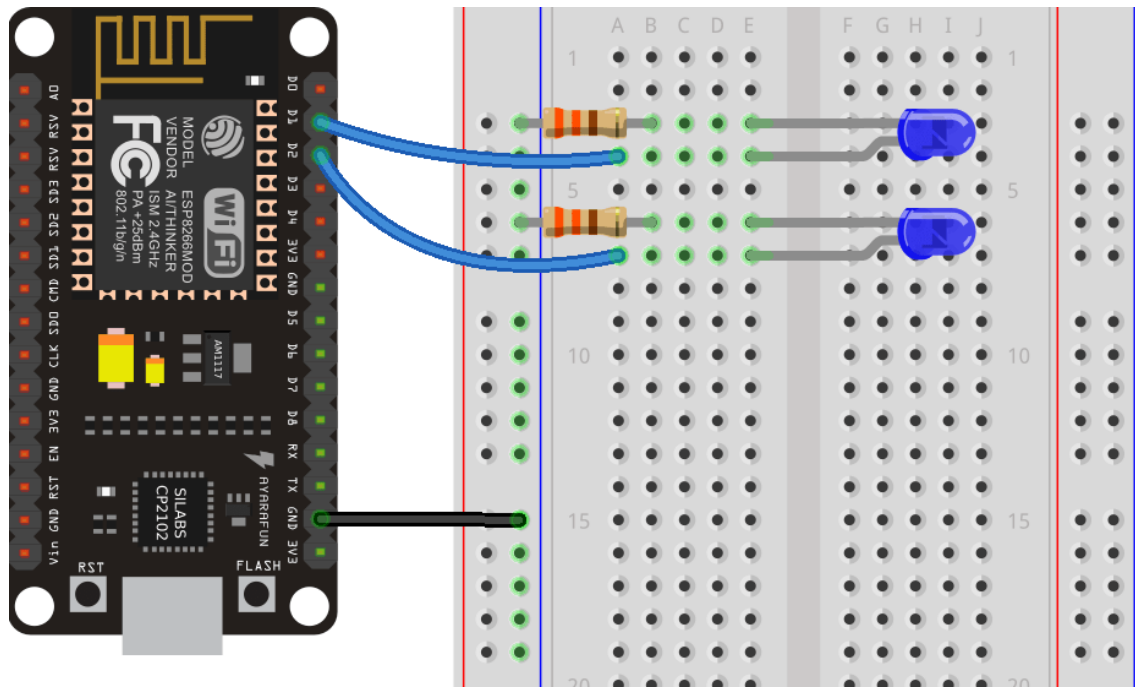
Then, you just need to connect the FTDI programmer to your computer, and upload the code to the ESP8266.

Schematics

To build the circuit you need the following parts:

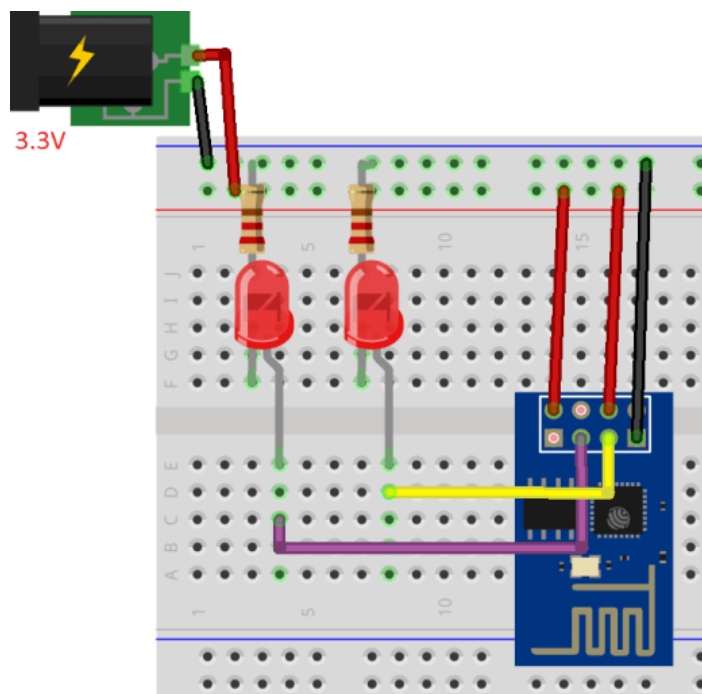
- [ESP8266 12-E](#) – read [Best ESP8266 Wi-Fi Development Boards](#)
- [2x LEDs](#)
- [2x Resistors](#) (220 or 330 ohms should work just fine)
- [Breadboard](#)
- [Jumper wires](#)
- If you're using [ESP-01](#), you also need an [FTDI programmer](#).

Connect two LEDs to your ESP8266 as shown in the following schematic diagram – with one LED connected to GPIO 4, and another to GPIO 5.



If you are using ESP-01...

If you're using the ESP8266-01, use the following schematic diagram as a reference, but you need change the GPIOs assignment in the code (to GPIO 2 and GPIO 0).

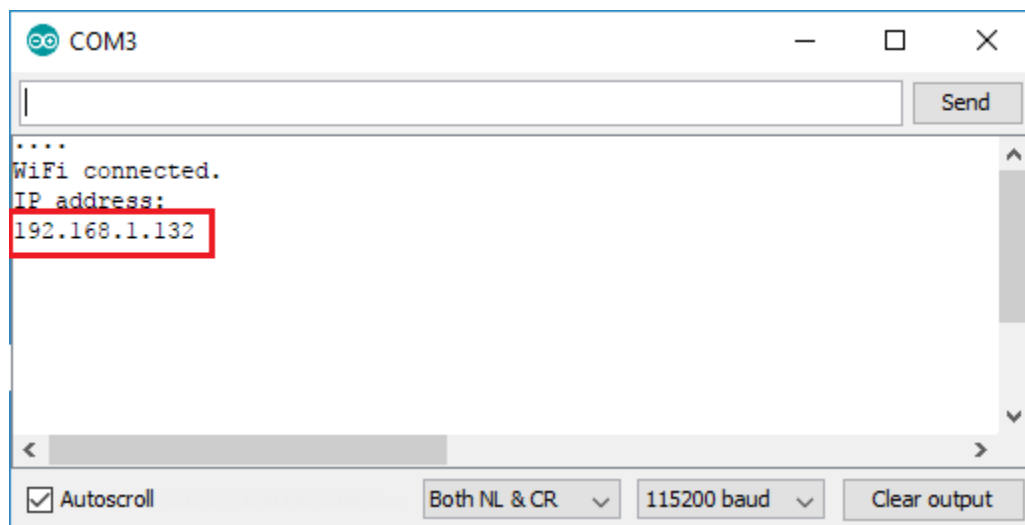


Testing the Web Server

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.

Finding the ESP8266 IP Address

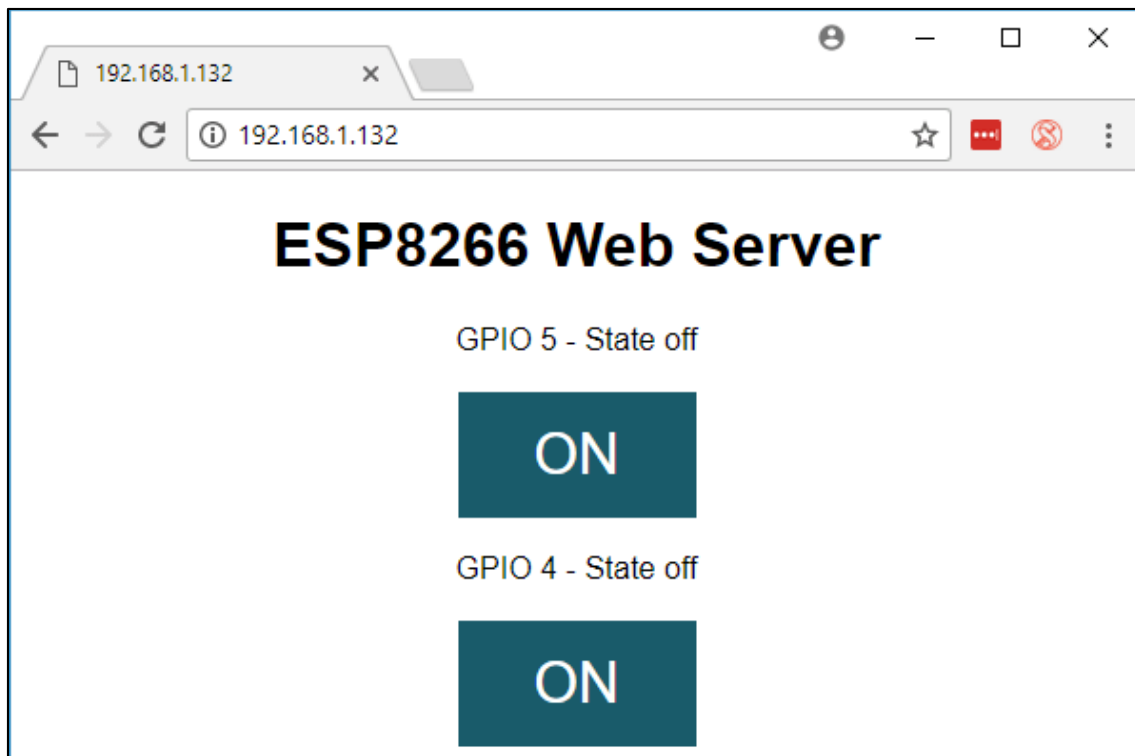
Press the ESP8266 RESET button, and it will output the ESP IP address on the Serial Monitor



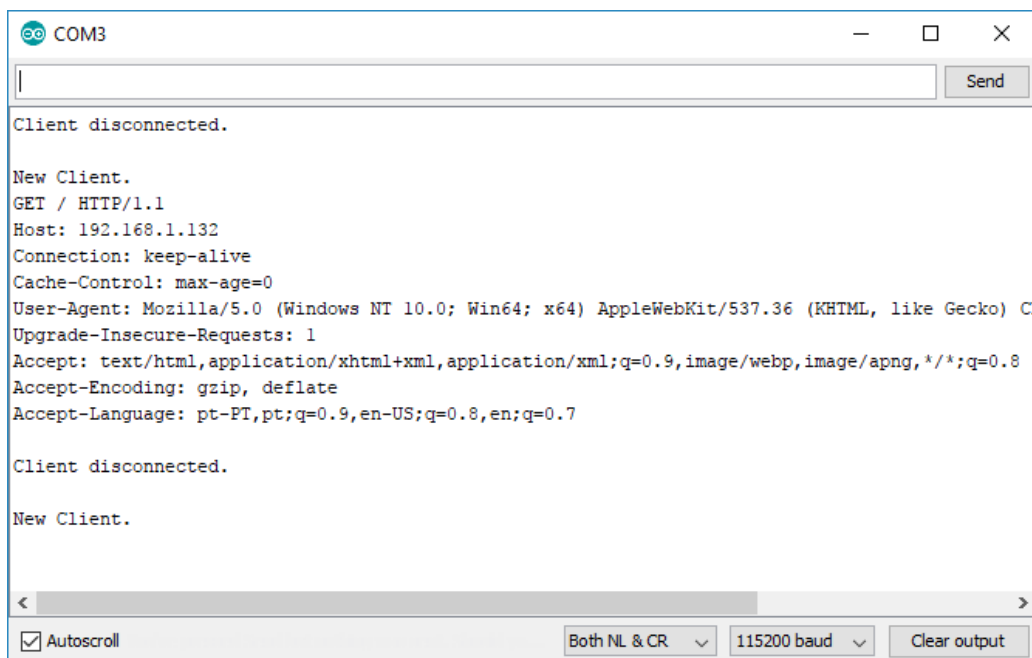
Copy that IP address, because you need it to access the web server.

Accessing the Web Server

Open your browser, type the ESP IP address, and you'll see the following page. This page is sent by the ESP8266 when you make a request on the ESP IP address.

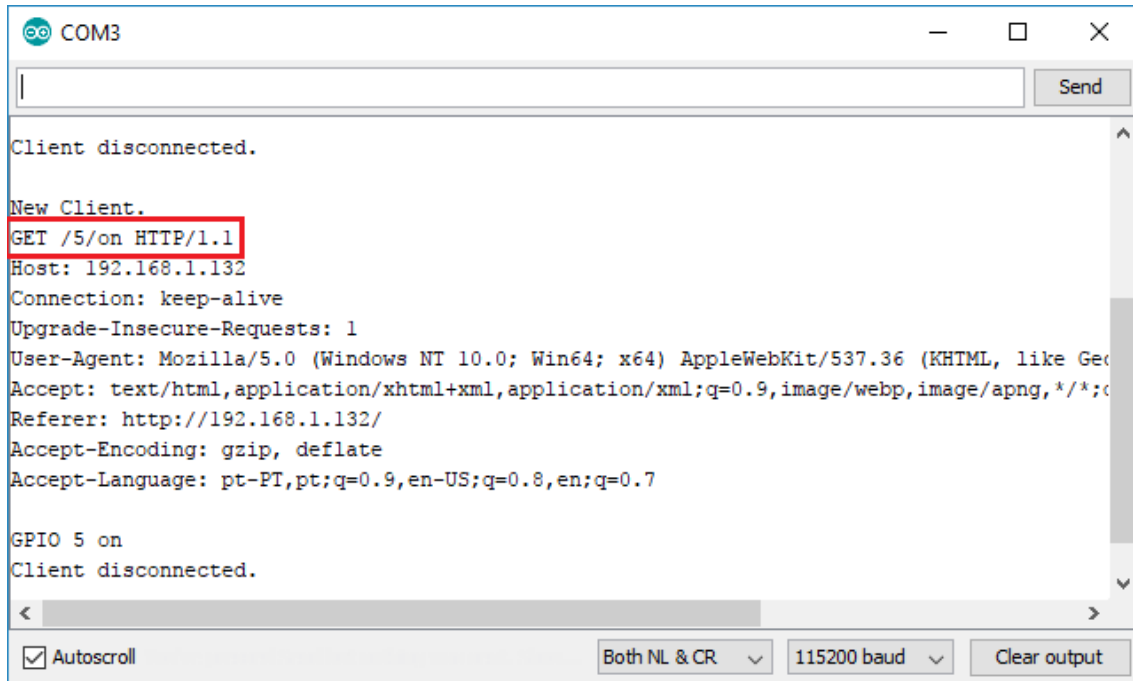


If you take a look at the Serial Monitor, you can see what's going on on the background. The ESP receives an HTTP request from a new client – in this case, your browser. You can also see other information about the HTTP request – these fields are called HTTP header fields, and they define the operating parameters of an HTTP transaction.



Testing the Web Server

Let's test the web server. Click the button to turn GPIO 5 ON. The ESP receives a request on the /5/on URL, and turns LED 5 ON.



```
COM3
|
| Send
Client disconnected.
New Client.
GET /5/on HTTP/1.1
Host: 192.168.1.132
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;
Referer: http://192.168.1.132/
Accept-Encoding: gzip, deflate
Accept-Language: pt-PT,pt;q=0.9,en-US;q=0.8,en;q=0.7

GPIO 5 on
Client disconnected.
```

☒ Autoscroll Both NL & CR 115200 baud Clear output

The LED state is also updated on the web page.



Test GPIO 4 button and check that it works in a similar way.

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 **ESP8266WiFi** library.

```
// Load Wi-Fi library
#include <ESP8266WiFi.h>
```

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

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

Then, you set your web server to port 80.

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

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

```
// Variable to store 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.

```
String output5State = "off";
String output4State = "off";
```

You also need to assign a GPIO to each of your outputs. Here we are using GPIO 5 and GPIO 4. You can use any other suitable GPIOs.

```
const int output5 = 5;
const int output4 = 4;
```

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.

```
pinMode(output5, OUTPUT);
pinMode(output4, OUTPUT);
// Set outputs to LOW
digitalWrite(output5, LOW);
digitalWrite(output4, LOW);
```

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

```
// 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();
```

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(); // Listen for incoming clients
```

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
            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 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();
                }
            }
        }
    }
}
```

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 /5/on") >= 0) {
    Serial.println("GPIO 5 on");
    output5State = "on";
    digitalWrite(output5, HIGH);
} else if (header.indexOf("GET /5/off") >= 0) {
    Serial.println("GPIO 5 off");
    output5State = "off";
    digitalWrite(output5, LOW);
} else if (header.indexOf("GET /4/on") >= 0) {
    Serial.println("GPIO 4 on");
    output4State = "on";
    digitalWrite(output4, HIGH);
} else if (header.indexOf("GET /4/off") >= 0) {
    Serial.println("GPIO 4 off");
    output4State = "off";
    digitalWrite(output4, LOW);
}
```

For example, if you've pressed the GPIO 5 ON button, the URL changes to the ESP IP address followed by **/5/ON**, and we receive that information on the HTTP header. So, we can check if the header contains the expression **GET /5/on**.

If it contains, the code prints a message on the serial monitor, changes the **output5State** variable 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 generate the web page. The ESP8266 will be sending a response to your browser with some HTML text to display the web page.

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

The first text you should always send is the following line that indicates that we're 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\">");
```

The next one is used to prevent requests related to 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 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 some properties to define color, size, border ...

```
client.println(".button { background-color: #195B6A; border: none; color:
white; padding: 16px 40px;");
client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor:
pointer;}");
```

Then, we 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: #77878A;}</style></head>");
```

Setting the Web Page First Heading

In the next line you set the first heading of your web page, you can change this text to whatever you like.

```
client.println("<body><h1>ESP8266 Web Server</h1>");
```

Displaying the Buttons and Corresponding State

Then, you write a paragraph to display the GPIO 5 current state. As you can see we use the **output5State** variable, so that the state updates instantly when this variable changes.

```
client.println("<p>GPIO 5 - State " + output5State + "</p>");
```

Then, we display the on or the off button, depending on the current state of the GPIO.

```
if (output5State=="off") {
    client.println("<p><a href=\"/5/on\"><button class=\"button\">ON</button></a></p>");
} else {
    client.println("<p><a href=\"/5/off\"><button class=\"button
button2\">OFF</button></a></p>");
}
```


We use the same procedure for GPIO 4.

```
// Display current state, and ON/OFF buttons for GPIO 4
client.println("<p>GPIO 4 - State " + output4State + "</p>");
// If the output4State is off, it displays the ON button
if (output4State=="off") {
    client.println("<p><a href=\"/4/on\"><button class=\"button\">ON</button></a></p>");
} else {
    client.println("<p><a href=\"/4/off\"><button class=\"button button2\">OFF</button></a></p>");
}
```

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();
```

Taking it Further

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. Instead of controlling two LEDs, you can control a relay to control practically any electronics appliances.