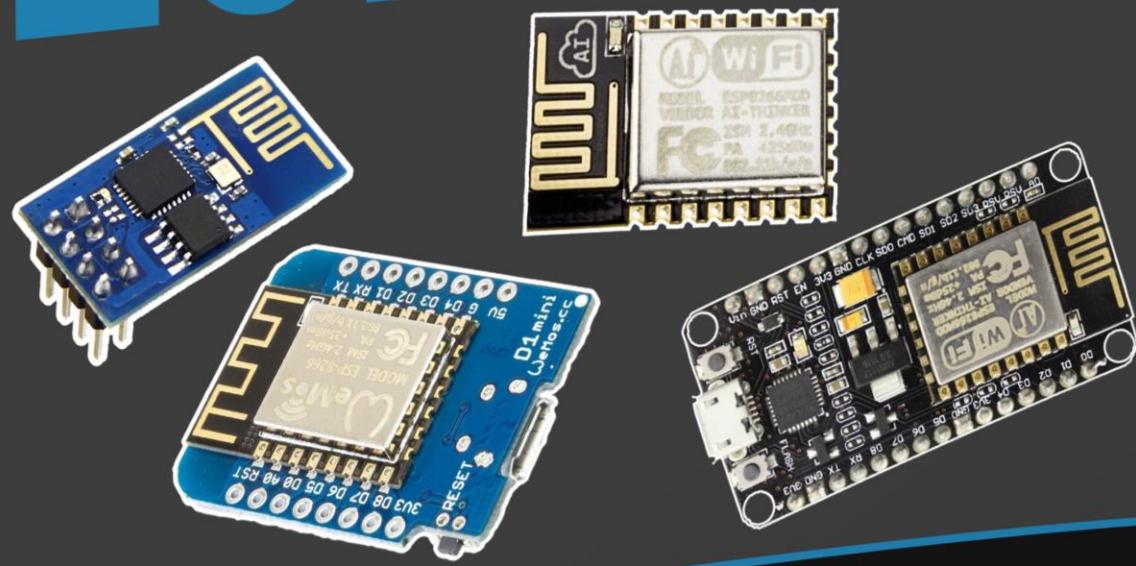


4th Edition

HOME AUTOMATION ESP8266



HOME AUTOMATION USING ESP8266

Build IoT devices to automate your home
using the low cost ESP8266 Wi-Fi module
with Arduino IDE and NodeMCU firmware

RUI SANTOS and SARA SANTOS

4TH EDITION

Security Notice

This is the kind of thing I hate having to write about, but the evidence is clear: piracy for digital products is over all the internet.

For that reason, I've taken certain steps to protect my intellectual property contained in this eBook.

This eBook contains hidden random strings of text that only apply to your specific eBook version that is unique to your email address. You probably won't see anything different, since those strings are hidden in this PDF. I apologize for having to do that – but it means if someone were to share this eBook I know exactly who shared it and I can take further legal consequences.

You cannot redistribute this eBook. This eBook is for personal use and is only available for purchase at:

- <https://randomnerdtutorials.com/courses>
- <https://rntlab.com>

Please send an email to the author (Rui Santos - hello@ruisantos.me), if you find this eBook anywhere else.

What I really want to say is thank you for purchasing this eBook and I hope you have fun and learn a ton with it!

Disclaimer

This eBook has been written for information purposes only. Every effort has been made to make this eBook as complete and accurate as possible. The purpose of this eBook is to educate. The authors (Rui Santos and Sara Santos) don't warrant that the information contained in this eBook is fully complete and shall not be responsible for any errors or omissions.

The authors shall have neither liability nor responsibility to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by this eBook.

Throughout this eBook you might find some links and some of them are affiliate links. This means the authors earn a small commission from each purchase with that link. Please understand that the authors have experience with those products, and they recommend them because they are useful, not because of the small commissions they earn if you decide to buy something. Please do not spend any money on these products unless you need them.

Other helpful links

- [Join Private Facebook Group](#)
- [Terms and Conditions](#)

Table of Contents

PART 0: Getting Started with ESP8266	8
Getting Started with ESP8266	9
ESP8266 GPIO Reference Guide	19
PART 1: ESP8266 with Arduino IDE.....	25
MODULE 1: Getting Started with Arduino IDE.....	26
Unit 1: ESP8266 with Arduino IDE	27
Unit 2: Blinking LED with Arduino IDE.....	33
MODULE 2: Interacting with GPIOs	44
Unit 1: Digital Inputs and Digital Outputs.....	45
Unit 2: Analog Inputs.....	51
Unit 3: PWM – Pulse Width Modulation.....	59
Unit 4: Interrupts and Timers.....	66
Unit 5: Deep Sleep	79
Unit 6: Reference Guide.....	90
MODULE 3: Interfacing with Sensors, Modules and Displays.....	91
Unit 1: DHT11 and DHT22 Digital Temperature and Humidity Sensors	92
Unit 2: BME280 Pressure, Temperature and Humidity Sensor	106
Unit 3: DS18B20 Digital Temperature Sensor.....	117
Unit 4: 0.96 inch OLED Display.....	132
Unit 5: Display Sensor Readings on OLED Display	168
MODULE 4: Web Servers.....	190
Unit 1: Web Server Introduction.....	191
Unit 2: Password Protected Web Server.....	197
Unit 3: Control Sockets Remotely via Web Server	215
Unit 4: Control ESP8266 with Android Widget App.....	227
Unit 5: Making Your Web Server Accessible from Anywhere in the World	237
Unit 6: DS18B20 Temperature Sensor Web Server.....	242
Unit 7: DHT11/DHT22 Asynchronous Web Server	252
Unit 8: RGB LED Strip with Color Picker Web Server.....	274

Unit 9: Web Server using SPIFFS.....	286
Unit 10: Set the ESP8266 as an Access Point	315
MODULE 5: ESP8266 IoT and Home Automation Projects.....	326
Unit 1: Infrared RGB LED Lamp Controller with ESP8266	327
Unit 2: Weather Station with OLED Display.....	341
Unit 3: Voltage Regulator (Prepared for LiPo/Li-ion Batteries).....	355
Unit 4: Wi-Fi Button (DIY Dash button)	366
Unit 5: ESP8266 Daily Task	385
Unit 6: SONOFF - \$5 WiFi Wireless Smart Switch.....	400
MODULE 6: MQTT	418
Unit 1: Introducing MQTT	419
Unit 2: Installing Mosquitto MQTT Broker on a Raspberry Pi.....	424
Unit 3: MQTT Project – MQTT Client ESP8266 #1	428
Unit 4: MQTT Project – MQTT Client ESP8266 #2.....	445
PART 2: ESP8266 with NodeMCU Firmware	458
Unit 1: ESP8266 with NodeMCU Firmware.....	459
Unit 2: Blinking LED with NodeMCU.....	469
Unit 3: Lua Programming Language -The Basics	482
Unit 4: Interacting with the ESP8266 GPIOs using NodeMCU Firmware.....	488
Unit 5: Web Server with ESP8266	492
Unit 6: Displaying Temperature and Humidity on a Web Page.....	504
Unit 7: Email Notifier with ESP8266 and PIR Motion Sensor	515
Unit 8: ESP8266 RGB Color Picker	529
Unit 9: DIY WiFi RGB LED Mood Light.....	537

About the Authors

This course was built, developed, and written by Rui Santos and Sara Santos. We both live in Porto, Portugal, and we know each other since 2009. If you want to learn more about us, feel free to read our [about page](#).



Hi! I'm Rui Santos, the founder of the Random Nerd Tutorials blog. I have a master's degree in Electrical and Computer Engineering from FEUP. I'm the author of "BeagleBone For Dummies", and Technical reviewer of the book "Raspberry Pi For Kids For Dummies". I wrote a book with Sara Santos for the [NoStarchPress publisher](#) about projects with the Raspberry Pi: "[20 Easy Raspberry Pi Projects: Toys, Tools, Gadgets, and More!](#)"



Hi! I'm Sara Santos! I started working at Random Nerd Tutorials in 2015 as a hobby. Back then, I knew nothing about electronics, programming, Arduino, etc.... Over time I started learning everything I could about those subjects and I just loved it! At first, I helped Rui once a week on Saturdays, but then, I started working on the RNT blog almost every day! Currently, I work full time at Random Nerd Tutorials as a Content Editor and I love what I do!

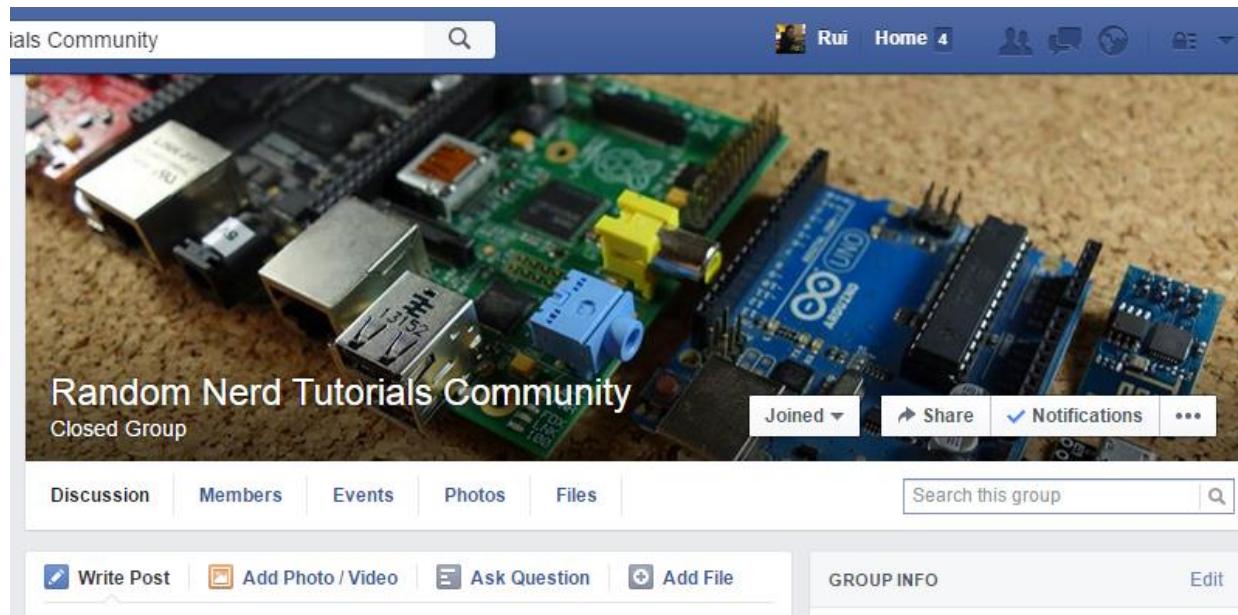
Join the Private Facebook Group

This eBook comes with an opportunity to join a private community of like-minded people. If you purchased this eBook, you can join our private Facebook Group today!

Inside the group, you can ask questions and create discussions about everything related to ESP8266, ESP32, Arduino, Raspberry Pi, BeagleBone, etc.

See it for yourself!

- Step #1: Go to -> <http://randomnerdtutorials.com/fb>
- Step #2: Click “Join Group” button
- Step #3: I’ll approve your request within less than 24 hours.



PART 0

Getting Started with ESP8266



Getting Started with ESP8266

Hello and thank you for purchasing this eBook!

This eBook is our step-by-step guide designed to help you get started with this amazing Wi-Fi module called ESP8266 and build projects that can be used to automate your home.

This eBook covers:

- Technical specifications of the ESP8266;
- ESP8266 GPIO reference guide;
- How to install the Arduino IDE and how it works;
- Use the ESP8266 GPIOs: digital inputs and outputs, analog inputs, PWM, interrupts and much more;
- Interact with several sensors and modules;
- How to create web servers to control your ESP8266 remotely: control outputs and display sensor readings;
- Control sockets remotely using the ESP8266;
- How to access your web server from anywhere;
- Use MQTT to communicate between different devices;
- How to flash the ESP8266 with NodeMCU firmware;
- Lua programming language;
- How to create an email notifier with a PIR motion sensor;
- And a lot more...

Download Source Code and Resources



Each Unit contains the source code and all the resources you need to follow the projects. You can download each code at the Unit page, or you can [download the Home Automation using ESP8266 GitHub repository](#) and instantly download all the resources for this eBook.

Introducing the ESP8266

The ESP8266 is a microcontroller chip from Espressif with on-board Wi-Fi. It can be used as a standalone device, or as a UART to Wi-Fi adaptor to allow other microcontrollers to connect to a Wi-Fi network. For example, you can connect an ESP8266 to an Arduino to add Wi-Fi capabilities to your Arduino board.

With the ESP8266, you can control inputs and outputs as you would do with an Arduino, but with Wi-Fi capabilities. This means that you can bring your projects online, which is great for home automation and internet of things applications.

This entire eBook was designed to take the most out of your ESP8266 board as a standalone device, so you don't even need an Arduino board. You just need an ESP8266 and a few other components!

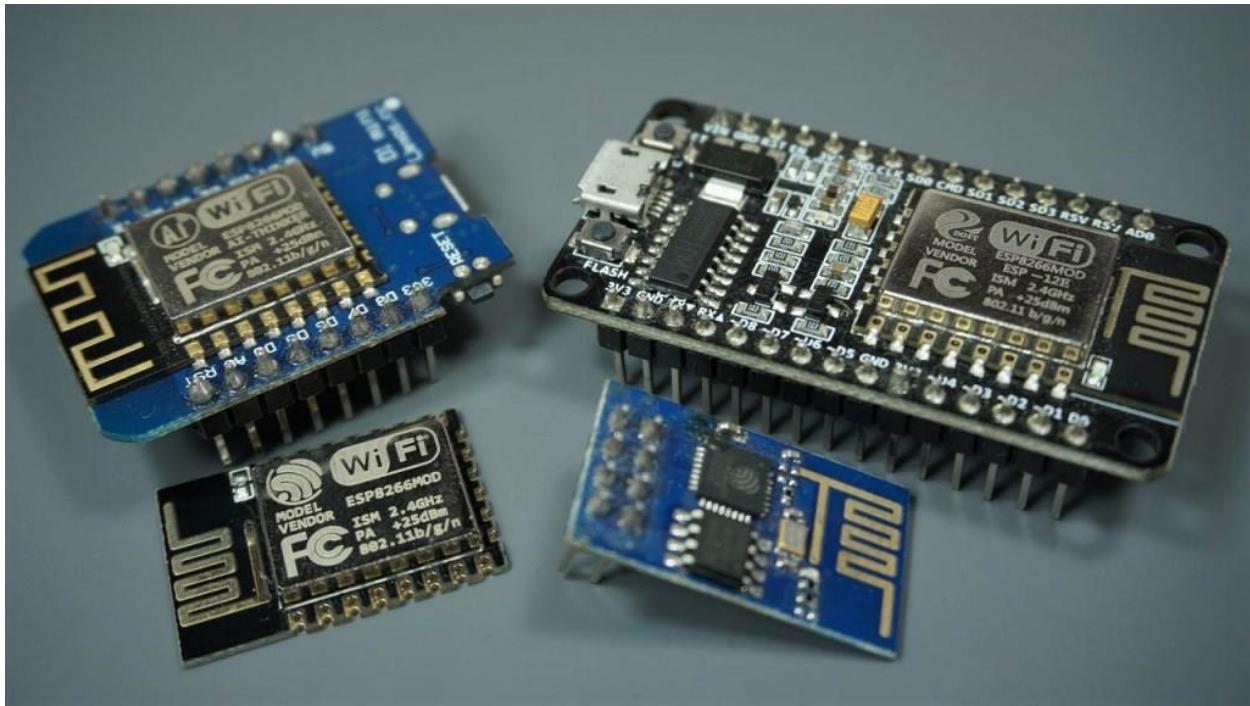
Comparing the ESP8266 with other Wi-Fi solutions on the market, it is a great option for most "Internet of Things" projects! It's easy to see why it's so popular: it only costs a few dollars and can be integrated in advanced projects.

So, what can you do with this low-cost module?

- Create a web server to control outputs;
- Create a web server to display sensor readings;
- Send HTTP requests;
- Control outputs, read inputs and set interrupts;
- Datalogging projects;
- Communicate with third-party services;
- Send emails, notifications, post tweets, etc.

ESP8266 Models

At the moment, there are a wide variety of development boards with the ESP8266 chip that differ in the number of GPIOs, antenna styles, size, breadboard compatibility, etc.



The best ESP8266 development board for your project will really depend on what you intend to do. The most widely used ESP8266 development boards are the [ESP-01](#), the [ESP8266-12E NodeMCU Kit](#) and the [WeMos D1 Mini](#).

For a quick comparison of these boards, you can take a look at the following table:

	ESP-01	ESP-12E NODEMCU	WeMos D1 MINI
			
Where to Buy?	ESP-01	ESP-12E NodeMCU	WeMos D1 Mini
GPIOs	4 (including TX and RX)	11	11
ADC Pins	1	1	1
Flash Memory	1MB (upgraded version)	4MB	4MB
Breadboard friendly	x	✓	✓
USB to Serial Converter	x	✓	✓
Size	24.75mm x 14.5mm (0.97" x 0.57")	48.55mm x 25.6mm (1.91" x 1")	34.2mm x 25.6mm (1.35" x 1")
Price	\$	\$\$	\$\$

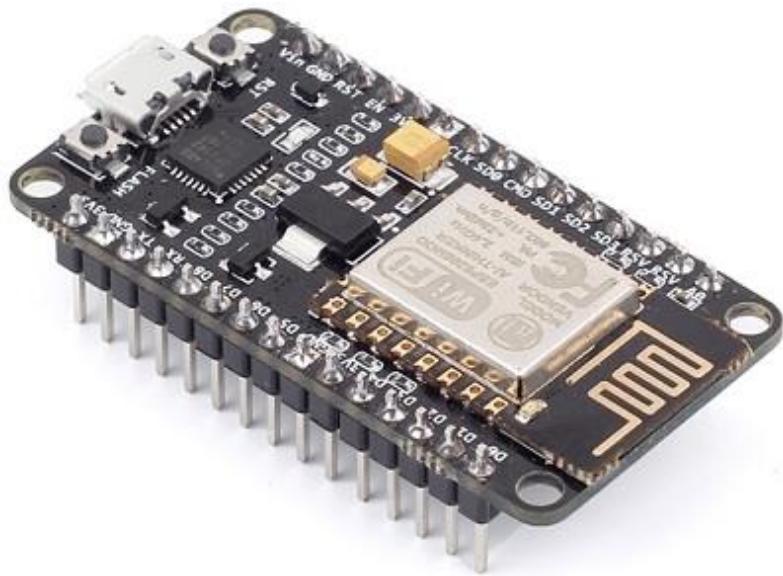
Recommended ESP8266 Board

This eBook was tested with [ESP-01](#), ESP-07, ESP-12, [ESP-12E NodeMCU Kit](#), and [WeMos D1 Mini](#). You can make the projects presented in this eBook using any of these boards. However, there are projects in which a certain board is more suitable than the other. Throughout the eBook we use the [ESP-01](#), the [ESP8266-12E NodeMCU Kit](#) and the [WeMos D1 Mini](#).

If you have to pick only one board, we highly recommend using the [ESP8266-12E NodeMCU Kit](#), the one that has built-in programmer. The built-in programmer makes it easy and faster to upload your programs.

Note: we cannot ensure that all code presented in this eBook works with other ESP8266 boards. Nevertheless, other boards should be compatible with the projects in this eBook with small changes in the pin assignment.

ESP-12E NodeMCU Kit Pinout

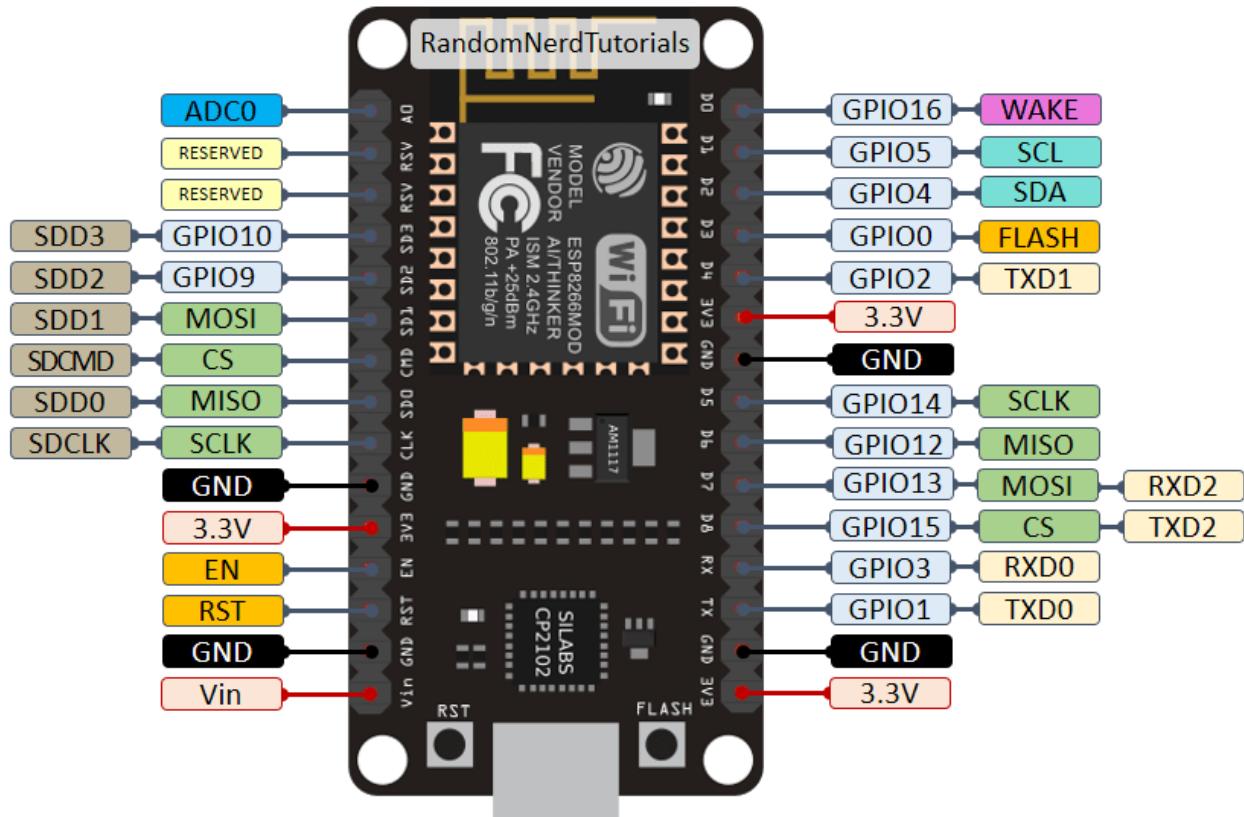


The ESP-12E NodeMCU Kit is one of the most used ESP8266 development boards. It features 4MB of flash memory, access to 11 GPIO pins, and one analog-to-digital

converter (ADC) pin with 10-bit resolution. The board has a built-in voltage regulator and you can power up the module using the mini USB socket or the Vin pin.

Uploading code to the board is as easy as uploading code to the Arduino. There's no need for an FTDI programmer, as it comes with a built-in USB-to-serial converter.

The following figure describes the ESP8266-12E GPIOs and its functionalities:



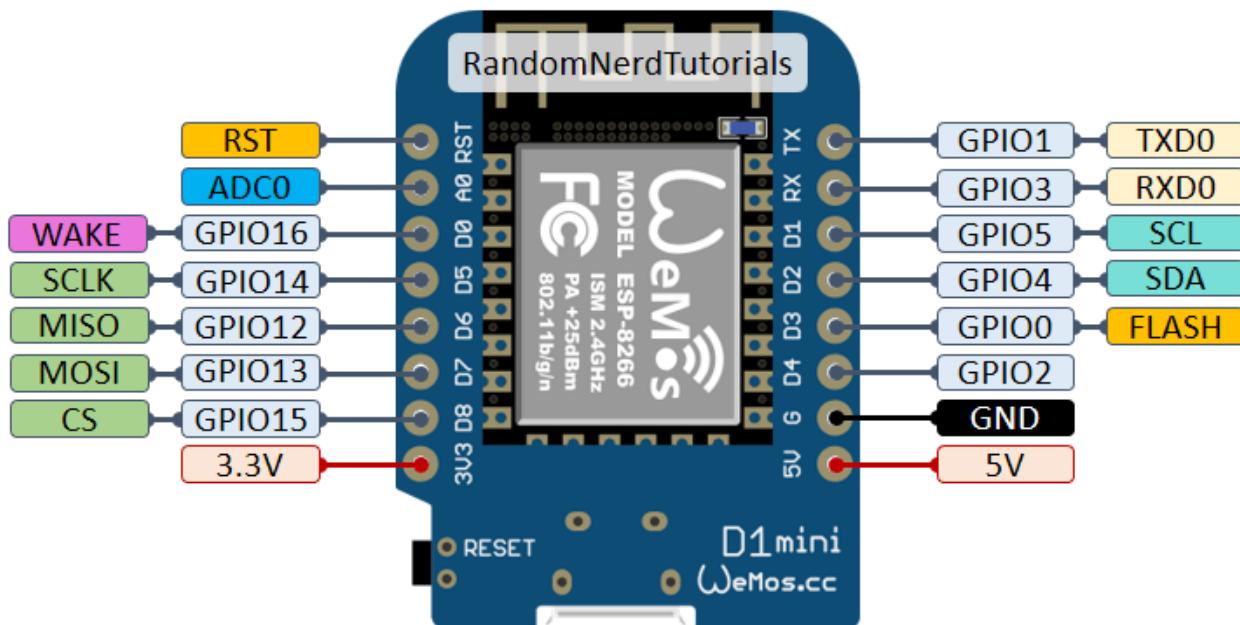
WeMos D1 Mini Pinout



The WeMos D1 Mini offers 4MB flash memory, 11GPIOs, 1 ADC pin in a minimal and small setup. So, if you need something that consumes less power for a more economical project, this is a good option. Additionally, the community has developed a wide variety of shields for the D1 mini board, which allows you to build small and simple setups with almost no wiring required.

The WeMos D1 mini board is also a great option for beginners. However, the board doesn't come with the header pins soldered, so you need to solder them yourself. If you need a soldering iron, you can check the recommend [soldering irons for beginners](#).

If this is your board, you can use the following pinout as a reference.



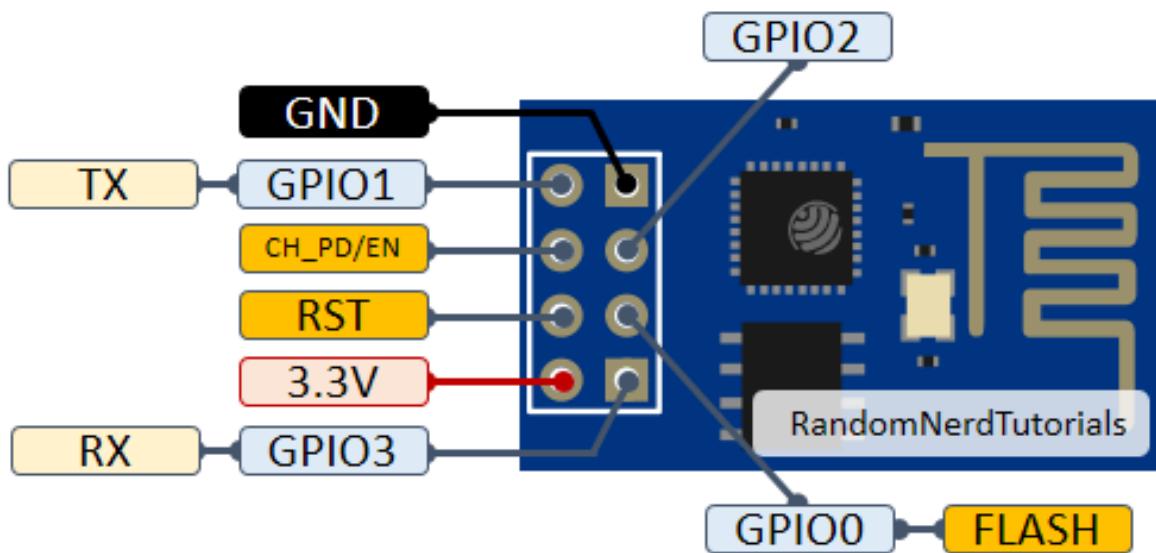
ESP-01 Pinout



The ESP-01 is very small and fits in any enclosure, so it is perfect for finished projects. The upgraded version features 1MB flash memory (the previous version had 512 kB). It offers four GPIOs to control and connect peripherals (two of which are TX and RX for serial communication).

If you need more peripherals in your projects, you should use one of the previous boards. The ESP-01 board doesn't have a built-in voltage regulator, so you need to use a 3V3 power source, or add a voltage regulator to drop the input voltage to 3V3. Additionally, it doesn't come with USB-to-serial converter, which means you need an [FTDI programmer](#) to upload code.

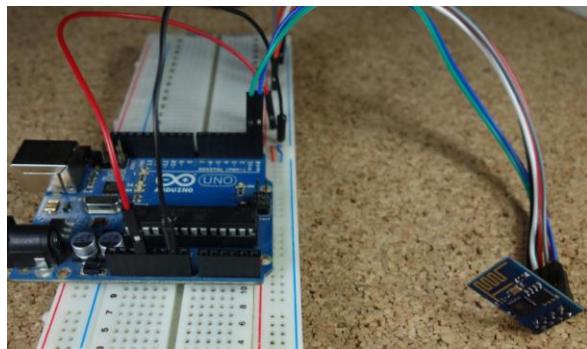
If your project requires very little pins to work, you might consider using the ESP-01. Here's a quick overview of the ESP-01 pinout:



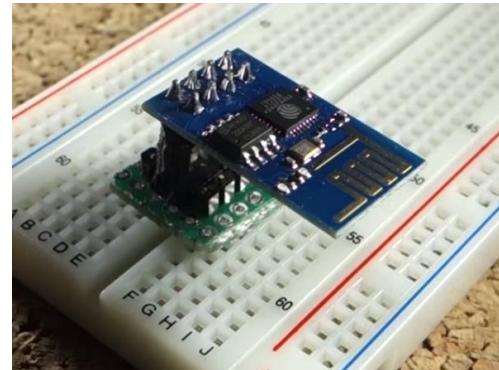
ESP-01 is Not Breadboard Friendly...

Sadly, out of the box your ESP-01 is not breadboard friendly. You can either use some female to male jumper wires (described below as Option A); or you can take an extra step and design a small PCB that fits nicely in your breadboard (shown below as Option B). Both ways work fine.

Option A – jumper wires



Option #2 – small PCB



Download the Boards Pinout

It is very handy to have your boards pinout at hand. So, we recommend printing the pinout diagrams for a future reference:

- [Download the PDF Pinout Diagrams](#)

Programming environments

The ESP8266 can be programmed in many different ways. In this eBook, we cover Arduino IDE and NodeMCU firmware.

- PART 1 – ESP8266 with **Arduino IDE**
- PART 2 – ESP8266 with **NodeMCU Firmware**

We personally prefer to program the ESP8266 with the Arduino IDE, but both methods are very popular and have a wide variety of examples available with a quick Google search.

We encourage you to follow this book linearly and try the Arduino IDE first, then try to program the ESP8266 with NodeMCU firmware.

Note: the ESP8266 can also be programmed with MicroPython: a re-implementation of Python programming language targeted to microcontrollers. If you want to learn how to program your ESP8266 with MicroPython, check our eBook: [MicroPython Programming with ESP32 and ESP8266 eBook](#).

ESP8266 GPIO Reference Guide

This section is a reference for the ESP8266 GPIOs. Here, we show you the function and behavior of each of the ESP8266 pins and which pins to choose for your electronics projects. Come back to this guide whenever you need.

ESP8266 Peripherals

The ESP8266 peripherals include:

- 17 GPIOs
- SPI
- I2C (implemented in software)
- I2S interfaces with DMA
- UART
- 10-bit ADC

Best Pins to Use – ESP8266

One important thing to notice about ESP8266 is that the GPIO number doesn't match the label on the board silkscreen. For example, D0 corresponds to GPIO 16 and D1 corresponds to GPIO 5.

The following table shows the correspondence between the labels on the silkscreen and the GPIO number, as well as what pins are the best to use in your projects, and which ones you need to be cautious.

The pins highlighted in green are OK to use. The ones highlighted in yellow are OK to use, but you need to pay attention because they may have unexpected behavior, mainly at boot. The pins highlighted in red are not recommended to use as inputs or outputs.

Label	GPIO	Input	Output	Notes
D0	GPIO 16	no interrupt	no PWM or I2C support	HIGH at boot used to wake up from deep sleep
D1	GPIO 5	OK	OK	often used as SCL (I2C)
D2	GPIO 4	OK	OK	often used as SDA (I2C)
D3	GPIO 0	pulled up	OK	connected to FLASH button, boot fails if pulled LOW
D4	GPIO 2	pulled up	OK	HIGH at boot connected to on-board LED, boot fails if pulled LOW
D5	GPIO 14	OK	OK	SPI (SCLK)
D6	GPIO 12	OK	OK	SPI (MISO)
D7	GPIO 13	OK	OK	SPI (MOSI)
D8	GPIO 15	pulled to GND	OK	SPI (CS) Boot fails if pulled HIGH
RX	GPIO 3	OK	RX pin	HIGH at boot
TX	GPIO 1	TX pin	OK	HIGH at boot debug output at boot, boot fails if pulled LOW
A0	ADC0	Analog Input	X	

GPIOs connected to the Flash Chip

GPIO 6 to GPIO 11 are usually connected to the flash chip in ESP8266 boards. So, these pins are not recommended to use.

Pins used during Boot

The ESP8266 can be prevented from booting if some pins are pulled LOW or HIGH.

The following list shows the state of the following pins on BOOT:

- **GPIO 16:** pin is HIGH at BOOT
- **GPIO 0:** boot failure if pulled LOW
- **GPIO 2:** pin is HIGH at BOOT, boot failure if pulled LOW
- **GPIO 15:** boot failure if pulled HIGH
- **GPIO 3:** pin is HIGH at BOOT
- **GPIO 1:** pin is HIGH at BOOT, boot failure if pulled LOW
- **GPIO 10:** pin is HIGH at BOOT
- **GPIO 9:** pin is HIGH at BOOT

Pins HIGH at Boot

There are certain pins that output a 3.3V signal when the ESP8266 boots. This may be problematic if you have relays or other peripherals connected to those GPIOs. The following GPIOs output a HIGH signal on boot:

- GPIO 16
- GPIO 3
- GPIO 1
- GPIO 10
- GPIO 9

Additionally, the other GPIOs, except GPIO 5 and GPIO 4, can output a low-voltage signal at boot, which can be problematic if these are connected to transistors or relays. You can [read this article](#) that investigates the state and behavior of each GPIO on boot.

GPIO 4 and GPIO 5 are the safest to use GPIOs if you want to operate relays.

Analog Input

The ESP8266 only supports analog reading in one GPIO. That GPIO is called ADC0 and it is usually marked on the silkscreen as A0.

The maximum input voltage of the ADC0 pin is 0 to 1V if you're using the ESP8266 bare chip. If you're using a development board like the ESP8266 12-E NodeMCU kit, the voltage input range is 0 to 3.3V because these boards come with an internal voltage divider.

On-board LED

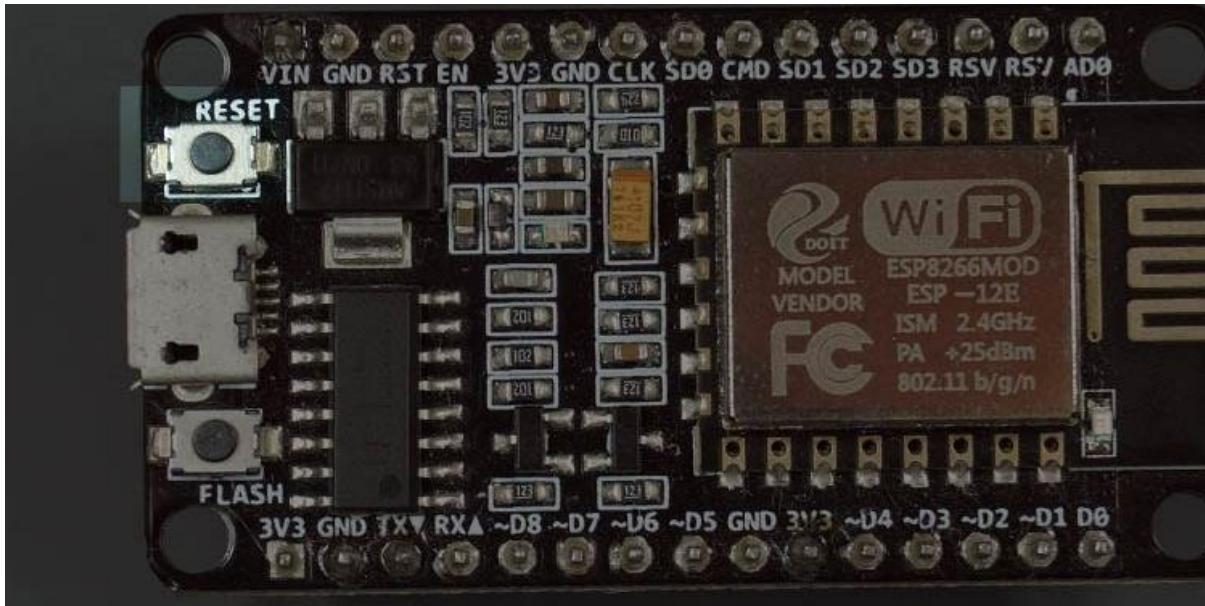
Most of the ESP8266 development boards have a built-in LED. This LED is usually connected to GPIO 2.



The LED is connected to a pull-down resistor, so when you send a HIGH signal the LED turns off and when you send a LOW signal the LED turns on.

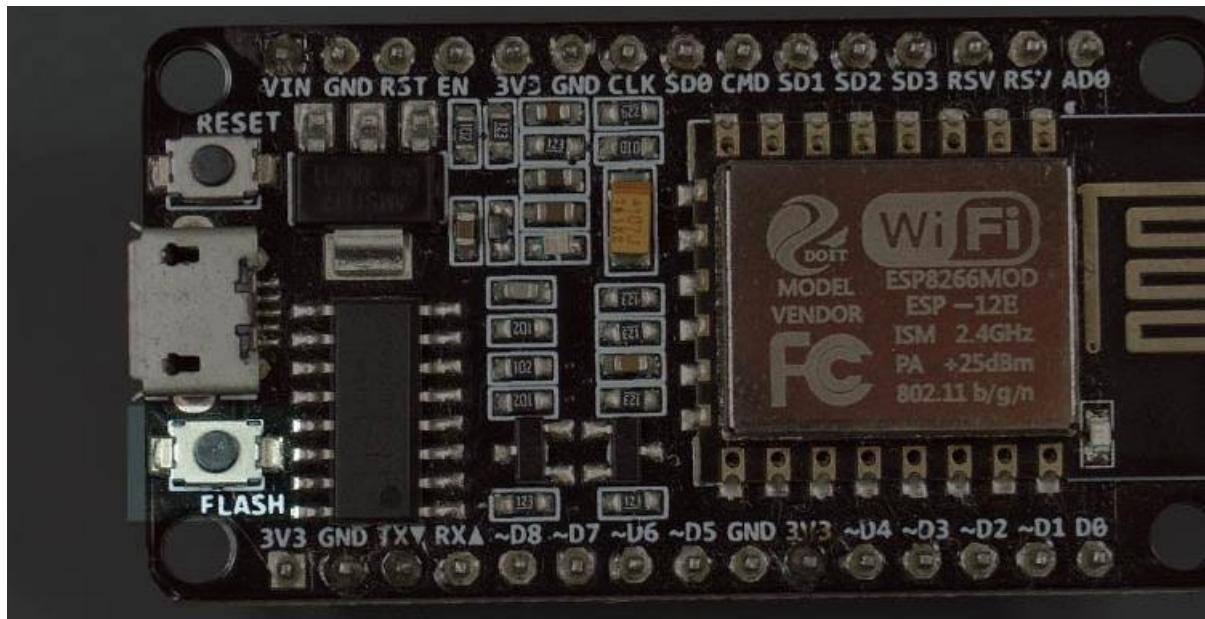
RST Pin

When the RST pin is pulled LOW, the ESP8266 resets. This is the same as pressing the on-board RESET button.



GPIO 0

When GPIO 0 is pulled LOW, it sets the ESP8266 into bootloader mode. This is the same as pressing the on-board FLASH/BOOT button.



GPIO 16

GPIO 16 can be used to wake up the ESP8266 from deep sleep. To wake up the ESP8266 from deep sleep, GPIO 16 should be connected to the RST pin. You'll learn how to implement it later in this eBook.

I2C

The ESP8266 doesn't have hardware I2C pins, but it can be implemented in software. You can use any GPIOs as I2C. However, the following GPIOs are used as default I2C pins:

- **GPIO 5:** SCL
- **GPIO 4:** SDA

SPI

The pins used as SPI in the ESP8266 are:

- **GPIO 12:** MOSI
- **GPIO 13:** MISO
- **GPIO 14:** SCLK
- **GPIO 15:** CS

PWM Pins

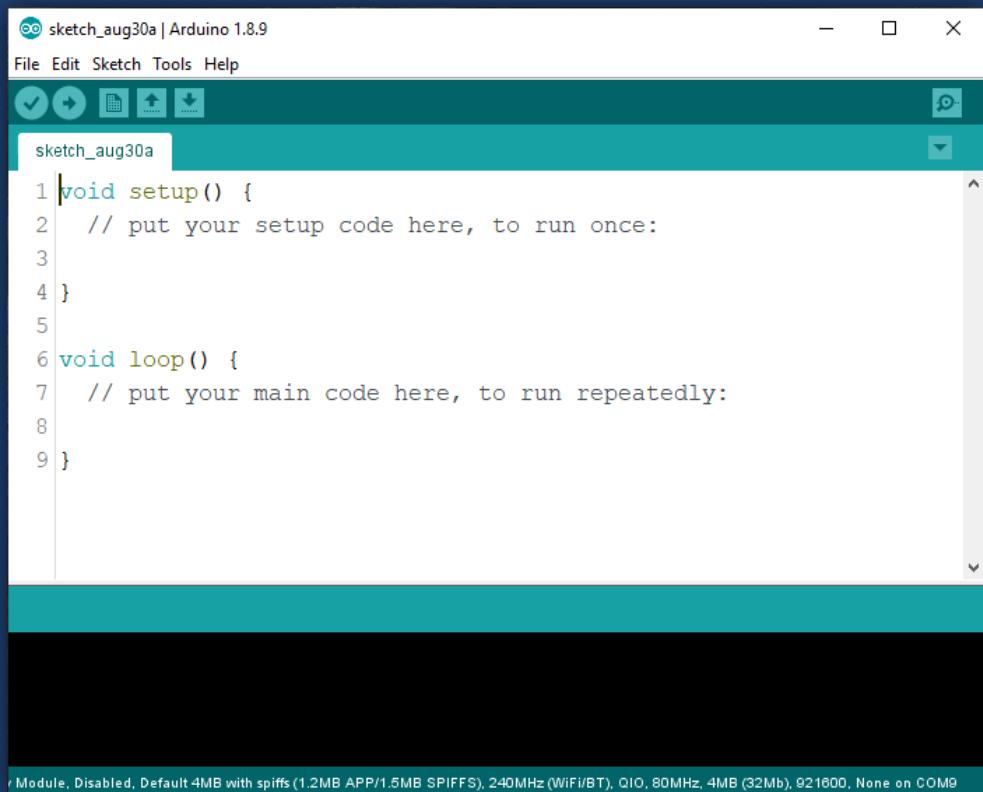
ESP8266 allows software PWM in all I/O pins: GPIO 0 to GPIO 16. PWM signals on ESP8266 have 10-bit resolution.

Interrupt Pins

The ESP8266 supports interrupts in any GPIO, except GPIO 16.

PART 1

ESP8266 with Arduino IDE



The screenshot shows the Arduino IDE interface with the title bar "sketch_aug30a | Arduino 1.8.9". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for back, forward, file operations, and a magnifying glass. The main window displays the code for "sketch_aug30a":

```
1 void setup() {
2     // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7     // put your main code here, to run repeatedly:
8
9 }
```

At the bottom of the IDE, there is a status bar with the text "Module, Disabled, Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS), 240MHz (WiFi/BT), QIO, 80MHz, 4MB (32Mb), 921600, None on COM9".

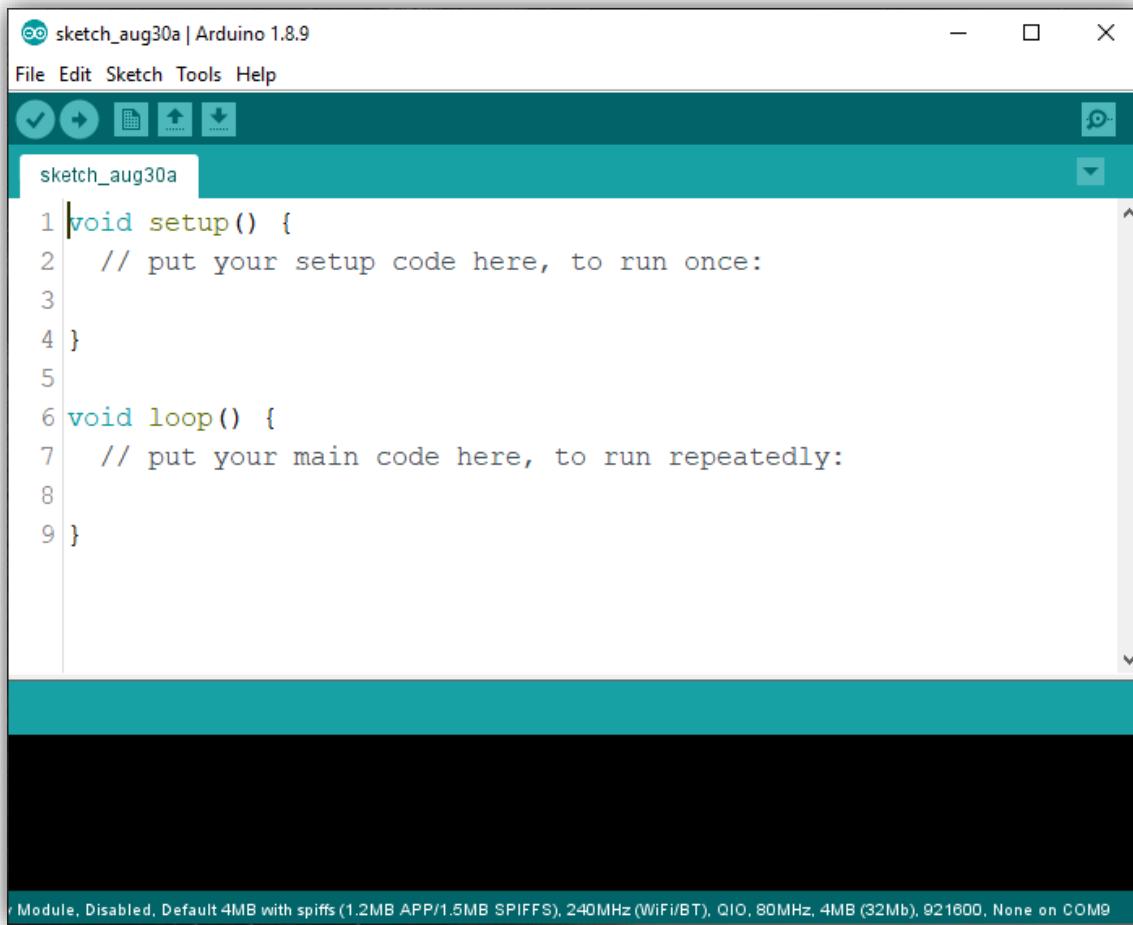
MODULE 1:

Getting Started with

Arduino IDE

In this Module, you'll download, install and prepare the Arduino IDE to work with the ESP8266. You'll also learn how to upload code to your ESP8266 board and blink your first LED.

Unit 1: ESP8266 with Arduino IDE



In this Unit you're going to download, install and prepare your Arduino IDE to work with the ESP8266.

What's the Arduino IDE?

The Arduino IDE is an open-source software that makes it easy to write and upload code to the Arduino board. [This GitHub repository](#) added support for the ESP8266 board to integrate with the Arduino IDE.

The Arduino IDE is a multiplatform software, which means that it runs on Windows, Mac OS X or Linux (it was created in JAVA).

Requirements

You need to have JAVA installed in your computer. If you don't have, go to this website: <http://java.com/download>, download and install the latest version.

Downloading Arduino IDE

To download the Arduino IDE, visit the following URL:

- <https://www.arduino.cc/en/Main/Software>

Select your operating system and download the software.



Installing Arduino IDE

Grab the folder you've just downloaded named "arduino-(...).zip" and unzip it. Run the highlighted file and follow the installation wizard that shows on your screen. Open the Arduino IDE application file (see figure below).

drivers	8/30/2019 11:56 AM	File folder
examples	3/15/2019 2:15 PM	File folder
hardware	8/30/2019 11:57 AM	File folder
java	8/30/2019 11:57 AM	File folder
lib	8/30/2019 11:57 AM	File folder
libraries	8/30/2019 11:57 AM	File folder
reference	3/15/2019 2:16 PM	File folder
tools	8/30/2019 11:58 AM	File folder
tools-builder	3/15/2019 2:15 PM	File folder
arduino.exe	8/30/2019 11:56 AM	Application 395 KB
arduino.l4j.ini	File description: Arduino IDE 1.8.9.0	Configuration setti... 1 KB
arduino_debug.exe	Company: Arduino LLC	Application 393 KB
arduino_debug.l4j.ini	File version: 1.8.9.0	Configuration setti... 1 KB
arduino-builder.exe	Date created: 3/15/2019 2:16 PM	Application 11,791 KB
libusb0.dll	Size: 395 KB	Application extens... 43 KB
msvcp100.dll	8/30/2019 11:56 AM	Application extens... 412 KB
msvcr100.dll	8/30/2019 11:56 AM	Application extens... 753 KB
revisions.txt	8/30/2019 11:56 AM	Text Document 89 KB
wrapper-manifest.xml	8/30/2019 11:56 AM	XML Document 1 KB

When the Arduino IDE first opens, this is what you should see:

```

sketch_aug30a | Arduino 1.8.9
File Edit Sketch Tools Help
sketch_aug30a
1 void setup() {
2   // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7   // put your main code here, to run repeatedly:
8
9 }

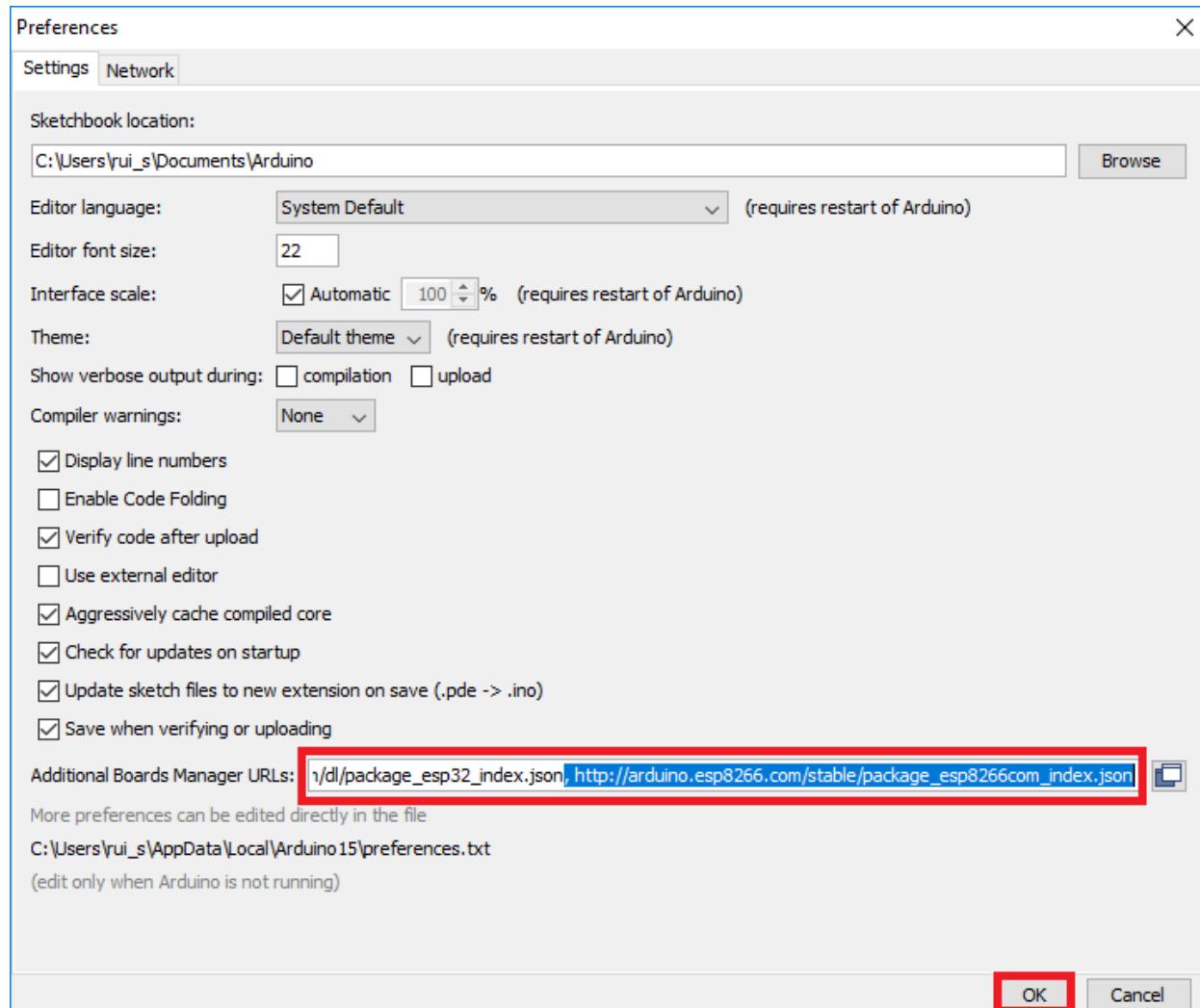
```

Module, Disabled, Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS), 240MHz (WiFi/BT), QIO, 80MHz, 4MB (32Mb), 921600, None on COM9

Installing the ESP8266 add-on

To install the ESP8266 add-on in your Arduino IDE, follow these next instructions:

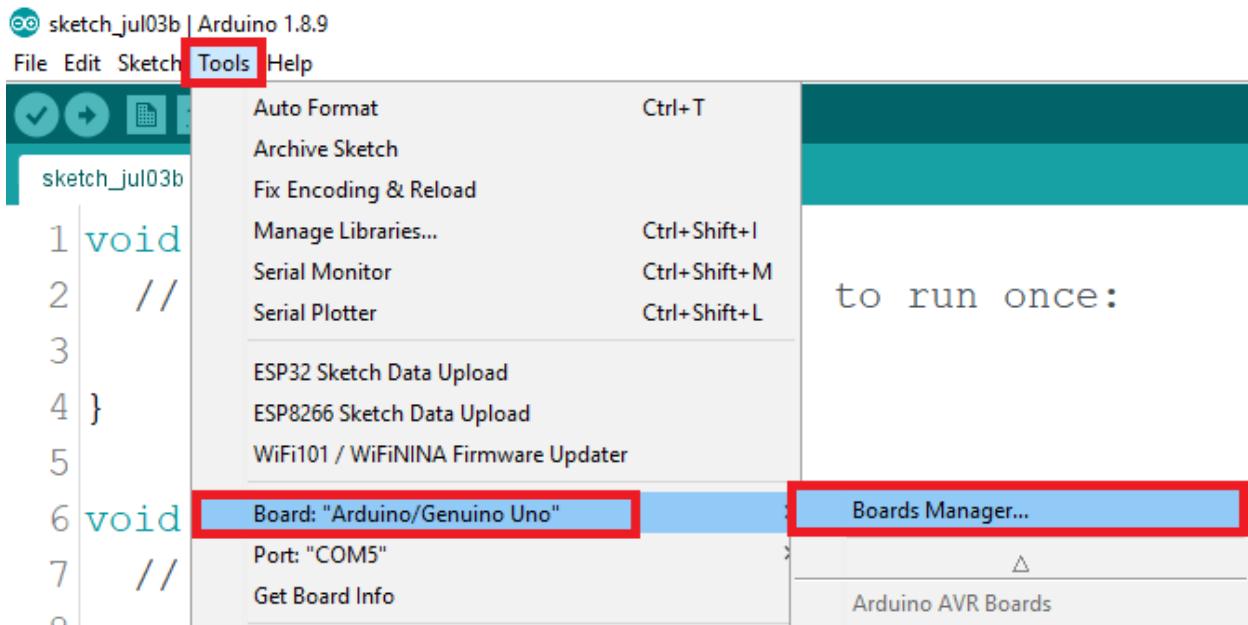
- 1) Open the preferences window in the Arduino IDE. Go to **File > Preferences**.
- 2) Enter "http://arduino.esp8266.com/stable/package_esp8266com_index.json" into **Additional Board Manager URLs** field and press the "**OK**" button.



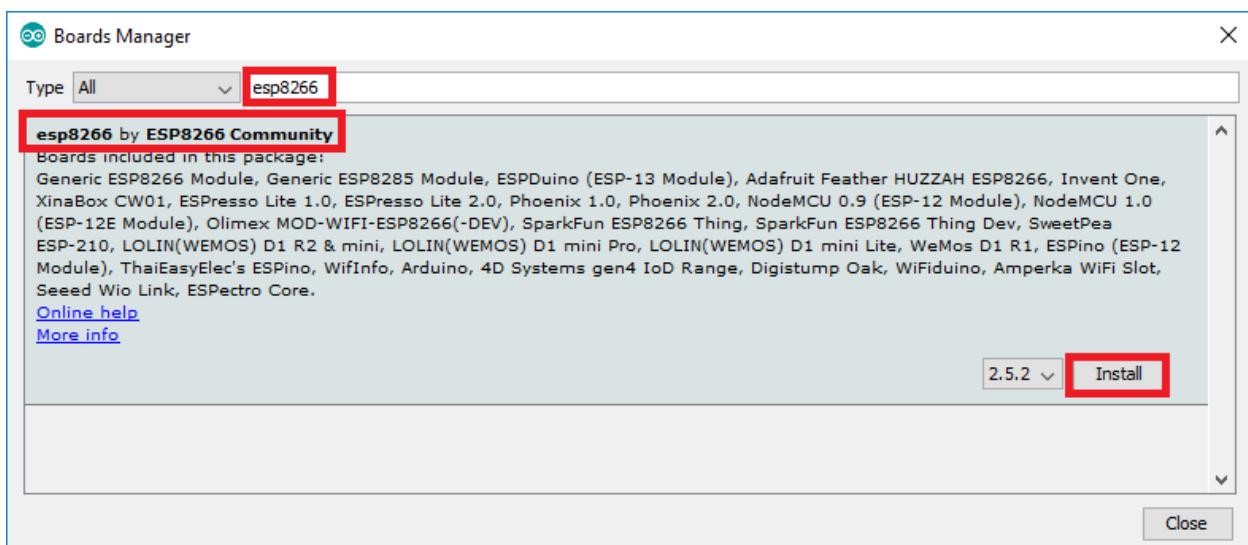
Note: if you already have the ESP32 boards URL, you can separate the URLs with a comma as follows:

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

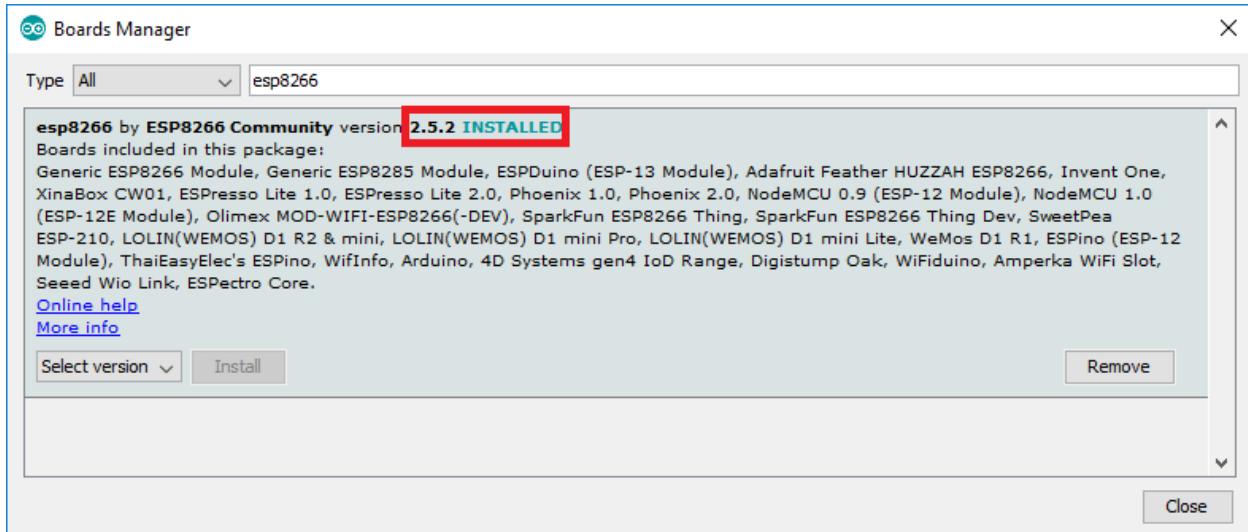
3) Go to **Tools > Board > Boards Manager...**



4) Search for **ESP8266** and press the **INSTALL** button for the “**ESP8266 by ESP8266 Community**”:



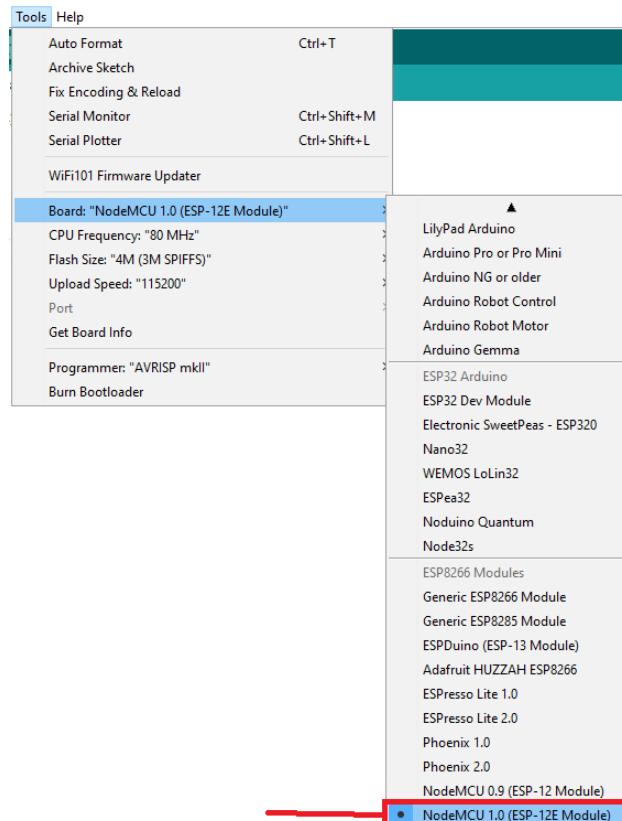
5) It should be installed after a few seconds.



6) After the installation, you can close the **Boards Manager** window.

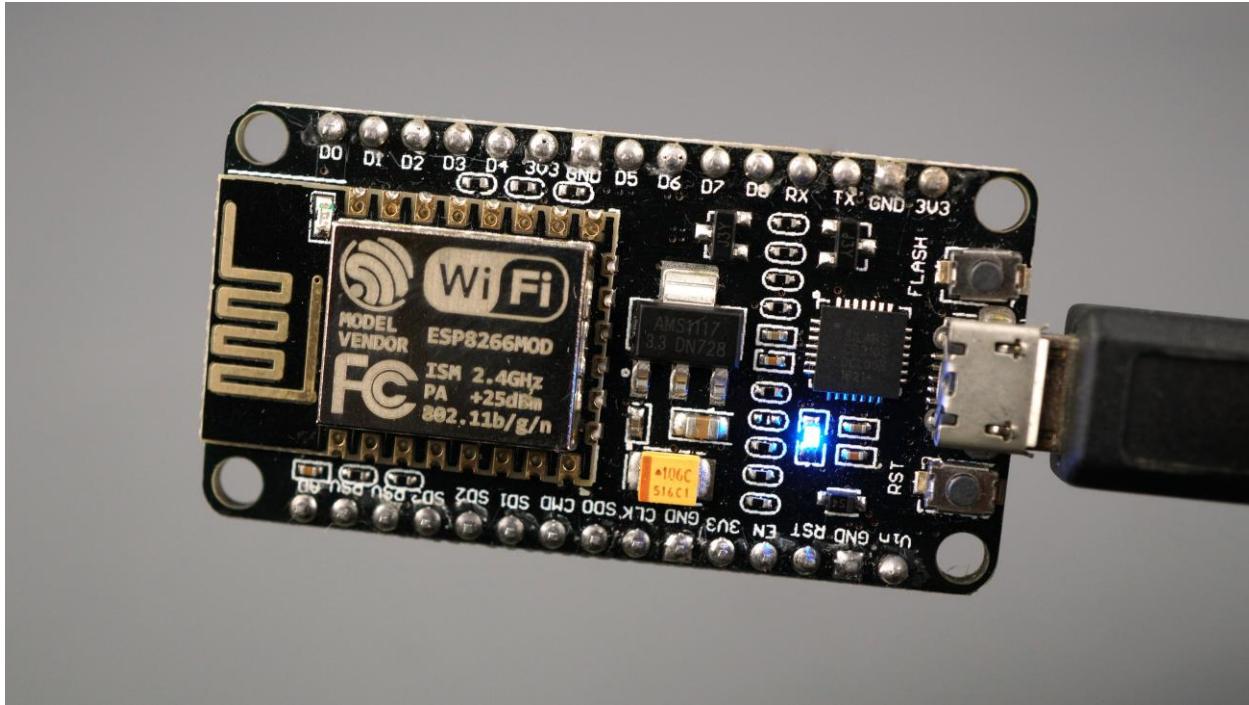
7) Open the Arduino **Tools** menu.

8) Select **Board > NodeMCU 1.0 (ESP-12E Module)**



9) Finally, re-open your Arduino IDE to ensure that it launches with the new boards installed

Unit 2: Blinking LED with Arduino IDE



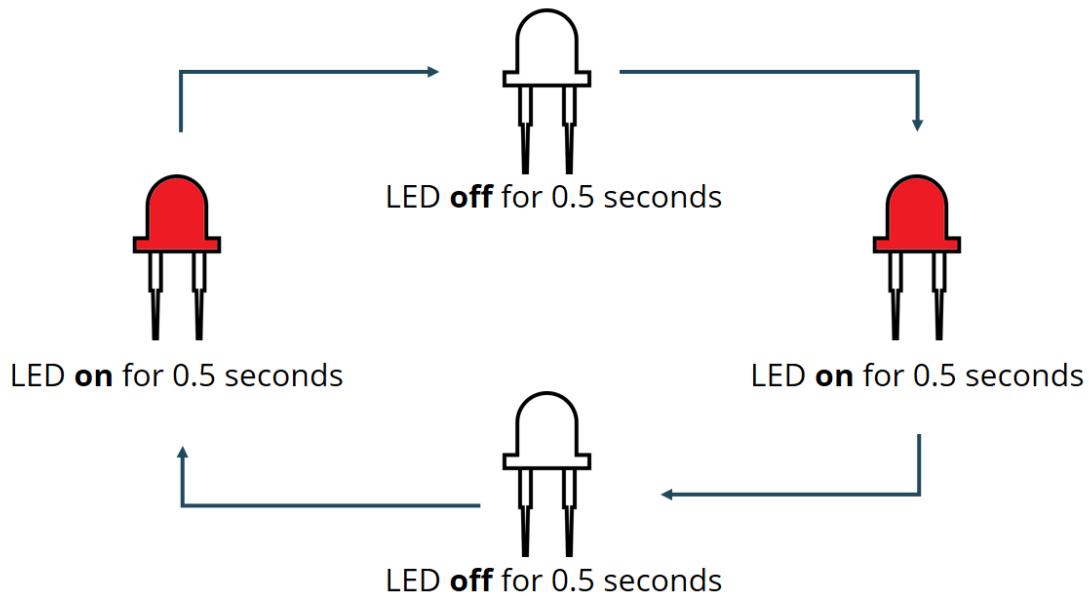
After all the theory we've been showing you, let's finally do something to interact with the physical world. In this Unit, you're going to learn how to blink an LED with the ESP8266 using Arduino IDE. The aim of this Unit is to get you familiar with the procedure to upload and run code and to check if you've installed the ESP8266 board add-on properly in the previous Unit.

Project Overview

The project we'll build consists of simply blinking the ESP8266 on-board LED. The on-board LED is internally connected to the ESP8266 GPIO 2. In simple terms, the blinking LED project works as follows:

- The LED turns on for half a second;
- The LED turns off for half a second;
- The LED is on again for half a second;
- The LED is off again for half a second.

This pattern continues until you tell the program to stop.



GPIO Assignment

One important thing about the ESP8266 is that the GPIO number doesn't match the label on the board silkscreen. For example, D2 corresponds to GPIO 4 and D1 corresponds to GPIO 5. This subject was already covered in a [previous Unit](#).

Use the next table as a quick reference on how to assign the ESP8266 GPIOs in the Arduino code.

IO index (Arduino code)	ESP8266 GPIO	Label
0	GPIO 0	D3
1	GPIO 1	D10
2	GPIO 2	D4
3	GPIO 3	D9

4	GPIO 4	D2
5	GPIO 5	D1
12	GPIO 12	D6
13	GPIO 13	D7
14	GPIO 14	D5
15	GPIO 15	D8
16	GPIO 16	D0
A0	ADC0	A0

To locate the pin on your board, you can check the pinout reference cards provided [in the previous Module](#).

So, in your Arduino IDE, when you define:

```
int pin = 0;
```

You are referring to **GPIO 0**.

On the other hand, if you define:

```
int pin = 2;
```

You are referring to **GPIO 2**.

This is how this firmware is internally defined. You don't need to worry about this, simply remember that 0 refers to GPIO 0 and 2 refers to GPIO 2. We'll explore this concept in more detail later in this eBook.

Writing Your Arduino Sketch

The sketch for blinking the LED is shown below:

```
int pin = 2;

void setup() {
    pinMode(pin, OUTPUT);
}

void loop() {
    digitalWrite(pin, HIGH);
    delay(500);
    digitalWrite(pin, LOW);
    delay(500);
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module1/Unit2/Blink_LED/Blink_LED.ino

How the sketch works

Create an integer (int) variable called `pin` that refers to **GPIO 2**:

```
int pin = 2;
```

In the `setup()`, you use the function `pinMode(pin, OUTPUT)` to set your **GPIO 2** as an **OUTPUT**. This code only runs once.

```
void setup() {
    pinMode(pin, OUTPUT);
}
```

Next, in the `loop()`, you use two functions: `digitalWrite()` and `delay()`. This section of code will run over and over again until you unplug or RESET your ESP8266.

You set the `digitalWrite()` function to control an output. This function accepts as arguments, the GPIO you are referring to, and the state, either `HIGH` or `LOW`.

```
digitalWrite(pin, HIGH);
```

After turning the LED on, wait for half a second using the `delay()` function. This function accepts as argument the number of milliseconds you want to wait. In this case, 500 milliseconds (corresponds to half a second).

```
delay(500);
```

Then, turn the LED off using `digitalWrite(pin, LOW)` and wait half a second with `delay(5000)`.

```
digitalWrite(pin, LOW);
delay(500);
```

The program keeps repeating the previous steps, which makes the LED blinking!

Note: the on-board LED works with inverted logic. This means that when you send a HIGH signal to GPIO 2, the LED turns off. When you send a LOW signal, the LED turns on. However, if you connect an external LED to GPIO 2, it will work as expected.

Uploading Code to ESP8266

There are two different sections to upload code to your ESP8266:

- **Option A:** If you're using an ESP8266-12E or any other board with a built-in programmer;
- **Option B:** If you're using the ESP-01 or ESP-07, you need an [FTDI programmer](#) or an [ESP8266-01 Serial Adapter](#).

Option A - Uploading code to ESP-12E

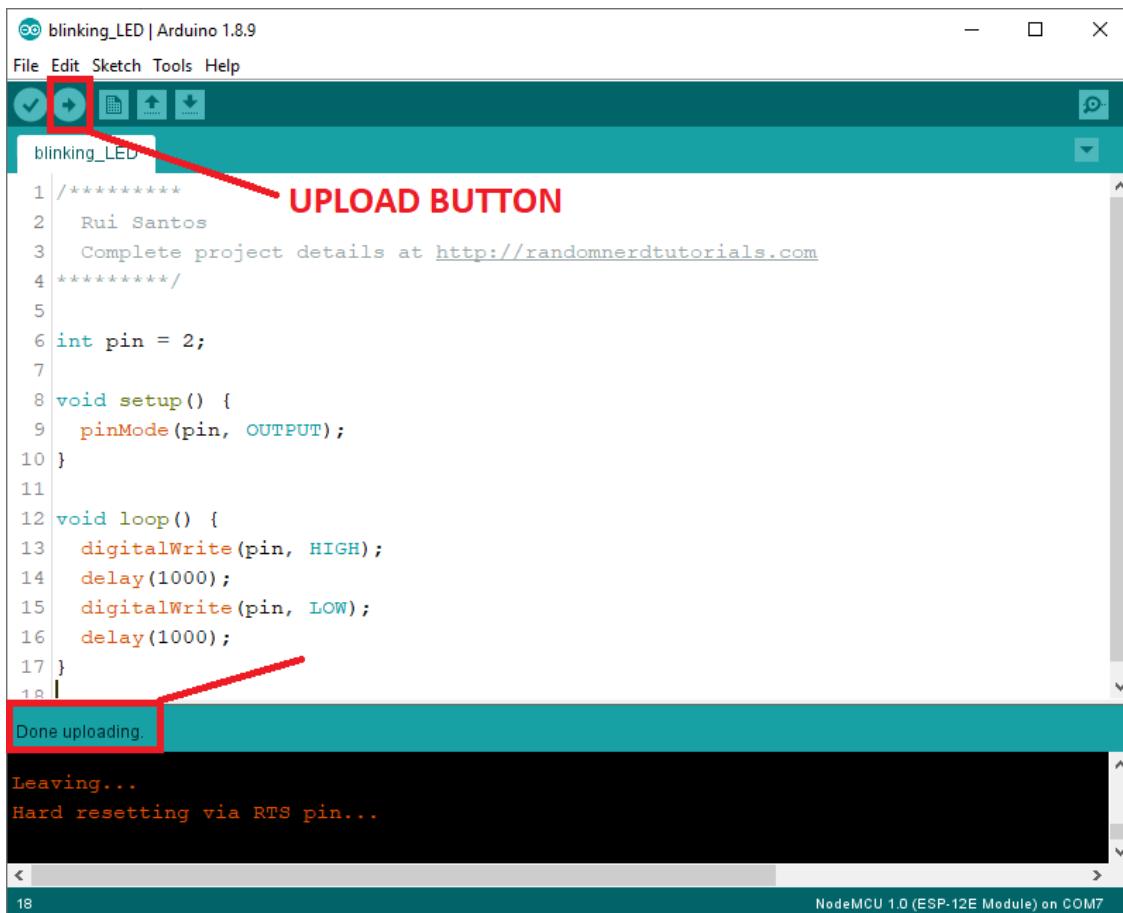
Uploading code to your ESP-12E NodeMCU Kit or any other ESP8266 board with built-in programmer is very simple. You just need to plug your board to your computer using an USB cable without

Note: your USB cable needs to have data wires, so it actually transfers the code. Many USB cables from phone chargers or power banks don't come with data wires. These are not suitable.

After connecting the EPS8266 board to a USB port in your computer, in your Arduino IDE, go to the **Tools** menu and select the “**NodeMCU 1.0 (ESP-12E Module)**” board.

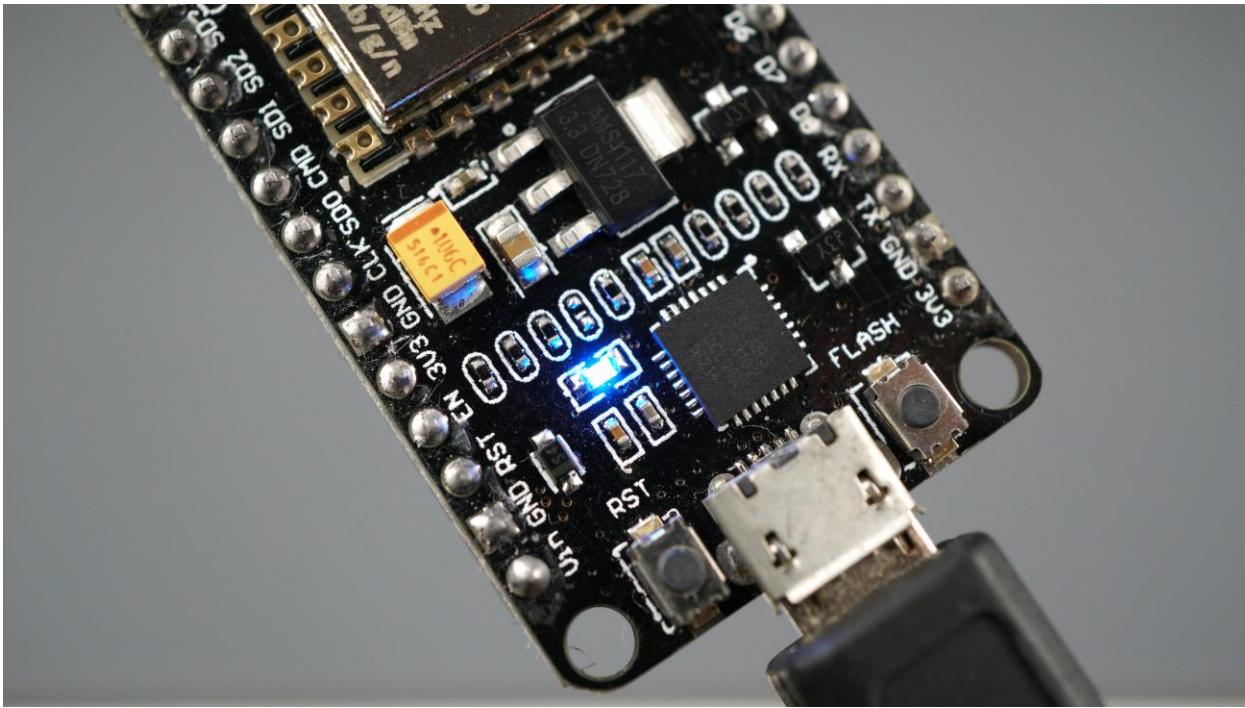
Go to **Tools** ▶ **Port** and select the COM Port the ESP8266 is connected to.

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.



Demonstration

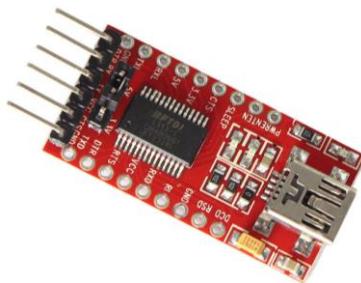
After uploading the code, restart your ESP8266 by pressing the RST button. The on-board LED should be blinking every half a second!



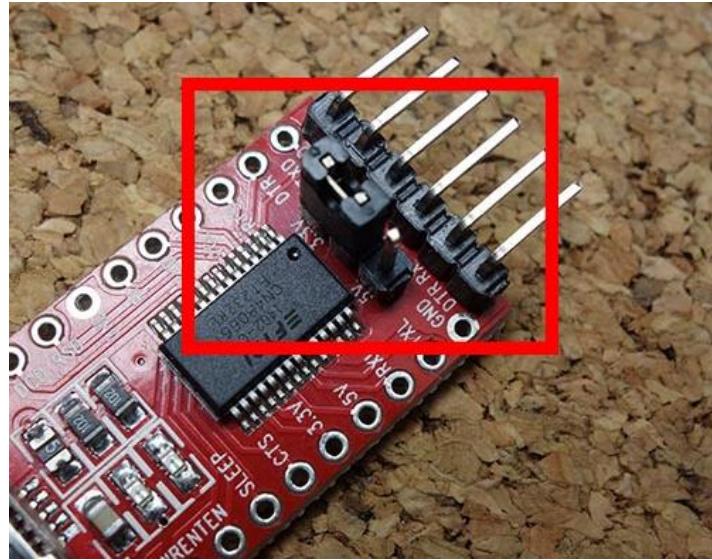
Option B - Uploading code to ESP-01

Uploading code to the ESP-01 requires establishing a serial communication between your ESP8266 and an FTDI Programmer.

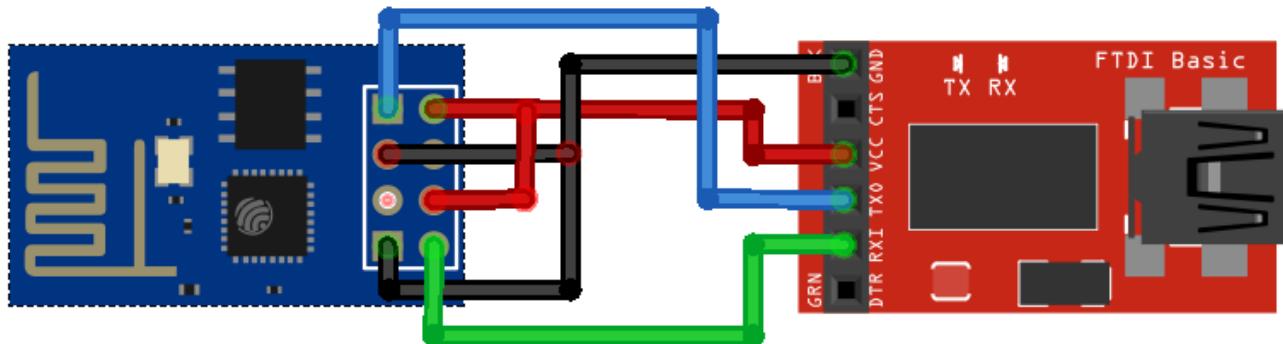
There are different types of FTDI programmers. We recommend the [FT232RL FTDI programmer](#) or an [ESP8266-01 Serial Adapter](#) but you can use any other module.



Important: most FTDI Programmers have a jumper to convert from 5V to 3.3V. Make sure your FTDI Programmer is set to 3.3V operation (as shown in the following figure).



Follow the circuit in the figure below to connect your ESP-01 to your FTDI Programmer to establish a serial communication.



The following table shows the connections between the ESP-01 and the FTDI programmer:

ESP-01	FTDI Programmer
RX	TX
TX	RX
CH_PD	3.3V
GPIO 0	GND
VCC	3.3V
GND	GND

Important: when you want to upload code, GPIO 0 needs to be connected to GND because it requires the ESP8266 to flash a new firmware. In normal usage (if you're not flashing your ESP with a new firmware) it should be connected to VCC.

Download FTDI drivers

If you have a brand new FTDI Programmer and you need to install your FTDI drivers on Windows PC, visit the website in the following link to download the official drivers:

- <http://www.ftdichip.com/Drivers/VCP.htm>

Unbricking the FTDI Programmer on Windows PC

If you're having trouble installing the FTDI drivers on Windows 7/8/8.1/10 it's very likely that FTDI is bricked. Watch the video tutorial in the following link to fix that:

- <http://youtu.be/SPdSKT6KdF8>

In the previous video, it is said that you need to download the drivers from the FTDI website. Read carefully the video description to find all the links. The following link provides the drivers you need:

- <http://www.ftdichip.com/Drivers/CDM/CDM%20v2.12.00%20WHQL%20Certified.zip>

Preparing your Arduino IDE

Once you have your ESP8266+FTDI Programmer connected to your computer, open the Arduino IDE. Go to the **Tools** menu, select the **Board: "Generic ESP8266 Module"**.

Go to **Tools** ▶ **Port** and select the COM Port the ESP8266 is connected to.

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.

```

1 // blink LED | Arduino 1.8.9
2 File Edit Sketch Tools Help
3 
4 
5 
6 int pin = 2;
7 
8 void setup() {
9   pinMode(pin, OUTPUT);
10 }
11 
12 void loop() {
13   digitalWrite(pin, HIGH);
14   delay(1000);
15   digitalWrite(pin, LOW);
16   delay(1000);
17 }
18

```

Done uploading.

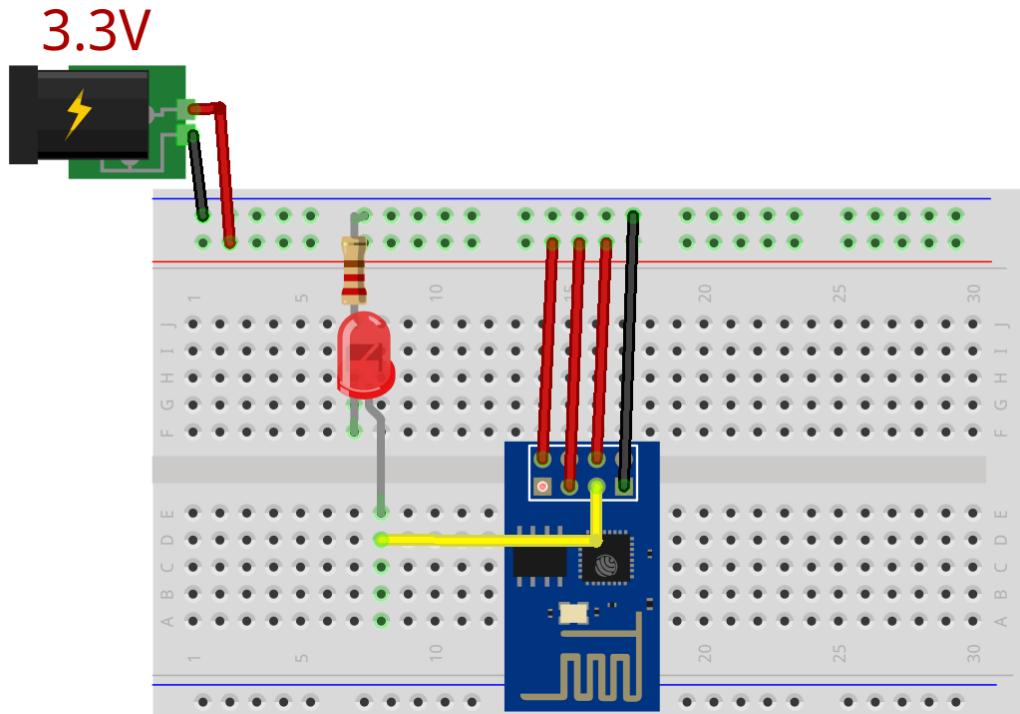
Leaving...

Hard resetting via RTS pin...

18 NodeMCU 1.0 (ESP-12E Module) on COM7

Final ESP-01 circuit

After uploading the code to the module, unplug it from your computer. Then, change the wiring to match the following diagram.

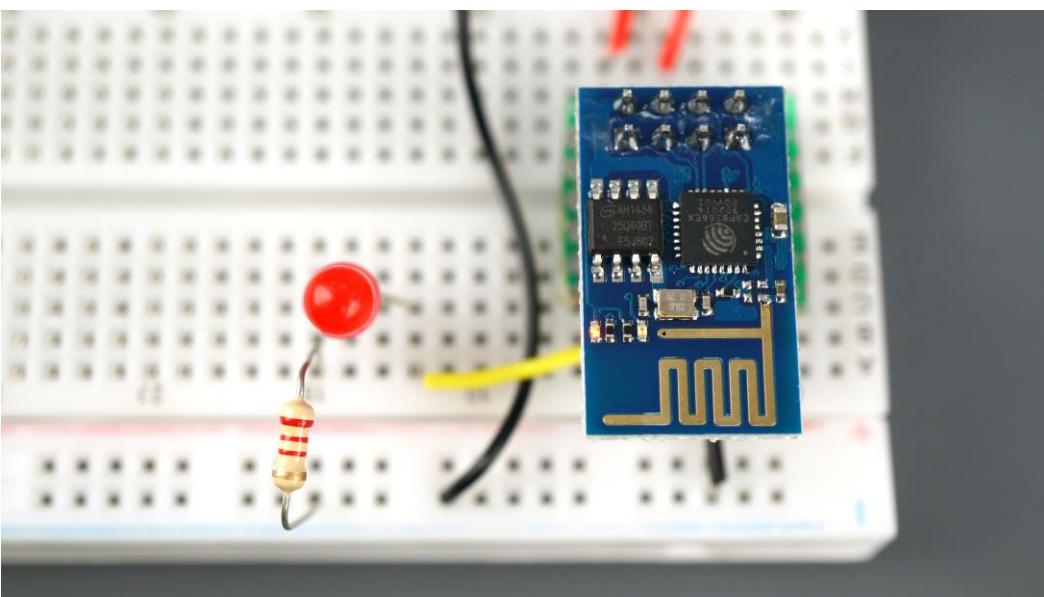
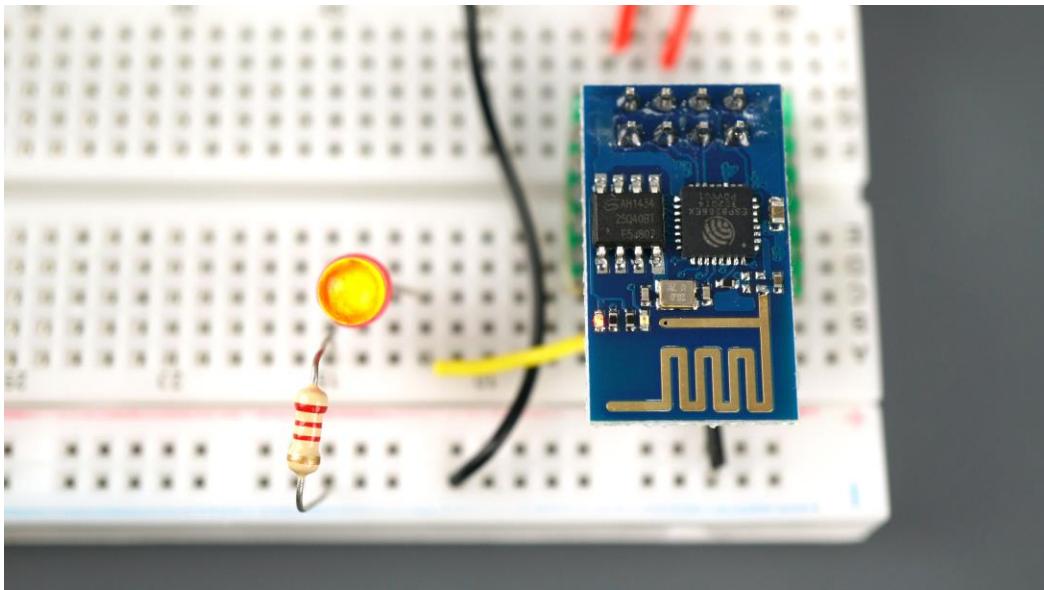


In this case, you need to connect a physical LED to GPIO 2 using a 220 or 330 Ohm resistor.

Apply power from a 3.3V source to the ESP-01 (for testing purposes, it can be the 3.3V pin from your FTDI programmer).

Demonstration

After uploading the code, building the circuit and applying power, your LED should start blinking.

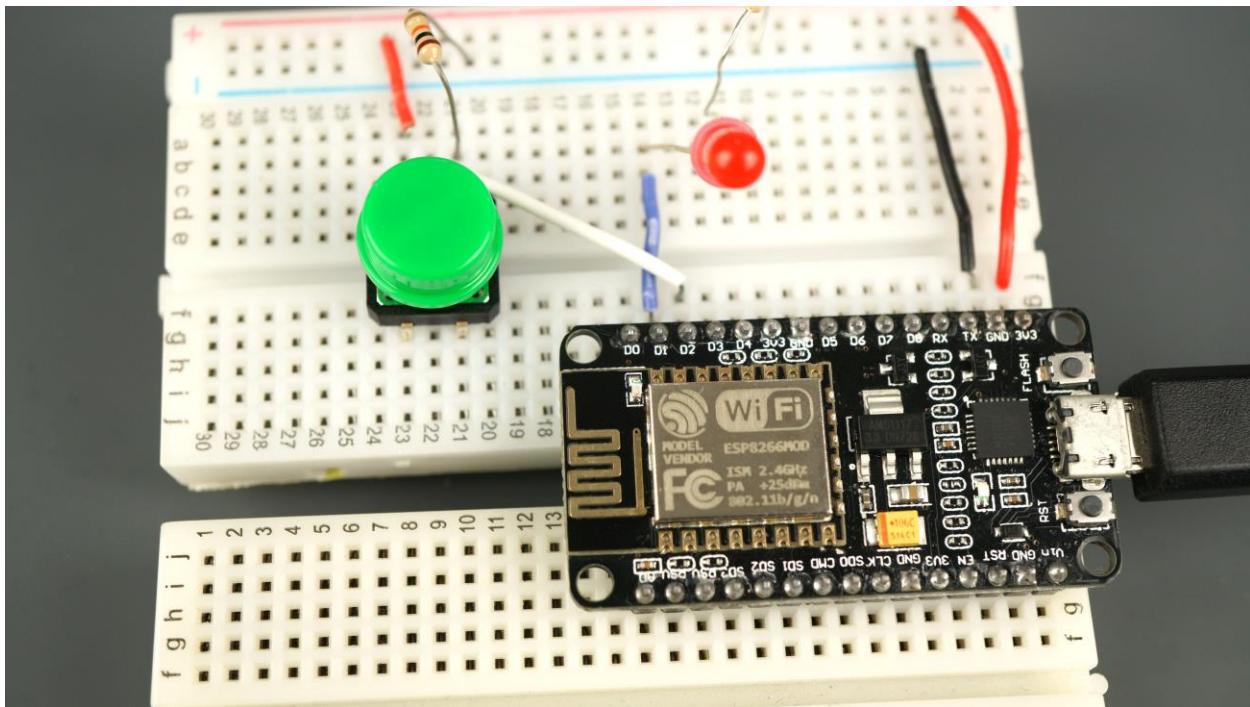


MODULE 2:

Interacting with GPIOs

In this Module, you'll learn how to interact with the ESP8266 GPIOs: digital inputs and outputs, analog inputs, PWM, interrupts and more. There's also a Unit about saving power using deep sleep.

Unit 1: Digital Inputs and Digital Outputs



In this section, we're going to show you how to read digital inputs like a button switch, and how to control a digital output, like an LED.

digitalWrite()

To control a digital output, use the `digitalWrite()` function, that accepts as arguments, the GPIO you are referring to, and the state, either `HIGH` or `LOW`.

```
digitalWrite(GPIO, STATE);
```

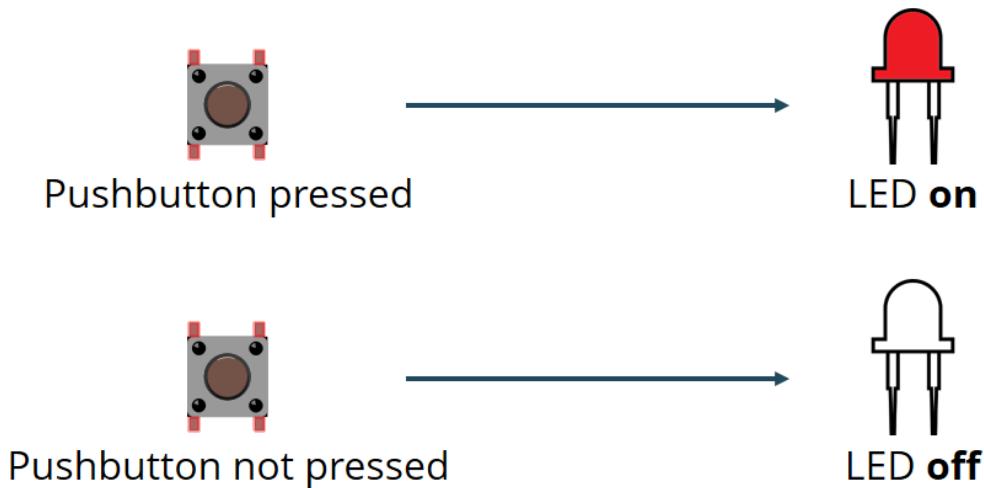
digitalRead()

To read a digital input, like a button, use the `digitalRead()` function, that accepts as argument, the GPIO you are referring to.

```
digitalRead(GPIO);
```

Project Example

Let's just make a simple example to see how these functions work with the ESP8266 using Arduino IDE. In this example, you'll read the state of a pushbutton, and light up an LED accordingly.



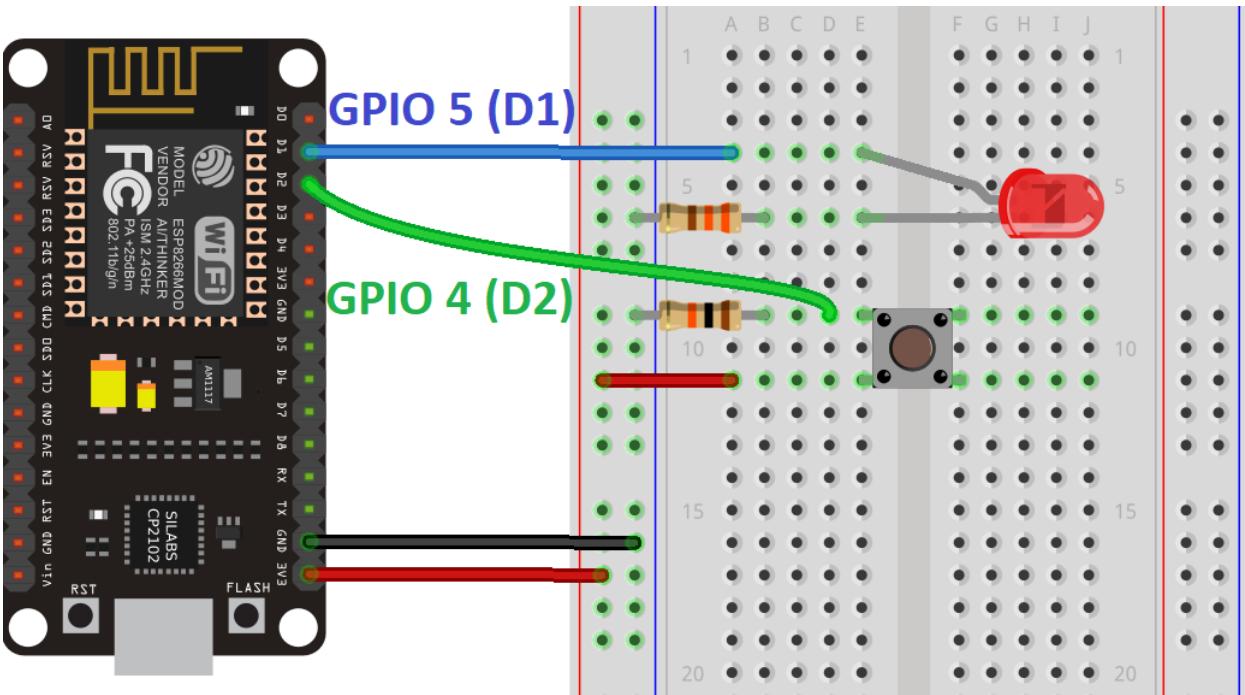
Schematic

Here's a list of the parts needed to build the circuit:

- [ESP8266](#)
- [5mm LED](#)
- [330 Ohm resistor](#)
- [Pushbutton](#)
- [10k Ohm resistor](#)
- [Breadboard](#)
- [Jumper wires](#)

Assemble a circuit with a pushbutton and an LED as shown in the following schematic diagram:

- LED: GPIO 5 (D1)
- Pushbutton: GPIO 4 (D2)



Code

With the circuit ready, copy the code below to your Arduino IDE.

```

const int buttonPin = 4;          // the number of the pushbutton pin
const int ledPin = 5;            // the number of the LED pin
// variable for storing the pushbutton status
int buttonState = 0;

void setup() {
  Serial.begin(115200);
  // initialize the pushbutton pin as an input
  pinMode(buttonPin, INPUT);
  // initialize the LED pin as an output
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the state of the pushbutton value
  buttonState = digitalRead(buttonPin);
  Serial.println(buttonState);
  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH
  if (buttonState == HIGH) {
    // turn LED on
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off
    digitalWrite(ledPin, LOW);
  }
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module2/Unit1/Pushbutton_LED/Pushbutton_LED.ino

How the code works

In the following two lines, create variables to assign pins:

```
const int buttonPin = 4;  
const int ledPin = 5;
```

The button is connected to GPIO 4 and the LED is connected to GPIO 5. When using the Arduino IDE with the ESP8266, 4 corresponds to GPIO 4 and 5 corresponds to GPIO 5.

Next, create a variable to hold the button state.

```
int buttonState = 0;
```

In the `setup()`, initialize the button as an INPUT, and the LED as an OUTPUT. For that, use the `pinMode()` function that accepts the pin you are referring to, and the state: either INPUT or OUTPUT.

```
pinMode(buttonPin, INPUT);  
pinMode(ledPin, OUTPUT);
```

In the `loop()` is where you read the button state and set the LED accordingly.

The next line reads the button state and saves it in the `buttonState` variable. As we've seen previously, you use the `digitalRead()` function.

```
buttonState = digitalRead(buttonPin);
```

The following if statement, checks whether the button state is HIGH. If it is, it turns the LED on using the `digitalWrite()` function that accepts as argument the `ledPin`, and the state HIGH.

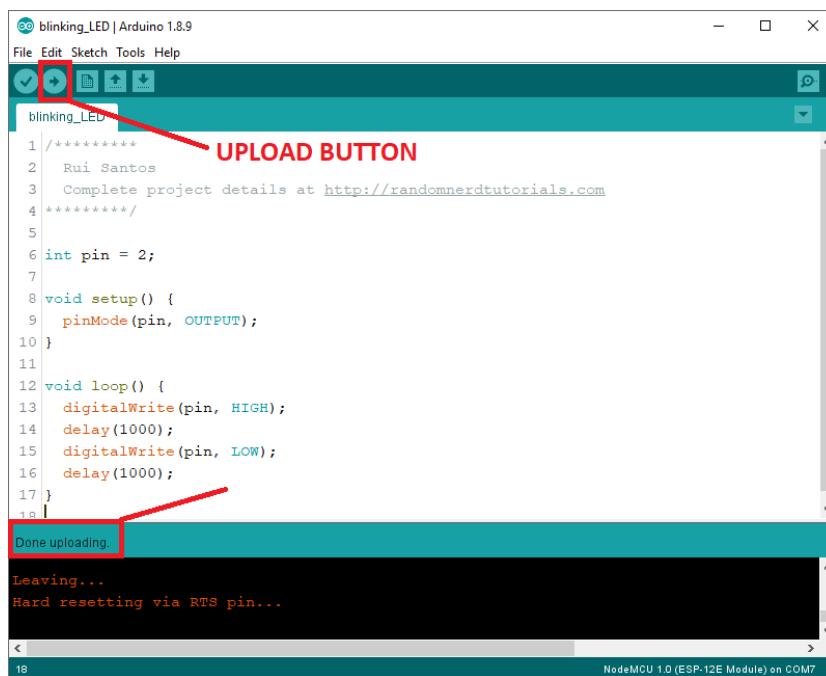
```
if (buttonState == HIGH) {  
    // turn LED on  
    digitalWrite(ledPin, HIGH);  
}
```

If the button state is not HIGH, you set the LED off, by writing LOW in the `digitalWrite()` function.

```
else {  
    // turn LED off  
    digitalWrite(ledPin, LOW);  
}
```

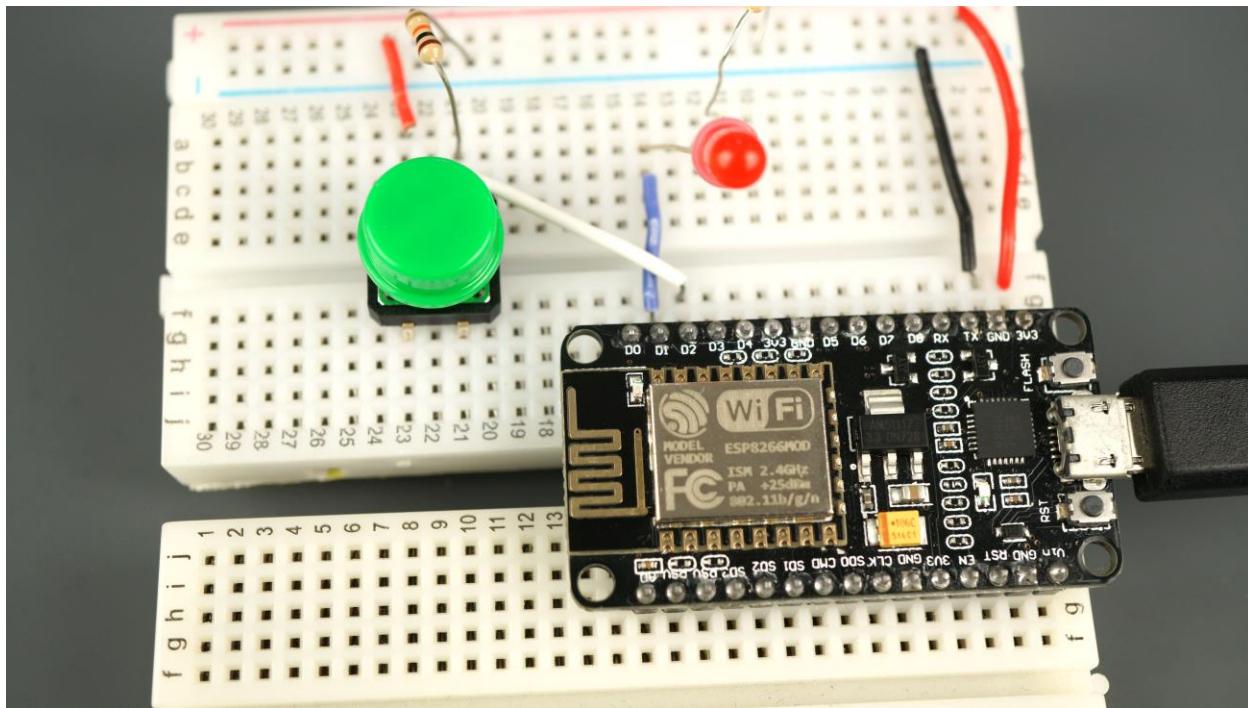
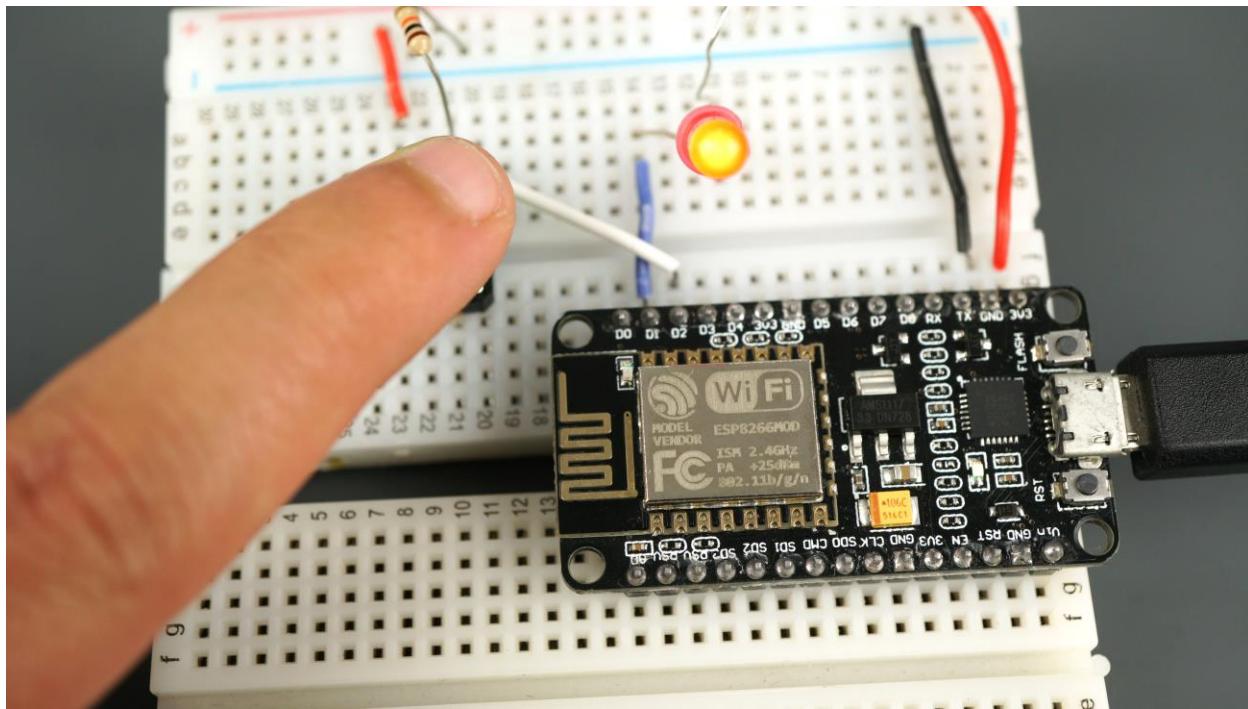
Uploading the code

Go to the **Tools** menu and select the “**NodeMCU 1.0 (ESP-12E Module)**” board or any other board that you’re using. Go to **Tools** ▶ **Port** and select the COM Port the ESP8266 is connected to. 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.

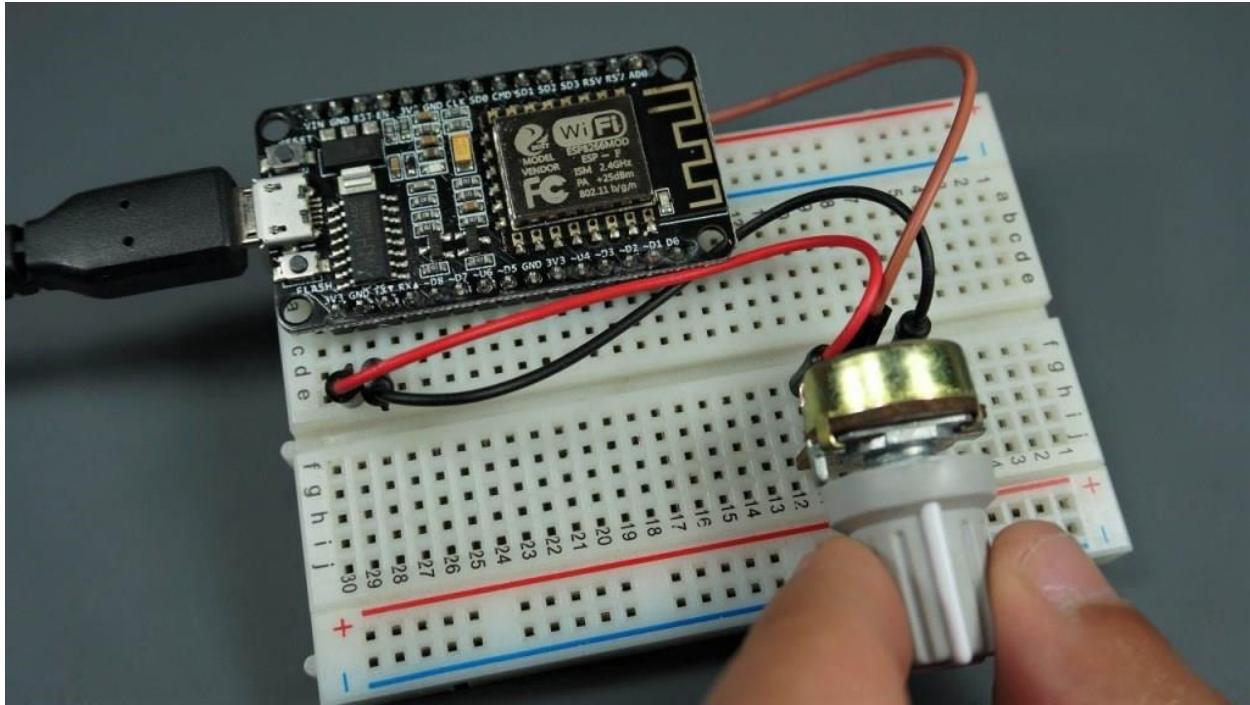


Demonstration

After uploading the code, test your circuit. The LED should light up when you press the button and stays off when you release it.



Unit 2: Analog Inputs



In this Unit, you'll learn how to read an analog input. This is useful to read values from variable resistors like potentiometers, or analog sensors.

Analog Pins

The ESP-01 doesn't offer any analog input pins. On the other hand, other versions such as the ESP8266-12E or ESP8266-07 have one ADC pin that is easily accessible.

As an example, we'll show you how to read analog values from a potentiometer.

ESP8266 ADC Specifications

When referring to the ESP8266 ADC pin you will often hear these different terms interchangeably: ADC0 (Analog-to-digital Converter); TOUT; Pin 6; A0; Analog Pin 0; ...

All these terms refer to the same pin in the ESP8266 that is highlighted in the next section.

ESP8266 ADC Resolution

The ADC pin has a 10-bit resolution, which means you'll get values between 0 and 1024.

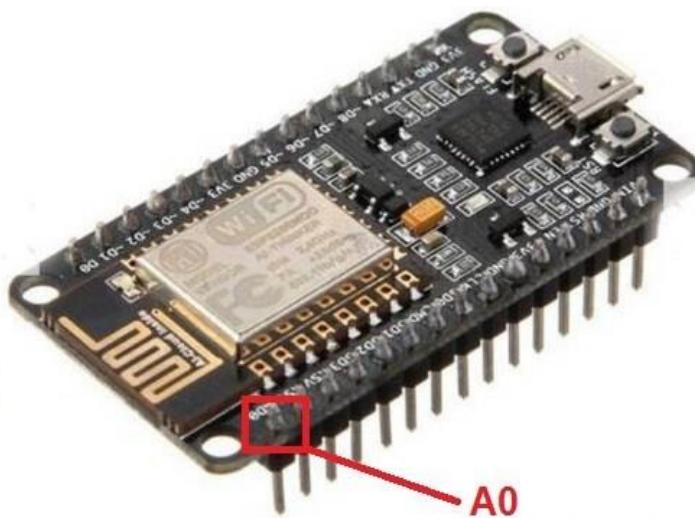
ESP8266 Input Voltage Range

The ESP8266 ADC pin input voltage range is 0 to 1V if you're using the bare chip. However, most ESP8266 development boards come with an internal voltage divider, so the input range is 0 to 3.3V. So, in summary:

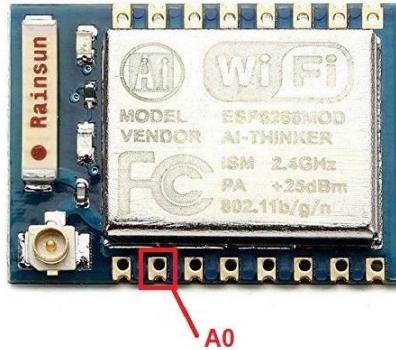
- ADC Voltage range in ESP8266 development boards: 0 to 3.3V (for example: [ESP8266-12E NodeMCU Kit](#), [WeMos D1 Mini](#), ...)
- ADC Voltage range in ESP8266 chip: 0 to 1V (for example: [ESP-07 chip](#), [ESP-12E chip](#), ...)

ESP8266 Analog Pin

With the ESP8266-12E NodeMCU kit and other ESP8266 development boards, it is very easy to access the A0, you simply connect a jumper wire to the pin (see figure below).



If you're using an ESP8266 chip, like the ESP8266-07, you need to solder a wire to the A0 pin.



analogRead()

Reading an analog input with the ESP8266 using the Arduino IDE is as simple as using the `analogRead()` function, that accepts as argument, the GPIO you want to read. In this case, the only GPIO is A0.

```
analogRead(A0);
```

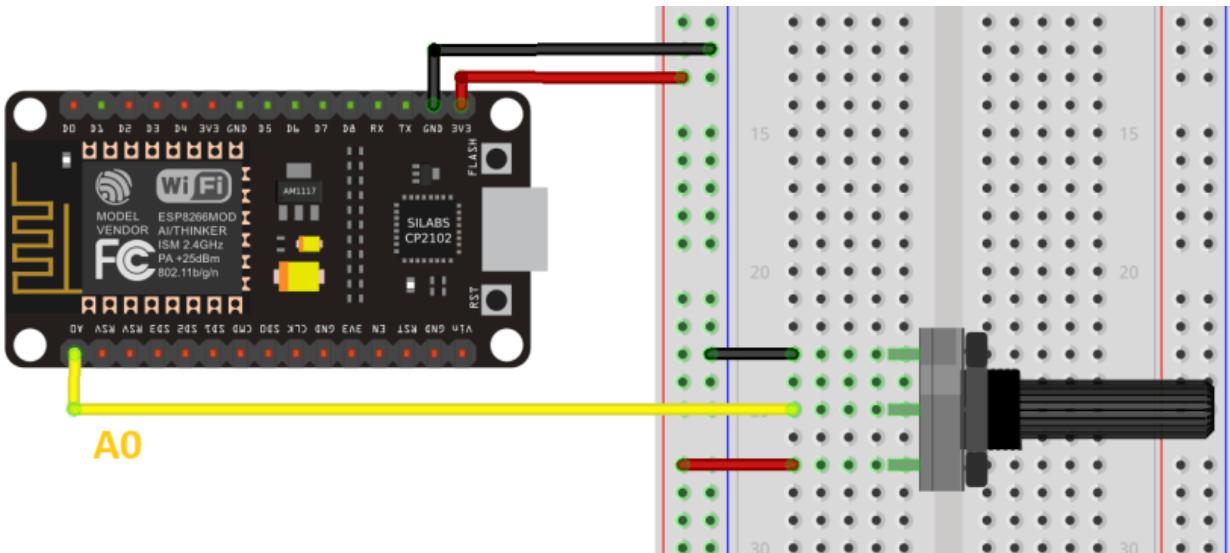
Schematic

To show you how to use analog reading with the ESP8266, we'll read the values from a potentiometer. For that, you need to wire a potentiometer to your board.

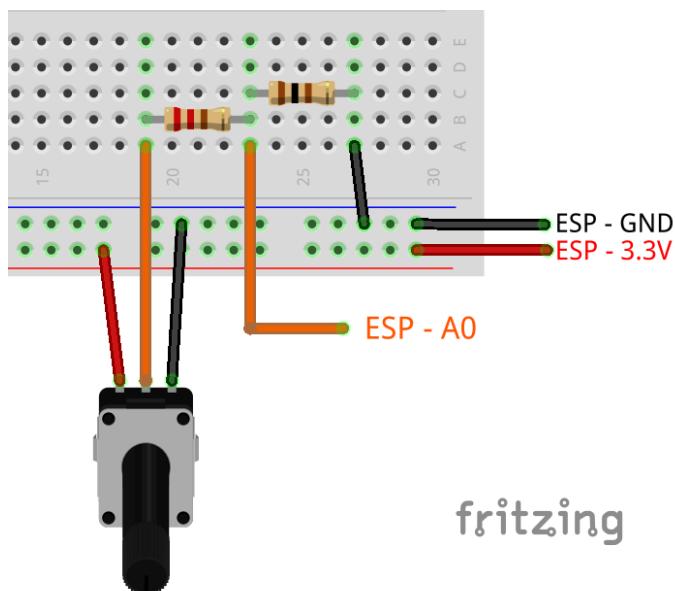
Here's the hardware that you need to complete this tutorial:

- [ESP8266](#)
- [1k Ohm Potentiometer](#)
- [Breadboard](#)
- [Jumper wires](#)
- [100 Ohm Resistor](#) (needed if you're using a bare chip)
- [220 Ohm Resistor](#) (needed if you're using a bare chip)

If you're using an ESP8266 development board, follow the next schematic diagram.



If you're using an [ESP8266 chip](#) with input voltage range of 0V to 1V, you need to make sure that the input voltage on the A0 pin doesn't exceed 1V. So, you need a voltage divider circuit, as shown below.



We're using a 100 Ohm and a 220 Ohm resistor, so that the Vout is 1V.

$$V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2}$$

Code

The following script for the ESP8266 reads analog values from the ADC0 pin. Copy the following code to your Arduino IDE.

```
const int analogInPin = A0; // ESP8266 Analog Pin ADC0 = A0

int sensorValue = 0; // value read from the pot

void setup() {
    // initialize serial communication at 115200
    Serial.begin(115200);
}

void loop() {
    // read the analog in value
    sensorValue = analogRead(analogInPin);

    // print the readings in the Serial Monitor
    Serial.print("sensor = ");
    Serial.print(sensorValue);

    delay(1000);
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module2/Unit2/analogRead/analogRead.ino

How the code works

The code starts by declaring the ESP8266 analog pin in the `analogInPin` variable:

```
const int analogInPin = A0; // ESP8266 Analog Pin ADC0 = A0
```

The potentiometer value will be stored on the `sensorValue` variable:

```
int sensorValue = 0; // value read from the pot
```

In the `setup()`, initialize the Serial Monitor to display the values read from the potentiometer:

```
void setup() {
    // initialize serial communication at 115200
    Serial.begin(115200);
}
```

In the `loop()`, read the analog value by using the `analogRead()` function. Pass the `analogInPin` variable as an argument. The result is saved on the `sensorValue` variable:

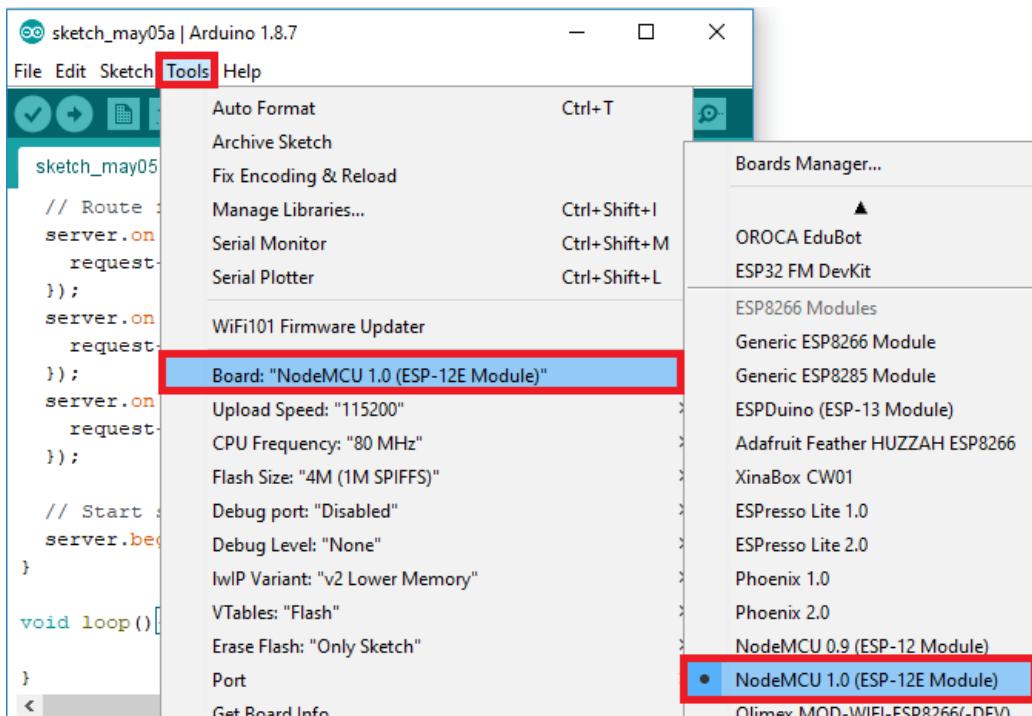
```
sensorValue = analogRead(analogInPin);
```

Finally, the readings are displayed on the Serial Monitor, so that you can actually see what is going on.

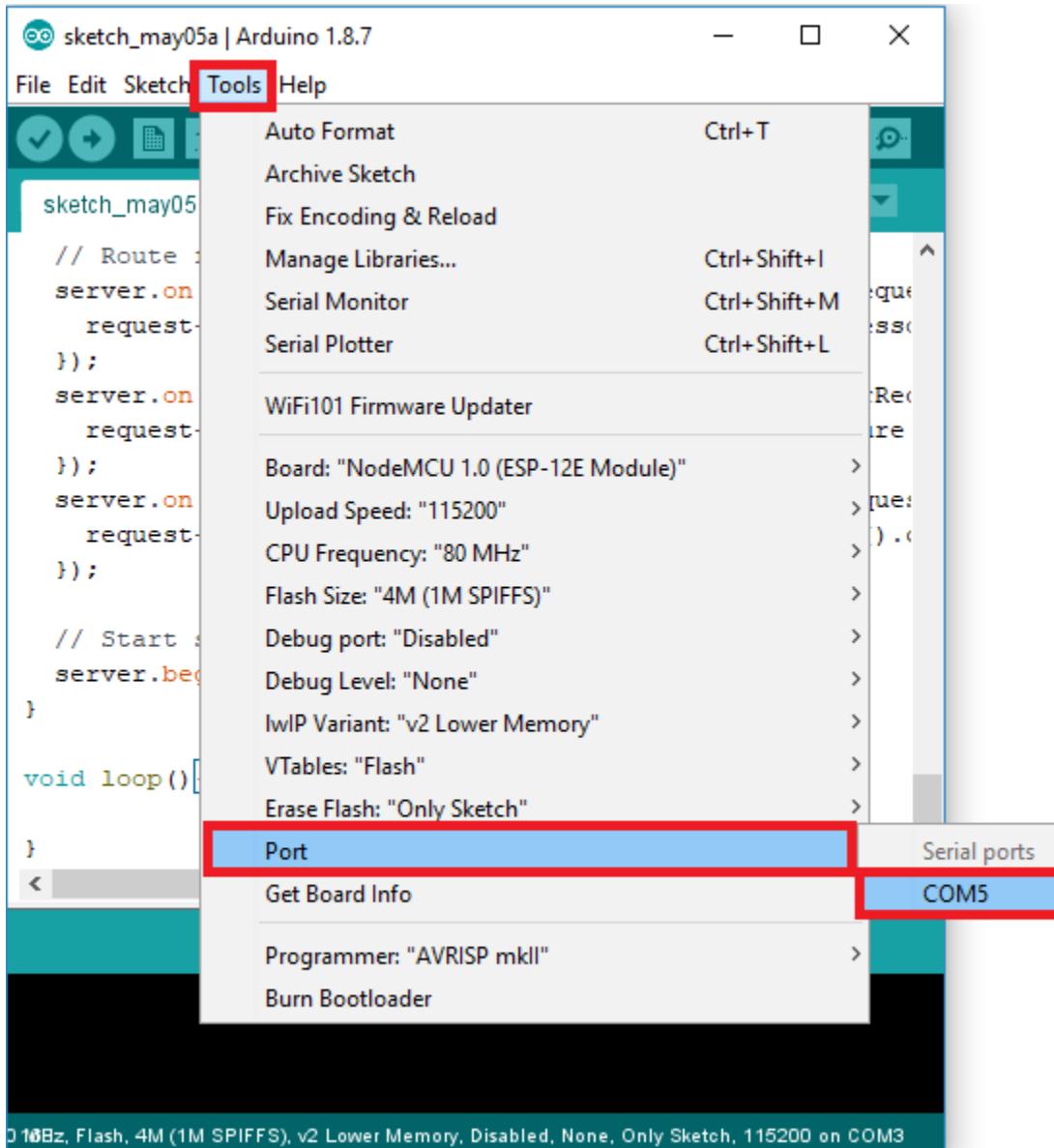
```
Serial.print("sensor = ");
Serial.print(sensorValue);
```

Uploading the Code

Upload the previous code to the ESP8266. Make sure you have the right board and COM port select. Go to **Tools** ▶ **Board** and select the ESP8266 model you're using. In our case, we're using the ESP8266 12-E NodeMCU Kit.



Go to **Tools** ▶ **Port** and select the COM port the ESP8266 is connected to.



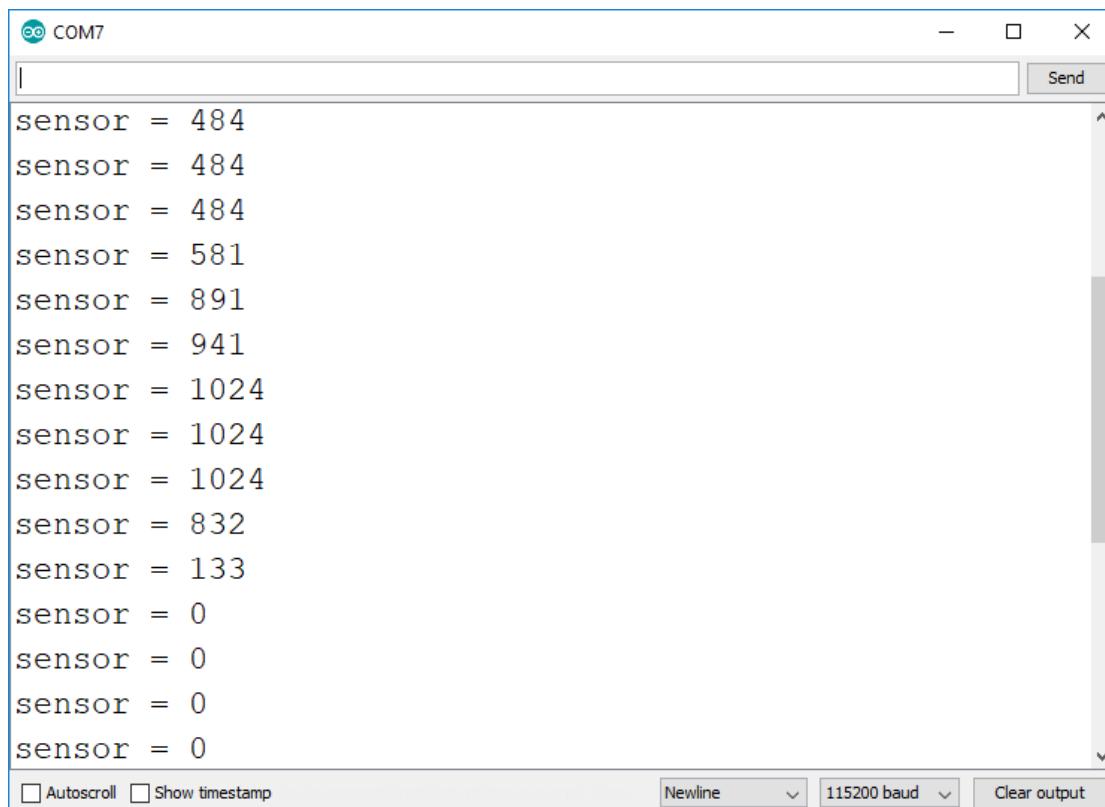
Press the Arduino IDE upload button.

Note: if you're using an ESP-07 or ESP-12E chip, you need an [FTDI programmer](#) to upload code.

Demonstration

After uploading the code, open the Serial Monitor at a baud rate of 115200. The analog readings should be displayed.

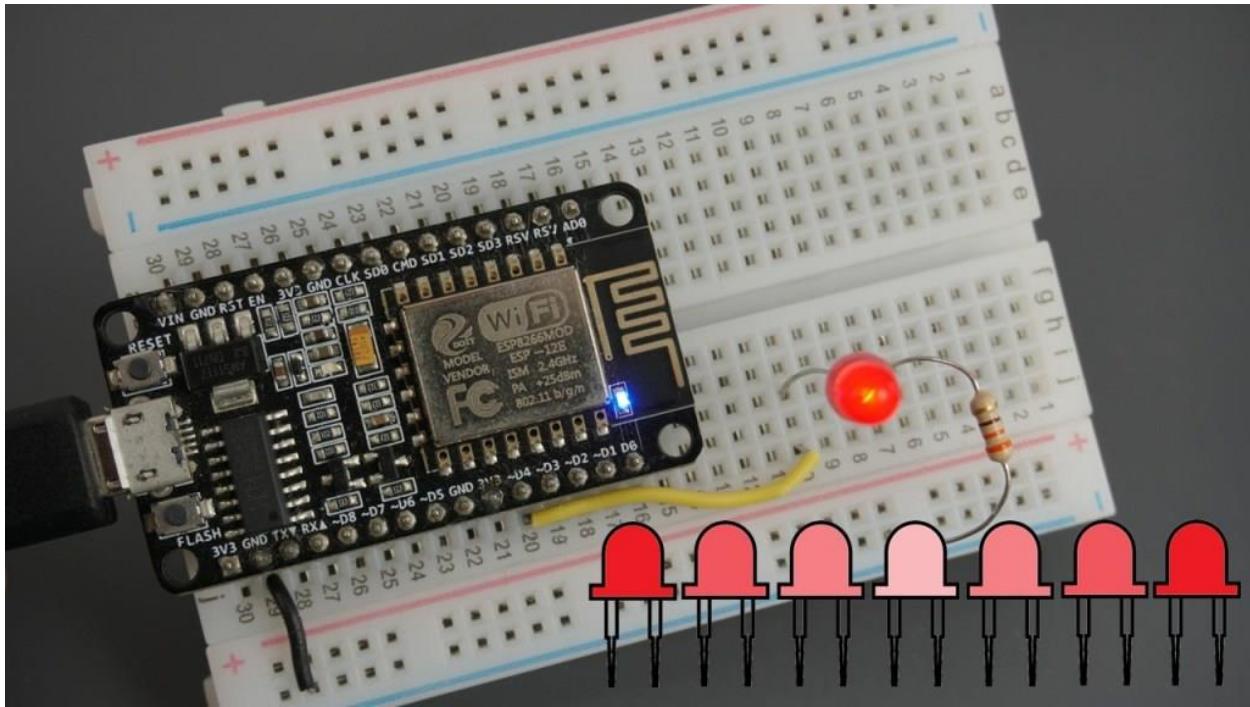
Rotate the potentiometer and see the values increasing or decreasing.



The screenshot shows the Arduino Serial Monitor window titled "COM7". The main text area displays a series of "sensor = <value>" messages. The values fluctuate between 0 and 1024, with several instances of 484, 581, 891, 941, 1024, 832, 133, 0, and one instance of 484. At the bottom of the window, there are three checkboxes: "Autoscroll", "Show timestamp", and "Clear output". To the right of these checkboxes are buttons for "Newline", "115200 baud", and "Send".

```
sensor = 484
sensor = 484
sensor = 484
sensor = 581
sensor = 891
sensor = 941
sensor = 1024
sensor = 1024
sensor = 1024
sensor = 832
sensor = 133
sensor = 0
sensor = 0
sensor = 0
sensor = 0
```

Unit 3: PWM – Pulse Width Modulation

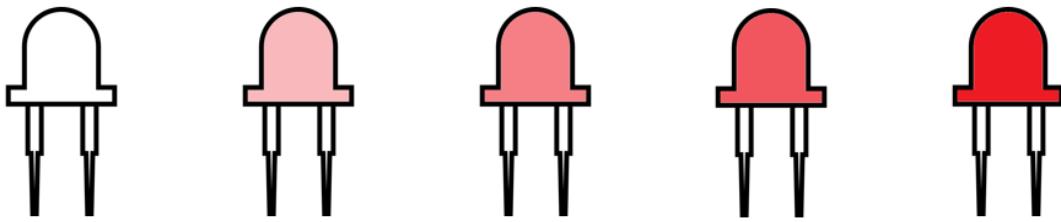


In this Unit, you'll learn how to generate PWM signals with the ESP8266. As an example, we'll dim the LED brightness by changing the duty cycle over time. Besides controlling the LEDs' brightness, PWM can also be used to control the speed of DC motors.

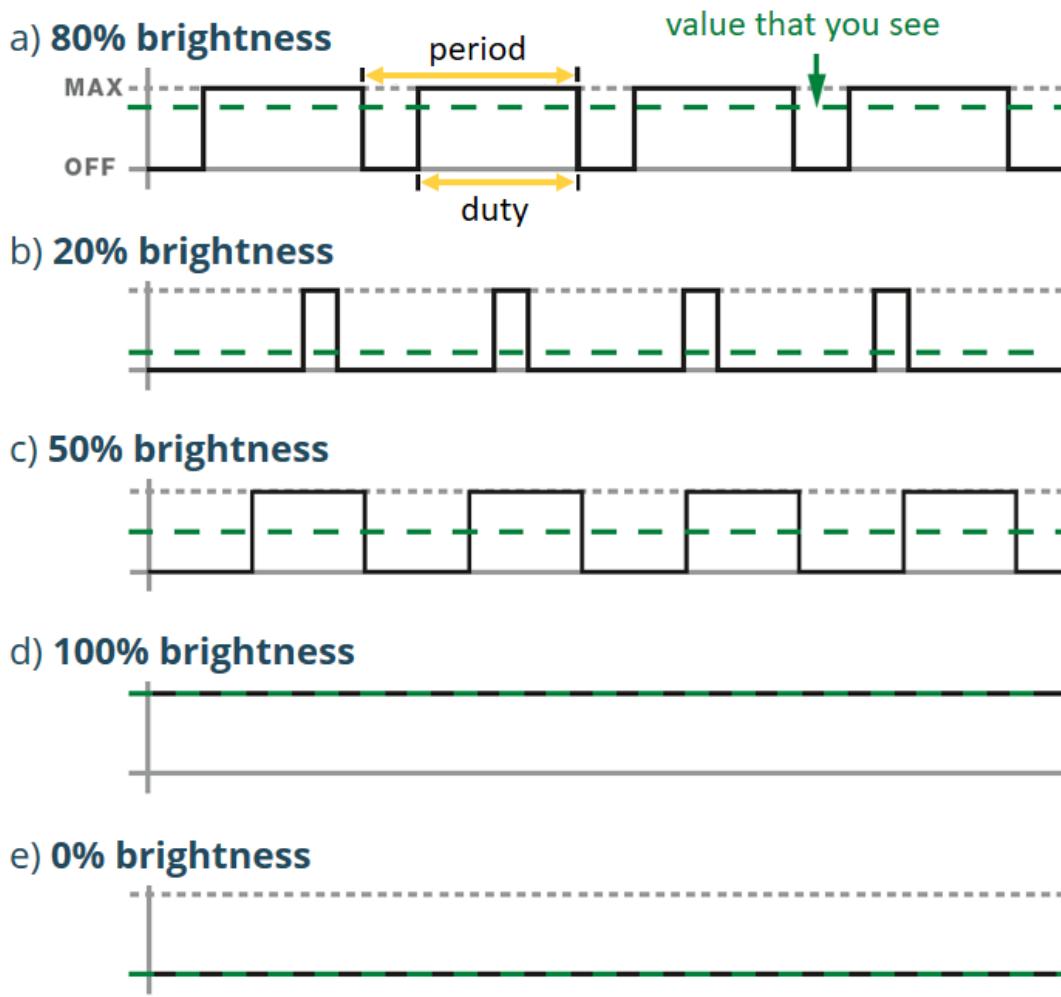
Pulse-Width Modulation

The ESP8266 GPIOs can be set either to output 0V or 3.3V, but they can't output any voltages in between. However, you can output "fake" mid-level voltages using pulse-width modulation (PWM), which is how you'll produce varying levels of LED brightness for this project.

If you alternate an LED's voltage between HIGH and LOW very fast, your eyes can't keep up with the speed at which the LED switches on and off; you'll simply see some gradations in brightness.



That's basically how PWM works—by producing an output that changes between HIGH and LOW at a very high frequency. The duty cycle is the fraction of the time period at which the LED is set to HIGH. The following figure illustrates how PWM works.



For instance, a duty cycle of 50 percent results in 50 percent LED brightness, a duty cycle of 0 means the LED is fully off, and a duty cycle of 100 means the LED is fully on. Changing the duty cycle is how you produce different levels of brightness, and that's what we're going to do here.

analogWrite()

To produce a PWM signal on a given pin you use the `analogWrite(pin, value)` function:

- **pin**: PWM may be used on pins 0 to 16;
- **value**: should be in range from 0 to `PWMRANGE`, which is equal to 1023 by default. When the value is 0, PWM is disable on that pin. A value of 1023 corresponds to 100% duty cycle.

You can change the PWM range by calling:

```
analogWriteRange(new_range);
```

By default, ESP8266 PWM frequency is 1kHz. You can change PWM frequency with:

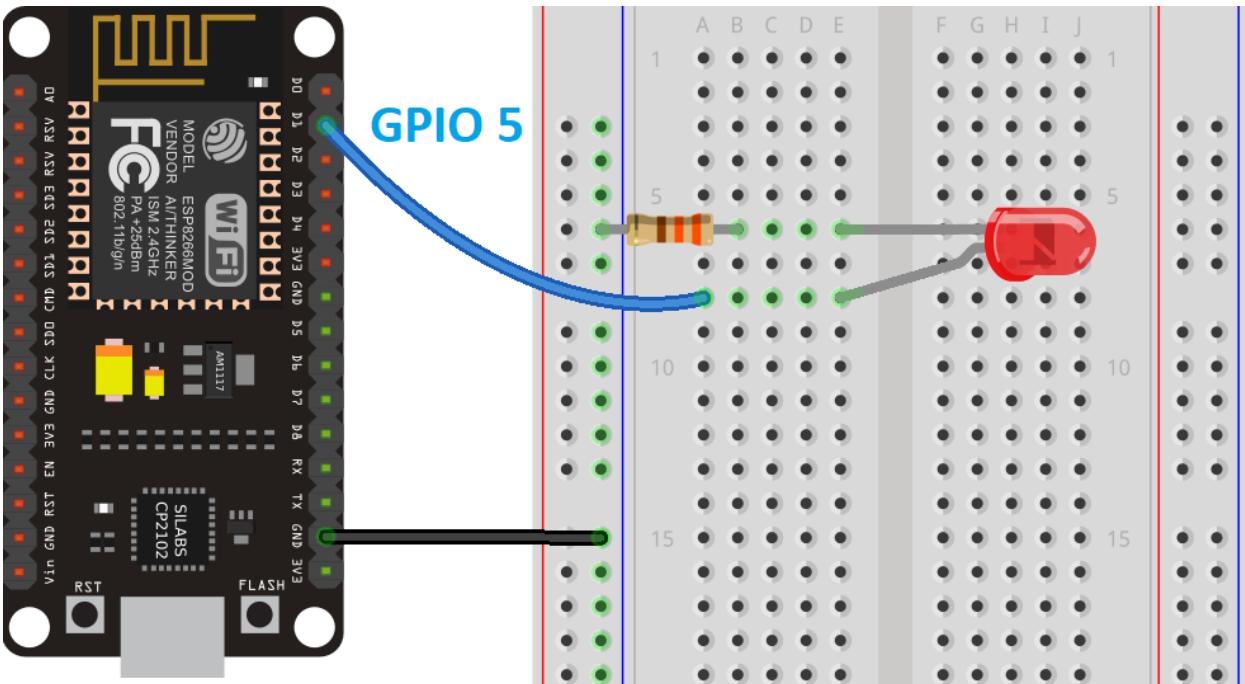
```
analogWriteFreq(new_frequency);
```

Schematic

For this example, wire an LED to your ESP8266 board. We'll wire the LED to GPIO 5, but you can choose another pin.

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

- [ESP8266](#)
- [5mm LED](#)
- [330 Ohm resistor](#)
- [Breadboard](#)
- [Jumper wires](#)



Code

Copy the following code to your Arduino IDE. It changes the LED brightness over time by increasing the duty cycle.

```

const int ledPin = 2;

void setup() {
}

void loop() {
    // increase the LED brightness
    for(int dutyCycle = 0; dutyCycle < 1023; dutyCycle++) {
        // changing the LED brightness with PWM
        analogWrite(ledPin, dutyCycle);
        delay(1);
    }

    // decrease the LED brightness
    for(int dutyCycle = 1023; dutyCycle > 0; dutyCycle--) {
        // changing the LED brightness with PWM
        analogWrite(ledPin, dutyCycle);
        delay(1);
    }
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module2/Unit3/PWM/PWM.ino

How the Code Works

Start by defining the pin the LED is attached to. In this case, the LED is attached to GPIO 2 (D4).

```
const int ledPin = 2;
```

In the `loop()`, you vary the duty cycle between 0 and 1023 to increase the LED brightness.

```
for(int dutyCycle = 0; dutyCycle < 1023; dutyCycle++) {
    // changing the LED brightness with PWM
    analogWrite(ledPin, dutyCycle);
    delay(1);
}
```

And then, between 1023 and 0 to decrease brightness.

```
for(int dutyCycle = 1023; dutyCycle > 0; dutyCycle--) {
    // changing the LED brightness with PWM
    analogWrite(ledPin, dutyCycle);
    delay(1);
}
```

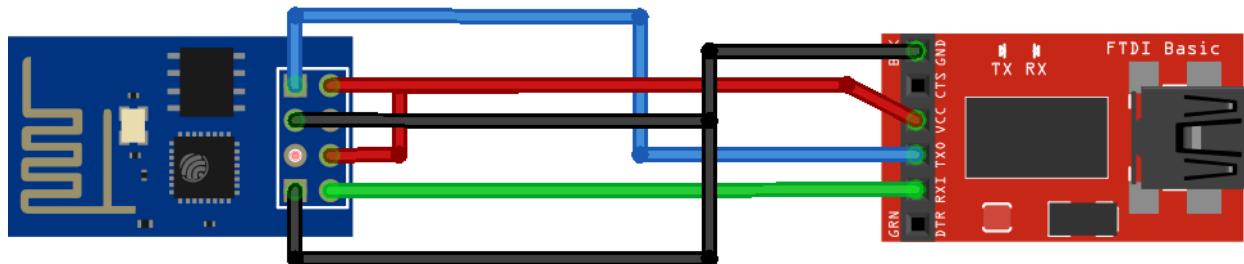
To set the LED brightness, you need to use the `analogWrite()` function that accepts as arguments, the GPIO where you want to get the PWM signal and a value between 0 and 1023 to set the duty cycle.

Upload the Code

In your Arduino IDE, go to **Tools** ▶ **Board** and select your ESP8266 model (If you're using an ESP-01, select "Generic ESP8266 Module").

Go to **Tools** ▶ **Port** and select COM port the ESP8266 is connected to.

If you're using an ESP-01, you need an [FTDI programmer](#) or [Serial Adapter](#) to upload code. Here's the connections you need to make if you're using an ESP-01 with an FTDI programmer:

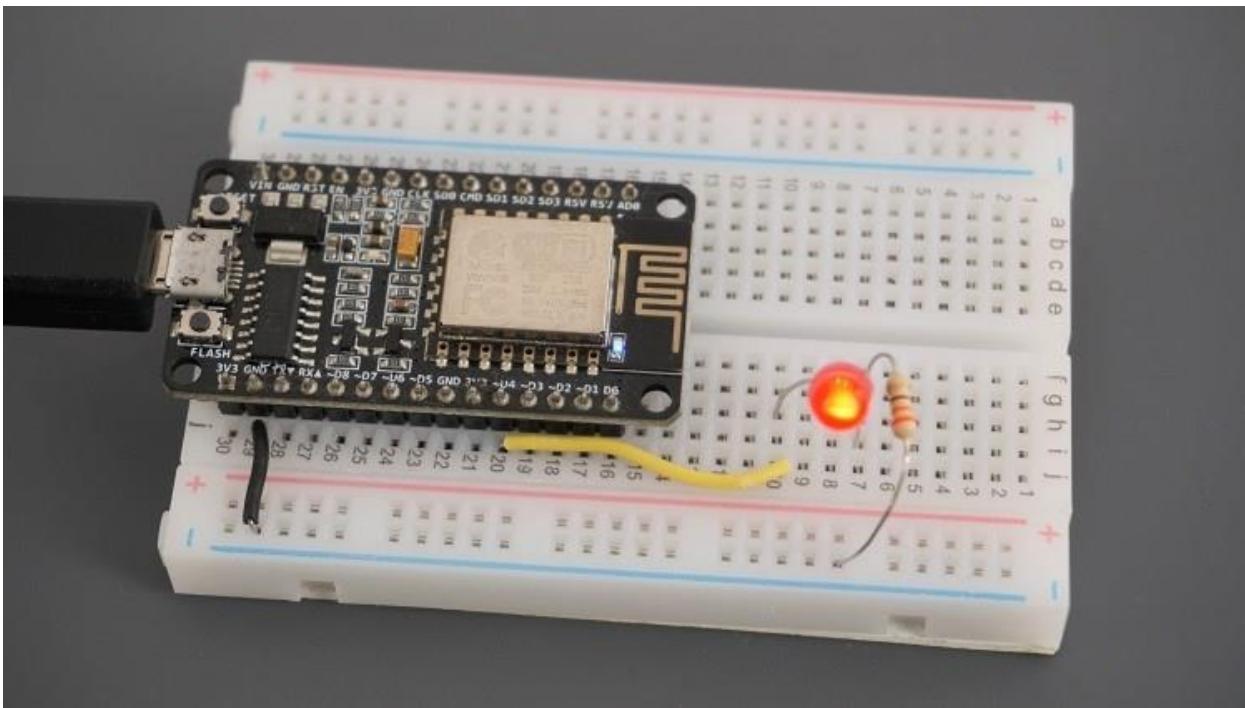


ESP-01	FTDI Programmer
RX	TX
TX	RX
CH_PD	3.3V
GPIO 0	GND
VCC	3.3V
GND	GND

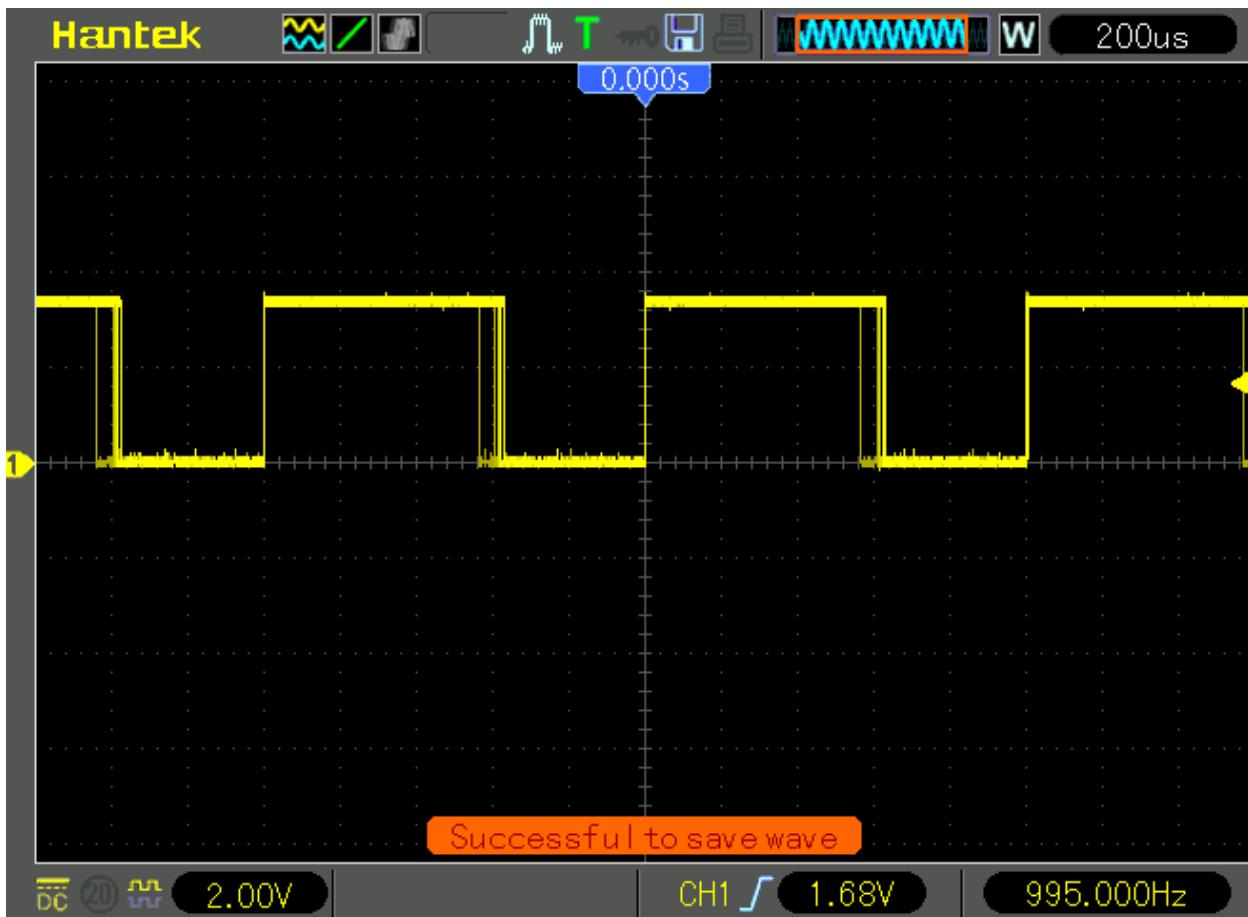
Finally, you can click the upload button.

Demonstration

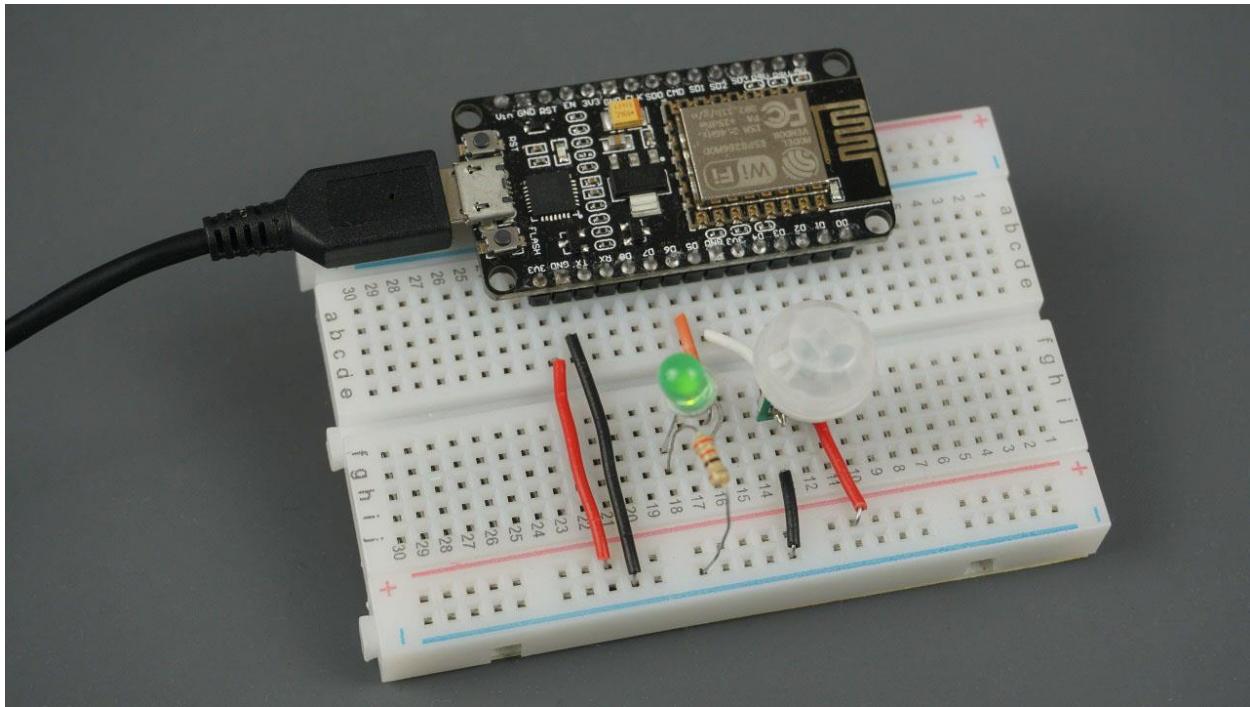
After uploading your sketch, the LED connected to GPIO 2 should increase and decrease its brightness over time.



You can connect GPIO 2 to an [oscilloscope](#) to see how the PWM signal changes over time.



Unit 4: Interrupts and Timers



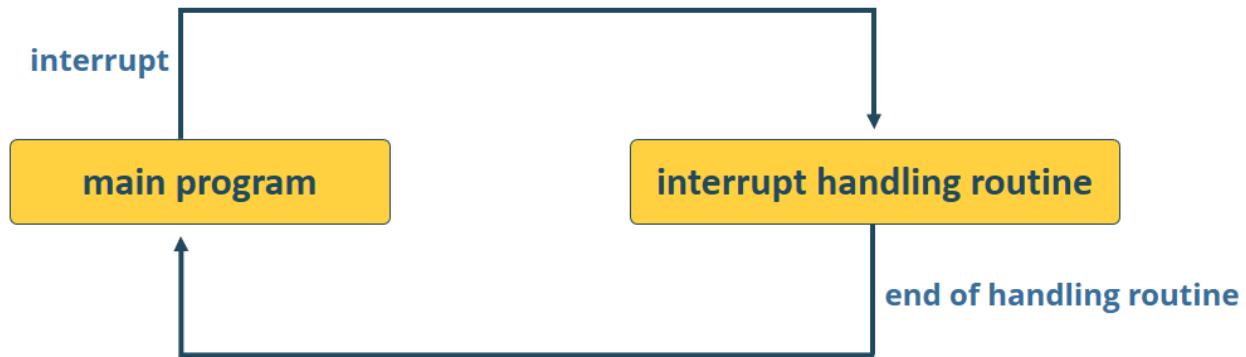
In this Unit, you'll learn how to use interrupts with the ESP8266. Interrupts allow you to detect changes in the state of a pin without the need to constantly check its current value. With interrupts, when a change is detected, an event is triggered (a function is called).

As an example, we'll detect motion using a PIR motion sensor: when motion is detected, the ESP8266 starts a timer and turns an LED on for a predefined number of seconds. When the timer finishes counting down, the LED is automatically turned off. With this example we'll introduce two new concepts: interrupts and timers.

Introducing Interrupts

Interrupts are useful for making things happen automatically in microcontroller programs and can help solve timing problems. With interrupts you don't need to constantly check the current pin value. When a change is detected, an event is triggered - a function is called. This function is called interrupt service routine (ISR).

When an interrupt happens, the processor stops the execution of the main program to execute a task, and then gets back to the main program as shown in the figure below.



This is especially useful to trigger an action whenever motion is detected or whenever a pushbutton is pressed without the need to constantly check its state.

attachInterrupt()

To set an interrupt in the Arduino IDE, you use the `attachInterrupt()` function, that accepts as arguments: the GPIO interrupt pin, the name of the function to be executed, and mode:

```
attachInterrupt(digitalPinToInterrupt(GPIO), ISR, mode);
```

GPIO Interrupt

The first argument is a GPIO interrupt.

You should use `digitalPinToInterrupt(GPIO)` to set the actual GPIO as an interrupt pin. For example, if you want to use GPIO 14 as an interrupt, use:

```
digitalPinToInterrupt(14)
```

The ESP8266 supports interrupts in any GPIO, except GPIO 16.

ISR

The second argument of the `attachInterrupt()` function is the name of the function that will be called every time the interrupt is triggered – the interrupt service routine (ISR).

The ISR function should be as simple as possible, so the processor gets back to the execution of the main program quickly.

The best approach is to signal the main code that the interrupt has happened by using a global variable and then from within the `loop()`, check and clear that flag, and execute code.

ISRs need to have `ICACHE_RAM_ATTR` before the function definition to run the interrupt code in RAM.

Mode

The third argument is the mode. There are 3 different modes:

- **CHANGE**: to trigger the interrupt whenever the pin changes value – for example from HIGH to LOW or LOW to HIGH;
- **FALLING**: for when the pin goes from HIGH to LOW.
- **RISING**: to trigger when the pin goes from LOW to HIGH;

For our example, we'll be using the RISING mode, because when the PIR motion sensor detects motion, the GPIO it is connected to goes from LOW to HIGH.

Introducing Timers



For this project, we'll also be introducing timers. We want the LED to stay on for a predetermined number of seconds after motion is detected. Instead of using a `delay()` function that blocks your code and doesn't allow you to do anything else for a determined number of seconds, we'll use a timer.

The `delay()` function

Until now, we've used the `delay()` function that is pretty straightforward. It accepts a single int number as an argument. This number represents the time in milliseconds the program has to wait until moving on to the next line of code.

```
delay(time in milliseconds);
```

When you call `delay(1000)`, your program stops on that line for 1 second.

`delay()` is a blocking function. Blocking functions prevent a program from doing anything else until that particular task is completed. If you need multiple tasks to occur at the same time, you cannot use `delay()`.

For most projects you should avoid using delays and use timers instead.

The millis() function

Using a function called `millis()`, you can return the number of milliseconds that have passed since the program first started.

```
millis();
```

Why is that function useful? Because by using some math, you can easily verify how much time has passed without blocking your code.

Blinking an LED with millis()

The following snippet of code shows how you can use the `millis()` function to create a blink project. It turns an LED on for 1000 milliseconds, and then turns it off.

```
const int ledPin = 2;           // the number of the LED pin

int ledState = LOW;

// You should use "unsigned long" for variables that hold time
// The value will quickly become too large for an int to store

// will store last time LED was updated
unsigned long previousMillis = 0;

// interval at which to blink (milliseconds)
const long interval = 1000;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // here is where you'd put code that needs to be running all the time.

  // check to see if it's time to blink the LED; that is, if the
  // difference between the current time and last time you blinked
  // the LED is bigger than the interval at which you want to
  // blink the LED.
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    // save the last time you blinked the LED
    previousMillis = currentMillis;

    // if the LED is off turn it on and vice-versa:
  }
}
```

```

    if (ledState == LOW) {
        ledState = HIGH;
    } else {
        ledState = LOW;
    }

    // set the LED with the ledState of the variable:
    digitalWrite(ledPin, ledState);
}
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module2/Unit4/Blink%20Without%20Delay/Blink%20Without%20Delay.ino

How the Code Works

Let's take a closer look at this blink sketch that works without a `delay()` function - it uses the `millis()` function instead.

Basically, this code subtracts the previous recorded time (`previousMillis`) from the current time (`currentMillis`). If the remainder is greater than the `interval` (in this case, 1000 milliseconds), the program updates the `previousMillis` variable to the current time, and either turns the LED on or off.

```

if (currentMillis - previousMillis >= interval) {
    // save the last time you blinked the LED
    previousMillis = currentMillis;

    // if the LED is off turn it on and vice-versa:
    if (ledState == LOW) {
        ledState = HIGH;
    } else {
        ledState = LOW;
    }

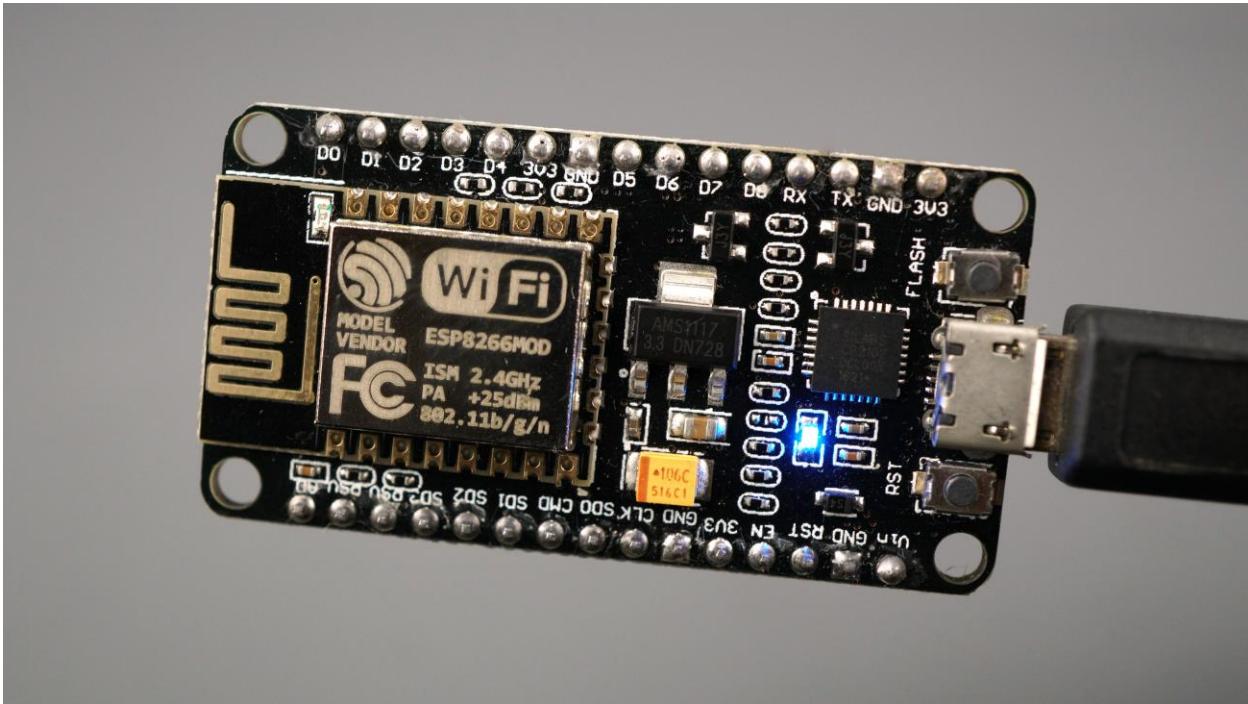
    // set the LED with the ledState of the variable:
    digitalWrite(ledPin, ledState);
}

```

Because this snippet is non-blocking, any code that's located outside of that first `if` statement should work normally.

Now you understand that you can add other tasks to your `loop()` function and your code will still be blinking the LED every one second.

You can upload this code to your ESP8266 and see the on-board LED blinking every second. You can also modify the `interval` variable to see how it works.



ESP8266 with PIR Motion Sensor

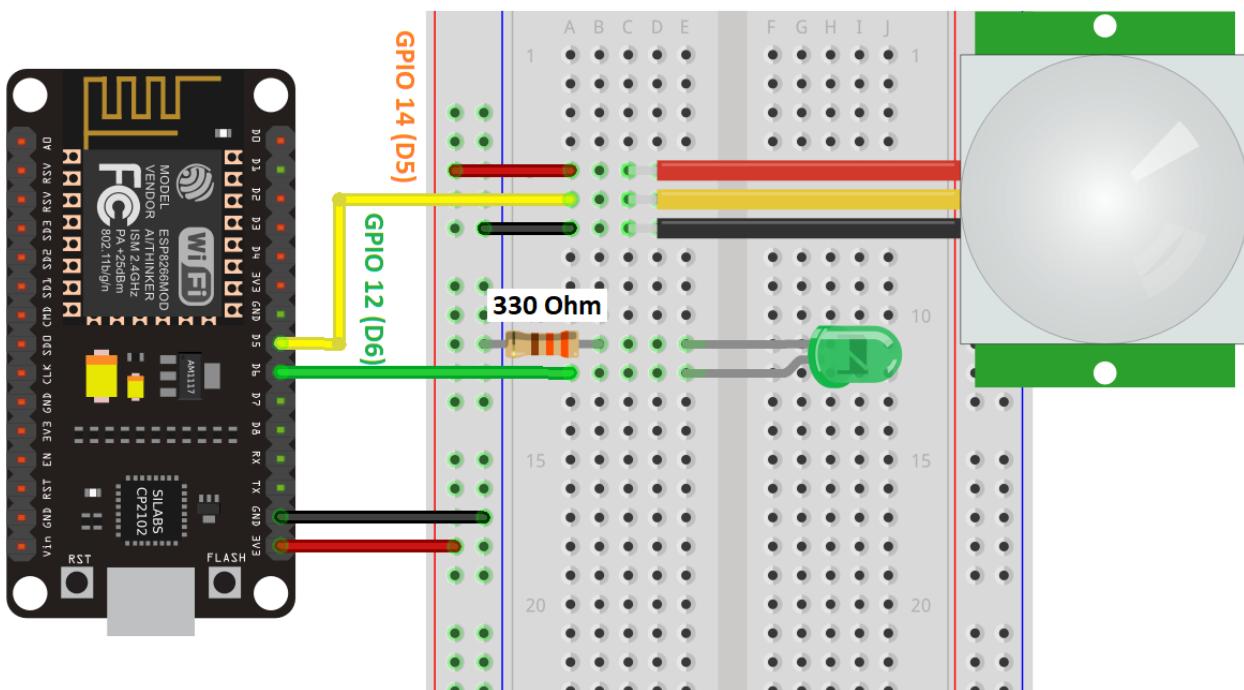
Now that you know how interrupts and timers work, we can build the project with the PIR motion sensor. Whenever motion is detected, an LED lights up for 10 seconds.

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

- [ESP8266](#)
- [PIR motion sensor \(AM312\)](#)
- [5mm LED](#)
- [330 Ohm resistor](#)
- [Breadboard](#)
- [Jumper wires](#)

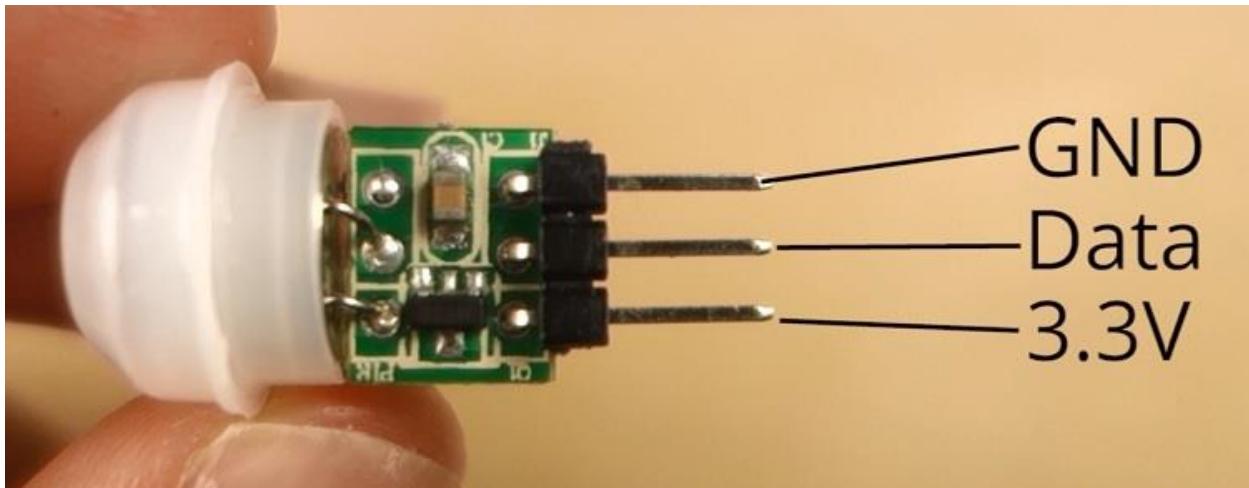
Schematic Diagram

Assemble the PIR motion sensor and an LED to your ESP8266. We'll connect the LED to GPIO 12 (D6) and the PIR motion sensor data pin to GPIO 14 (D5).



Important: the Mini AM312 PIR Motion Sensor used in this project operates at 3.3V. However, if you're using another PIR motion sensor like the [HC-SR501](#), it operates at 5V. You can either [modify it to operate at 3.3V](#) or simply power it using the Vin pin.

The following figure shows the AM312 PIR motion sensor pinout.



Code

After wiring the circuit as shown in the schematic diagram, copy the code provided to your Arduino IDE.

You can upload the code as it is, or you can modify the number of seconds the LED is lit after detecting motion. Simply change the `timeSeconds` variable with the number of seconds you want.

```
#define timeSeconds 10

// Set GPIOs for LED and PIR Motion Sensor
const int led = 12;
const int motionSensor = 14;

// 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
ICACHE_RAM_ATTR void 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;
    }
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module2/Unit4/Detect%20Motion%20PIR/Detect%20Motion%20PIR.ino

How the Code Works

Start by assigning two GPIO pins to the `led` and `motionSensor` variables.

```
const int led = 12;
const int motionSensor = 14;
```

Then, create variables that will allow you set a timer to turn the LED off after motion is detected.

```
unsigned long now = millis();
unsigned long lastTrigger = 0;
```

```
boolean startTimer = false;
```

The `now` variable holds the current time. The `lastTrigger` variable holds the time when the PIR sensor detects motion. The `startTimer` is a boolean variable that starts the timer when motion is detected.

setup()

In the `setup()`, start by initializing the serial port at 115200 baud rate.

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

Set the PIR motion sensor as an `INPUT_PULLUP`.

```
pinMode(motionSensor, INPUT_PULLUP);
```

To set the PIR sensor pin as an interrupt, use the `attachInterrupt()` function as described earlier.

```
attachInterrupt(digitalPinToInterrupt(motionSensor), detectsMovement, RISING);
```

The pin that will detect motion is GPIO 14 and it will call the function `detectsMovement()` on `RISING` mode.

The LED is an `OUTPUT` whose state starts at `LOW`.

```
pinMode(led, OUTPUT);
digitalWrite(led, LOW);
```

loop()

The `loop()` function is constantly running over and over again. In every loop, the `now` variable is updated with the current time.

```
now = millis();
```

Nothing else is done in the `loop()`. But, when motion is detected, the `detectsMovement()` function is called because we've set an interrupt previously on the `setup()`.

The `detectsMovement()` function prints a message in the Serial Monitor, turns the LED on, sets the `startTimer` boolean variable to `true` and updates the `lastTrigger` variable with the current time.

```
ICACHE_RAM_ATTR void detectsMovement() {
    Serial.println("MOTION DETECTED!!!");
    digitalWrite(led, HIGH);
    startTimer = true;
    lastTrigger = millis();
}
```

After this step, the code goes back to the `loop()`.

This time, the `startTimer` variable is true. So, when the time defined in seconds has passed (since motion was detected), the following if statement will be true.

```
if(startTimer && (now - lastTrigger > (timeSeconds*1000))) {
    Serial.println("Motion stopped...");
    digitalWrite(led, LOW);
    startTimer = false;
}
```

The “Motion stopped...” message will be printed in the Serial Monitor, the LED is turned off, and the `startTimer` variable is set to false.

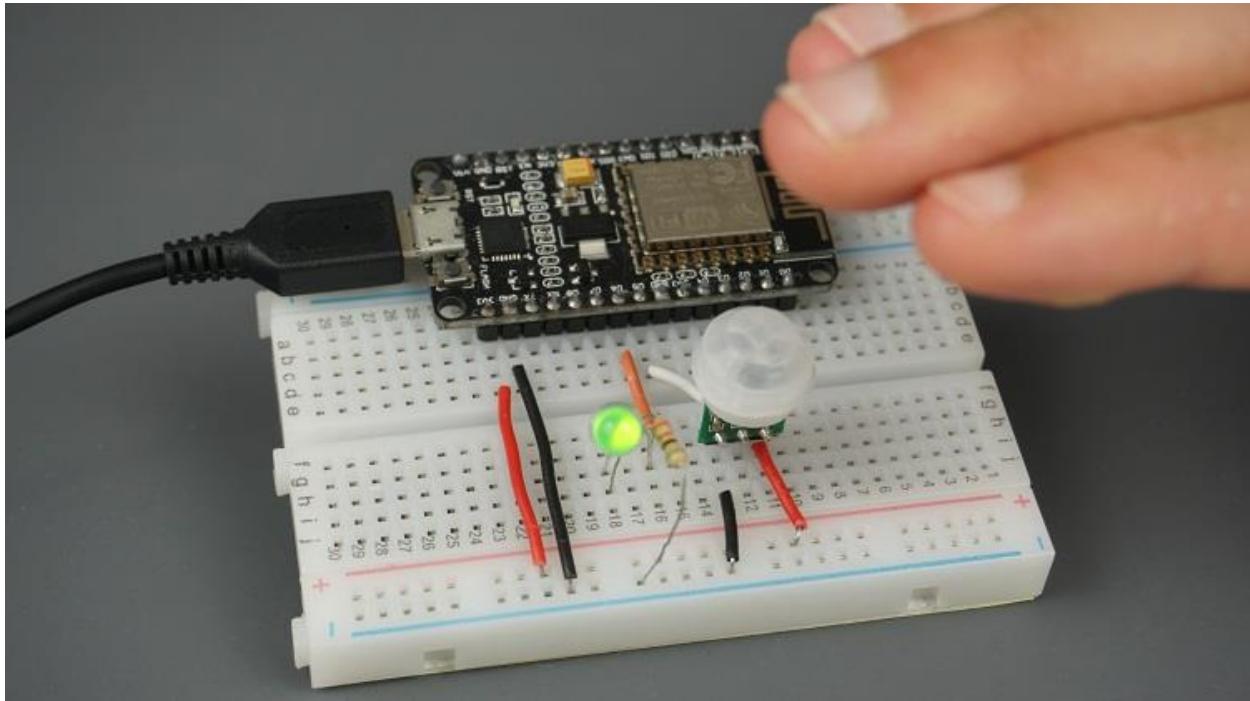
Demonstration

Upload the code to your ESP8266. Make sure you have the right board and COM port selected.

Open the Serial Monitor at a baud rate of 115200.



Move your hand in front of the PIR sensor. The LED should turn on, and a message is printed in the Serial Monitor saying “MOTION DETECTED!!!”.



After 10 seconds the LED should turn off.

Unit 5: Deep Sleep



In this section, you'll learn what is deep sleep, how to put the ESP8266 in deep sleep mode and different ways to wake it up.

Introducing Deep Sleep Mode

If you've made a project with an ESP8266 board that is battery powered, or if you just connected your ESP8266 NodeMCU board to a power bank, after running it for a while, you realize the battery doesn't last long, specially if you're using Wi-Fi.



LiPo Battery



Power Bank



Li-ion

If you put your ESP8266 in deep sleep mode, it reduces power consumption and your batteries will last longer.

Having the ESP8266 in deep sleep mode means cutting with the activities that consume more power while operating (like Wi-Fi) but leave just enough activity to wake up the processor when something interesting happens.

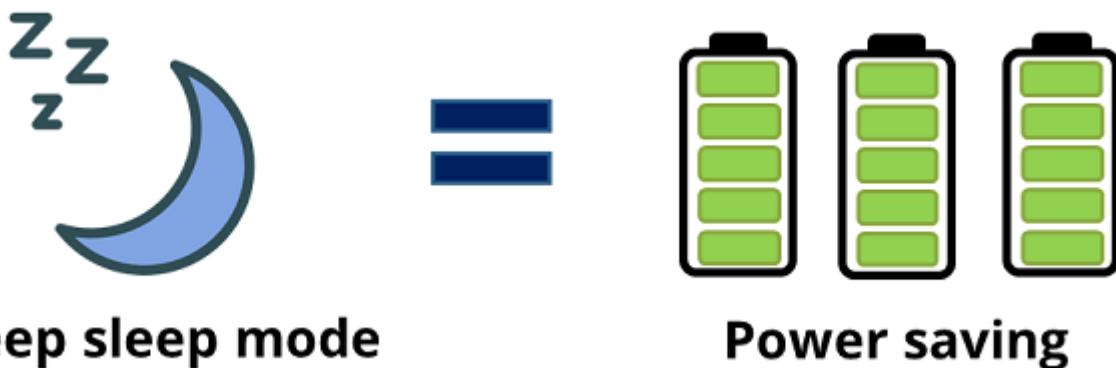
Types of Sleep

There are three different types of sleep mode: **modem sleep**, **light sleep**, and **deep sleep**. The table below shows the differences between each mode (information from the ESP8266 datasheet).

Item	Modem-sleep	Light-sleep	Deep-sleep
Wi-Fi	OFF	OFF	OFF
System clock	ON	OFF	OFF
RTC	ON	ON	ON
CPU	ON	Pending	OFF
Substrate current	15 mA	0.4 mA	~20uA
Average current (DTIM =1)	16.2 mA	1.8 mA	-
Average current (DTIM =3)	15.4 mA	0.9 mA	-
Average current (DTIM =10)	15.2 mA	0.55 mA	-

These modes have different purposes and should be used in different applications.

We'll cover deep sleep mode. Everything is always off, except the Real Time Clock (RTC), which is how the ESP8266 keeps track of time.



This is the most power efficient option and the ESP chip only draws approximately $20\mu\text{A}$. However, if you use a full-feature development board with built-in programmer, LEDs, and so on, you won't be able to achieve such a low power state.

Deep Sleep Sketch

With deep sleep, an example application looks like this:

- The ESP8266 connects to Wi-Fi;
- The ESP8266 performs a task (reads a sensor, publishes an MQTT message, etc.);
- Sleeps for a predefined period of time;
- The ESP8266 wakes up;
- The process is repeated over and over again.

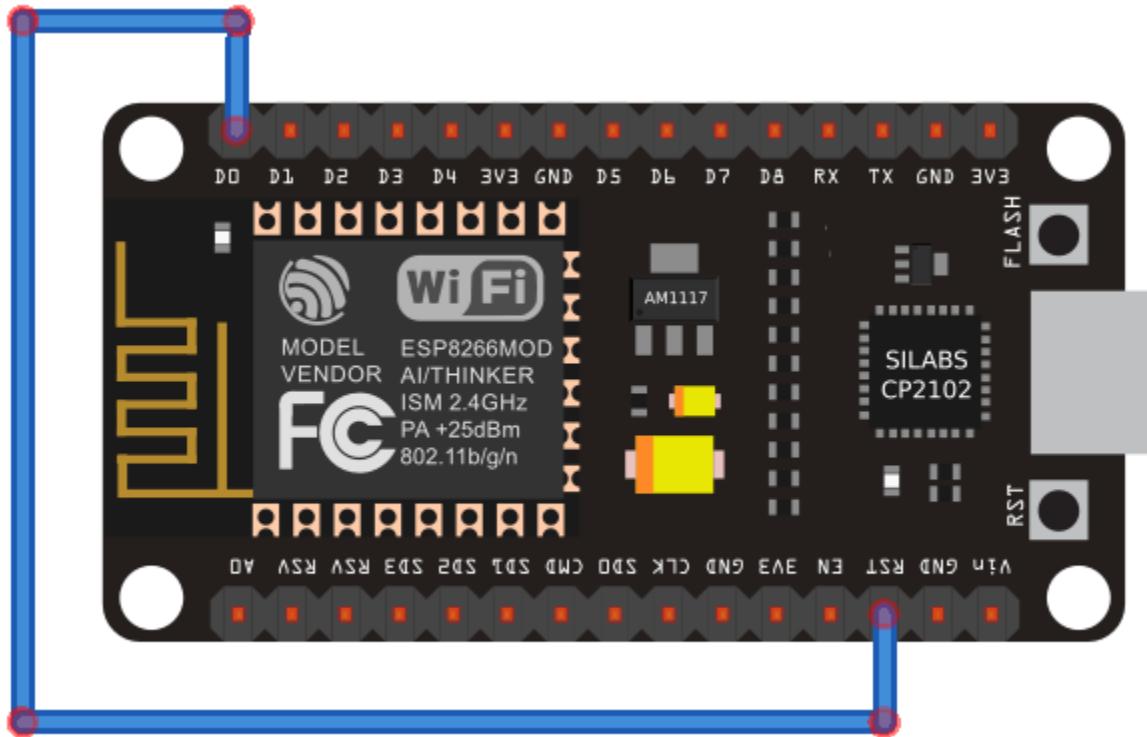
Wake up Sources

After putting the ESP8266 in deep sleep mode, there are different ways to wake it up:

- **#1 timer wake up:** the ESP8266 wakes itself up after a predefined period of time;
- **#2 external wake up:** the ESP8266 wakes up when you press the RST button (the ESP8266 restarts);

#1 ESP8266 Deep Sleep with Timer Wake Up

To use timer wake up with ESP8266, you need to connect the RST pin to GPIO 16 which is labeled as D0, in a NodeMCU board. Simply follow the next schematic diagram:



Important: connect the RST pin to GPIO 16 only after uploading the code.

GPIO 16 is a special pin with a WAKE feature.

The RST pin of the ESP8266 is always HIGH while the ESP8266 is running. However, when the RST pin receives a LOW signal, it restarts the microcontroller.

If you set deep sleep with timer wake up with the ESP8266, when the timer ends, GPIO 16 sends a LOW signal. That means that GPIO 16, when connected to the RST pin, can wake up the ESP8266 after a set period of time.

Code – Timer Wake Up

The following code puts the ESP8266 in deep sleep mode for 30 seconds. When it wakes up, it prints a message in the Serial Monitor and goes back to sleep.

```
void setup() {  
    Serial.begin(115200);  
    Serial.setTimeout(2000);  
  
    // Wait for serial to initialize.  
    while(!Serial) { }  
  
    // Deep sleep mode for 30 seconds, the ESP8266 wakes up by itself when GPIO  
    // 16 (D0 in NodeMCU board) is connected to the RESET pin  
    Serial.println("I'm awake, but I'm going into deep sleep mode for 30  
    seconds");  
    ESP.deepSleep(30e6);  
  
    // Deep sleep mode until RESET pin is connected to a LOW signal (for example  
    // pushbutton or magnetic reed switch)  
    //Serial.println("I'm awake, but I'm going into deep sleep mode until RESET  
    //pin is connected to a LOW signal");  
    //ESP.deepSleep(0);  
}  
  
void loop() {  
}
```

SOURCE CODE

[https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module2/Unit5/Deep Sleep Timer Wake Up/Deep Sleep Timer Wake Up.ino](https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module2/Unit5/Deep%20Sleep%20Timer%20Wake%20Up/Deep%20Sleep%20Timer%20Wake%20Up.ino)

In this example, we print a message in the Serial Monitor:

```
Serial.println("I'm awake, but I'm going into deep sleep mode for 30  
seconds");
```

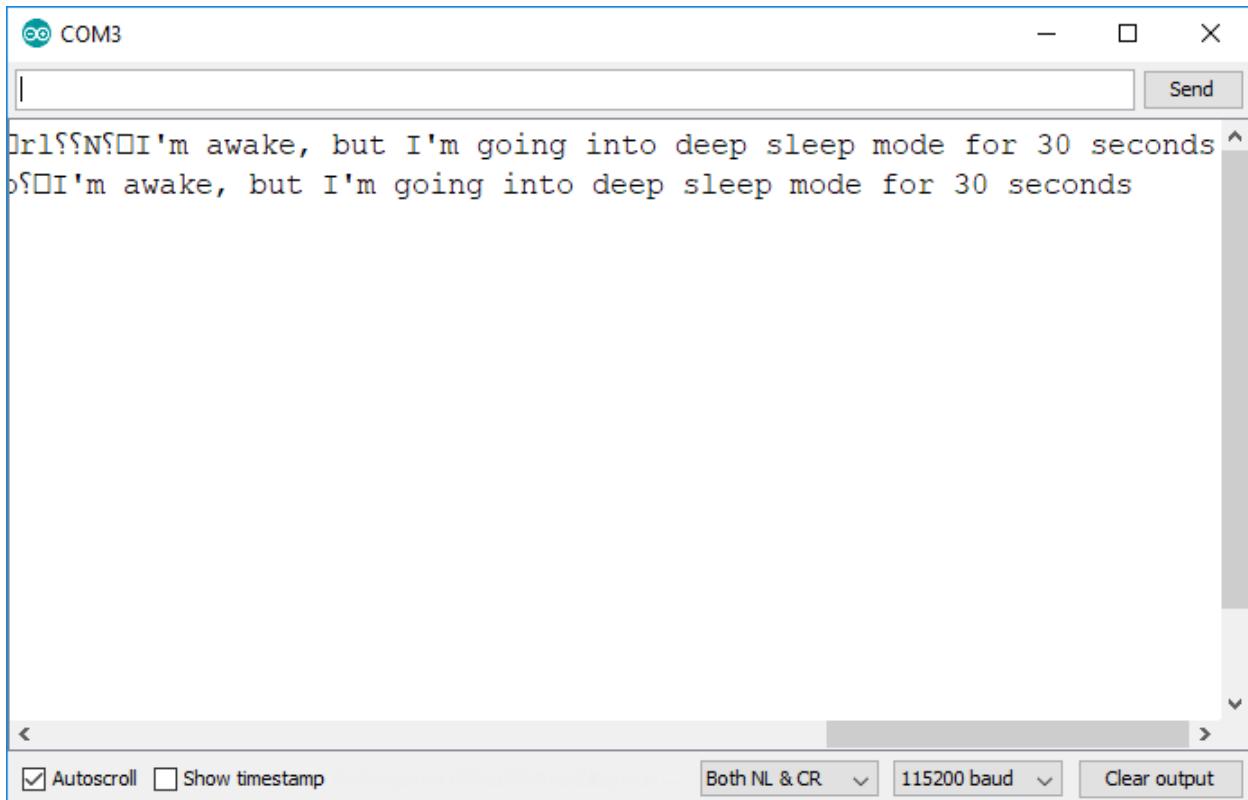
After that, the ESP8266 goes to sleep for 30 seconds.

```
ESP.deepSleep(30e6);
```

To put the ESP8266 in deep sleep, you use `ESP.deepsleep(uS)` and pass as argument the sleep time in microseconds.

In this case, 30e6 corresponds to 30000000 microseconds which is equal to 30 seconds.

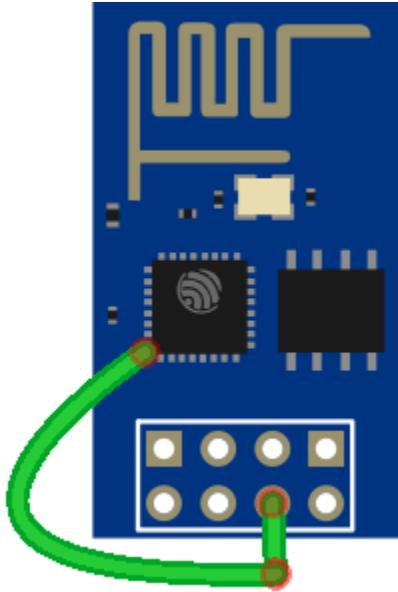
After uploading the code, press the RST button to start running the code, and then connect RST to GPIO 16. The ESP8266 should wake up every 30 seconds and print a message in the Serial Monitor as shown below.



This example simply prints a message in the Serial Monitor, but in a real-world application, you'll perform a useful task like making a request, publish sensor readings, etc.

ESP-01 Timer Wake Up Circuit

If you want to make a similar setup with an ESP-01 board, you need to solder a wire as shown below. That tiny pin is GPIO 16 and it needs to be connected to RST pin.

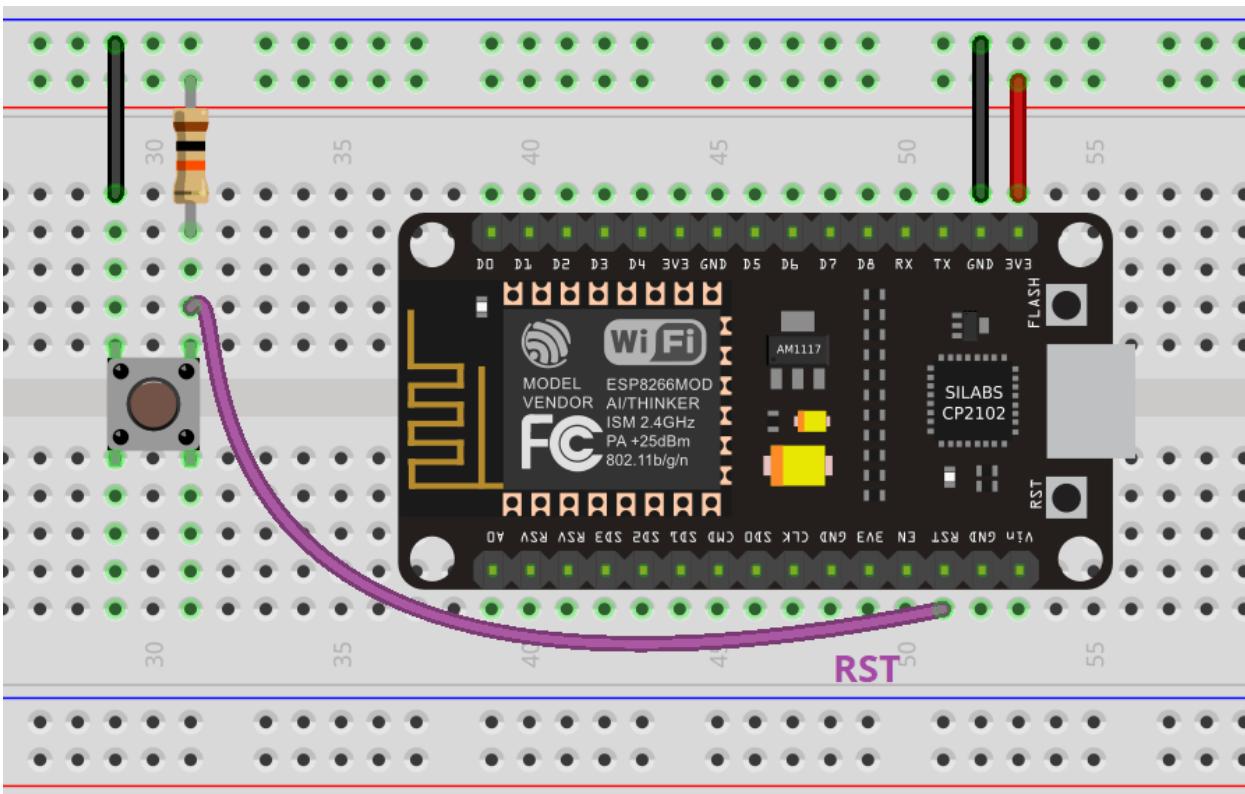


However, the pins are so tiny that it is really hard to solder a wire like that to GPIO 16 on the ESP-01... So, for this wake-up mode, we recommend using a NodeMCU board or a bare ESP12-E chip.

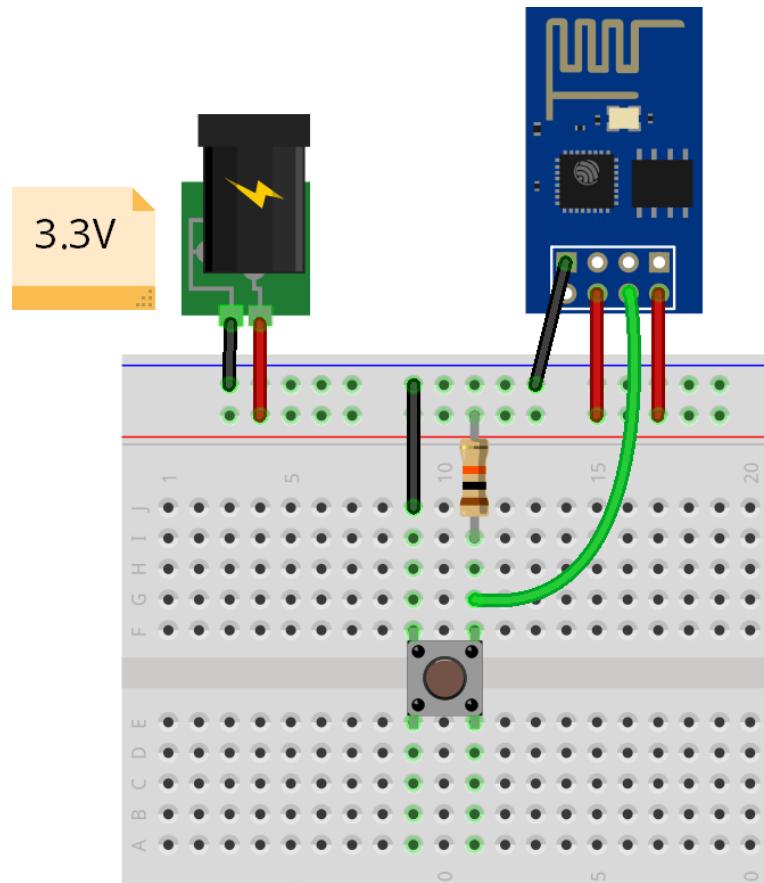
#2 ESP8266 Deep Sleep with External Wake Up

You can also wake up the ESP8266 with an external wake up, like the press of a button or a reed switch. You just need to put the ESP8266 in deep sleep mode for an indefinite period of time, and then set the RST pin to LOW to wake it up.

To test this setup, wire a pushbutton to your ESP8266 board as shown in the following schematic diagram. When you press the pushbutton, the RST pin goes LOW.



If you're using an ESP-01, follow the next diagram instead.



Then, upload the following code to your ESP8266 board.

```
void setup() {  
    Serial.begin(115200);  
    Serial.setTimeout(2000);  
  
    // Wait for serial to initialize.  
    while(!Serial) { }  
  
    // Deep sleep mode for 30 seconds, the ESP8266 wakes up by itself when GPIO  
    // 16 (D0 in NodeMCU board) is connected to the RESET pin  
    //Serial.println("I'm awake, but I'm going into deep sleep mode for 30  
    //seconds");  
    //ESP.deepSleep(30e6);  
  
    // Deep sleep mode until RESET pin is connected to a LOW signal (for example  
    //pushbutton or magnetic reed switch)  
    Serial.println("I'm awake, but I'm going into deep sleep mode until RESET pin  
    is connected to a LOW signal");  
    ESP.deepSleep(0);  
}  
  
void loop() {  
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module2/Unit5/Deep_Sleep_External_Wake_Up/Deep_Sleep_External_Wake_Up.ino

This code puts the ESP8266 in deep sleep mode for an indefinite period of time. For that, you just need to pass 0 as an argument to the `deepSleep()` function:

`ESP.deepSleep(0);`

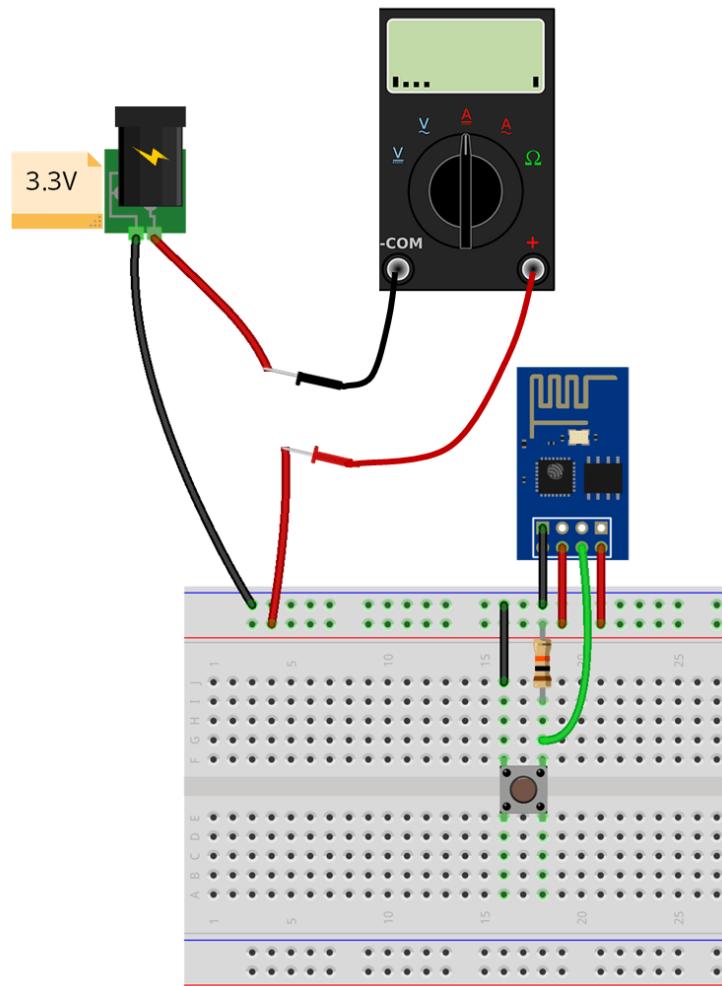
The ESP will only wake up when something resets the board. In this case, it's the press of a pushbutton that pulls the RST pin to GND.

When you press the pushbutton, the ESP8266 wakes up, does the programmed task and goes back to sleep until a new reset event is triggered.

Measuring Current Consumption

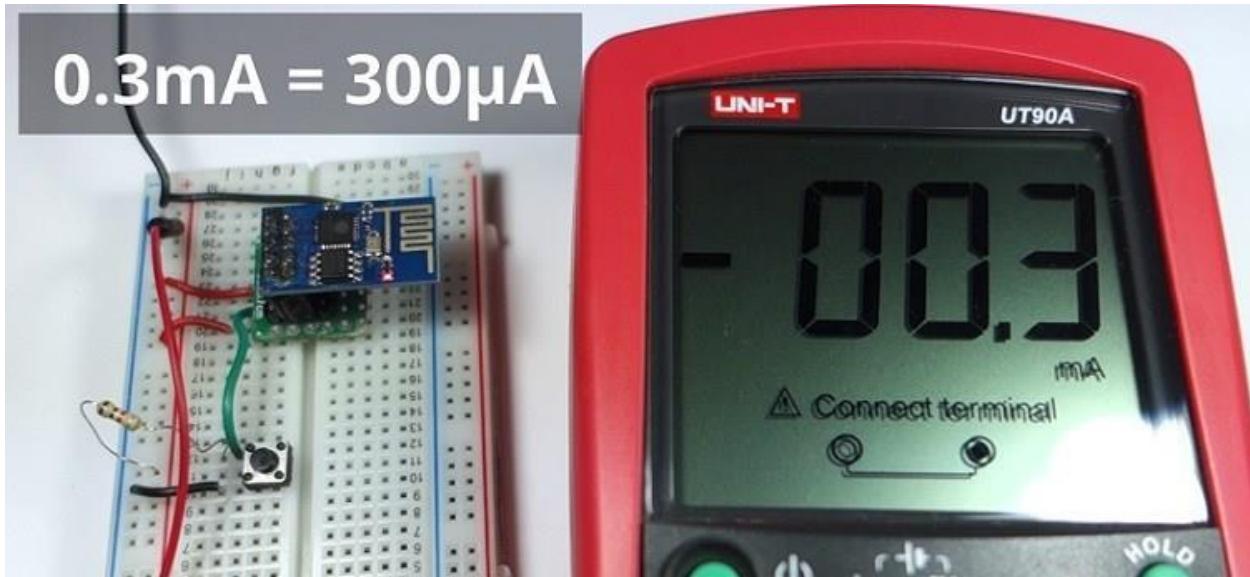
When the board is in deep sleep mode, measure the current consumption with a multimeter to see how much power it is draining.

Here's how you should place your multimeter probes.



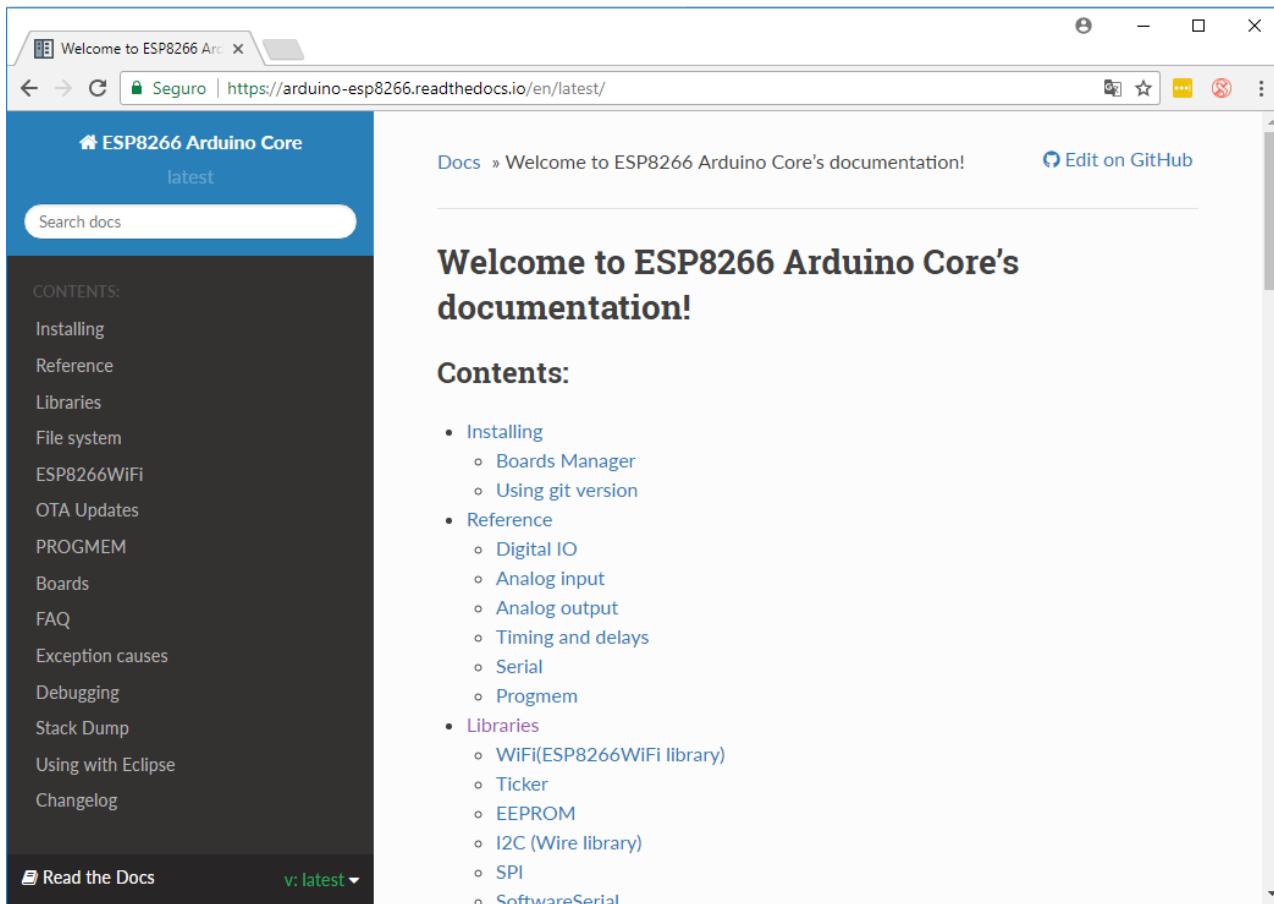
When the ESP-01 is in deep sleep mode it's only using 0.3mA which is approximately 300uA.

0.3mA = 300µA



Keep in mind that during normal usage with Wi-Fi, the ESP8266 can consume between 50 mA and 170 mA.

Unit 6: Reference Guide



More documentation and details about the ESP8266 and programming the ESP8266 using Arduino IDE can be found in the following URL:

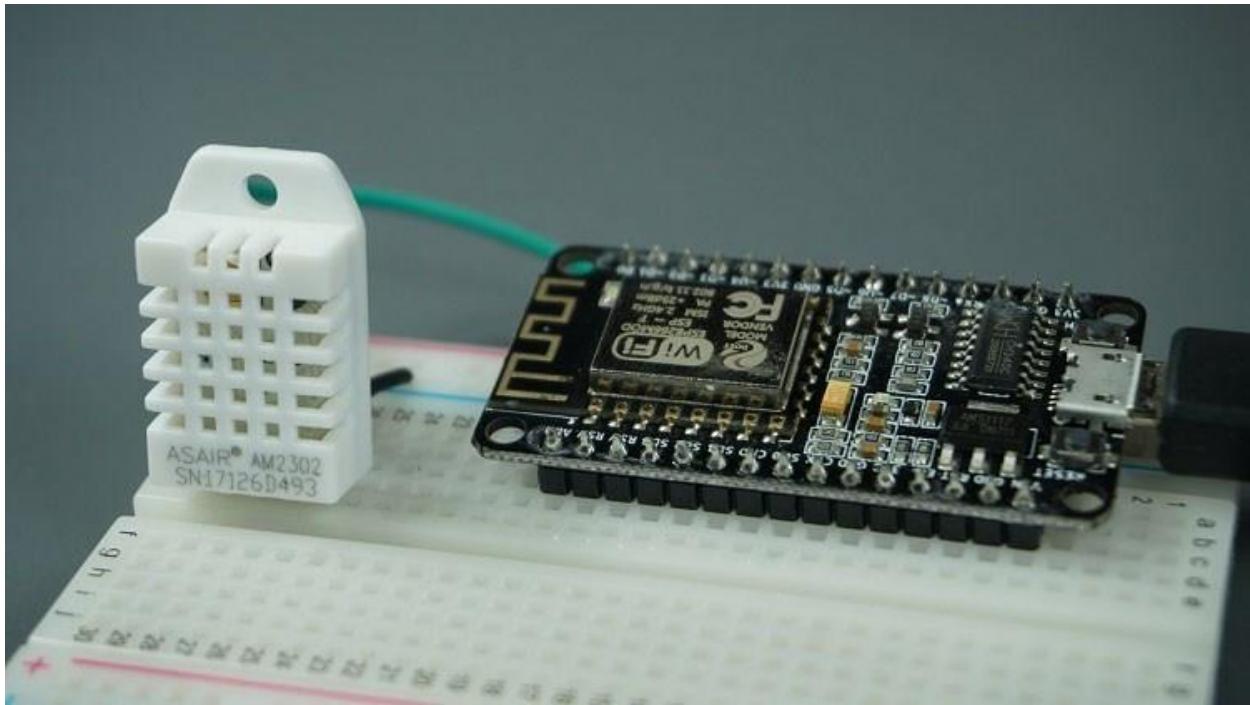
- <https://arduino-esp8266.readthedocs.io/en/latest>

MODULE 3:

Interfacing with Sensors, Modules and Displays

This Module covers how to interface different temperature sensors, modules and displays with the ESP8266. We'll show you how to wire the sensors and write simple sketch examples to understand how the code works. In the next Module, you'll use these components to start building your first IoT projects.

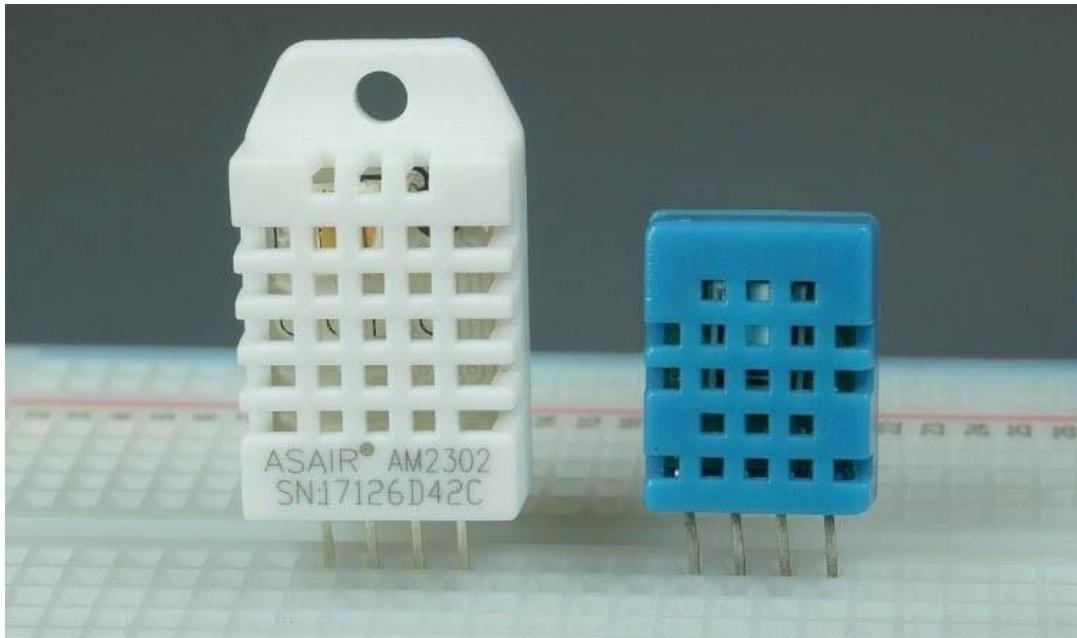
Unit 1: DHT11 and DHT22 Digital Temperature and Humidity Sensors



In this Unit, you'll learn how to use the DHT11 and DHT22 temperature and humidity sensors. We'll go through a quick introduction to these sensors, pinout, wiring diagram, and finally the Arduino sketch.

Introducing the DHT11 and DHT22 Sensors

The DHT11 and DHT22 sensors are used to measure temperature and relative humidity. These are very popular among makers and electronics hobbyists.



These sensors contain a chip that does analog to digital conversion and spit out a digital signal with the temperature and humidity. This makes them very easy to use with any microcontroller.

DHT11 vs DHT22

The DHT11 and DHT22 are very similar, but differ in their specifications. The following table compares some of the most important specifications of the DHT11 and DHT22 temperature and humidity sensors. For a more in-depth analysis of these sensors, please check the sensors' datasheet.

	DHT11	DHT22
Temperature range	0 to 50 °C (+/- 2 °C)	-40 to 80 °C (+/- 0.5°C)
Humidity range	20 to 90% (+/-5%)	0 to 100% (+/-2%)
Resolution	Humidity: 1% Temperature: 1°C	Humidity: 1% Temperature: 1°C
Operating voltage	3 – 5.5V DC	3 – 6V DC

Current supply	0.5 – 2.5 mA	1 – 1.5 mA
Sampling period	1 second	2 seconds
Price	\$1 to \$5	\$4 to \$10
Where to buy?	Check prices	Check prices

The DHT22 sensor has a better resolution and a wider temperature and humidity measurement range. However, it is a bit more expensive, and you can only request readings with 2 seconds interval. The DHT11 has a smaller range and it's less accurate. However, you can request sensor readings every second. It's also a bit cheaper. Despite their differences, they work in a similar way, and you can use the same code to read temperature and humidity. You just need to select in the code the sensor type you're using.

DHT Pinout

DHT sensors have four pins as shown in the following figure. However, if you get your DHT sensor in a breakout board, it comes with only three pins and with an internal pull-up resistor on pin 2.



The following table shows the DHT22/DHT11 pinout. When the sensor is facing you, pin numbering starts at 1 from left to right

DHT pin	Connect to
1	3.3V
2	Any digital GPIO; also connect a 4.7k Ohm pull-up resistor (or similar)
3	Don't connect
4	GND

Parts Required

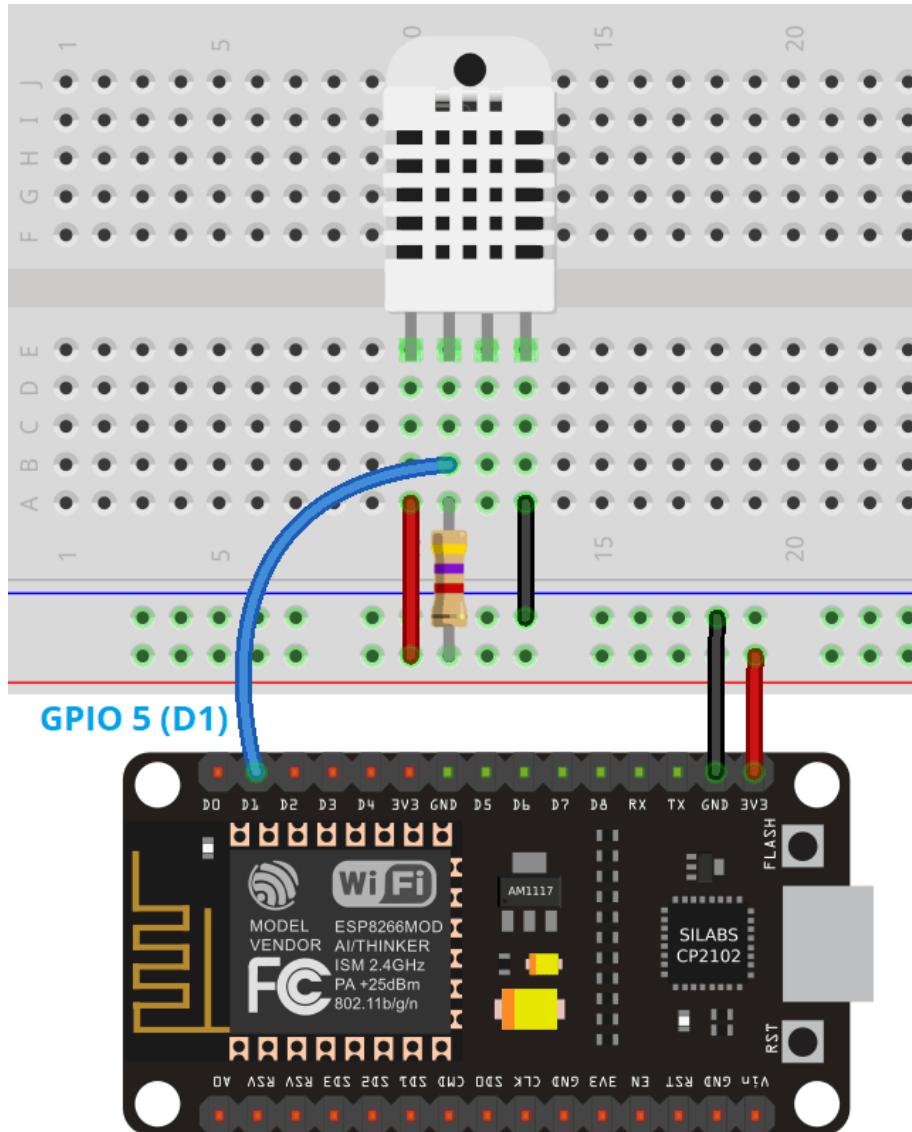
To follow this tutorial, you need to wire the DHT11 or DHT22 temperature sensor to the ESP8266. You need to use a 4.7k Ohm pull-up resistor.

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

- [ESP8266](#)
- [DHT11](#) or [DHT22](#) temperature and humidity sensor
- [4.7k Ohm resistor](#)
- [Breadboard](#)
- [Jumper wires](#)

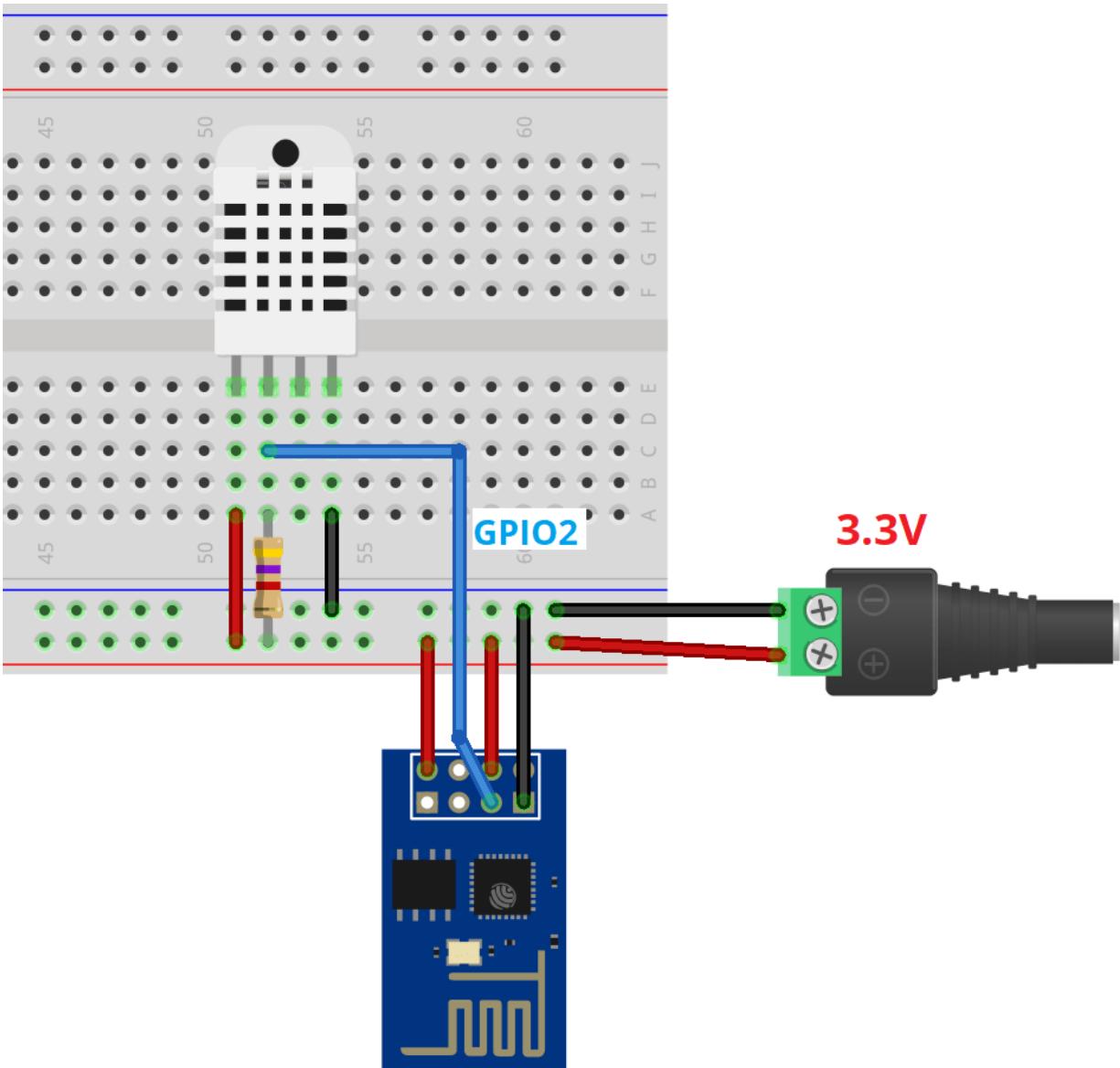
Schematic Diagram

Wire the DHT22 or DHT11 sensor to the ESP8266 development board as shown in the following schematic diagram.



In this example, we're connecting the DHT data pin to GPIO 5 (D1). However, you can use any other suitable digital pin.

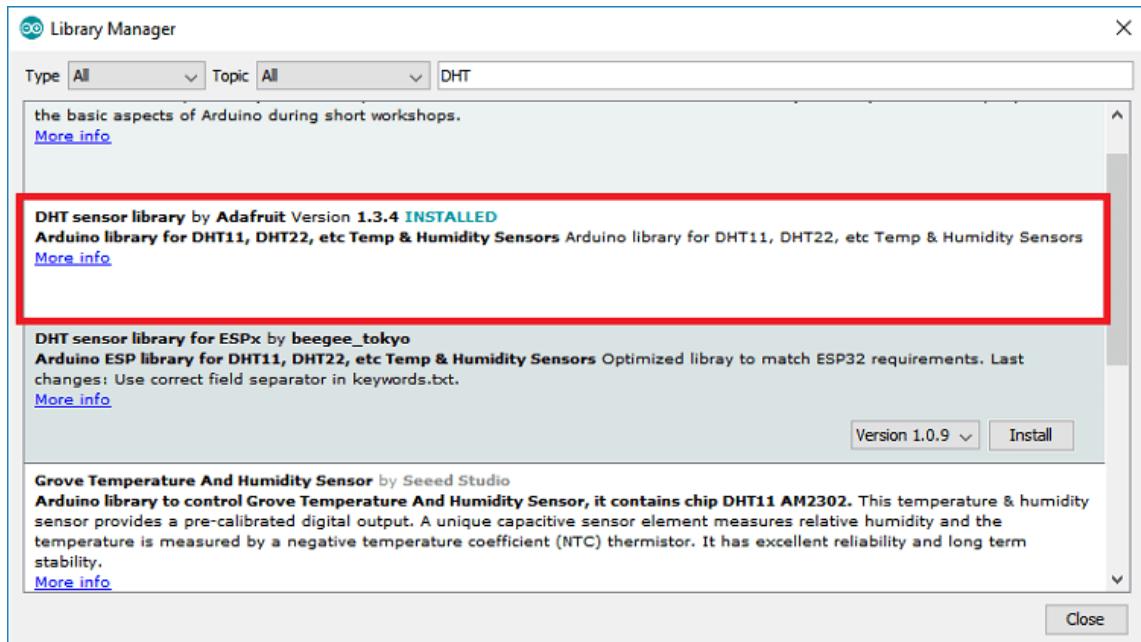
If you're using an ESP-01, GPIO 2 is the most suitable pin to connect to the DHT data pin, as shown in the next diagram.



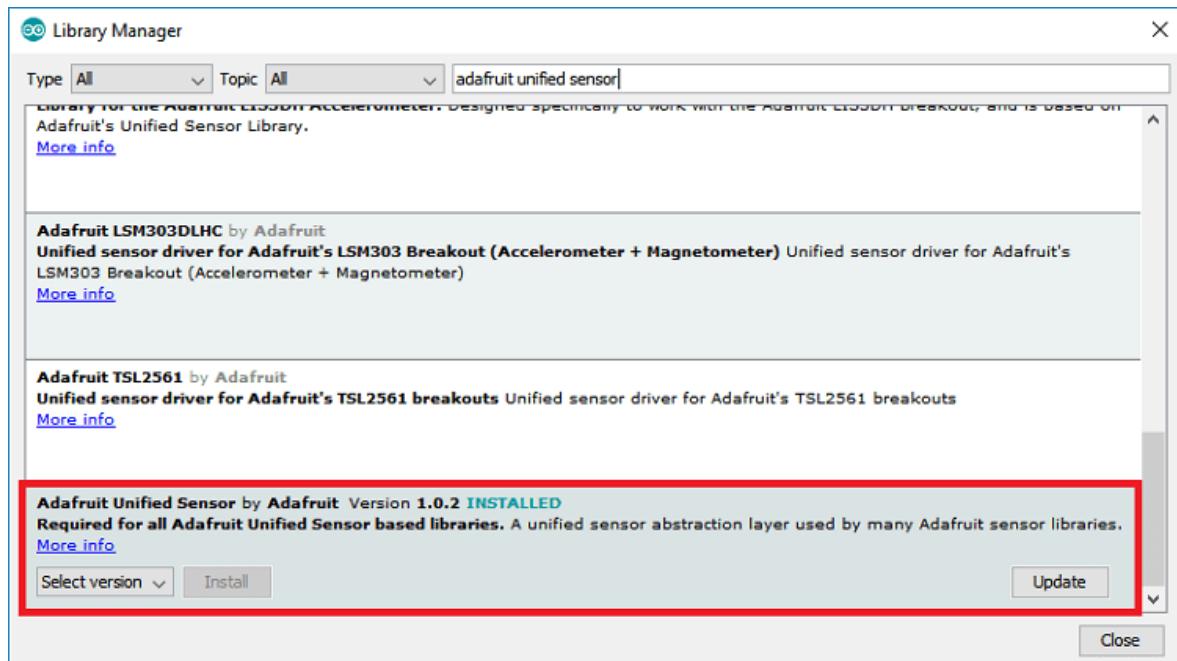
Installing Libraries

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.

1. Open your Arduino IDE and go to **Sketch > Include Library > Manage Libraries**. The Library Manager should open.
2. Search for “**DHT**” on the Search box and install the DHT library from Adafruit.



3. 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.



4. After installing the libraries, restart your Arduino IDE.

Read Temperature and Humidity Sketch

To read temperature and humidity from the DHT sensor, we'll use an example based on the Adafruit DHT library. Copy the following code to your Arduino IDE.

```
#include "DHT.h"

#define DHTPIN 4      // Digital pin connected to the DHT sensor
// Feather HUZZAH ESP8266 note: use pins 3, 4, 5, 12, 13 or 14 --
// Pin 15 can work but DHT must be disconnected during program upload.

// Uncomment whatever type you're using!
// #define DHTTYPE DHT11    // DHT 11
#define DHTTYPE DHT22    // DHT 22 (AM2302), AM2321
// #define DHTTYPE DHT21    // DHT 21 (AM2301)

// Connect pin 1 (on the left) of the sensor to +5V
// NOTE: If using a board with 3.3V logic like an Arduino Due connect
pin 1
// to 3.3V instead of 5V!
// Connect pin 2 of the sensor to whatever your DHTPIN is
// Connect pin 4 (on the right) of the sensor to GROUND
// Connect a 10K resistor from pin 2 (data) to pin 1 (power) of the
sensor

// Initialize DHT sensor.
// Note that older versions of this library took an optional third
parameter to
// tweak the timings for faster processors. This parameter is no longer
needed
// as the current DHT reading algorithm adjusts itself to work on faster
procs.
DHT dht(DHTPIN, DHTTYPE);

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

  dht.begin();
}

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

  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow
sensor)
  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);

// 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;
}

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

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"));
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module3/Unit1/DHT_Example/DHT_Example.ino

There are many comments throughout the code with useful information. So, you might want to take a look at those comments. Continue reading to learn how the code works.

How the Code Works

First, you need to import the DHT library:

```
#include "DHT.h"
```

Then, define the digital pin that the DHT sensor data pin is connected to. In this case, it's connected to GPIO 4.

```
#define DHTPIN 4
```

Then, you need to select the DHT sensor type you're using. The library supports DHT11, DHT22, and DHT21. Uncomment the sensor type you're using and comment all the others. In this case, we're using the DHT22 sensor.

```
//#define DHTTYPE DHT11 // DHT 11
#define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321
//#define DHTTYPE DHT21 // DHT 21 (AM2301)
```

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

```
DHT dht(DHTPIN, DHTTYPE);
```

In the `setup()`, initialize the Serial debugging at a baud rate of 9600, and print a message in the Serial Monitor.

```
Serial.begin(9600);
Serial.println(F("DHTxx test!"));
```

Finally, initialize the DHT sensor.

```
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.

```
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, as shown below:

```
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.

```
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 below:

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

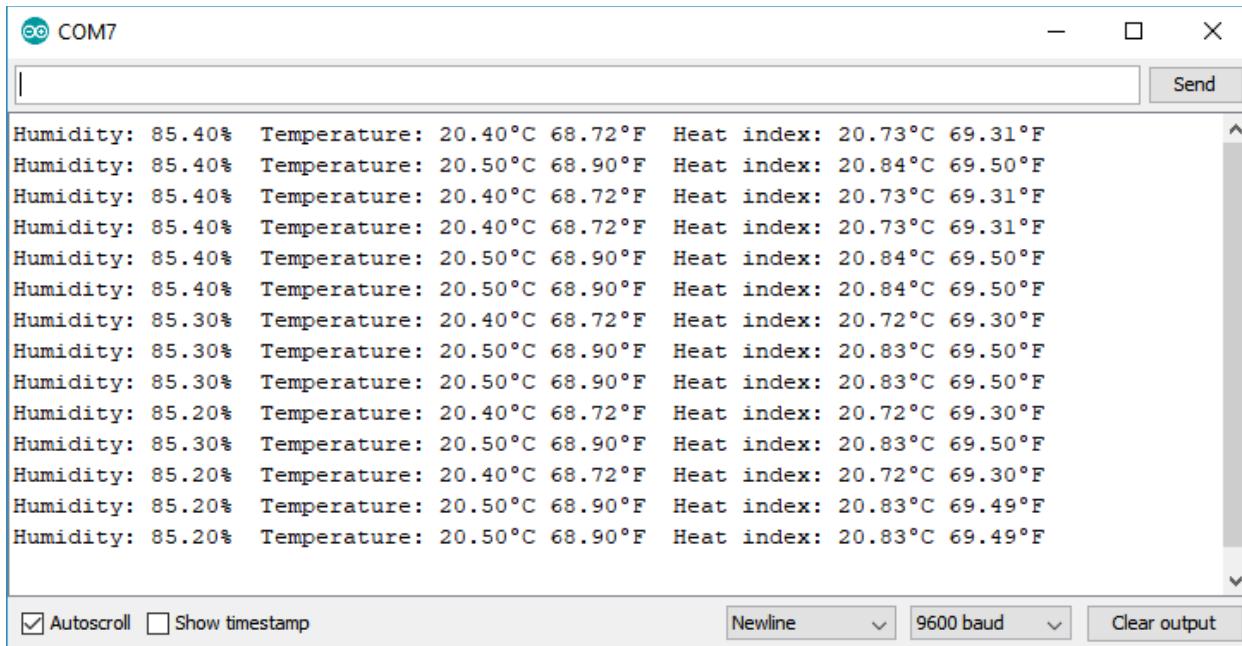
Finally, print all the readings on the Serial Monitor with the following commands:

```
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 ESP8266 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.



Troubleshooting – Failed to read from DHT sensor

If you're trying to read the temperature and humidity from the DHT11/DHT22 sensor and you get an error message in your Serial Monitor, follow the next steps to see if you can make your sensor work.

“Failed to read from DHT sensor!” or Nan readings

If your DHT sensor returns the error message “Failed to read from DHT sensor!” or the DHT readings return “Nan”:

The screenshot shows a terminal window titled "COM7". The main area displays a series of identical error messages: "Failed to read from DHT sensor!" repeated 12 times. Below the terminal window, there is a control panel with several settings:

- Autoscroll
- Show timestamp
- Newline dropdown menu (set to "Newline")
- 9600 baud dropdown menu (set to "9600 baud")
- Clear output button

Try one of the following troubleshooting tips:

Wiring: when you're building electronics project, you need to double-check the wiring or pin assignment. After checking and testing that your circuit is properly connected, if it still doesn't work, continue reading the next troubleshooting tips.

Power: the DHT sensor has an operating range of 3V to 5.5V (DHT11) or 3V to 6V (DHT22). If you're powering the sensor from the 3.3V pin, in some cases powering the DHT with 5V solves the problem.

Bad USB port or USB cable: sometimes powering the ESP8266 directly from a PC USB port is not enough. Try to plug it to a USB hub powered by an external power source. It might also help replacing the USB cable with a better or shorter one. Having a USB port that supplies enough power or using a good USB cable often fixes this problem.

Power source: as mentioned in the previous tip, your ESP might not be supplying enough power to properly read from the DHT sensor. In some cases, you might need to power the ESP with a power source that provides more current.

Sensor type: double-check that you've uncommented/commented in your code the right sensor for your project. In this project, we were using the DHT22:

```
//#define DHTTYPE DHT11    // DHT 11
#define DHTTYPE DHT22     // DHT 22  (AM2302), AM2321
//#define DHTTYPE DHT21    // DHT 21  (AM2301)
```

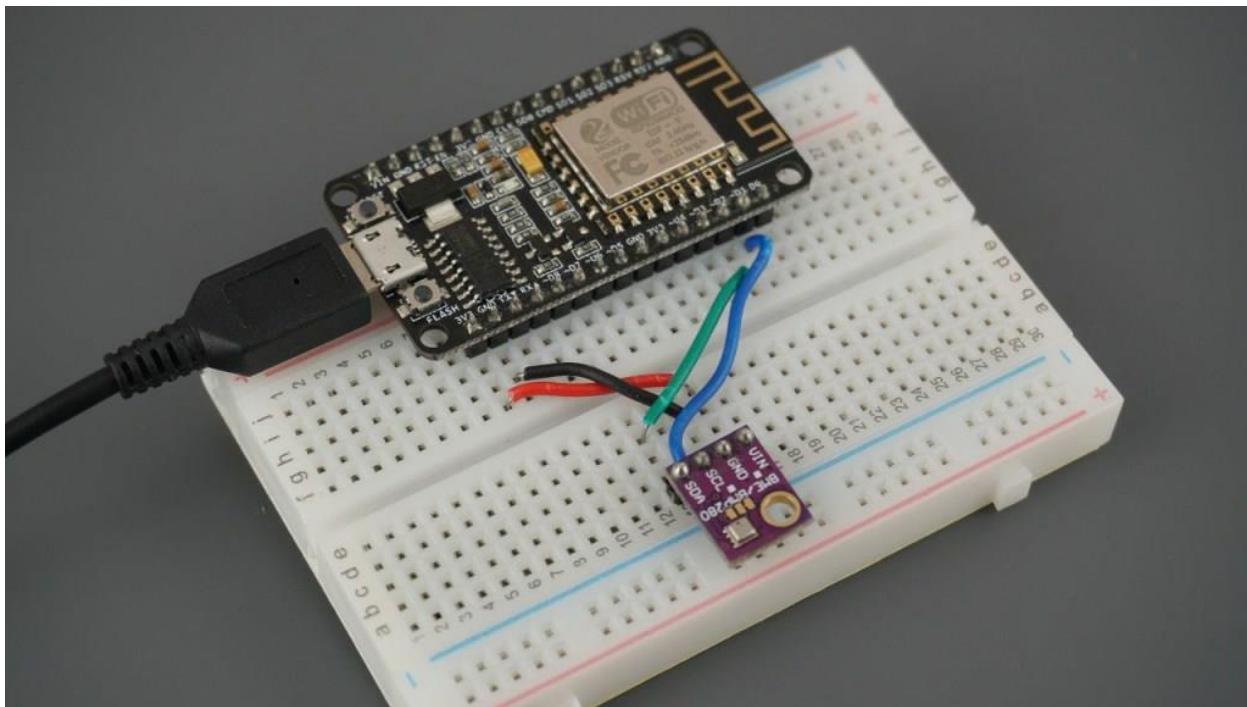
Sampling rate: the DHT sensor is very slow getting the readings (the sensor readings may take up to 2 seconds). In some cases, increasing the time between readings solves the problem.

DHT sensor is fried or broken: unfortunately, these cheap sensors sometimes look totally fine, but they are fried/broken. So, even though you assembled the right circuit and code, it will still fail to get the readings. Try to use a different sensor to see if it fixes your problem.

Wrong baud rate or failed to upload code: if you don't see anything in your Arduino IDE Serial Monitor double-check that you've selected the right baud rate, COM port or that you've uploaded the code successfully.

While building our projects, we've experienced similar issues with the DHT and it was always solved by following one of the methods described earlier.

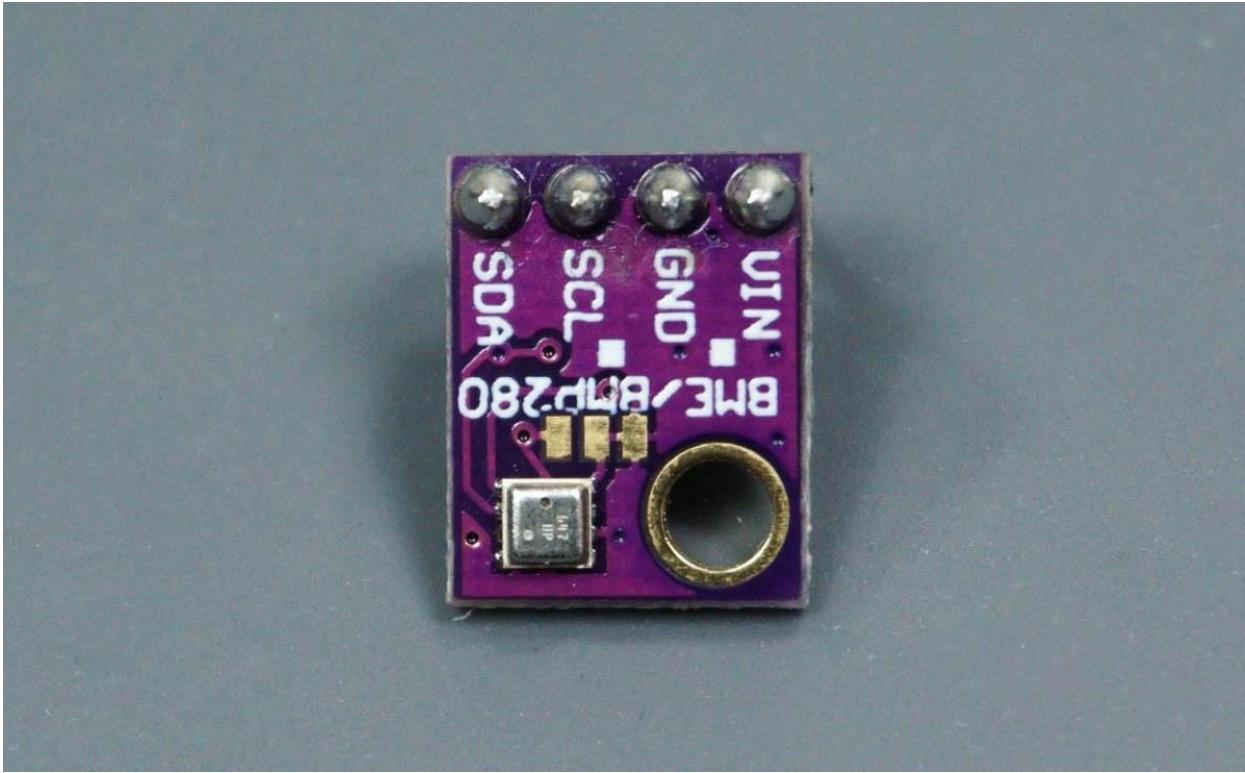
Unit 2: BME280 Pressure, Temperature and Humidity Sensor



This Unit shows how to use the BME280 sensor module with the ESP8266 to read pressure, temperature, humidity and estimate altitude. You'll learn how to wire the sensor, how to install the required libraries, and write a simple sketch that displays the sensor readings on the Serial Monitor.

Introducing BME280 Sensor Module

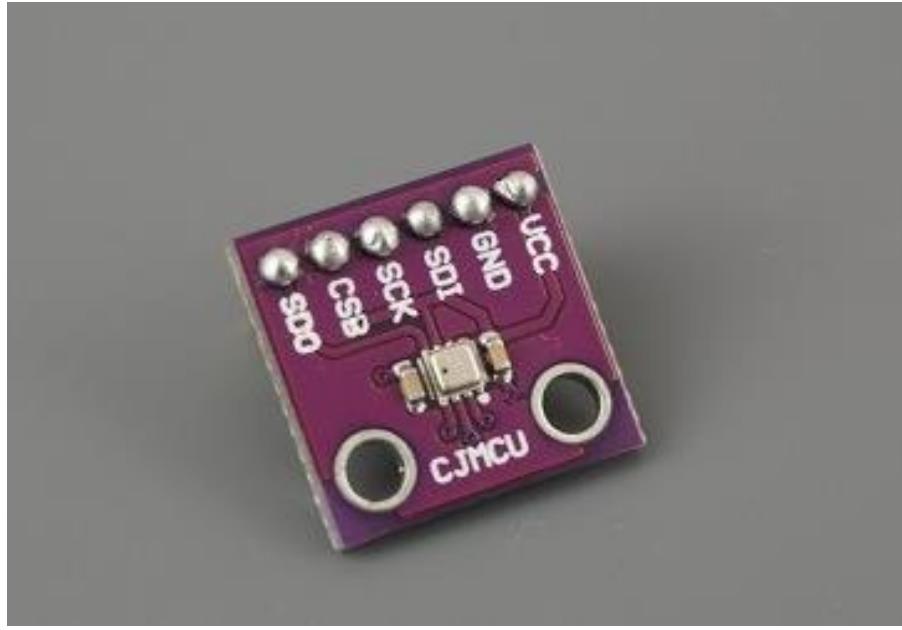
The BME280 sensor module reads barometric pressure, temperature, and humidity. Because pressure changes with altitude, you can also estimate altitude. There are several versions of this sensor module. In this Unit, we'll be using the one illustrated in the figure below.



This sensor communicates using I2C communication protocol, so the wiring is very simple. You can use the default ESP8266 I2C pins as shown in the following table:

BME280	ESP8266
Vin	3.3V
GND	GND
SCL	GPIO 5
SDA	GPIO 4

There are other versions of this sensor that can use either SPI or I2C communication protocols, like the module shown in the next figure:



If you're using one of these sensors, to use I2C communication protocol, use the following pins:

BME280	ESP8266
SCK (SCL Pin)	GPIO 5 (D1)
SDI (SDA pin)	GPIO 4 (D2)

If you want to use SPI communication protocol, you need to use the following pins:

BME280	ESP8266
SCK (SPI Clock)	GPIO 14 (D5)
SDO (MISO)	GPIO 12 (D6)
SDI (MOSI)	GPIO 13 (D7)
CS (Chip Select)	GPIO 15 (D8)

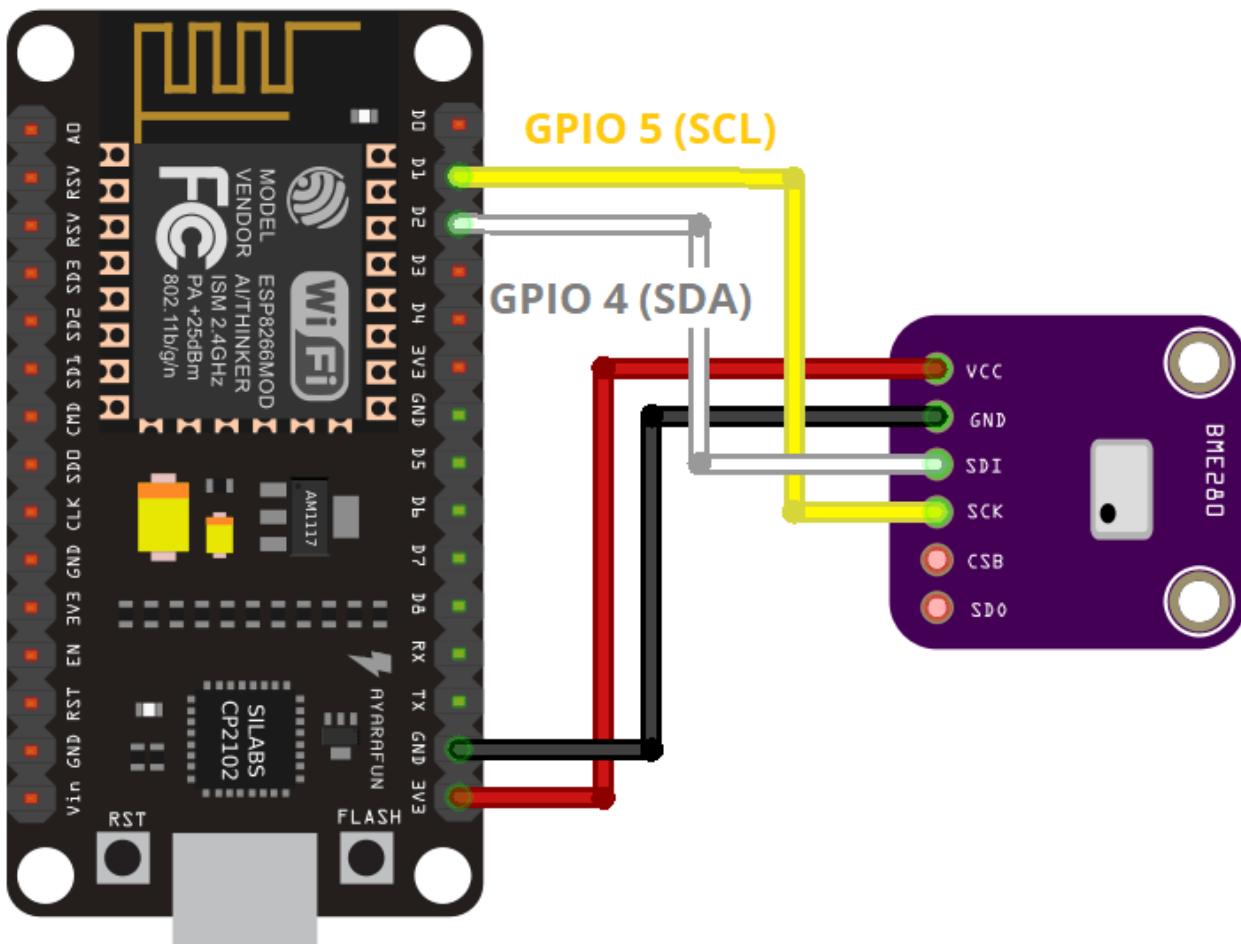
Parts Required

To complete this tutorial, you need the following parts:

- [BME280 sensor module](#)
- [ESP8266](#)
- [Breadboard](#)
- [Jumper wires](#)

Schematic Diagram

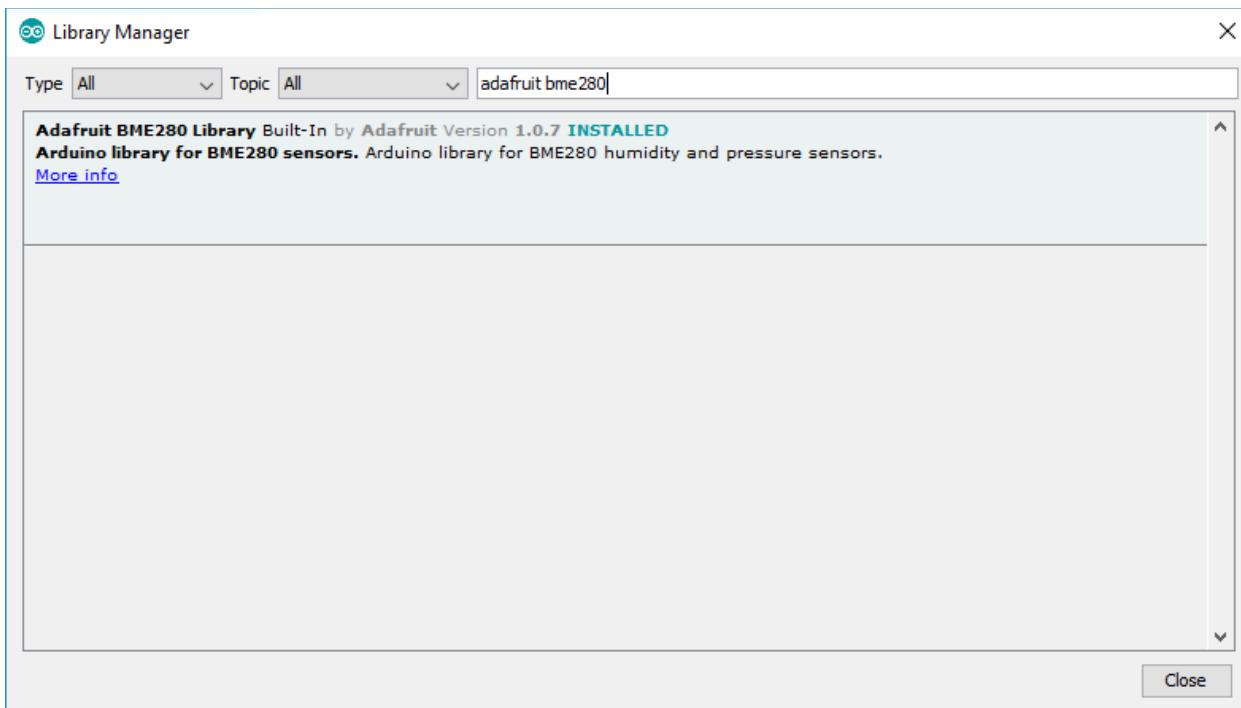
We're going to use I2C communication with the BME280 sensor module. For that, wire the sensor to the ESP8266 SDA and SCL pins, as shown in the following schematic diagram.



Installing the BME280 library

To get readings from the BME280 sensor module you need to use the Adafruit_BME280 library. Follow the next steps to install the library in your Arduino IDE:

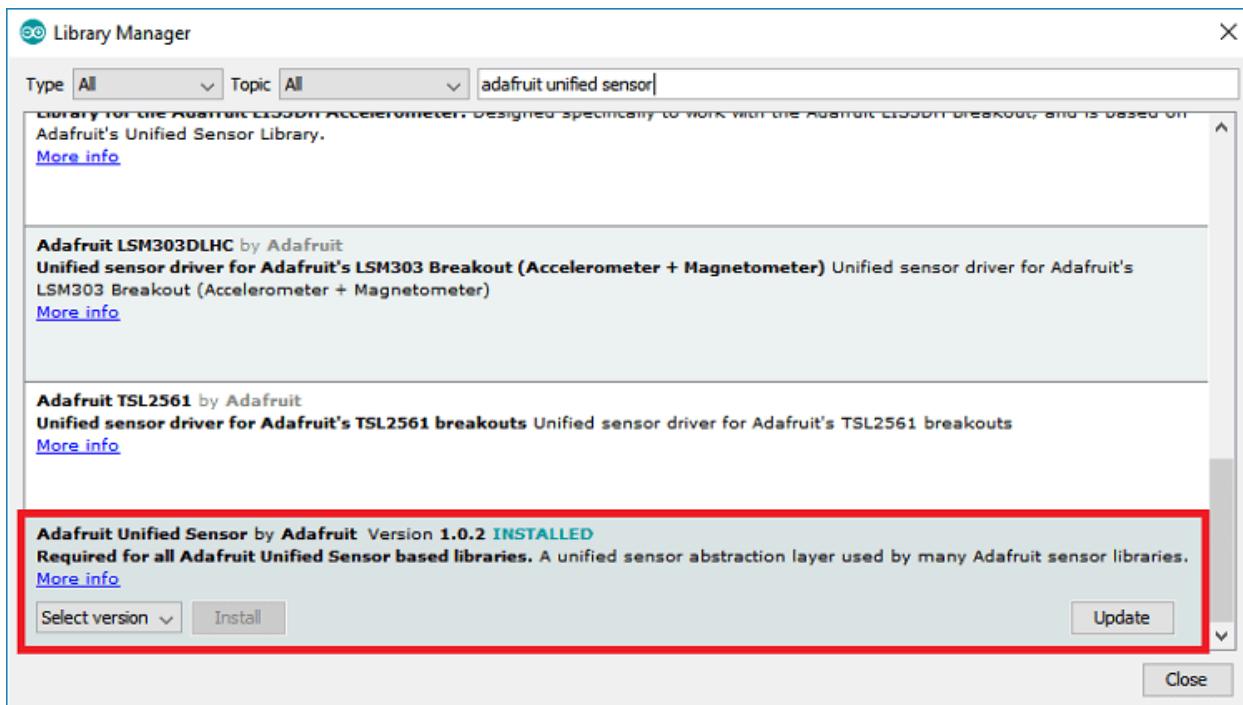
1. Open your Arduino IDE and go to **Sketch > Include Library > Manage Libraries**. The Library Manager should open.
2. Search for “**adafruit bme280**” on the Search box and install the library.



Installing the Adafruit_Sensor library

To use the BME280 library, you also need to install the Adafruit_Sensor library. Follow the next steps to install the library in your Arduino IDE:

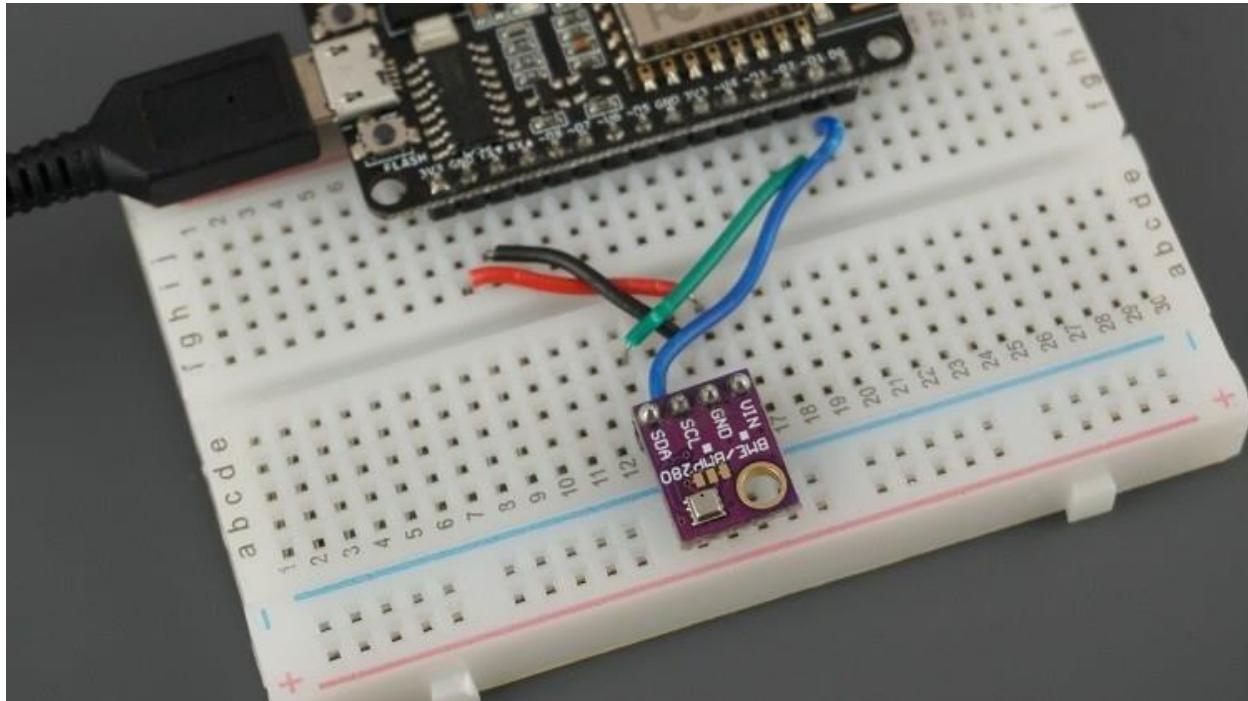
1. Go to **Sketch > Include Library > Manage Libraries**. The Library Manager should open.
2. Type “**Adafruit Unified Sensor**” in the search box. Scroll all the way down to find the library and install it.



After installing the libraries, restart your Arduino IDE.

Reading Pressure, Temperature, and Humidity

To read pressure, temperature, and humidity we'll use a sketch example from the library.



After installing the BME280 library, and the Adafruit_Sensor library, open the Arduino IDE and, go to **File > Examples > Adafruit BME280 library > bme280 test**.

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

/*#include <SPI.h>
#define BME_SCK 14
#define BME_MISO 12
#define BME_MOSI 13
#define BME_CS 15*/

#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BME280 bme; // I2C
//Adafruit_BME280 bme(BME_CS); // hardware SPI
//Adafruit_BME280 bme(BME_CS, BME_MOSI, BME_MISO, BME_SCK); // software
SPI

unsigned long delayTime;

void setup() {
    Serial.begin(9600);
    Serial.println(F("BME280 test"));

    bool status;

    // default settings
    // (you can also pass in a Wire library object like &Wire2)
    status = bme.begin(0x76);
    if (!status) {
        Serial.println("Could not find a valid BME280 sensor, check
wiring!");
        while (1);
    }

    Serial.println("-- Default Test --");
    delayTime = 1000;

    Serial.println();
}

void loop() {
    printValues();
    delay(delayTime);
}

void printValues() {
    Serial.print("Temperature = ");
    Serial.print(bme.readTemperature());
    Serial.println(" *C");
}
```

```

// Convert temperature to Fahrenheit
/*Serial.print("Temperature = ");
Serial.print(1.8 * bme.readTemperature() + 32);
Serial.println(" *F");*/

Serial.print("Pressure = ");
Serial.print(bme.readPressure() / 100.0F);
Serial.println(" hPa");

Serial.print("Approx. Altitude = ");
Serial.print(bme.readAltitude(SEALEVELPRESSURE_HPA));
Serial.println(" m");

Serial.print("Humidity = ");
Serial.print(bme.readHumidity());
Serial.println(" %");

Serial.println();
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module3/Unit2/BME280_Example/BME280_Example.ino

We've made a few modifications to the sketch to make it fully compatible with the ESP8266, so make sure you use our version of the code.

How the Code Works

Continue reading this section to learn how the code works, or skip to the "Demonstration" section.

Libraries

The code starts by including the needed libraries: the `wire` library to use I²C, and the `Adafruit_Sensor` and `Adafruit_BME280` libraries to interface with the BME280 sensor.

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
```

SPI communication

As we're going to use I2C communication, the following lines that define the SPI pins are commented:

```
/*#include <SPI.h>
#define BME_SCK 14
#define BME_MISO 12
#define BME_MOSI 13
#define BME_CS 15*/
```

Note: if you're using SPI communication, use the ESP8266 default SPI pins:

MOSI	MISO	CLK	CS
GPIO 13 (D7)	GPIO 12 (D6)	GPIO 14 (D5)	GPIO 15 (D8)

Sea level pressure

A variable called `SEALEVELPRESSURE_HPA` is created.

```
#define SEALEVELPRESSURE_HPA (1013.25)
```

This variable saves the pressure at the sea level in hectopascal (is equivalent to millibar). This variable is used to estimate the altitude for a given pressure by comparing it with the sea level pressure. This example uses the default value, but for more accurate results, replace the value with the current sea level pressure at your location.

I2C

This example uses I2C communication protocol by default. As you can see, you just need to create an `Adafruit_BME280` object called `bme`.

```
Adafruit_BME280 bme; // I2C
```

To use SPI, you need to comment this previous line and uncomment one of the following lines.

```
//Adafruit_BME280 bme(BME_CS); // hardware SPI  
//Adafruit_BME280 bme(BME_CS, BME_MOSI, BME_MISO, BME_SCK); // software SPI
```

setup()

In the `setup()`, start a serial communication:

```
Serial.begin(9600);
```

And initialize the sensor:

```
status = bme.begin(0x76);  
if (!status) {  
    Serial.println("Could not find a valid BME280 sensor, check  
wiring!");  
    while (1);  
}
```

Note: we initialize the sensor with the **0x76** address. In case you're not getting sensor readings, check the I2C address of your sensor. With the BME280 sensor wired to your ESP8266, run [this I2C scanner sketch](#) to check the address of your sensor. Then, change the address if needed.

Printing values

In the `loop()`, the `printValues()` function reads the values from the BME280 and prints the results in the Serial Monitor.

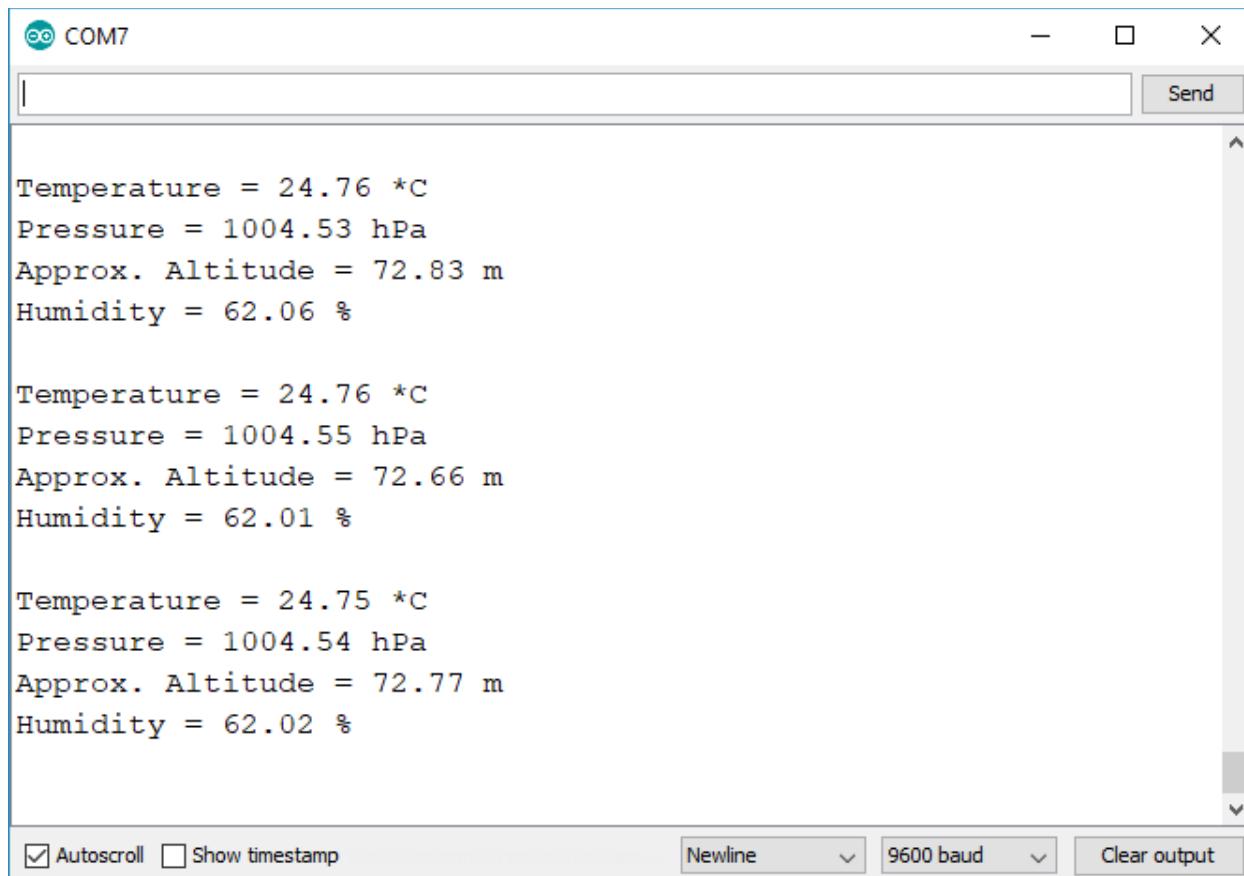
```
void loop() {  
    printValues();  
    delay(delayTime);  
}
```

Reading temperature, humidity, pressure, and estimate altitude is as simple as using the following methods on the `bme` object:

- `bme.readTemperature()` – reads temperature in Celsius;
- `bme.readHumidity()` – reads relative humidity;
- `bme.readPressure()` – reads pressure in hPa (hectopascal = millibar);
- `bme.readAltitude(SEALEVELPRESSURE_HPA)` – estimates altitude in meters based on the pressure at the sea level.

Demonstration

Upload the code to your ESP8266, and open the Serial Monitor at a baud rate of 9600. Press the on-board RST button to run the code. You should see the readings displayed on the Serial Monitor.



The screenshot shows the Arduino Serial Monitor window titled "COM7". The window has a text input field and a "Send" button at the top. Below the text input is a scrollable text area displaying three sets of sensor readings. Each set consists of four lines: Temperature, Pressure, Approx. Altitude, and Humidity. The first set shows values of 24.76 °C, 1004.53 hPa, 72.83 m, and 62.06 %. The second set shows values of 24.76 °C, 1004.55 hPa, 72.66 m, and 62.01 %. The third set shows values of 24.75 °C, 1004.54 hPa, 72.77 m, and 62.02 %. At the bottom of the window, there are checkboxes for "Autoscroll" and "Show timestamp", and dropdown menus for "Newline" and "9600 baud". A "Clear output" button is also present.

```

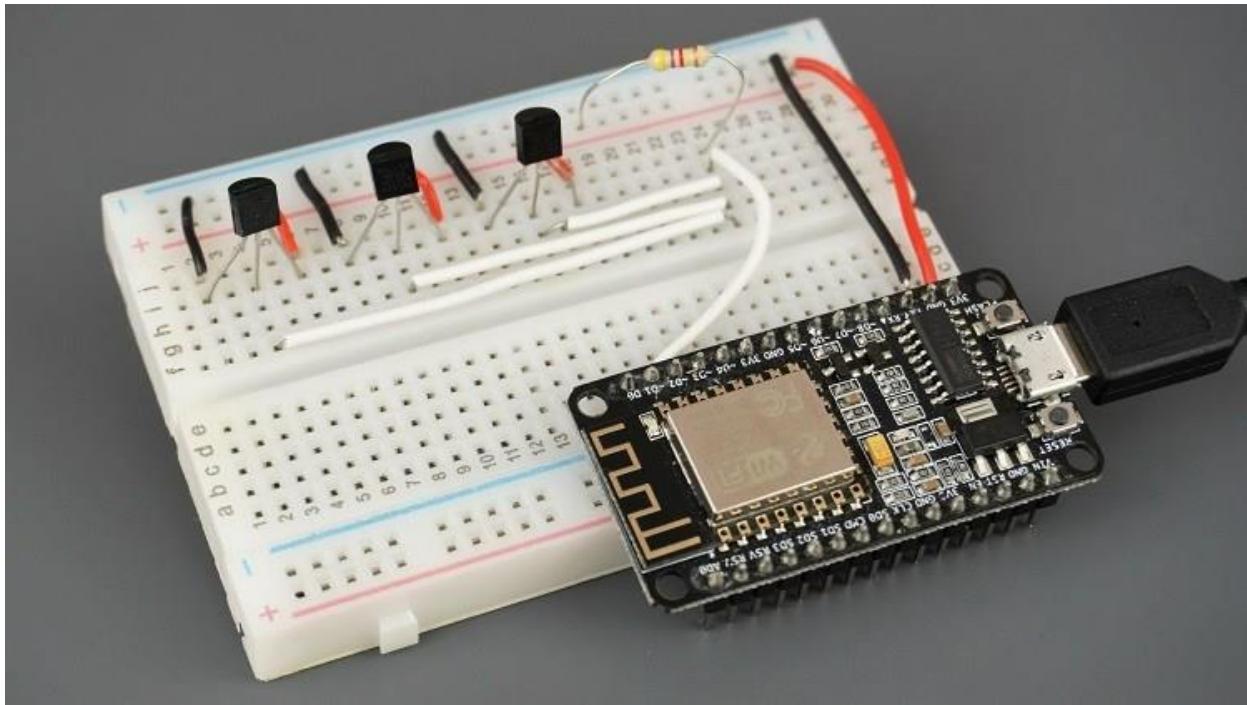
Temperature = 24.76 *C
Pressure = 1004.53 hPa
Approx. Altitude = 72.83 m
Humidity = 62.06 %

Temperature = 24.76 *C
Pressure = 1004.55 hPa
Approx. Altitude = 72.66 m
Humidity = 62.01 %

Temperature = 24.75 *C
Pressure = 1004.54 hPa
Approx. Altitude = 72.77 m
Humidity = 62.02 %

```

Unit 3: DS18B20 Digital Temperature Sensor

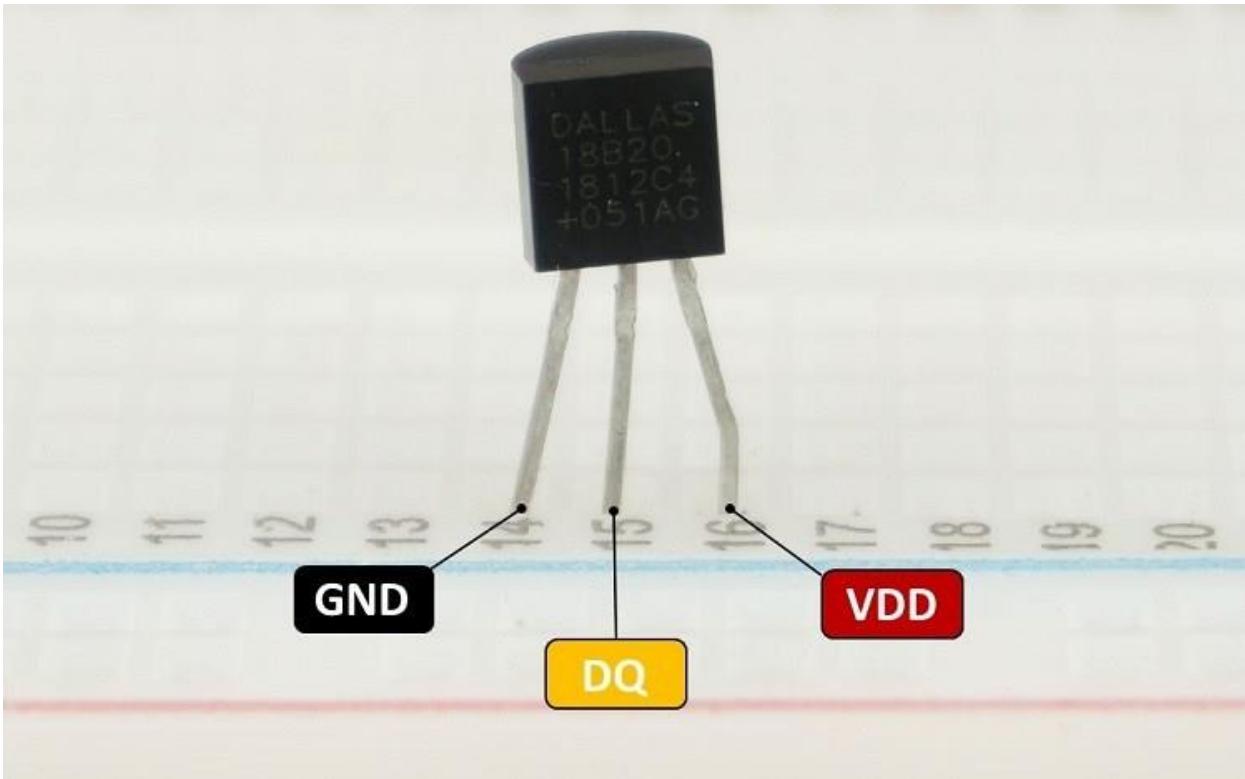


In this Unit, you'll learn how to use another temperature sensor: the DS18B20. We'll cover how to wire the sensor, install the required libraries, and write the code to get the sensor readings from one and multiple sensors.

Introducing DS18B20 Temperature Sensor

The DS18B20 temperature sensor is a one-wire digital temperature sensor. This means that it just requires one data line (and GND) to communicate with your ESP8266.

It can be powered by an external power supply or it can derive power from the data line (called "parasite mode"), which eliminates the need for an external power supply.



Each DS18B20 temperature sensor has a unique 64-bit serial code. This allows you to wire multiple sensors to the same data wire. So, you can get temperature from multiple sensors using just one GPIO.

The DS18B20 temperature sensor is also available in waterproof version.



Here's a summary of the most relevant specs of the DS18B20 temperature sensor:

- Communicates over one-wire bus
- Power supply range: 3.0V to 5.5V
- Operating temperature range: -55°C to +125°C
- Accuracy +/- 0.5 °C (between the range -10°C to 85°C)

For more information consult the DS18B20 datasheet.

Parts Required

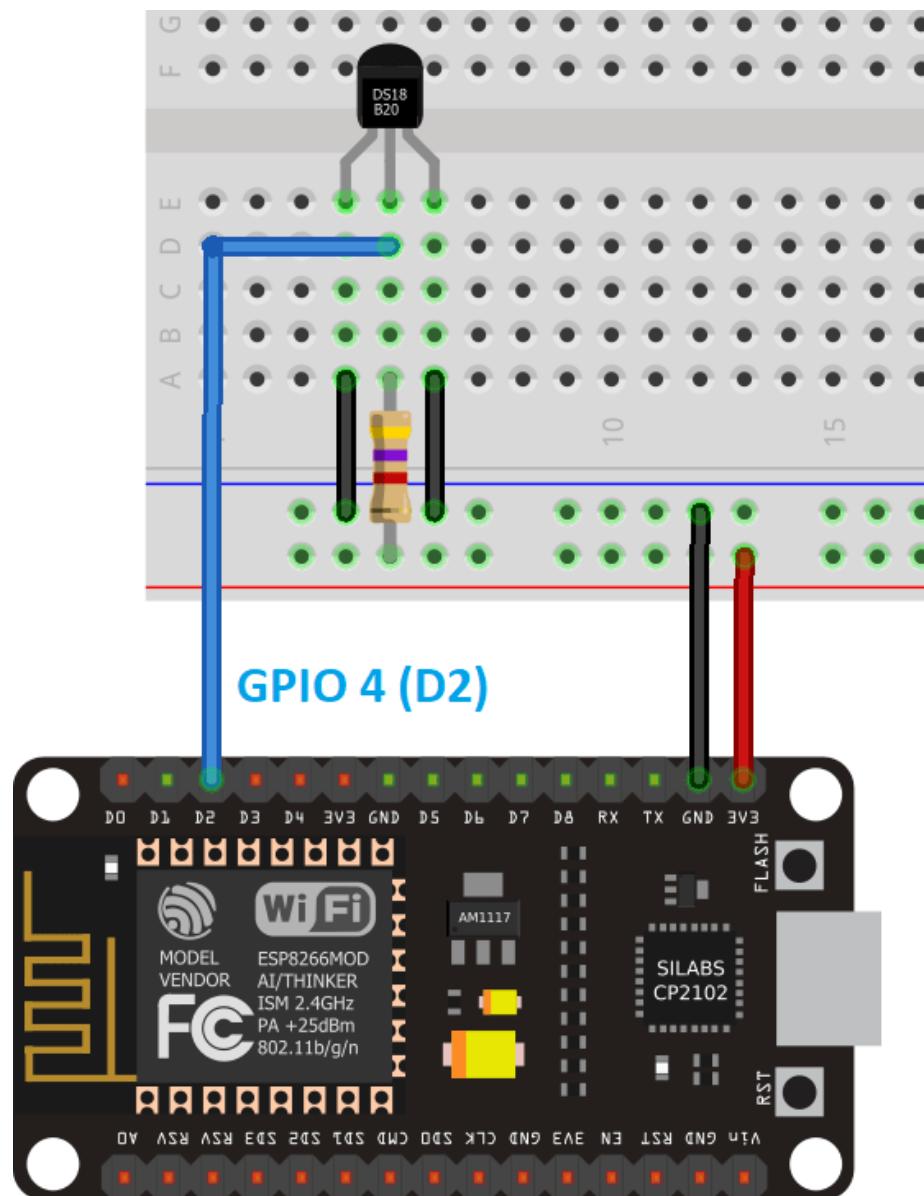
To complete this tutorial, you need the following components:

- [ESP8266](#)
- [DS18B20 temperature sensor](#) (one or multiple sensors) – [waterproof version](#)
- [4.7k Ohm Resistor](#)
- [Breadboard](#)
- [Jumper wires](#)

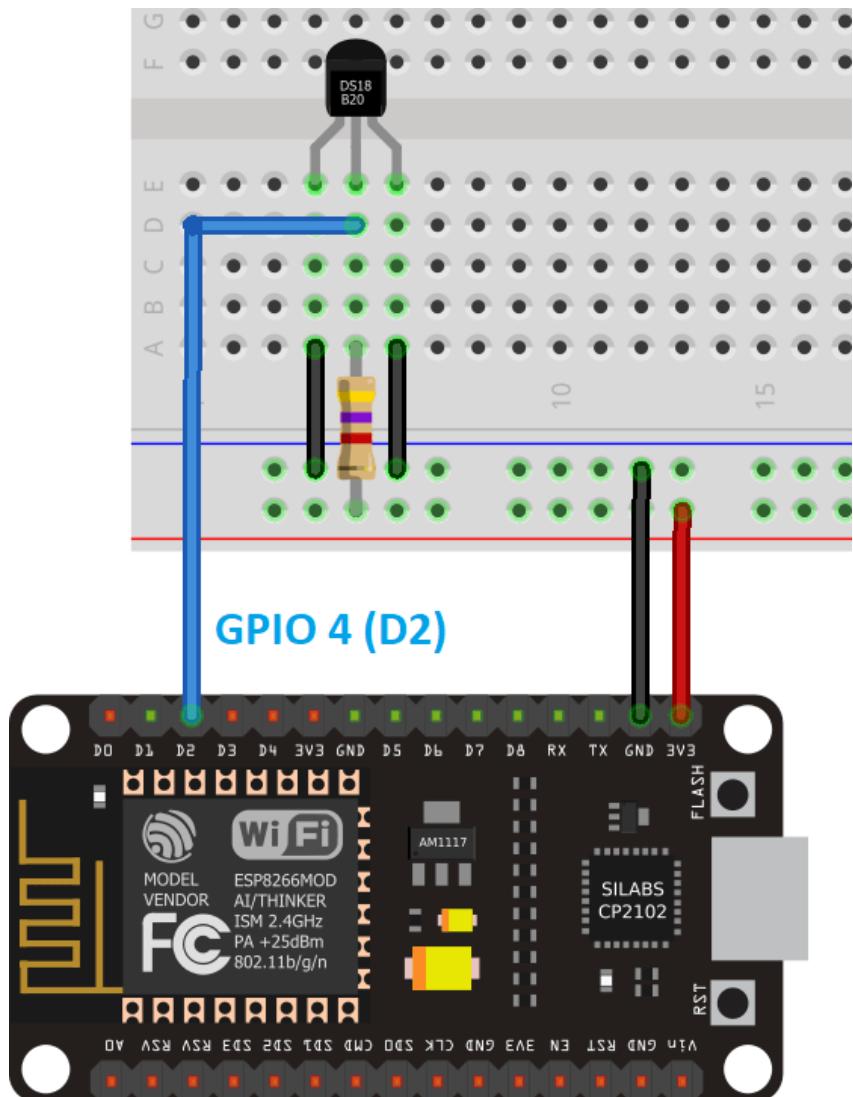
Schematic Diagram

As mentioned previously, the DS18B20 temperature sensor can be powered through the VDD pin (normal mode), or it can derive its power from the data line (parasite mode). You can choose either modes.

Parasite Mode



Normal Mode



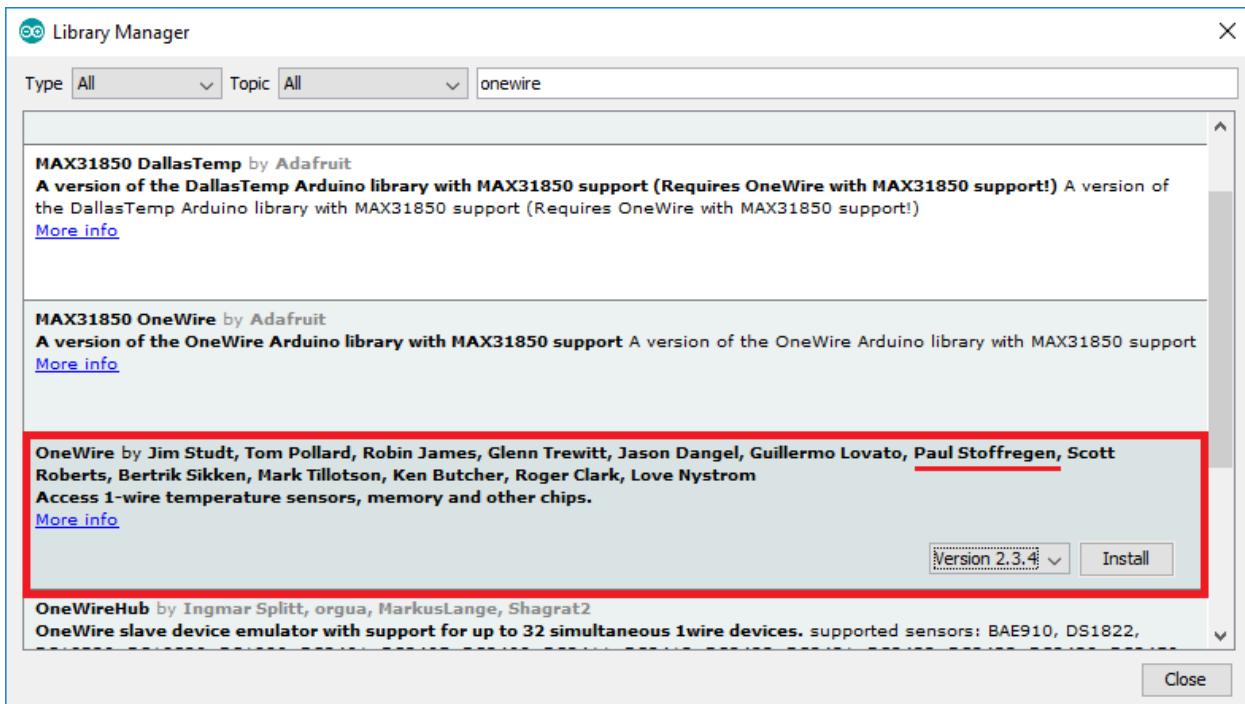
Note: in this tutorial we're connecting the DS18B20 data line to GPIO 4, but you can use any other suitable GPIO.

Note: if you're using an ESP-01, GPIO 2 is the most suitable pin to connect to the DS18B20 data pin.

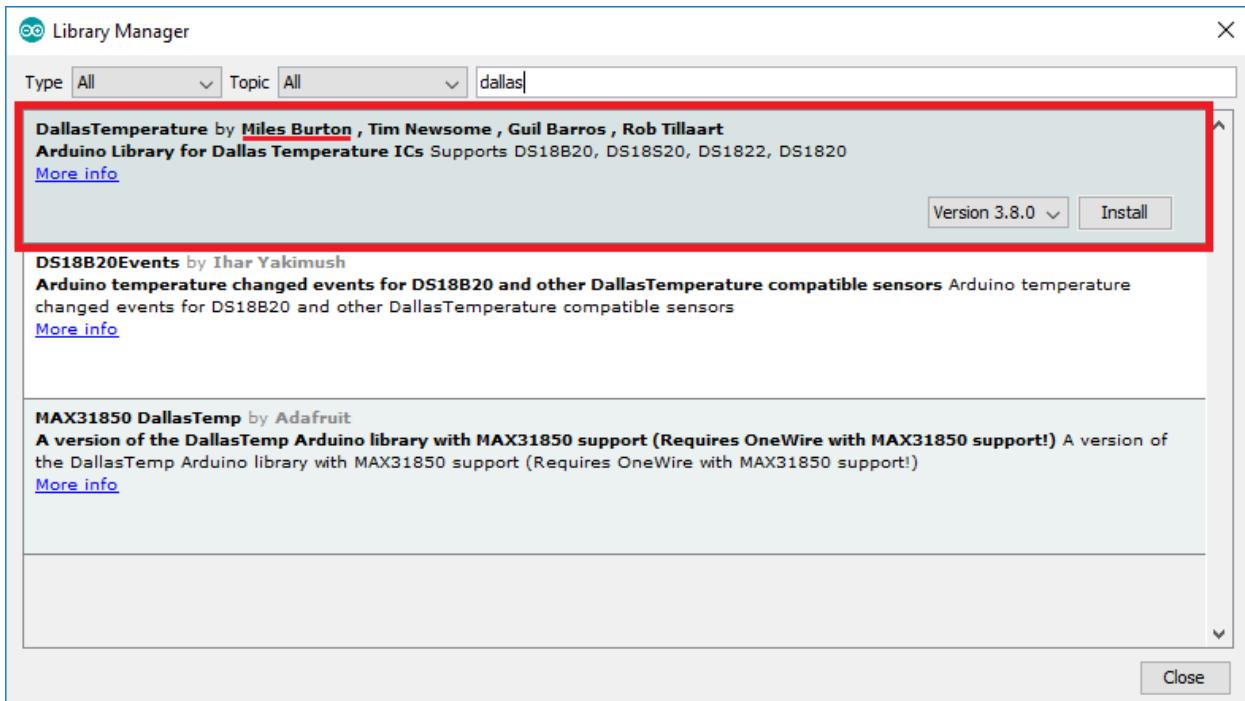
Installing Libraries for DS18B20

To interface with the DS18B20 temperature sensor, you need to install the [One Wire library by Paul Stoffregen](#) and the [Dallas Temperature library](#). Follow the next steps to install those libraries.

1. Open your Arduino IDE and go to **Sketch** ▶ **Include Library** ▶ **Manage Libraries**. The Library Manager should open.
2. Type “**onewire**” in the search box and install the **OneWire library by Paul Stoffregen**.



3. Then, search for “**Dallas**” and install the Dallas Temperature library by Miles Burton.



4. After installing the libraries, restart your Arduino IDE.

Code (Single DS18B20)

After installing the required libraries, you can upload the following code to the ESP8266. The code reads temperature from the DS18B20 temperature sensor and displays the readings on the Arduino IDE Serial Monitor.

```
#include <OneWire.h>
#include <DallasTemperature.h>

// GPIO where the DS18B20 is connected to
const int oneWireBus = 4;

// Setup a OneWire instance to communicate with any OneWire devices
OneWire oneWire(oneWireBus);

// Pass our OneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);

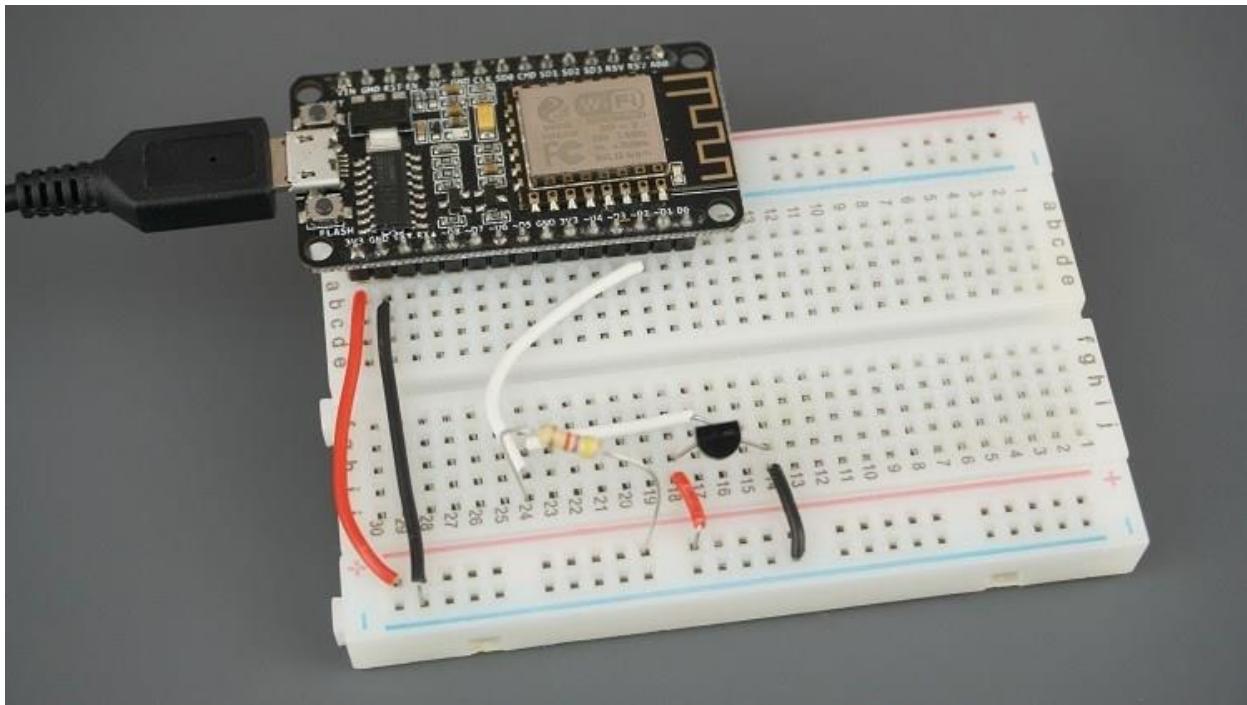
void setup() {
    // Start the Serial Monitor
    Serial.begin(115200);
    // Start the DS18B20 sensor
    sensors.begin();
}
```

```
void loop() {
    sensors.requestTemperatures();
    float temperatureC = sensors.getTempCByIndex(0);
    float temperatureF = sensors.getTempFByIndex(0);
    Serial.print(temperatureC);
    Serial.println("°C");
    Serial.print(temperatureF);
    Serial.println("°F");
    delay(5000);
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module3/Unit3/DS18B20_Single/DS18B20_Single.ino

There are different ways to get the temperature from DS18B20 temperature sensors. However, if you're using just one single sensor, this is one of the easiest.



How the Code Works

Start by including the OneWire and the DallasTemperature libraries.

```
#include <OneWire.h>
#include <DallasTemperature.h>
```

Create the instances needed for the temperature sensor. The temperature sensor is connected to GPIO 4.

```
// GPIO where the DS18B20 is connected to
const int oneWireBus = 4;

// Setup a OneWire instance to communicate with any OneWire devices
OneWire oneWire(oneWireBus);

// Pass our OneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);
```

In the `setup()`, initialize the Serial Monitor at a baud rate of 115200.

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

Initialize the DS18B20 temperature sensor:

```
sensors.begin();
```

Before actually getting the temperature, you need to call the `requestTemperatures()` method.

```
sensors.requestTemperatures();
```

Then, get the temperature in Celsius by using the `getTempCByIndex()` method as shown below:

```
float temperatureC = sensors.getTempCByIndex(0);
```

Or use the `getTempFByIndex()` to get the temperature in Fahrenheit.

```
float temperatureF = sensors.getTempFByIndex(0);
```

The `getTempCByIndex()` and the `getTempFByIndex()` methods accept the index of the temperature sensor. Because we're using just one sensor its index is 0. If you

want to read more than one sensor, you use index 0 for one sensor, index 1 for another sensor and so on.

Finally, print the results in the Serial Monitor.

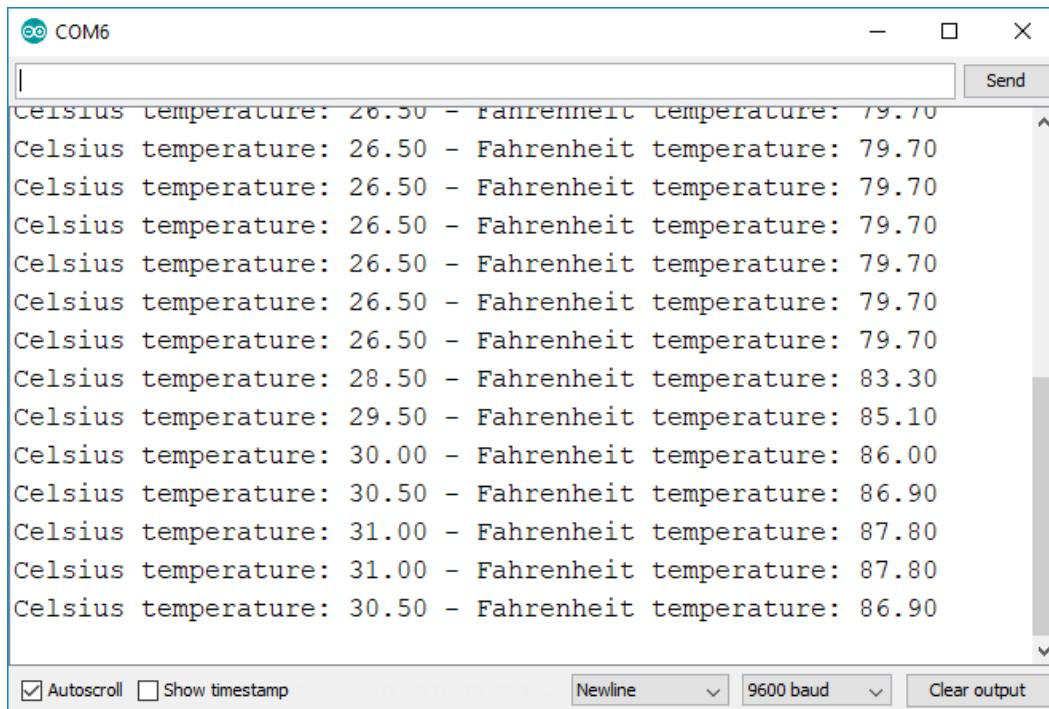
```
Serial.print(temperatureC);
Serial.println("°C");
Serial.print(temperatureF);
Serial.println("°F");
```

New temperature readings are requested every 5 seconds.

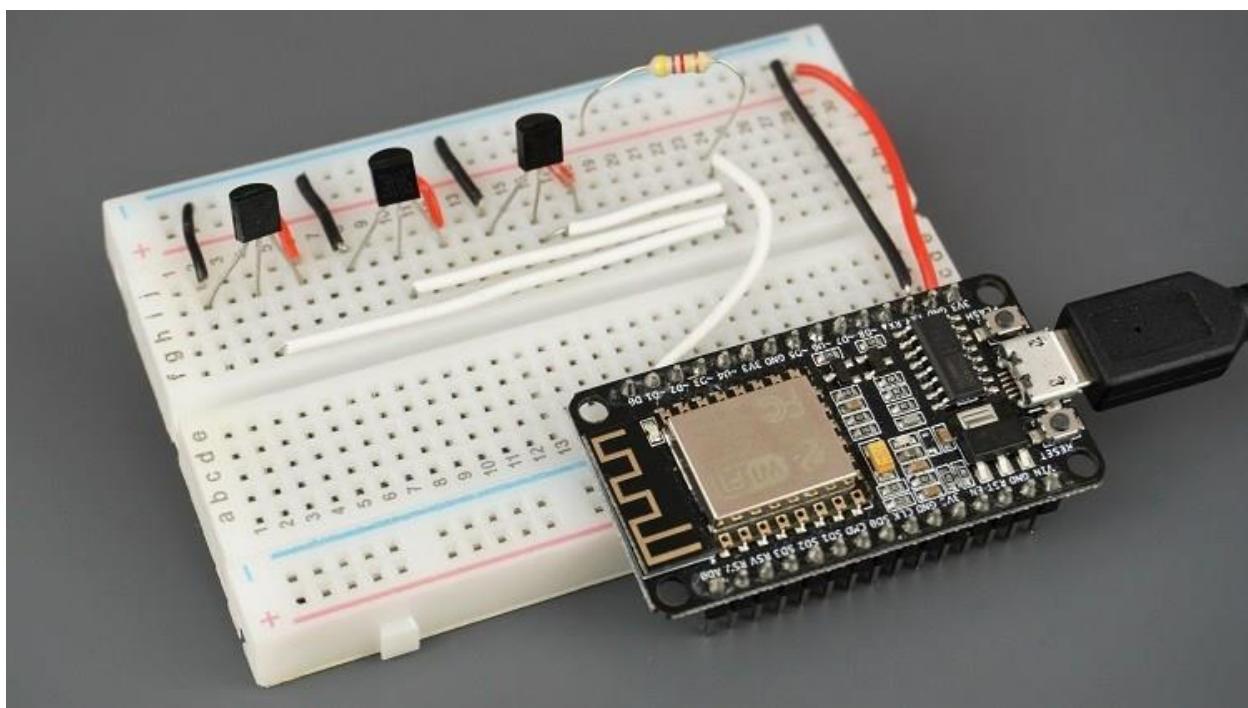
```
delay(5000);
```

Demonstration

After uploading the code, open the Arduino IDE Serial Monitor at a 9600 baud rate. You should get the temperature displayed in both Celsius and Fahrenheit:



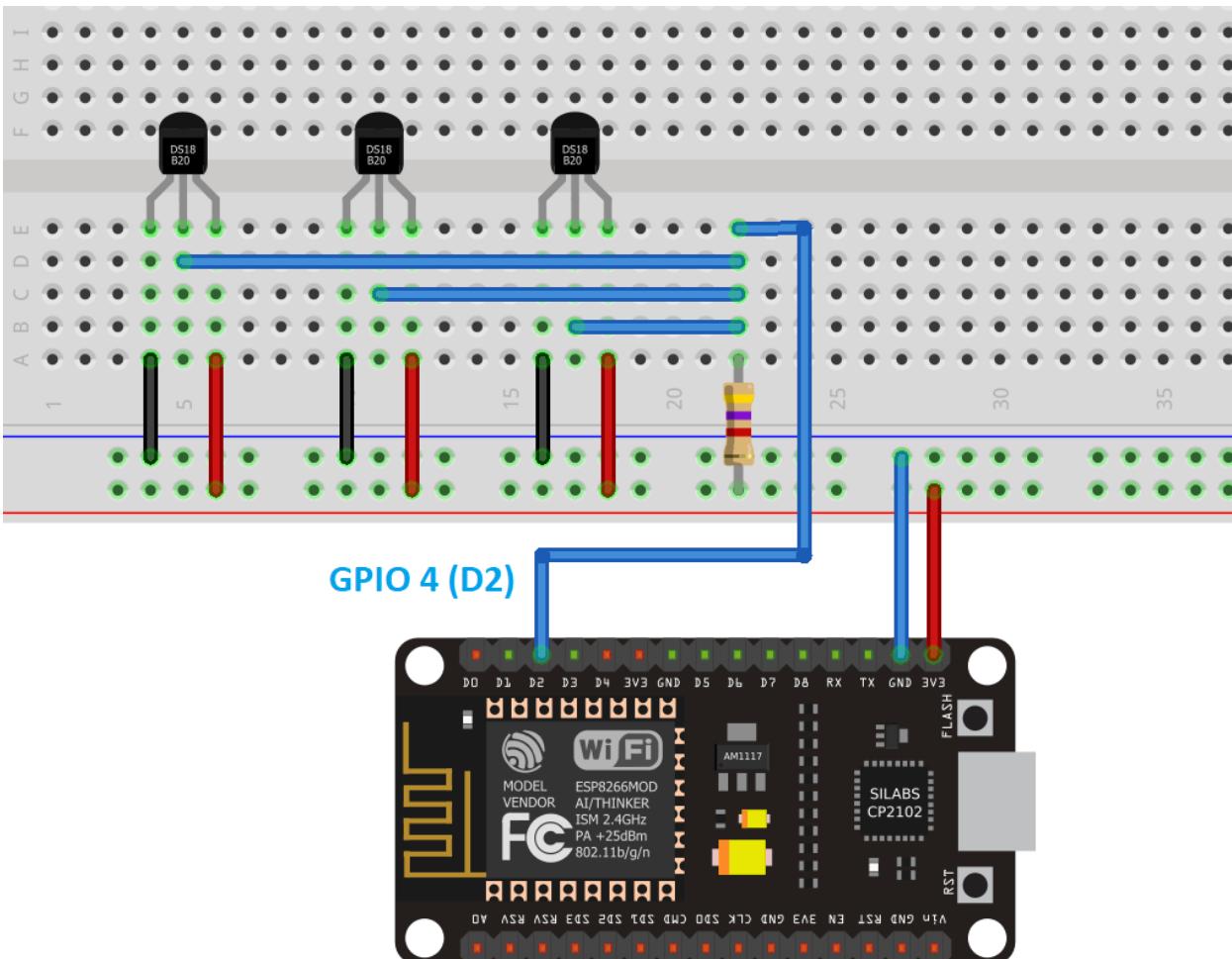
Getting Temperature from Multiple DS18B20 Temperature Sensors



The DS18B20 temperature sensor communicates using one-wire protocol and each sensor has a unique 64-bit serial code, so you can read the temperature from multiple sensors using just one single digital Pin.

Schematic

To read the temperature from multiple sensors, you just need to wire all data lines together as shown in the next schematic diagram:



Code (Multiple DS18B20s)

Then, upload the following code. It scans for all devices on GPIO 4 and prints the temperature for each one. This sketch is based on the example provided by the DallasTemperature library.

```
#include <OneWire.h>
#include <DallasTemperature.h>

// Data wire is plugged TO GPIO 4
#define ONE_WIRE_BUS 4

// Setup a OneWire instance to communicate with any OneWire devices (not
// just Maxim/Dallas temperature ICs)
OneWire oneWire(ONE_WIRE_BUS);

// Pass our OneWire reference to Dallas Temperature.
DallasTemperature sensors(&oneWire);
```

```

// Number of temperature devices found
int numberOfDevices;

// We'll use this variable to store a found device address
DeviceAddress tempDeviceAddress;

void setup(){
    // start serial port
    Serial.begin(115200);

    // Start up the library
    sensors.begin();

    // Grab a count of devices on the wire
    numberOfDevices = sensors.getDeviceCount();

    // locate devices on the bus
    Serial.print("Locating devices...");
    Serial.print("Found ");
    Serial.print(numberOfDevices, DEC);
    Serial.println(" devices.");

    // Loop through each device, print out address
    for(int i=0;i<numberOfDevices; i++){
        // Search the wire for address
        if(sensors.getAddress(tempDeviceAddress, i)){
            Serial.print("Found device ");
            Serial.print(i, DEC);
            Serial.print(" with address: ");
            printAddress(tempDeviceAddress);
            Serial.println();
        } else {
            Serial.print("Found ghost device at ");
            Serial.print(i, DEC);
            Serial.print(" but could not detect address. Check power and
cabling");
        }
    }
}

void loop(){
    sensors.requestTemperatures(); // Send the command to get temperatures

    // Loop through each device, print out temperature data
    for(int i=0;i<numberOfDevices; i++){
        // Search the wire for address
        if(sensors.getAddress(tempDeviceAddress, i)){
            // Output the device ID
            Serial.print("Temperature for device: ");
            Serial.println(i,DEC);
            // Print the data
            float tempC = sensors.getTempC(tempDeviceAddress);
            Serial.print("Temp C: ");
            Serial.print(tempC);
            Serial.print(" Temp F: ");

```

```

        Serial.println(DallasTemperature::toFahrenheit(tempC));
    }
}
delay(5000);
}

// function to print a device address
void printAddress(DeviceAddress deviceAddress) {
    for (uint8_t i = 0; i < 8; i++) {
        if (deviceAddress[i] < 16) Serial.print("0");
        Serial.print(deviceAddress[i], HEX);
    }
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module3/Unit3/DS18B20_Multiple/DS18B20_Multiple.ino

How the code works

The code uses several useful methods to handle multiple DS18B20 sensors.

You use the `getDeviceCount()` method to get the number of DS18B20 sensors on the data line.

`numberOfDevices = sensors.getDeviceCount();`

The `getAddress()` method finds the sensors addresses:

`if(sensors.getAddress(tempDeviceAddress, i)) {`

The address is unique for each sensor. So, each sensor can be identified by its address.

Then, you use the `getTempC()` method that accepts as argument the device address.

With this method, you can get the temperature from a specific sensor:

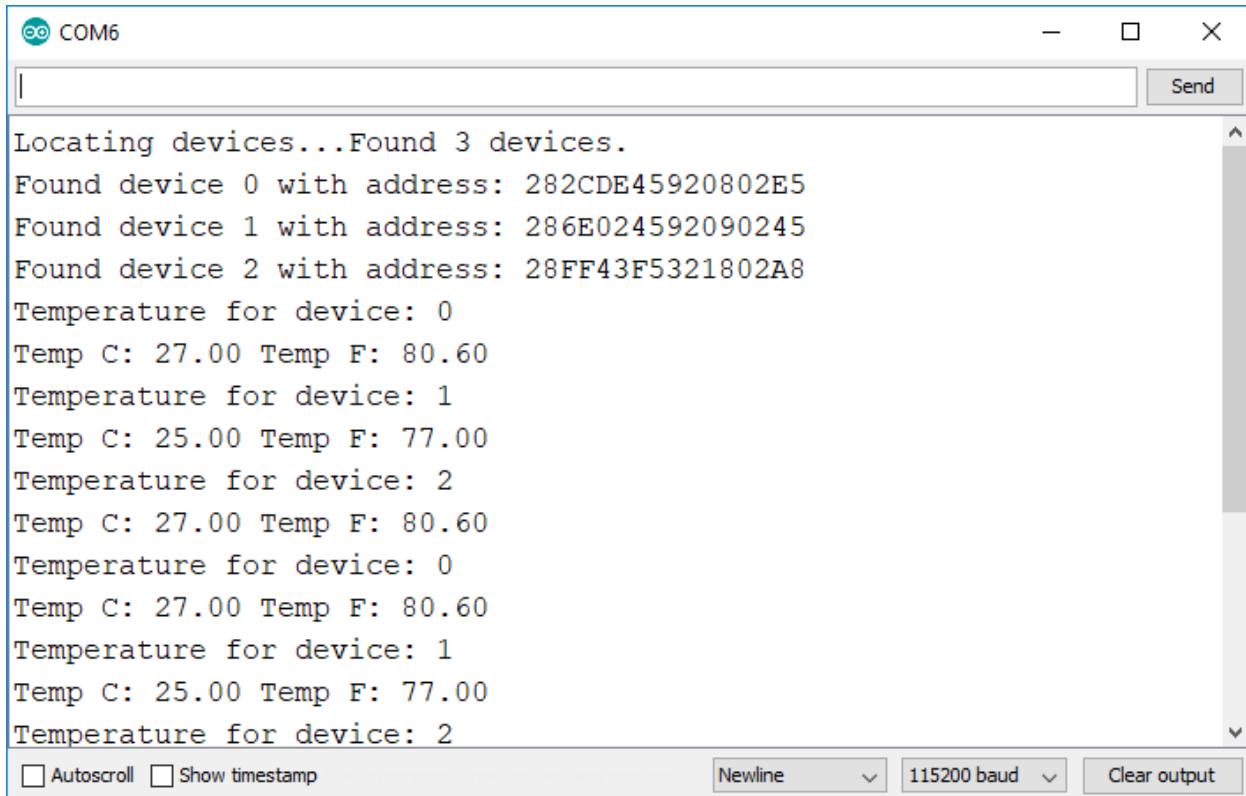
`float tempC = sensors.getTempC(tempDeviceAddress);`

To get the temperature in Fahrenheit degrees, you can use the `getTemF()`. Alternatively, you can convert the temperature in Celsius to Fahrenheit as follows:

```
DallasTemperature::toFahrenheit(tempC);
```

Demonstration

After uploading the code, open your Serial Monitor at a baud rate of 115200. You should get the sensor readings from all your sensors.

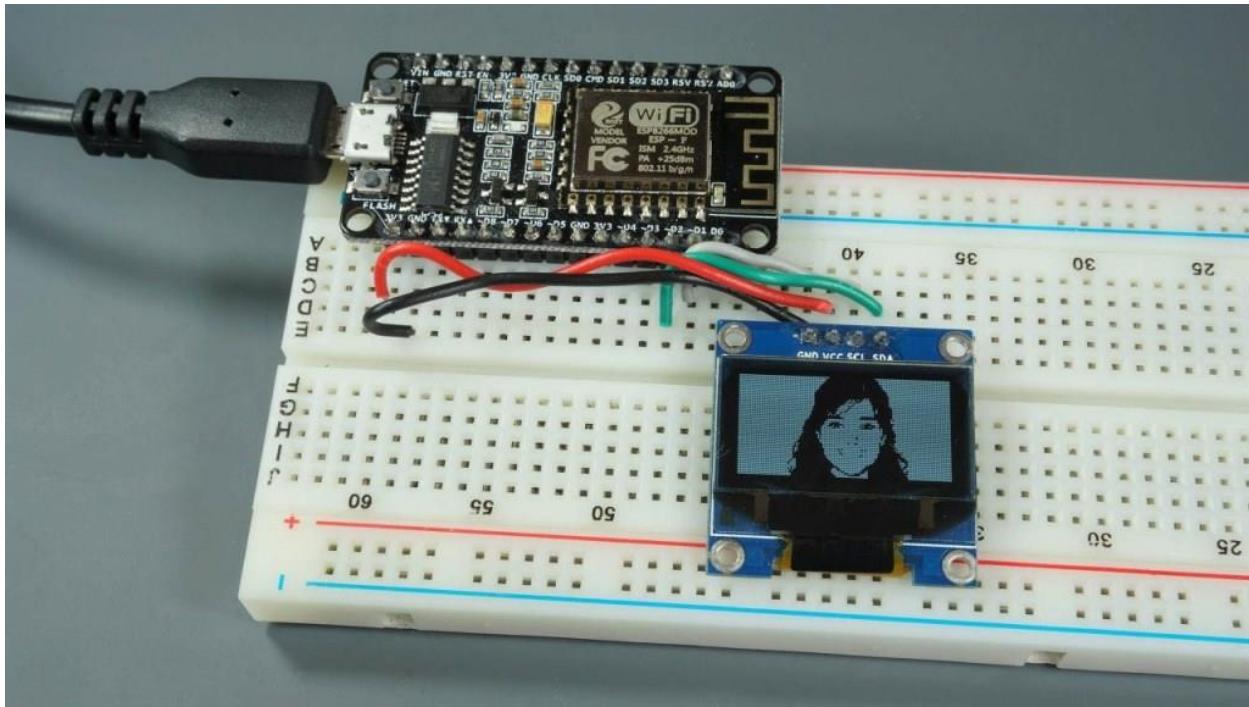


The screenshot shows the Arduino Serial Monitor window titled "COM6". The window displays the output of a sketch that locates devices and prints their addresses and temperature readings. The text in the monitor is as follows:

```
Locating devices...Found 3 devices.  
Found device 0 with address: 282CDE45920802E5  
Found device 1 with address: 286E024592090245  
Found device 2 with address: 28FF43F5321802A8  
Temperature for device: 0  
Temp C: 27.00 Temp F: 80.60  
Temperature for device: 1  
Temp C: 25.00 Temp F: 77.00  
Temperature for device: 2  
Temp C: 27.00 Temp F: 80.60  
Temperature for device: 0  
Temp C: 27.00 Temp F: 80.60  
Temperature for device: 1  
Temp C: 25.00 Temp F: 77.00  
Temperature for device: 2
```

At the bottom of the window, there are three checkboxes: "Autoscroll", "Show timestamp", and "Clear output". There are also dropdown menus for "Newline" and "115200 baud".

Unit 4: 0.96 inch OLED Display

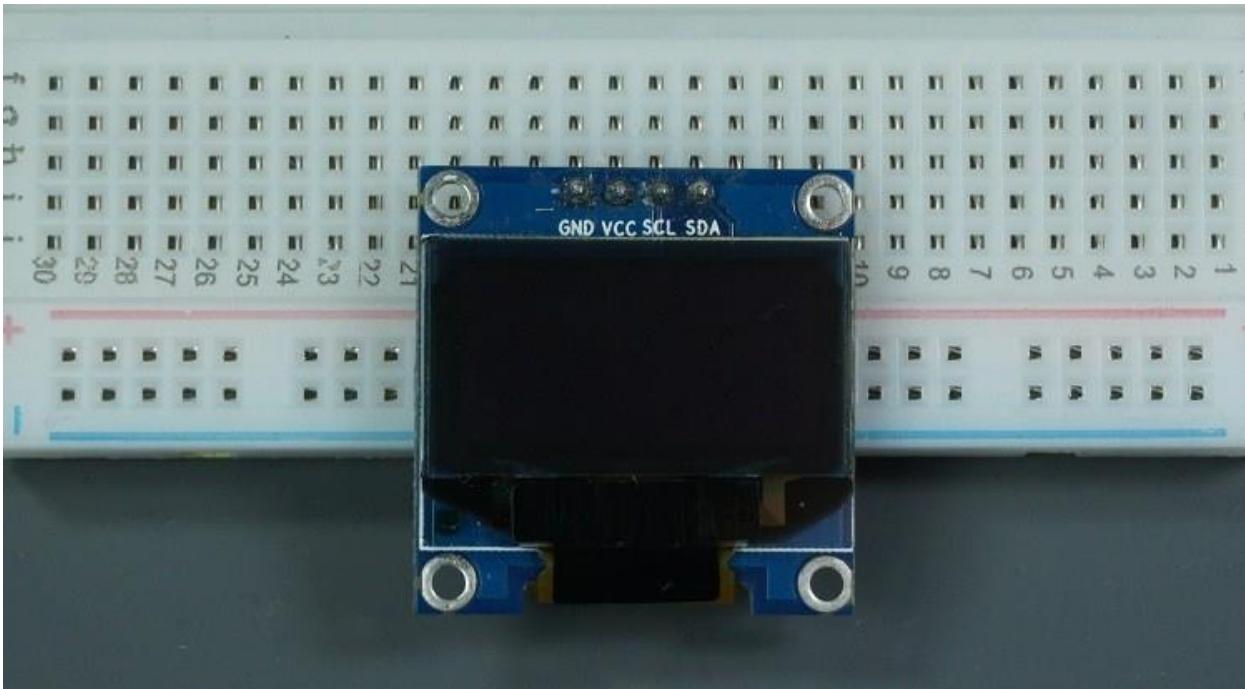


In this section, we'll show you how to use the 0.96 inch SSD1306 OLED display with the ESP8266 using Arduino IDE. This is an in-depth guide: we'll show you how to write text, set different fonts, draw shapes and display bitmaps images. You don't need to follow all sections at once, instead you should use them as a reference for your future projects.

However, we recommend following the section that explains how to display text.

Introducing the 0.96 inch OLED Display

The OLED display that we'll use in this tutorial is the SSD1306 model: a monicolor, 0.96 inch display with 128×64 pixels as shown in the following figure.



The OLED display doesn't require backlight, which results in a very nice contrast in dark environments. Additionally, its pixels consume energy only when they are on, so the OLED display consumes less power when compared to other displays.

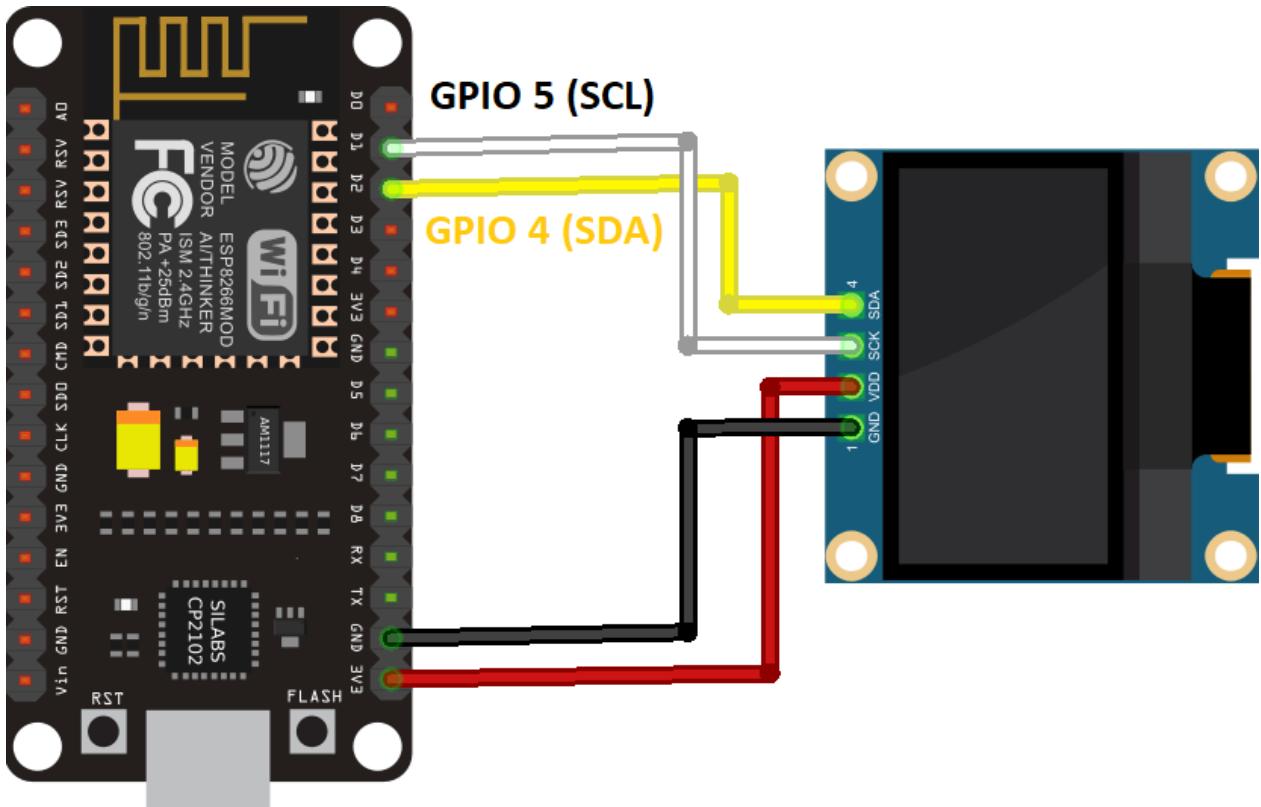
The model we're using has four pins and communicates with any microcontroller using I2C communication protocol. There are models that come with an extra RESET pin or that communicate using SPI communication protocol.

OLED Display SSD1306 Pin Wiring

Because the OLED display uses I2C communication protocol, wiring is very simple. You can use the following table as a reference.

OLED	ESP8266
Vin	3.3V
GND	GND
SCL	GPIO 5
SDA	GPIO 4

Alternatively, you can follow the next schematic diagram to wire the ESP8266 to the OLED display.



In this example, we're using I²C communication protocol. The most suitable pins for I²C communication with the ESP8266 are GPIO 5 (SCL) and GPIO 4 (SDA).

If you're using an OLED display with SPI communication protocol, use the following GPIOs.

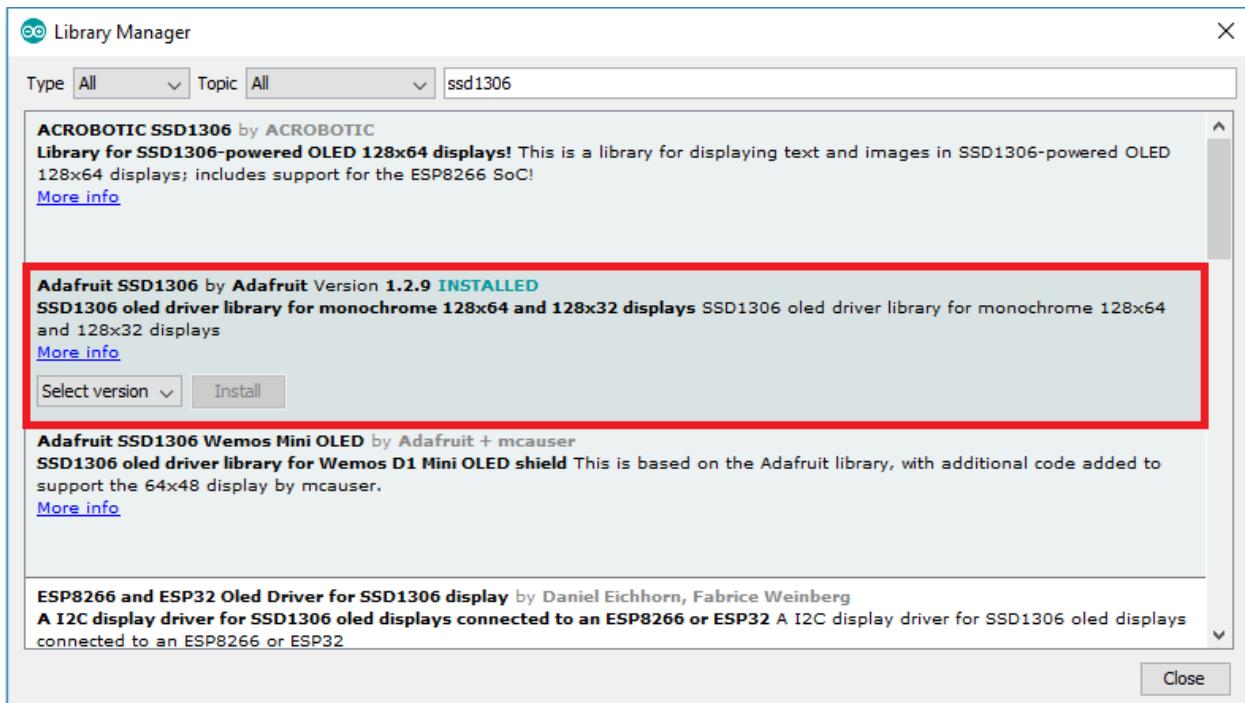
- GPIO 14: CLK
- GPIO 12: MISO
- GPIO 13: MOSI
- GPIO 15: CS

Installing SSD1306 OLED Library – ESP8266

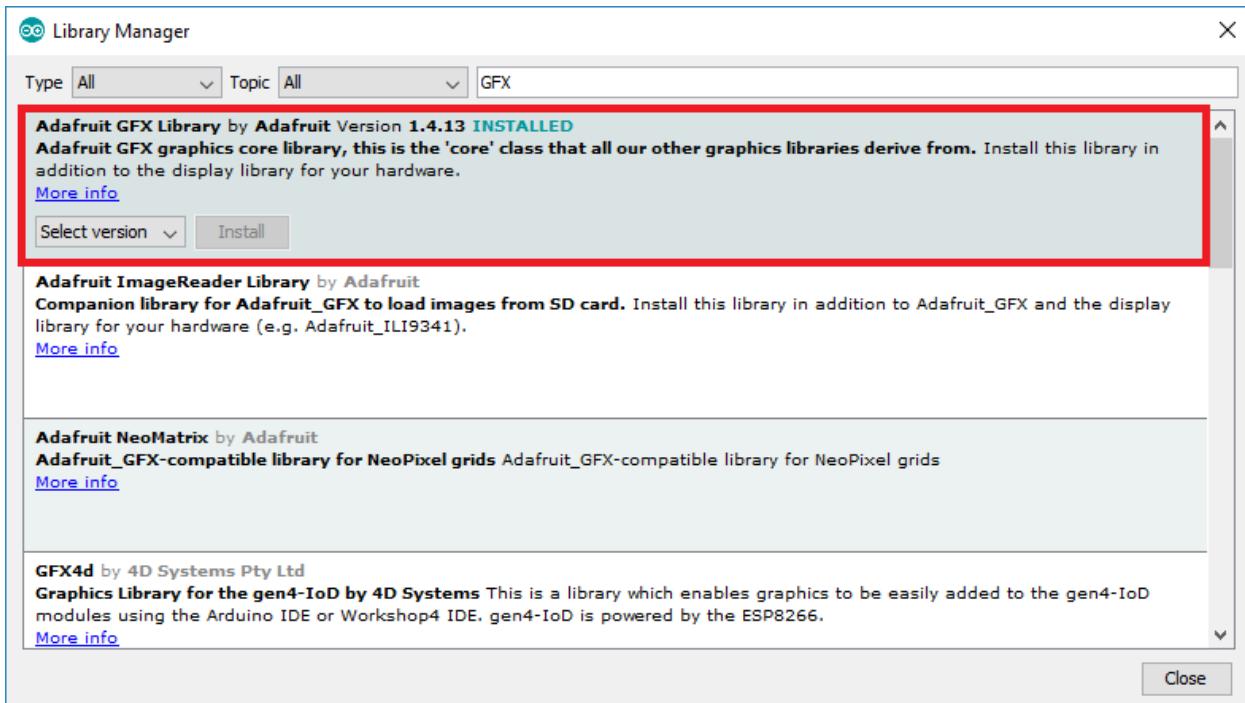
There are several libraries available to control the OLED display with the ESP8266. In this tutorial we'll use two Adafruit libraries: Adafruit_SSD1306 library and Adafruit_GFX library.

Follow the next steps to install those libraries.

1. Open your Arduino IDE and go to **Sketch > Include Library > Manage Libraries**. The Library Manager should open.
2. Type “**SSD1306**” in the search box and install the SSD1306 library from Adafruit.



3. After installing the SSD1306 library from Adafruit, type “**GFX**” in the search box and install the library.

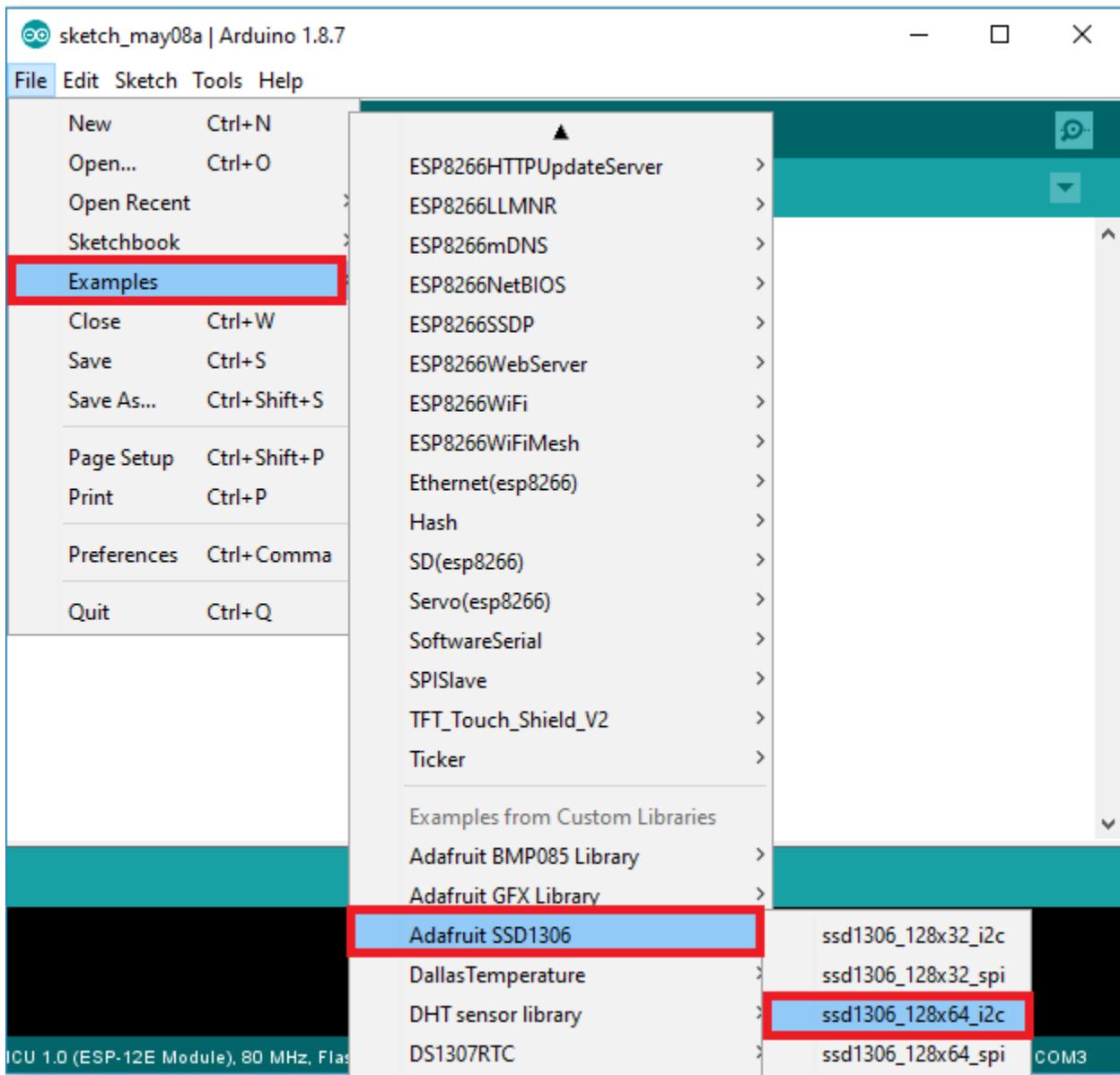


4. After installing the libraries, restart your Arduino IDE.

Testing OLED Display with ESP8266

After wiring the OLED display to the ESP8266 and installing all required libraries, you can use one example from the library to see if everything is working properly.

In your Arduino IDE, go to **File** ▶ **Examples** ▶ **Adafruit SSD1306** and select the example for the display you're using.



The following code should load:

```
/*
  This is an example for our Monochrome OLEDs based on SSD1306 drivers. Pick
  one up today in the adafruit shop! -----> www.adafruit.com/category/63\_98
  This example is for a 128x32 pixel display using I2C to communicate 3 pins
  are required to interface (two I2C and one reset).
  Adafruit invests time and resources providing this open source code, please
  support Adafruit and open-source hardware by purchasing products from Adafruit!
  Written by Limor Fried/Ladyada for Adafruit Industries, with contributions
  from the open source community. BSD license, check license.txt for more
  information All text above, and the splash screen below must be included in any
  redistribution.
***/
```

```
#include <SPI.h>
#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)
#define OLED_RESET -1 // Reset pin # (or -1 if sharing Arduino reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

#define NUMFLAKES 10 // Number of snowflakes in the animation example

#define LOGO_HEIGHT 16
#define LOGO_WIDTH 16
static const unsigned char PROGMEM logo_bmp[] =
{ B00000000, B11000000,
  B00000001, B11000000,
  B00000001, B11000000,
  B00000011, B11100000,
  B11110011, B11100000,
  B11111110, B11111000,
  B01111110, B11111111,
  B00110011, B10011111,
  B00011111, B11111100,
  B00001101, B01110000,
  B00011011, B10100000,
  B00111111, B11100000,
  B00111111, B11110000,
  B01111100, B11110000,
  B01110000, B01110000,
  B00000000, B00110000 };

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

  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;) // Don't proceed, loop forever
  }

  // Show initial display buffer contents on the screen --
  // the library initializes this with an Adafruit splash screen.
  display.display();
  delay(2000); // Pause for 2 seconds

  // Clear the buffer
  display.clearDisplay();

  // Draw a single pixel in white
  display.drawPixel(10, 10, WHITE);

  // Show the display buffer on the screen. You MUST call display() after
  // drawing commands to make them visible on screen!
  display.display();
  delay(2000);
  // display.display() is NOT necessary after every single drawing command,
  // unless that's what you want...rather, you can batch up a bunch of
  // drawing operations and then update the screen all at once by calling
  // display.display(). These examples demonstrate both approaches...

  testdrawline();      // Draw many lines
}

```

```

testdrawrect();           // Draw rectangles (outlines)
testfillrect();           // Draw rectangles (filled)
testdrawcircle();          // Draw circles (outlines)
testfillcircle();          // Draw circles (filled)
testdrawroundrect(); // Draw rounded rectangles (outlines)
testfillroundrect(); // Draw rounded rectangles (filled)
testdrawtriangle(); // Draw triangles (outlines)
testfilltriangle(); // Draw triangles (filled)
testdrawchar();           // Draw characters of the default font
testdrawstyles();          // Draw 'stylized' characters
testscrolltext();          // Draw scrolling text
testdrawbitmap();          // Draw a small bitmap image

// Invert and restore display, pausing in-between
display.invertDisplay(true);
delay(1000);
display.invertDisplay(false);
delay(1000);

testanimate(logo_bmp, LOGO_WIDTH, LOGO_HEIGHT); // Animate bitmaps
}

void loop() {
}

void testdrawline() {
    int16_t i;

    display.clearDisplay(); // Clear display buffer

    for(i=0; i<display.width(); i+=4) {
        display.drawLine(0, 0, i, display.height()-1, WHITE);
        display.display(); // Update screen with each newly-drawn line
        delay(1);
    }
    for(i=0; i<display.height(); i+=4) {
        display.drawLine(0, 0, display.width()-1, i, WHITE);
        display.display();
        delay(1);
    }
    delay(250);

    display.clearDisplay();

    for(i=0; i<display.width(); i+=4) {
        display.drawLine(0, display.height()-1, i, 0, WHITE);
        display.display();
        delay(1);
    }
}

```

```

        for(i=display.height()-1; i>=0; i-=4) {
            display.drawLine(0, display.height()-1, display.width()-1, i, WHITE);
            display.display();
            delay(1);
        }
        delay(250);

        display.clearDisplay();

        for(i=display.width()-1; i>=0; i-=4) {
            display.drawLine(display.width()-1, display.height()-1, i, 0, WHITE);
            display.display();
            delay(1);
        }
        for(i=display.height()-1; i>=0; i-=4) {
            display.drawLine(display.width()-1, display.height()-1, 0, i, WHITE);
            display.display();
            delay(1);
        }
        delay(250);

        display.clearDisplay();

        for(i=0; i<display.height(); i+=4) {
            display.drawLine(display.width()-1, 0, 0, i, WHITE);
            display.display();
            delay(1);
        }
        for(i=0; i<display.width(); i+=4) {
            display.drawLine(display.width()-1, 0, i, display.height()-1, WHITE);
            display.display();
            delay(1);
        }
    }

    delay(2000); // Pause for 2 seconds
}

void testdrawrect(void) {
    display.clearDisplay();

    for(int16_t i=0; i<display.height()/2; i+=2) {
        display.drawRect(i, i, display.width()-2*i, display.height()-2*i, WHITE);
        display.display(); // Update screen with each newly-drawn rectangle
        delay(1);
    }

    delay(2000);
}

void testfillrect(void) {
    display.clearDisplay();

    for(int16_t i=0; i<display.height()/2; i+=3) {
        // The INVERSE color is used so rectangles alternate white/black
        display.fillRect(i, i, display.width()-i*2, display.height()-i*2,
INVERSE);
        display.display(); // Update screen with each newly-drawn rectangle
        delay(1);
    }

    delay(2000);
}

```

```

}

void testdrawcircle(void) {
    display.clearDisplay();

    for(int16_t i=0; i<max(display.width(),display.height())/2; i+=2) {
        display.drawCircle(display.width()/2, display.height()/2, i, WHITE);
        display.display();
        delay(1);
    }

    delay(2000);
}

void testfillcircle(void) {
    display.clearDisplay();

    for(int16_t i=max(display.width(),display.height())/2; i>0; i-=3) {
        // The INVERSE color is used so circles alternate white/black
        display.fillCircle(display.width() / 2, display.height() / 2, i, INVERSE);
        display.display(); // Update screen with each newly-drawn circle
        delay(1);
    }

    delay(2000);
}

void testdrawroundrect(void) {
    display.clearDisplay();

    for(int16_t i=0; i<display.height()/2-2; i+=2) {
        display.drawRoundRect(i, i, display.width()-2*i, display.height()-2*i,
            display.height()/4, WHITE);
        display.display();
        delay(1);
    }

    delay(2000);
}

void testfillroundrect(void) {
    display.clearDisplay();

    for(int16_t i=0; i<display.height()/2-2; i+=2) {
        // The INVERSE color is used so round-rects alternate white/black
        display.fillRoundRect(i, i, display.width()-2*i, display.height()-2*i,
            display.height()/4, INVERSE);
        display.display();
        delay(1);
    }

    delay(2000);
}

void testdrawtriangle(void) {
    display.clearDisplay();

    for(int16_t i=0; i<max(display.width(),display.height())/2; i+=5) {
        display.drawTriangle(
            display.width()/2 , display.height()/2-i,
            display.width()/2-i, display.height()/2+i,
            display.width()/2+i, display.height()/2+i,
    }
}

```

```

        display.width()/2+i, display.height()/2+i, WHITE);
    display.display();
    delay(1);
}

delay(2000);
}

void testfilltriangle(void) {
    display.clearDisplay();

for(int16_t i=max(display.width(),display.height())/2; i>0; i-=5) {
    // The INVERSE color is used so triangles alternate white/black
    display.fillTriangle(
        display.width()/2 , display.height()/2-i,
        display.width()/2-i, display.height()/2+i,
        display.width()/2+i, display.height()/2+i, INVERSE);
    display.display();
    delay(1);
}

delay(2000);
}

void testdrawchar(void) {
    display.clearDisplay();

    display.setTextSize(1);      // Normal 1:1 pixel scale
    display.setTextColor(WHITE); // Draw white text
    display.setCursor(0, 0);    // Start at top-left corner
    display.cp437(true);       // Use full 256 char 'Code Page 437' font

    // Not all the characters will fit on the display. This is normal.
    // Library will draw what it can and the rest will be clipped.
    for(int16_t i=0; i<256; i++) {
        if(i == '\n') display.write(' ');
        else           display.write(i);
    }

    display.display();
    delay(2000);
}

void testdrawstyles(void) {
    display.clearDisplay();

    display.setTextSize(1);      // Normal 1:1 pixel scale
    display.setTextColor(WHITE); // Draw white text
    display.setCursor(0,0);     // Start at top-left corner
    display.println(F("Hello, world!"));

    display.setTextColor(BLACK, WHITE); // Draw 'inverse' text
    display.println(3.141592);

    display.setTextSize(2);      // Draw 2X-scale text
    display.setTextColor(WHITE);
    display.print(F("0x")); display.println(0xDEADBEEF, HEX);

    display.display();
    delay(2000);
}

```

```

void testscrolltext(void) {
    display.clearDisplay();

    display.setTextSize(2); // Draw 2X-scale text
    display.setTextColor(WHITE);
    display.setCursor(10, 0);
    display.println(F("scroll"));
    display.display(); // Show initial text
    delay(100);

    // Scroll in various directions, pausing in-between:
    display.startscrollright(0x00, 0x0F);
    delay(2000);
    display.stopscroll();
    delay(1000);
    display.startscrollleft(0x00, 0x0F);
    delay(2000);
    display.stopscroll();
    delay(1000);
    display.startscrolldiagright(0x00, 0x07);
    delay(2000);
    display.startscrolldiagleft(0x00, 0x07);
    delay(2000);
    display.stopscroll();
    delay(1000);
}

void testdrawbitmap(void) {
    display.clearDisplay();

    display.drawBitmap(
        (display.width() - LOGO_WIDTH) / 2,
        (display.height() - LOGO_HEIGHT) / 2,
        logo_bmp, LOGO_WIDTH, LOGO_HEIGHT, 1);
    display.display();
    delay(1000);
}

#define XPOS 0 // Indexes into the 'icons' array in function below
#define YPOS 1
#define DELTAY 2

void testanimate(const uint8_t *bitmap, uint8_t w, uint8_t h) {
    int8_t f, icons[NUMFLAKES][3];

    // Initialize 'snowflake' positions
    for(f=0; f< NUMFLAKES; f++) {
        icons[f][XPOS] = random(1 - LOGO_WIDTH, display.width());
        icons[f][YPOS] = -LOGO_HEIGHT;
        icons[f][DELTAY] = random(1, 6);
        Serial.print(F("x: "));
        Serial.print(icons[f][XPOS], DEC);
        Serial.print(F(" y: "));
        Serial.print(icons[f][YPOS], DEC);
        Serial.print(F(" dy: "));
        Serial.println(icons[f][DELTAY], DEC);
    }

    for(;;) { // Loop forever...
        display.clearDisplay(); // Clear the display buffer

```

```

// Draw each snowflake:
for(f=0; f< NUMFLAKES; f++) {
    display.drawBitmap(Icons[f][XPOS], Icons[f][YPOS], bitmap, w, h, WHITE);
}

display.display(); // Show the display buffer on the screen
delay(200); // Pause for 1/10 second

// Then update coordinates of each flake...
for(f=0; f< NUMFLAKES; f++) {
    Icons[f][YPOS] += Icons[f][DELTAY];
    // If snowflake is off the bottom of the screen...
    if (Icons[f][YPOS] >= display.height()) {
        // Reinitialize to a random position, just off the top
        Icons[f][XPOS] = random(1 - LOGO_WIDTH, display.width());
        Icons[f][YPOS] = -LOGO_HEIGHT;
        Icons[f][DELTAY] = random(1, 6);
    }
}
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module3/Unit4/OLED_Display_Adafuit_Example/OLED_Display_Adafuit_Example.ino

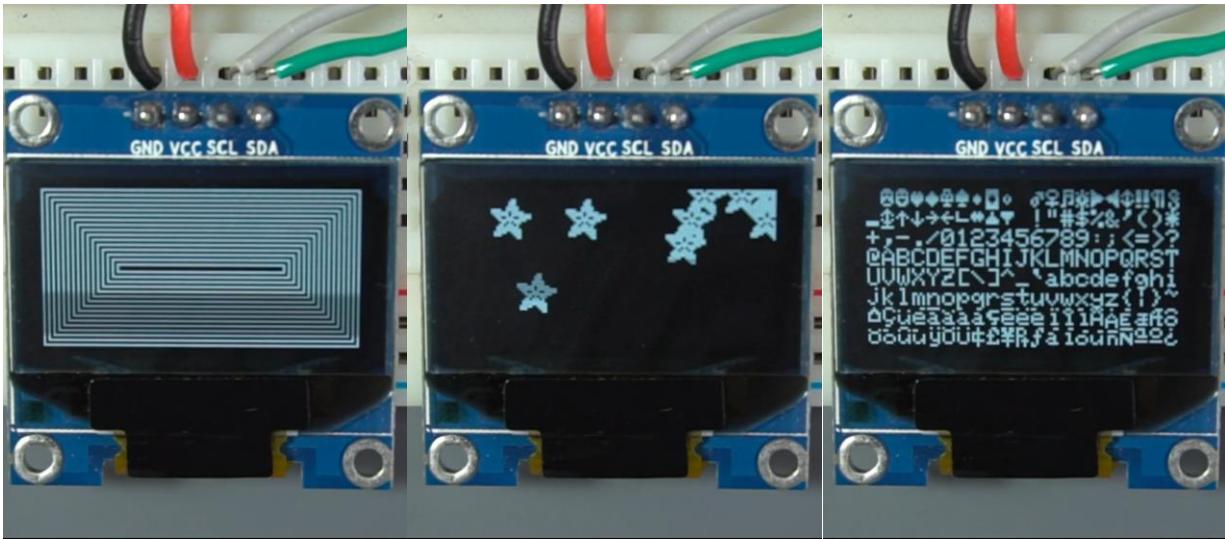
If your OLED doesn't have a RESET pin, you should set the `OLED_RESET` variable to -1 as shown below:

```
#define OLED_RESET -1 // Reset pin # (or -1 if sharing Arduino reset pin)
```

Note: if your OLED has a RESET pin, you should connect it to a different GPIO than GPIO 4, because that pin is being used for I2C communication in the EPS8266.

Upload the code to your ESP8266 board. Don't forget to select the right board and COM port in the **Tools** menu.

You should get a series of different animations in the OLED as shown in the following images.



If your OLED display is not showing anything:

- Check that the OLED display is properly wired to the ESP8266
- Double-check the OLED display I2C address: with the OLED connected to the ESP8266, [upload this I2C Scanner code](#) and check the I2C address in the Serial Monitor.

You should change the OLED address in the following line, if necessary. In our case, the address is 0x3C.

```
if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
```

Write Text

The Adafruit library for the OLED display comes with several functions to write text. In this section, you'll learn how to write and scroll text using the library functions.

“Hello, world!” OLED Display

The following sketch displays an “Hello, world!” message in the OLED display.

```
#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)) { // Address 0x3D for 128x64
        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() {
```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module3/Unit4/OLED_Hello_World/OLED_Hello_World.ino

After uploading the code, this is what you'll get in your OLED:



Let's take a quick look on how the code works.

Importing libraries

First, you need to import the necessary libraries. The `Wire` library to use I2C and the Adafruit libraries to write to the display: `Adafruit_GFX` and `Adafruit_SSD1306`.

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

Initialize the OLED display

Then, define your OLED width and height. In this example, we're using a 128×64 OLED display. If you're using other sizes, you can change that in the `SCREEN_WIDTH`, and `SCREEN_HEIGHT` variables.

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

Then, initialize a display object with the width and height defined earlier with I2C communication protocol (`&Wire`).

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

The (-1) parameter means that your OLED display doesn't have a RESET pin. If your OLED display does have a RESET pin, it should be connected to a GPIO. In that case, you should pass the GPIO number as an argument.

In the `setup()`, initialize the Serial Monitor at a baud rate of 115200 for debugging purposes.

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

Initialize the OLED display with the `begin()` method as follows:

```
if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {  
    Serial.println(F("SSD1306 allocation failed"));  
    for(;;);  
}
```

This snippet also prints a message on the Serial Monitor, in case we're not able to connect to the display.

```
Serial.println(F("SSD1306 allocation failed"));
```

In case you're using a different OLED display, you may need to change the OLED address. In our case, the address is **0x3C**.

```
if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
```

If this address doesn't work, you can run an I2C scanner sketch to find your OLED address. You can [find the I2C scanner sketch here](#).

After initializing the display, add a two second delay, so that the OLED has enough time to initialize before writing text:

```
delay(2000);
```

Clear display, set font size, color and write text

After initializing the display, clear the display buffer with the `clearDisplay()` method:

```
display.clearDisplay();
```

Before writing text, you need to set the text size, color and where the text will be displayed in the OLED.

Set the font size using the `setTextSize()` method:

```
display.setTextSize(1);
```

Set the font color with the `setTextColor()` method:

```
display.setTextColor(WHITE);
```

WHITE sets white font and black background.

Define the position where the text starts using the `setCursor(x, y)` method. In this case, we're setting the text to start at the (0,10) coordinates.

```
display.setCursor(0, 10);
```

Finally, you can send the text to the display using the `println()` method, as follows:

```
display.println("Hello, world!");
```

Then, you need to call the `display()` method to actually display the text on the screen.

```
display.display();
```

Scrolling Text

The Adafruit OLED library provides useful methods to easily scroll text.

- `startscrollright(0x00, 0x0F)`: scroll text from left to right
- `startscrollleft(0x00, 0x0F)`: scroll text from right to left
- `startscrolldiagright(0x00, 0x07)`: scroll text from left bottom corner to right upper corner
- `startscrolldiagleft(0x00, 0x07)`: scroll text from right bottom corner to left upper corner

The following sketch implements those methods.

```
#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)) { // Address 0x3D for 128x64
        Serial.println(F("SSD1306 allocation failed"));
        for(;;);
    }
    delay(2000);
    display.clearDisplay();

    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(0, 0);
    // Display static text
    display.println("Scrolling Hello");
    display.display();
    delay(100);

}

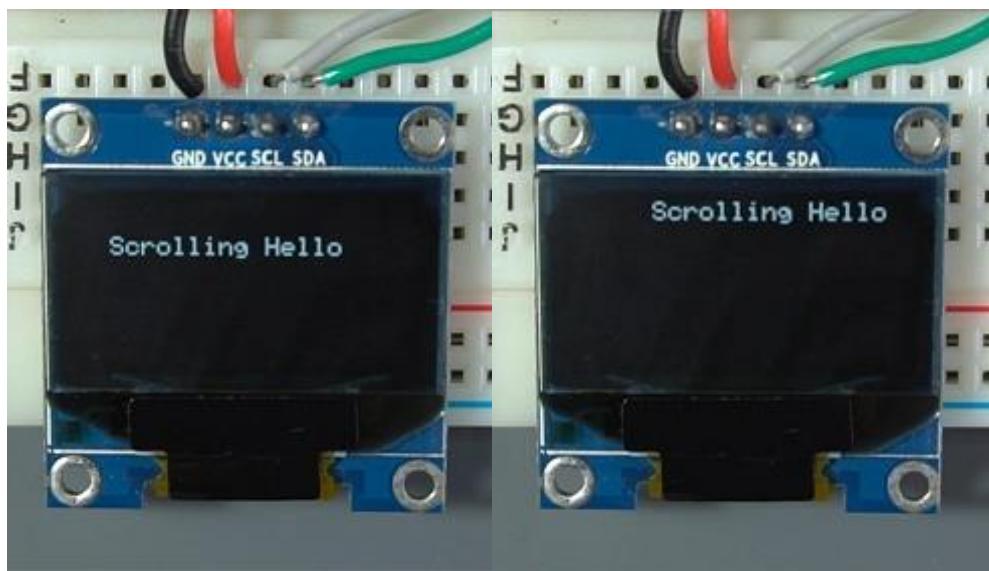
void loop() {
    // Scroll in various directions, pausing in-between:
    display.startscrollright(0x00, 0x0F);
    delay(2000);
    display.stopscroll();
    delay(1000);
    display.startscrollleft(0x00, 0x0F);
    delay(2000);
```

```
display.stopscroll();
delay(1000);
display.startscrolldiagright(0x00, 0x07);
delay(2000);
display.startscrolldiagleft(0x00, 0x07);
delay(2000);
display.stopscroll();
delay(1000);
}
```

SOURCE CODE

[https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module3/Unit4/OLED Scroll Text/OLED Scroll Text.ino](https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module3/Unit4/OLED%20Scroll%20Text/OLED%20Scroll%20Text.ino)

The text will scroll horizontally and diagonally.



Using Other Fonts – OLED Display

The Adafruit GFX library allows us to use some alternate fonts besides the built-in fonts. It allows you to choose between Serif, Sans, and Mono. Each font is available in bold, italic and in different sizes.

The sizes are set by the actual font. So, the `setTextSize()` method doesn't work with these fonts. The fonts are available in 9, 12, 18 and 24 point sizes and also contain 7-bit characters (ASCII codes) (described as 7b in the font name).

You can choose from the next selection of fonts:

FreeMono12pt7b.h	FreeSansBoldOblique12pt7b.h
FreeMono18pt7b.h	FreeSansBoldOblique18pt7b.h
FreeMono24pt7b.h	FreeSansBoldOblique24pt7b.h
FreeMono9pt7b.h	FreeSansBoldOblique9pt7b.h
FreeMonoBold12pt7b.h	FreeSansOblique12pt7b.h
FreeMonoBold18pt7b.h	FreeSansOblique18pt7b.h
FreeMonoBold24pt7b.h	FreeSansOblique24pt7b.h
FreeMonoBold9pt7b.h	FreeSansOblique9pt7b.h
FreeMonoBoldOblique12pt7b.h	FreeSerif12pt7b.h
FreeMonoBoldOblique18pt7b.h	FreeSerif18pt7b.h
FreeMonoBoldOblique24pt7b.h	FreeSerif24pt7b.h
FreeMonoBoldOblique9pt7b.h	FreeSerif9pt7b.h
FreeMonoOblique12pt7b.h	FreeSerifBold12pt7b.h
FreeMonoOblique18pt7b.h	FreeSerifBold18pt7b.h
FreeMonoOblique24pt7b.h	FreeSerifBold24pt7b.h
FreeMonoOblique9pt7b.h	FreeSerifBold9pt7b.h
FreeSans12pt7b.h	FreeSerifBoldItalic12pt7b.h
FreeSans18pt7b.h	FreeSerifBoldItalic18pt7b.h
FreeSans24pt7b.h	FreeSerifBoldItalic24pt7b.h
FreeSans9pt7b.h	FreeSerifBoldItalic9pt7b.h
FreeSansBold12pt7b.h	FreeSerifItalic12pt7b.h
FreeSansBold18pt7b.h	FreeSerifItalic18pt7b.h
FreeSansBold24pt7b.h	FreeSerifItalic24pt7b.h
FreeSansBold9pt7b.h	FreeSerifItalic9pt7b.h

The fonts that work better with the OLED display are the 9 and 12 points size.

To use one of those fonts, first you need to include it in your sketch, for example:

```
#include <Fonts/FreeSerif9pt7b.h>
```

Next, you just need to use the `setFont()` method and pass as argument, the specified font:

```
display.setFont(&FreeSerif9pt7b);
```

After specifying the font, all methods to write text will use that font. To get back to use the original font, you just need to call the `setFont()` method with no arguments:

```
display.setFont();
```

Upload the next sketch to your board:

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Fonts/FreeSerif9pt7b.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)) {
        Serial.println("SSD1306 allocation failed");
        for(;;);
    }
    delay(2000);

    display.setFont(&FreeSerif9pt7b);
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(0,20);
    display.println("Hello, world!");
    display.display();
    delay(2000);
}
void loop() {

}
```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module3/Unit4/OLED_Different_Font/OLED_Different_Font.ino

Now, your display prints the “Hello, world!” message in FreeSerif font.



Draw Shapes in the OLED Display

The Adafruit OLED library provides useful methods to draw pixels, lines and shapes.

Let's take a quick look at those methods.

Draw a pixel



To draw a pixel in the OLED display, you can use the `drawPixel(x, y, color)` method that accepts as arguments the x and y coordinates where the pixel appears, and color. For example:

```
display.drawPixel(64, 32, WHITE);
```

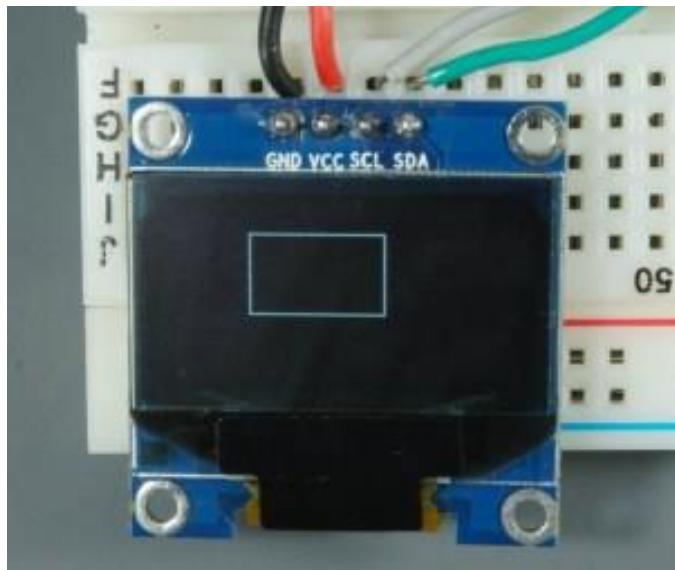
Draw a line



Use the `drawLine(x1, y1, x2, y2, color)` method to create a line. The (x_1, y_1) coordinates indicate the start of the line, and the (x_2, y_2) coordinates indicates where the line ends. For example:

```
display.drawLine(0, 0, 127, 20, WHITE);
```

Draw a rectangle



The `drawRect(x, y, width, height, color)` provides an easy way to draw a rectangle. The (x, y) coordinates indicate the top left corner of the rectangle. Then, you need to specify the width, height and color:

```
display.drawRect(30, 10, 50, 30, WHITE);
```

You can use the `fillRect(x, y, width, height, color)` to draw a filled rectangle. This method accepts the same arguments as `drawRect()`.

```
display.fillRect(30, 10, 50, 30, WHITE);
```



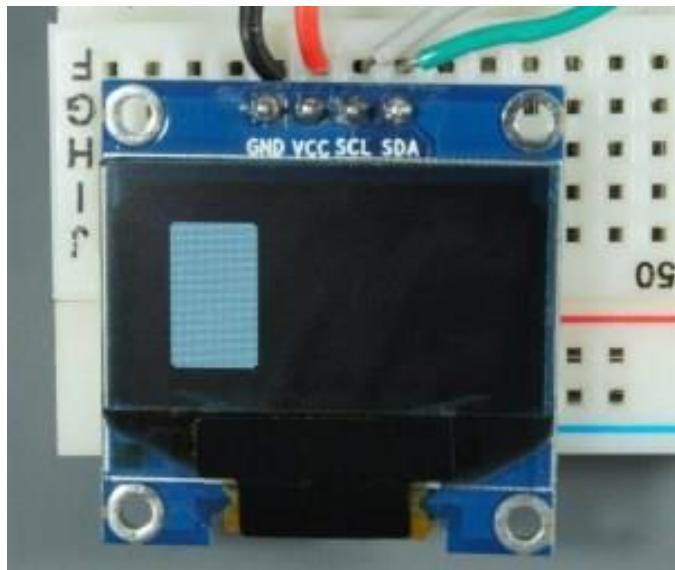
The library also provides methods to display rectangles with round corners: `drawRoundRect()` and `fillRoundRect()`. These methods accept the same arguments as previous methods plus the radius of the corner. For example:

```
display.drawRoundRect(10, 10, 30, 50, 2, WHITE);
```



Or a filled round rectangle:

```
display.fillRoundRect(10, 10, 30, 50, 2, WHITE);
```



Draw a circle



To draw a circle, use the `drawCircle(x, y, radius, color)` method. The (x,y) coordinates indicate the center of the circle. You should also pass the radius as an argument. For example:

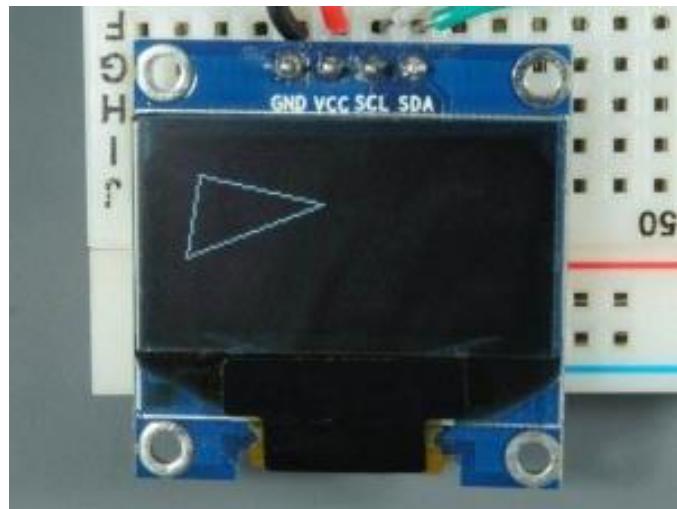
```
display.drawCircle(64, 32, 10, WHITE);
```

In the same way, to build a filled circle, use the `fillCircle()` method with the same arguments:

```
display.fillCircle(64, 32, 10, WHITE);
```



Draw a triangle

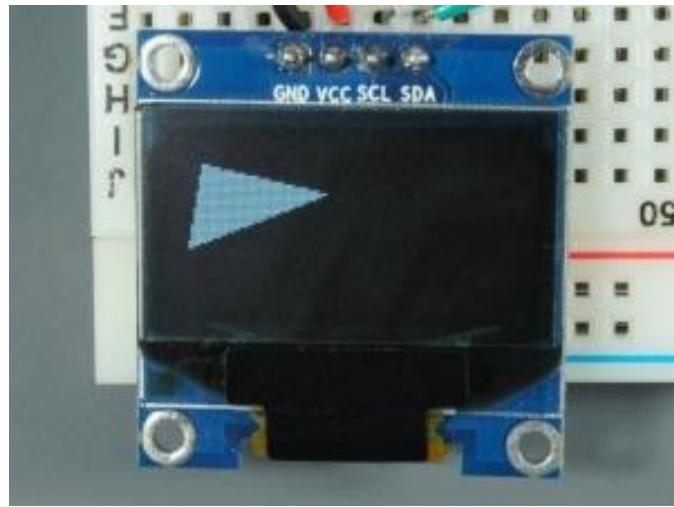


Use the `drawTriangle(x1, y1, x2, y2, x3, y3, color)` method to build a triangle. This method accepts as arguments the coordinates of each corner and the color.

```
display.drawTriangle(10, 10, 55, 20, 5, 40, WHITE);
```

Use the `fillTriangle()` method to draw a filled triangle.

```
display.fillTriangle(10, 10, 55, 20, 5, 40, WHITE);
```



Invert

The library provides an additional method that you can use with shapes or text: the `invertDisplay()` method. Pass `true` as argument to invert the colors of the screen or `false` to get back to the original colors. If you call the following command after defining the triangle:

```
display.invertDisplay(true);
```

You'll get an inverted triangle as follows:



Code - Draw Shapes

Upload the following sketch that implements each snippet of code we've covered previously and goes through all the shapes.

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

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

// 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)) {
        Serial.println(F("SSD1306 allocation failed"));
        for(;;);
    }
    delay(2000); // Pause for 2 seconds

    // Clear the buffer
    display.clearDisplay();

    // Draw a single pixel in white
    display.drawPixel(64, 32, WHITE);
    display.display();
    delay(3000);

    // Draw line
    display.clearDisplay();
    display.drawLine(0, 0, 127, 20, WHITE);
    display.display();
    delay(3000);

    // Draw rectangle
    display.clearDisplay();
    display.drawRect(30, 10, 50, 30, WHITE);
    display.display();
    delay(3000);
    // Fill rectangle
    display.fillRect(30, 10, 50, 30, WHITE);
    display.display();
    delay(3000);

    // Draw round rectangle
    display.clearDisplay();
    display.drawRoundRect(10, 10, 30, 50, 2, WHITE);
    display.display();
    delay(3000);
    // Fill round rectangle
    display.clearDisplay();
    display.fillRoundRect(10, 10, 30, 50, 2, WHITE);
    display.display();
```

```

delay(3000);

// Draw circle
display.clearDisplay();
display.drawCircle(64, 32, 10, WHITE);
display.display();
delay(3000);
// Fill circle
display.fillCircle(64, 32, 10, WHITE);
display.display();
delay(3000);

// Draw triangle
display.clearDisplay();
display.drawTriangle(10, 10, 55, 20, 5, 40, WHITE);
display.display();
delay(3000);
// Fill triangle
display.fillTriangle(10, 10, 55, 20, 5, 40, WHITE);
display.display();
delay(3000);

// Invert and restore display, pausing in-between
display.invertDisplay(true);
delay(3000);
display.invertDisplay(false);
delay(3000);
}

void loop() {
}

```

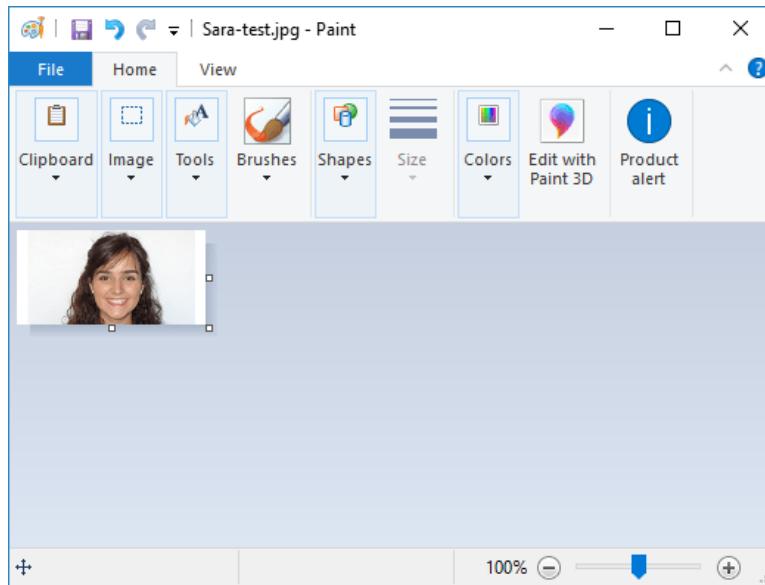
SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module3/Unit4/OLED_Draw_Shapes/OLED_Draw_Shapes.ino

Display Bitmap Images

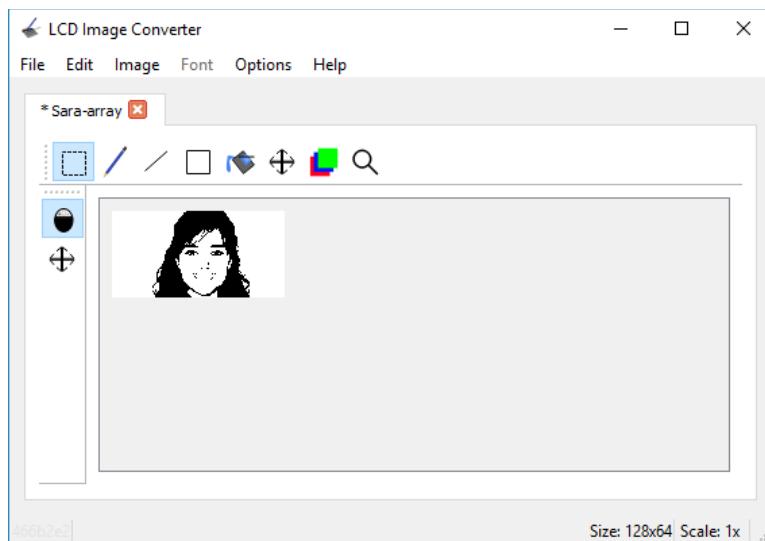
You can display 128x64 bitmap monocolored images on the OLED display.

First, use an imaging program to resize a photo or picture to 128x64 and save it as monochrome bitmap. If you're on a Windows PC, you can use Paint.



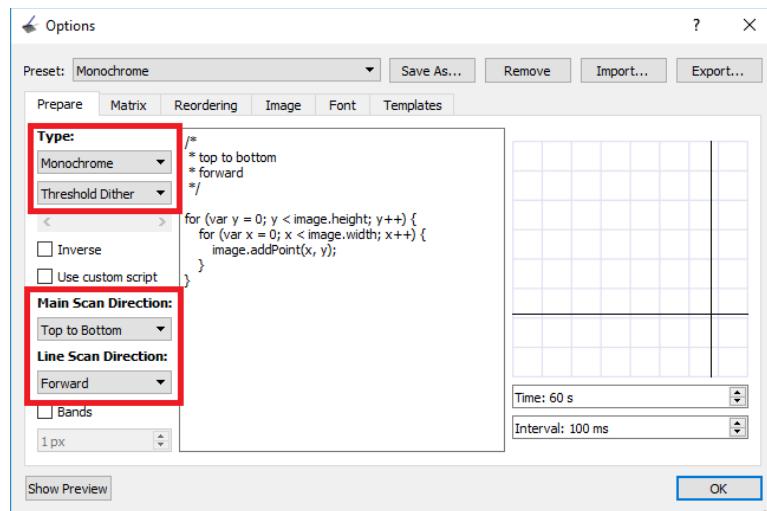
Then, use an Image to C Array converter to convert the image into an array. For example, [LCD Image Converter](#).

Run the program and start with a new image. Go to **Image** ▶ **Import** and select the bitmap image you've created earlier.



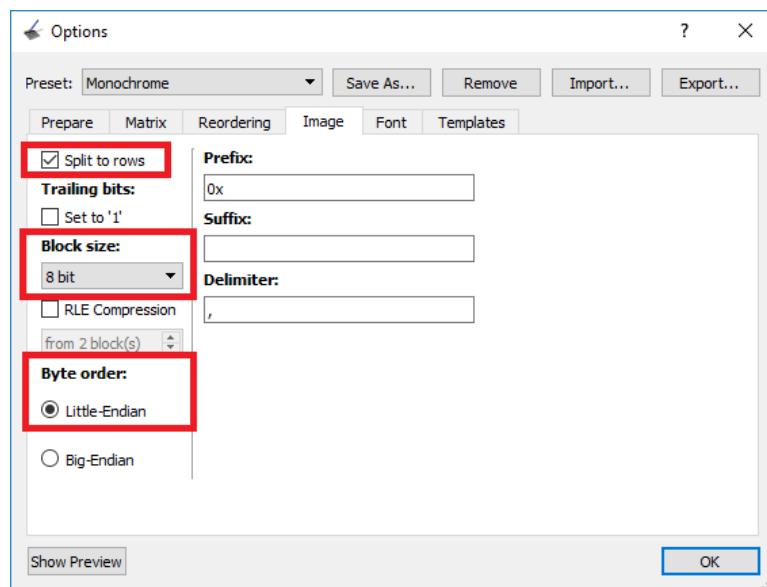
Go to **Options > Conversion** and in the **Prepare** tab, select the following options:

- **Type:** Monochrome, Threshold Dither
- **Main Scan Direction:** Top to Bottom
- **Line Scan Direction:** Forward



Go to the **Image** tab and select the following options:

- Split to rows
- **Block size:** 8 bit
- **Byte order:** Little-Endian



Then, click **OK**. Finally, in the main menu, go to **File > Convert**. A new file with .c extension should be saved. That file contains the C array for the image. Open that file with a text editor, and copy the array. The array that we get [can be found here](#).

Copy your array to the sketch. Then, to display the array, use the `drawBitmap()` method that accepts the following arguments (`x, y, image array, image width, image height, rotation`). The `(x, y)` coordinates define where the image starts to be displayed.

Copy the code below to display your bitmap image in the OLED.

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

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

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

static const uint8_t image_data_Saraarray[1024] = {
    0xff, 0xff, 0xff, 0xff, 0xff, 0xfe, 0x00, 0x00, 0x00, 0x1f, 0xff, 0xff, 0xff,
    0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf8, 0x00, 0x00, 0x00, 0x0f, 0xff, 0xff, 0xff,
    0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xf8, 0x00, 0x00, 0x00, 0x07, 0xff, 0xff, 0xff,
    0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0x00, 0x00, 0x00, 0x07, 0xff, 0xff, 0xff,
    0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x03, 0xff, 0xff, 0xff,
    0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xc0, 0x00, 0x00, 0x00, 0x01, 0xff, 0xff, 0xff,
    0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0x80, 0x00, 0x00, 0x00, 0x01, 0xff, 0xff, 0xff,
    0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0x80, 0x00, 0x00, 0x00, 0x00, 0xff, 0xff, 0xff,
    0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7f, 0xff, 0xff,
    0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3f, 0xff, 0xff,
    0xff, 0xff,
    0xff, 0xff, 0xff, 0xfc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3f, 0xff, 0xff,
    0xff, 0xff,
    0xff, 0xff, 0xff, 0xfc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1f, 0xff, 0xff,
    0xff, 0xff,
    0xff, 0xff, 0xff, 0xfc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1f, 0xff, 0xff,
    0xff, 0xff,
    0xff, 0xff, 0xff, 0xfc, 0x00, 0x00, 0x00, 0x00, 0x0a, 0x00, 0x00, 0x1f, 0xff, 0xff,
    0xff, 0xff,
    0xff, 0xff, 0xff, 0xfc, 0x00, 0x00, 0x14, 0x9e, 0x00, 0x1f, 0xff, 0xff,
    0xff, 0xff,
    0xff, 0xff, 0xff, 0xfc, 0x00, 0x00, 0x36, 0x3f, 0x00, 0x1f, 0xff, 0xff,
    0xff, 0xff,
    0xff, 0xff, 0xff, 0xfc, 0x00, 0x00, 0x6d, 0xff, 0x00, 0x1f, 0xff, 0xff,
    0xff, 0xff,
```

0xff, 0xff, 0xff, 0xff, 0xfc, 0x00, 0x00, 0xfb, 0xff, 0x80, 0x1f, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xfc, 0x00, 0x03, 0xd7, 0xff, 0x80, 0x0f, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xf8, 0x00, 0x07, 0xef, 0xff, 0x80, 0x07, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xf8, 0x00, 0x0f, 0xdf, 0xff, 0x90, 0x07, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xf8, 0x00, 0x0f, 0xbf, 0xff, 0xd0, 0x07, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xf0, 0x00, 0x1d, 0x7f, 0xff, 0xd0, 0x07, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xf0, 0x01, 0x1b, 0xff, 0xff, 0xc0, 0x07, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xf0, 0x02, 0xa7, 0xff, 0xff, 0xc0, 0x07, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xe0, 0x00, 0x03, 0xff, 0xc0, 0x00, 0x07, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xe0, 0x00, 0x00, 0xff, 0x80, 0x00, 0xb, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xc0, 0x03, 0xff, 0xff, 0xf0, 0x0f, 0xff, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0x80, 0x07, 0xff, 0xff, 0xf0, 0x0f, 0xff, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0x0f, 0x07, 0xff, 0xf8, 0xf8, 0x03, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0x0e, 0x01, 0xff, 0xc0, 0x38, 0x07, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0x00, 0x1c, 0x46, 0xff, 0xb1, 0x18, 0x07, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xfe, 0x00, 0x3f, 0x97, 0xff, 0xc0, 0x7a, 0x07, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xfe, 0x00, 0x3f, 0xff, 0xff, 0xfe, 0x03, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xfe, 0x00, 0x3f, 0xff, 0xff, 0xfe, 0x03, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xfe, 0x01, 0x3f, 0xff, 0xff, 0xfe, 0x01, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xfe, 0x01, 0xb, 0xff, 0xff, 0xff, 0xfe, 0x81, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xfc, 0x00, 0xb, 0xff, 0xff, 0xff, 0xfc, 0x81, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xfc, 0x00, 0xff, 0xfe, 0xff, 0xfd, 0x83, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xf8, 0x00, 0xb, 0xff, 0xfe, 0xff, 0xfd, 0x01, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xf8, 0x00, 0x7f, 0xff, 0xff, 0xff, 0x01, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xf0, 0x00, 0x7f, 0xff, 0xff, 0xfb, 0x03, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xe0, 0x00, 0x3f, 0xff, 0xdc, 0xff, 0xfa, 0x03, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xd8, 0x00, 0x1f, 0xff, 0xff, 0xf8, 0x03, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xd0, 0x00, 0x1f, 0xff, 0xff, 0xf8, 0x01, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0x90, 0x00, 0x1f, 0xff, 0xff, 0xf8, 0x02, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xb0, 0x00, 0x0f, 0xf5, 0xff, 0xd7, 0xf8, 0x01, 0xff, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xb0, 0x00, 0x0f, 0xff, 0xff, 0xf8, 0x00, 0x5f, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0xa0, 0x00, 0x0f, 0xfb, 0xff, 0xff, 0xf0, 0x00, 0x3f, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0x80, 0x00, 0x0f, 0xfd, 0xff, 0xdf, 0xf0, 0x00, 0x3f, 0xff,
0xff, 0xff,
0xff, 0xff, 0xff, 0x80, 0x00, 0x07, 0xff, 0xff, 0xb, 0xf0, 0x00, 0x0f, 0xff,
0xff, 0xff,

```

        0xff, 0xff, 0xff, 0xff, 0x80, 0x00, 0x07, 0xff, 0xff, 0xe0, 0x00, 0x87, 0xff,
0xff, 0xff,
        0xff, 0xff, 0xff, 0x80, 0x00, 0x03, 0xff, 0xff, 0xc0, 0x00, 0x43, 0xff,
0xff, 0xff,
        0xff, 0xff, 0xff, 0x60, 0x00, 0x01, 0xff, 0xff, 0xc0, 0x00, 0x73, 0xff,
0xff, 0xff,
        0xff, 0xff, 0xfe, 0xe0, 0x00, 0x00, 0xff, 0xff, 0xff, 0x80, 0x00, 0x7b, 0xff,
0xff, 0xff,
        0xff, 0xff, 0xfd, 0xe0, 0x00, 0x00, 0x7f, 0xff, 0xfe, 0x00, 0x00, 0x33, 0xff,
0xff, 0xff,
        0xff, 0xff, 0xfd, 0xe0, 0x00, 0x00, 0x3f, 0xff, 0xf8, 0x00, 0x00, 0x27, 0xff,
0xff, 0xff,
        0xff, 0xff, 0xe0, 0x00, 0x00, 0x0f, 0xff, 0xf0, 0x00, 0x00, 0x0f, 0xff,
0xff, 0xff,
        0xff, 0xff, 0xff, 0x60, 0x00, 0x00, 0x67, 0xff, 0xe0, 0x00, 0x00, 0x1b, 0xff,
0xff, 0xff,
        0xff, 0xff, 0xfd, 0x40, 0x00, 0x00, 0xf3, 0xff, 0xc4, 0x00, 0x00, 0x0b, 0xff,
0xff, 0xff,
        0xff, 0xff, 0xfe, 0x80, 0x00, 0x00, 0xfc, 0xff, 0x8c, 0x00, 0x00, 0x03, 0xff,
0xff, 0xff,
        0xff, 0xff, 0x00, 0x00, 0x7f, 0x3c, 0x3c, 0x00, 0x00, 0x07, 0xff,
0xff, 0xff,
        0xff, 0xff, 0x00, 0x00, 0x3f, 0xc0, 0x7c, 0x00, 0x00, 0x03, 0xff,
0xff, 0xff,
        0xff, 0xff, 0x00, 0x00, 0x00, 0x1f, 0xff, 0xfc, 0x00, 0x00, 0x03, 0xff,
0xff, 0xff
};

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

    if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
        Serial.println(F("SSD1306 allocation failed"));
        for(;;);
    }
    delay(2000); // Pause for 2 seconds

    // Clear the buffer.
    display.clearDisplay();

    // Draw bitmap on the screen
    display.drawBitmap(0, 0, image_data_Saraarray, 128, 64, 1);
    display.display();
}

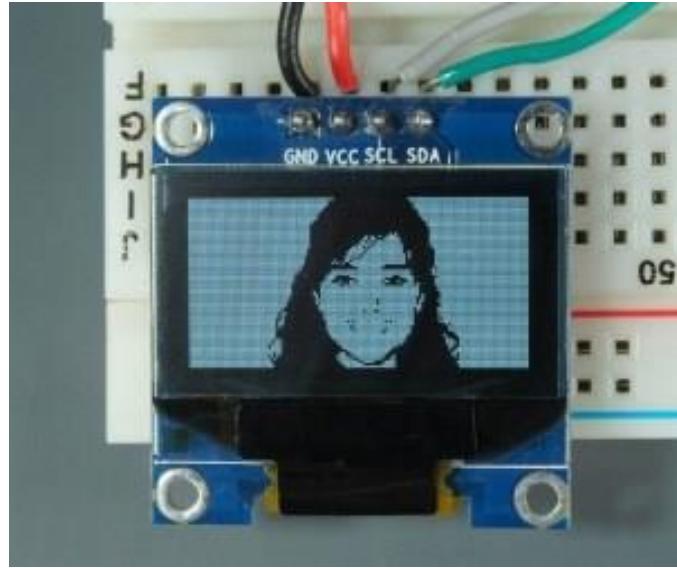
void loop() {
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module3/Unit4/OLED_Display_Bitmap_Images/OLED_Display_Bitmap_Images.ino

After uploading the code, this is what we get on the display.

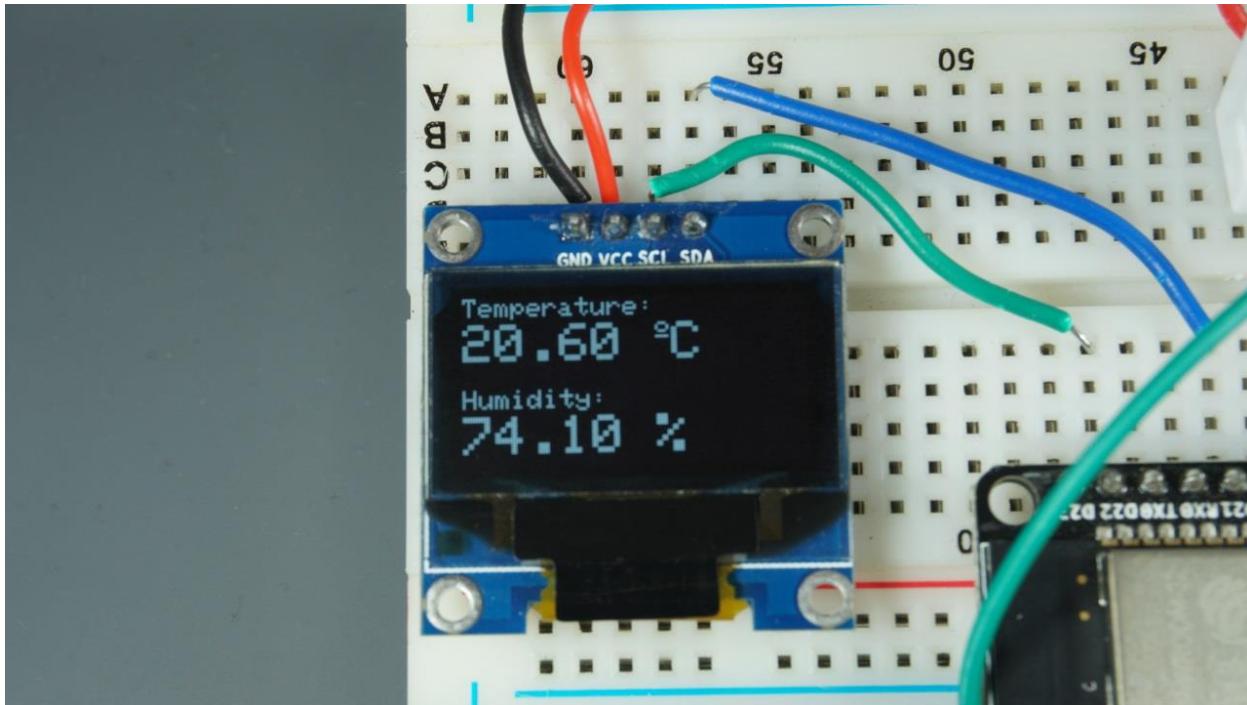


Troubleshooting

If you get the “SSD1306 allocation failed” error or if the OLED is not displaying anything in the screen, it can be one of the following issues:

- **Wrong I2C address:** the I2C address for the OLED display we are using is **0x3C**. However, yours may be different. So, make sure you check your display I2C address using an [I2C scanner sketch](#).
- **SDA and SCL not connected properly:** make sure that you have the SDA and SCL pins of the OLED display wired correctly. Connect the OLED SDA pin to GPIO 4 (D2) and the SCL pin to GPIO 5 (D1)

Unit 5: Display Sensor Readings on OLED Display



In this Unit, we'll make a simple project to display DHT sensor readings on an OLED display. You should already be familiar with the DHT11 and DHT22 temperature and humidity sensors and with the OLED display from previous Units. If you're not familiar with these modules, take a look at the previous Units.

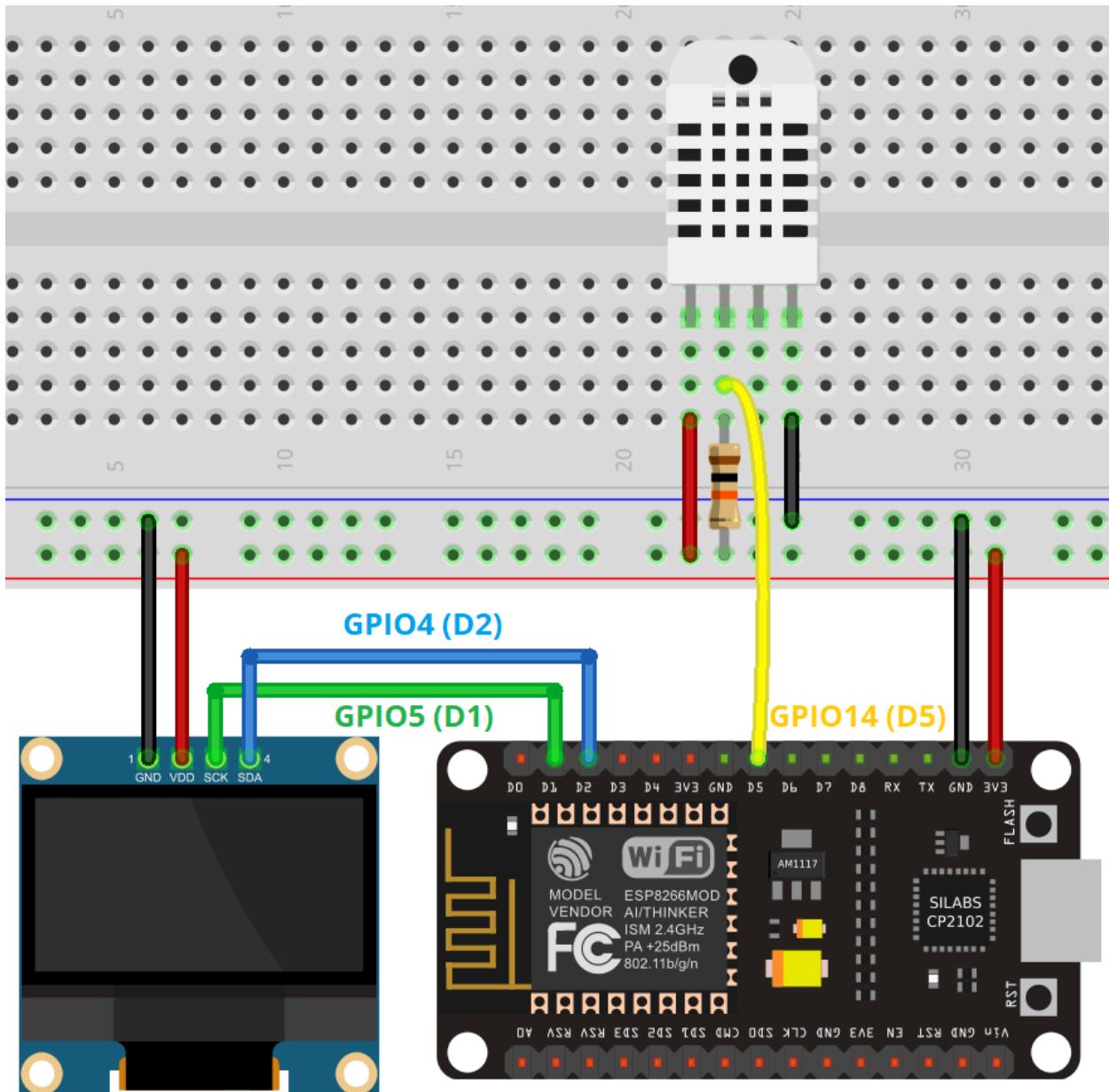
Parts required

For this tutorial you need the following components:

- [ESP8266](#)
- [0.96 inch OLED display](#)
- [DHT22](#) or [DHT11](#) temperature and humidity sensor
- [Breadboard](#)
- [4.7k Ohm resistor](#)
- [Jumper wires](#)

Schematic

Assemble the circuit as shown in the following schematic diagram. In this case, we're connecting the DHT data pin to GPIO 14, but you can use any other suitable GPIO.



Installing Libraries

Before uploading the code, you need to install the libraries to write to the OLED display and the libraries to read from the DHT sensor. If you've followed the previous Units, you should already have these libraries installed. If don't, follow the next instructions.

Open your Arduino IDE and go to **Sketch > Include Library > Manage Libraries**.

The **Library Manager** should open.

Search for the following libraries and install them:

- Adafruit SSD1306
- Adafruit GFX
- Adafruit DHT
- Adafruit Unified Sensor

Code

After installing the necessary libraries, you can copy the following code to your Arduino IDE and upload it to your ESP8266 board.

```
#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 14      // Digital pin connected to the DHT sensor

// Uncomment the type of sensor in use:
// #define DHTTYPE DHT11    // DHT 11
#define DHTTYPE DHT22    // DHT 22 (AM2302)
// #define DHTTYPE DHT21    // DHT 21 (AM2301)

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(5000);

  //read temperature and humidity
  float t = dht.readTemperature();
  float h = dht.readHumidity();
  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
  }
  // 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();
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module3/Unit5/Display_Sensor_Readings_OLED/Display_Sensor_Readings_OLED.ino

How the code works

Let's take a quick look on how the code works.

Importing libraries

The code starts by including the necessary libraries. The `Wire`, `Adafruit_GFX` and `Adafruit_SSD1306` are used to interface with the OLED display. The `Adafruit_Sensor` and the `DHT` libraries are needed to interface with the DHT22 or DHT11 sensors.

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

Create a display object

Then, define your OLED display dimensions. In this case, we're using a 128×64 pixel display.

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

Then, initialize a `display` object with the width and height defined earlier with I2C communication protocol (`&Wire`).

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

The (-1) parameter means that your OLED display doesn't have a RESET pin. If your OLED display does have a RESET pin, it should be connected to a GPIO. In that case, you should pass the GPIO number as a parameter.

Create a DHT object

Then, define the DHT sensor type you're using. If you're using a DHT22 you don't need to change anything on the code. If you're using another sensor, just uncomment the sensor you're using and comment the others.

```
//#define DHTTYPE      DHT11      // DHT 11
#define DHTTYPE      DHT22      // DHT 22 (AM2302)
//#define DHTTYPE      DHT21      // DHT 21 (AM2301)
```

Initialize a DHT sensor object with the pin and type defined earlier.

```
DHT dht(DHTPIN, DHTTYPE);
```

setup()

In the `setup()`, initialize the serial monitor for debugging purposes.

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

Initialize the DHT sensor:

```
dht.begin();
```

Then, initialize the OLED display.

```
if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
  Serial.println(F("SSD1306 allocation failed"));
  for(;;);
}
```

In this case, the address of the OLED display we're using is **0x3C**. If this address doesn't work, you can run an I2C scanner sketch to find your OLED address. You can [find the I2C scanner sketch here](#).

Add a delay to give time for the display to initialize, clear the display and set the text color to white:

```
delay(2000);
```

```
display.clearDisplay();
display.setTextColor(WHITE);
```

In the `loop()` is where we read the sensor and display the temperature and humidity on the display.

Get temperature and humidity readings from DHT

The temperature and humidity are saved on the `t` and `h` variables, respectively. Reading temperature and humidity is as simple as using the `readTemperature()` and `readHumidity()` methods on the `dht` object.

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

In case we're not able to get the readings, display an error message:

```
if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
}
```

Display sensor readings on the OLED display

The following lines display the temperature on the OLED display.

```
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");
```

We use the `setTextSize()` method to define the font size, the `setCursor()` sets where the text should start being displayed and the `print()` method is used to write something on the display.

To print the temperature and humidity you just need to pass their variables to the `print()` method as follows:

```
display.print(t);
```

The “Temperature” label is displayed in size 1, and the actual reading is displayed in size 2.

To display the ° symbol, we use the [Code Page 437](#) font. For that, you need to set the `cp437` to true as follows:

```
display.cp437(true);
```

Then, use the `write()` method to display your chosen character. The ° symbol corresponds to character 167.

```
display.write(167);
```

A similar approach is used to display the humidity:

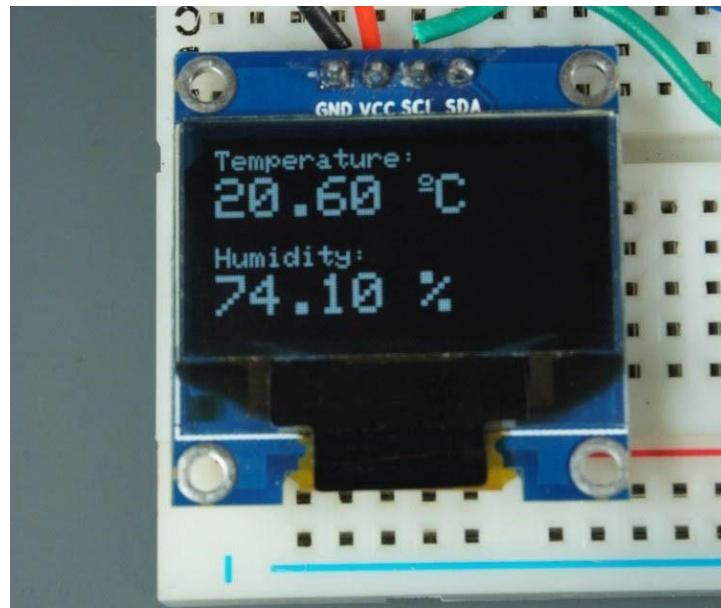
```
display.setTextSize(1);
display.setCursor(0, 35);
display.print("Humidity: ");
display.setTextSize(2);
display.setCursor(0, 45);
display.print(h);
display.print("%");
```

Don't forget that you need to call `display.display()` at the end, so that you can actually display something on the OLED.

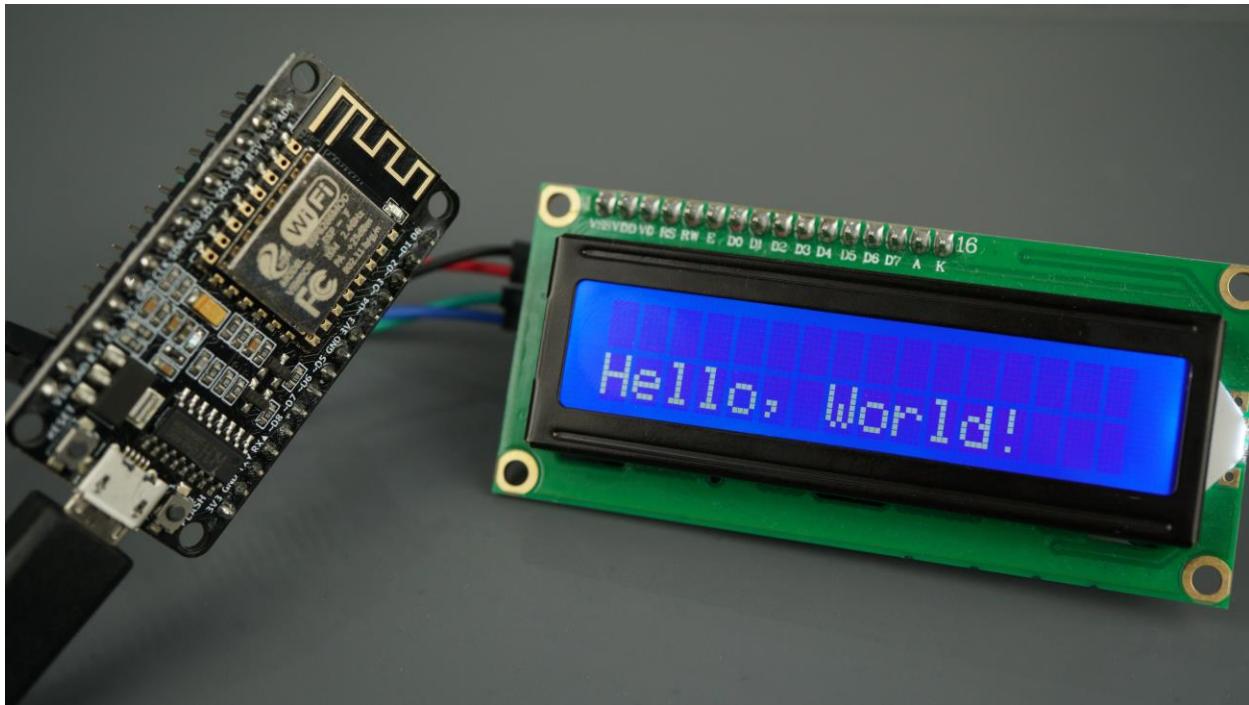
```
display.display();
```

Demonstration

The following figure shows what you should after completing this tutorial. Humidity and temperature readings are displayed on the OLED display.



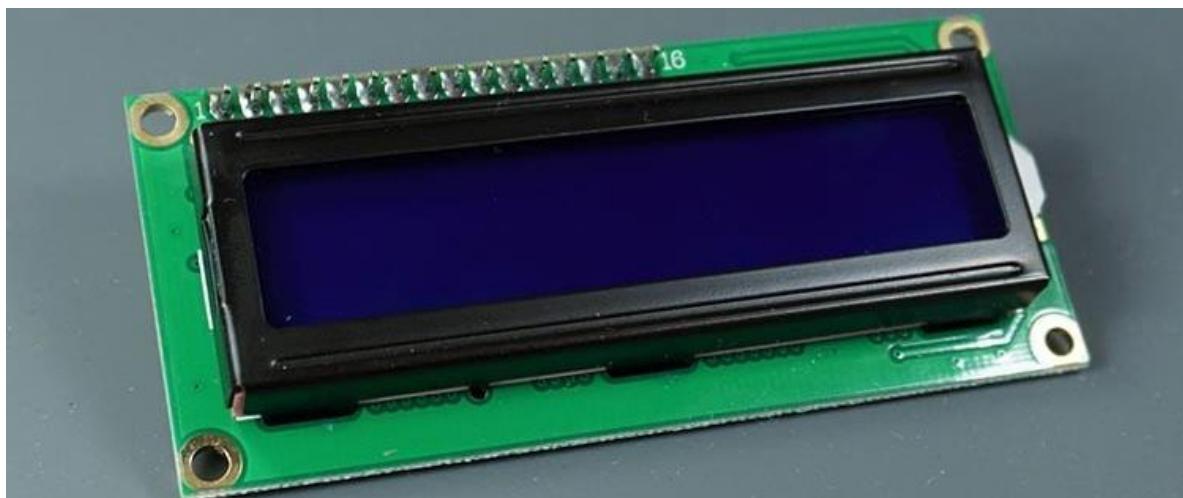
Unit 6: I2C Liquid Crystal Display (LCD)



This Unit shows how to use the I2C Liquid Crystal Display (LCD) with the ESP8266. Learn how to write on the LCD: static text, scroll long messages and display icons.

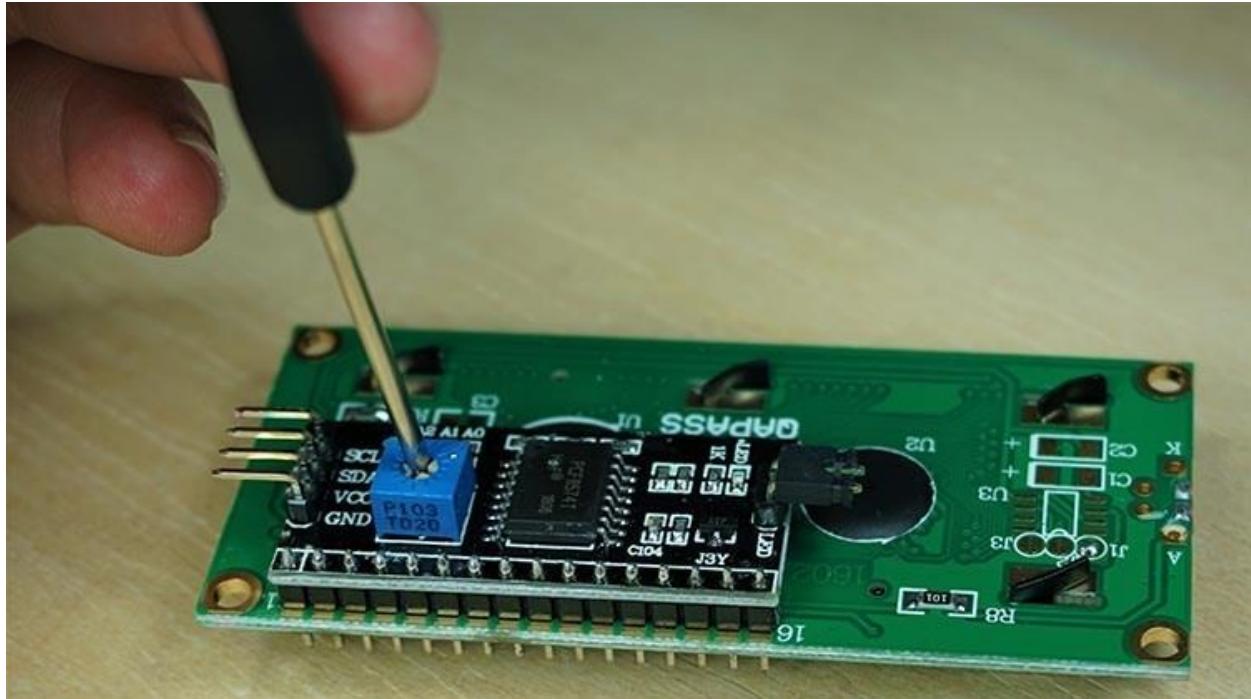
16×2 I2C Liquid Crystal Display

We'll use a 16×2 I2C LCD display, but LCDs with other sizes should also work.



The advantage of using an I2C LCD is that the wiring is really simple. You just need to wire the SDA and SCL pins.

Additionally, it comes with a built-in potentiometer you can use to adjust the contrast between the background and the characters on the LCD. On a “regular” LCD you need to add a potentiometer to the circuit to adjust the contrast.



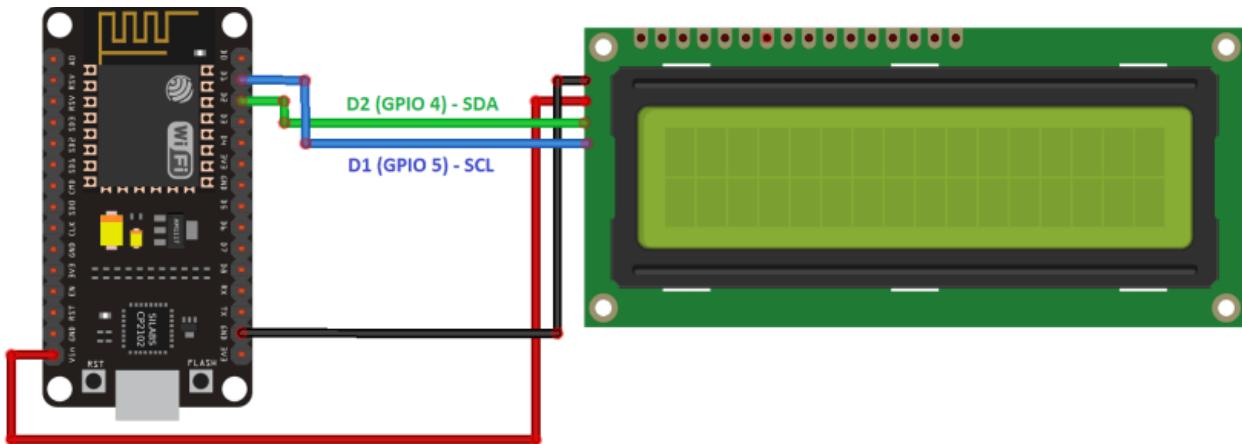
Parts Required

To follow this tutorial, you need these parts:

- [ESP8266](#)
- [16x2 I2C Liquid Crystal Display \(LCD\)](#)
- [Female to female jumper wires](#)

Schematic diagram

This display uses I2C communication, which makes wiring really simple. Wire your LCD to the ESP8266 by following the next schematic diagram. We're using the ESP8266 default I2C pins (GPIO 4 and GPIO 5).



You can also use the following table as a reference.

I2C LCD	ESP8266
GND	GND
VCC	VIN
SDA	GPIO 4 (D2)
SCL	GPIO 5 (D1)

Installing the LiquidCrystal_I2C Library

There are several libraries that work with the I2C LCD. We're using [this library by Marco Schwartz](#). Follow the next steps to install the library:

1. [Click here to download the LiquidCrystal_I2C library](#). You should have a *.zip* folder in your *Downloads*.
2. Unzip the *.zip* folder and you should get *LiquidCrystal_I2C-master* folder.
3. Rename your folder from *LiquidCrystal_I2C-master* to *LiquidCrystal_I2C*.
4. Move the *LiquidCrystal_I2C* folder to your Arduino IDE
5. installation libraries folder.
6. Finally, re-open your Arduino IDE

Alternatively, you can go **Sketch** ▶ **Include Library** ▶ **Add .ZIP library** and select the library you've just downloaded.

Getting the LCD Address

Before displaying text on the LCD, you need to find the LCD I2C address. With the LCD properly wired to the ESP8266, upload the following I2C Scanner sketch.

```
#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) {
```

```

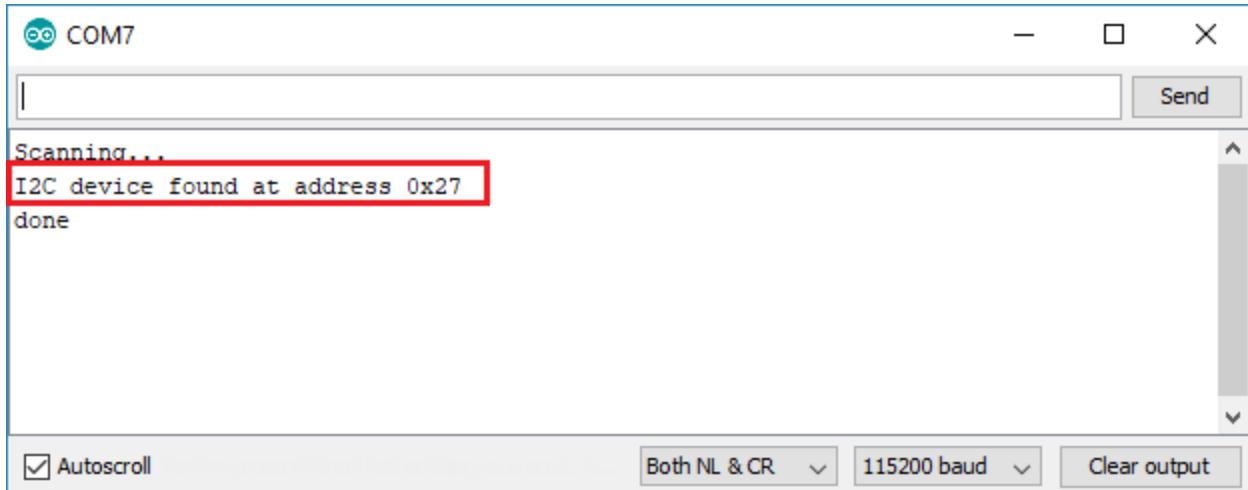
        Serial.print("0");
    }
    Serial.println(address,HEX);
    nDevices++;
}
else if (error==4) {
    Serial.print("Unknow error at address 0x");
    if (address<16) {
        Serial.print("0");
    }
    Serial.println(address,HEX);
}
if (nDevices == 0) {
    Serial.println("No I2C devices found\n");
}
else {
    Serial.println("done\n");
}
delay(5000);
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module3/Unit6/I2C_Scanner/I2C_Scanner.ino

After uploading the code, open the Serial Monitor at a baud rate of 115200. Press the ESP8266 RST button. The I2C address should be displayed in the Serial Monitor.



In this case the address is **0x27**. If you're using a similar 16×2 display, you'll probably get the same address.

Display Static Text on the LCD

Displaying static text on the LCD is very simple. All you have to do is select where you want the characters to be displayed on the screen, and then send the message to the display.

Here's a very simple sketch example that displays "Hello, World!".

```
#include <LiquidCrystal_I2C.h>

// set the LCD number of columns and rows
int lcdColumns = 16;
int lcdRows = 2;

// set LCD address, number of columns and rows
// if you don't know your display address, run an I2C scanner sketch
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);

void setup() {
    // initialize LCD
    lcd.init();
    // turn on LCD backlight
    lcd.backlight();
}

void loop(){
    // set cursor to first column, first row
    lcd.setCursor(0, 0);
    // print message
    lcd.print("Hello, World!");
    delay(1000);
    // clears the display to print new message
    lcd.clear();
    // set cursor to first column, second row
    lcd.setCursor(0,1);
    lcd.print("Hello, World!");
    delay(1000);
    lcd.clear();
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module3/Unit6/LCD_Hello_World/LCD_Hello_World.ino

The LCD displays the message in the first row for one second, and then, in the second row for another second.



In this simple sketch we show you the most useful and important functions from the LiquidCrystal_I2C library. So, let's take a quick look at how the code works.

How the code works

First, you need to include the `LiquidCrystal_I2C` library.

```
#include <LiquidCrystal_I2C.h>
```

The next two lines set the number of columns and rows of your LCD display. If you're using a display with another size, you should modify those variables.

```
int lcdColumns = 16;  
int lcdRows = 2;
```

Then, you need to set the display address, the number of columns and number of rows. You should use the display address you've found in the previous step.

```
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);
```

In the `setup()`, first initialize the display with the `init()` method.

```
lcd.init();
```

Then, turn on the LCD backlight, so that you're able to read the characters on the display.

```
lcd.backlight();
```

To display a message on the screen, first you need to set the cursor to where you want your message to be written. The following line sets the cursor to the first column, first row.

```
lcd.setCursor(0, 0);
```

Note: 0 corresponds to the first column, 1 to the second column, and so on...

Then, you can finally print your message on the display using the `print()` method.

```
lcd.print("Hello, World!");
```

Wait one second, and then clean the display with the `clear()` method.

```
lcd.clear();
```

After that, set the cursor to a new position: first column, second row.

```
lcd.setCursor(0,1);
```

Then, the process is repeated.

Here's a summary of the functions to manipulate and write on the display:

- `lcd.init()`: initializes the display;
- `lcd.backlight()`: turns the LCD backlight on;
- `lcd.setCursor(int column, int row)`: sets the cursor to the specified column and row;

- `lcd.print(String message)`: displays the message on the display;
- `lcd.clear()`: clears the display;

This example works well to display static text no longer than 16 characters.

Display Scrolling Text on the LCD

Scrolling text on the LCD is especially useful when you want to display messages longer than 16 characters. The library comes with built-in functions that allows you to scroll text. However, many people experience problems with those functions because:

- The function scrolls text on both rows. So, you can't have a fixed row and a scrolling row;
- It doesn't work properly if you try to display messages longer than 16 characters.

So, we've created a sample sketch with a function you can use in your projects to scroll longer messages.

The following sketch displays a static message in the first row and a scrolling message longer than 16 characters in the second row.

```
#include <LiquidCrystal_I2C.h>

// set the LCD number of columns and rows
int lcdColumns = 16;
int lcdRows = 2;

// set LCD address, number of columns and rows
// if you don't know your display address, run an I2C scanner sketch
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);

String messageStatic = "Static message";
String messageToScroll = "This is a scrolling message with more than 16
characters";

// Function to scroll text
// The function accepts the following arguments:
// row: row number where the text will be displayed
// message: message to scroll
// delayTime: delay between each character shifting
```

```

// lcdColumns: number of columns of your LCD
void scrollText(int row, String message, int delayTime, int lcdColumns) {
    for (int i=0; i < lcdColumns; i++) {
        message = " " + message;
    }
    message = message + " ";
    for (int pos = 0; pos < message.length(); pos++) {
        lcd.setCursor(0, row);
        lcd.print(message.substring(pos, pos + lcdColumns));
        delay(delayTime);
    }
}

void setup(){
    // initialize LCD
    lcd.init();
    // turn on LCD backlight
    lcd.backlight();
}

void loop(){
    // set cursor to first column, first row
    lcd.setCursor(0, 0);
    // print static message
    lcd.print(messageStatic);
    // print scrolling message
    scrollText(1, messageToScroll, 250, lcdColumns);
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module3/Unit6/LCD_Scroll_Text/LCD_Scroll_Text.ino

After reading the previous section, you should be familiar on how this sketch works, so we'll just take a look at the newly created function: `scrollText()`.

```

void scrollText(int row, String message, int delayTime, int lcdColumns) {
    for (int i=0; i < lcdColumns; i++) {
        message = " " + message;
    }
    message = message + " ";
    for (int pos = 0; pos < message.length(); pos++) {
        lcd.setCursor(0, row);
        lcd.print(message.substring(pos, pos + lcdColumns));
        delay(delayTime);
    }
}

```

To use this function, you should pass four arguments:

- `row`: row number where the text will be displayed;
- `message`: message to scroll;
- `delayTime`: delay between each character shifting. Higher delay times will result in slower text shifting, and lower delay times will result in faster text shifting;
- `lcdColumns`: number of columns of your LCD.

In our code, here's how we use the `scrollText()` function:

```
scrollText(1, messageToScroll, 250, lcdColumns);
```

The `messageToScroll` variable is displayed in the second row (1 corresponds to the second row), with a delay time of 250 milliseconds.



Display Custom Characters

In a 16×2 LCD there are 32 blocks where you can display characters. Each block is made out of 5×8 tiny pixels. You can display custom characters by defining the state of each tiny pixel. For that, you can create a byte variable to hold the state of each pixel.

To create your custom character, you can go [here](#) to generate the byte variable for your character. For example, a heart:

Pixels					Output	
	<pre>byte customChar[8] = { 0b00000, 0b01010, 0b11111, 0b11111, 0b11111, 0b01110, 0b00100, 0b00000 };</pre>					

Pixels

Output

Clear **Invert**

Copy the byte variable to your code (before the `setup()`). You can call it `heart`:

```
byte heart[8] = {  
    0b00000,  
    0b01010,  
    0b11111,  
    0b11111,  
    0b11111,  
    0b01110,  
    0b00100,  
    0b00000  
};
```

Then, in the `setup()`, create a custom character using the `createChar()` function. This function accepts as arguments a location to allocate the char and the char variable as follows:

```
lcd.createChar(0, heart);
```

Then, in the `loop()`, set the cursor to where you want the character to be displayed:

```
lcd.setCursor(0, 0);
```

Use the `write()` method to display the character. Pass the location where the character is allocated, as follows:

```
lcd.write(0);
```

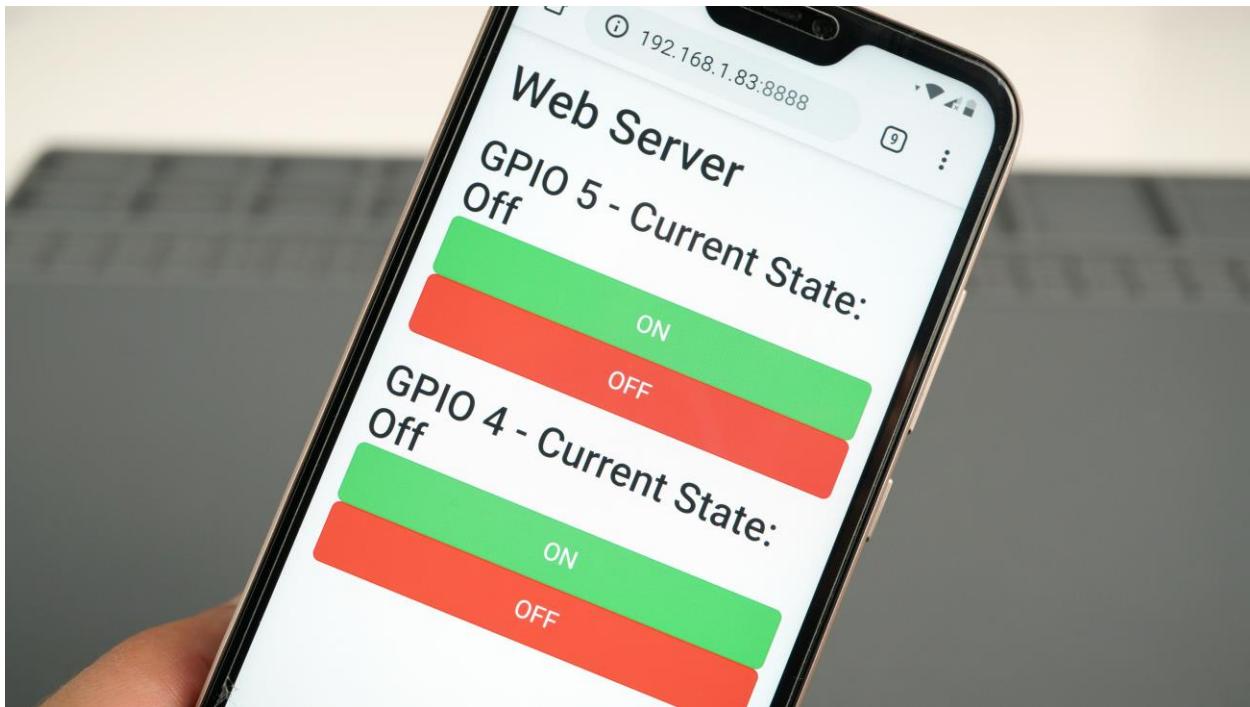
Now that you know how to use the LCD, you can try displaying sensor readings or any other info.

MODULE 4:

Web Servers

In this Module, you'll learn how to build web servers with the ESP8266 to control outputs and display sensor readings remotely. We'll show you different methods to build web servers using different libraries and different sensors, so that you can modify the examples presented and apply them in your own projects.

Unit 1: Web Server Introduction



In this Unit you'll learn what a web server is and we'll look at some terms that you've probably heard before, but you may not know exactly what they mean.

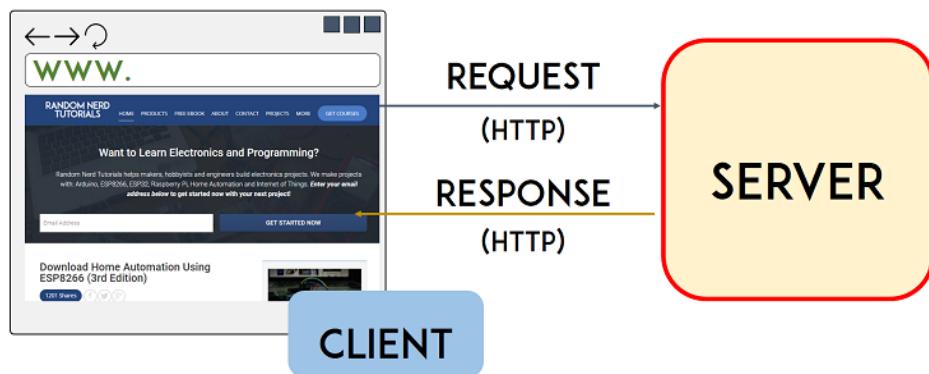
Request-Response

Request-response is a message exchange pattern, in which a requestor sends a request message to a replier system that receives and processes the request and returns a message in response. This is a simple, yet powerful messaging pattern especially in client-server architectures.



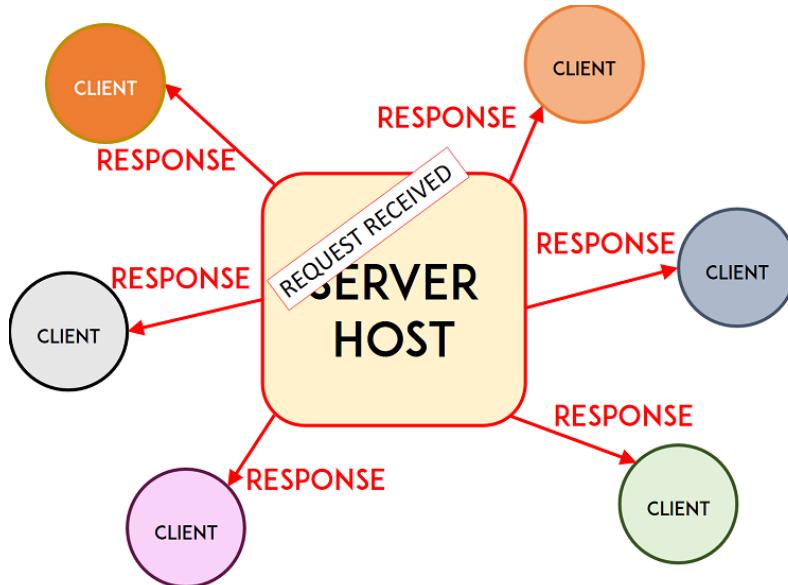
Client-Server

When you type an URL in your browser, what happens in the background is that you (the client) send a request via Hypertext Transfer Protocol (HTTP) to a server. When the server receives the request, it sends a response also through HTTP, and you see the web page you requested in your browser. Clients and servers communicate over a computer network.

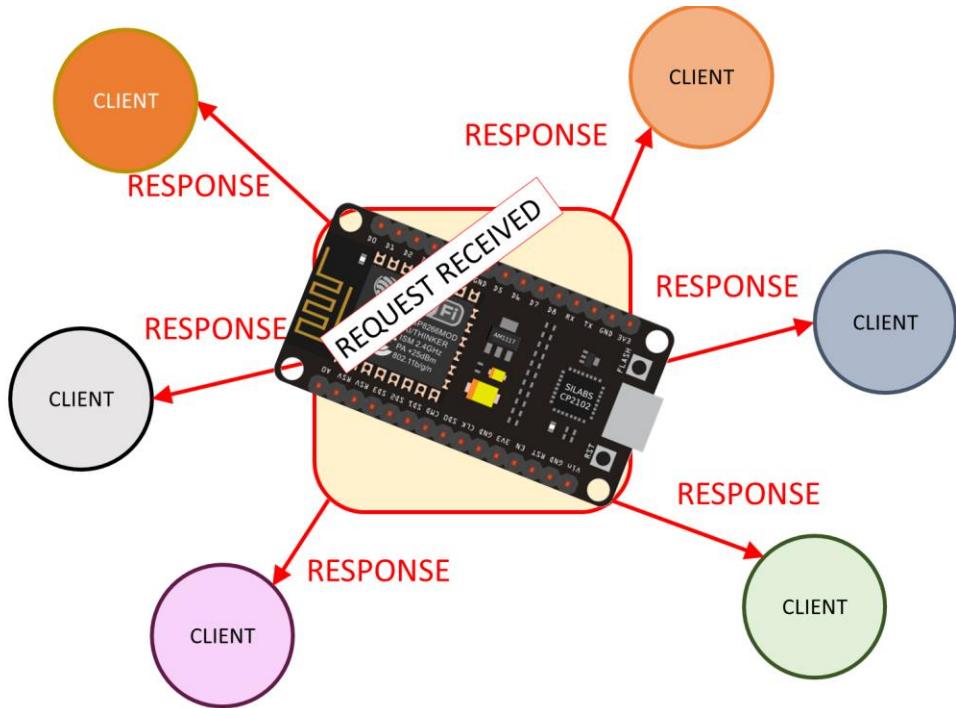


Server Host

A server host runs one or more server programs to share their resources with clients. So, you can imagine a web server as a piece of software that listens for incoming HTTP requests and sends responses when requested.

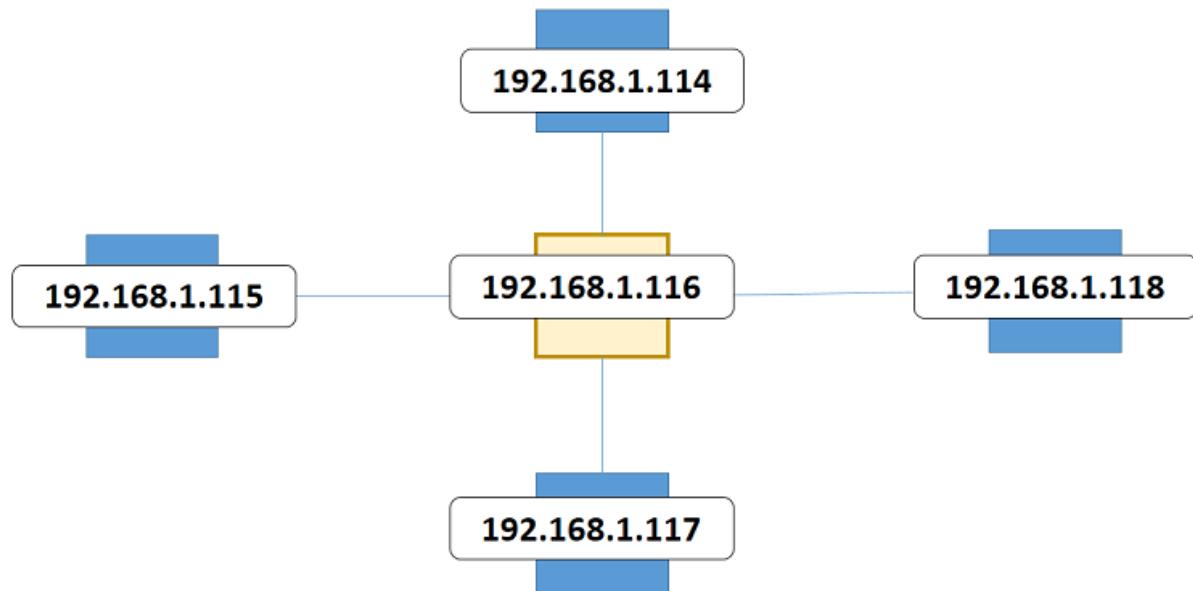


Your ESP8266 can act as a server host, listening for HTTP requests from clients. When a new client makes a request, the ESP sends an HTTP response.



IP Address

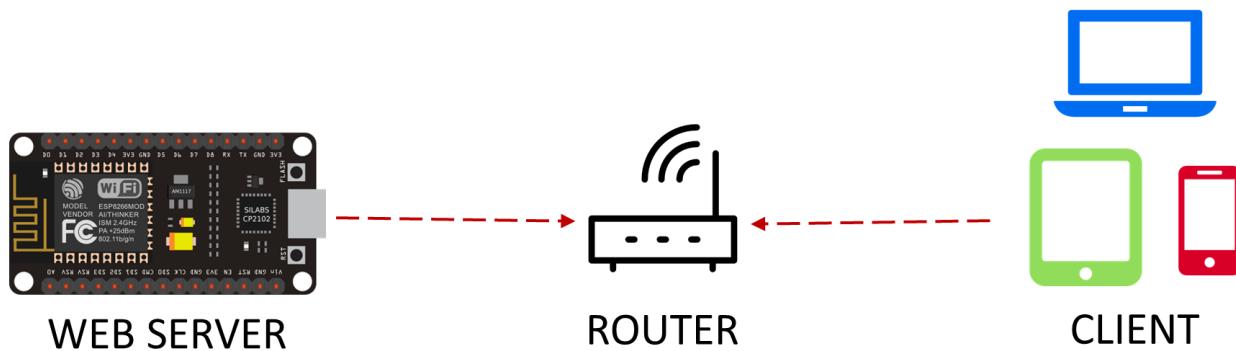
An IP address is a numerical label assigned to each device connected to a computer network. This way, any information sent to that device can reach it by referring to its IP address. So, your ESP8266 has an IP address too.



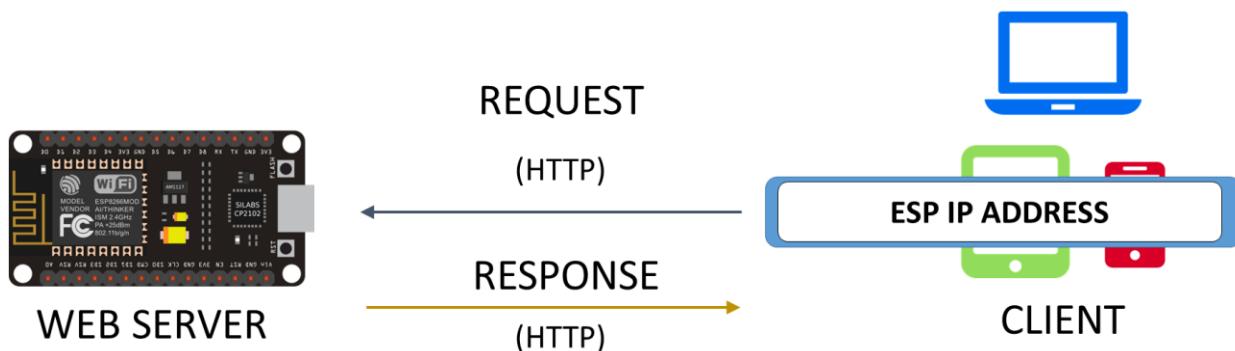
ESP8266 Web Server

Let's look at a practical example with the ESP8266 that can act as a web server in the local network.

Typically, a web server with the ESP8266 in the local network looks something like this: the ESP board running as a web server is connected via Wi-Fi to your router. Your computer, smartphone, or tablet are also connected to your router via Wi-Fi or Ethernet cable. So, the ESP and your browser are on the same network.



When you type the ESP IP address in your browser, you are sending an HTTP request to your ESP8266. Then, the ESP responds back with a response that can contain a value, a reading, HTML text to display a web page, or any data you program in your ESP8266.

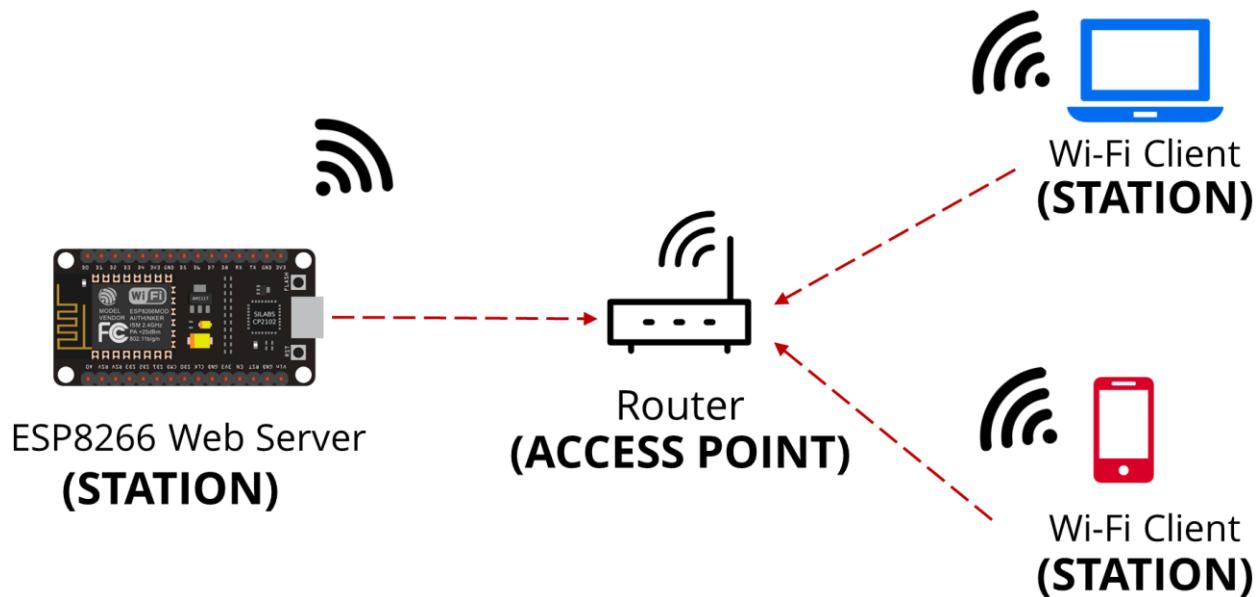


Access Point vs Station

The ESP8266 can act as a Wi-Fi station, as an access point, or both. In the previous example, the ESP8266 is set as a Wi-Fi station. Let's look at the differences between station and access point.

Station

When the ESP8266 is set as a station, we can access it through the local network. In this scenario, the router acts as an access point and the ESP8266 is set as a station. So, you need to be connected to your router (local network) to control the ESP8266.

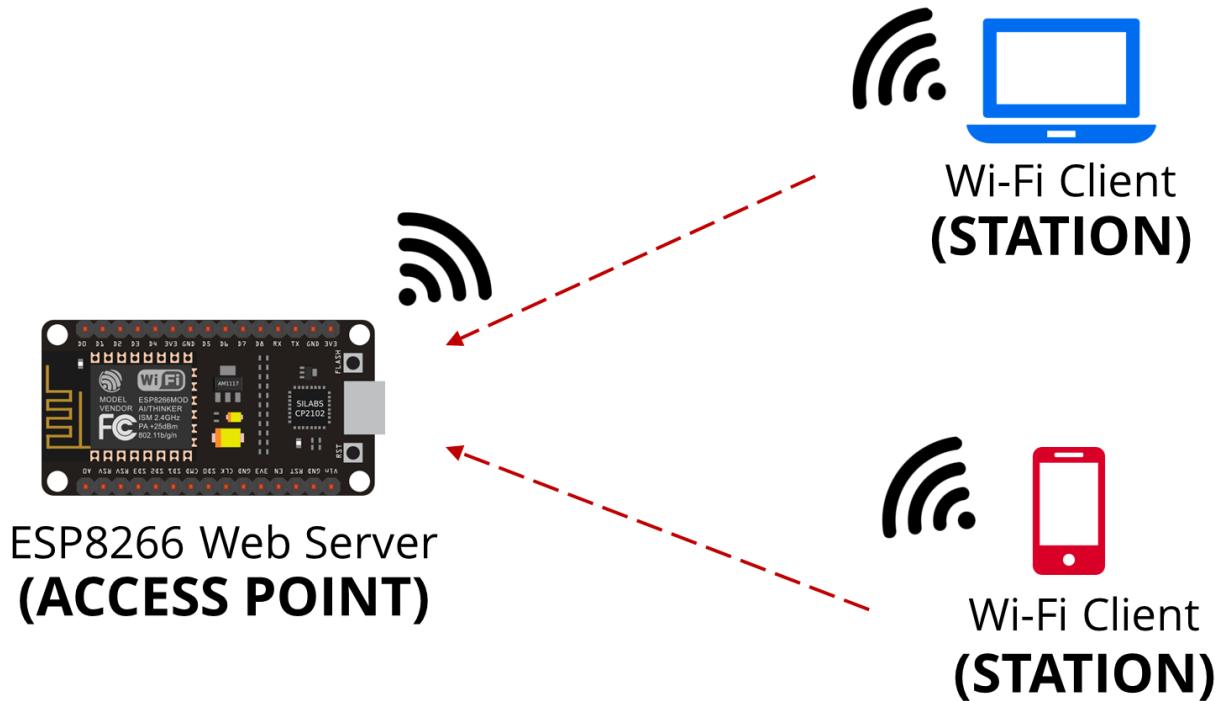


Access Point

If you set the ESP8266 as an access point (hotspot), you can be connected to the ESP8266 using any device with Wi-Fi capabilities without the need to connect to your router.

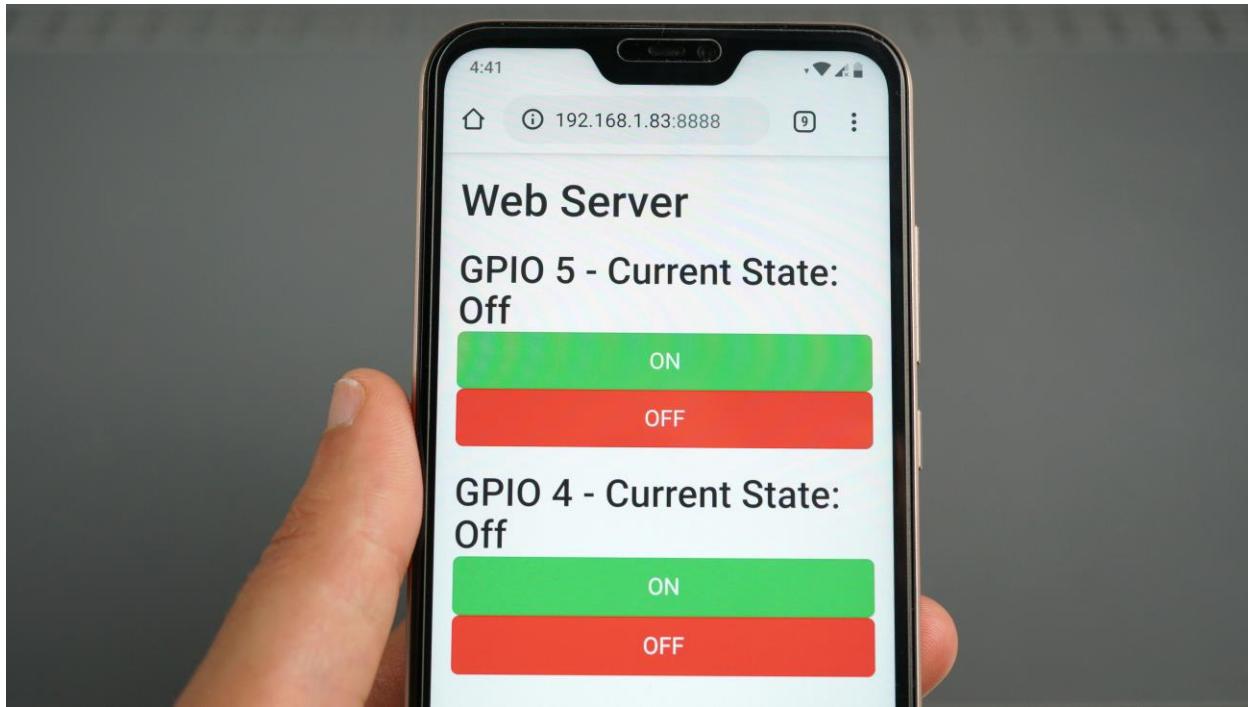
In simple words, when you set the ESP8266 as an access point you create its own Wi-Fi network and nearby Wi-Fi devices (stations) can connect to it (like your

smartphone or your computer). This way, you don't need to be connected to a router to control your ESP8266.



Because the ESP8266 doesn't connect further to a wired network (like your router), it is called soft-AP (soft Access Point).

Unit 2: Password Protected Web Server



In this Unit you're going to create a password protected web server with your ESP8266 that can be accessed with any device with a browser. This means you can control the ESP8266 GPIOs from your laptop, smartphone, tablet and so on.

As an example, you're going to control two LEDs. The idea is to replace those LEDs with any other output like a relay to control any electronic devices that you want.

This web server will be password protected and accessible from anywhere in the world.

Code

The following code creates a web server that controls two outputs (GPIO 5 and GPIO 4) on your local network and it is password protected. Later, you'll learn how to make your web server accessible from anywhere.

```
// Including the ESP8266 WiFi library
#include <ESP8266WiFi.h>

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

// Web Server on port 8888
WiFiServer server(8888);

// variables
String header;
String gpio5_state = "Off";
String gpio4_state = "Off";
int gpio5_pin = 5;
int gpio4_pin = 4;

// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in milliseconds (example: 2000ms = 2s)
const long timeoutTime = 2000;

// only runs once
void setup() {
    // Initializing serial port for debugging purposes
    Serial.begin(115200);
    delay(10);

    // preparing GPIOs
    pinMode(gpio5_pin, OUTPUT);
    digitalWrite(gpio5_pin, LOW);
    pinMode(gpio4_pin, OUTPUT);
    digitalWrite(gpio4_pin, LOW);

    // Connecting to WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
```

```

        Serial.print(".");
    }
    Serial.println("");
    Serial.println("WiFi connected");

    // Starting the web server
    server.begin();
    Serial.println("Web server running. Waiting for the ESP IP..."); 
    delay(5000);

    // Printing the ESP IP address
    Serial.print("Go to: http://");
    Serial.print(WiFi.localIP());
    Serial.println(":8888");
}

// runs over and over again
void loop() {
    // Listenning for new clients
    WiFiClient client = server.available();

    if (client) {
        currentTime = millis();
        previousTime = currentTime;
        Serial.println("New client");
        // boolean to locate when the http request ends
        boolean blank_line = true;
        while (client.connected() && currentTime - previousTime <= timeoutTime) {
            currentTime = millis();
            if (client.available()) {
                char c = client.read();
                header += c;

                if (c == '\n' && blank_line) {

                    // checking if header is valid
                    // dXNlcjpwYXNz = 'user:pass' (user:pass) base64 encode

                    Serial.print(header);

                    // Finding the right credential string
                    if(header.indexOf("dXNlcjpwYXNz") >= 0) {
                        //successful login
                        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 / HTTP/1.1") >= 0) {
                            Serial.println("Main Web Page");
                        }
                        else if(header.indexOf("GET /gpio5on HTTP/1.1") >= 0) {
                            Serial.println("GPIO 5 On");
                            gpio5_state = "On";
                        }
                    }
                }
            }
        }
    }
}

```

```

        digitalWrite(gpio5_pin, HIGH);
    }
    else if(header.indexOf("GET /gpio5off HTTP/1.1") >= 0) {
        Serial.println("GPIO 5 Off");
        gpio5_state = "Off";
        digitalWrite(gpio5_pin, LOW);
    }
    else if(header.indexOf("GET /gpio4on HTTP/1.1") >= 0) {
        Serial.println("GPIO 4 On");
        gpio4_state = "On";
        digitalWrite(gpio4_pin, HIGH);
    }
    else if(header.indexOf("GET /gpio4off HTTP/1.1") >= 0) {
        Serial.println("GPIO 4 Off");
        gpio4_state = "Off";
        digitalWrite(gpio4_pin, LOW);
    }
    // your web page
    client.println("<!DOCTYPE HTML>");
    client.println("<html>");
    client.println("<head>");
    client.println("<meta name=\"viewport\""
content="width=device-width, initial-scale=1\"");
        client.println("<link rel=\"stylesheet\""
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/css/bootstrap.min.css">");

        client.println("</head><div class=\"container\"");
        client.println("<h1>Web Server</h1>");
        client.println("<h2>GPIO 5 - Current State: " +
gpio5_state);
        client.println("<div class=\"row\"");
        client.println("<div class=\"col-md-2"><a"
href="/gpio5on" class="btn btn-block btn-lg btn-success\""
role="button">ON</a></div>");
        client.println("<div class=\"col-md-2"><a"
href="/gpio5off" class="btn btn-block btn-lg btn-danger\""
role="button">OFF</a></div>");

        client.println("</div>");
        client.println("<h2>GPIO 4 - Current State: " +
gpio4_state);
        client.println("<div class=\"row\"");
        client.println("<div class=\"col-md-2"><a"
href="/gpio4on" class="btn btn-block btn-lg btn-success\""
role="button">ON</a></div>");

        client.println("<div class=\"col-md-2"><a"
href="/gpio4off" class="btn btn-block btn-lg btn-danger\""
role="button">OFF</a></div>");

        client.println("</div></div></html>");
    }
    // wrong user or passm, so http request fails...
    else {
        client.println("HTTP/1.1 401 Unauthorized");
    }
}

```

```

        client.println("WWW-Authenticate: Basic
realm=\"Secure\"");
        client.println("Content-Type: text/html");
        client.println();
        client.println("<html>Authentication failed</html>");
    }
    header = "";
    break;
}
if (c == '\n') {
    // when starts reading a new line
    blank_line = true;
}
else if (c != '\r') {
    // when finds a character on the current line
    blank_line = false;
}
}
// closing the client connection
delay(1);
client.stop();
Serial.println("Client disconnected.");
}
}

```

This sketch is quite long. We recommend that you follow the code through the following link. It will be more readable.

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module4/Unit2/Password_Protected_Web_Server/Password_Protected_Web_Server.ino

How the Code Works

The next snippet of code starts by including the ESP8266WiFi library. Then, you configure your ESP8266 with your own network credentials (*ssid* and *password*).

Note: you need to replace those two lines with your credentials, so that your ESP can connect to your network.

```
// Including the ESP8266 WiFi library
#include <ESP8266WiFi.h>

// Replace with your network details
const char* ssid = "YOUR_NETWORK_NAME";
const char* password = "YOUR_NETWORK_PASSWORD";
```

The next thing to do is declaring your web server on port 8888:

```
// Web Server on port 8888
WiFiServer server(8888);
```

Create one variable `header` to store the header response of the request; two variables `gpio5_state` and `gpio4_state` to store the GPIOs current state; two variables `gpio5_pin` and `gpio4_pin` that refer to GPIO 5 and GPIO 4, respectively.

```
String header;
String gpio5_state = "Off";
String gpio4_state = "Off";
int gpio5_pin = 5;
int gpio4_pin = 4;
```

You also create some auxiliary timer variables that will allow us to close the connection with the client after 2 seconds.

```
// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in milliseconds (example: 2000ms = 2s)
const long timeoutTime = 2000;
```

Note: you need these timer variables because some browsers make an additional request without closing the connection. When a connection is left open, you're not

able to receive responses to other requests. So, we add a timer variable that closes all the connections after 2 seconds.

setup()

In the `setup()`, start a serial communication at a `115200` baud rate for debugging purposes. Define your GPIOs as OUTPUTs and set them `LOW`.

```
Serial.begin(115200);
delay(10);

// preparing GPIOs
pinMode(gpio5_pin, OUTPUT);
digitalWrite(gpio5_pin, LOW);
pinMode(gpio4_pin, OUTPUT);
digitalWrite(gpio4_pin, LOW);
```

The following code snippet: starts the Wi-Fi connection, waits for a sucessful connection and prints the ESP8266 IP address in the Serial Monitor.

```
// Connecting to WiFi network
Serial.println();
Serial.print("Connecting to ");
Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}

Serial.println("");
Serial.println("WiFi connected");

// Starting the web server
server.begin();
Serial.println("Web server running. Waiting for the ESP IP...");
delay(5000);

// Printing the ESP IP address
Serial.print("Go to: http://");
Serial.print(WiFi.localIP());
Serial.println(":8888");
```

loop()

The `loop()` function programs what happens when a new client establishes a connection with the web server. The code is always listening for new clients. When a new client connects, it starts a connection.

```
void loop() {
    // Listenning for new clients
    WiFiClient client = server.available();
```

There's a boolean variable `blank_line` to act as a control to help determine when the HTTP request ends. You also have a `while()` loop that will be running for as long as the client stays connected. Here, we also keep track of time so that we're able to close the connection after 2 seconds.

```
if (client) {
    currentTime = millis();
    previousTime = currentTime;
    Serial.println("New client");
    // boolean to locate when the http request ends
    boolean blank_line = true;
    while (client.connected() && currentTime - previousTime <= timeoutTime) {
        currentTime = millis();
        if (client.available()) {
```

To make your web server a bit more secure there's an authentication mechanism. After implementing this feature, when someone tries to access your web server, they need to enter a username and a password.

By default, your username is `user` and your password is `pass`. We'll show you how to change that in just a moment. If the user enters the correct username and password the web server shows the web page that controls the ESP8266.

```
// Finding the right credential string
if(header.indexOf("dxNlcjpwYXNz") >= 0) {
```

This next snippet of code checks which button in your web page was pressed. Basically, it checks the URL that you have just clicked.

```

else if(header.indexOf("GET /gpio5on HTTP/1.1") >= 0) {
    Serial.println("GPIO 5 On");
    gpio5_state = "On";
    digitalWrite(gpio5_pin, HIGH);
}
else if(header.indexOf("GET /gpio5off HTTP/1.1") >= 0) {
    Serial.println("GPIO 5 Off");
    gpio5_state = "Off";
    digitalWrite(gpio5_pin, LOW);
}
else if(header.indexOf("GET /gpio4on HTTP/1.1") >= 0) {
    Serial.println("GPIO 4 On");
    gpio4_state = "On";
    digitalWrite(gpio4_pin, HIGH);
}
else if(header.indexOf("GET /gpio4off HTTP/1.1") >= 0) {
    Serial.println("GPIO 4 Off");
    gpio4_state = "Off";
    digitalWrite(gpio4_pin, LOW);
}

```

Let's see a practical example. When you click the **GPIO 5 OFF** button you open this URL: <http://192.168.1.70:8888/gpio5off>. Your code checks that URL and with some *if... else* statements, it knows that you want your GPIO 5 (which is defined as `gpio5_pin`) to be set to `LOW`. A similar approach is used to control the other GPIOs on and off.

The web page is sent to the client using the `client.println()` function. It's just a basic web page that uses the Bootstrap framework (see code below).

Learn more about the Bootstrap framework: <http://getbootstrap.com/>.

```

client.println("<!DOCTYPE HTML>");
client.println("<html>");
client.println("<head>");
client.println("<meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">");
client.println("<link rel=\"stylesheet\" href=\"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/css/bootstrap.min.css\">");
client.println("</head><div class=\"container\">");
client.println("<h1>Web Server</h1>");
client.println("<h2>GPIO 5 - Current State: " + gpio5_state);
client.println("<div class=\"row\">");
client.println("<div class=\"col-md-2\"><a href=\"/gpio5on\" class=\"btn btn-block btn-lg btn-success\" role=\"button\">ON</a></div>");
client.println("<div class=\"col-md-2\"><a href=\"/gpio5off\" class=\"btn btn-block btn-lg btn-danger\" role=\"button\">OFF</a></div>");
client.println("</div>");

```

```

client.println("<h2>GPIO 4 - Current State: " + gpio4_state);
client.println("<div class=\"row\">");
client.println("<div class=\"col-md-2\"><a href=\"/gpio4on\" class=\"btn btn-block btn-lg btn-success\""
role=\"button\">ON</a></div>");

client.println("<div class=\"col-md-2\"><a href=\"/gpio4off\" class=\"btn btn-block btn-lg btn-danger\" role=\"button\">OFF</a></div>");
client.println("</div></div></html>");

```

Your web page contains four buttons to turn your LEDs on and off: two buttons for GPIO 5 and other two for GPIO 4.

Your buttons are `` HTML tags with a CSS class that gives them a button look. When you press a button, you open another web page that has a different URL. And that's how your ESP8266 knows what it needs to do.

If you've entered the wrong credentials, it prints a text message in your browser saying "Authentication failed".

```

else {
    client.println("HTTP/1.1 401 Unauthorized");
    client.println("WWW-Authenticate: Basic realm=\"Secure\"");
    client.println("Content-Type: text/html");
    client.println();
    client.println("<html>Authentication failed</html>");
}

```

The final lines of code do the following: clean the `header` variable, stop the `loop()` and close the connection.

```

header = "";
break;
}
if (c == '\n') {
    // when starts reading a new line
    blank_line = true;
}
else if (c != '\r') {
    // when finds a character on the current line
    blank_line = false;
}
}
// closing the client connection
delay(1);
client.stop();
Serial.println("Client disconnected.");

```

Encoding Your Username and Password

At this point if you upload the code presented in the preceding section, your username is *user* and your password is *pass*. I'm sure you want to change and customize this example with your own credentials.

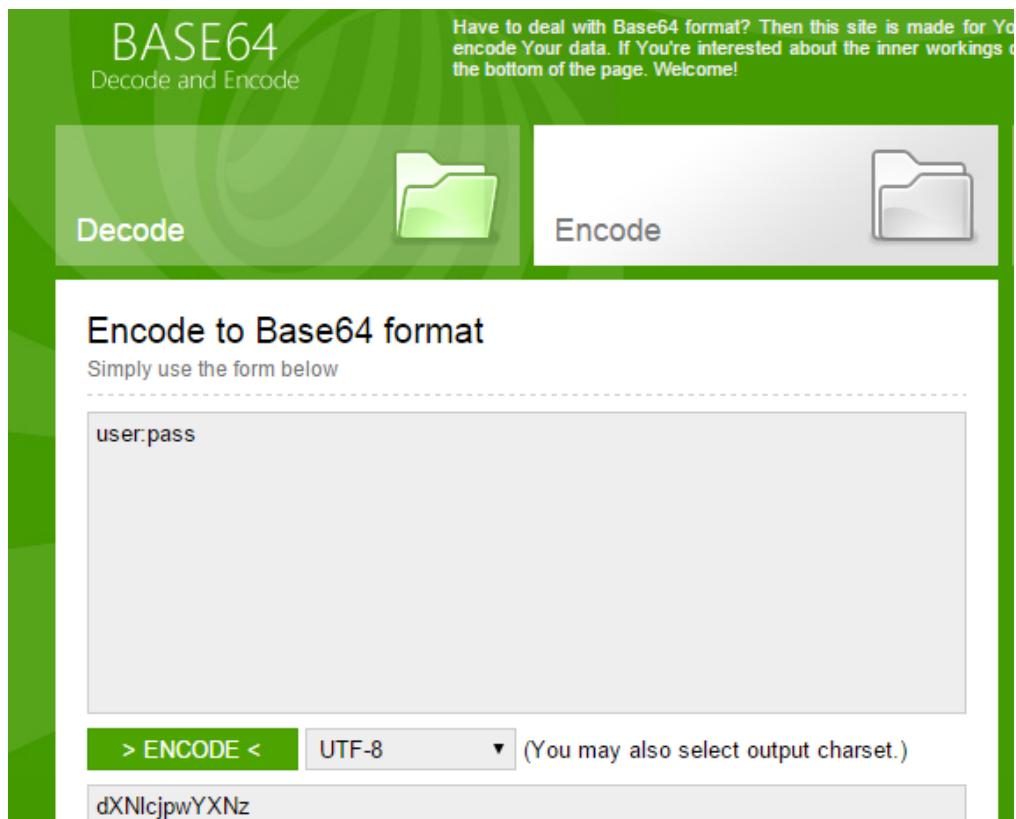
Go to the following URL: <https://www.base64encode.org>.

In the first field, type the following:

```
your_username:your_password
```

Note: you need to type the ":" between your username and your password.

In my example, I've entered *user:pass* (as illustrated in the figure below):



Then, press the "**Encode**" green button to generate your base64 encoded string. In my example it's `dXNlcjpwYXNz`.

Copy your encoded string and replace it in your sketch:

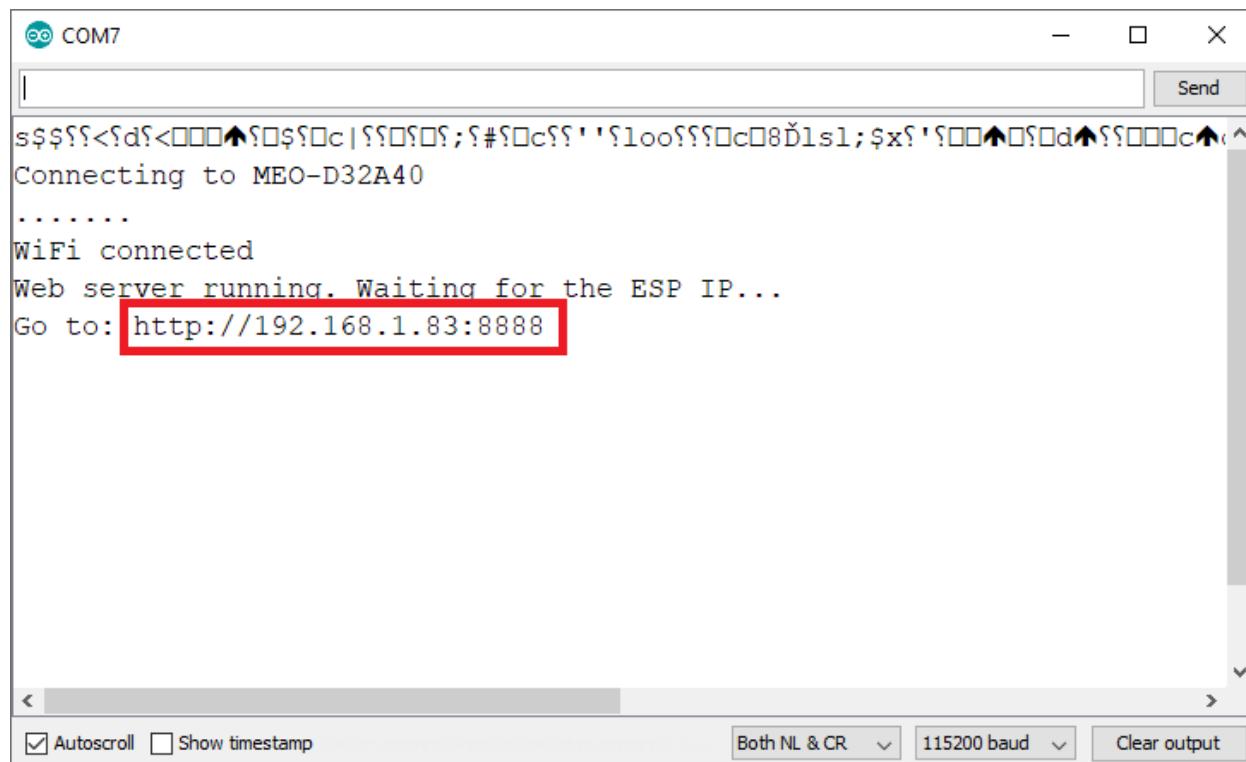
```
if(header.indexOf("dXNlcjpwYXNz") >= 0)
```

Tip: you can find that *if* statement above on [line 92 of the code](#).

ESP8266 IP Address

Upload the code to your ESP8266 board. After uploading your web server Sketch to your ESP8266, go to **Tools > Serial Monitor**. In your Serial Monitor window you're going to see your ESP8266 IP address showing up when your ESP8266 first boots (it may take a few seconds).

My IP address is 192.168.1.83 (as shown in the figure below). Your IP should be different, **save your ESP8266 IP** so that you can access the web server later in this Unit.



The screenshot shows the Arduino Serial Monitor window titled "COM7". The text output is as follows:

```
s$#??<d?<□□□▲?□$?□c | ??□?□?;?#?□c?? ''?looo??□c□8D1s1;?x?,'?□□▲?□?d▲?□□□c▲??
Connecting to MEO-D32A40
.....
WiFi connected
Web server running. Waiting for the ESP IP...
Go to: http://192.168.1.83:8888
```

The line "Go to: http://192.168.1.83:8888" is highlighted with a red box. At the bottom of the monitor, there are checkboxes for "Autoscroll" and "Show timestamp", and dropdown menus for "Both NL & CR", "115200 baud", and "Clear output".

Important: set the Serial Monitor baud rate to 115200, otherwise you won't see anything. You may also need to press the ESP8266 on-board RST button once the Serial Monitor is initialized.

Install an IP Scanner Software

If you still can't see the ESP IP address. We suggest installing an IP Scanner software. This software searches for all the devices in your network. Follow these next instructions to install an IP Scanner software:

1. Download this free software:
 - o Windows PC: www.advanced-ip-scanner.com
 - o MAC OS X, Windows or Linux: <http://angryip.org>
2. Install the software (while having your ESP running with that web server script);
3. Open the IP Scanner software and press "**Scan**";
4. Let that process finish (it can take a couple of minutes) and you should be able to identify your ESP8266 board IP address.

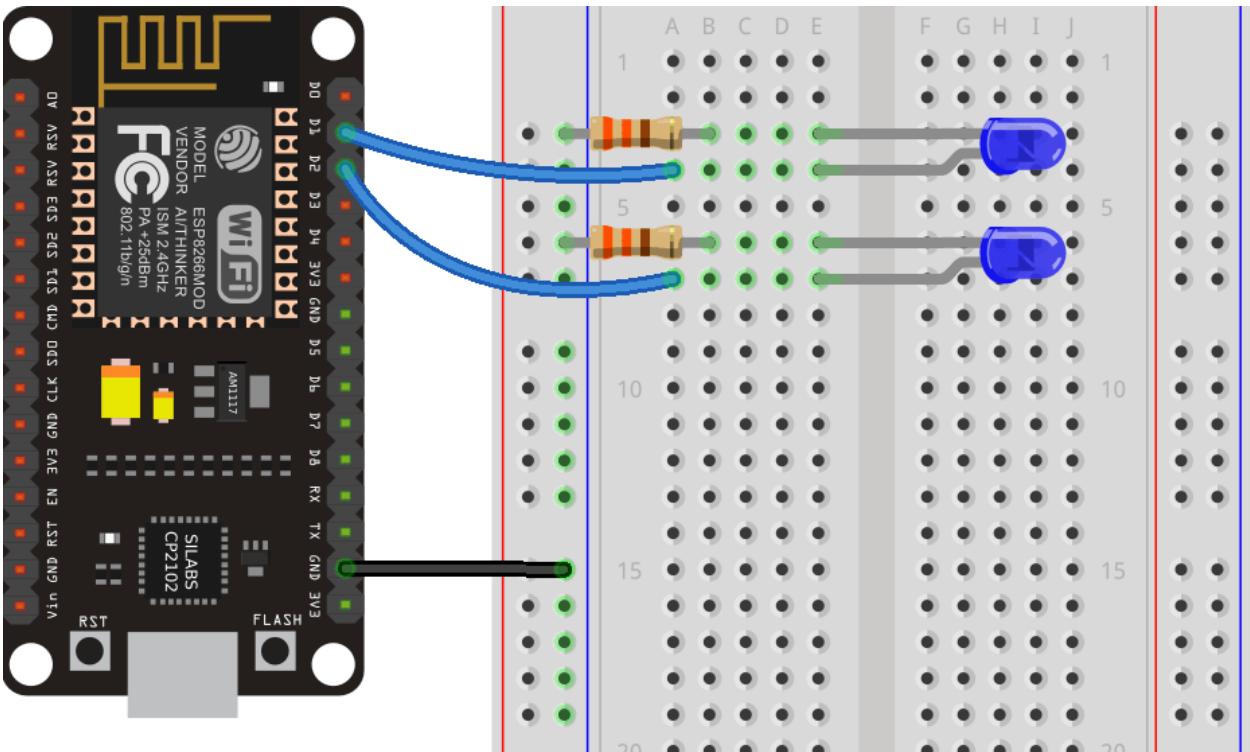
Schematic

To build the circuit for this project you need the following parts:

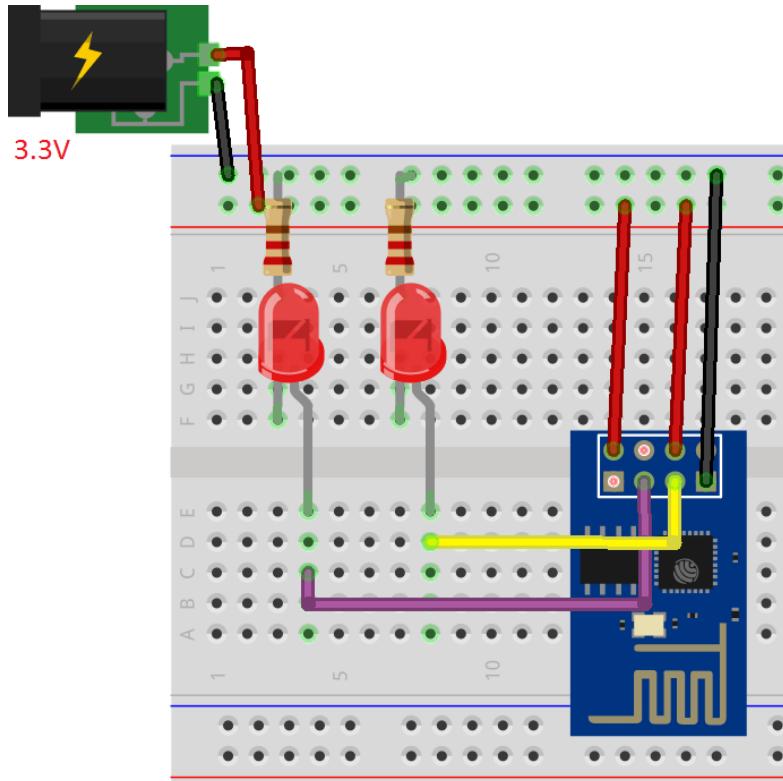
- [ESP8266](#)
- 2x [LEDs](#)
- 2x 330 Ohm [Resistors](#)
- [Breadboard](#)
- [Jumper wires](#)

If you're using [ESP-01](#), you also need an [FTDI programmer](#) or an [ESP8266 Serial Adapter](#).

Connect two LEDs to your ESP8266 as shown in the following schematic diagram – with one LED connected to GPIO 4 (D2), and another to GPIO 5 (D1).



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).



Accessing Your Web Server

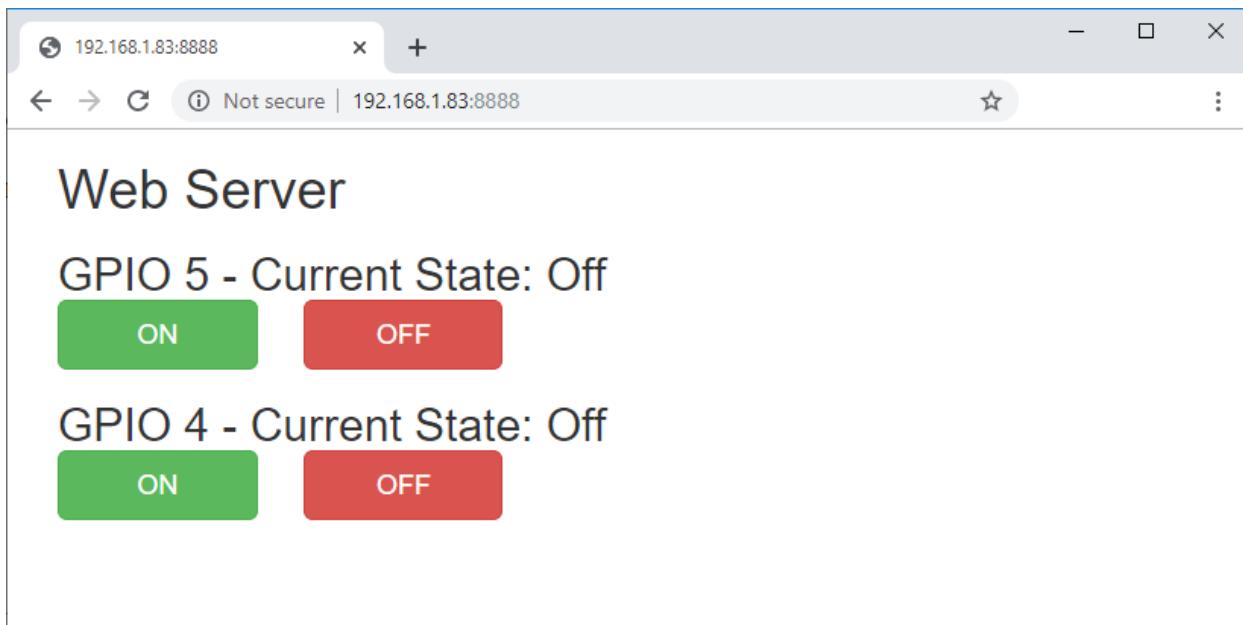
Now follow the next instructions before accessing your web server:

1. Restart your ESP8266 (press the on-board RST button);
2. Open a browser;
3. Type the IP address that you've previously saved in the URL bar followed by :8888 (in my case: <http://192.168.1.83:8888>) ;
4. It should require that you enter your username and password in order to open your web server. This is what you should see:



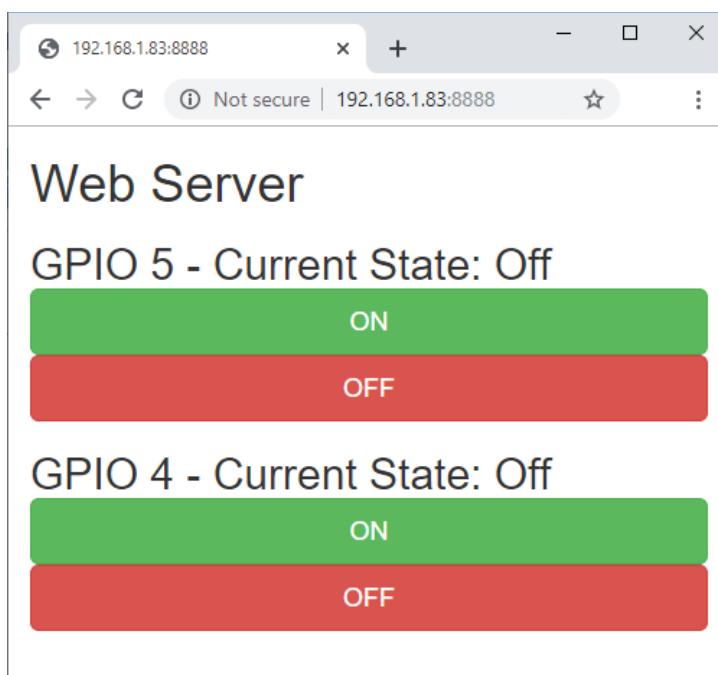
5. Enter your username and password;
6. Press Login.

The following web page should load:



Note: at this point, to access your web server, you need to be connected to the same router that your ESP8266 is (the ESP8266 needs to be connected to your local network).

That was fun! Having such an inexpensive WiFi board that can act as a web server and serves mobile responsive web page is amazing. You should be able to control the LEDs by pressing the buttons. Notice that the state of the GPIOs is also updated on the web page.

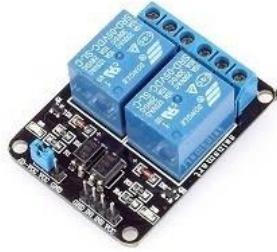


Taking It Further

We hope you're happy to see those LEDs turning on and off! We know those are just LEDs, but creating a web server just like you did is an extremely useful concept.

Controlling some house appliances may be more exciting than lighting up an LED. You can easily and immediately replace the LED with a new component that allows you to control any device that connects directly to the sockets on the wall. For example:

Option A – Relay



A relay is an electrically operated switch that can be turned on or off, letting the current go through or not, and can be controlled with low voltages, like the 3.3V provided by the ESP8266 pins. So, you can control mains voltage with 3.3V signals.

This requires a bit of extra knowledge and wariness because you're dealing with alternating current (AC) and it can be very dangerous if you don't know what you're dealing with.

I won't discuss this subject in detail, but here's a good starting point:

- <http://randomnerdtutorials.com/guide-for-relay-module-with-arduino/>

The preceding link takes you to the Random Nerd Tutorials blog and shows how to control relays using an Arduino, but you can apply the same concepts to your ESP8266 module.

Option B – Remote controlled sockets

With this option you can build a web server with the ESP8266 to remotely control any sockets safely.



That's what we're going to do in the next Unit.

Unit 3: Control Sockets Remotely via Web Server



In this Unit, you're going to build a web server that can remotely control any sockets safely. We'll be using 433 MHz remotely controlled sockets.

Project Overview

This project is divided into two sections:

- **Decode and Send 433 MHz RF Signals:** you'll "discover" the signals sent from the remote to control the sockets, and you'll replicate those signals using your ESP8266 and a 433 MHz transmitter module.
- **Control the sockets via web server:** you'll create a web server to control the sockets remotely using your smartphone. When you click one of the buttons, the ESP8266 will send a 433 MHz signal to control the corresponding socket.

Parts Required

For this project, you'll need the following parts:

- [ESP-01](#) or [ESP8266-12E NodeMCU Kit](#)
- [FTDI Programmer](#) or [Serial Adapter](#) (if using an ESP-01)
- [Arduino UNO](#)
- [433MHz Receiver/Transmitter module](#)
- [RF remote controlled sockets that operate at 433MHz](#)

Remote controlled sockets (433MHz)

You can buy remote controlled sockets in any store or you can [find them here](#). Keep in mind that they need to communicate via RF at 433MHz.



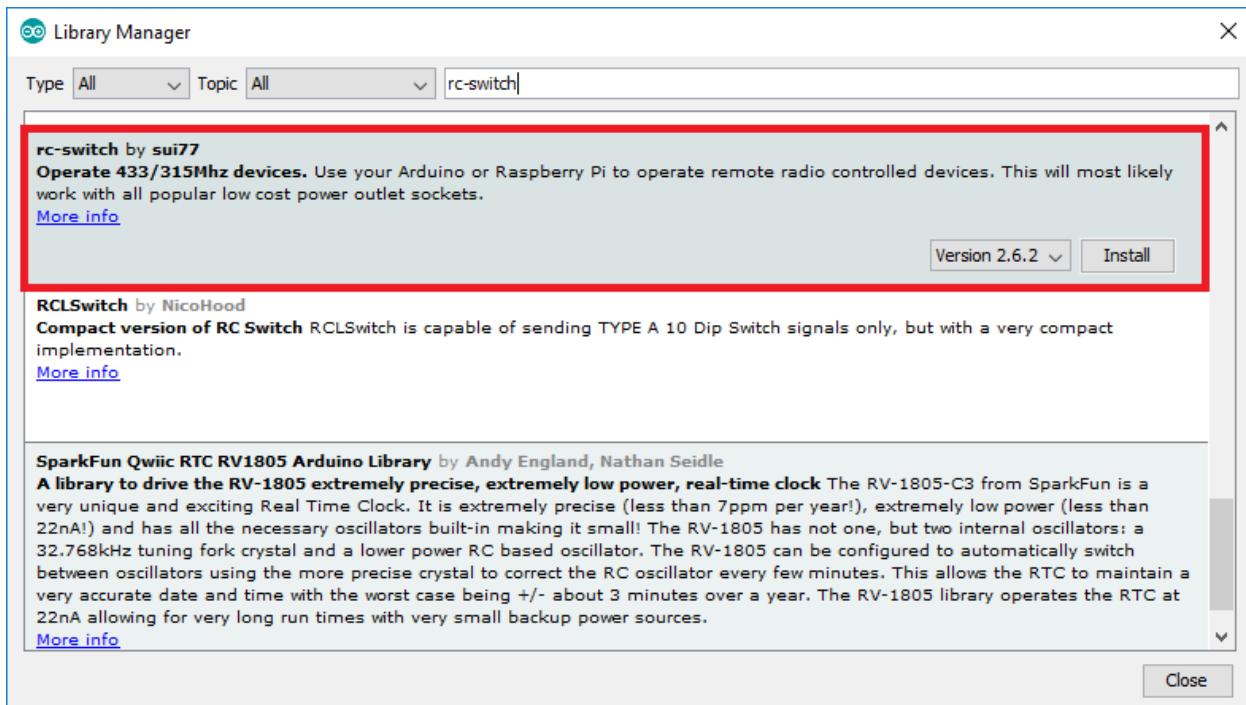
Here's my setup:

- Remote control – channel 1
- Socket 1 – channel 1 and mode 1
- Socket 2 – channel 1 and mode 3

RC Switch Library

To decode the 433 MHz signals sent from the remote to control the sockets, we'll use the RC Switch library. Follow the next steps to install that library:

1. Open your Arduino IDE and go to **Sketch > Include Library > Manage Libraries**. The Library Manager should open;
2. Type “**rc-switch**” in the search box and install the **rc-switch library by sui77**;

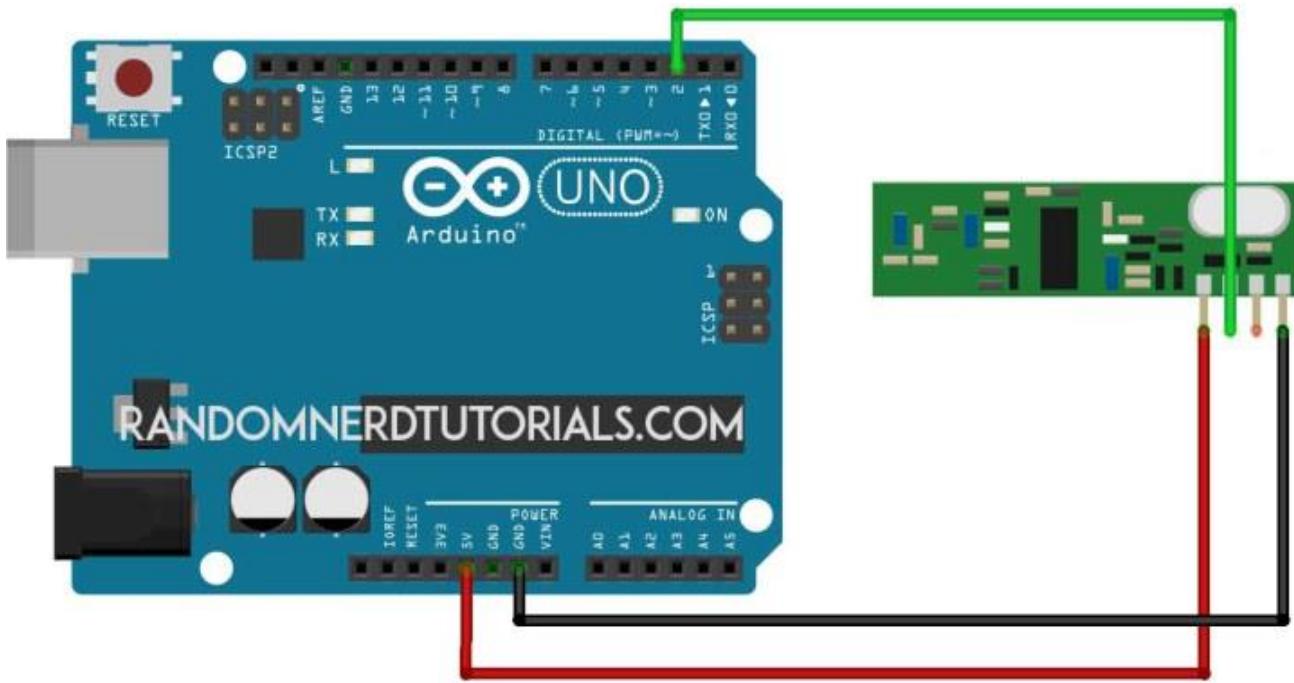


3. After installing the libraries, restart your Arduino IDE.

The RC Switch library is great, and it works with almost any remote-controlled sockets in the market.

Receiver

Connect a 433 MHz receiver to an Arduino as shown in the next schematic diagram.



Follow the circuit above for your receiver. Then, in your Arduino IDE, go to **File** ▶ **Examples** ▶ **RC Switch** ▶ **ReceiveDemo_Advanced**.

```
#include <RCSwitch.h>
RCSwitch mySwitch = RCSwitch();
void setup() {
  Serial.begin(9600);
  mySwitch.enableReceive(0); // Receiver on interrupt 0 => that is pin #2
}
void loop() {
  if (mySwitch.available()) {
    output(mySwitch.getReceivedValue(), mySwitch.getReceivedBitlength(),
    mySwitch.getReceivedDelay(),
    mySwitch.getReceivedRawdata(), mySwitch.getReceivedProtocol());
    mySwitch.resetAvailable();
  }
}
```

Save the TriState values

Open the Serial Monitor at a baud rate of 9600 and start pressing the buttons of your remote. Save the TriState values (highlighted in red) of each key in a notepad.



```
Decimal: 1381717 (24Bit) Binary: 000101010001010101010101 Tri-State: 0FFFF0FFFFFF PulseLength: 417 microseconds Protocol: 1
Raw data: 12948,400,1272,396,1268,400,1272,1232,432,396,1272,1232,440,396,1268,1232,440,396,1272,396,1272,1228,440,396
Decimal: 1381716 (24Bit) Binary: 000101010001010101010100 Tri-State: 0FFF0FFFFFF0 PulseLength: 417 microseconds Protocol: 1
Raw data: 12948,400,1268,400,1268,400,1268,1236,436,396,1272,1232,436,400,1272,1232,440,392,1272,396,1272,1232,440,396
Decimal: 1397077 (24Bit) Binary: 000101010101000101010101 Tri-State: 0FFFFF0FFFFF PulseLength: 417 microseconds Protocol: 1
Raw data: 12952,396,1272,396,1272,400,1268,1232,436,396,1268,1232,440,396,1272,1232,440,392,1276,1228,440,396,1272,1232,444,388
Decimal: 1397076 (24Bit) Binary: 000101010101000101010100 Tri-State: 0FFFFF0FFFF0 PulseLength: 417 microseconds Protocol: 1
Raw data: 12948,396,1268,400,1272,400,1268,1232,436,400,1272,1232,436,400,1268,1236,436,396,1272,1232,440,396,1268,1228,440,396
```

ESP8266 Code

Copy the next sketch to your Arduino IDE. Replace the SSID and password with your own credentials. You also need to change the TriState values. After modifying those variables, you can upload the sketch to your ESP8266.

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include <RCswitch.h>

RCswitch mySwitch = RCswitch();
MDNSResponder mdns;

// Replace with your network credentials
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";
ESP8266WebServer server(8888);

// Replace with your remote TriState values
char* socket1TriStateOn = "0FFF0FFFFFF";
char* socket1TriStateOff = "0FFF0FFFFFF0";
char* socket2TriStateOn = "0FFFFF0FFFFF";
char* socket2TriStateOff = "0FFFFF0FFFF0";

String webPage = "";

void setup(void) {
```

```

webPage += "<h1>ESP8266 Web Server</h1><p>Socket #1 <a href=\"socket1On\"><button>ON</button></a>&nbsp;<a href=\"socket1Off\"><button>OFF</button></a></p>";
webPage += "<p>Socket #2 <a href=\"socket2On\"><button>ON</button></a>&nbsp;<a href=\"socket2Off\"><button>OFF</button></a></p>";
mySwitch.enableTransmit(2);
delay(1000);
Serial.begin(115200);
WiFi.begin(ssid, password);
Serial.println("");
// Wait for connection
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("");
Serial.print("Connected to ");
Serial.println(ssid);
Serial.print("IP address: ");
Serial.println(WiFi.localIP());

if (mdns.begin("esp8266", WiFi.localIP())) {
    Serial.println("MDNS responder started");
}

server.on("/", [](){
    server.send(200, "text/html", webPage);
});
server.on("/socket1On", [](){
    server.send(200, "text/html", webPage);
    mySwitch.sendTriState(socket1TriStateOn);
    delay(1000);
});
server.on("/socket1Off", [](){
    server.send(200, "text/html", webPage);
    mySwitch.sendTriState(socket1TriStateOff);
    delay(1000);
});
server.on("/socket2On", [](){
    server.send(200, "text/html", webPage);
    mySwitch.sendTriState(socket2TriStateOn);
    delay(1000);
});
server.on("/socket2Off", [](){
    server.send(200, "text/html", webPage);
    mySwitch.sendTriState(socket2TriStateOff);
    delay(1000);
});
server.begin();
Serial.println("HTTP server started");
}
void loop(void){
    server.handleClient();
}

```

SOURCE CODE

[https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module4/Unit3/Remote Controlled Sockets Web Server/Remote Controlled_Sockets_Web_Server.ino](https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module4/Unit3/Remote%20Controlled%20Sockets%20Web%20Server/Remote_Controlled_Sockets_Web_Server.ino)

How the Code Works

You start by including the following libraries. In this example, we'll build a web server using the `ESP8266WebServer` library.

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include <RCSwitch.h>
```

Create an `RCSwitch` object so that you're able to send 433 MHz signals with the ESP8266 and a 433 MHz transmitter.

```
RCSwitch mySwitch = RCSwitch();
```

You need to create an `MDNSResponder` to handle the web server.

```
MDNSResponder mdns;
```

Insert your network credentials in the following variables, so that the ESP8266 can connect to your local network.

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

Create a webserver object on port 8888

```
ESP8266WebServer server(8888);
```

Save your own tristate values on the following variables:

```
char* socket1TriStateOn = "0FFF0FFFFFF";
char* socket1TriStateOff = "0FFF0FFFFFF0";
char* socket2TriStateOn = "0FFFF0FFFFF";
char* socket2TriStateOff = "0FFFFF0FFFF0";
```

Create a String variable to hold the HTML text to build the web page:

```
String webPage = "";
```

Then, in the setup(), add the HTML text to build four buttons. Two to control one socket, and other two to control the other socket.

```
void setup(void) {
    webPage += "<h1>ESP8266 Web Server</h1><p>Socket #1 <a href=\"socket1On\"><button>ON</button></a>&nbsp;<a href=\"socket1Off\"><button>OFF</button></a></p>";
    webPage += "<p>Socket #2 <a href=\"socket2On\"><button>ON</button></a>&nbsp;<a href=\"socket2Off\"><button>OFF</button></a></p>";
```

Enable transmission of the 433 MHz signals. The transmitter is connected to GPIO 2.

```
mySwitch.enableTransmit(2);
```

Connect to your local network and print the ESP8266 IP address:

```
Serial.begin(115200);
WiFi.begin(ssid, password);
Serial.println("");
// Wait for connection
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("");
Serial.print("Connected to ");
Serial.println(ssid);
Serial.print("IP address: ");
Serial.println(WiFi.localIP());
```

Then, handle what happens when a new request is made on a specific URL.

When, you access the root URL, for example: <http://192.168.1.83:8888>, you send the HTML text stored on the webPage String variable.

```
server.on("/", []() {
    server.send(200, "text/html", webPage);
});
```

When you press the socket #1 ON button, you send a request on the /socket1On URL and you send the signal with the corresponding tristate value.

```
server.on("/socket1On", []() {
    server.send(200, "text/html", webPage);
    mySwitch.sendTriState(socket1TriStateOn);
    delay(1000);
});
```

There's a similar approach to handle the other requests:

```
server.on("/socket1Off", []() {
    server.send(200, "text/html", webPage);
    mySwitch.sendTriState(socket1TriStateOff);
    delay(1000);
});
server.on("/socket2On", []() {
    server.send(200, "text/html", webPage);
    mySwitch.sendTriState(socket2TriStateOn);
    delay(1000);
});
server.on("/socket2Off", []() {
    server.send(200, "text/html", webPage);
    mySwitch.sendTriState(socket2TriStateOff);
    delay(1000);
});
server.begin();
Serial.println("HTTP server started");
}
```

In the loop(), you need to add the following line to handle the client.

```
void loop(void) {
    server.handleClient();
}
```

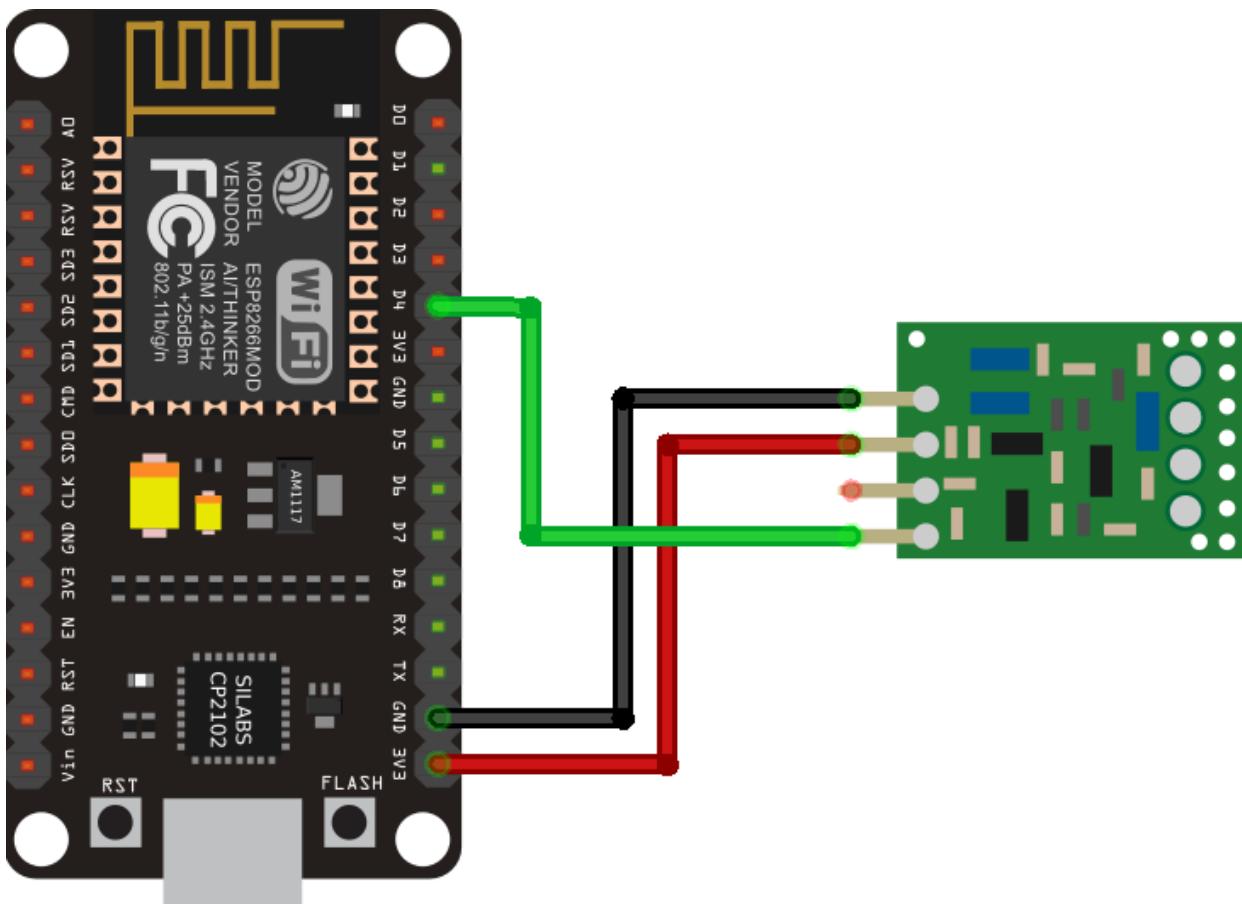
ESP8266 IP address

Open the Arduino IDE Serial monitor at a baud rate of 115200. If you're using an ESP-01, connect GPIO 0 to VCC and reset your board. After a few seconds your IP address should appear. In our case, it's **192.168.1.83:8888**.

Final circuit

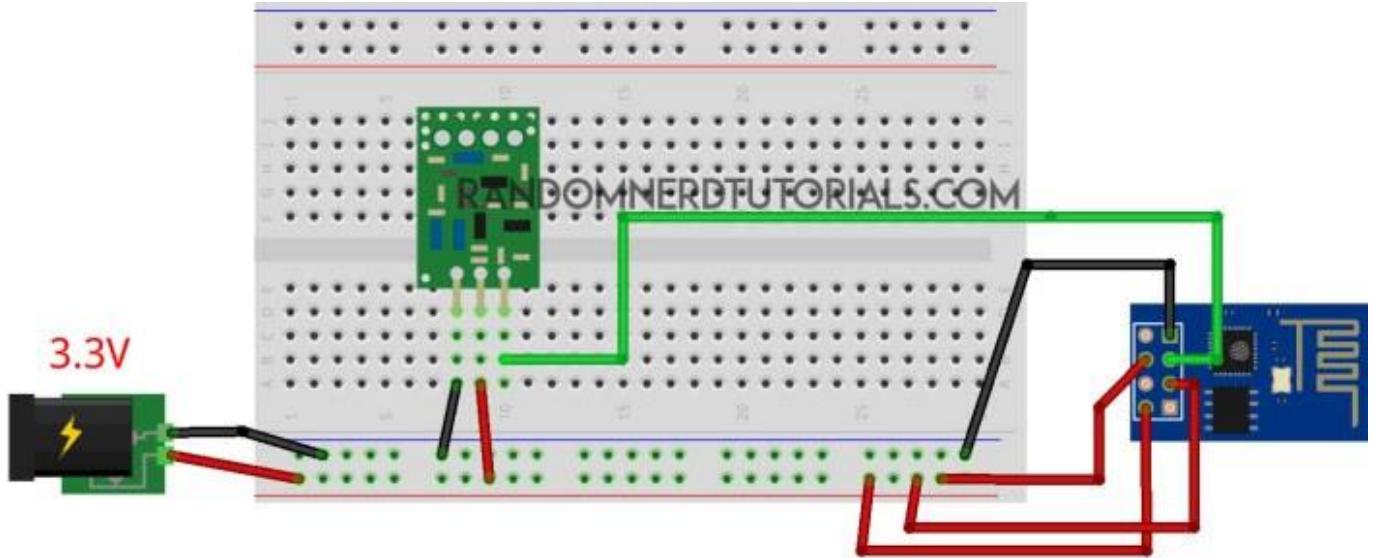
Finally, add the 433 MHz transmitter module to the ESP8266, so it is able to send the 433 MHz signals to control your sockets.

Follow the next schematic diagram if you're using an ESP8266-12E.



Note: double-check the wiring for the transmitter you're using and connect the data pin to GPIO 3.

Follow the next schematic diagram if you're using an ESP-01.



Demonstration

Now, test your setup. Open any browser from a device that is connected to the same router that your ESP8266. Type the IP address and press Enter.

The following page should load.



When you press the buttons in your web server, you can control both sockets on and off. Then, you can connect any appliances you want to control to those sockets.



Unit 4: Control ESP8266 with Android Widget App



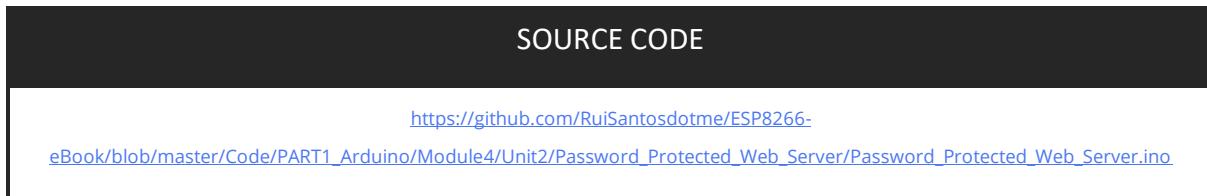
For this project we'll be using an Arduino sketch that you are already familiar with. It uses the sketch from Unit 2 and creates a password protected web server that can be accessed in your web browser to control two GPIOs.

However, instead of opening the browser and type the IP address, it's easier if you can control your ESP8266 GPIOs directly from your smartphone home screen with a simple tap. That's what this project is all about.

Uploading Code

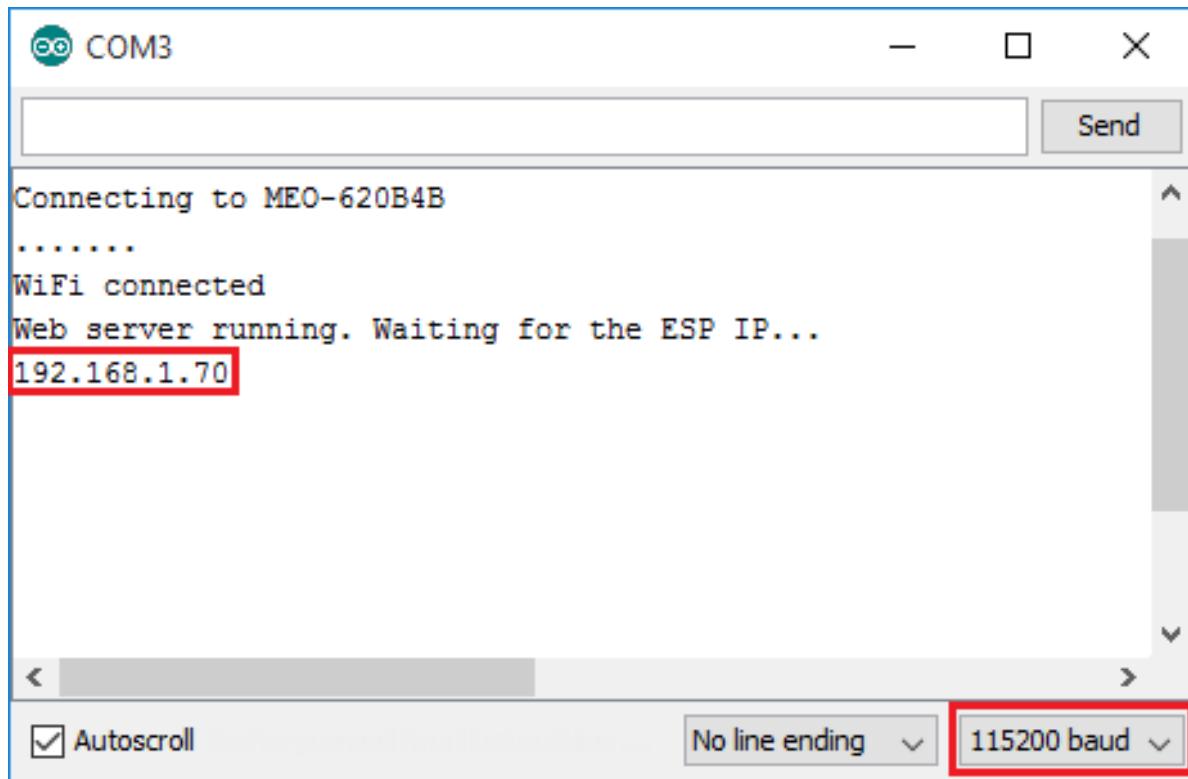
Once your ESP8266 is connected to your computer, open the Arduino IDE. Look at the **Tools** menu, select Board "**NodeMCU 1.0 (ESP-12E Module)**" and all the configurations, by default, should be fine.

Before uploading the sketch to your ESP, you need to replace two variables with your credentials (*SSID* and *password*), so that your ESP can connect to your network. Finally, press the “**Upload**” button.



ESP8266 IP Address

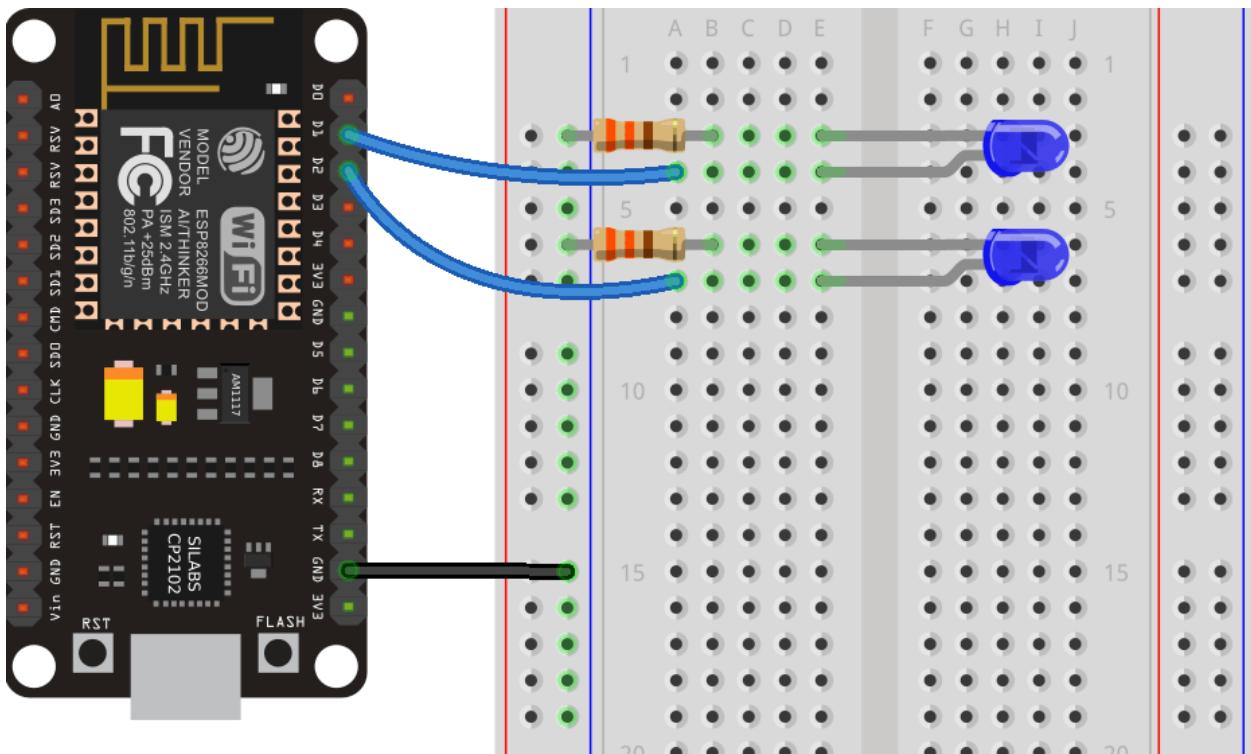
After uploading your sketch to the ESP, go to **Tools > Serial Monitor**. In your Serial Monitor window you'll see your ESP IP address when your ESP first boots.



In this case, the IP address is **192.168.1.70** (as shown in the preceding figure). Your IP should be different, save your ESP8266 IP so that you can access it later in this project.

Schematic

After uploading your code to your ESP8266, follow the next schematic diagram. Wire two LEDs, one to GPIO 5 (D1) and another to GPIO 4 (D2).



Accessing Your Web Server

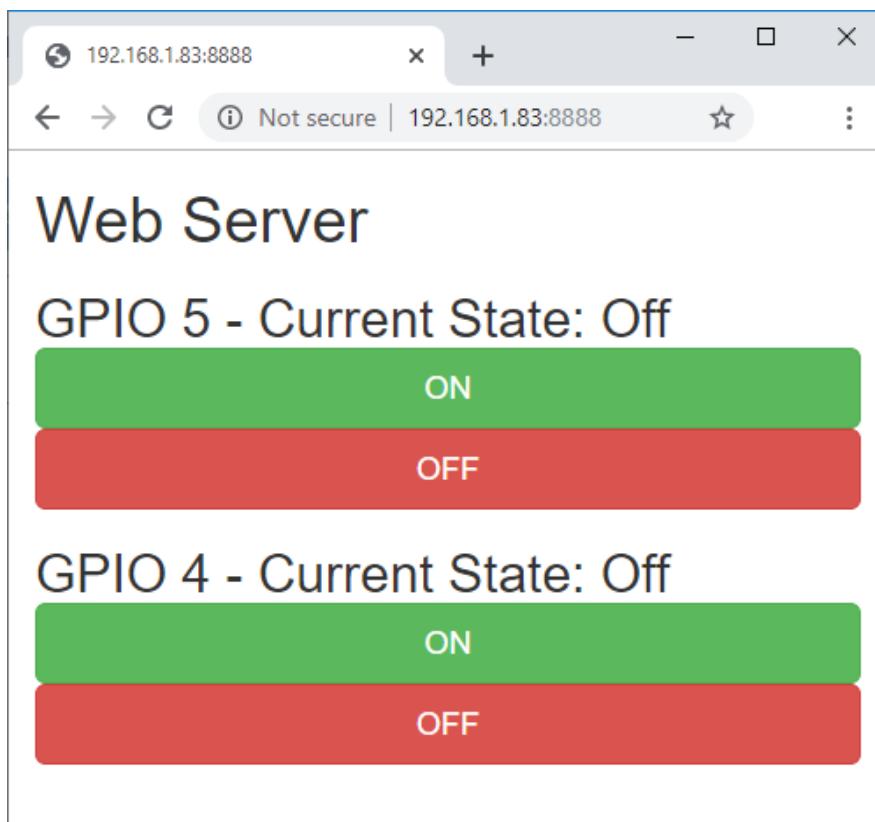
Follow the next instructions to access your web server:

- 1) Restart your ESP8266;
- 2) Open a browser;
- 3) Type the IP address that you've previously saved in the URL bar followed by :8888 (for example: <http://192.168.1.83:8888>). Enter your username and password to open your web server;

Note: the default username is **user** and password is **pass**.



- 4) Enter your username **user** and password **pass**;
- 5) Press the **Login** button. The following web page should load.

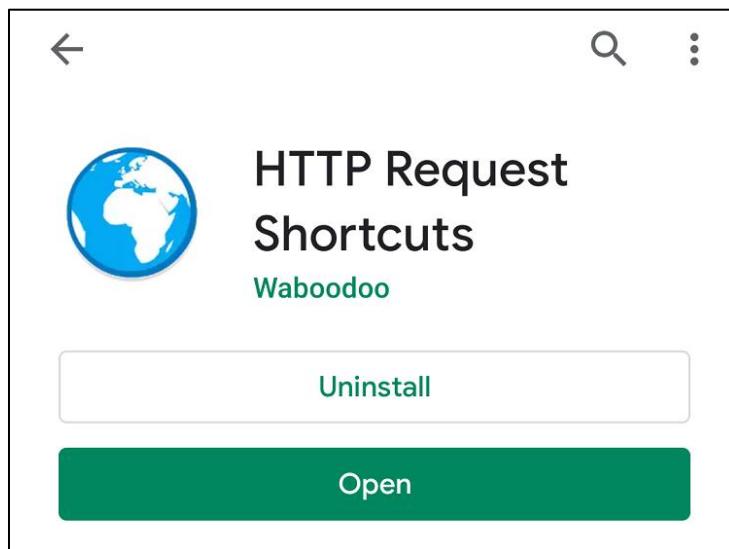


You should be able to turn your LEDs on and off through the web interface.

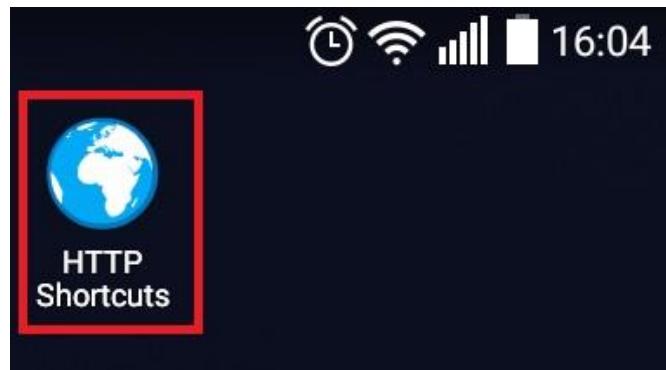
Installing the HTTP Request Shortcuts App

For this project we'll be using a free Android application that allows your smartphone to perform HTTP GET/POST requests easily with a press of a button directly from your home screen. With this app, you no longer need to type the IP address in your web browser to turn the GPIOs on or off. We'll be using the [HTTP Request Shortcuts](#) Android app.

Search for **HTTP Request Shortcuts** on Google Play Store and install the app. Then, press the **OPEN** button.

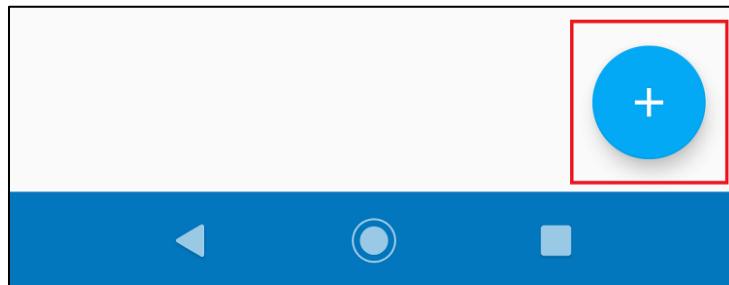


The app should create a shortcut in your home screen. Tap it to open the app.

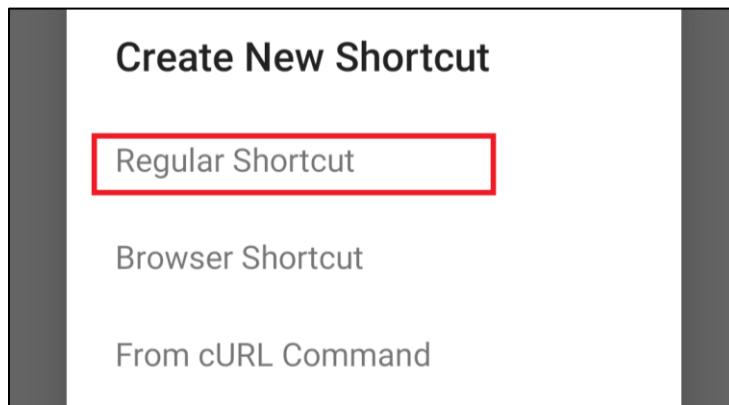


Creating ON – GPIO 5 button

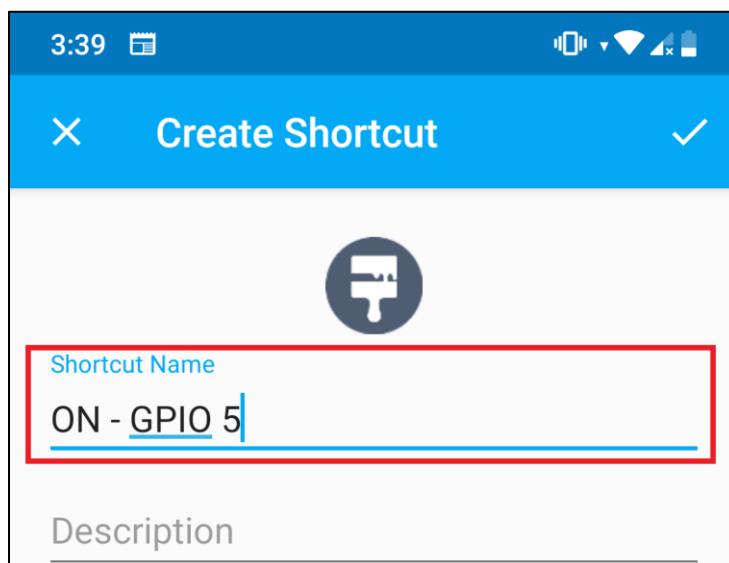
To create a widget that turns GPIO 5 on when pressed, tap the plus signal in the bottom right corner.



Select the “Regular **shortcut**” option:

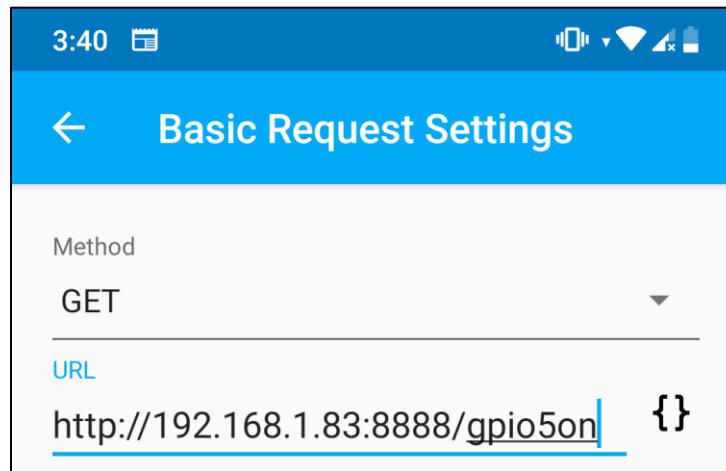


Set the shortcut name to “**ON – GPIO 5**”. You can also click the paintbrush icon to change the icon of the shortcut.



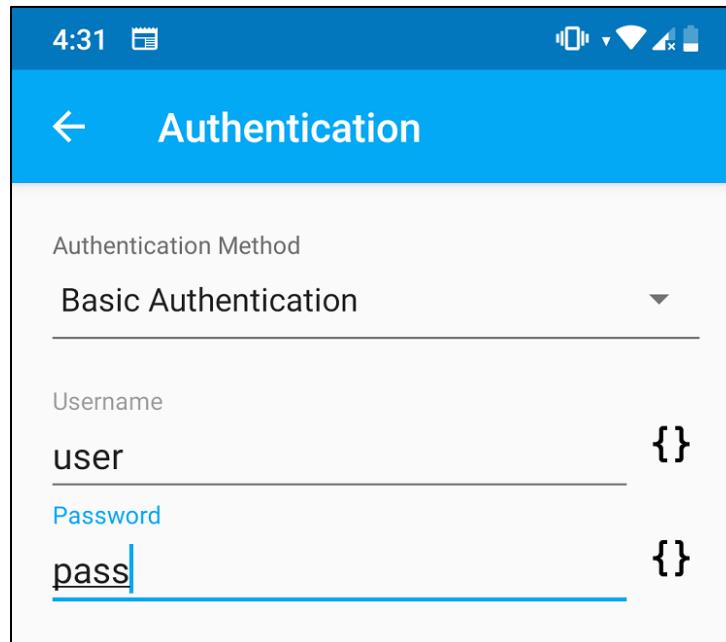
Select the **Basic Request Settings** menu and set the following:

- **Method:** GET
- **URL:** `http://YOUR_ESP_IP_ADDRESS:8888/gpio5on` - in this example, the ESP8266 IP address was 192.168.1.83

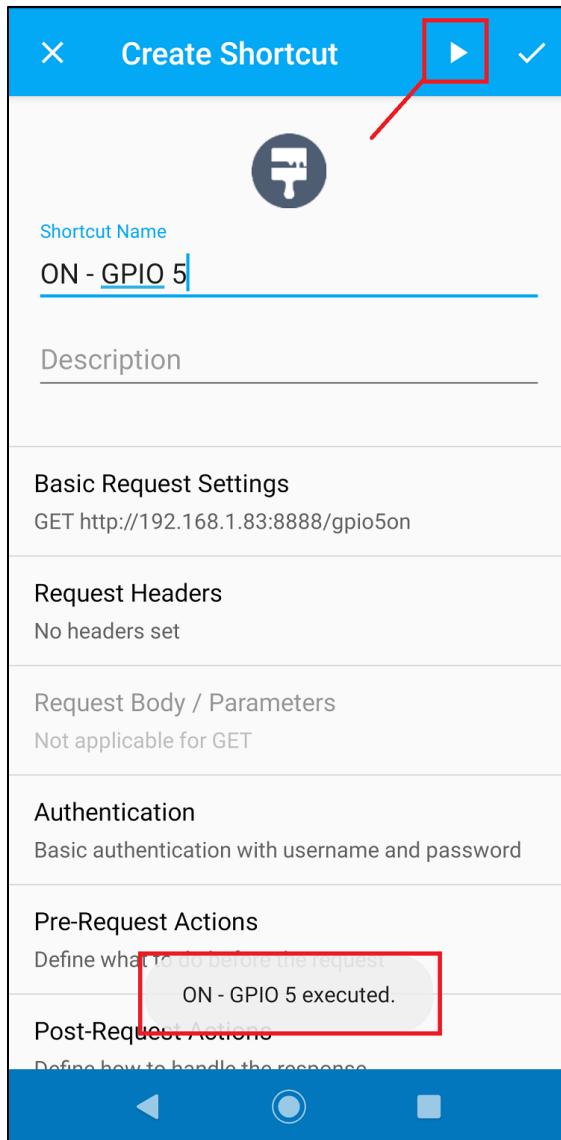


Then, go to the **Authentication** menu and select:

- **Autentication method:** basic authentication
- **Username:** the username you've defined in Unit 2
- **Password:** the corresponding password



Then, you can click the ► icon to execute the shortcut and see if it works. When you press that button, the LED connected to GPIO 5 should light up. You can also open the serial monitor and see if the ESP8266 received a request on that URL.



If everything is working as expected click ✓ to create the shortcut.

Creating OFF – GPIO 5 button

You also need to create an off button for GPIO 5. You can edit the next settings for your widget shortcut:

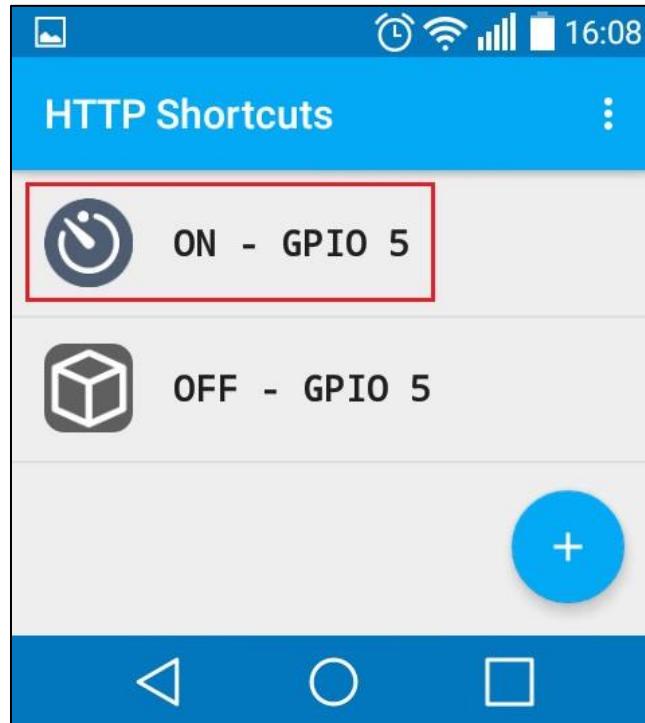
- **Shortcut Name:** OFF – GPIO 5

- **Icon:** the icon that you will appear in your home screen
- **Basic Request Settings:**
 - **Method:** GET
 - **URL:** `http://YOUR_ESP_IP_ADDRESS:8888/gpio5off`
- **Authentication:**
 - **Authentication Method:** Basic Authentication
 - **Username:** the username you've defined in Unit 2
 - **Password:** the corresponding password

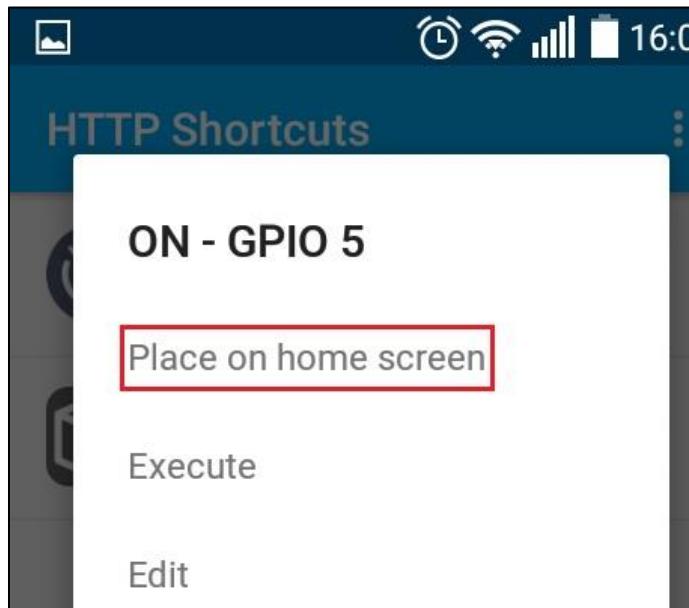
When you're done, press the tick button on the top right corner to create the shortcut.

Adding Widgets to Home Screen

To add a widget to your home screen, you simply hold down the **ON – GPIO 5** button:



Then, select the “**Place on home screen**” option that appears (repeat that process to OFF – GPIO 5 widget).



Testing

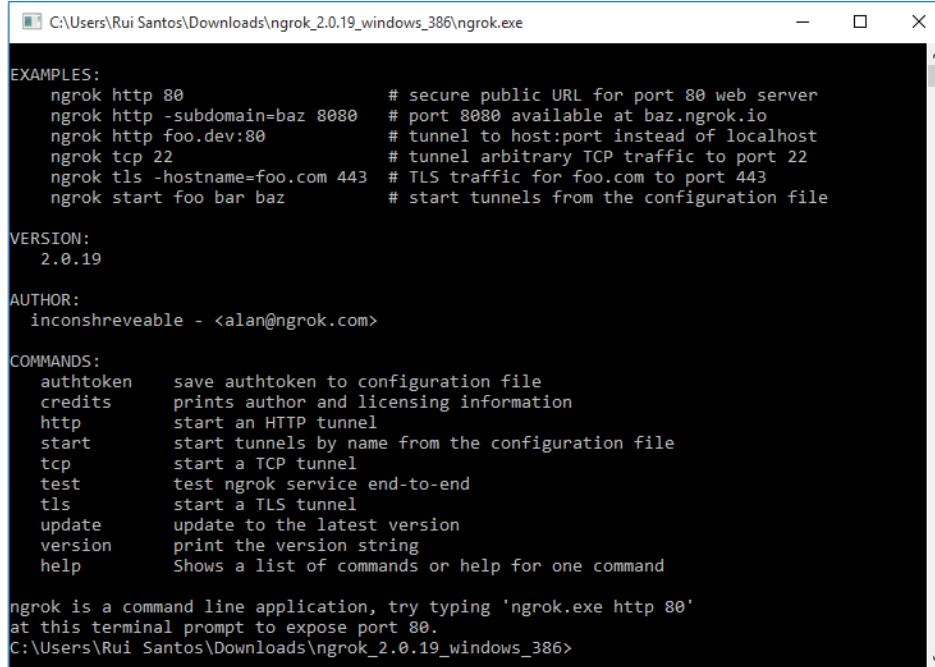
The icons should be displayed in your home screen. Now, when you press each button widget, you should be able to control GPIO 5 from your smartphone home screen.



The ESP8266 code is also prepared to control GPIO 4. We recommend following the exact same steps to add GPIO 4 control to your home screen.

You can also use this app with the project from Unit 3 to control the sockets. You just need to insert the URLs to control the GPIOs.

Unit 5: Making Your Web Server Accessible from Anywhere in the World



```
C:\Users\Rui Santos\Downloads\ngrok_2.0.19_windows_386\ngrok.exe

EXAMPLES:
  ngrok http 80          # secure public URL for port 80 web server
  ngrok http -subdomain=foo 8080  # port 8080 available at foo.ngrok.io
  ngrok http foo.dev:80    # tunnel to host:port instead of localhost
  ngrok tcp 22            # tunnel arbitrary TCP traffic to port 22
  ngrok tls -hostname=foo.com 443 # TLS traffic for foo.com to port 443
  ngrok start foo bar baz   # start tunnels from the configuration file

VERSION:
  2.0.19

AUTHOR:
  inconshreveable - <alan@ngrok.com>

COMMANDS:
  auth token      save auth token to configuration file
  credits        prints author and licensing information
  http           start an HTTP tunnel
  start          start tunnels by name from the configuration file
  tcp            start a TCP tunnel
  test           test ngrok service end-to-end
  tls            start a TLS tunnel
  update         update to the latest version
  version        print the version string
  help           Shows a list of commands or help for one command

ngrok is a command line application, try typing 'ngrok.exe http 80'
at this terminal prompt to expose port 80.
C:\Users\Rui Santos\Downloads\ngrok_2.0.19_windows_386>
```

In this Unit you're going to make your web server accessible from anywhere in the world. You'll be using a free service to create a secure tunnel to your ESP8266 which is running in your local network.

Creating nkgrok Account

This service is called **ngrok** and it's free. Go to <https://ngrok.com> to create your account. Click the green "Sign up for free" button:

Secure tunnels to localhost

"I want to expose a local server behind a NAT or firewall to the internet."

[Sign up for free →](#)

or start by [downloading ngrok](#)

Enter your details in the forms in the left box (as shown in the figure below).

Sign up

Your Name

Your Email

Confirm Email

Password

[Sign up](#)

 [Sign up with Github](#)

Why should I sign up?

Signing up is free! Many useful features are only available after you sign up, including:

Password Protected
Set http auth credentials to protect access to your tunnel and those you share it with.

```
ngrok http -auth "user:password" 80
```

TCP Tunnels
Expose any networked service to the internet, even ones that don't use HTTP.

```
ngrok tcp 22
```

After creating your account, login and go to the main dashboard and follow the steps to start running ngrok.

1 Download ngrok

ngrok is easy to install. Download a single binary with zero run-time dependencies.

[Mac OS X](#)

↓ Download for Windows

[Linux](#) [Mac \(32-Bit\)](#) [Windows \(32-Bit\)](#) [Linux \(ARM\)](#)
[Linux \(32-Bit\)](#) [FreeBSD \(64-Bit\)](#) [FreeBSD \(32-Bit\)](#)

2 Unzip to install

On Linux or OSX you can unzip ngrok from a terminal with the following command. On Windows, just double click ngrok.zip.

\$ unzip /path/to/ngrok.zip

Most people keep ngrok in their user folder or set an alias for easy access.

3 Connect your account

Running this command will add your account's auth token to your ngrok.yml file. This will give you more features and all open tunnels will be listed here in the dashboard.

\$./ngrok authtoken 1P0Ej3XC5vrL02D0ph15YveRf72

4 Fire it up

Read [the documentation](#) on how to use ngrok. Try it out by running it from the command line:

\$./ngrok help

To start a HTTP tunnel on port 80, run this next:

\$./ngrok http 80

1. Download ngrok to your operating system.
2. Unzip the folder that you have just downloaded.
3. Copy your unique AUTH token.
4. Open the ngrok software. You should see a window that looks as follows:

```
C:\Users\Rui Santos\Downloads\ngrok_2.0.19_windows_386\ngrok.exe

EXAMPLES:
  ngrok http 80                      # secure public URL for port 80 web server
  ngrok http -subdomain=baz 8080       # port 8080 available at baz.ngrok.io
  ngrok http foo.dev:80                # tunnel to host:port instead of localhost
  ngrok tcp 22                        # tunnel arbitrary TCP traffic to port 22
  ngrok tls -hostname=foo.com 443      # TLS traffic for foo.com to port 443
  ngrok start foo bar baz             # start tunnels from the configuration file

VERSION:
  2.0.19

AUTHOR:
  inconstreable - <alan@ngrok.com>

COMMANDS:
  auth token    save authtoken to configuration file
  credits       prints author and licensing information
  http          start an HTTP tunnel
  start         start tunnels by name from the configuration file
  tcp           start a TCP tunnel
  test          test ngrok service end-to-end
  tls           start a TLS tunnel
  update        update to the latest version
  version       print the version string
  help          Shows a list of commands or help for one command

ngrok is a command line application, try typing 'ngrok.exe http 80'
at this terminal prompt to expose port 80.
C:\Users\Rui Santos\Downloads\ngrok_2.0.19_windows_386>
```

Launching ngrok Secure Tunnel

In your terminal enter the following command and replace the red text with your own IP address and ngrok's tunnel AUTH token:

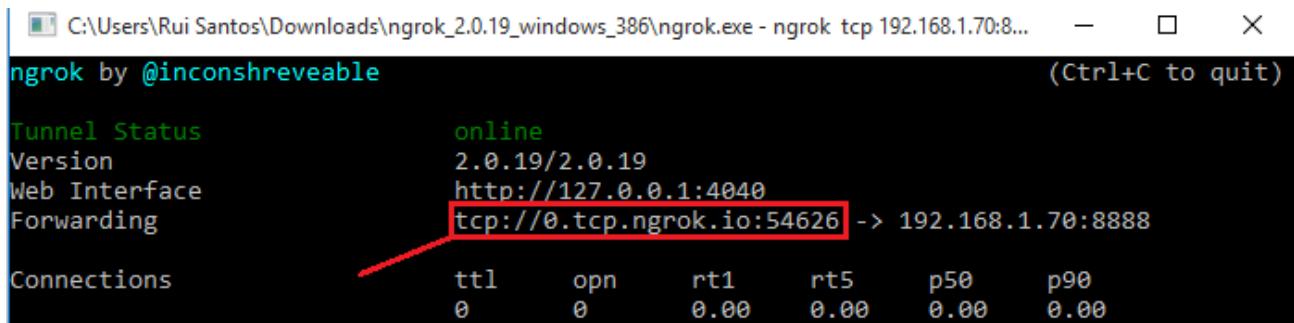
ngrok tcp 192.168.1.83:8888 --auth token 3V1MfHcNMD9rhBif8TRs_2whamY91tqX4

Here's how it looks.

```
86\ngrok.exe
\_windows_386\ngrok tcp 192.168.1.70:8888 --auth token 3V1MfHcNMD9rhBif8TRs_2whamY91tqX4
```

Press Enter to run this command.

If everything runs smoothly, you should notice that your Tunnel is **online** and a forward URL should appear in your terminal.



Tunnel Status	online
Version	2.0.19/2.0.19
Web Interface	http://127.0.0.1:4040
Forwarding	tcp://0.tcp.ngrok.io:54626 -> 192.168.1.70:8888
Connections	ttl open rt1 rt5 p50 p90 0 0 0.00 0.00 0.00 0.00

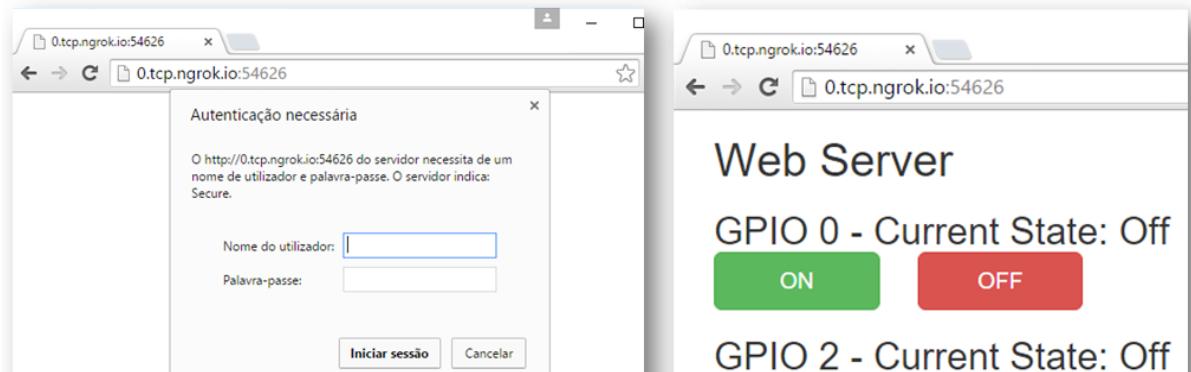
You can access your web server from anywhere in the world by typing your unique URL (in our case <http://0.tcp.ngrok.io:54626/>) in a browser.

Note: even though it's a TCP connection you type http in your web browser.

Important: you need to let your computer on with ngrok running to keep your tunnel online.

Troubleshooting: if you go to your ngrok.io URL and nothing happens, open your ESP8266 IP in your browser to see if your web server is still running. If it's still running, make sure you've entered the right IP address and Authtoken on the ngrok command executed earlier.

You'll always be asked to enter your username and password to open your web server.



Unit 6: DS18B20 Temperature Sensor Web Server

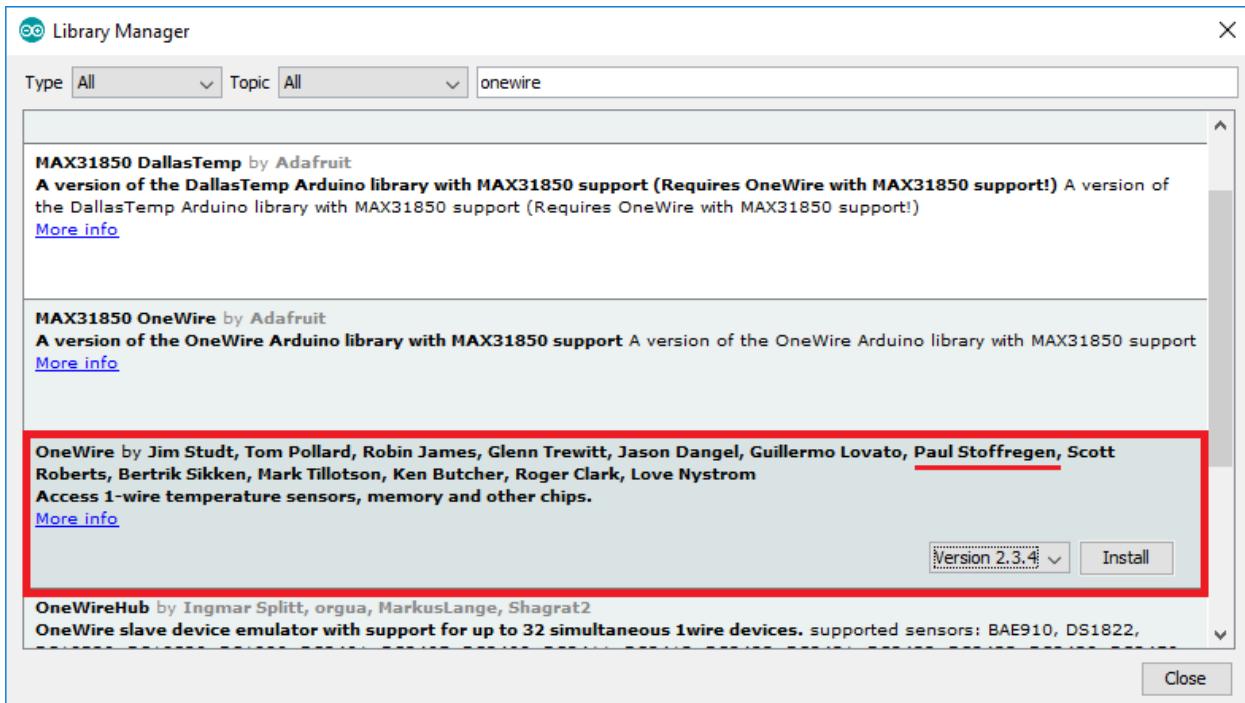


In this Unit, we'll show you how to build a simple HTTP web server that displays the temperature from a DS18B20 temperature sensor in a raw HTML page. This web server sends an HTTP response with the latest temperature readings when your browser makes a request on the ESP8266 IP address.

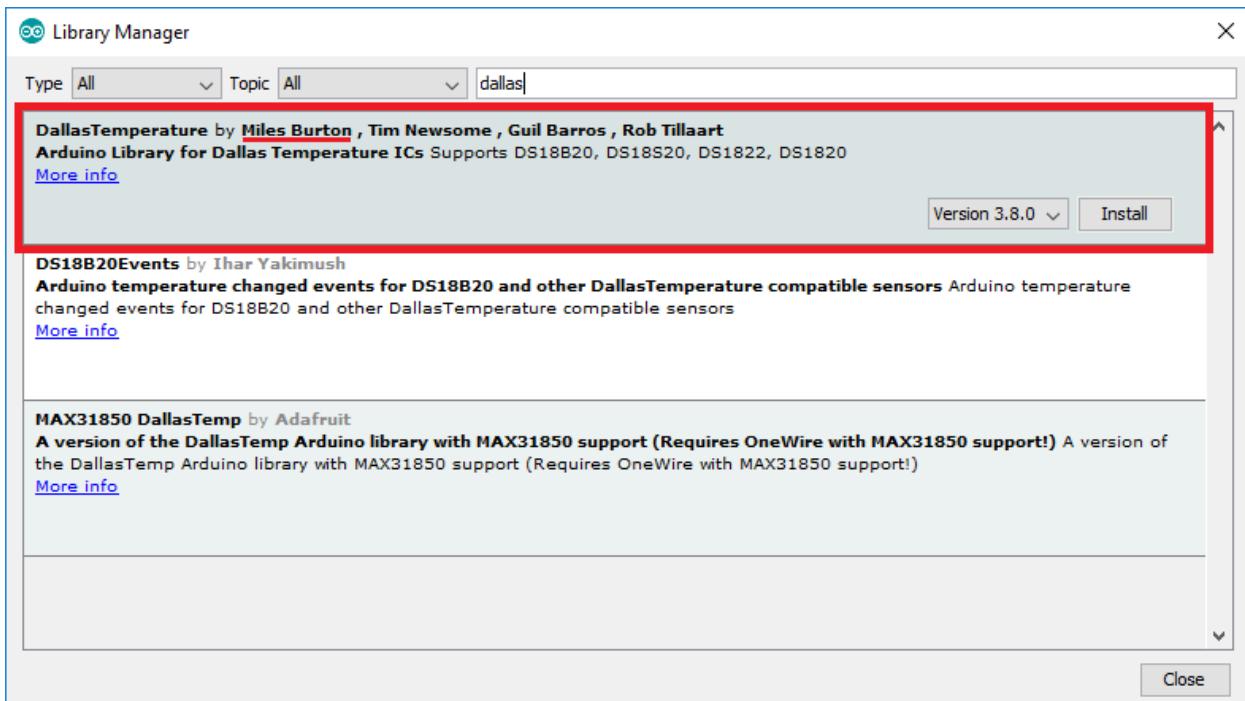
Installing Libraries

If you've followed previous Units, you should have the libraries required to work with the DS18B20 temperature sensor. If you haven't, follow the next steps to install those libraries:

1. Open your Arduino IDE and go to **Sketch** ▶ **Include Library** ▶ **Manage Libraries**. The Library Manager should open;
2. Type “**onewire**” in the search box and install the **OneWire library by Paul Stoffregen**;



3. Then, search for “**Dallas**” and install the Dallas Temperature library by Miles Burton;



4. After installing the libraries, restart your Arduino IDE.

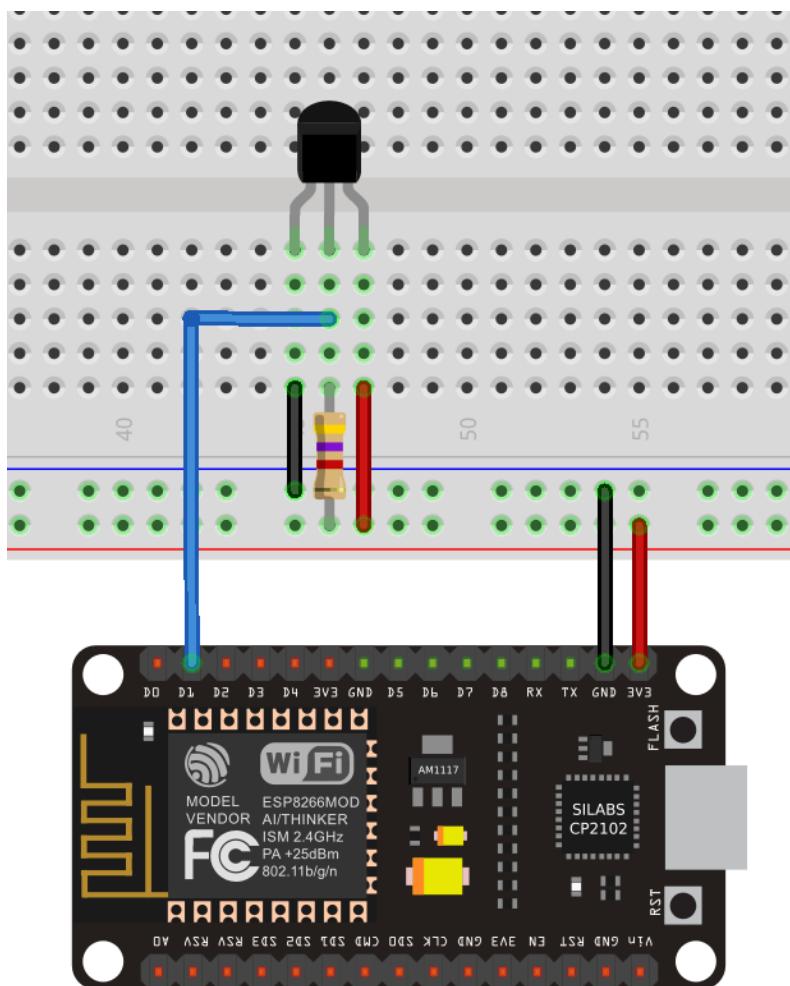
Parts Required

To complete this tutorial, you need the following components:

- [ESP8266](#)
- [DS18B20 temperature sensor](#)
- [Breadboard](#)
- [4.7k Ohm resistor](#)
- [Jumper wires](#)

Schematic

Connect the DS18B20 temperature sensor to the ESP8266 as shown in the next schematic diagram (data pin connected to GPIO 5).



Code

Copy the sketch below to your Arduino IDE. Replace the SSID and password with your network credentials. After modifying the sketch, upload it to your ESP8266.

```
// Including the ESP8266 WiFi library
#include <ESP8266WiFi.h>
#include <OneWire.h>
#include <DallasTemperature.h>

// Replace with your network details
const char* ssid = "MEO-D32A40";
const char* password = "384e6d3cec";

// Data wire is plugged into pin D1 on the ESP8266 12-E - GPIO 5
#define ONE_WIRE_BUS 5

// Setup a oneWire instance to communicate with any OneWire devices (not just
Maxim/Dallas temperature ICs)
OneWire oneWire(ONE_WIRE_BUS);

// Pass our oneWire reference to Dallas Temperature.
DallasTemperature DS18B20(&oneWire);
char temperatureCString[6];
char temperatureFString[6];

// Web Server on port 80
WiFiServer server(80);

// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in milliseconds (example: 2000ms = 2s)
const long timeoutTime = 2000;

// only runs once on boot
void setup() {
    // Initializing serial port for debugging purposes
    Serial.begin(115200);
    delay(10);

    DS18B20.begin(); // IC Default 9 bit. If you have troubles consider upping it
12. Ups the delay giving the IC more time to process the temperature measurement

    // Connecting to WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
}
```

```

Serial.println("");
Serial.println("WiFi connected");

// Starting the web server
server.begin();
Serial.println("Web server running. Waiting for the ESP IP...");
delay(10000);

// Printing the ESP IP address
Serial.println(WiFi.localIP());
}

void getTemperature() {
    float tempC;
    float tempF;
    do {
        DS18B20.requestTemperatures();
        tempC = DS18B20.getTempCByIndex(0);
        dtostrf(tempC, 2, 2, temperatureCString);
        tempF = DS18B20.getTempFByIndex(0);
        dtostrf(tempF, 3, 2, temperatureFString);
        delay(100);
    } while (tempC == 85.0 || tempC == (-127.0));
}

// runs over and over again
void loop() {
    // Listenning for new clients
    WiFiClient client = server.available();

    if (client) {
        currentTime = millis();
        previousTime = currentTime;
        Serial.println("New client");
        // boolean to locate when the http request ends
        boolean blank_line = true;
        while (client.connected() && currentTime - previousTime <= timeoutTime)
{
            currentTime = millis();
            if (client.available()) {
                char c = client.read();

                if (c == '\n' && blank_line) {
                    getTemperature();
                    client.println("HTTP/1.1 200 OK");
                    client.println("Content-Type: text/html");
                    client.println("Connection: close");
                    client.println();
                    // your actual web page that displays temperature
                    client.println("<!DOCTYPE HTML>");
                    client.println("<html><head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\"></head>");
                    client.println("<body><h1>ESP8266 Temperature</h1><h3>Temperature in Celsius: ");
                    client.println(temperatureCString);
                    client.println("*C</h3><h3>Temperature in Fahrenheit: ");
                    client.println(temperatureFString);
                    client.println("*F</h3></body></html>");
                    break;
                }
                if (c == '\n') {

```

```

        // when starts reading a new line
        blank_line = true;
    }
    else if (c != '\r') {
        // when finds a character on the current line
        blank_line = false;
    }
}
// closing the client connection
delay(1);
client.stop();
Serial.println("Client disconnected.");
}
}

```

SOURCE CODE

[https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module4/Unit6/DS18B20 Web Server/DS18B20 Web Server.ino](https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module4/Unit6/DS18B20%20Web%20Server/DS18B20%20Web%20Server.ino)

How the Code Works

The code starts by including the necessary libraries. The `ESP8266WiFi` library to connect the ESP8266 to your network; the `OneWire` and the `DallasTemperature` libraries to interface with the DS18B20 sensor.

```
#include <ESP8266WiFi.h>
#include <OneWire.h>
#include <DallasTemperature.h>
```

Insert your network credentials in the following variables so that the ESP8266 can connect to your local network:

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

Define the GPIO that the DS18B20 data pin is connected to. In this case, it is connected to GPIO 5 (D1). You can use any other suitable pin.

```
#define ONE_WIRE_BUS 5
```

Create a `oneWire` instance on the pin defined earlier to communicate with the sensor.

```
OneWire oneWire(ONE_WIRE_BUS);
```

Create an instance for the DS18B20 sensor with the one wire defined earlier.

```
DallasTemperature DS18B20(&oneWire);
```

Create two char variables that will hold the temperature values in Celsius and in Fahrenheit:

```
char temperatureCString[6];
char temperatureFString[6];
```

Set your web server to port 80:

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

The following timer variables are used to close the connection after a request (as we've seen previously on other web server projects):

```
// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in milliseconds (example: 2000ms = 2s)
const long timeoutTime = 2000;
```

In the `setup()`, initialize the Serial Monitor:

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

Initialize the temperature sensor:

```
DS18B20.begin();
```

Connect to Wi-Fi and print the ESP8266 IP address:

```
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
```

```

    Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");

// Starting the web server
server.begin();
Serial.println("Web server running. Waiting for the ESP IP...");
delay(10000);

// Printing the ESP IP address
Serial.println(WiFi.localIP());

```

After that, there's a function called `getTemperature()`. This function gets the temperature in Celsius and in Fahrenheit from the DS18B20 temperature sensor and saves them in the `tempC` and `tempF` variables:

```

tempC = DS18B20.getTempCByIndex(0);
tempF = DS18B20.getTempFByIndex(0);

```

Then, we convert the temperature that comes in float variable to a char array:

```

dtostrf(tempC, 2, 2, temperatureCString);
dtostrf(tempF, 3, 2, temperatureFString);

```

Sometimes the temperature sensor can't get valid temperature readings. When that happens, it retrieves 85°F or -127°C. So, to ensure that we get valid temperature readings we check that scenario:

```

while (tempC == 85.0 || tempC == (-127.0));

```

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

Basically, we send HTML text to build the web page with the sensor readings to the client, using `client.println()`.

As you can see, we send all the information using the following lines:

```

client.println("<!DOCTYPE HTML>");
client.println("<html><head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\"></head>");

```

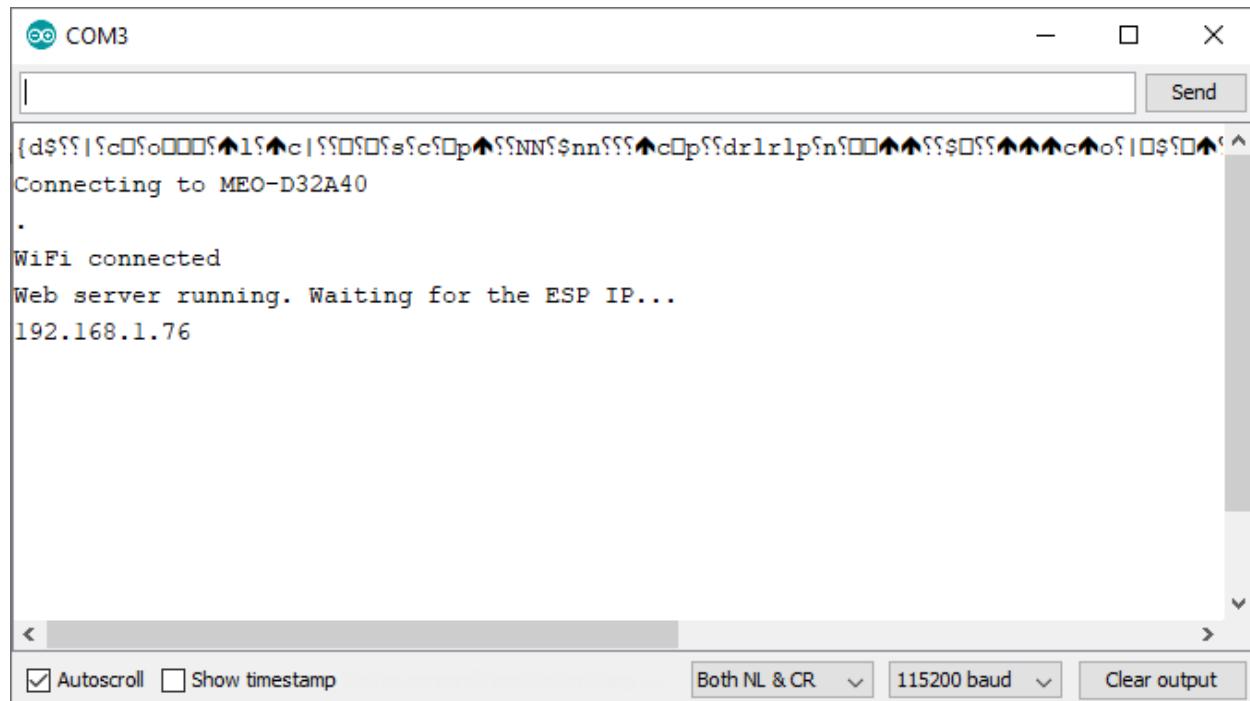
```
client.println("<body><h1>ESP8266 - Temperature</h1><h3>Temperature in Celsius: ");
client.println(temperatureCString);
client.println("*C</h3><h3>Temperature in Fahrenheit: ");
client.println(temperatureFString);
client.println("*F</h3></body></html>");
```

The temperature readings are sent in the `temperatureCString` and `temperatureFString` variables.

You can use this web server template to display any other variables like readings from other sensors or GPIO state.

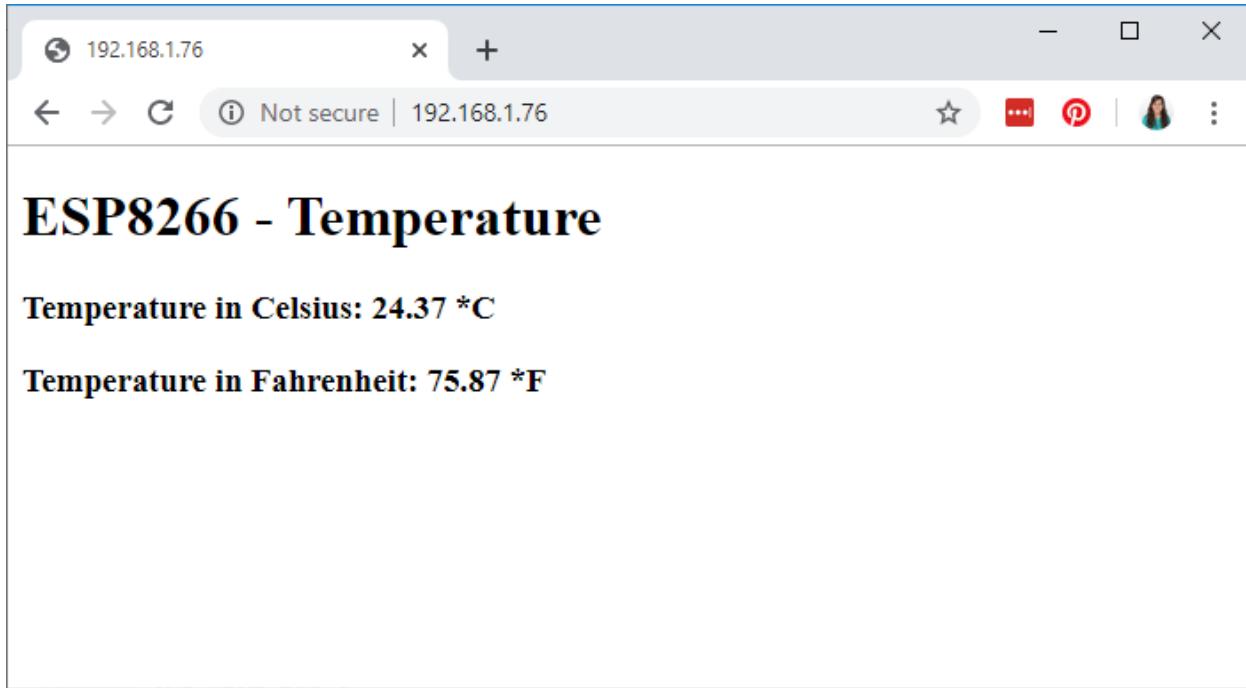
ESP8266 IP Address

Open the Arduino IDE serial monitor at a baud rate of 115200. Reset the ESP8266 and after a few seconds the ESP8266 IP address should appear. In our case, it's, 192.168.1.76.



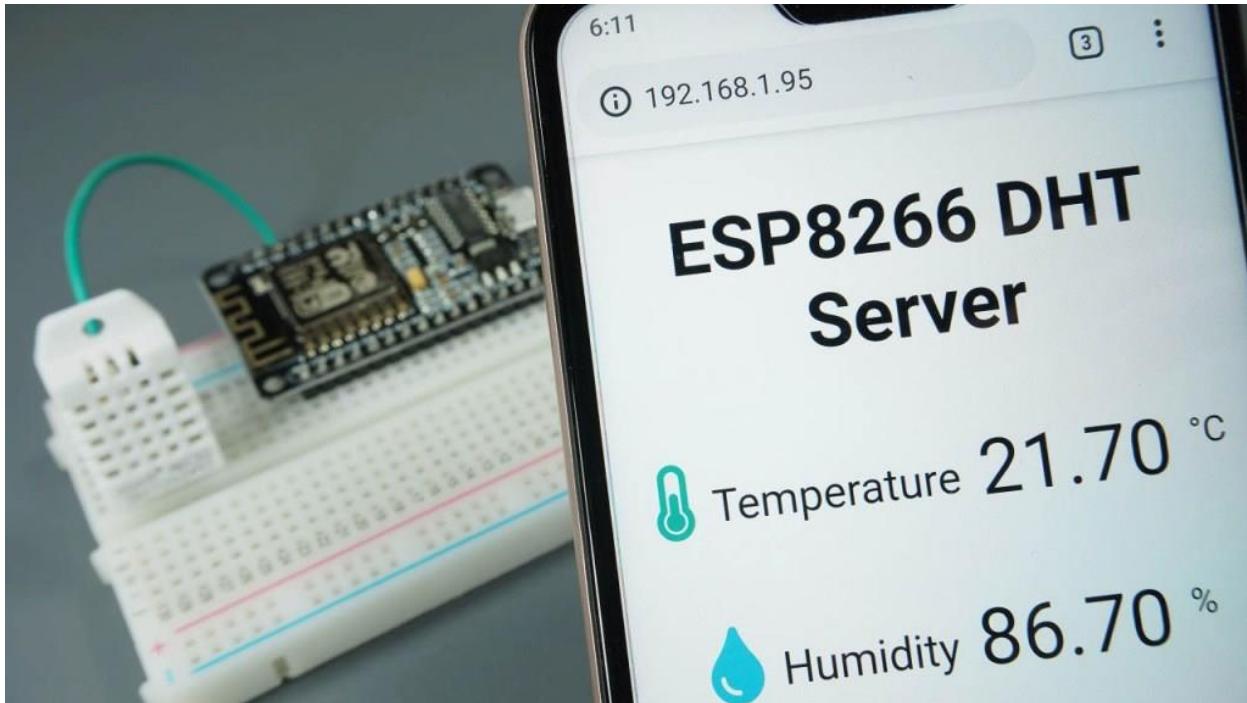
Demonstration

For the final demonstration open any browser from a device that is connected to the same router that your ESP8266.



Now, you can see temperature in Celsius and Fahrenheit in your web server. To see the latest readings simply refresh the web page.

Unit 7: DHT11/DHT22 Asynchronous Web Server



In this project, you'll learn how to build an asynchronous web server to display temperature and humidity readings from a DHT22 or DHT11 temperature sensor. The web server updates automatically without the need to refresh the web page.

The web server we'll build here can be easily adapted to work with other sensors or for other purposes.

Asynchronous Web Server

To build the web server we'll use the [ESPAsyncWebServer library](#) that provides an easy way to build an asynchronous web server. Building an asynchronous web server has several advantages as mentioned in the library GitHub page, such as:

- “Handle more than one connection at the same time”;
- “When you send the response, you are immediately ready to handle other connections while the server is taking care of sending the response in the background”;
- “Simple template processing engine to handle templates”;
- And much more...

Learn more about this library by reading the [documentation on its GitHub page](#).

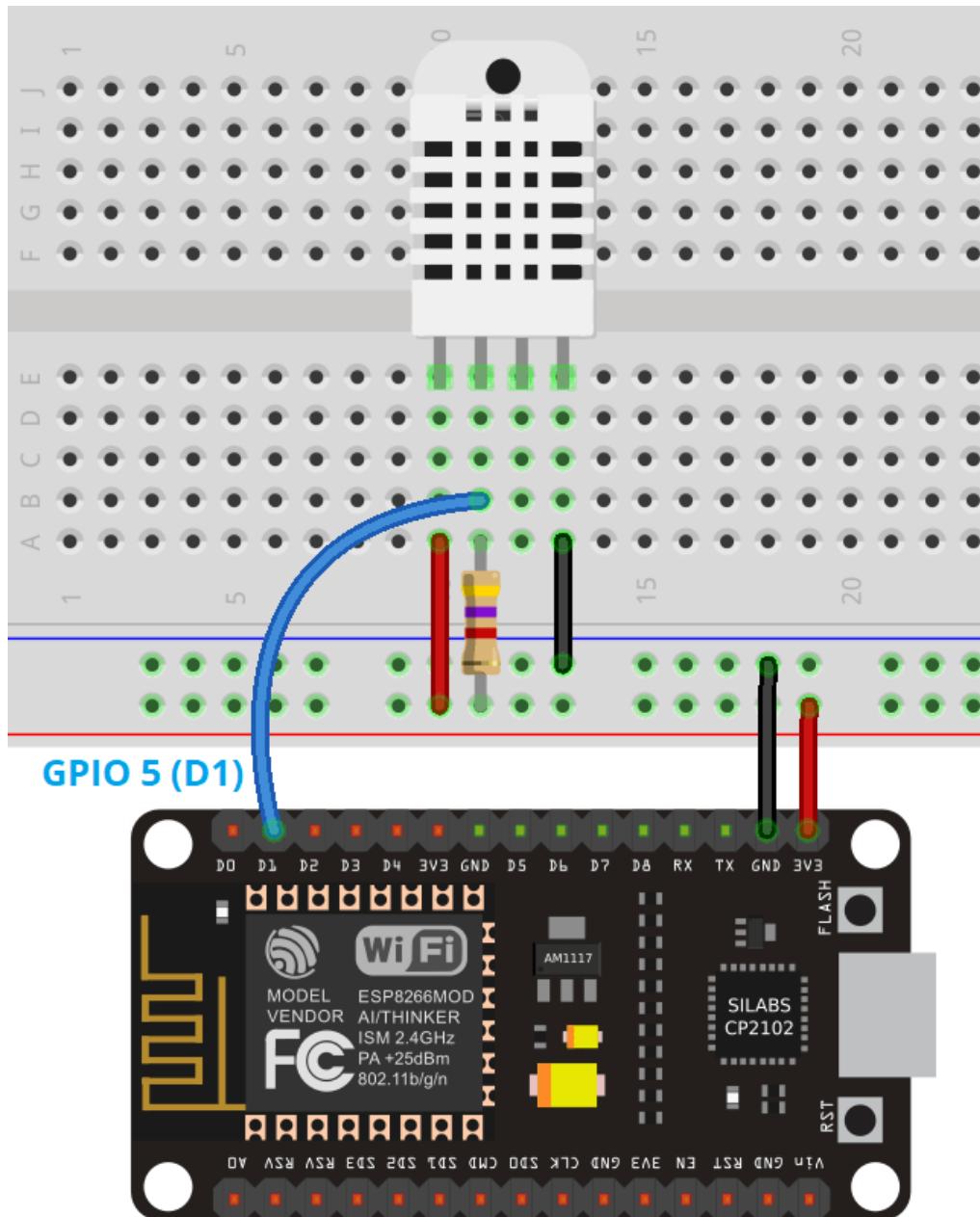
Parts Required

To complete this tutorial, you need the following parts:

- [ESP8266](#)
- [DHT22](#) or [DHT11](#) Temperature and Humidity Sensor
- [4.7k Ohm Resistor](#)
- [Breadboard](#)
- [Jumper wires](#)

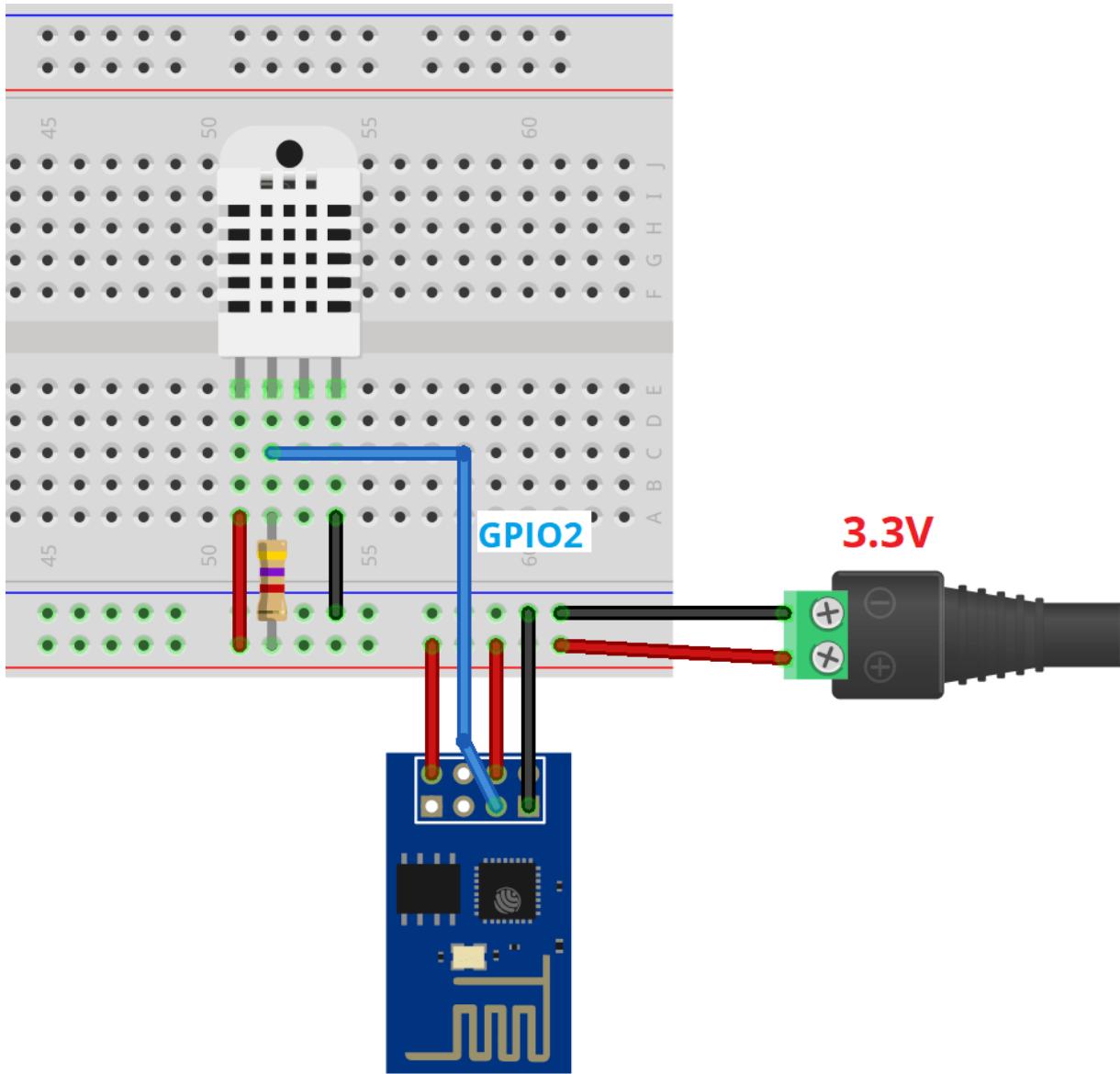
Schematic Diagram

Before proceeding with the tutorial, wire the DHT11 or DHT22 temperature and humidity sensor to the ESP8266 as shown in the following schematic diagram.



In this example, we're wiring the DHT data pin to GPIO 5 (D1), but you can use any other suitable GPIO.

If you're using an ESP-01, GPIO 2 is the most suitable pin to connect to the DHT data pin, as shown in the next diagram.

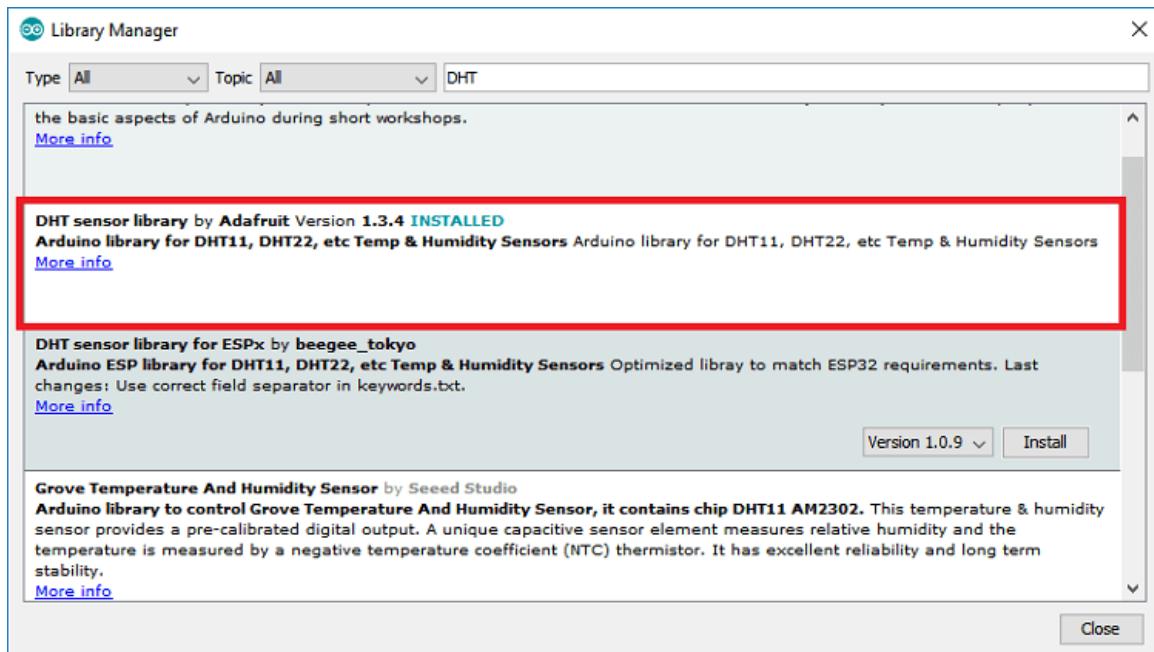


Installing DHT Libraries

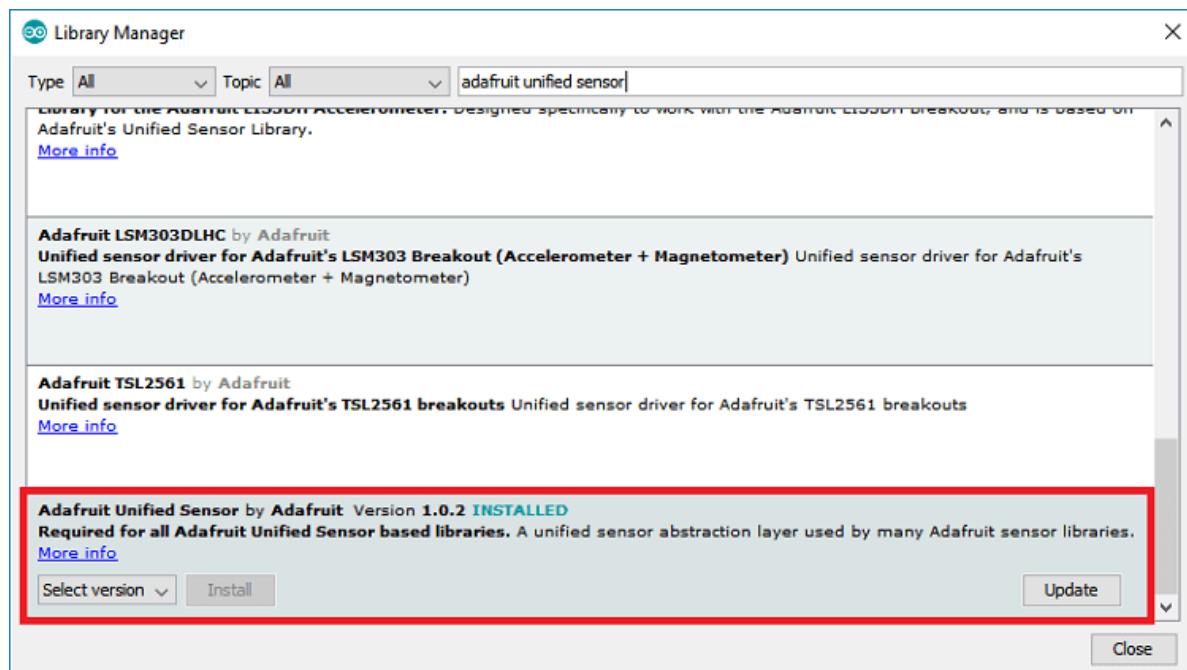
If you've followed the previous Units, you should have already installed the required libraries to work with the DHT sensor.

Follow the next steps to install the required libraries to work with the DHT sensor, if you haven't already.

1. Open your Arduino IDE and go to **Sketch > Include Library > Manage Libraries**. The Library Manager should open;
2. Search for “**DHT**” on the Search box and install the DHT library from Adafruit;



3. 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;



4. After installing the libraries, restart your Arduino IDE.

Installing the ESPAsyncWebServer library

To build the web server we'll use the [ESPAsyncWebServer](#) library that provides an easy way to build an asynchronous web server.

The ESPAsyncWebServer library is not available to install in the Arduino IDE Library Manager. So, you need to install it manually. Follow the next steps to install the ESPAsyncWebServer library:

1. [Click here to download the ESPAsyncWebServer library](#). You should have a .zip folder in your Downloads folder
2. Unzip the .zip folder and you should get *ESPAsyncWebServer-master* folder
3. Rename your folder from *ESPAsyncWebServer-master* to *ESPAsyncWebServer*
4. Move the ESPAsyncWebServer folder to your Arduino IDE installation libraries folder

The ESPAsyncWebServer library requires the [ESPAsyncTCP](#) library to work. Follow the next steps to install that library:

1. [Click here to download the ESPAsyncTCP library](#). You should have a .zip folder in your Downloads folder
2. Unzip the .zip folder and you should get *ESPAsyncTCP-master* folder
3. Rename your folder from *ESPAsyncTCP-master* to *ESPAsyncTCP*
4. Move the ESPAsyncTCP folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Alternatively, to install these libraries in your Arduino IDE, you can go to **Sketch ▶ Include Library ▶ Add .ZIP library** and select the libraries you've just downloaded.

Code

Open your Arduino IDE and copy the following code.

```
// Import required libraries
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <Hash.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>

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

#define DHTPIN 5      // Digital pin connected to the DHT sensor

// Uncomment the type of sensor in use:
// #define DHTTYPE DHT11    // DHT 11
#define DHTTYPE DHT22    // DHT 22 (AM2302)
// #define DHTTYPE DHT21    // DHT 21 (AM2301)

DHT dht(DHTPIN, DHTTYPE);

// current temperature & humidity, updated in loop()
float t = 0.0;
float h = 0.0;

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

// Generally, you should use "unsigned long" for variables that hold time
// The value will quickly become too large for an int to store
unsigned long previousMillis = 0;      // will store last time DHT was updated

// Updates DHT readings every 10 seconds
const long interval = 10000;

const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css" integrity="sha384-fmqOCqbTlWI1j8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHPvLbHG9Sr" crossorigin="anonymous">
  <style>
    html {
      font-family: Arial;
      display: inline-block;
      margin: 0px auto;
      text-align: center;
    }
    h2 { font-size: 3.0rem; }
```

```

        p { font-size: 3.0rem; }
        .units { font-size: 1.2rem; }
        .dht-labels{
            font-size: 1.5rem;
            vertical-align:middle;
            padding-bottom: 15px;
        }
    </style>
</head>
<body>
    <h2>ESP8266 DHT Server</h2>
    <p>
        <i class="fas fa-thermometer-half" style="color:#059e8a;"></i>
        <span class="dht-labels">Temperature</span>
        <span id="temperature">%TEMPERATURE%</span>
        <sup class="units">C</sup>
    </p>
    <p>
        <i class="fas fa-tint" style="color:#00add6;"></i>
        <span class="dht-labels">Humidity</span>
        <span id="humidity">%HUMIDITY%</span>
        <sup class="units">%</sup>
    </p>
</body>
<script>
setInterval(function () {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("temperature").innerHTML = this.responseText;
        }
    };
    xhttp.open("GET", "/temperature", true);
    xhttp.send();
}, 10000 ) ;

setInterval(function () {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("humidity").innerHTML = this.responseText;
        }
    };
    xhttp.open("GET", "/humidity", true);
    xhttp.send();
}, 10000 ) ;
</script>
</html>) rawliteral";

// Replaces placeholder with DHT values
String processor(const String& var){
    //Serial.println(var);
    if(var == "TEMPERATURE"){
        return String(t);
    }
    else if(var == "HUMIDITY"){
        return String(h);
    }
    return String();
}

```

```

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

    // Connect to Wi-Fi
    WiFi.begin(ssid, password);
    Serial.println("Connecting to WiFi");
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println(".");
    }

    // Print ESP8266 Local IP Address
    Serial.println(WiFi.localIP());
}

// Route for root / web page
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/html", index_html, processor);
});
server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", String(t).c_str());
});
server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", String(h).c_str());
});

// Start server
server.begin();
}

void loop(){
    unsigned long currentMillis = millis();
    if (currentMillis - previousMillis >= interval) {
        // save the last time you updated the DHT values
        previousMillis = currentMillis;
        // Read temperature as Celsius (the default)
        float newT = dht.readTemperature();
        // Read temperature as Fahrenheit (isFahrenheit = true)
        //float newT = dht.readTemperature(true);
        // if temperature read failed, don't change t value
        if (isnan(newT)) {
            Serial.println("Failed to read from DHT sensor!");
        }
        else {
            t = newT;
            Serial.println(t);
        }
        // Read Humidity
        float newH = dht.readHumidity();
        // if humidity read failed, don't change h value
        if (isnan(newH)) {
            Serial.println("Failed to read from DHT sensor!");
        }
        else {
            h = newH;
            Serial.println(h);
        }
    }
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module4/Unit7/DHT_Asynchronous_Web_Server/DHT_Asynchronous_Web_Server.ino

Insert your network credentials in the following variables and the code will work straight away.

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

How the Code Works

In the following paragraphs we'll explain how the code works. Keep reading if you want to learn more or jump to the Demonstration section to see the final result.

Importing libraries

First, import the required libraries.

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <Hash.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
```

Setting your network credentials

Insert your network credentials in the following variables, so that the ESP8266 can connect to your local network.

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

Variables definition

Define the GPIO that the DHT data pin is connected to. In this case, it's connected to GPIO 5 (D1).

```
#define DHTPIN 5      // Digital pin connected to the DHT sensor
```

Then, select the DHT sensor type you're using. In this example, we're using the DHT22. If you're using another type, you just need to uncomment your sensor and comment all the others.

```
#define DHTTYPE DHT22      // DHT 22 (AM2302)
```

Instantiate a `DHT` object with the type and pin defined earlier.

```
DHT dht(DHTPIN, DHTTYPE);
```

Create an `AsyncWebServer` object on port 80.

```
AsyncWebServer server(80);
```

Create float variables to hold the current temperature and humidity values. The temperature and humidity are updated in the `loop()`.

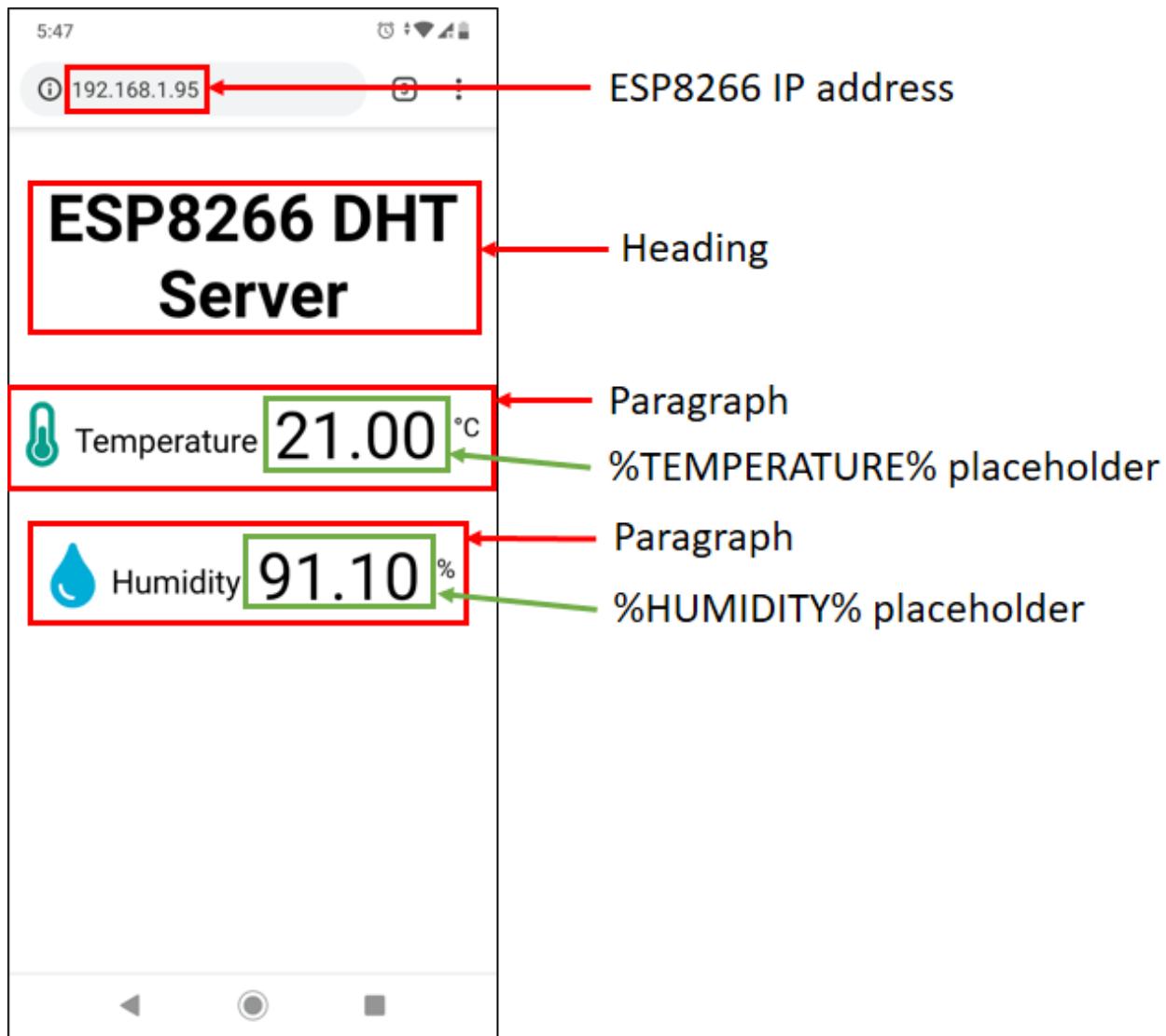
```
float t = 0.0;
float h = 0.0;
```

Create timer variables needed to update the temperature readings every 10 seconds.

```
unsigned long previousMillis = 0;    // will store last time DHT was updated
// Updates DHT readings every 10 seconds
const long interval = 10000;
```

Building the Web Page

Proceeding to the web server page.



As you can see in the above figure, the web page shows one heading and two paragraphs. There is a paragraph to display the temperature and another to display the humidity. There are also two icons to style the page.

Let's see how this web page is created.

All the HTML text with styles included is stored in the `index_html` variable. Now we'll go through the HTML text and see what each part does.

The following `<meta>` tag makes your web page responsive in any browser.

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

The [<link>](#) tag is needed to load the icons from the fontawesome website.

```
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css" integrity="sha384-fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMhpvLbHG9Sr" crossorigin="anonymous">
```

Styles

Between the [<style></style>](#) tags, we add some CSS to style the web page.

```
<style>
  html {
    font-family: Arial;
    display: inline-block;
    margin: 0px auto;
    text-align: center;
  }
  h2 { font-size: 3.0rem; }
  p { font-size: 3.0rem; }
  .units { font-size: 1.2rem; }
  .dht-labels{
    font-size: 1.5rem;
    vertical-align:middle;
    padding-bottom: 15px;
  }
</style>
```

Basically, we're setting the HTML page to display the text with Arial font in block without margin, and aligned at the center.

We set the font size for the heading (h2), paragraph (p) and the units (.units) of the readings.

The labels for the readings are styled as shown below:

```
.dht-labels{
  font-size: 1.5rem;
  vertical-align:middle;
  padding-bottom: 15px;
}
```

All of the previous tags should go between the [<head>](#) and [</head>](#) tags. These tags are used to include content that is not directly visible to the user, like the [<meta>](#) , the [<link>](#) tags, and the styles.

HTML Body

Inside the `<body></body>` tags is where we add the web page content.

The `<h2></h2>` tags add a heading to the web page. In this case, the “ESP8266 DHT server” text, but you can add any other text.

```
<h2>ESP8266 DHT Server</h2>
```

Then, there are two paragraphs. One to display the temperature and the other to display the humidity. The paragraphs are delimited by the `<p>` and `</p>` tags. The paragraph for the temperature is the following:

```
<p>
  <i class="fas fa-thermometer-half" style="color:#059e8a;"></i>
  <span class="dht-labels">Temperature</span>
  <span id="temperature">%TEMPERATURE%</span>
  <sup class="units">&deg;C</sup>
</p>
```

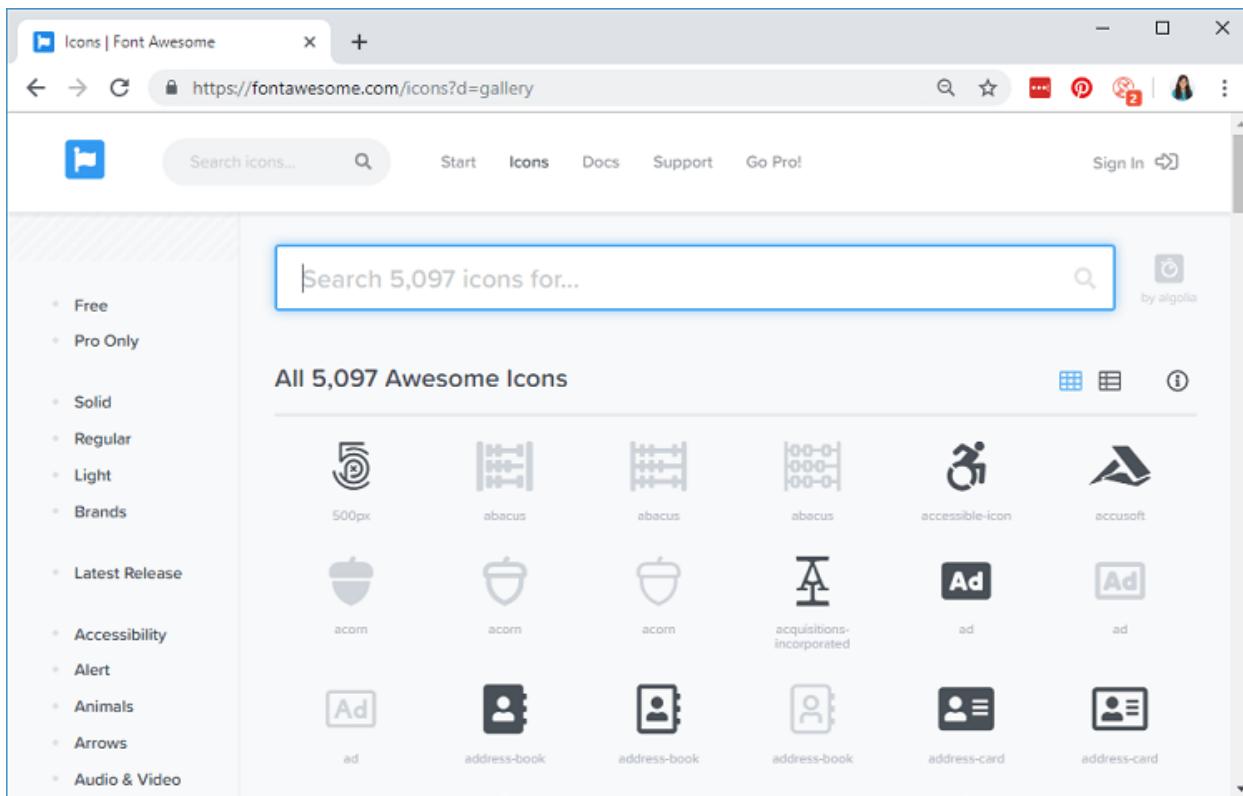
And the paragraph for the humidity is on the following snippet:

```
<p>
  <i class="fas fa-tint" style="color:#00add6;"></i>
  <span class="dht-labels">Humidity</span>
  <span id="humidity">%HUMIDITY%</span>
  <sup class="units">%</sup>
</p>
```

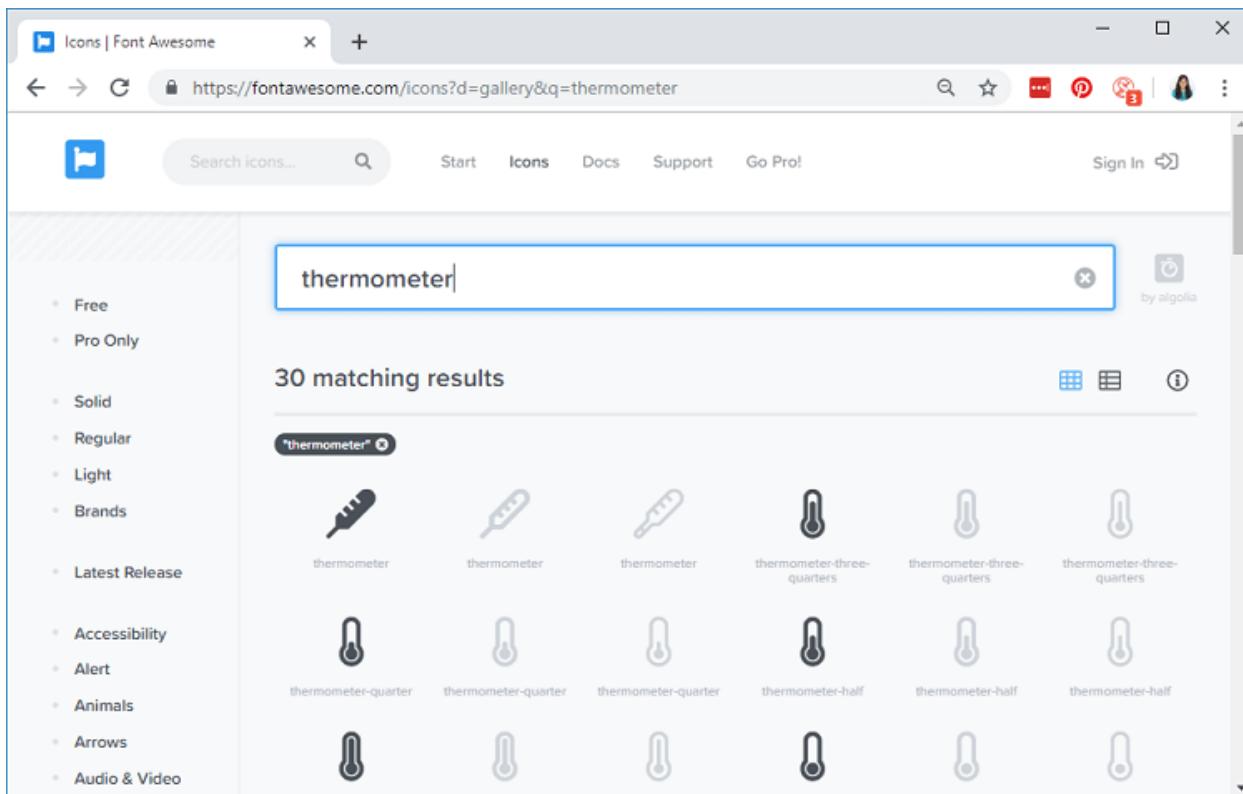
The `<i>` tags display the fontawesome icons.

How to display icons

To choose the icons, go to the [Font Awesome Icons website](#).

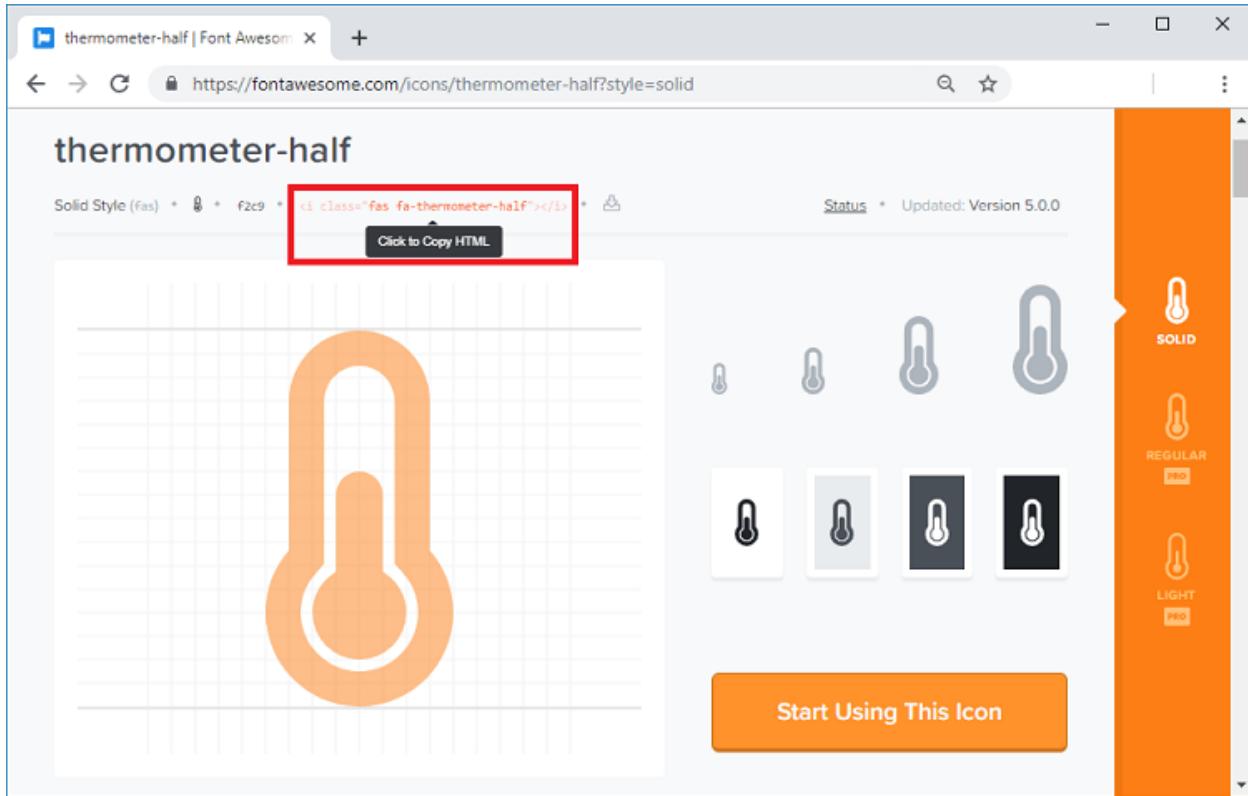


Search the icon you're looking for. For example, "thermometer".



Click the desired icon. Then, you just need to copy the HTML text provided.

```
<i class="fas fa-thermometer-half"></i>
```



To choose the color, you just need to pass the style parameter with the color in hexadecimal, as follows:

```
<i class="fas fa-tint" style="color:#00add6;"></i>
```

Proceeding with the HTML text...

The next line writes the word “Temperature” into the web page.

```
<span class="dht-labels">Humidity</span>
```

The TEMPERATURE text between % signs is a placeholder for the temperature value.

```
<span id="temperature">%TEMPERATURE%</span>
```

This means that this **%TEMPERATURE%** text is like a variable that will be replaced by the actual temperature value from the DHT sensor. The placeholders on the HTML text should go between % signs.

Finally, we add the degree symbol.

```
<sup class="units">&deg;C</sup>
```

The `` tags make the text superscript.

We use the same approach for the humidity paragraph, but it uses a different icon and the **%HUMIDITY%** placeholder.

Automatic Updates

Finally, there's some JavaScript code in our web page that updates the temperature and humidity automatically, every 10 seconds.

Scripts in HTML text should go between the `<script></script>` tags.

```
<script>
setInterval(function () {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("temperature").innerHTML =
this.responseText;
        }
    };
    xhttp.open("GET", "/temperature", true);
    xhttp.send();
}, 10000 ) ;

setInterval(function () {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("humidity").innerHTML =
this.responseText;
        }
    };
    xhttp.open("GET", "/humidity", true);
    xhttp.send();
}, 10000 ) ;
</script>
```

To update the temperature on the background, we have a `setInterval()` function that runs every 10 seconds.

Basically, it makes a request in the `/temperature` URL to get the latest temperature reading.

```
xhttp.open("GET", "/humidity", true);
xhttp.send();
}, 10000 )
```

When it receives that value, it updates the HTML element whose id is temperature.

```
if (this.readyState == 4 && this.status == 200) {
    document.getElementById("humidity").innerHTML = this.responseText;
}
```

In summary, this previous section is responsible for updating the temperature asynchronously. The same process is repeated for the humidity readings.

Processor

Now, we need to create the `processor()` function, that will replace the placeholders in our HTML text with the actual temperature and humidity values.

```
String processor(const String& var) {
    //Serial.println(var);
    if(var == "TEMPERATURE") {
        return readDHTTemperature();
    }
    else if(var == "HUMIDITY") {
        return readDHTHumidity();
    }
    return String();
}
```

When the web page is requested, we check if the HTML has any placeholders. If it finds the `%TEMPERATURE%` placeholder, we return the latest temperature reading.

```
if(var == "TEMPERATURE") {
    return String(t);
}
```

If the placeholder is **%HUMIDITY%**, we return the humidity value.

```
else if(var == "HUMIDITY") {
    return String(h);
}
```

setup()

In the `setup()`, initialize the Serial Monitor for debugging purposes.

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

Initialize the DHT sensor.

```
dht.begin();
```

Connect to your local network and print the ESP8266 IP address.

```
WiFi.begin(ssid, password);
Serial.println("Connecting to WiFi");
while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println(".");
}
// Print ESP8266 Local IP Address
Serial.println(WiFi.localIP());
```

Finally, add the next lines of code to handle the web server.

```
// Route for root / web page
server.on("/", HTTP_GET, [] (AsyncWebServerRequest *request) {
    request->send_P(200, "text/html", index_html, processor);
});
server.on("/temperature", HTTP_GET, [] (AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", String(t).c_str());
});
server.on("/humidity", HTTP_GET, [] (AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", String(h).c_str());
});
```

When we make a request on the root URL, we send the HTML text that is stored on the `index_html` variable. We also need to pass the `processor` function, that will replace all the placeholders with the right values.

```
server.on("/", HTTP_GET, [] (AsyncWebServerRequest *request) {
    request->send_P(200, "text/html", index_html, processor);
});
```

We need to add two additional handlers to update the temperature and humidity readings. When we receive a request on the `/temperature` URL, we simply need to send the updated temperature value. It is plain text, and it should be sent as a char, so, we use the `c_str()` method.

```
server.on("/temperature", HTTP_GET, [] (AsyncWebRequest *request) {
    request->send_P(200, "text/plain", String(t).c_str());
});
```

The same process is repeated for the humidity.

```
server.on("/humidity", HTTP_GET, [] (AsyncWebRequest *request) {
    request->send_P(200, "text/plain", String(h).c_str());
});
```

Lastly, we can start the server.

```
server.begin();
```

In the `loop()` is where we get new temperature readings from the sensor every 10 seconds.

Basically, we check if it is time to get new sensor readings:

```
if (currentMillis - previousMillis >= interval) {
```

If it is, we store a new temperature reading on the `newT` variable

```
float newT = dht.readTemperature();
```

If the `newT` variable is a valid temperature reading, we update the `t` variable.

```
else {
    t = newT;
    Serial.println(t);
}
```

The same process is repeated for the humidity.

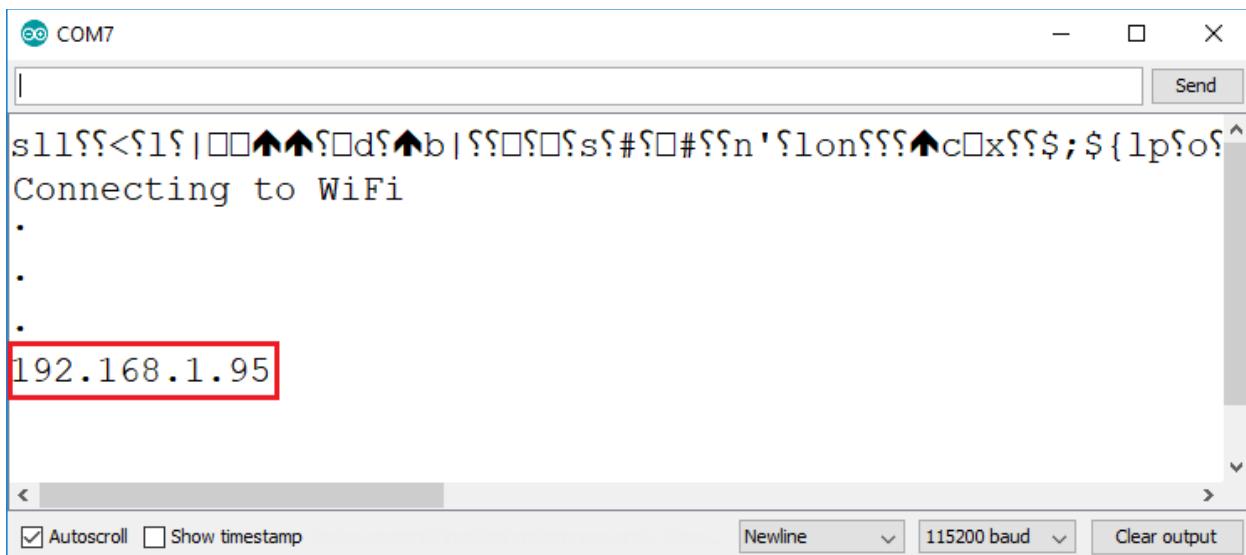
```
// Read Humidity
float newH = dht.readHumidity();
```

```
// if humidity read failed, don't change h value
if (isnan(newH)) {
    Serial.println("Failed to read from DHT sensor!");
}
else {
    h = newH;
    Serial.println(h);
}
```

That's pretty much how the code works.

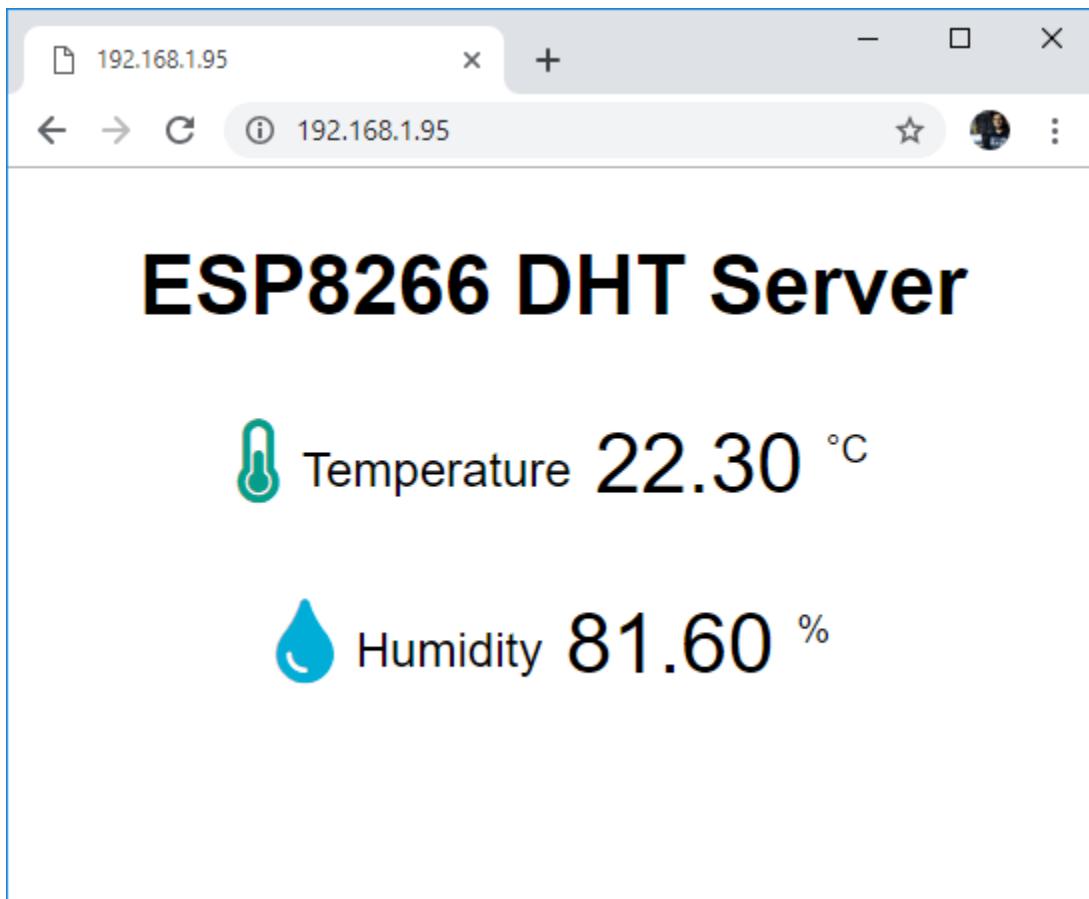
ESP8266 IP Address

After uploading the code, open the Serial Monitor at a baud rate of 115200. Press the ESP8266 reset button. The ESP8266 IP address will be printed in the serial monitor as shown in the following figure.



Demonstration

In your local network, go to a browser and type the ESP8266 IP address. It should display the following web page with the latest sensor readings.



The temperature and humidity readings update automatically every 10 seconds without the need to refresh the web page.

Unit 8: RGB LED Strip with Color Picker Web Server



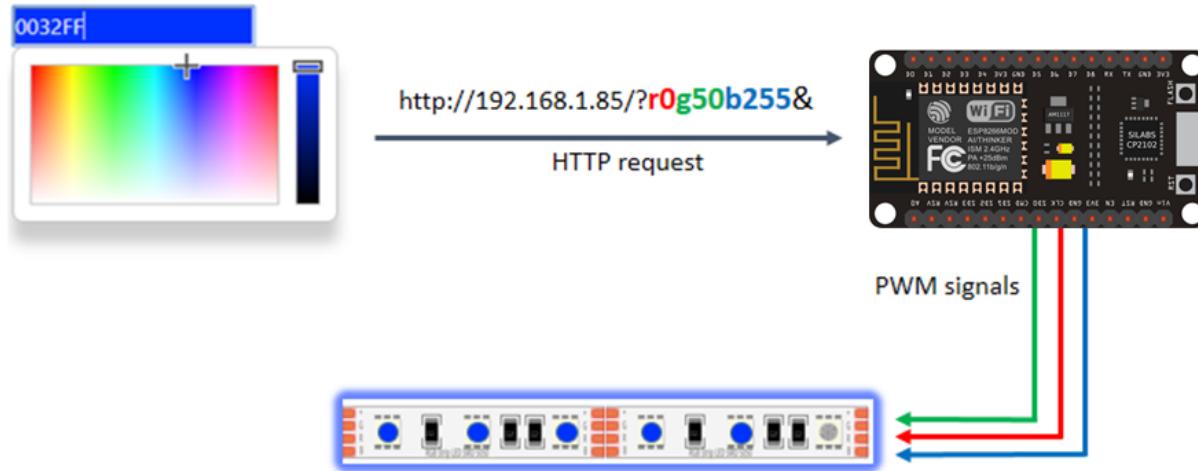
In this project we'll show you how to remotely control a 5V RGB LED strip with an ESP8266 board using a web server with a color picker.

To better understand this project, there are a few tutorials that you may want to take a look first (this step is optional):

- [How RGB LEDs work](#)
- [ESP8266 Web Server - Control Outputs](#)

Project Overview

Before getting started, let's take a quick look on how this project works:



The ESP8266 web server displays a color picker.

- When you chose a color, your browser makes a request on a URL that contains the R, G, and B parameters of the selected color.
- Your ESP8266 receives the request and splits the value for each color parameter.
- Then, it sends a PWM signal with the corresponding value to the GPIOs that are controlling the strip.

Parts Required

For this project you need the following parts:

- [ESP8266](#)
- [RGB LED Strip \(5V\)](#)
- 3x NPN transistors (see how to choose the proper transistor in the project description)
- 3x [1k ohm resistors](#)

- [Jumper wires](#)
- [Breadboard](#)

Schematic

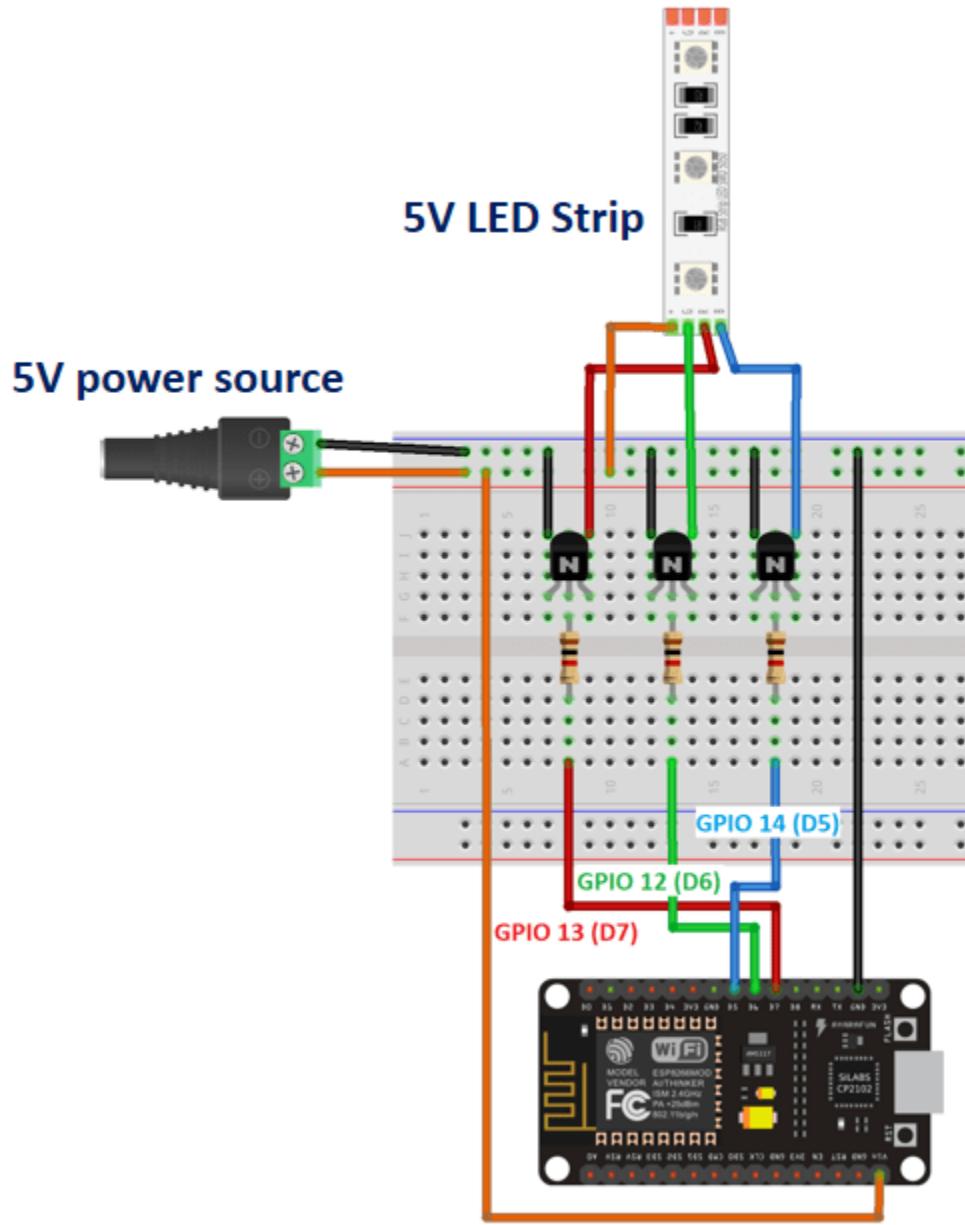
For this project, we'll be using an RGB LED strip that can be powered with 5V.



Note: there are other similar strips that require 12V to operate. You can still use them with the code provided, but you need a suitable power supply.

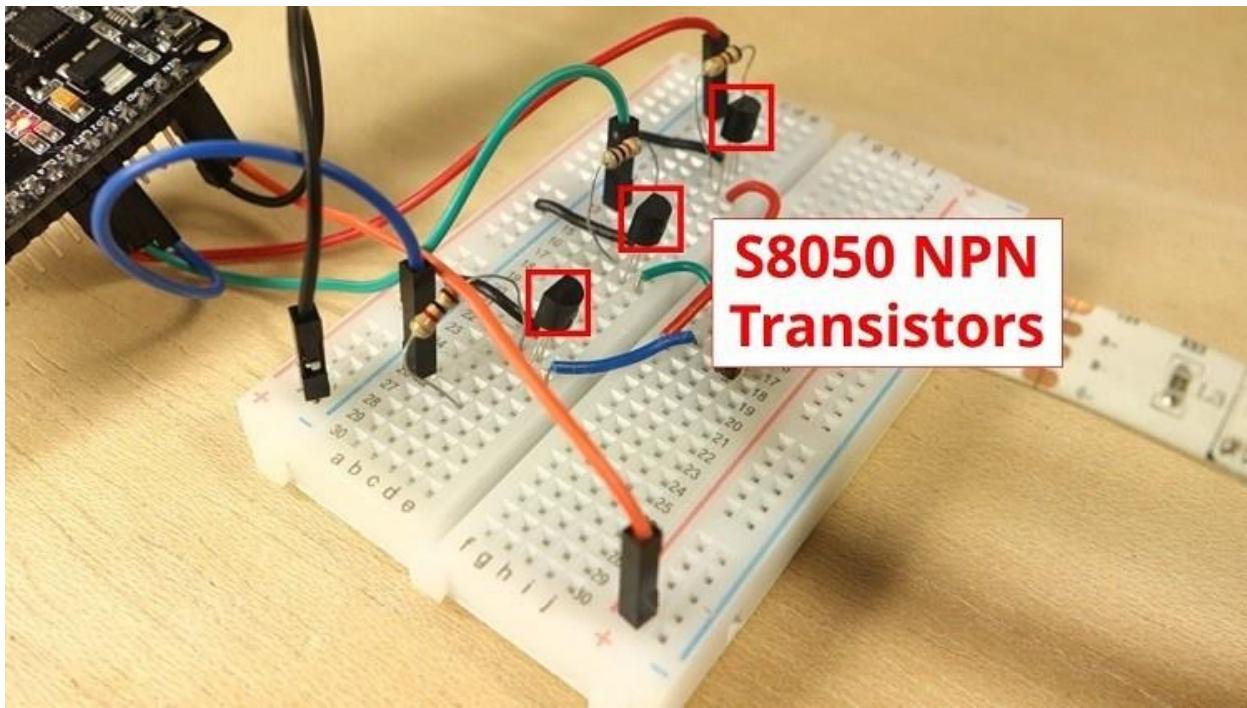
In this example, we'll be powering the LED strip and the ESP8266 using the same 5V power supply.

Follow the next schematic diagram to connect the strip to your ESP8266.



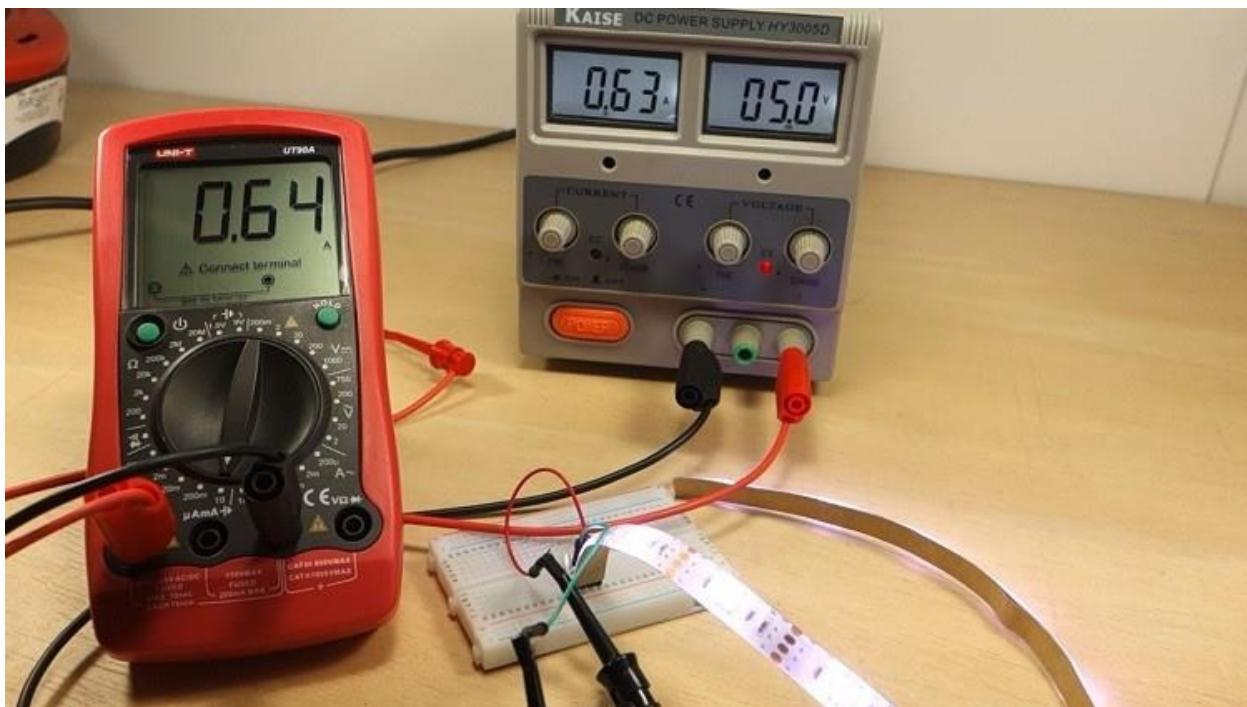
NPN Transistors

In this circuit, we're using the S8050 NPN transistor. However, depending on how many LEDs you have in your strip, you might need to use an NPN transistor that can handle more continuous current in the collector pin.



To determine the max current used by your LED strip, you can measure the current consumption when all the LEDs are at maximum brightness (when the color is white).

Since we're using 12 LEDs, the maximum current is approximately 630 mA at full brightness in white color. So, we can use the S8050 NPN transistor that can handle up to 700mA.



Note: your strip consumes the maximum current when you set white color (this is the same as setting all three colors to the maximum brightness). Setting other colors draws less current, so you'll probably won't have your strip using the maximum current.

Code

After assembling the circuit, copy the following code to your Arduino IDE to program the ESP8266.

```
#include <ESP8266WiFi.h>

const char* ssid      = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

WiFiServer server(80);

// Decode HTTP GET value
String redString = "0";
String greenString = "0";
String blueString = "0";
int pos1 = 0;
int pos2 = 0;
int pos3 = 0;
int pos4 = 0;

String header;

// Red, green, and blue pins for PWM control
const int redPin = 13;      // 13 corresponds to GPIO13
const int greenPin = 12;    // 12 corresponds to GPIO12
const int bluePin = 14;     // 14 corresponds to GPIO14

// Setting PWM bit resolution
const int resolution = 256;

// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in milliseconds (example: 2000ms = 2s)
const long timeoutTime = 2000;

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

  // Configure LED PWM resolution/range and set pins to LOW
  analogWriteRange(resolution);
  analogWrite(redPin, 0);
  analogWrite(greenPin, 0);
  analogWrite(bluePin, 0);
```

```

// 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();

    if (client) {
        currentTime = millis();
        previousTime = currentTime;
        Serial.println("New Client.");
        String currentLine = "";
        while (client.connected() && currentTime - previousTime <= timeoutTime)
{
            currentTime = millis();
            if (client.available()) {
                char c = client.read();
                Serial.write(c);
                header += c;
                if (c == '\n') {
                    if (currentLine.length() == 0) {
                        client.println("HTTP/1.1 200 OK");
                        client.println("Content-type:text/html");
                        client.println("Connection: close");
                        client.println();
                    }

                    // 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:,\"");
                    client.println("<link rel=\"stylesheet\"");
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css\"");
                     client.println("<script");
src="https://cdnjs.cloudflare.com/ajax/libs/jscolor/2.0.4/jscolor.min.js\"");
</script>");
                    client.println("</head><body><div class=\"container\"");
class="row"><h1>ESP Color Picker</h1></div>");
                    client.println("<a class=\"btn btn-primary btn-lg\" href=\"#\"");
id="change_color\" role=\"button\">Change Color</a> ");
                    client.println("<input class=\"jscolor
{onFineChange:'update(this)'}\" id=\"rgb\"></div>\"");
                    client.println("<script>function update(picker)
{document.getElementById('rgb').innerHTML = Math.round(picker.rgb[0]) + ', ' +
Math.round(picker.rgb[1]) + ', ' + Math.round(picker.rgb[2]);}");
                    client.println("document.getElementById(\"change_color\").href=\"?
r\" + Math.round(picker.rgb[0]) + \"g\" + Math.round(picker.rgb[1]) + \"b\" +
Math.round(picker.rgb[2]) + \"&\";}</script></body></html>\"");
}
}
}

```

```

// The HTTP response ends with another blank line
client.println();

// Request sample: /?r201g32b255&
// Red = 201 | Green = 32 | Blue = 255
if(header.indexOf("GET /?r") >= 0) {
    pos1 = header.indexOf('r');
    pos2 = header.indexOf('g');
    pos3 = header.indexOf('b');
    pos4 = header.indexOf('&');
    redString = header.substring(pos1+1, pos2);
    greenString = header.substring(pos2+1, pos3);
    blueString = header.substring(pos3+1, pos4);
    /*Serial.println(redString.toInt());
    Serial.println(greenString.toInt());
    Serial.println(blueString.toInt());*/
    analogWrite(redPin, redString.toInt());
    analogWrite(greenPin, greenString.toInt());
    analogWrite(bluePin, blueString.toInt());
}
break;
} else {
    currentLine = "";
}
} else if (c != '\r') {
    currentLine += c;
}
}
header = "";
client.stop();
Serial.println("Client disconnected.");
Serial.println("");
}
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module4/Unit8/ESP8266_RGB_LED_Strip/ESP8266_RGB_LED_Strip.ino

Before uploading the code, don't forget to insert your network credentials so that the ESP8266 can connect to your local network.

```

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

```

The code for this project is very similar with previous web servers in this ebook. It adds the color picker to the web page and decodes the request to control the strip color. So, we'll just take a look at the relevant parts for this project.

How the code works

The following lines define string variables to hold the R, G, and B parameters from the request.

```
String redString = "0";
String greenString = "0";
String blueString = "0";
```

The next four variables are used to decode the HTTP request later on.

```
int pos1 = 0;
int pos2 = 0;
int pos3 = 0;
int pos4 = 0;
```

Create three variables for the GPIOs that will control the strip R, G, and B parameters.

In this case we're using GPIO 13, GPIO 12, and GPIO 14.

```
const int redPin = 13;
const int greenPin = 12;
const int bluePin = 14;
```

Set the PWM channel resolution:

```
const int resolution = 256;
```

In the `setup()`, configure the PWM resolution/range and set all the pins to LOW.

```
analogWriteRange(resolution);
analogWrite(redPin, 0);
analogWrite(greenPin, 0);
analogWrite(bluePin, 0);
```

The following code section displays the color picker in your web page and makes a request based on the color you've picked.

```

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:,\">");
client.println("<link rel=\"stylesheet\" href=\"https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css\">");
client.println("<script src=\"https://cdnjs.cloudflare.com/ajax/libs/jscolor/2.0.4/jscolor.min.js\"></script>");
client.println("</head><body><div class=\"container\"><div class=\"row\"><h1>ESP Color Picker</h1></div>");
client.println("<a class=\"btn btn-primary btn-lg\" href=\"#\" id=\"change_color\" role=\"button\">Change Color</a> ");
client.println("<input class=\"jscolor {onFineChange:'update(this)' }\" id=\"rgb\"></div>");
client.println("<script>function update(picker)
{document.getElementById('rgb').innerHTML = Math.round(picker.rgb[0])
+ ', ' + Math.round(picker.rgb[1]) + ', ' +
Math.round(picker.rgb[2]);}
client.println("document.getElementById(\"change_color\").href=\"?r\" + Math.round(picker.rgb[0]) + \"g\" + Math.round(picker.rgb[1]) +
\"b\" + Math.round(picker.rgb[2]) + \"&\";}</script></body></html>");
The HTTP response ends with another blank line
client.println();

```

When you pick a color, you receive a request with the following format.

/?r201g32b255&

So, we need to split this string to get the R, G, and B parameters. The parameters are saved in `redString`, `greenString`, and `blueString` variables and can have values between 0 and 255.

```

pos1 = header.indexOf('r');
pos2 = header.indexOf('g');
pos3 = header.indexOf('b');
pos4 = header.indexOf('&');
redString = header.substring(pos1+1, pos2);
greenString = header.substring(pos2+1, pos3);
blueString = header.substring(pos3+1, pos4);

```

To control the strip with the ESP8266, use the `analogWrite()` function to generate PWM signals with the values decoded from the HTTP request.

```

analogWrite(redPin, redString.toInt());
analogWrite(greenPin, greenString.toInt());

```

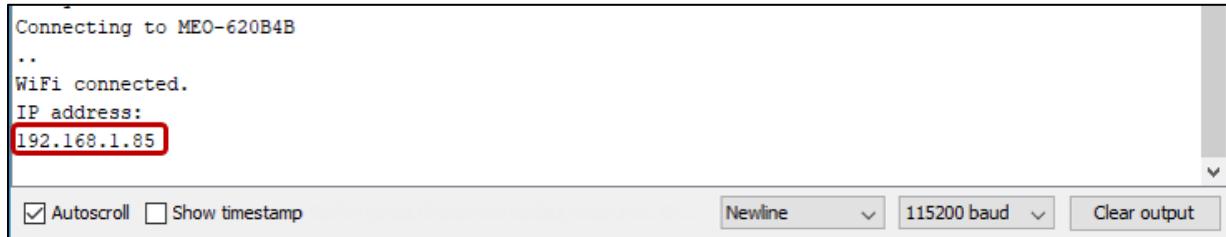
```
analogWrite(bluePin, blueString.toInt());
```

Because we get the values in a string variable, we need to convert them to integers using the `toInt()` method.

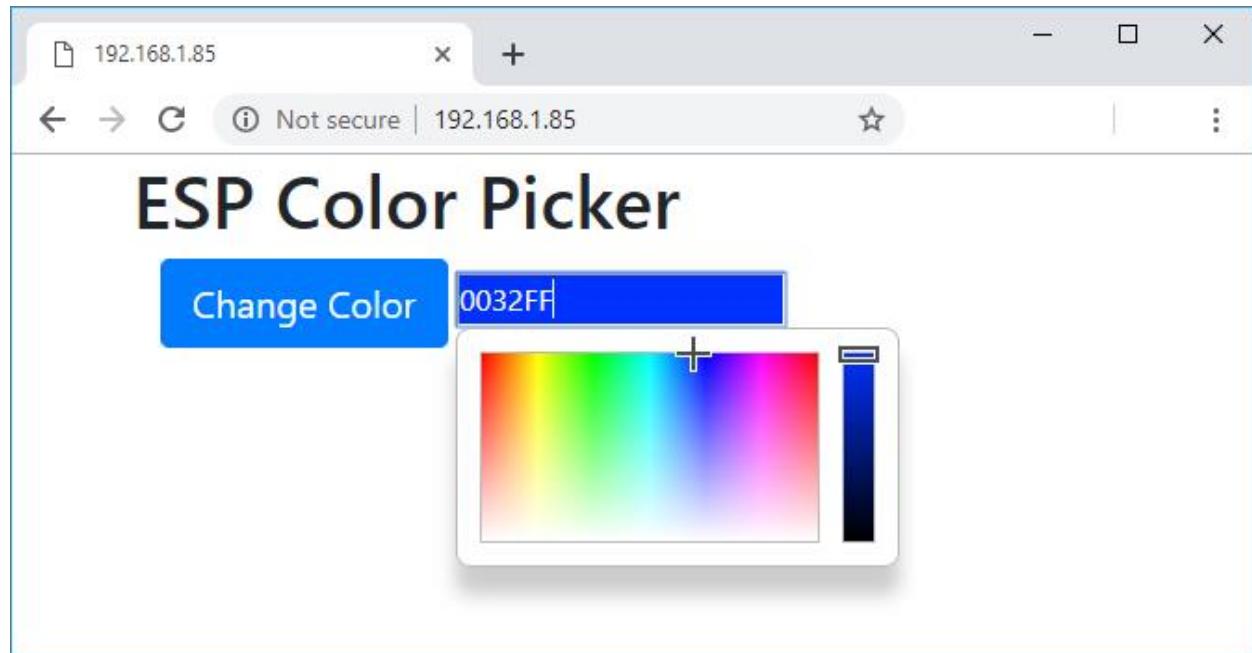
Demonstration

After inserting your network credentials, select the right board and COM port and upload the code to your ESP8266 board.

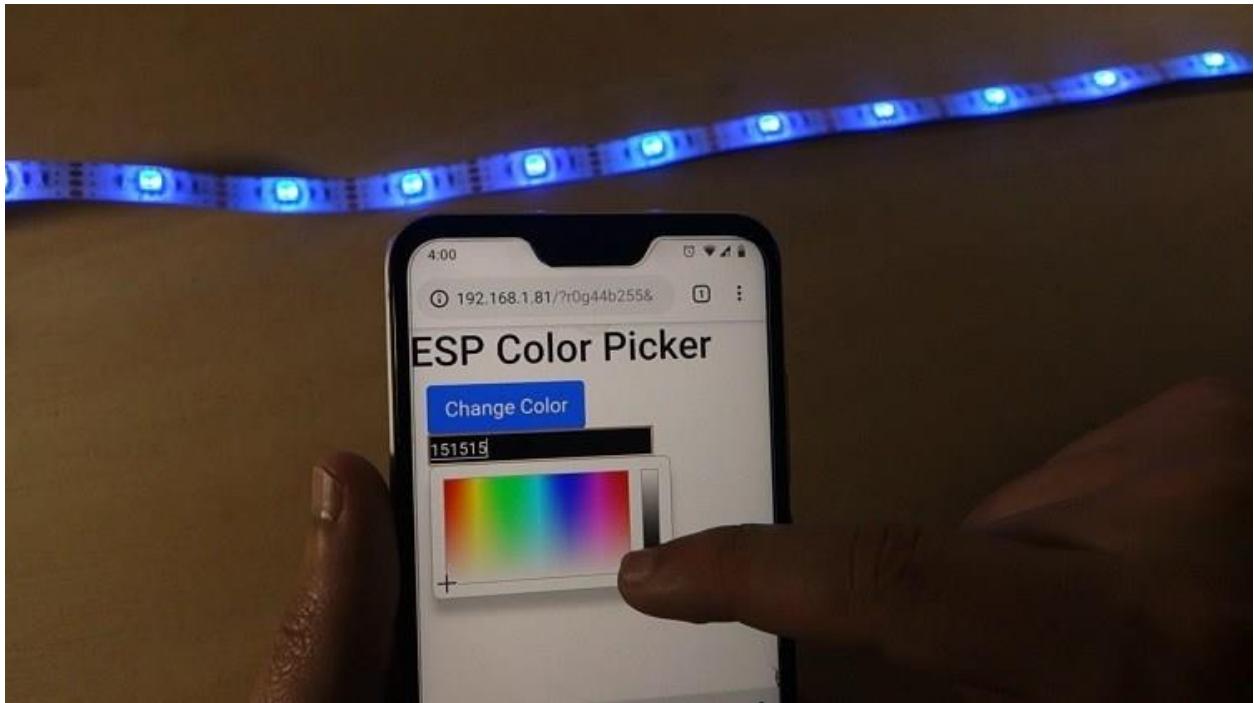
After uploading, open the Serial Monitor at a baud rate of 115200 and press the ESP8266 Enable/Reset button. You should get the board IP address.



Open your browser and insert the ESP8266 IP address. Now, use the color picker to set a color for the strip.

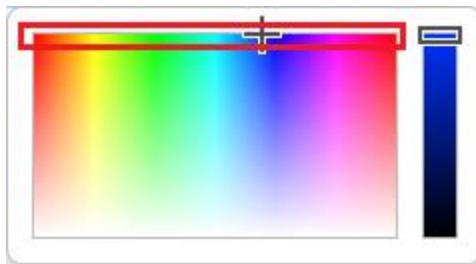


Then, you need to press the “Change Color” button to actually set that color.



To turn off the RGB LED strip, select the black color.

The strongest colors (at the top of the color picker), are the ones that will produce better results.



Now, you can use the strip to decorate your house: under the bed, behind a TV, under the kitchen cabinet, and much more.

Unit 9: Web Server using SPIFFS

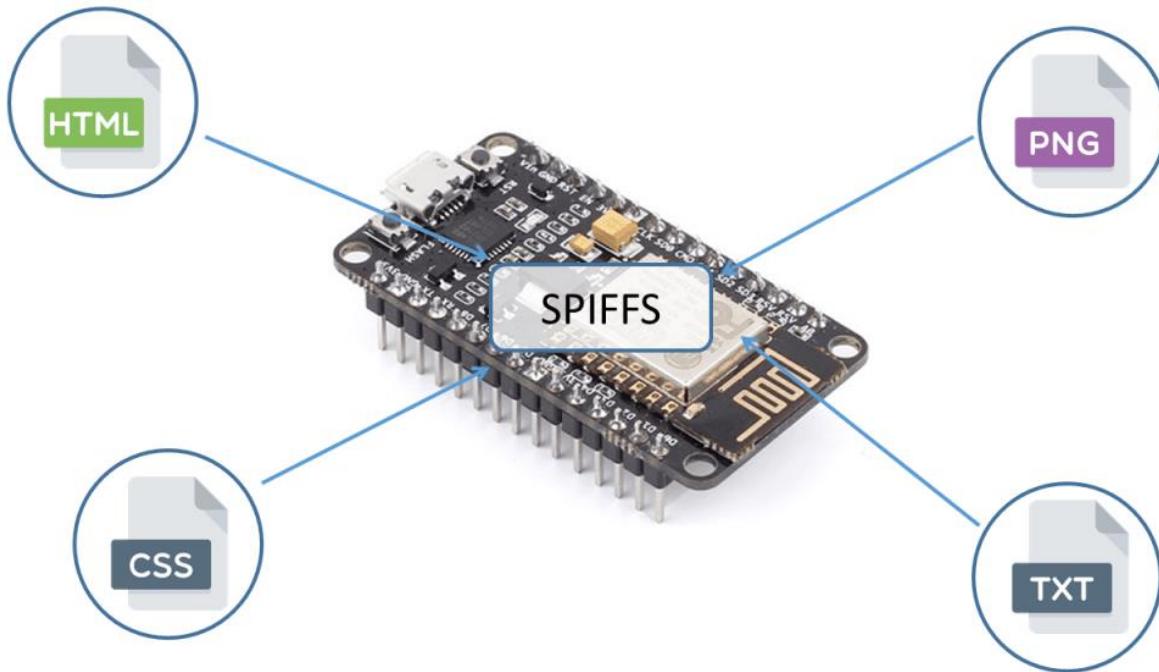


This tutorial shows how to build a web server that serves HTML and CSS files stored on the ESP8266 NodeMCU filesystem (SPIFFS) using Arduino IDE. Instead of having to write the HTML and CSS text into the Arduino sketch, we'll create separate HTML and CSS files.

Introducing SPIFFS

The ESP8266 contains a Serial Peripheral Interface Flash File System (SPIFFS). SPIFFS is a lightweight filesystem created for microcontrollers with a flash chip.

SPIFFS lets you access the flash chip memory like you would do in a normal filesystem in your computer, but simpler and more limited. You can read, write, close, and delete files.



Using SPIFFS with the [ESP8266 board](#) is specially useful to:

- Create configuration files with settings;
- Save data permanently;
- Create files to save small amounts of data instead of using a microSD card;
- Save HTML and CSS files to build a web server;
- [Save images, figures and icons](#);
- And much more.

In the previous web server projects, we've written the HTML code for the web server as a String directly on the Arduino sketch. With SPIFFS, you can write the HTML and CSS in separated files and save them on the ESP8266 filesystem. That's what we're going to do in this Unit.

Installing the Arduino ESP8266 Filesystem Uploader

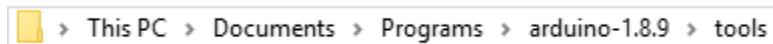
You can create, save and write files to the ESP8266 filesystem by writing the code yourself in Arduino IDE. This is not very useful, because you'd have to type the content of your files in the Arduino sketch.

Fortunately, there is a plugin for the Arduino IDE that allows you to upload files directly to the ESP8266 filesystem from a folder in your computer. This makes it really easy and simple to work with files. Let's install it.

- 1) Go to the [releases page](#) and click the `ESP8266FS-X.zip` file to download.

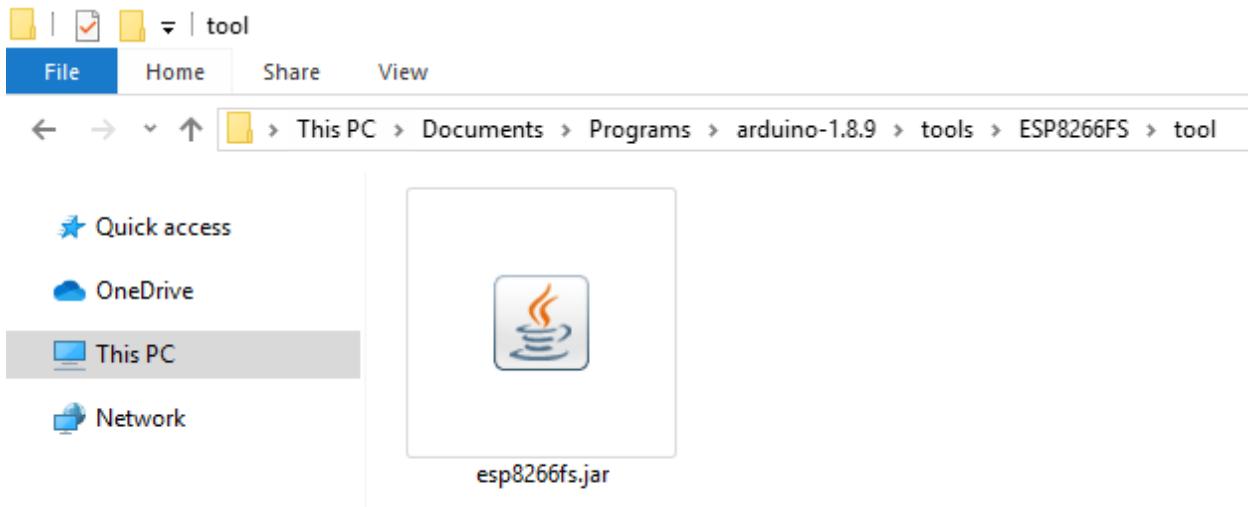
The screenshot shows a GitHub release page for the 'Support for uploading to ESP8266 using esptool.py' project. The release is labeled 'Latest release' and is version 0.4.0, released by igrr on Feb 25. The notes mention that the latest version of ESP8266 Arduino uses esptool.py for uploads. Below the notes, there is a section for 'Assets' containing three items: 'ESP8266FS-0.4.0.zip', 'Source code (zip)', and 'Source code (tar.gz)'. The 'Assets' section has a count of 3.

- 2) Go to the Arduino IDE directory, and open the **Tools** folder.



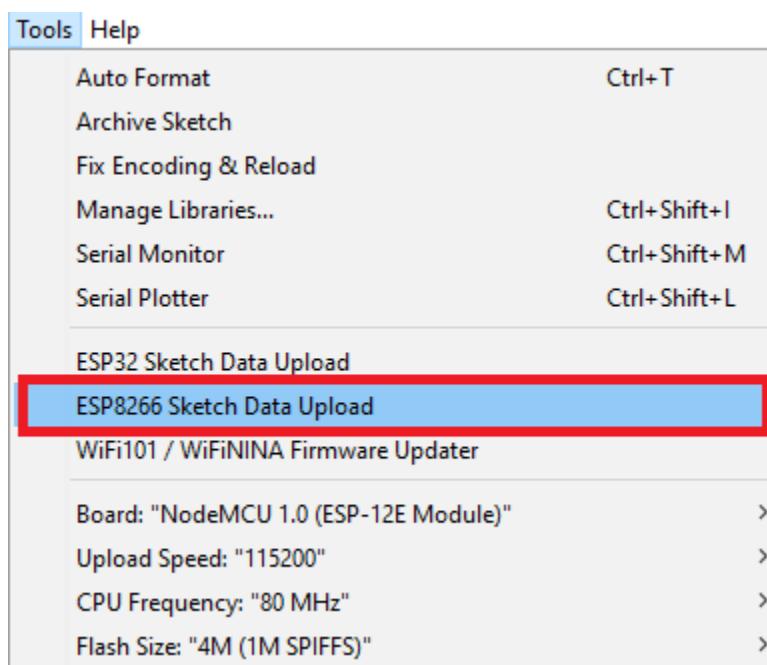
- 3) Unzip the downloaded `.zip` folder to the **Tools** folder. You should have a similar folder structure:

```
<home_dir>/Arduino-<version>/tools/ESP8266FS/tool/esp8266fs.jar
```



4) Finally, restart your Arduino IDE.

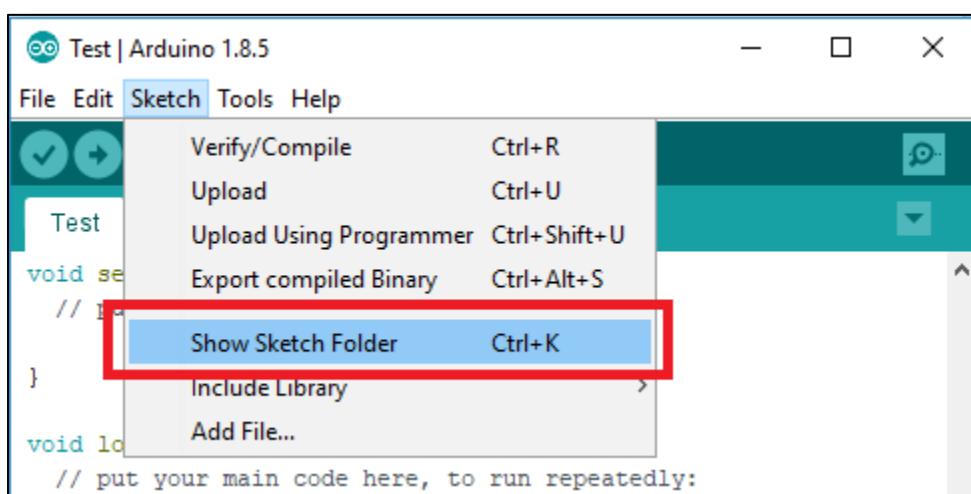
To check if the plugin was successfully installed, open your Arduino IDE and select your ESP8266 board. In the **Tools** menu check that you have the option "[ESP8266 Sketch Data Upload](#)".



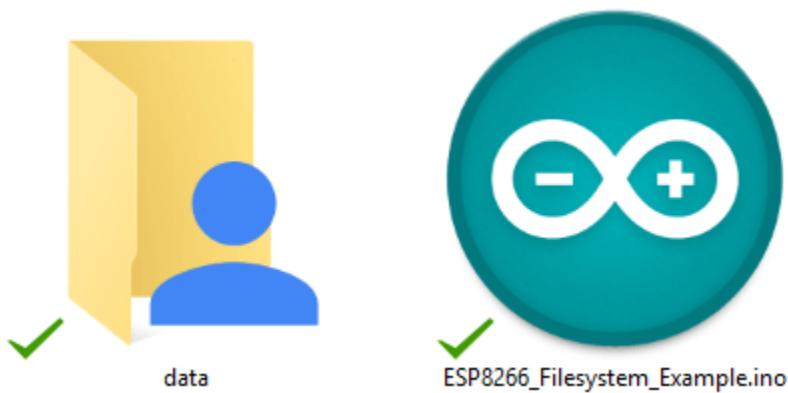
Uploading Files using the Filesystem Uploader

To upload files to the ESP8266 filesystem follow the next instructions.

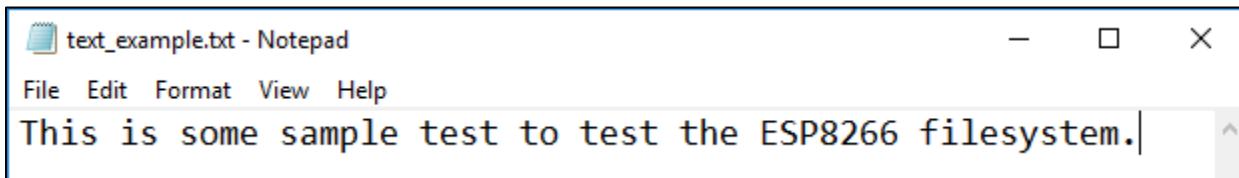
- 1) Create an Arduino sketch and save it. For demonstration purposes, you can save an empty sketch.
- 2) Then, open the sketch folder. You can go to **Sketch > Show Sketch Folder**. The folder where your sketch is saved should open.



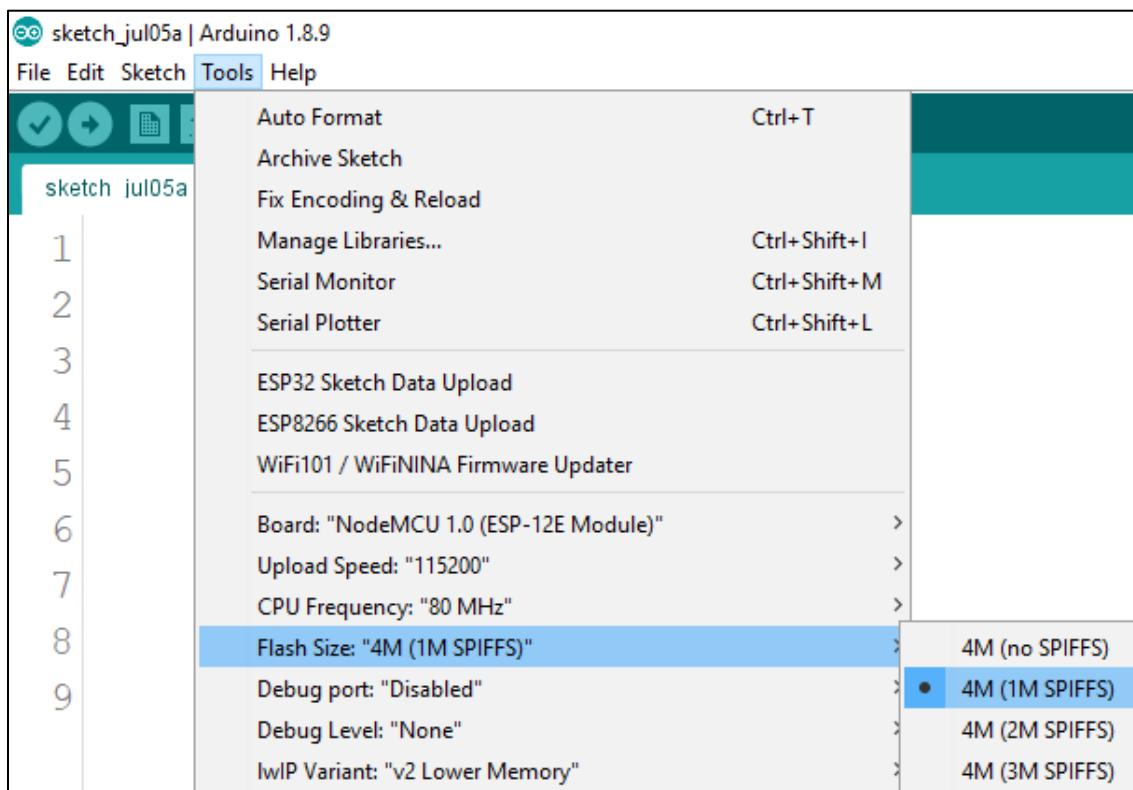
- 3) Inside that folder, create a new folder called **data**.



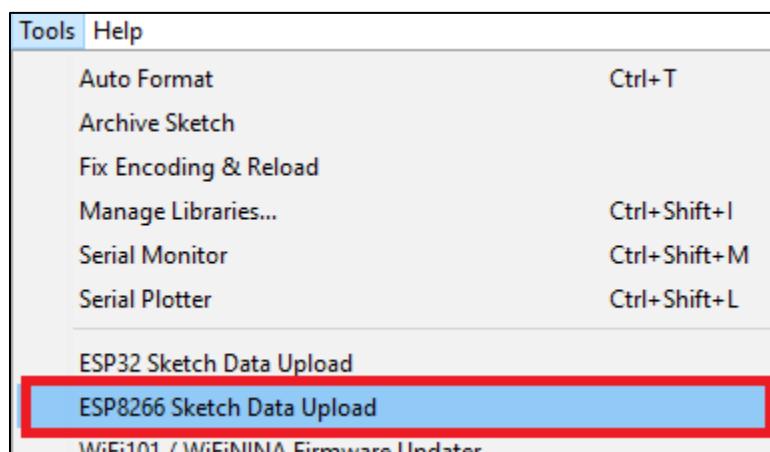
- 4) Inside the **data** folder is where you should put the files you want to be saved into the ESP8266 filesystem. As an example, create a **.txt** file with some text called *test_example*.



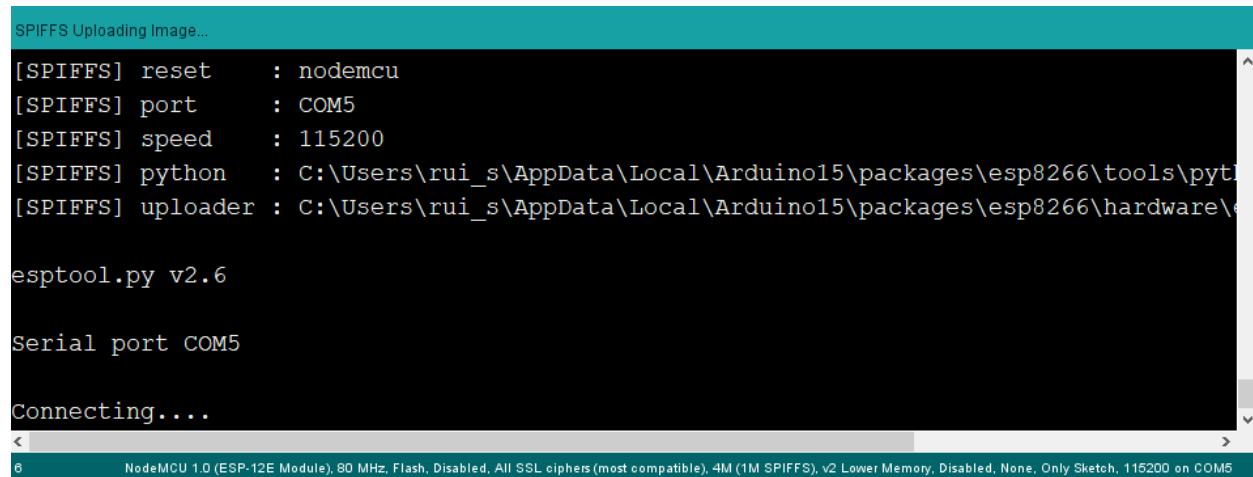
- 5) In the Arduino IDE, in the **Tools** menu, select the desired SPIFFS size (this will depend on the size of your files).



- 6) Then, to upload the files, in the Arduino IDE, you just need to go to **Tools** ▶ **ESP8266 Sketch Data Upload**.



You should get a similar message on the debugging window. The files were successfully uploaded to the ESP8266 filesystem.



The screenshot shows a terminal window titled "SPIFFS Uploading Image...". It displays the configuration for the SPIFFS uploader, including the reset pin (nodeMCU), port (COM5), speed (115200), Python path (C:\Users\rui_s\AppData\Local\Arduino15\packages\esp8266\tools\pyt...), and uploader path (C:\Users\rui_s\AppData\Local\Arduino15\packages\esp8266\hardware\...). Below this, it shows "esptool.py v2.6", "Serial port COM5", and "Connecting....". At the bottom, it shows the NodeMCU 1.0 (ESP-12E Module) configuration: 80 MHz, Flash Disabled, All SSL ciphers (most compatible), 4M (1M SPIFFS), v2 Lower Memory, Disabled, None, Only Sketch, 115200 on COM5.

Testing the Uploader

Now, let's just check if the file was actually saved into the ESP8266 filesystem. Simply upload the following code to your ESP8266 board.

```
#include "FS.h"

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

    if(!SPIFFS.begin()){
        Serial.println("An Error has occurred while mounting SPIFFS");
        return;
    }

    File = SPIFFS.open("/test_example.txt", "r");
    if(!file){
        Serial.println("Failed to open file for reading");
        return;
    }

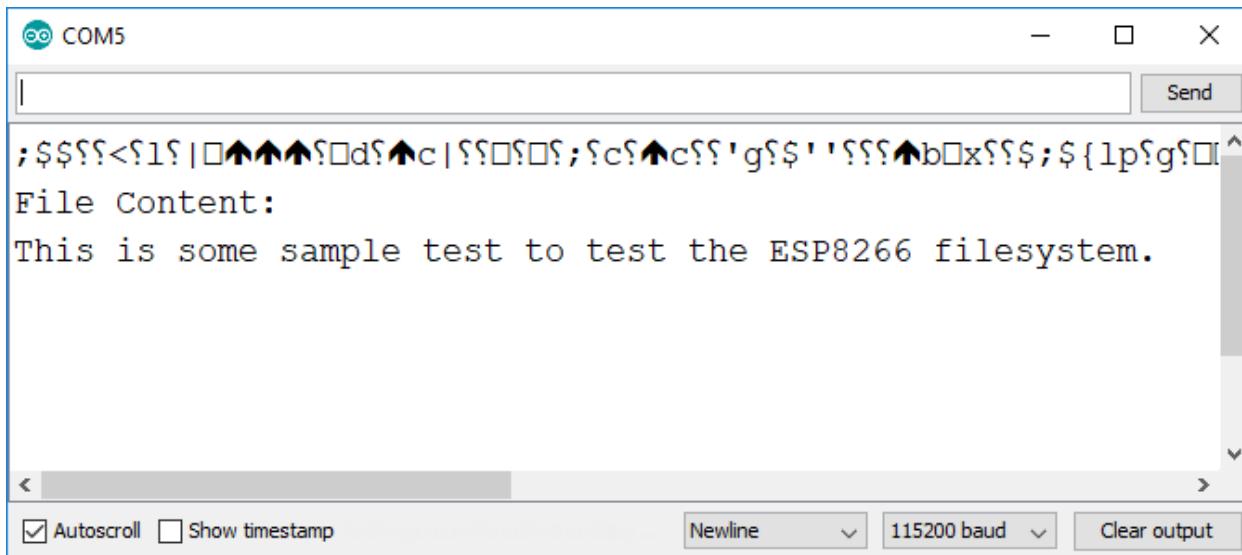
    Serial.println();
    Serial.println("File Content:");
    while(file.available()){
        Serial.write(file.read());
    }
    file.close();
}

void loop() {
```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module4/Unit9/Read_File_SPIFFS/Read_File_SPIFFS.ino

After uploading, open the Serial Monitor at a baud rate of 115200. Press the ESP8266 “RST” button. It should print the content of your .txt file on the Serial Monitor.



You've successfully uploaded files to the ESP8266 filesystem using the plugin.

Now that you have the ESP8266 Filesystem Uploader plugin installed and that you know how to upload files to your ESP8266 board, let's proceed with the web server project.

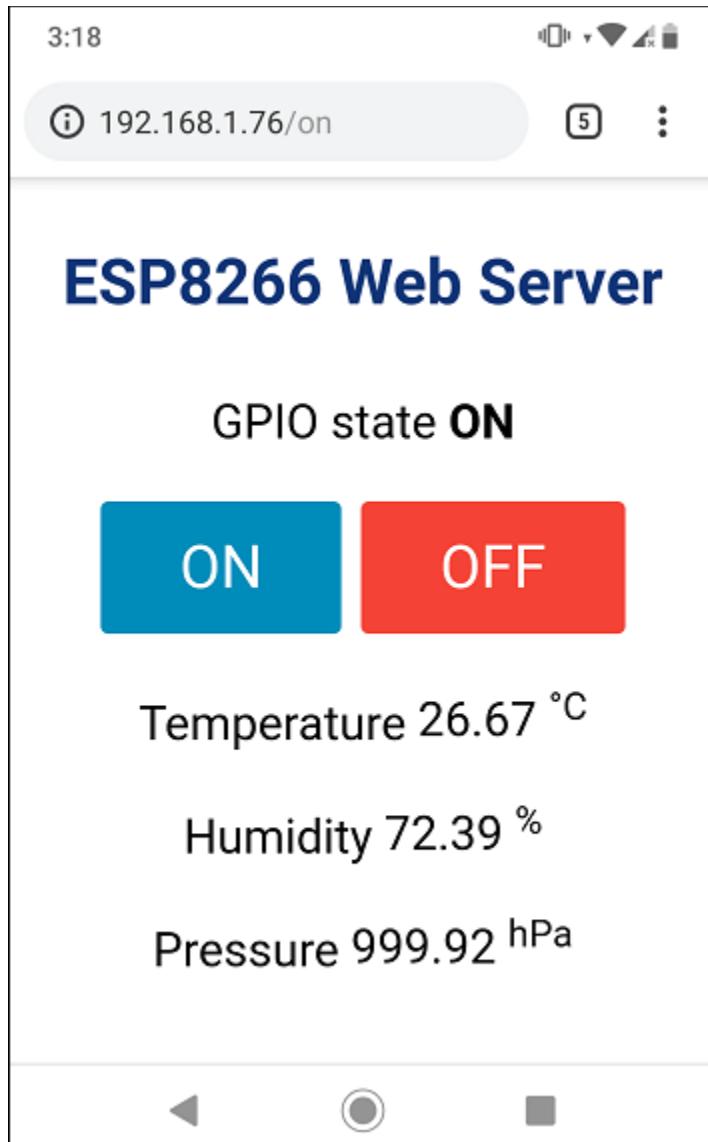
ESP8266 Web Server using SPIFFS

The web server we'll build shows how to control the ESP8266 outputs and how to display sensor readings. As an example, we'll control an LED and display sensor readings from a BME280 sensor.

You can use the concepts learned in this tutorial to control any output or display sensor readings from other sensors.

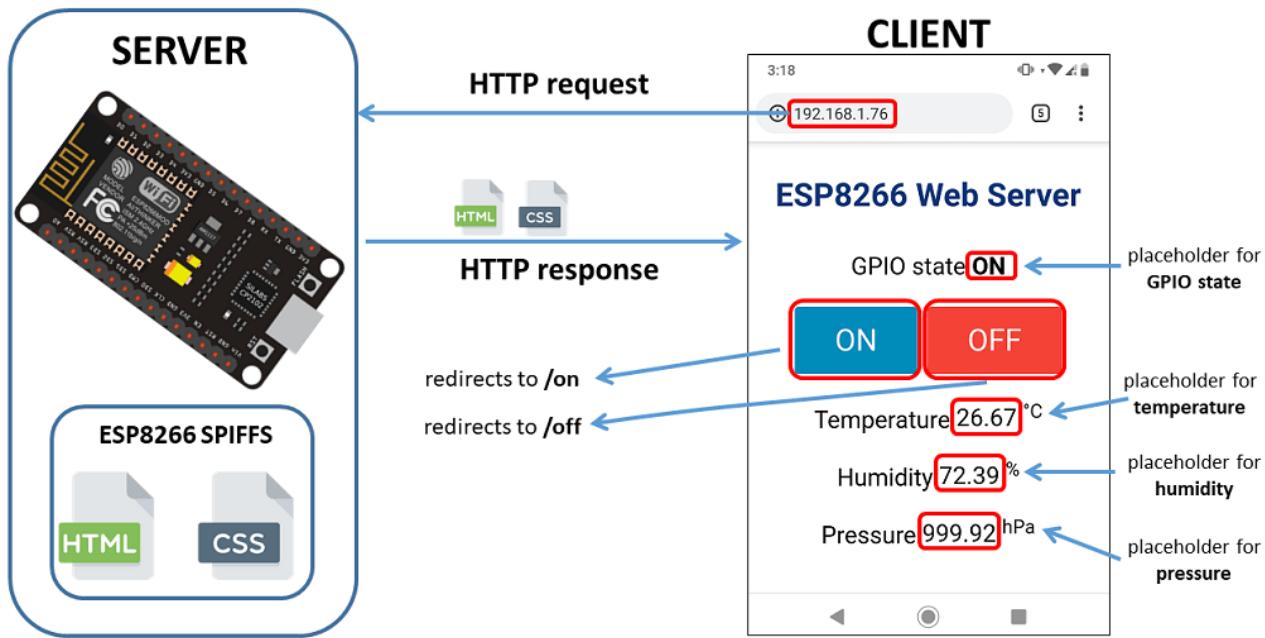
Project Overview

Before going straight to the project, it's important to outline what our web server will do, so that it's easier to understand.



- The web server controls an LED connected to the ESP8266 GPIO 2. This is the ESP8266 on-board LED. You can control any other GPIO;
- The web server page shows two buttons: ON and OFF to control GPIO 2;
- The web server page also shows the current GPIO state;
- You'll also use a BME280 sensor to display sensor readings (temperature, humidity, and pressure).

The following figure shows a simplified diagram to demonstrate how everything works.



Prerequisites

Before proceeding with this project, make sure you check all the following prerequisites.

1. Filesystem Uploader Plugin

To upload files to the ESP8266 SPI Flash Filesystem (SPIFFS), we'll use the Filesystem Uploader Plugin. Make sure you have the plugin installed before proceeding. The instructions to install the plugin were given before in this Unit.

2. Installing Libraries

One of the easiest ways to build a web server using files from the filesystem is using the [ESPAsyncWebServer](#) library. If you've followed previous Units in this course, you should already have the following libraries installed. If don't, follow the instructions to install the libraries.

Installing the ESPAsyncWebServer library

This library is not available to download through the Arduino IDE libraries manager. So, you need to follow the next steps to install the library:

1. [Click here to download the ESPAsyncWebServer library](#). You should have a *.zip* folder in your *Downloads* folder
2. Unzip the *.zip* folder and you should get *ESPAsyncWebServer-master* folder
3. Rename your folder from *ESPAsyncWebServer-master* to *ESPAsyncWebServer*
4. Move the *ESPAsyncWebServer* folder to your Arduino IDE installation libraries folder

Alternatively, you can go to **Sketch > Include Library > .zip Library** and select the previously downloaded library.

Installing the ESPAsyncTCP

The ESPAsyncWebServer library also needs the [ESPAsyncTCP library](#) to operate properly. Follow the next steps to install the ESPAsyncTCP library:

1. [Click here to download the ESPAsyncTCP library](#). You should have a *.zip* folder in your *Downloads* folder
2. Unzip the *.zip* folder and you should get *ESPAsyncTCP-master* folder
3. Rename your folder from *ESPAsyncTCP-master* to *ESPAsyncTCP*
4. Move the *ESPAsyncTCP* folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Alternatively, you can go to **Sketch > Include Library > .zip Library** and select the previously downloaded library.

Installing BME280 libraries

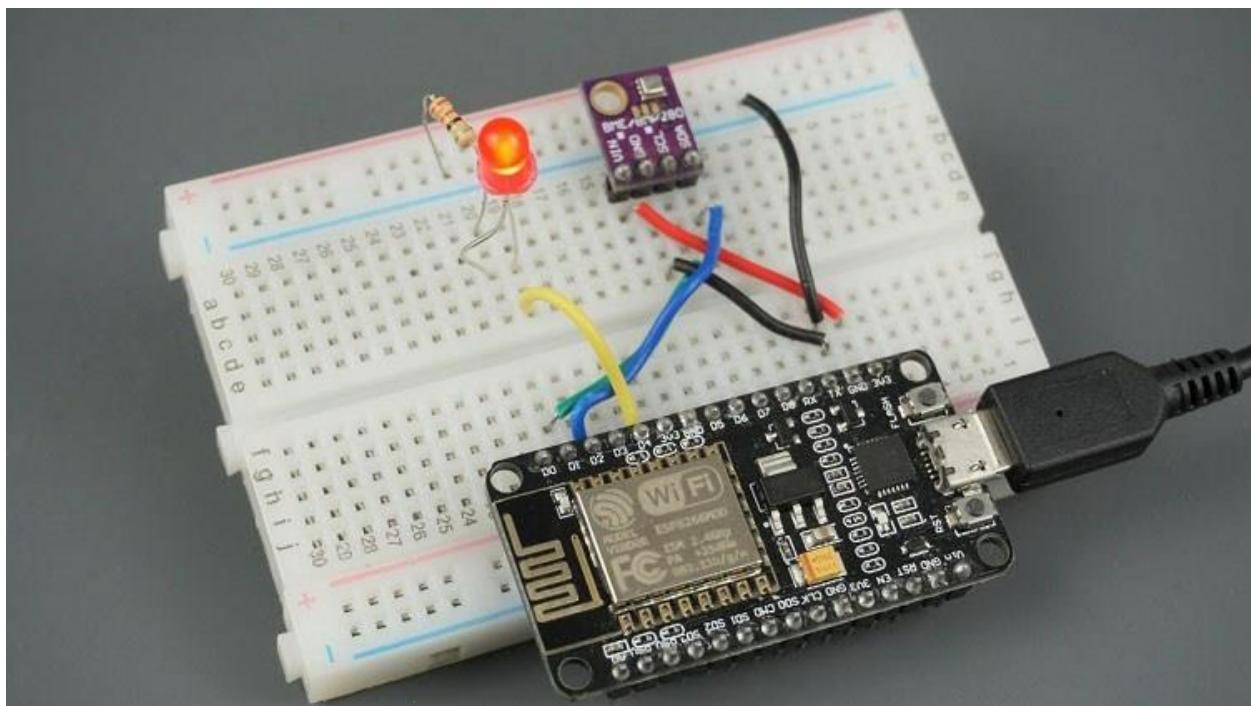
In this tutorial, we'll display readings from a BME280 sensor. You need to install the following libraries:

- [Adafruit BME280 Library](#)
- [Adafruit Sensor](#)

You can install these libraries through the Arduino IDE Libraries Manager. Go to **Sketch > Include Libraries > Manage Libraries**. Then, search for the libraries' name to install them.

If you've followed previous Units in this course, you should already have these libraries installed.

Parts Required

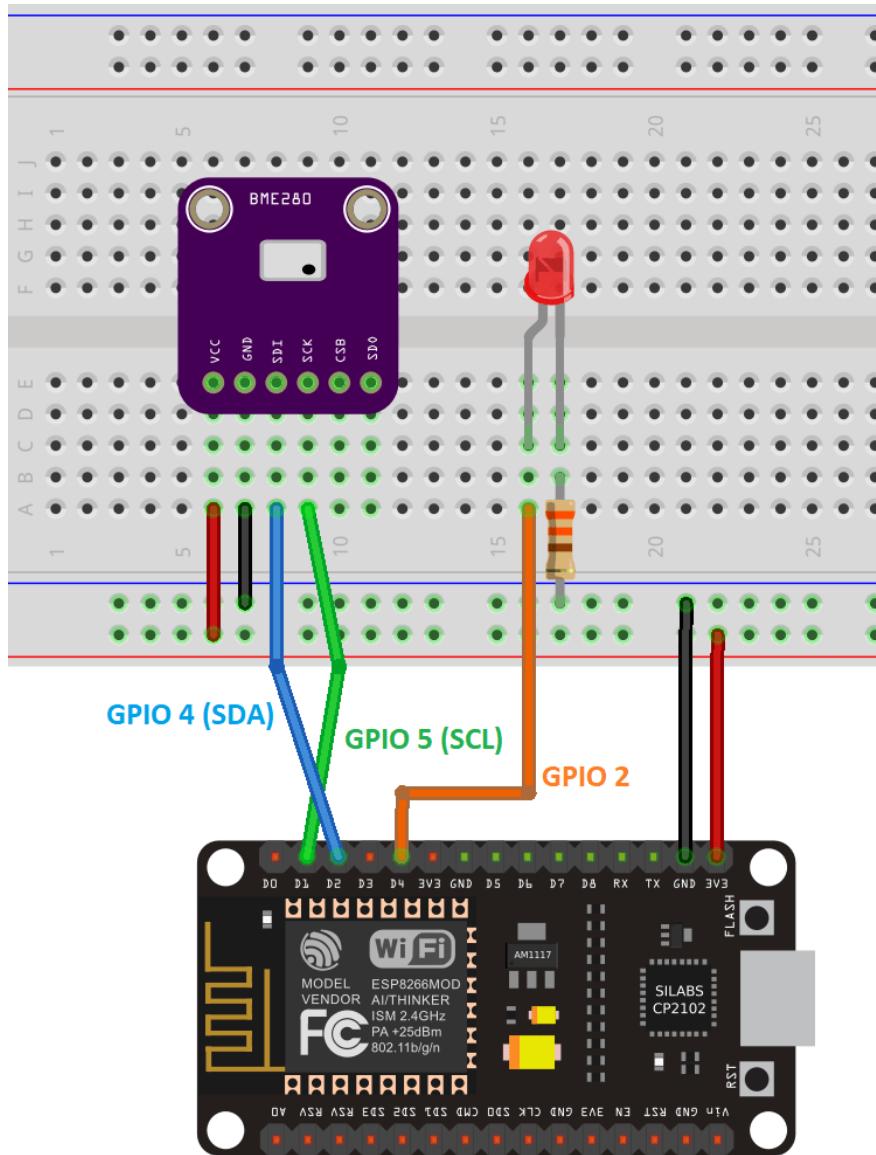


To proceed with this project, you need the following parts:

- [ESP8266](#)
- [BME280 sensor module](#)
- [5 mm LED](#)
- [330 Ohm resistor](#)
- [Breadboard](#)
- [Jumper wires](#)

Schematic Diagram

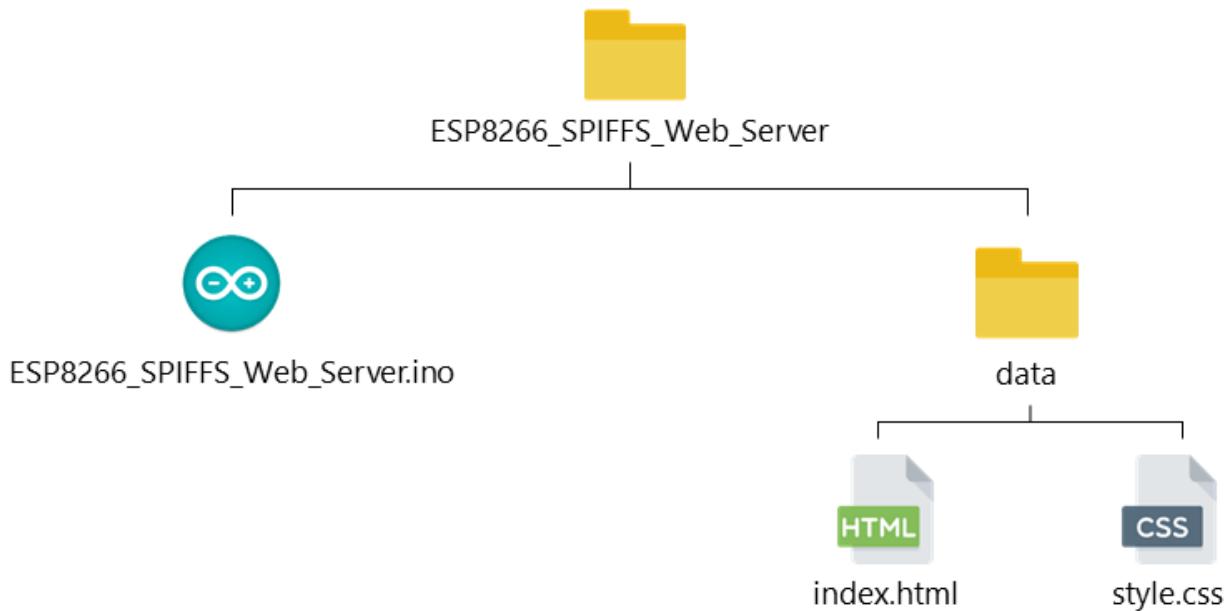
Connect all the components by following the next schematic diagram.



BME280	ESP8266
Vin	3.3V
GND	GND
SCL	GPIO 5
SDA	GPIO 4

Organizing Your Files

To build the web server you need three different files. The Arduino sketch, the HTML file and the CSS file. The HTML and CSS files should be saved inside a folder called **data** inside the Arduino sketch folder, as shown below:



Creating the HTML File

Create an *index.html* file with the following content or [download all project files here](#):

```
<!DOCTYPE html>
<!--
  Rui Santos
  Complete project details at https://RandomNerdTutorials.com
-->
<html>
<head>
  <title>ESP8266 Web Server</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" href="data:,">
  <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
  <h1>ESP8266 Web Server</h1>
  <p>GPIO state<strong> %STATE%</strong></p>
  <p>
    <a href="/on"><button class="button">ON</button></a>
    <a href="/off"><button class="button button2">OFF</button></a>
  </p>
</body>
</html>
```

```

</p>
<p>
    <span class="sensor-labels">Temperature</span>
    <span id="temperature">%TEMPERATURE%</span>
    <sup class="units">&deg;C</sup>
</p>
<p>
    <span class="sensor-labels">Humidity</span>
    <span id="humidity">%HUMIDITY%</span>
    <sup class="units">&#37;</sup>
</p>
<p>
    <span class="sensor-labels">Pressure</span>
    <span id="pressure">%PRESSURE%</span>
    <sup class="units">hPa</sup>
</p>
</body>
<script>
    setInterval(function () {
        var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                document.getElementById("temperature").innerHTML =
this.responseText;
            }
        };
        xhttp.open("GET", "/temperature", true);
        xhttp.send();
    }, 10000 ) ;

    setInterval(function () {
        var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                document.getElementById("humidity").innerHTML =
this.responseText;
            }
        };
        xhttp.open("GET", "/humidity", true);
        xhttp.send();
    }, 10000 ) ;

    setInterval(function () {
        var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                document.getElementById("pressure").innerHTML =
this.responseText;
            }
        };
        xhttp.open("GET", "/pressure", true);
        xhttp.send();
    }, 10000 ) ;
</script>
</html>

```

SOURCE CODE

```
https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1\_Arduino/Module4/Unit9/Web\_Server\_SPIFFS/data/index.html
```

Because we're using CSS and HTML in different files, we need to reference the CSS file on the HTML text.

```
<link rel="stylesheet" type="text/css" href="style.css">
```

The `<link>` tag tells the HTML file that you're using an external style sheet to format how the page looks. The `rel` attribute specifies the nature of the external file, in this case that it is a stylesheet—the CSS file—that will be used to alter the appearance of the page.

The `type` attribute is set to “`text/css`” to indicate that you're using a CSS file for the styles. The `href` attribute indicates the file location; since both the CSS and HTML files will be in the same folder, you just need to reference the filename: `style.css`.

In the following line, we write the first heading of our web page. In this case we have “ESP8266 Web Server”. You can change the heading.

```
<h1>ESP8266 Web Server</h1>
```

Then, add a paragraph with the text “`GPIO state:`” followed by the GPIO state. Because the GPIO state changes accordingly to the state of the GPIO, we can add a placeholder that will then be replaced for whatever value we set on the Arduino sketch.

To add a placeholder use `%` signs. To create a placeholder for the state, you can use `%STATE%`, for example.

```
<p>GPIO state<strong> %STATE%</strong></p>
```

You attribute a value to the STATE placeholder in the Arduino sketch.

Then, create the ON and OFF buttons. When you click the ON button, we redirect the web page to the root followed by /on url. When you click the off button, you are redirected to the /off url.

```
<a href="/on"><button class="button">ON</button></a>
<a href="/off"><button class="button button2">OFF</button></a>
```

Finally, create three paragraphs to display the temperature, humidity and pressure.

```
<p>
  <span class="sensor-labels">Temperature</span>
  <span id="temperature">%TEMPERATURE%</span>
  <sup class="units">&deg;C</sup>
</p>
<p>
  <span class="sensor-labels">Humidity</span>
  <span id="humidity">%HUMIDITY%</span>
  <sup class="units">&#37;</sup>
</p>
<p>
  <span class="sensor-labels">Pressure</span>
  <span id="pressure">%PRESSURE%</span>
  <sup class="units">hPa</sup>
</p>
```

We use the %TEMPERATURE%, %HUMIDITY% and %PRESSURE% placeholders. These will then be replaced by the actual temperature readings in the Arduino sketch.

Automatic Updates

We also add a bit of JavaScript in our HTML file that is responsible for updating the temperature readings without the need to refresh the web page. The following snippet of code is responsible for the temperature.

```
setInterval(function () {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("temperature").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/temperature", true);
  xhttp.send();
}, 10000);
```

To update the temperature, we have a `setInterval()` function that runs every 10 seconds.

Basically, it makes a request in the `/temperature` URL to get the latest temperature reading.

```
xhttp.open("GET", "/temperature", true);
xhttp.send();
}, 10000 );
```

When it receives that value, it updates the HTML element with temperature id.

```
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    document.getElementById("temperature").innerHTML = this.responseText;
  }
}
```

In summary, this previous section is responsible for updating the temperature asynchronously. The same process is repeated for the humidity and pressure readings.

Creating the CSS File

Create the `style.css` file with the following content or [download all project files here](#):

```
html {
  font-family: Arial;
  display: inline-block;
  margin: 0px auto;
  text-align: center;
}
h1 {
  color: #0F3376;
  padding: 2vh;
}
p {
  font-size: 1.5rem;
}
.button {
  display: inline-block;
  background-color: #008CBA;
  border: none;
  border-radius: 4px;
  color: white;
  padding: 16px 40px;
  text-decoration: none;
  font-size: 30px;
```

```

    margin: 2px;
    cursor: pointer;
}
.button2 {
    background-color: #f44336;
}
.units {
    font-size: 1.2rem;
}
.sensor-labels {
    font-size: 1.5rem;
    vertical-align:middle;
    padding-bottom: 15px;
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module4/Unit9/Web_Server_SPIFFS/data/style.css

This is just a basic CSS file to set the font size, style and color of the buttons and align the page. We won't explain how CSS works. A good place to learn about CSS is the [W3Schools website](#).

ESP8266 Asynchronous Web Server Sketch

Copy the following code to the Arduino IDE or [download all project files here](#). Then, you need to type your network credentials (SSID and password) to connect the ESP8266 to your local network.

```

// Import required libraries
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <FS.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

Adafruit_BME280 bme; // I2C
//Adafruit_BME280 bme(BME_CS); // hardware SPI
//Adafruit_BME280 bme(BME_CS, BME_MOSI, BME_MISO, BME_SCK); // software SPI

```

```

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

// Set LED GPIO
const int ledPin = 2;
// Stores LED state
String ledState;

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

String getTemperature() {
    float temperature = bme.readTemperature();
    // Read temperature as Fahrenheit (isFahrenheit = true)
    //float temperature = 1.8* bme.readTemperature()+32;
    Serial.println(temperature);
    return String(temperature);
}

String getHumidity() {
    float humidity = bme.readHumidity();
    Serial.println(humidity);
    return String(humidity);
}

String getPressure() {
    float pressure = bme.readPressure() / 100.0F;
    Serial.println(pressure);
    return String(pressure);
}

// Replaces placeholder with LED state value
String processor(const String& var) {
    Serial.println(var);
    if(var == "STATE") {
        if(digitalRead(ledPin)) {
            ledState = "ON";
        }
        else{
            ledState = "OFF";
        }
        Serial.print(ledState);
        return ledState;
    }
    else if (var == "TEMPERATURE") {
        return getTemperature();
    }
    else if (var == "HUMIDITY") {
        return getHumidity();
    }
    else if (var == "PRESSURE") {
        return getPressure();
    }
}

```

```

void setup() {
    // Serial port for debugging purposes
    Serial.begin(115200);
    pinMode(ledPin, OUTPUT);

    // Initialize the sensor
    if (!bme.begin(0x76)) {
        Serial.println("Could not find a valid BME280 sensor, check wiring!");
        while (1);
    }

    // Initialize SPIFFS
    if(!SPIFFS.begin()){
        Serial.println("An Error has occurred while mounting SPIFFS");
        return;
    }

    // Connect to Wi-Fi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connecting to WiFi..");
    }

    // Print ESP8266 Local IP Address
    Serial.println(WiFi.localIP());

    // Route for root / web page
    server.on("/", HTTP_GET, [] (AsyncWebServerRequest *request) {
        request->send(SPIFFS, "/index.html", String(), false, processor);
    });

    // Route to load style.css file
    server.on("/style.css", HTTP_GET, [] (AsyncWebServerRequest *request) {
        request->send(SPIFFS, "/style.css", "text/css");
    });

    // Route to set GPIO to HIGH
    server.on("/on", HTTP_GET, [] (AsyncWebServerRequest *request) {
        digitalWrite(ledPin, HIGH);
        request->send(SPIFFS, "/index.html", String(), false, processor);
    });

    // Route to set GPIO to LOW
    server.on("/off", HTTP_GET, [] (AsyncWebServerRequest *request) {
        digitalWrite(ledPin, LOW);
        request->send(SPIFFS, "/index.html", String(), false, processor);
    });

    server.on("/temperature",      HTTP_GET,      [] (AsyncWebServerRequest
*request) {
        request->send_P(200, "text/plain", getTemperature().c_str());
    });
}

```

```

server.on("/humidity", HTTP_GET, [] (AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", getHumidity().c_str());
});

server.on("/pressure", HTTP_GET, [] (AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", getPressure().c_str());
});

// Start server
server.begin();
}

void loop() {
}

```

SOURCE CODE

[https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module4/Unit9/Web Server SPIFFS/Web Server SPIFFS.ino](https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module4/Unit9/Web%20Server%20SPIFFS/Web%20Server%20SPIFFS.ino)

How the code works

Continue reading to learn how the code works, or skip to the next section.

First, include the necessary libraries:

```
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <FS.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
```

You need to type your network credentials in the following variables:

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

Create an instance that refers to the BME280 sensor called `bme`:

```
Adafruit_BME280 bme; // I2C
```

Next, create a variable that refers to GPIO 2 called `ledPin`, and a String variable to hold the led state: `ledState`.

```
// Set LED GPIO
const int ledPin = 2;
// Stores LED state
String ledState;
```

Create an `AsynWebServer` object called `server` that is listening on port 80.

```
AsyncWebServer server(80);
```

Get Sensor Readings

We create three functions to return the sensor readings as strings: the `getTemperature()`, `getHumidity()` and `getPressure()` functions.

Here's how the `getTemperature()` function looks like (the other functions are similar).

```
String getTemperature() {
    float temperature = bme.readTemperature();
    // Read temperature as Fahrenheit (isFahrenheit = true)
    //float temperature = 1.8*bme.readTemperature()+32;
    Serial.println(temperature);
    return String(temperature);
}
```

If you want to display temperature in Fahrenheit degrees, you just need to uncomment the corresponding line in the `getTemperature()` function:

```
float temperature = 1.8 * bme.readTemperature()+32;
```

processor()

The `processor()` function attributes a value to the placeholders we've created on the HTML file. It accepts as argument the placeholder and should return a String that will replace the placeholder. The `processor()` function should have the following structure:

```
String processor(const String& var) {
    Serial.println(var);
    if(var == "STATE") {
        if(digitalRead(ledPin)){
            ledState = "ON";
        }
        else{
            ledState = "OFF";
        }
        Serial.print(ledState);
        return ledState;
    }
    else if (var == "TEMPERATURE") {
        return getTemperature();
    }
    else if (var == "HUMIDITY") {
        return getHumidity();
    }
    else if (var == "PRESSURE") {
        return getPressure();
    }
}
```

This function first checks if the placeholder is the STATE we've created on the HTML file.

```
if(var == "STATE") {
```

If it is, then, accordingly to the LED state, we set the ledState variable to either ON or OFF.

```
if(digitalRead(ledPin)) {
    ledState = "ON";
}
else{
    ledState = "OFF";
}
```

Finally, we return the ledState variable. This replaces the STATE placeholder with the ledState string value.

```
return ledState;
```

If it finds the %TEMPERATURE% placeholder, we return the temperature by calling the getTemperature() function created previously.

```
else if (var == "TEMPERATURE") {
    return getTemperature();
}
```

The same happens with the %HUMIDITY% and %PRESSURE% placeholders by calling the corresponding functions:

```
else if (var == "TEMPERATURE") {
    return getTemperature();
}
else if (var == "HUMIDITY") {
    return getHumidity();
}
else if (var == "PRESSURE") {
    return getPressure();
}
```

setup()

In the `setup()`, start by initializing the Serial Monitor and setting the GPIO as an output.

```
Serial.begin(115200);
pinMode(ledPin, OUTPUT);
```

Initialize the BME280 sensor:

```
if (!bme.begin(0x76)) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
}
```

Initialize SPIFFS:

```
if (!SPIFFS.begin()) {
    Serial.println("An Error has occurred while mounting SPIFFS");
    return;
}
```

Wi-Fi connection

Connect to Wi-Fi and print the ESP8266 address:

```
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
```

```

    delay(1000);
    Serial.println("Connecting to WiFi..");
}
// Print ESP8266 Local IP Address
Serial.println(WiFi.localIP());

```

Async Web Server

The ESPAsyncWebServer library allows us to configure the routes where the server will be listening for incoming HTTP requests and execute functions when a request is received on that route. For that, use the `on()` method on the `server` object as follows:

```

// Route for root / web page
server.on("/", HTTP_GET, [] (AsyncWebRequest *request) {
    request->send(SPIFFS, "/index.html", String(), false, processor);
});

```

When the server receives a request on the root "/" URL, it will send the `index.html` file to the client. The last argument of the `send()` function is the `processor`, so that we can replace the placeholder with the value we want – in this case the `ledState`.

Because we've referenced the CSS file on the HTML file, the client will make a request for the CSS file. When that happens, the CSS file is sent to the client:

```

// Route to load style.css file
server.on("/style.css", HTTP_GET, [] (AsyncWebRequest *request) {
    request->send(SPIFFS, "/style.css", "text/css");
});

```

You also need to define what happens on the `/on` and `/off` routes. When a request is made on those routes, the LED is either turned on or off, and the ESP8266 serves the HTML file.

```

// Route to set GPIO to HIGH
server.on("/on", HTTP_GET, [] (AsyncWebRequest *request) {
    digitalWrite(ledPin, HIGH);
    request->send(SPIFFS, "/index.html", String(), false, processor);
});

// Route to set GPIO to LOW
server.on("/off", HTTP_GET, [] (AsyncWebRequest *request) {

```

```
digitalWrite(ledPin, LOW);
request->send(SPIFFS, "/index.html", String(), false, processor);
});
```

In the HTML file, we've written a JavaScript code that requests the temperature, humidity and pressure on the **/temperature**, **/humidity**, **/pressure** routes, respectively, every 10 seconds. So, we also need to handle what happens when we receive a request on those routes.

We simply need to send the updated sensor readings. The updated sensor readings are returned by the `getTemperature()`, `getHumidity()` and `getPressure()` functions we've created previously.

The readings are plain text, and should be sent as a char, so, we use the `c_str()` method.

```
server.on("/temperature", HTTP_GET, [] (AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", getTemperature().c_str());
});

server.on("/humidity", HTTP_GET, [] (AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", getHumidity().c_str());
});

server.on("/pressure", HTTP_GET, [] (AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", getPressure().c_str());
});
```

In the end, we use the `begin()` method on the `server` object, so that the server starts listening for incoming clients.

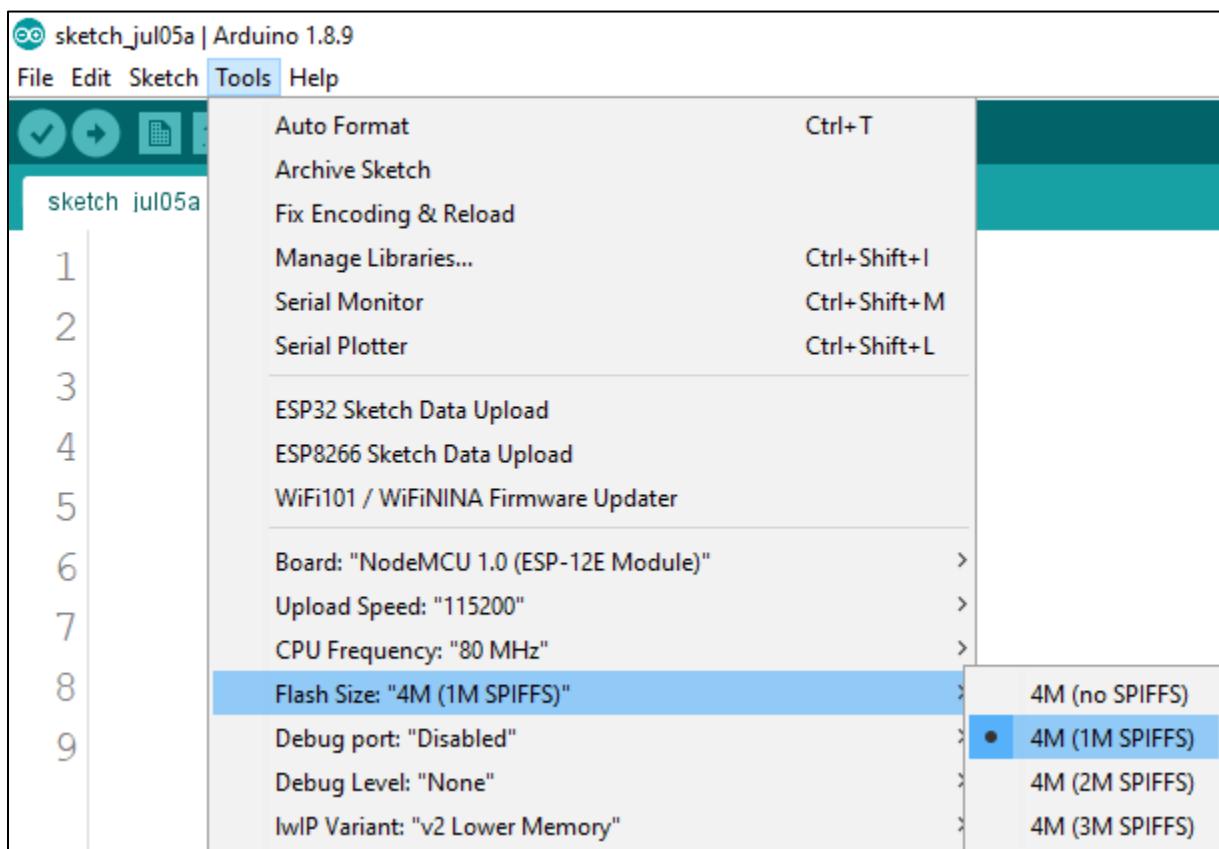
```
server.begin();
```

Because this is an asynchronous web server, you can define all the requests in the `setup()`. Then, you can add other code to the `loop()` while the server is listening for incoming clients.

Uploading Code and Files

Save the Arduino sketch as *ESP8266_SPIFFS_Web_Server* or [download all project files here](#).

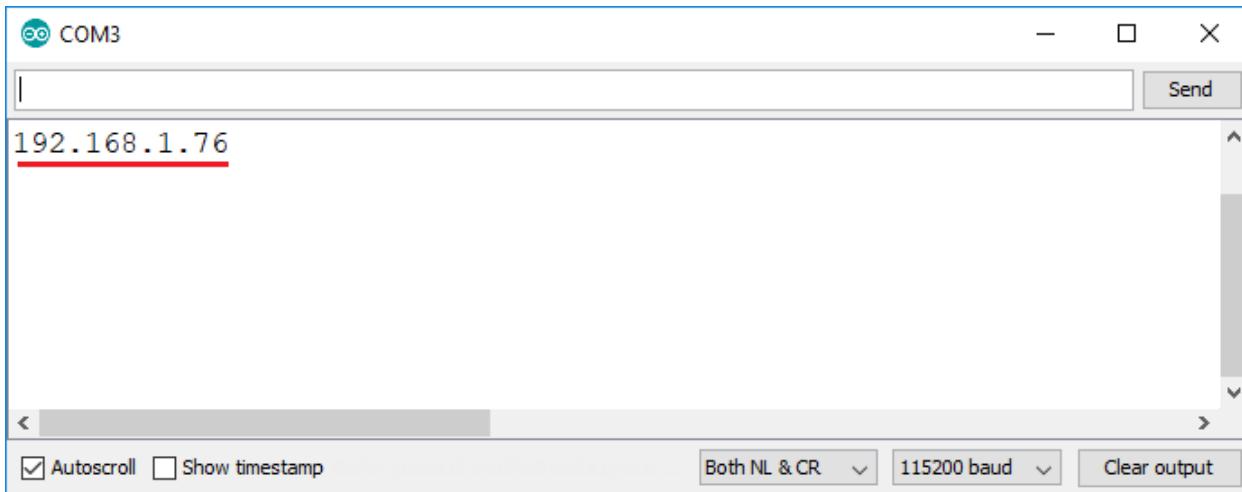
1. Go to **Sketch** ▶ **Show Sketch folder**, and create a folder called **data**. Save the HTML and CSS files inside that folder;
2. In **Tools** ▶ **Board**, select the ESP8266 board you're using;
3. Then, go to **Tools** > **Flash size** and select **4M (1M SPIFFS)**.



4. Finally, upload the files to your board. Go to **Tools** ▶ **ESP8266 Data Sketch Upload** and wait for the files to be uploaded.

Then, press the Arduino IDE upload button to upload the sketch to the ESP8266.

When everything is successfully uploaded, open the Serial Monitor at a baud rate of 115200. Press the ESP8266 on-board RST button, and it should print the ESP8266 IP address.



Demonstration

Open a browser and type your ESP8266 IP address. The following web page should load.



Press the ON and OFF buttons to control the LED. You can also visualize the latest sensor readings. The sensor readings are updated automatically without the need to refresh the web page.

Unit 10: Set the ESP8266 as an Access Point

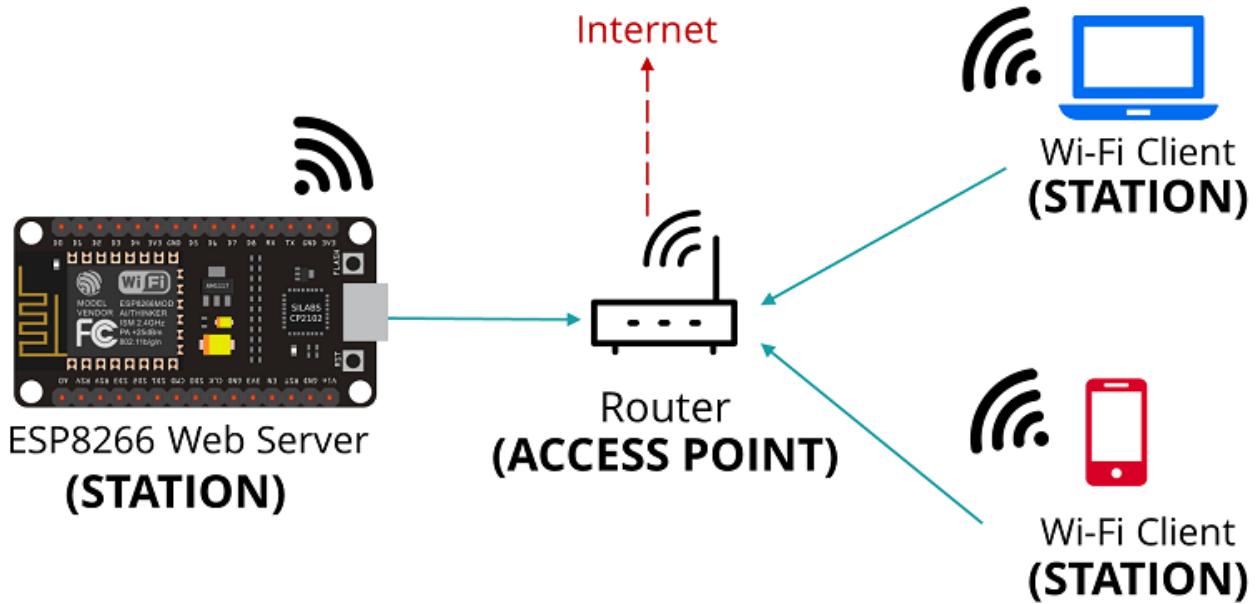


In this Unit you'll learn how to set your ESP8266 as an Access Point (AP) using Arduino IDE. This allows you to connect directly to your ESP8266 via Wi-Fi without a wireless router.

ESP8266 Station and Access Point

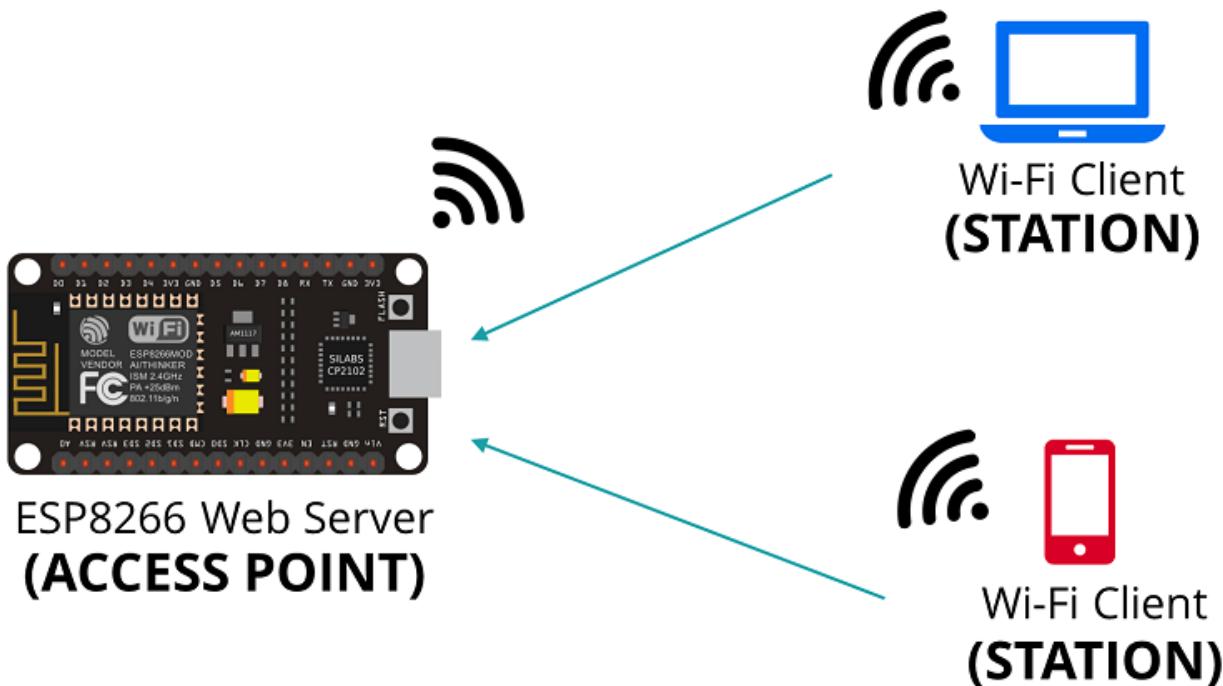
In our previous ESP8266 web server projects, we connect the ESP8266 to a wireless router. In this configuration, we can access the ESP8266 through the local network.

In this scenario, the router acts as an access point and the ESP8266 is set as a station. So, you need to be connected to your router (local network) to control the ESP8266.



In some cases, this might not be the best configuration (when you don't have a router nearby). But if you set the ESP8266 as an access point (hotspot), you can be connected to the ESP8266 using any device with Wi-Fi capabilities without the need to connect to your router.

In simple words, when you set the ESP8266 as an access point you create its own Wi-Fi network and nearby Wi-Fi devices (stations) can connect to it (like your smartphone or your computer).



In this tutorial, we'll show you how to set the ESP8266 as an access point in your web server projects. This way, you don't need to be connected to a router to control your ESP8266. Because the ESP8266 doesn't connect further to a wired network (like your router), it is called soft-AP (soft Access Point).

This means that if you try to load libraries or use firmware from the internet, it will not work. It also doesn't work if you try to make HTTP requests to services on the internet like publishing sensor readings to the cloud.

Installing the DHT Library for ESP8266

For this example, we'll use a previous web server project that [displays sensor readings from a DHT sensor](#).

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](#).

If you've followed previous projects, you should already have these libraries installed. If don't, follow the next steps.

1. Open your Arduino IDE and go to **Sketch > Include Library > Manage Libraries**. The Library Manager should open.
2. Search for “**DHT**” on the Search box and install the DHT library from Adafruit.
3. 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.
4. After installing the libraries, restart your Arduino IDE.

ESP8266 Access Point (AP)

In this example, we'll modify the [ESP8266 Web Server from a previous tutorial](#) to add access point capabilities. What we'll show you here can be used with any ESP8266 web server example.

Upload the sketch provided below to set the ESP8266 as an access point.

```
// Import required libraries
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <Hash.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>

const char* ssid      = "ESP8266-Access-Point";
const char* password = "123456789";

#define DHTPIN 5          // Digital pin connected to the DHT sensor

// Uncomment the type of sensor in use:
// #define DHTTYPE    DHT11      // DHT 11
#define DHTTYPE    DHT22      // DHT 22 (AM2302)
// #define DHTTYPE    DHT21      // DHT 21 (AM2301)

DHT dht(DHTPIN, DHTTYPE);

// current temperature & humidity, updated in loop()
float t = 0.0;
float h = 0.0;

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

// Generally, you should use "unsigned long" for variables that hold
time
// The value will quickly become too large for an int to store
unsigned long previousMillis = 0;      // will store last time DHT was
updated

// Updates DHT readings every 10 seconds
const long interval = 10000;

const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <style>
    html {
      font-family: Arial;
      display: inline-block;
      margin: 0px auto;
      text-align: center;
    }
    h2 { font-size: 3.0rem; }
    p { font-size: 3.0rem; }
    .units { font-size: 1.2rem; }
    .dht-labels{
```

```

        font-size: 1.5rem;
        vertical-align:middle;
        padding-bottom: 15px;
    }

```

```

</style>
</head>
<body>
    <h2>ESP8266 DHT Server</h2>
    <p>
        <span class="dht-labels">Temperature</span>
        <span id="temperature">%TEMPERATURE%</span>
        <sup class="units">&deg;C</sup>
    </p>
    <p>
        <span class="dht-labels">Humidity</span>
        <span id="humidity">%HUMIDITY%</span>
        <sup class="units">%</sup>
    </p>
</body>
<script>
setInterval(function () {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("temperature").innerHTML =
this.responseText;
        }
    };
    xhttp.open("GET", "/temperature", true);
    xhttp.send();
}, 10000 ) ;

setInterval(function () {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("humidity").innerHTML =
this.responseText;
        }
    };
    xhttp.open("GET", "/humidity", true);
    xhttp.send();
}, 10000 ) ;
</script>
</html>) rawliteral;

// Replaces placeholder with DHT values
String processor(const String& var){
    //Serial.println(var);
    if(var == "TEMPERATURE") {
        return String(t);
    }
    else if(var == "HUMIDITY") {
        return String(h);
    }
}

```

```

        return String();
    }

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

    Serial.print("Setting AP (Access Point)...");
    // Remove the password parameter, if you want the AP (Access Point)
    to be open
    WiFi.softAP(ssid, password);

    IPAddress IP = WiFi.softAPIP();
    Serial.print("AP IP address: ");
    Serial.println(IP);

    // Print ESP8266 Local IP Address
    Serial.println(WiFi.localIP());

    // Route for root / web page
    server.on("/", HTTP_GET, [] (AsyncWebServerRequest *request) {
        request->send_P(200, "text/html", index_html, processor);
    });
    server.on("/temperature",      HTTP_GET,      [] (AsyncWebServerRequest
*request) {
        request->send_P(200, "text/plain", String(t).c_str());
    });
    server.on("/humidity",        HTTP_GET,      [] (AsyncWebServerRequest *request) {
        request->send_P(200, "text/plain", String(h).c_str());
    });

    // Start server
    server.begin();
}

void loop() {
    unsigned long currentMillis = millis();
    if (currentMillis - previousMillis >= interval) {
        // save the last time you updated the DHT values
        previousMillis = currentMillis;
        // Read temperature as Celsius (the default)
        float newT = dht.readTemperature();
        // Read temperature as Fahrenheit (isFahrenheit = true)
        //float newT = dht.readTemperature(true);
        // if temperature read failed, don't change t value
        if (isnan(newT)) {
            Serial.println("Failed to read from DHT sensor!");
        }
        else {
            t = newT;
            Serial.println(t);
        }
        // Read Humidity
        float newH = dht.readHumidity();

```

```
// if humidity read failed, don't change h value
if (isnan(newH)) {
    Serial.println("Failed to read from DHT sensor!");
}
else {
    h = newH;
    Serial.println(h);
}
}
```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module4/Unit10/Access_Point_AP_Web_Server/Access_Point_AP_Web_Server.ino

Customize the SSID and Password

You need to define a SSID name and a password to access the ESP8266. We're setting the ESP8266 SSID name to **ESP8266-Access-Point**, but you can modify the name to whatever you want. The password is **123456789**, but you can also modify it.

```
const char* ssid      = "ESP8266-Access-Point";
const char* password = "123456789";
```

Setting the ESP8266 as an Access Point (AP)

There's a section in the `setup()` to set the ESP8266 as an access point using the `softAP()` method:

```
WiFi.softAP(ssid, password);
```

There are also other optional parameters you can pass to the `softAP()` method. Here's all the parameters:

```
.softAP(const char* ssid, const char* password, int channel, int
ssid_hidden, int max_connection)
```

- `ssid` (defined earlier): maximum of 31 characters.

- `password` (defined earlier): minimum of 8 characters. If not specified, the access point will be open (maximum 63 characters).
- `channel`: Wi-Fi channel number (1-13). Default is 1.
- `ssid_hidden`: if set to true will hide SSID.
- `max_connection`: max simultaneous connected stations, from 0 to 8.

Next, get the access point IP address using the `softAPIP()` method and print it in the Serial Monitor.

```
IPAddress IP = WiFi.softAPIP();
Serial.print("AP IP address: ");
Serial.println(IP);
```

Note: by default, the access point IP address is 192.168.4.1

These are the snippets of code you need to include in your web server sketches to set the ESP8266 as a soft access point.

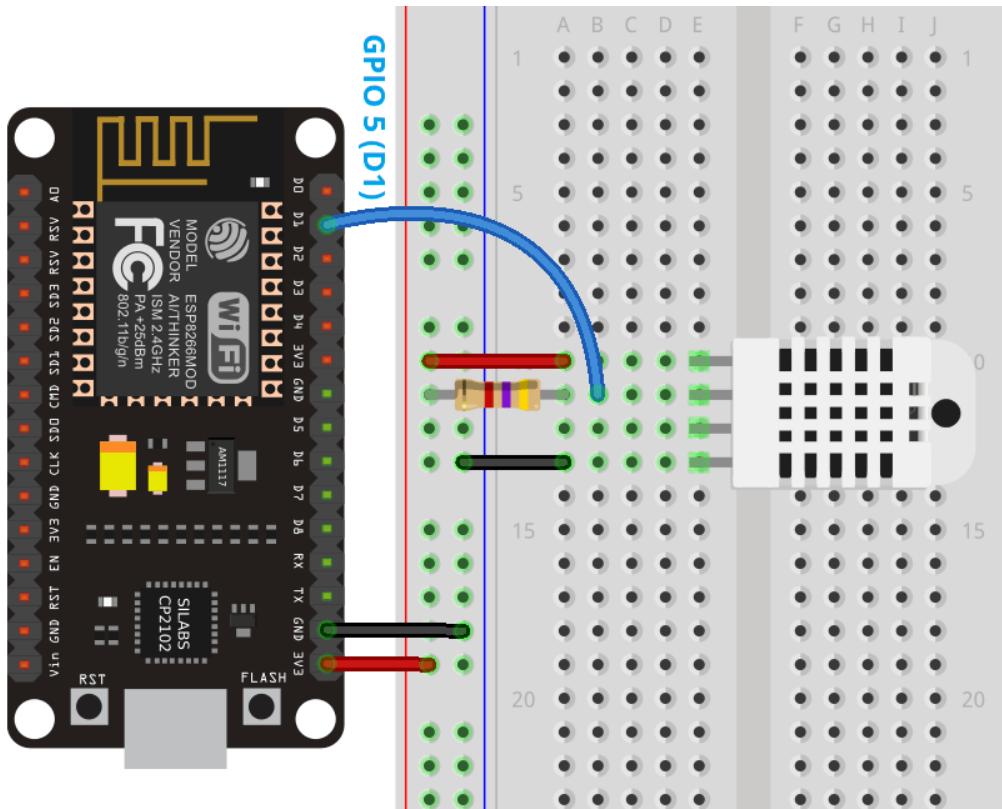
Parts Required

To proceed with this tutorial, you need the following parts:

- [ESP8266](#)
- [DHT22](#) or [DHT11](#) Temperature and Humidity Sensor
- [4.7k Ohm Resistor](#)
- [Breadboard](#)
- [Jumper wires](#)

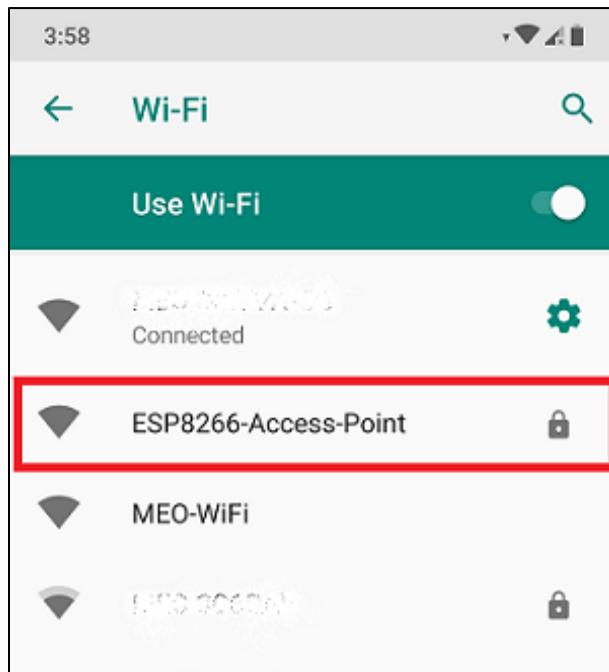
Schematic Diagram

Assemble all the parts by following the next schematic diagram:

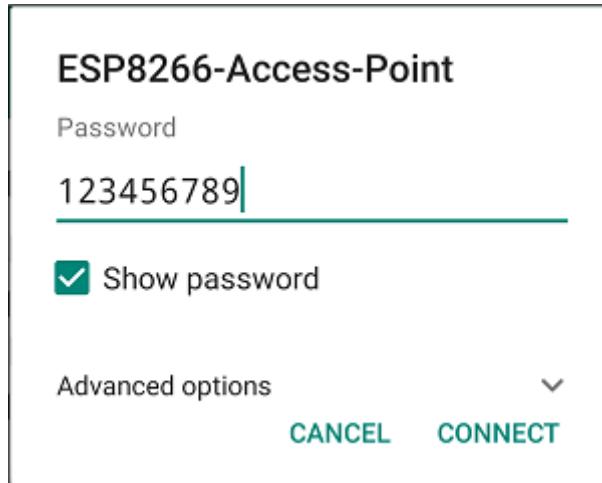


Connecting to the ESP8266 Access Point

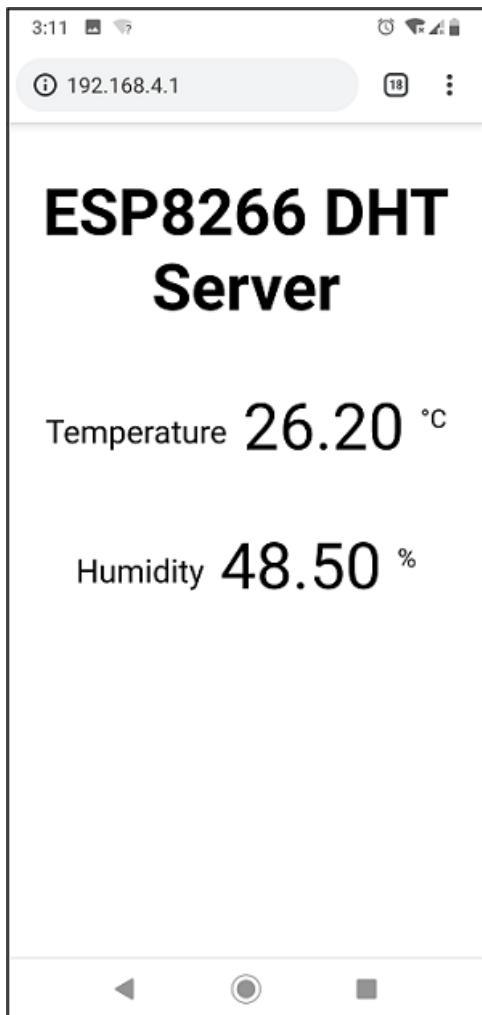
Having the ESP8266 running the sketch, in your smartphone open your Wi-Fi settings and tap the ESP8266-Access-Point network:



Enter the password you've defined earlier.



Open your web browser and type the IP address <http://192.168.4.1>. The web server page should load:



The web server page is a bit different from the [original web server](#). That web server displays two icons next to the temperature and humidity. Those icons are loaded from the Font Awesome website. However, because the ESP8266 acts a soft access point (it is not connected to the internet) we cannot load those icons.

Note: when you set the ESP8266 as an access point, it is not connected to the internet, so you can't make HTTP requests to other services to publish sensor data or to get data from the internet (like loading the icons).

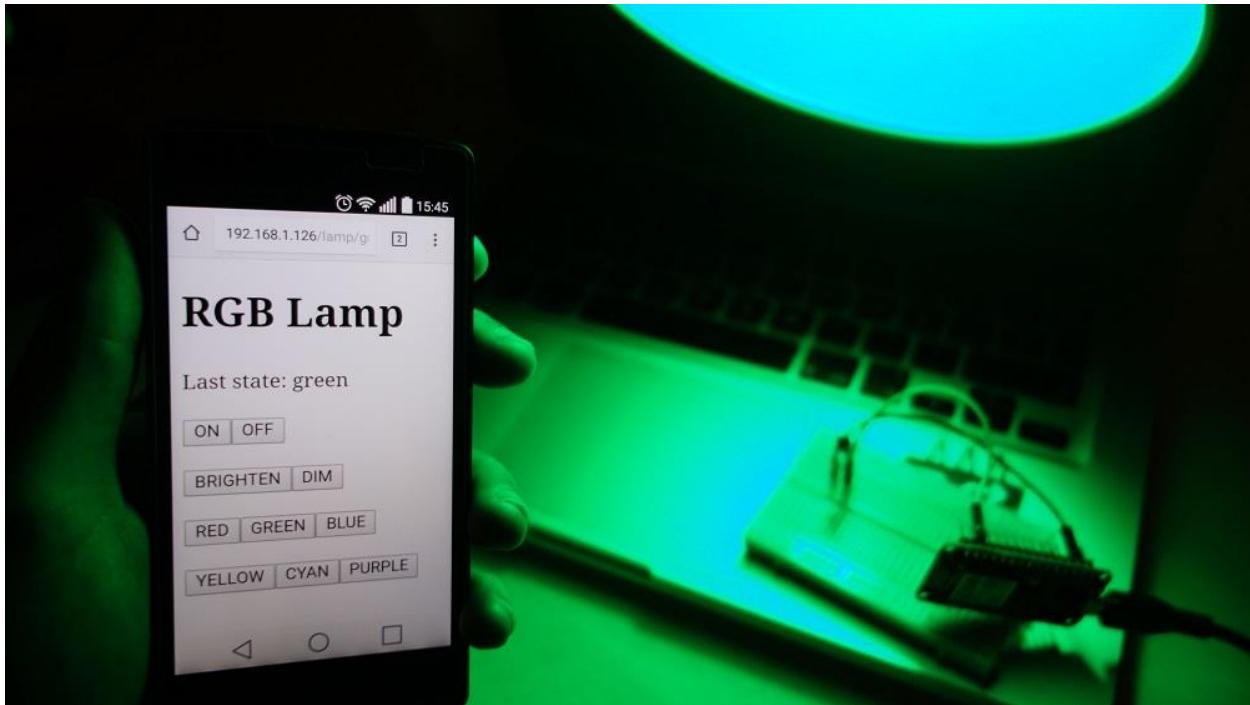
MODULE 5:

ESP8266 IoT and Home

Automation Projects

This Module contains miscellaneous IoT and Home Automation projects using the ESP8266. For example, you'll learn how to build a weather forecaster, control RGB LED lamps remotely, create a dash button clone to execute any task, datalogging, and much more.

Unit 1: Infrared RGB LED Lamp Controller with ESP8266

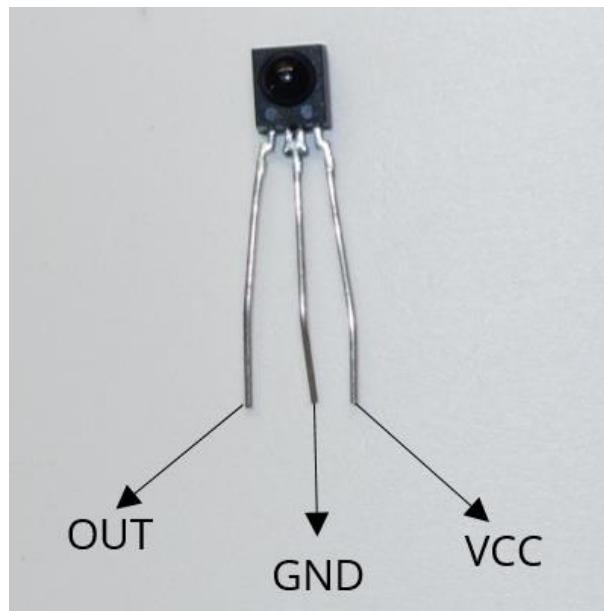


In this project you'll use an infrared (IR) emitter to control an RGB LED lamp with an ESP8266 web server. You'll send ON, OFF, DIM, BRIGHT, RED, GREEN, BLUE, PURPLE and other commands. This project is divided into two parts:

- 1) You'll decode the IR signals transmitted by your lamp's remote control;
- 2) You'll send those IR signals with your ESP8266 to control the lamp remotely using a web server.

IR Receiver and Emitter

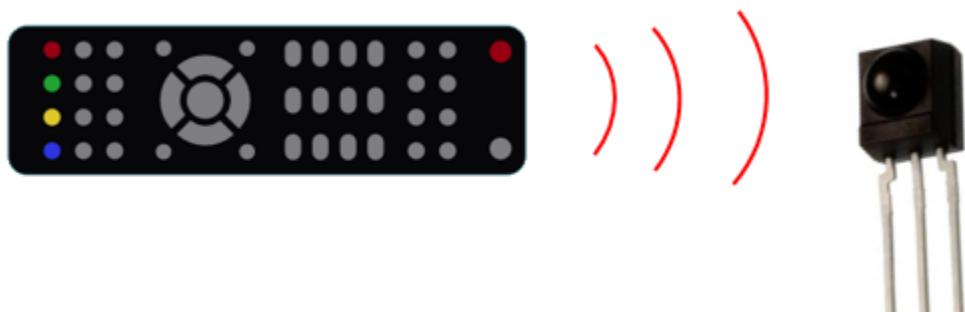
The following figure shows an infrared receiver (TSOP4838).



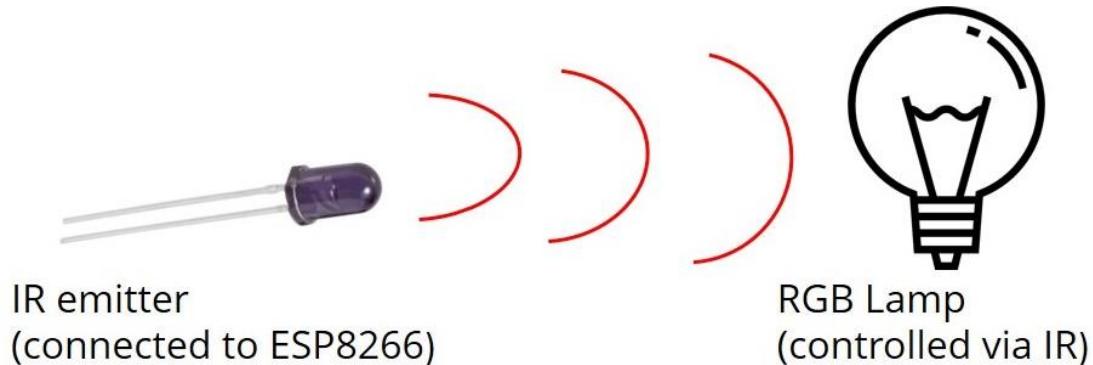
The receiver has three pins:

- OUT (OUTPUT);
- GND (Ground);
- VCC (Voltage source);

When you press your remote control, it sends infrared modulated signals. These signals contain information that can be collected by your receiver.



Each button of your remote control sends a specific IR signal. So, we can assign each signal to a specific ESP8266 web server button. That signal can then be sent by an IR emitter (connected to the ESP8266) to control the lamp.



Parts Required

For this project you'll need [E27/B22 IR RGB LED Lamp with remote control](#).

If you don't want to control an IR RGB lamp, you can control any other device that uses an IR remote (like your thermostat, TV, etc...).

- [ESP8266](#)
- [TSOP4838 IR receiver](#)
- [IR LED transmitter](#)
- [E27/B22 IR RGB LED Lamp with remote control](#)
- [Breadboard](#)
- [Jumper wires](#)

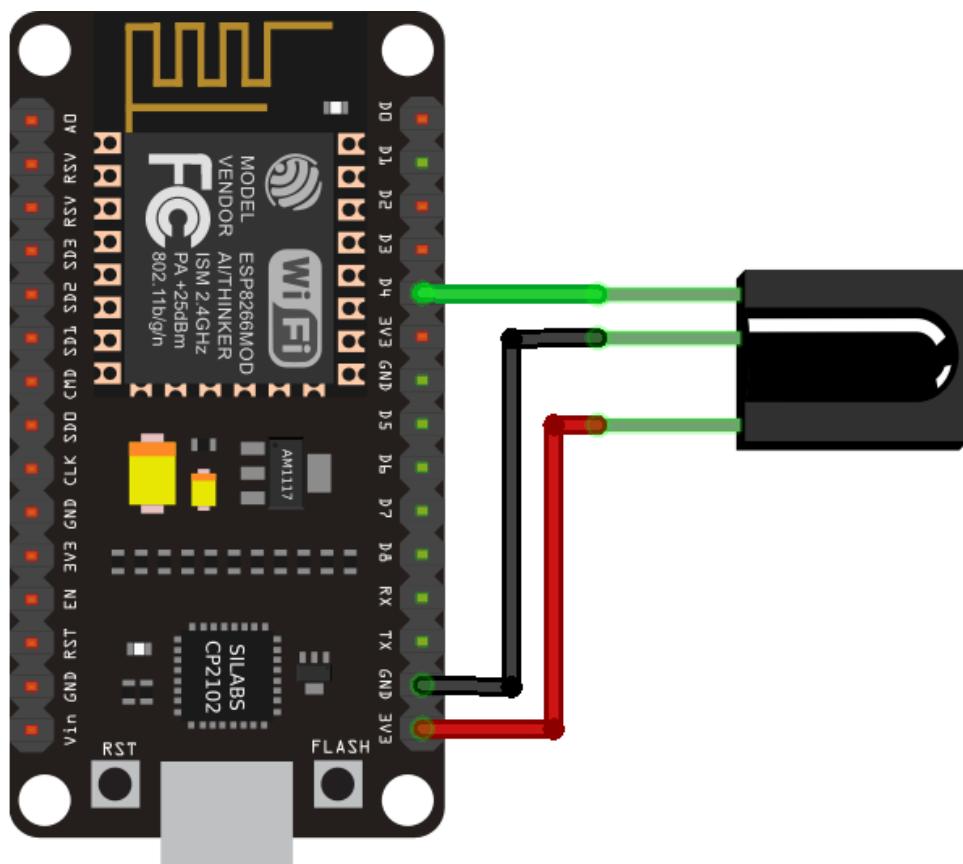


1. Decode IR signals

In this part of the project you need to decode the IR signals of the buttons you want to emulate with your ESP8266.

Schematic

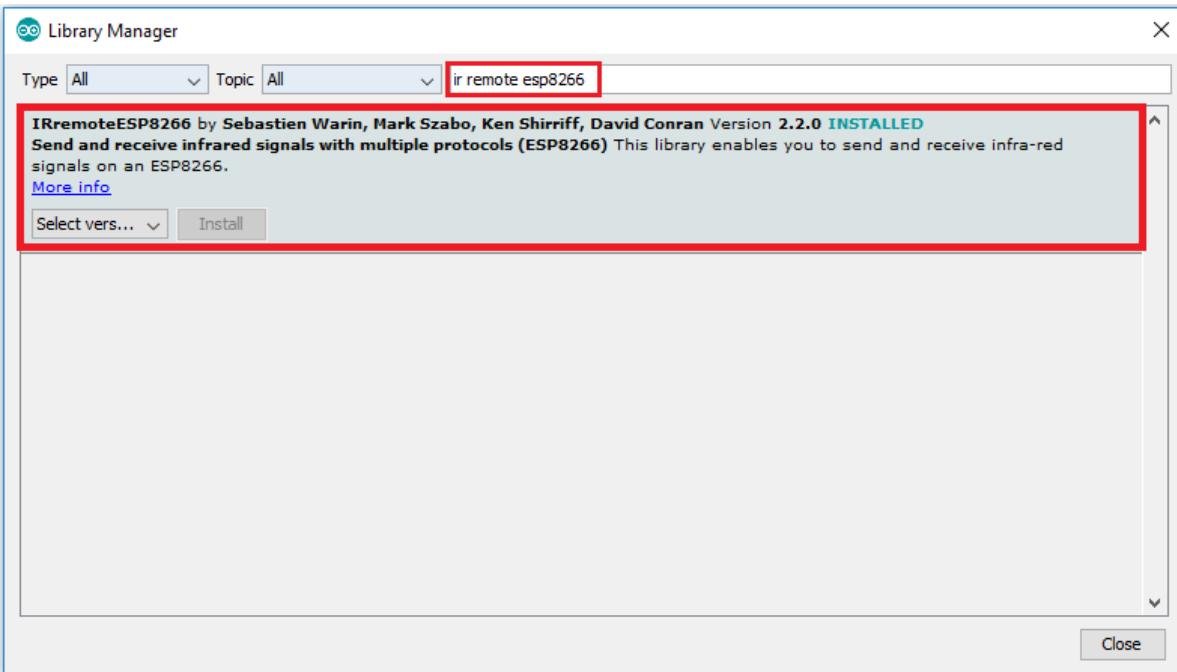
Connect the IR receiver accordingly to the following schematic diagram, with the data pin connected to GPIO 2.



Code

To control the IR receiver and emitter, you need to install the [IRremoteESP8266 Library](#) in the Arduino IDE. Go to **Sketch > Include Library > Manage Libraries...**

Then, search for **ir remote esp8266** (or [download here](#)). Press the “**Install**” button.



Copy the following code to your Arduino IDE, and upload it to your ESP8266 board.

Make sure that you have the right board and COM port selected.

```
#ifndef UNIT_TEST
#include <Arduino.h>
#endif
#include <IRremoteESP8266.h>
#include <IRrecv.h>
#include <IRutils.h>

// An IR detector/demodulator is connected to GPIO pin 14 (D5 on a NodeMCU
// board).
uint16_t RECV_PIN = 14;

IRrecv irrecv(RECV_PIN);

decode_results results;

void setup() {
  Serial.begin(115200);
  irrecv.enableIRIn(); // Start the receiver
}
void loop() {
  if (irrecv.decode(&results)) {
    // print() & println() can't handle printing long longs. (uint64_t)
    serialPrintInt64(results.value, HEX);
    Serial.println("");
    irrecv.resume(); // Receive the next value
  }
  delay(100);
}
```

SOURCE CODE

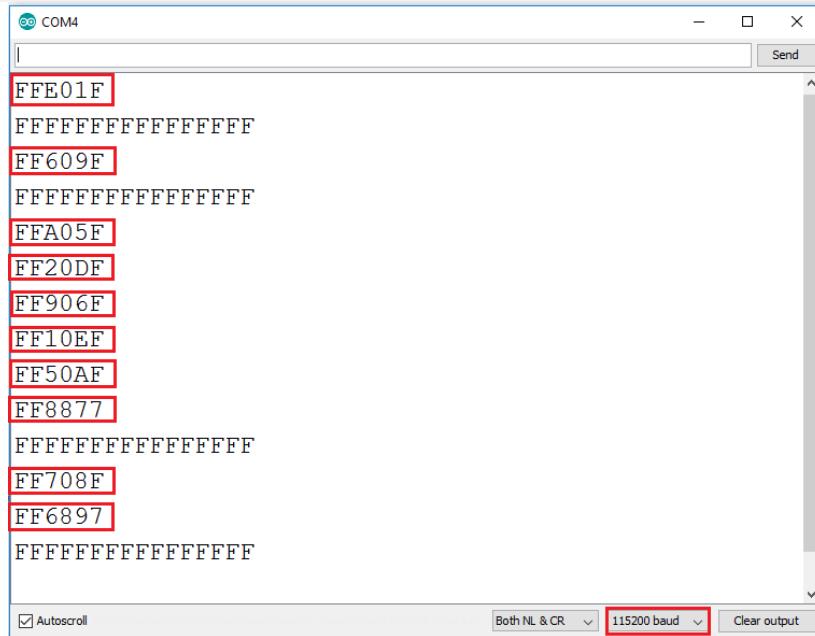
https://github.com/RuiSantosdotme/ESP8266-eBook/tree/master/Code/PART1_Arduino/Module5/Unit1/IR_Receiver

Open the serial monitor at a baud rate of 115200.

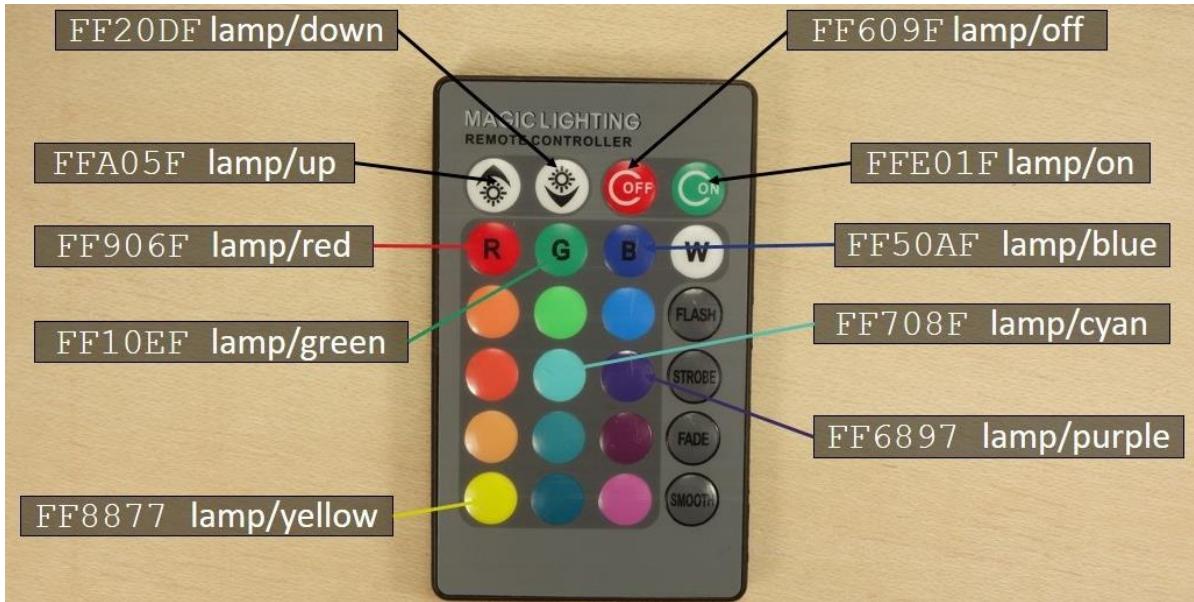


In this project, we'll emulate 10 buttons of the remote control. Press the "ON" button. You should see a code in the serial monitor.

Note: you should press the same button several times to make sure you have the right code for that button. If you see something like FFFFFFFF ignore it, it's trash.



Repeat the same process for the other buttons. Write down the code associated with each button, because you'll need that information later. Below you can find the IR signals associated with each button press for this remote.

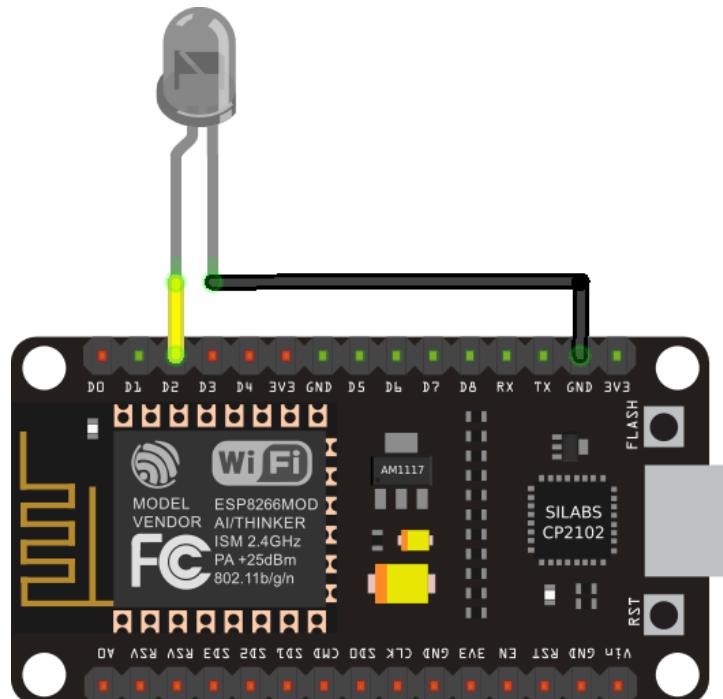


2. Building the Final Circuit

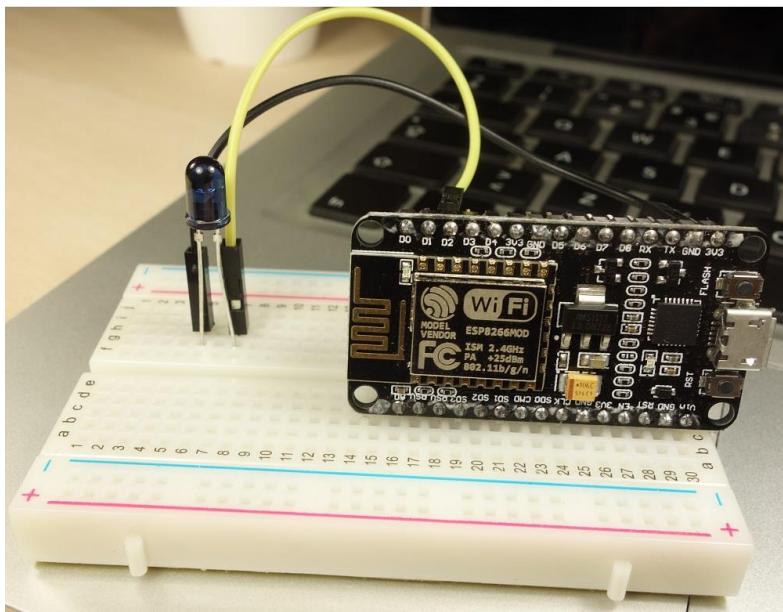
In this section, you'll build the circuit with an IR LED emitter connected to your ESP8266 to send IR signals to control your lamp.

Schematic

Connect the IR emitter to the ESP8266 by following the next schematic diagram.



Here's how the circuit looks like.



You should place the emitter next to the lamp, so it can receive the IR signals.

Code

The code runs a web server on your ESP's port 80. The web server displays a simple web page with 10 buttons to control the RGB LED lamp.

A screenshot of a web browser window titled "192.168.1.126/lamp/on". The page content is as follows:

RGB Lamp

Last state: on

ON **OFF**

BRIGHTEN **DIM**

RED **GREEN** **BLUE**

YELLOW **CYAN** **PURPLE**

Copy the next sketch to your Arduino IDE, but don't upload it, yet. Keep reading the next section to see what you need to change to make it work for you:

```
#include <ESP8266WiFi.h>

#ifndef UNIT_TEST
#include <Arduino.h>
#endif
#include <IRremoteESP8266.h>
#include <IRsend.h>

// An IR LED is controlled by GPIO pin 4 (D2)
IRsend irsend(4);

// REPLACE WITH YOUR SSID AND PASSWORD
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

// Stores last button pressed on web server
String lastState;

// Create an instance of the server on port 80
WiFiServer server(80);

void setup() {
    irsend.begin();

    Serial.begin(115200, SERIAL_8N1, SERIAL_TX_ONLY);
    delay(10);

    // prepare GPIO2
    pinMode(4, OUTPUT);
    digitalWrite(4, 0);

    // Connect to WiFi network
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("WiFi connected");

    // Start the server
    server.begin();
    Serial.println("Server started");

    // Print the IP address
    Serial.println(WiFi.localIP());
}
```

```

void loop() {
    // Check if a client has connected
    WiFiClient client = server.available();
    if (!client) {
        return;
    }

    // Wait until the client sends some data
    Serial.println("new client");
    while(!client.available()) {
        delay(1);
    }

    // Read the first line of the request
    String req = client.readStringUntil('\r');
    Serial.println(req);
    client.flush();

    // Check which request was triggered
    int irCode;
    if (req.indexOf("/lamp/on") != -1) {
        irCode = 0xFFE01F;
        lastState = "on";
    }
    else if (req.indexOf("/lamp/off") != -1) {
        irCode = 0xFF609F;
        lastState = "off";
    }
    else if (req.indexOf("/lamp/brighten") != -1) {
        irCode = 0xFFA05F;
        lastState = "brighten";
    }
    else if (req.indexOf("/lamp/dim") != -1) {
        irCode = 0xFF20DF;
        lastState = "dim";
    }
    else if (req.indexOf("/lamp/red") != -1) {
        irCode = 0xFF906F;
        lastState = "red";
    }
    else if (req.indexOf("/lamp/green") != -1) {
        irCode = 0xFF10EF;
        lastState = "green";
    }
    else if (req.indexOf("/lamp/blue") != -1) {
        irCode = 0xFF50AF;
        lastState = "blue";
    }
    else if (req.indexOf("/lamp/yellow") != -1) {
        irCode = 0xFF8877;
        lastState = "yellow";
    }
    else if (req.indexOf("/lamp/cyan") != -1) {
        irCode = 0xFF708F;
    }
}

```

```

        lastState = "cyan";
    }
    else if (req.indexOf("/lamp/purple") != -1) {
        irCode = 0xFF6897;
        lastState = "purple";
    }
    else {
        lastState = "none";
        Serial.println("Invalid request - IR signal not sent");
    }
    // Send IR signal to RGB LED lamp
    Serial.print("NEC: ");
    Serial.println(irCode, HEX);
    irsend.sendNEC(irCode, 32);
    client.flush();

    // Prepare the client response (contains web page)
    String s = "HTTP/1.1 200 OK\r\nContent-Type:
text/html\r\n\r\n<!DOCTYPE HTML>\r\n<html>";
    s += "<head><meta name=\"viewport\" content=\"width=device-width,
initial-scale=1\"></head>";
    s += "<h1>RGB Lamp</h1>Last state: ";
    s += lastState;
    s += "<p><a href=\"/lamp/on\"><button>ON</button></a><a
href=\"/lamp/off\"><button>OFF</button></a></p>";
    s += "<p><a href=\"/lamp/brighten\"><button>BRIGHTEN</button></a><a
href=\"/lamp/dim\"><button>DIM</button></a></p>";
    s += "<p><a href=\"/lamp/red\"><button>RED</button></a><a
href=\"/lamp/green\"><button>GREEN</button></a><a
href=\"/lamp/blue\"><button>BLUE</button></a></p>";
    s += "<p><a href=\"/lamp/yellow\"><button>YELLOW</button></a><a
href=\"/lamp/cyan\"><button>CYAN</button></a><a
href=\"/lamp/purple\"><button>PURPLE</button></a></p>";
    s += "</html>\n";

    // Send the response to the client and stop client
    client.print(s);
    delay(1);
    Serial.println("Client disconnected");
    client.stop();
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module5/Unit1/IR_Lamp_Sender/IR_Lamp_Sender.ino

Preparing your sketch

Before uploading the sketch, you need to add your SSID and password:

```
// REPLACE WITH YOUR SSID AND PASSWORD
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Now, grab the codes you captured in the previous part and use them in your sketch.

Replace all the `irCode` variables with your own IR signal codes:

```
// Check which request was triggered
int irCode;
if (req.indexOf("/lamp/on") != -1) {
    irCode = 0xFFE01F;
    lastState = "on";
}
else if (req.indexOf("/lamp/off") != -1) {
    irCode = 0xFF609F;
    lastState = "off";
}
else if (req.indexOf("/lamp/brighten") != -1) {
    irCode = 0xFFA05F;
    lastState = "brighten";
}
else if (req.indexOf("/lamp/dim") != -1) {
    irCode = 0xFF20DF;
    lastState = "dim";
}
(...)
```

Continue adding the IR signals to all the other `irCode` variables. You can easily add more buttons if you create other **if... else** statements in your code.

The web page is simple HTML text to build buttons.

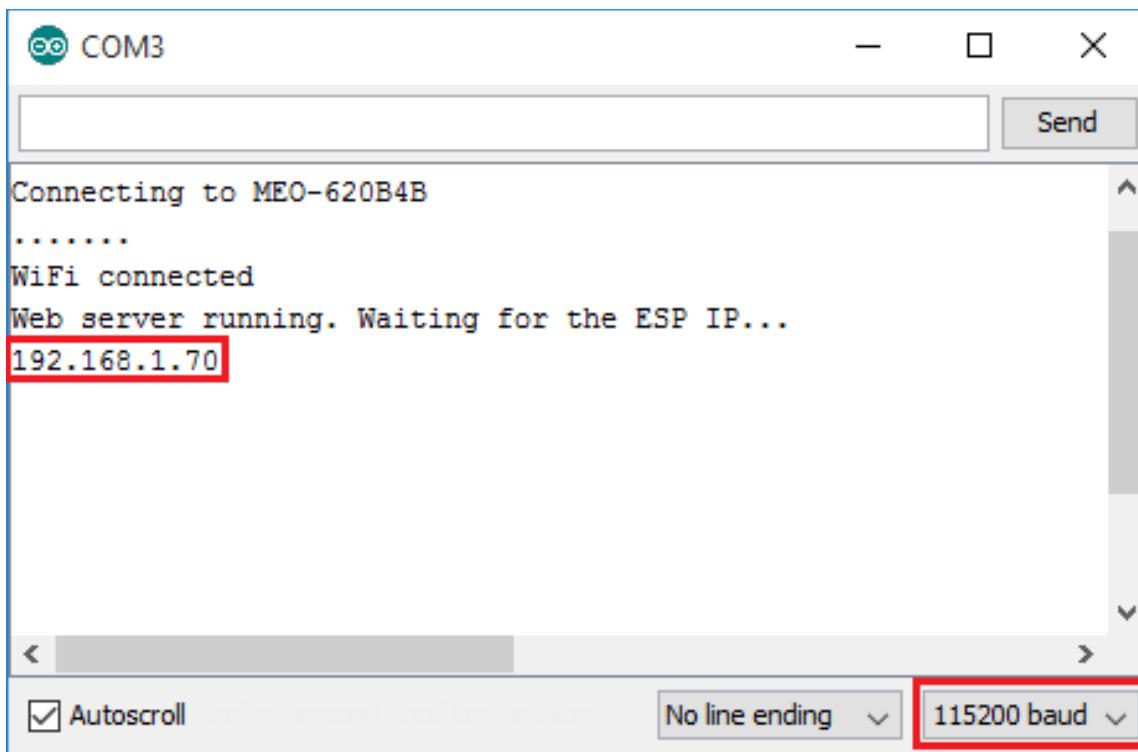
```
String s = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n<!DOCTYPE
HTML>\r\n<html>";
s += "<head><meta name=\"viewport\" content=\"width=device-width,
initial-scale=1\"></head>";
s += "<h1>RGB Lamp</h1>Last state: ";
s += lastState;
s += "<p><a href=\"/lamp/on\"><button>ON</button></a><a
href=\"/lamp/off\"><button>OFF</button></a></p>";
s += "<p><a href=\"/lamp/brighten\"><button>BRIGHTEN</button></a><a
href=\"/lamp/dim\"><button>DIM</button></a></p>";
```

```
s      +=    "<p><a href=\"/lamp/red\"><button>RED</button></a><a href=\"/lamp/green\"><button>GREEN</button></a><a href=\"/lamp/blue\"><button>BLUE</button></a></p>";  
s      +=    "<p><a href=\"/lamp/yellow\"><button>YELLOW</button></a><a href=\"/lamp/cyan\"><button>CYAN</button></a><a href=\"/lamp/purple\"><button>PURPLE</button></a></p>";  
s += "</html>\n";
```

Once you've made all the changes to your script, upload the sketch to your ESP8266.

ESP8266 IP Address

After uploading your sketch to the ESP8266, go to **Tools > Serial Monitor**. In your Serial Monitor window you'll see your ESP IP address when your ESP first boots or press the RST button.



Our IP address is **192.168.1.70** (as shown in the preceding figure). Your IP should be different, save your ESP8266 IP so that you can access the web server to control the lamp.

Demonstration

Open a web browser in any device on the same network your ESP is, and enter the ESP8266 IP address.

With the IR LED emitter pointing to the RGB LED lamp, you can tap the buttons to control the lamp: change color, control brightness, turn on/off, etc.



Wrapping Up

This is a great project to learn how to integrate the IR receiver and emitter in your ESP8266 projects. There are endless possibilities for what you can do with it. For example, instead of controlling an IR RGB LED lamp, you can control a TV, AC, thermostat, RGB LED strip, or any other device that it's controlled with an IR remote.

Unit 2: Weather Station with OLED Display



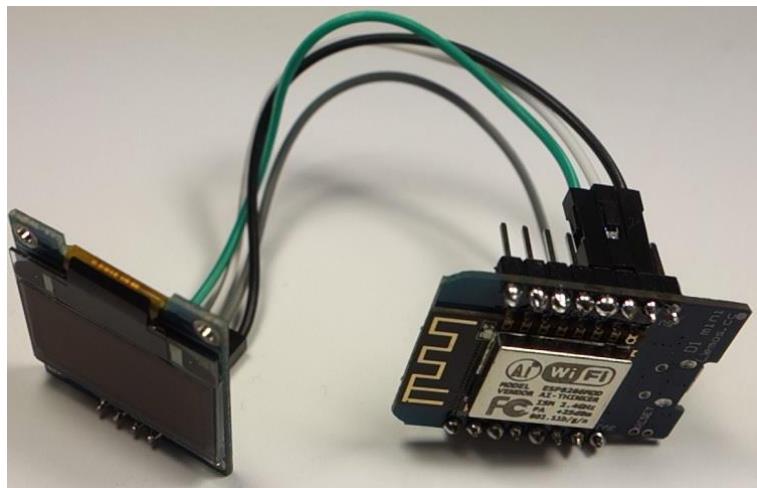
In this project, you're going to build a simple weather station with an ESP8266 and an OLED display. The weather station shows the current time, date, day of the week, current temperature and weather forecast.

Parts Required

For this project, we'll be using the [WeMos D1 Mini](#) ESP8266 board due to its small form factor. However, you can use any other ESP8266 board. We're also going to use a small [I2C OLED display SSD1306](#).

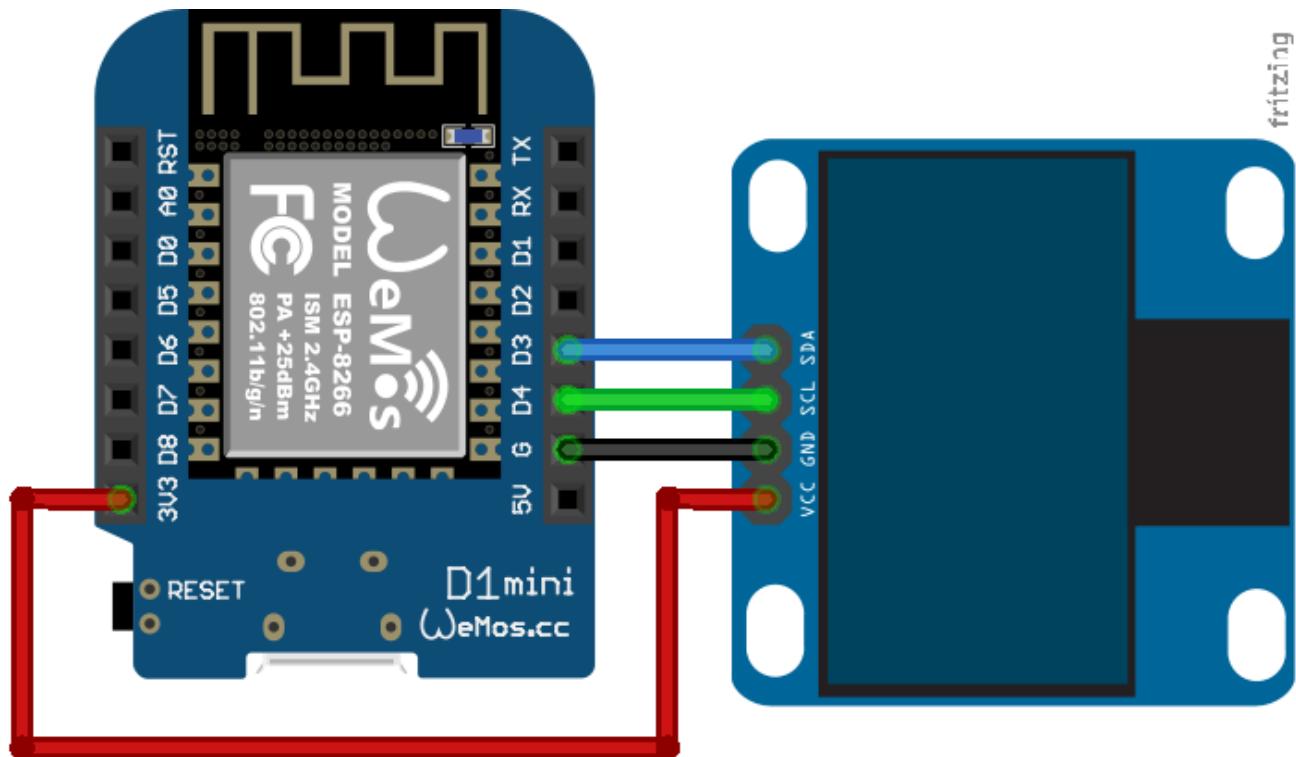
Schematic

The wiring is very straightforward, and it doesn't require any soldering, you can use female to female jumper wires to make all the connections.



Simply follow the next diagram:

- VCC – 3V3
- GND – G
- SCL – D4
- SDA – D3



ESP8266 Weather Station Library/Framework

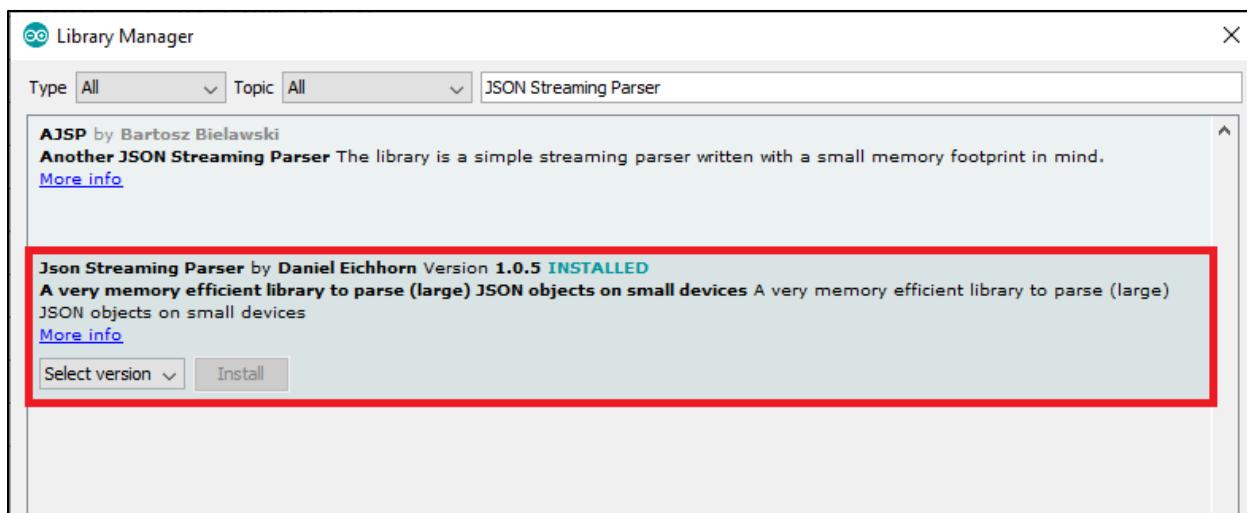
There's a framework on [GitHub](#) that allows you to easily build a weather station with an ESP8266.

It's also available in your Arduino IDE, just go to **Sketch > Include Library > Manage Libraries...**

Then, search for **esp8266 weather station** (or [download here](#)) and select the library by **ThingPulse**. Then, press the “Install” button.

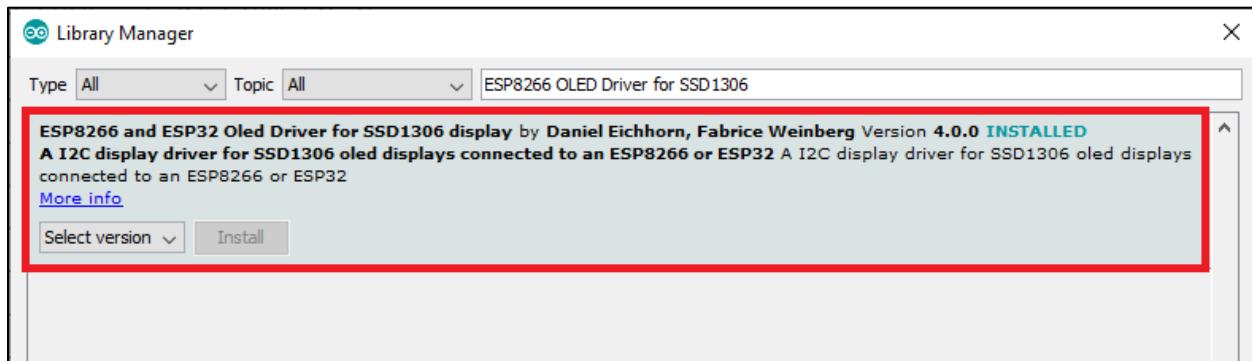


You'll also have to install two additional libraries. Start by installing the **json streaming parser** ([GitHub](#)):



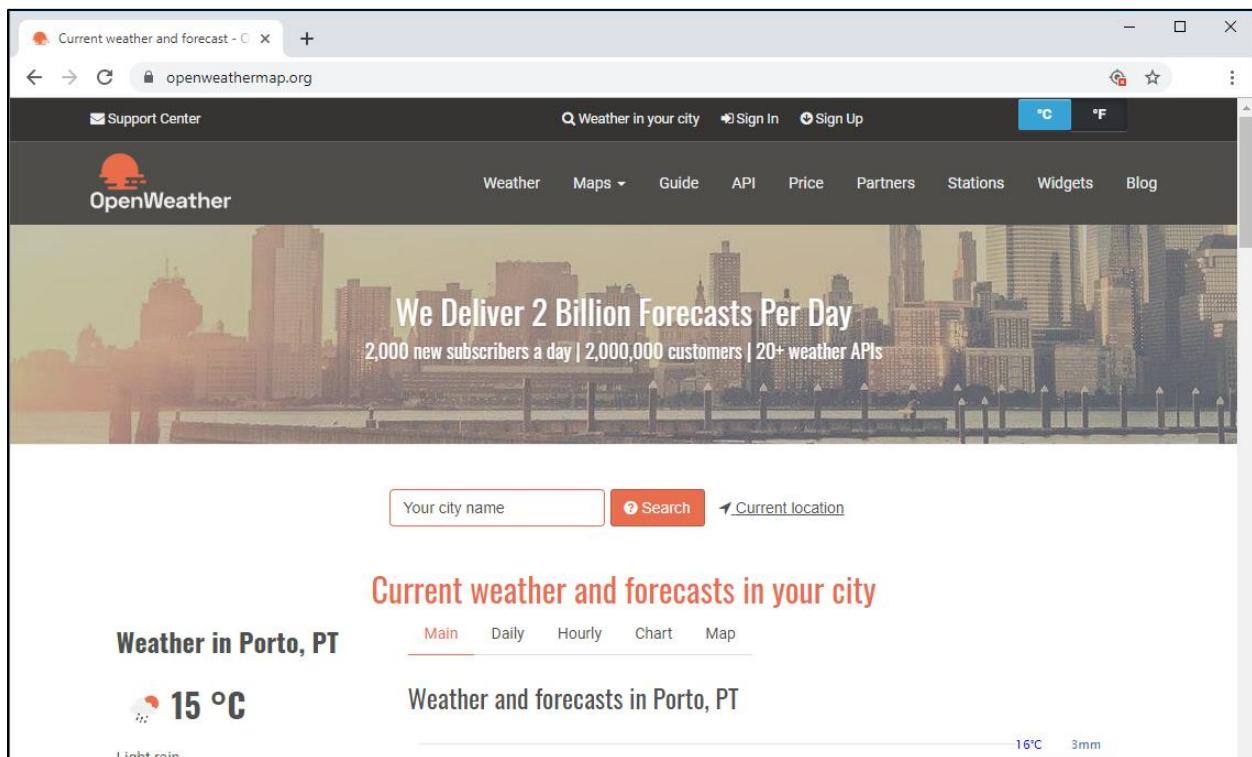
The weather station library also uses a specific OLED library. It won't work with the Adafruit libraries you've installed previously. So, you need to install this additional OLED library.

Search for **esp8266 oled driver for ssd1306** library and install it.



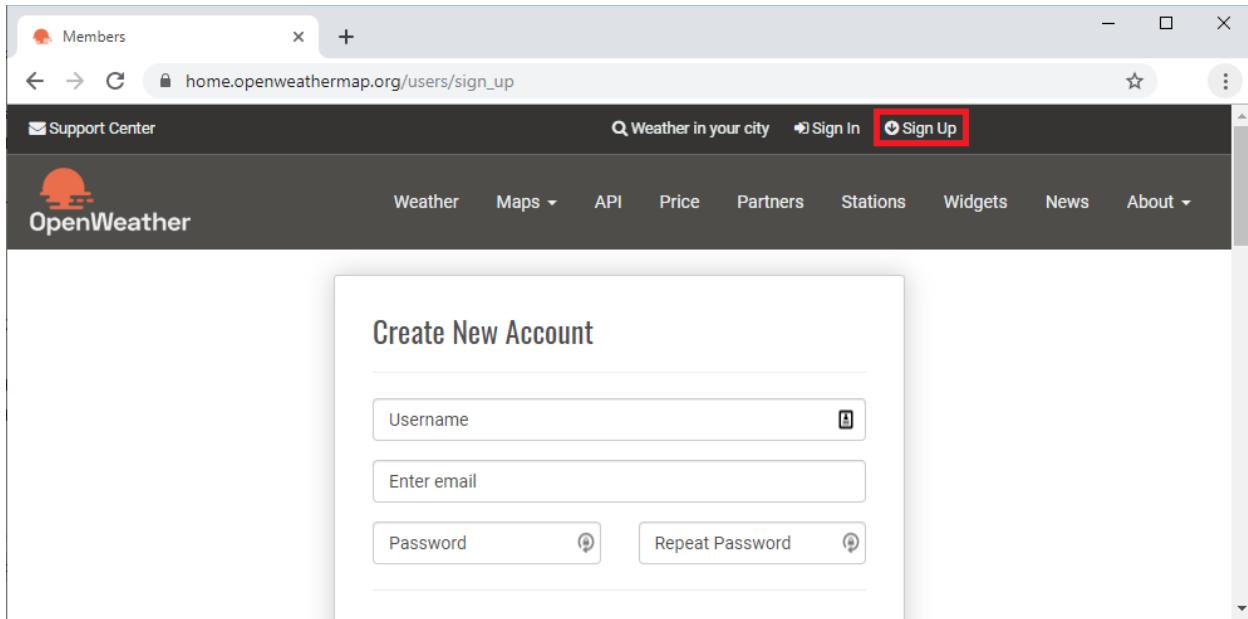
OpenWeatherMap API

The ESP8266 Weather Station library integrates with a service called OpenWeatherMap that has an API that you can use to request the weather for your location.



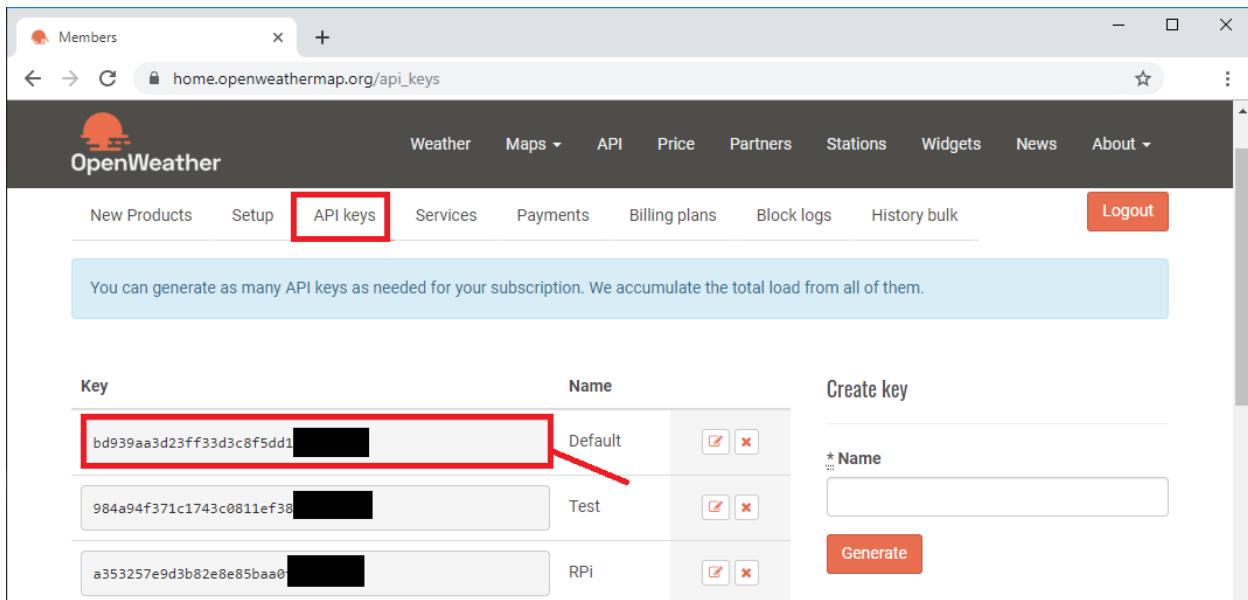
Open a browser and go to openweathermap.org.

Create a new account if you don't have one.



After that, login and you'll be presented with a dashboard that contains several tabs.

In your account settings, select the **API keys** tab.



On the API keys tab, you'll see a default key as shown in the previous figure. This is a unique key you'll need to pull information from the site. Copy and paste this key somewhere; you'll need it in a moment.

Then, go to: openweathermap.org/find?q and search for your location.

The screenshot shows a web browser window titled 'Find - OpenWeatherMap'. The URL in the address bar is 'openweathermap.org/find?q'. The page header includes the OpenWeather logo and navigation links for Weather, Maps, Guide, API, Price, Partners, Stations, Widgets, and Blog. A temperature selector between '°C' and '°F' is also present. The main content area has a red background and displays the heading 'Weather in your city'. Below this, there is a search bar with 'Porto, Portugal' entered and a 'Search' button. Two weather forecast entries are shown: one for Porto, PT (moderate rain) with a temperature of 13.9°C and another for Porto, BR (clear sky) with a temperature of 31.2°C. Each entry includes a small icon, the city name, country code, weather condition, current temperature, and a detailed description of current conditions and coordinates.

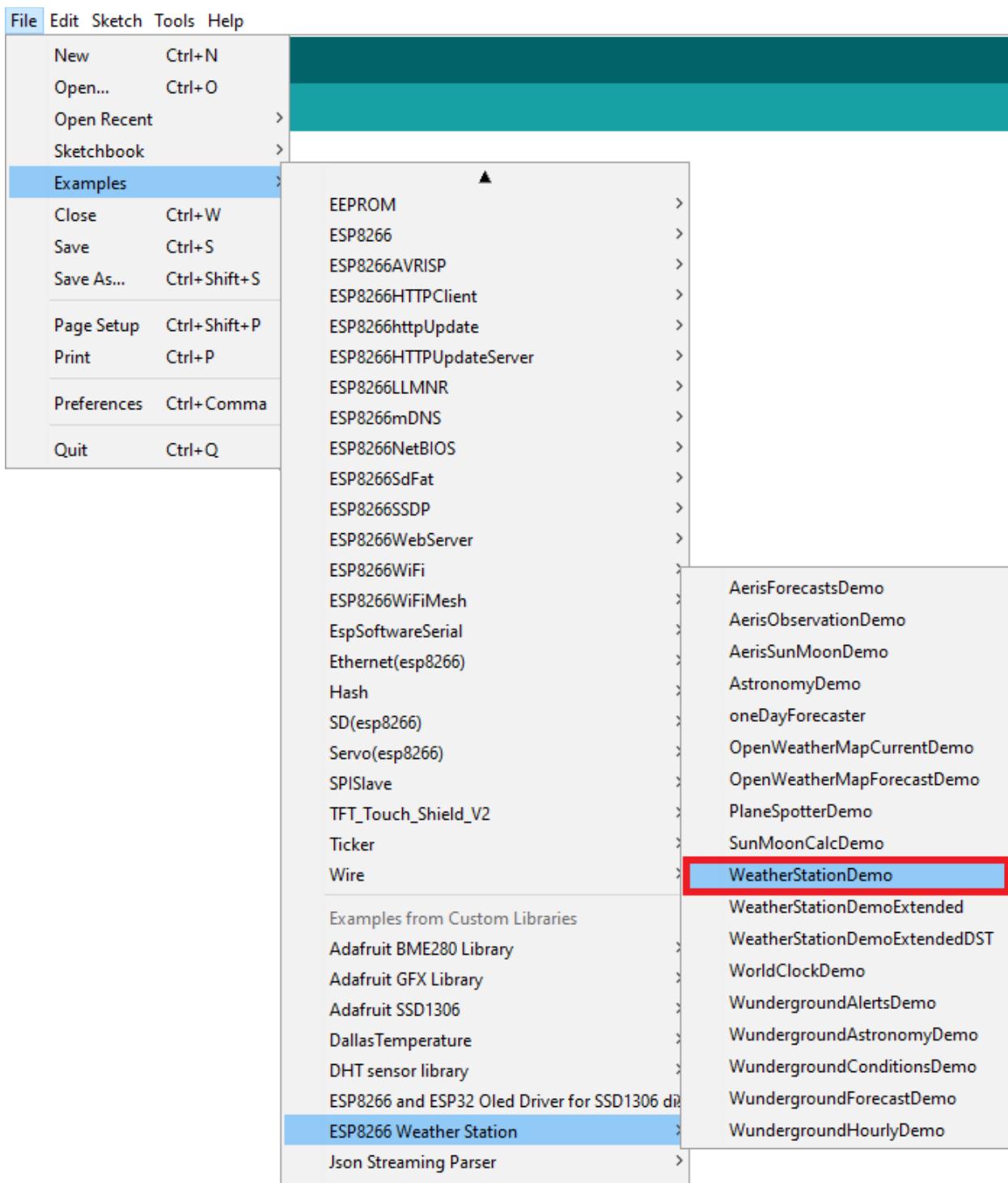
Click on your desired location. A page as shown in the following image should load.

The screenshot shows a web browser window titled 'Weather forecast - OpenWeather'. The URL in the address bar is 'openweathermap.org/city/2735943', with the city ID highlighted by a red box. The page header and navigation links are identical to the previous screenshot. The main content area features a large search bar with 'Your city name' and a 'Search' button. Below it, a link to 'Current location' is visible. The heading 'Current weather and forecasts in your city' is displayed in red. Underneath, the section 'Weather in Porto, PT' is shown. It includes a large temperature reading of '14 °C' with a weather icon, followed by the text 'Moderate rain'.

Copy the number for your city on the URL tab. You'll need it later.

Weather Station Demo

The ESP8266 Weather Station library comes with some useful examples. In your Arduino IDE, go to **File > Examples > ESP8266 Weather Station > WeatherStationDemo** as shown in the next figure.



This example displays the time and date, weather forecast and outdoor temperature and humidity retrieved from the OpenWeatherMap API. You need to modify the code with your own settings to make it work for you.

WiFi settings

Add your network credentials in the following variables.

```
const char* WIFI_SSID = "yourssid";
const char* WIFI_PWD = "yourpassw0rd";
```

Configure time

Modify the following variables to adjust the time in your country.

```
#define TZ          2    // (utc+) TZ in hours
#define DST_MN      60   // use 60mn for summer time in some countries
```

OpenWeatherMap settings

You'll also need to configure the OpenWeatherMap settings, so that your ESP8266 can pull information directly from the weather API.

Add your unique API Key in the following variable:

```
String OPEN_WEATHER_MAP_APP_ID = "XXX";
```

Insert the number for your location:

```
String OPEN_WEATHER_MAP_LOCATION_ID = "2657896";
```

You can also set the weather station language. The code shows the code you need to use for your specified language.

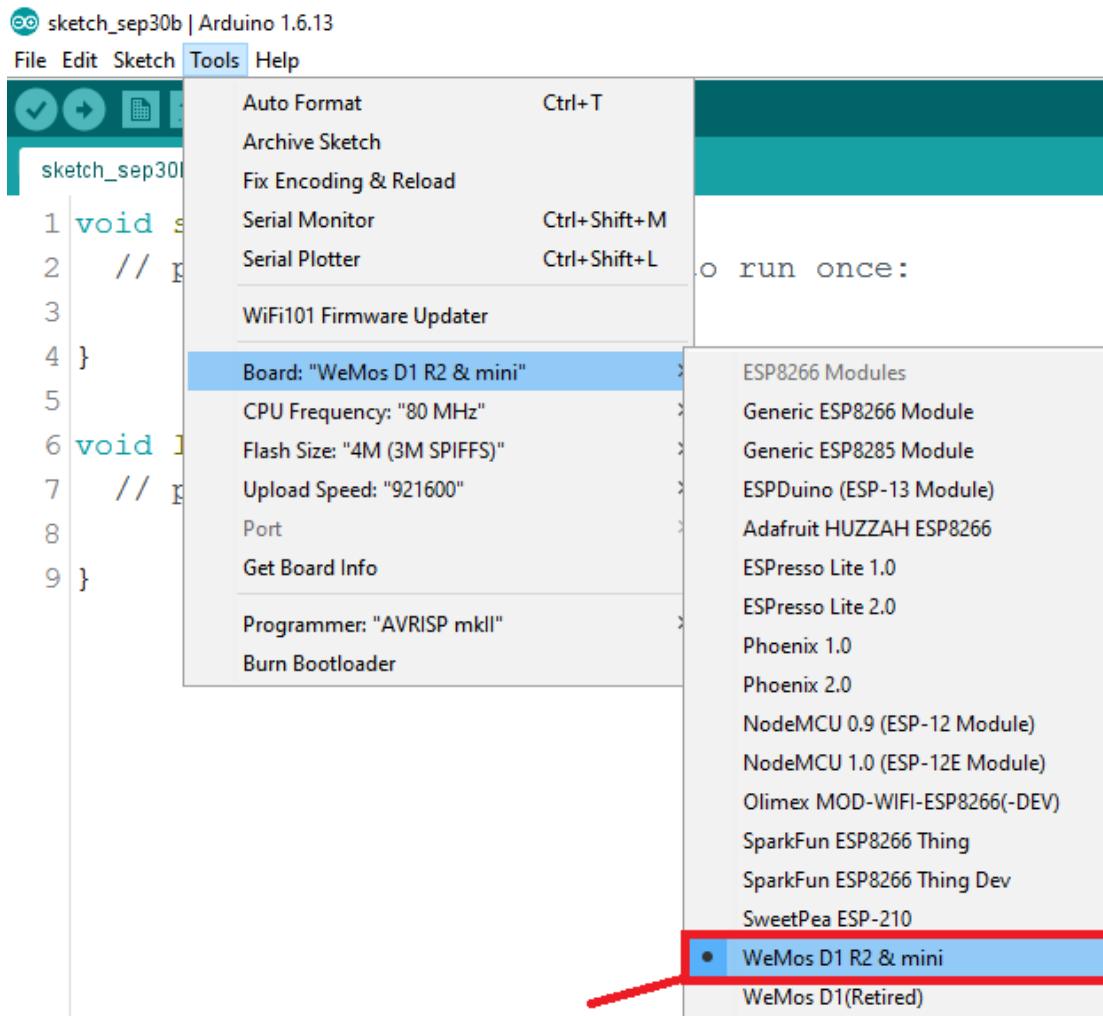
```
String OPEN_WEATHER_MAP_LANGUAGE = "de";
```

If you want to use English, set this variable to `en`, as follows:

```
String OPEN_WEATHER_MAP_LANGUAGE = "en";
```

Code

After adding all your settings, you can upload the code to your ESP8266. Don't forget to select the board "**WeMos D1 R2 & Mini**" and the right COM port.



3D Printed Enclosure

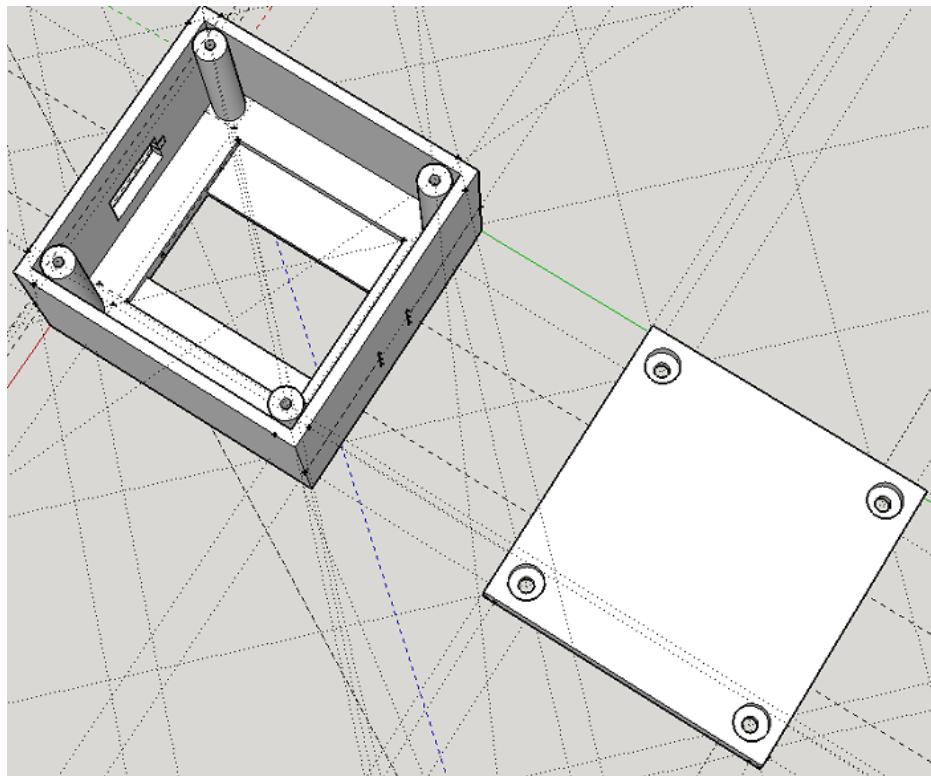
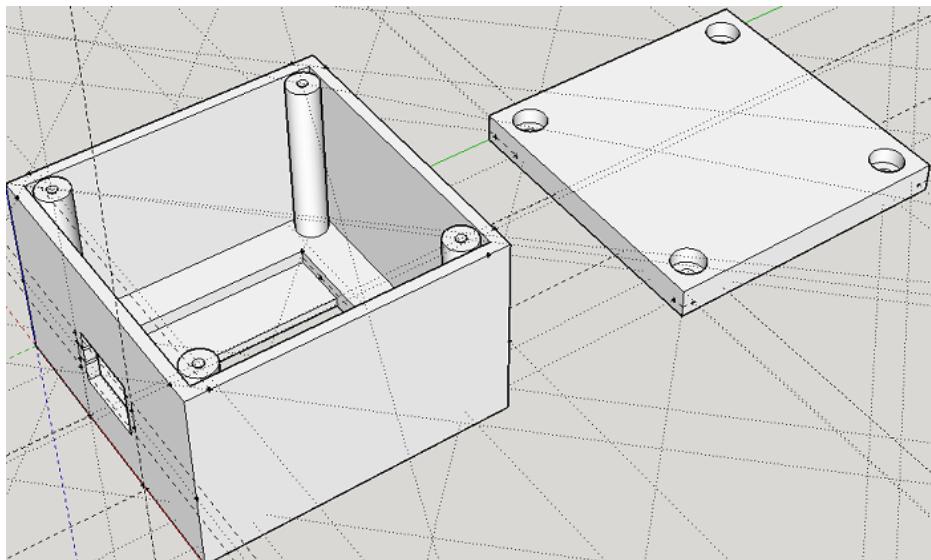
As an optional step, we've 3D printed an enclosure to store the final circuit. We've used [SketchUp](#) to create it, but you can use any other 3D modeling software.

You can click the next links to download the files necessary to edit the 3D model or upload it to your slicer software:

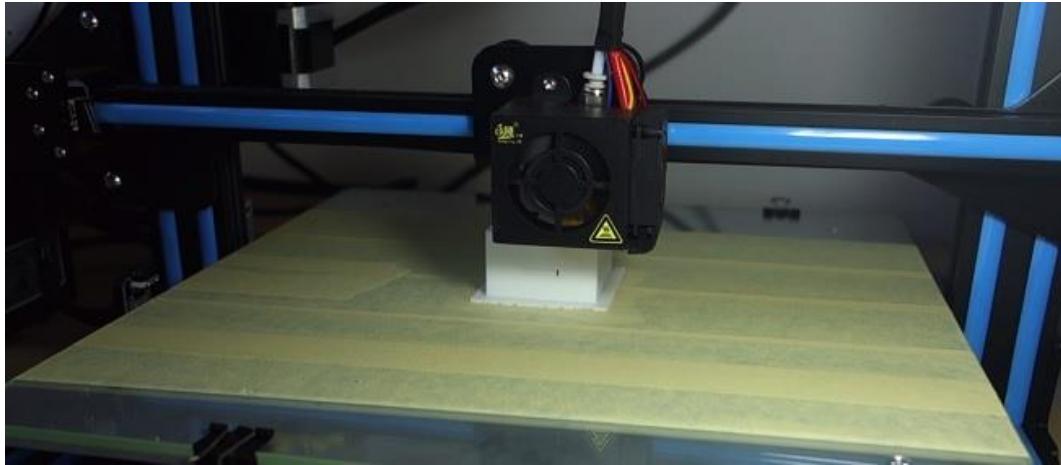
- Complete SketchUp file – Preview [here](#)
- Bottom part STL file – Preview [here](#)
- Top part STL file – Preview [here](#)

[Click here to download all files.](#)

The design is simple: it has a place for the OLED display and a hole on the side where the USB cable can go through to power the circuit.

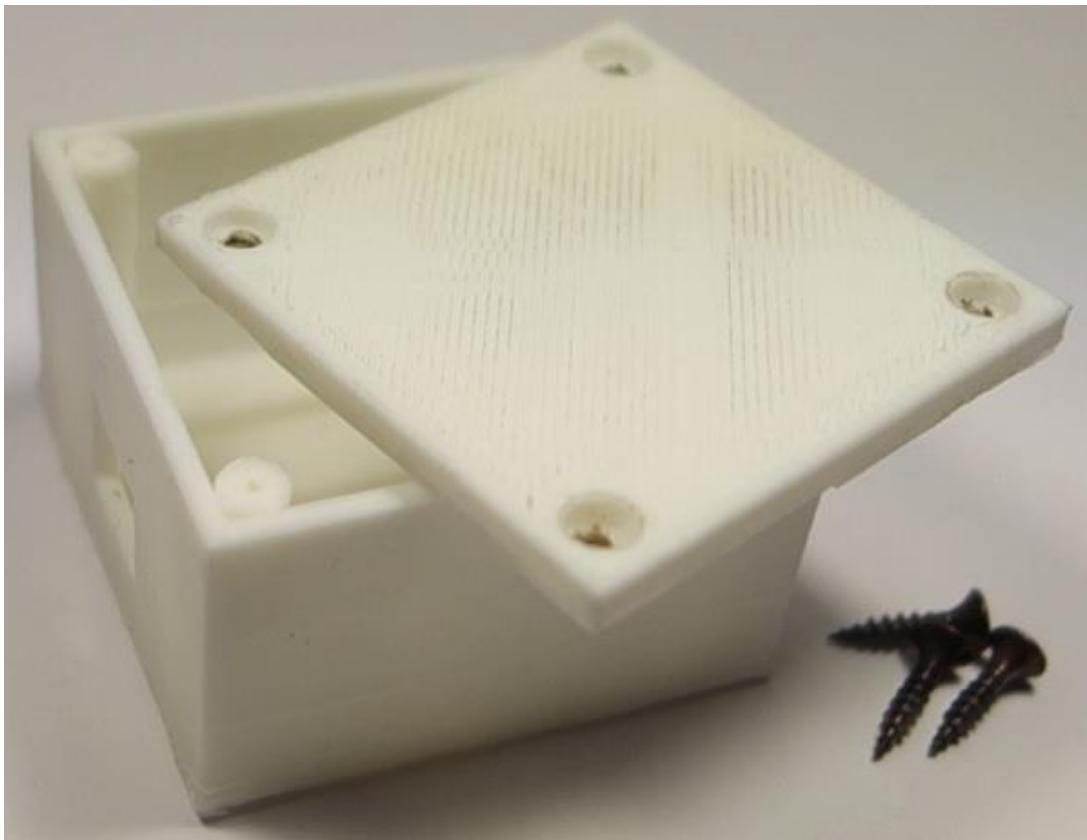


For this print we've used the Creality 3D CR-10 3D printer and you can [click here to read the best hobby 3D Printer guide.](#)

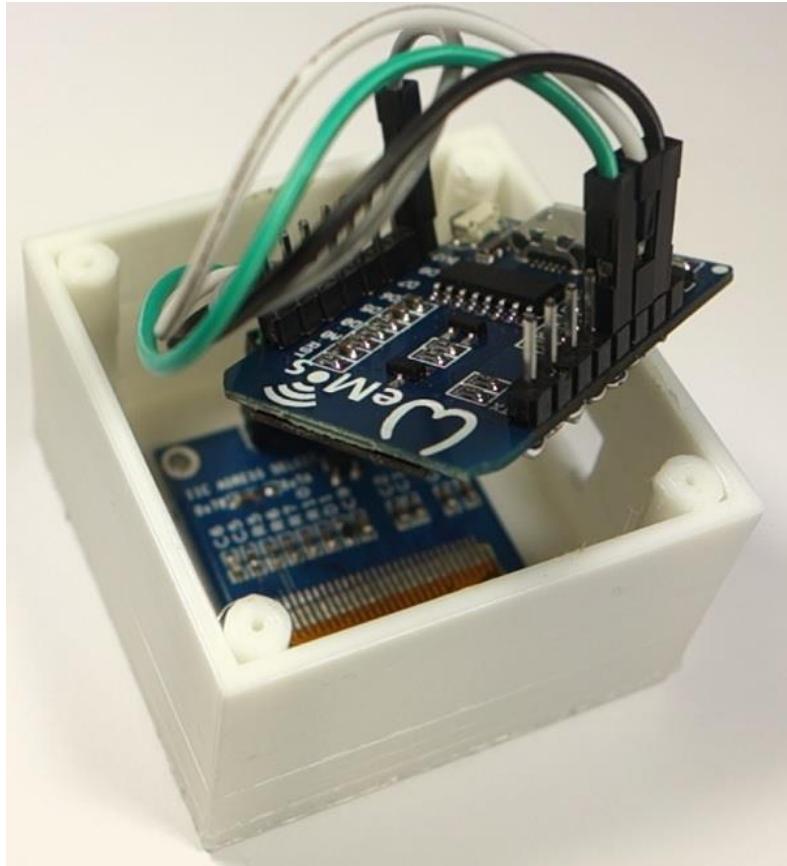


Assembly

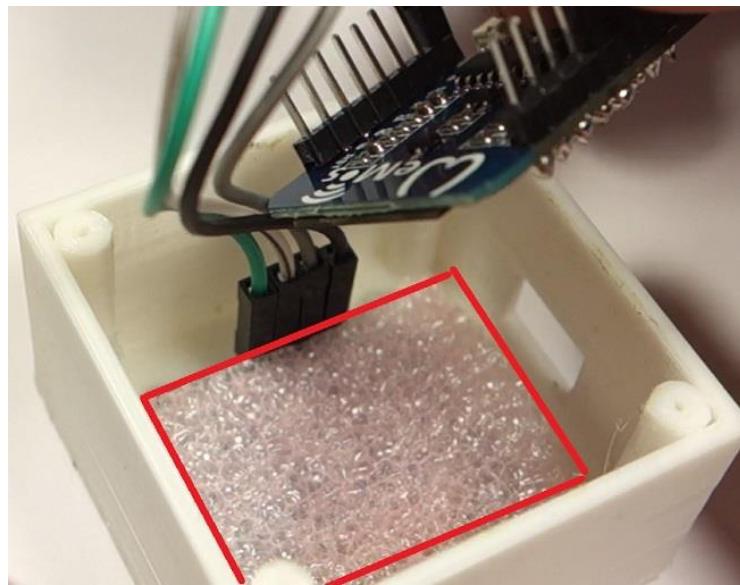
After having the 3D printed parts ready, you can start the assembly process.



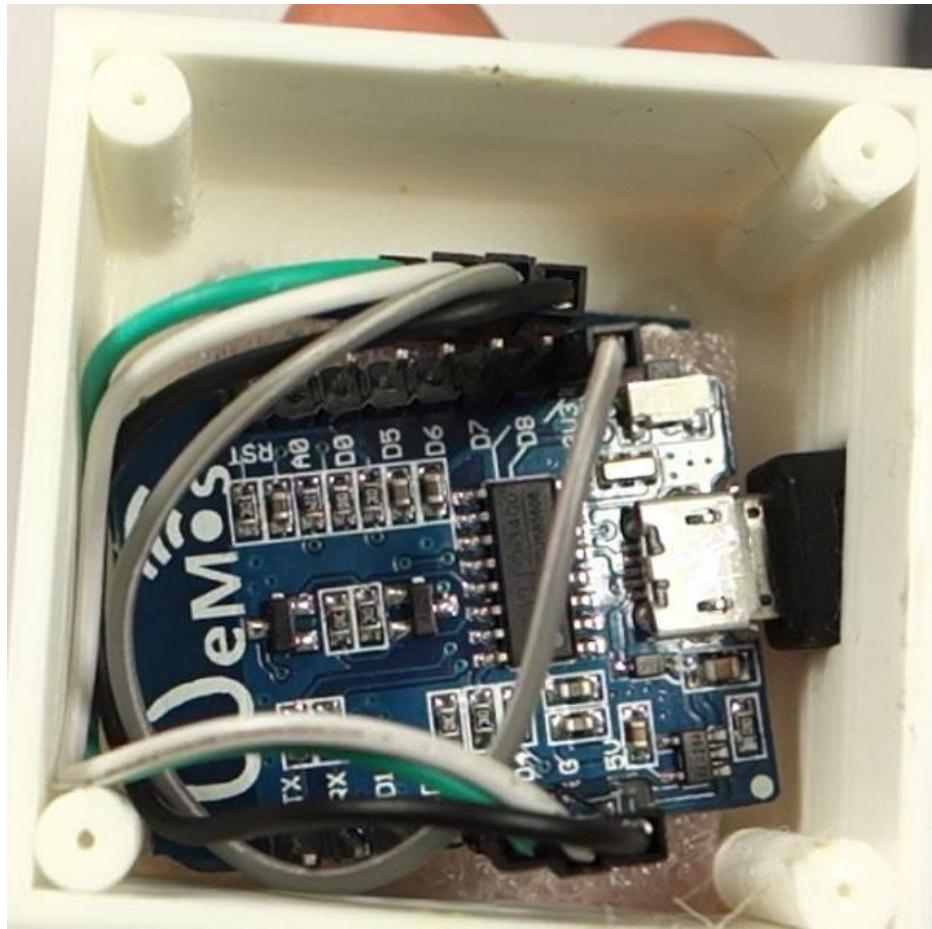
Start by placing the circuit inside enclosure.



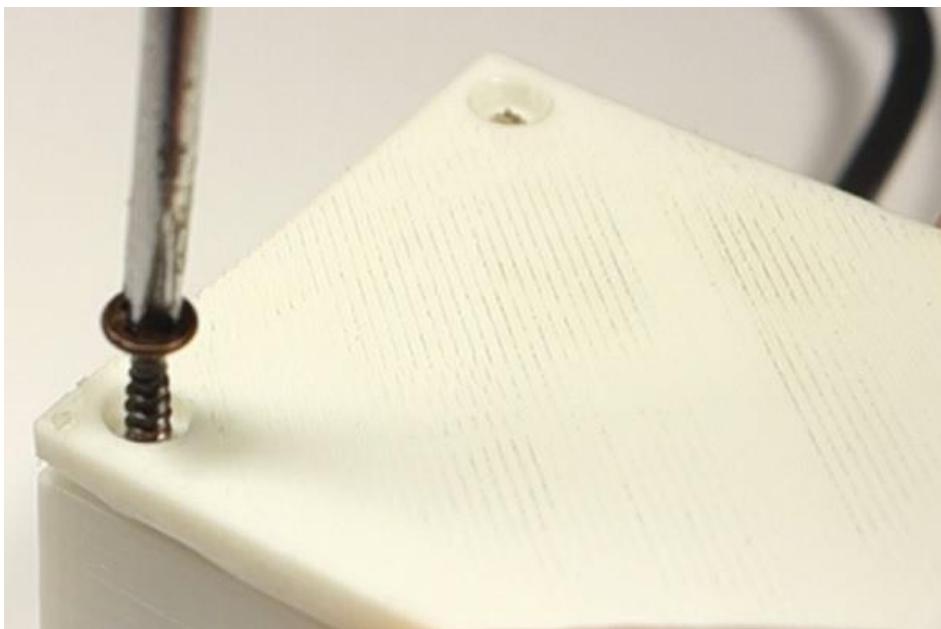
Use some hot glue to secure the OLED screen and add a small piece of plastic to isolate the circuit.



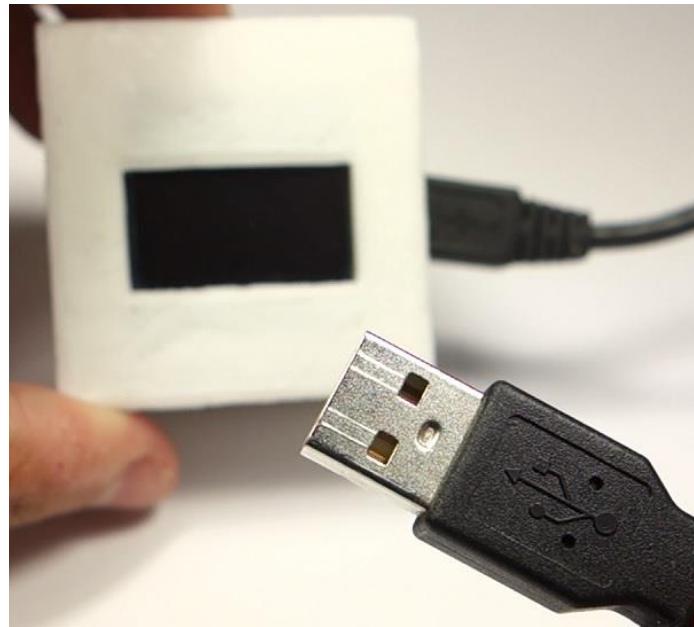
Connect the USB cable to your WeMos board.



Use a screwdriver to attach all four screws.

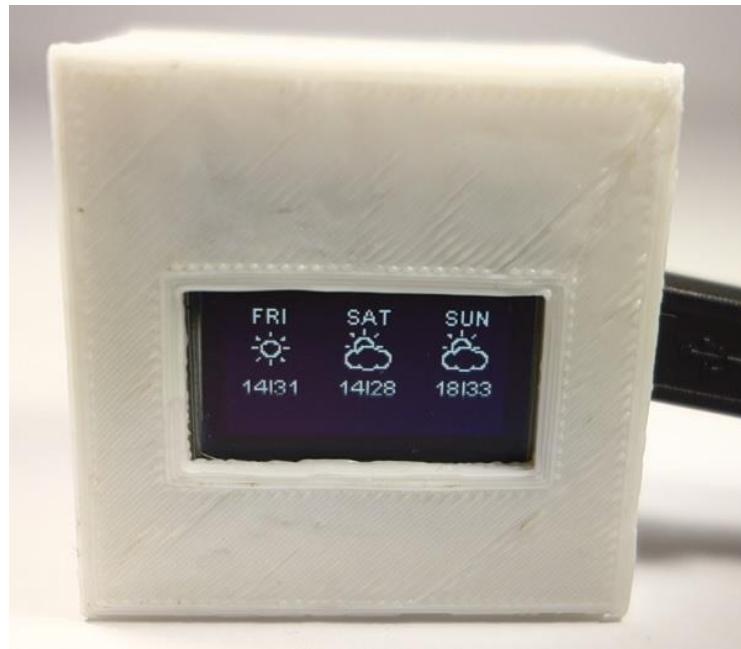


Finally, you can power up the circuit using a power bank or your laptop USB port.



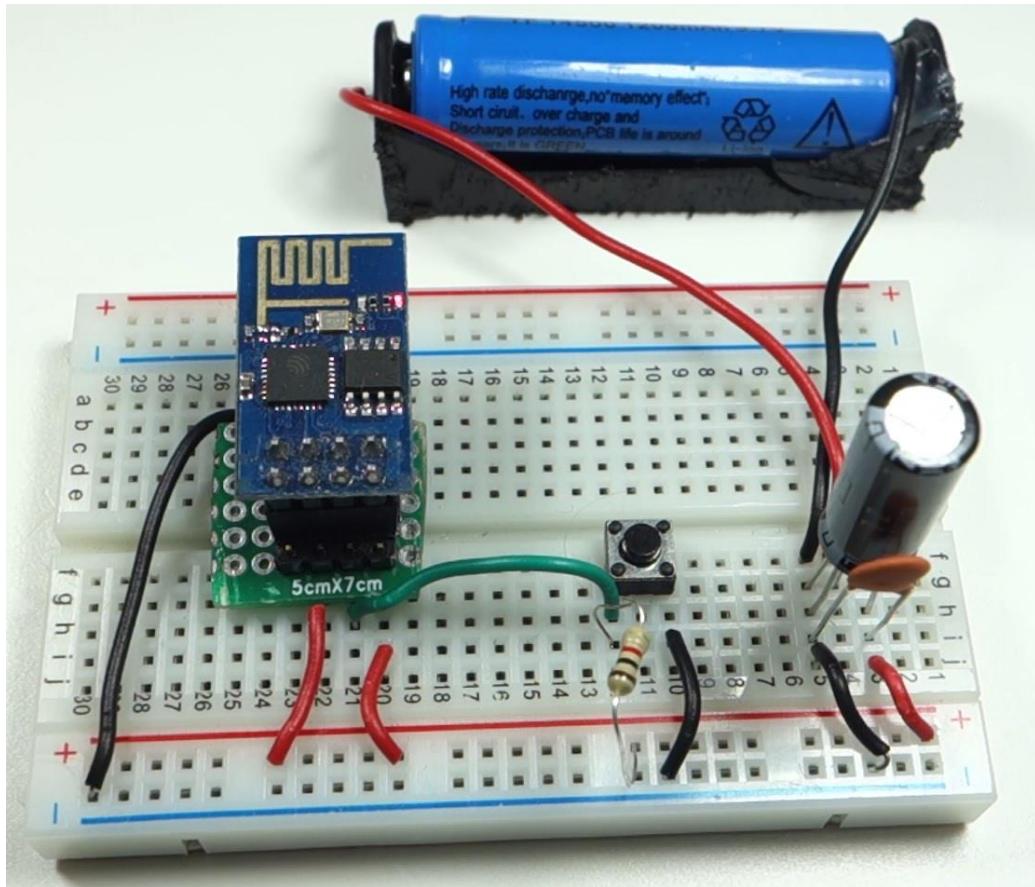
Final Result

The screen displays the time and date, current weather condition, weather forecast for the next days, outdoor temperature and humidity.



Unit 3: Voltage Regulator

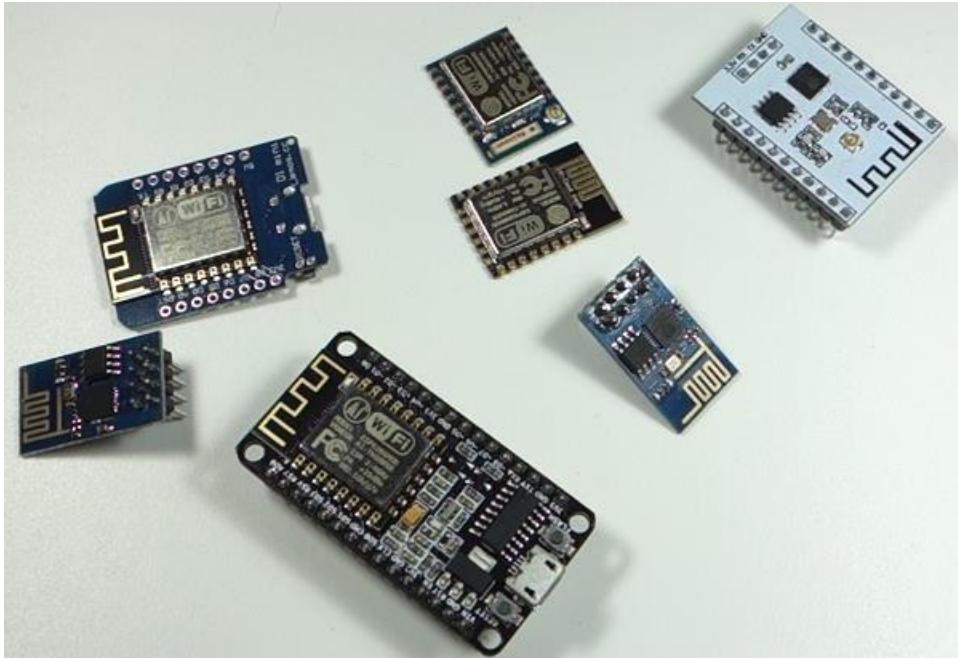
(Prepared for LiPo/Li-ion Batteries)



In this Unit, you'll build a voltage regulator circuit to prepare the ESP8266 to work with LiPo/Li-Ion batteries. It is important following this Unit before proceeding to the next one.

The ESP8266 is well known for being power hungry when performing Wi-Fi tasks. It can consume from **50mA** to **170mA**. So, for a lot of applications it's not ideal to use a battery with it.

It is better to use a power adapter connected to mains voltage, so that you don't have to worry about power consumption or charging batteries.

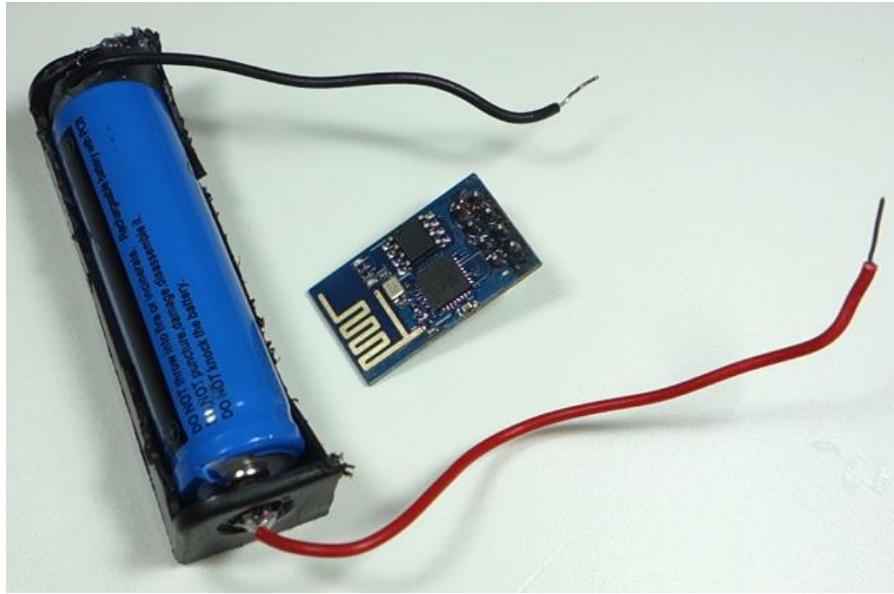


ESP8266 with LiPo Batteries

However, for some projects that use deep sleep or don't require constant Wi-Fi connection, using the ESP8266 with rechargeable LiPo batteries is a great solution.



For battery powered applications the ESP-01 version is the recommended board, because it has few on-board components so, it consumes less power.



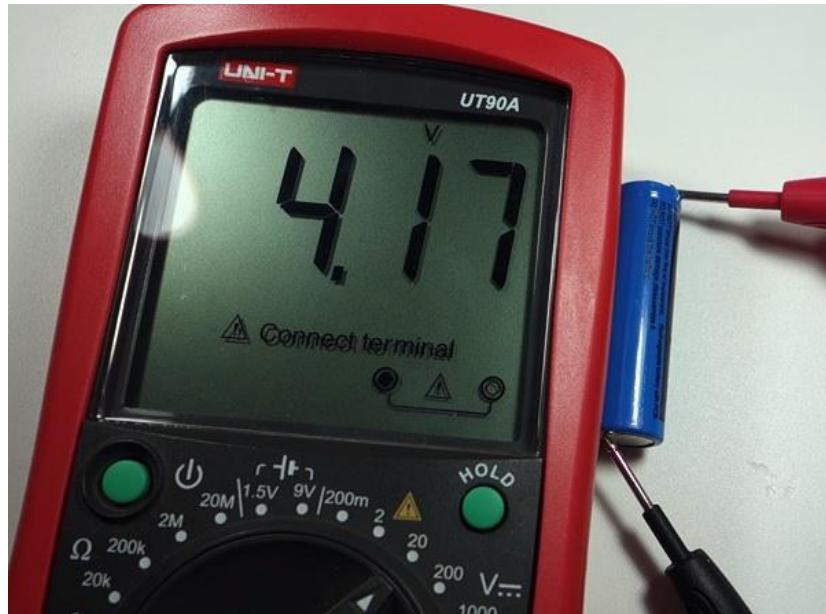
Boards like the ESP8266-12E NodeMCU consume more power, because they have extra components like resistors, capacitors, chips, etc.

Since LiPo batteries are so easily available, we'll show you how to power the ESP8266 using those batteries.

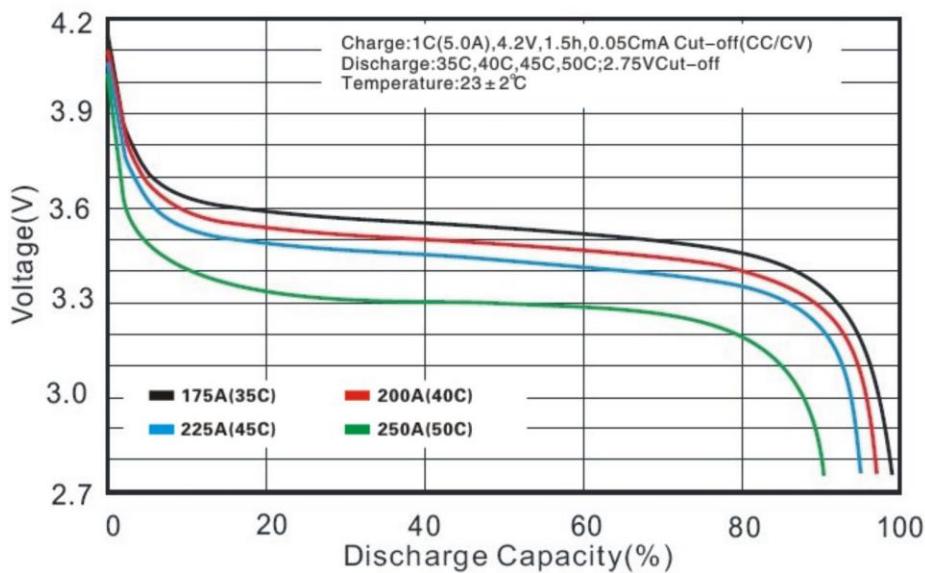
This section is not about different types of batteries and we won't explain how LiPo batteries work. We'll just give you the relevant information to complete the circuit presented.

LiPo Batteries Fully Charged

LiPo batteries are rechargeable with the appropriate charger and they output approximately **4.2V** when fully charged.



However, as the battery keeps discharging the voltage starts to drop:



The **ESP8266 recommended operating voltage is 3.3V**, but it can operate with voltages between 3V and 3.6V. So, you can't plug the LiPo battery directly to an ESP8266, you need a voltage regulator.

Typical Linear Voltage Regulator

Using a typical linear voltage regulator to drop the voltage from 4.2V to 3.3V isn't a good idea.

For example: if the battery discharges to 3.7V, your voltage regulator would stop working, because it has a high cutoff voltage.



Low-dropout or LDO Regulator

To drop the voltage efficiently with batteries, you need to use low-dropout regulator or also known as LDO regulator that can regulate the output voltage.

Having a low dropout voltage means that even if the battery is only outputting 3.4V it would still work. Keep in mind that you should never empty the LiPo battery completely, because it will damage the battery or decrease its lifetime.

After researching LDOs, I found a couple of good alternatives. One of the best LDOs I've found was the **MCP1700-3302E**.



It's small and it looks like a transistor. There is also a suitable alternative like the **HT7333-A**.



Any LDO that has similar specifications to the ones shown on the datasheet below are also good alternatives. Your LDO should have similar specs when it comes to:

- Output voltage (3.3V)
- Quiescent current (~1.6uA)
- Output current (~250mA)
- Low-dropout voltage (~178mV)

MICROCHIP

MCP1700

Low Quiescent Current LDO

Features:

- 1.6 μ A Typical Quiescent Current
- Input Operating Voltage Range: 2.3V to 6.0V
- Output Voltage Range: 1.2V to 5.0V
- 250 mA Output Current for Output Voltages \geq 2.5V
- 200 mA Output Current for Output Voltages $<$ 2.5V
- Low Dropout (LDO) Voltage
 - 178 mV Typical @ 250 mA for $V_{OUT} = 2.8V$
- 0.4% Typical Output Voltage Tolerance
- Standard Output Voltage Options:
 - 1.2V, 1.8V, 2.5V, 2.8V, 3.0V, **3.3V**, 5.0V
- Stable with 1.0 μ F Ceramic Output Capacitor
- Short Circuit Protection
- Overtemperature Protection

General Description:

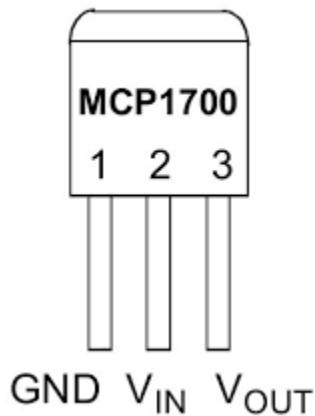
The MCP1700 is a family of CMOS low dropout (LDO) voltage regulators that can deliver up to 250 mA of current while consuming only 1.6 μ A of quiescent current (typical). The input operating range is specified from 2.3V to 6.0V, making it an ideal choice for two and three primary cell battery-powered applications, as well as single cell Li-Ion-powered applications.

The MCP1700 is capable of delivering 250 mA with only 178 mV of input to output voltage differential ($V_{OUT} = 2.8V$). The output voltage tolerance of the MCP1700 is typically $\pm 0.4\%$ at $+25^\circ C$ and $\pm 3\%$ maximum over the operating junction temperature range of $-40^\circ C$ to $+125^\circ C$.

Output voltages available for the MCP1700 range from 1.2V to 5.0V. The LDO output is stable when using only 1 μ F output capacitance. Ceramic, tantalum or aluminum electrolytic capacitors can all be used for

MCP1700-3302E Pinout

Here's the MCP1700-3302E pinout. It has GND, Vin and Vout:



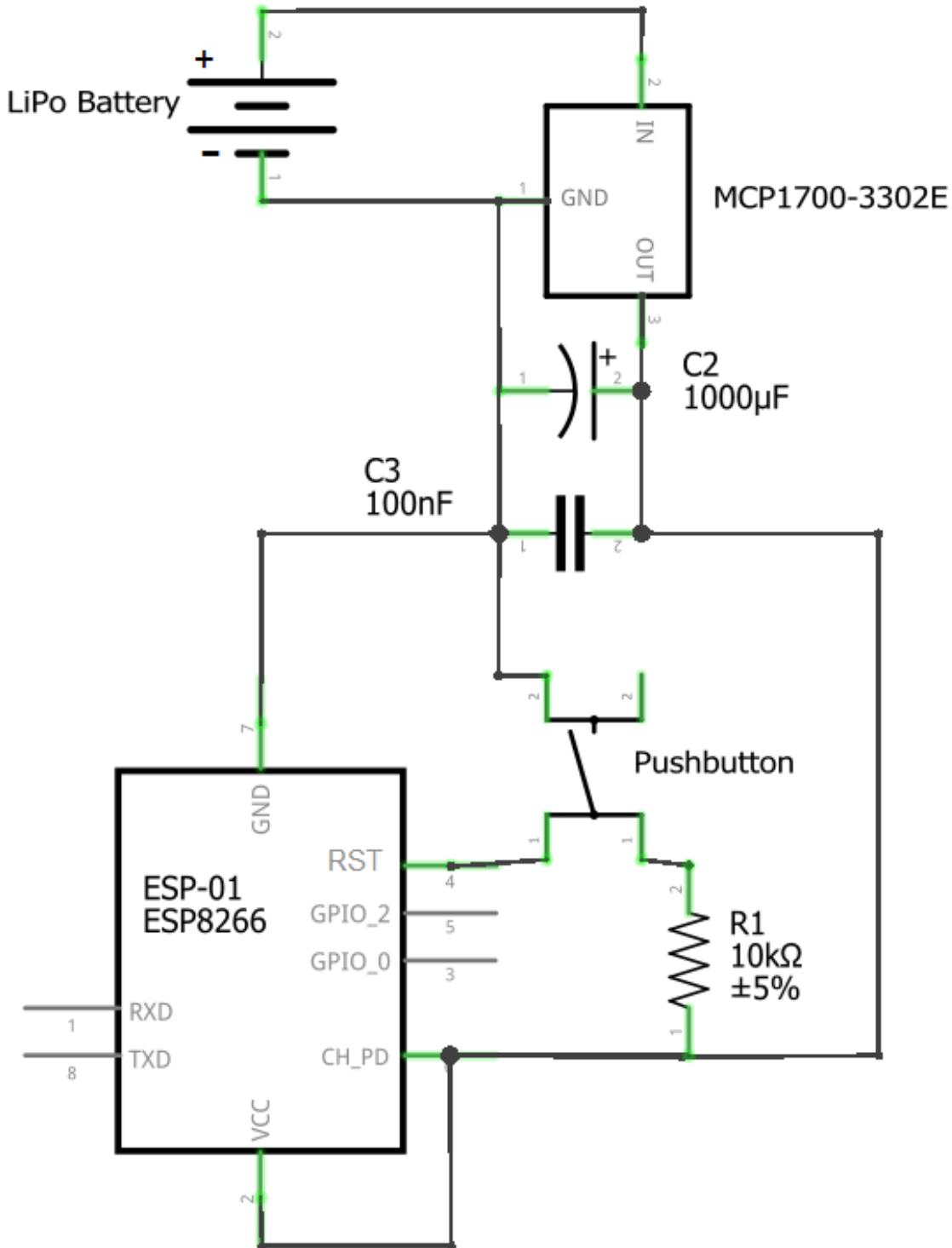
Other LDOs should have the same pinout, but you should always search for your LDO's datasheet to verify its pinout.

ESP8266 Circuit with LDO and LiPo Battery

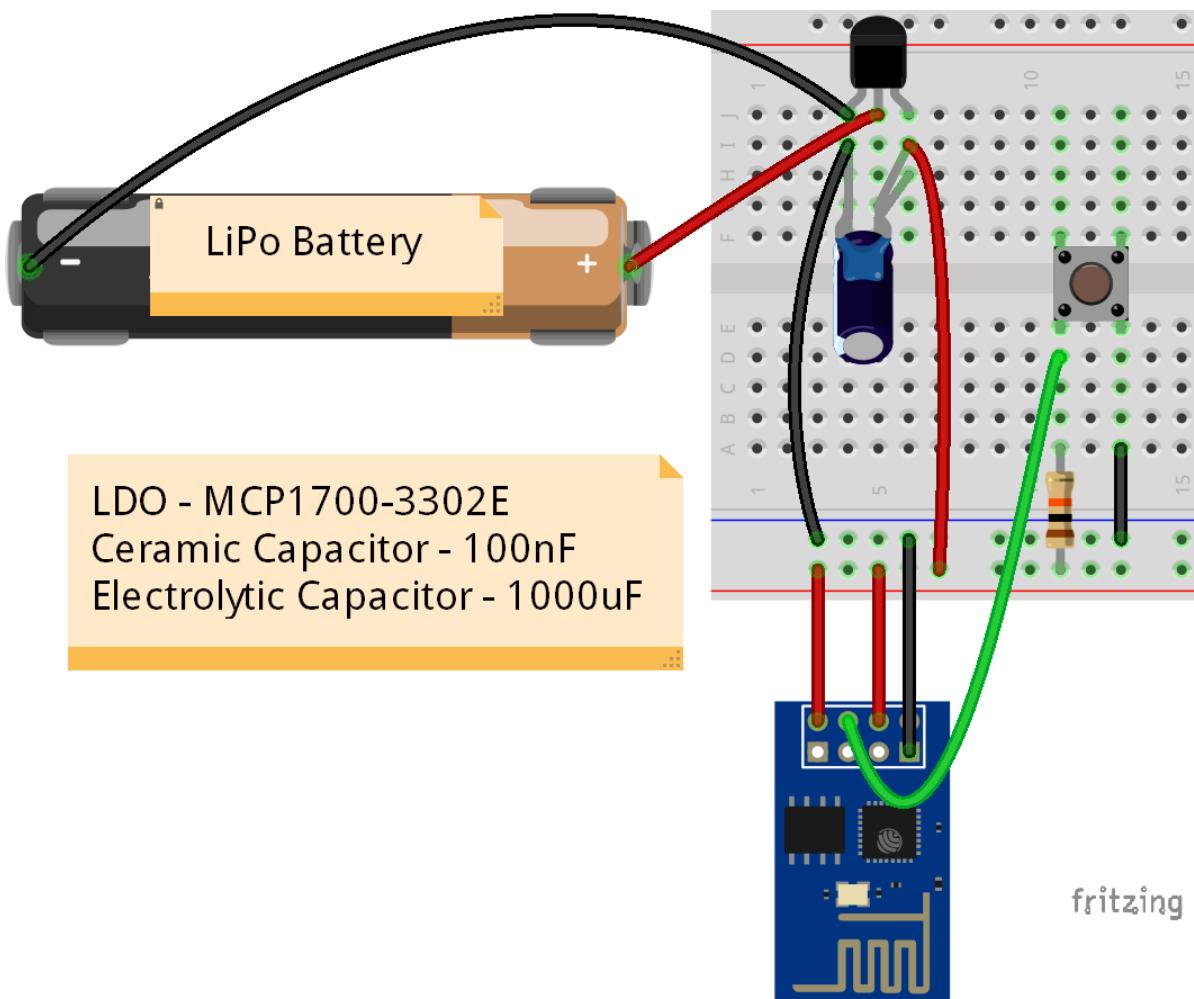
Here's the parts required to design the voltage regulator for the ESP-01:

- [Li-ion/LiPo battery](#) + battery holder
- [Low-dropout or LDO regulator \(MCP1700-3302E\)](#)
- [1000uF electrolytic capacitor](#)
- [100nF ceramic capacitor](#)
- [Pushbutton](#)
- [10k Ohm resistor](#)
- [ESP-01](#)
- [Breadboard](#)
- [Jumper wires](#)

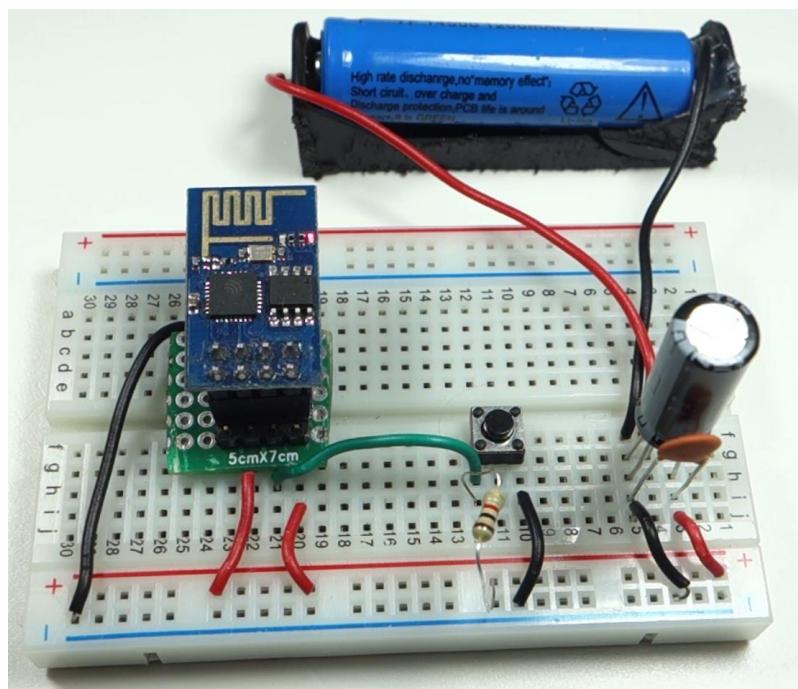
Take a look at the diagram below to design your own voltage regulator circuit.



Or you can take a look a Fritzing diagram (the ceramic capacitor and electrolytic capacitor are in parallel with GND and Vout of the LDO):

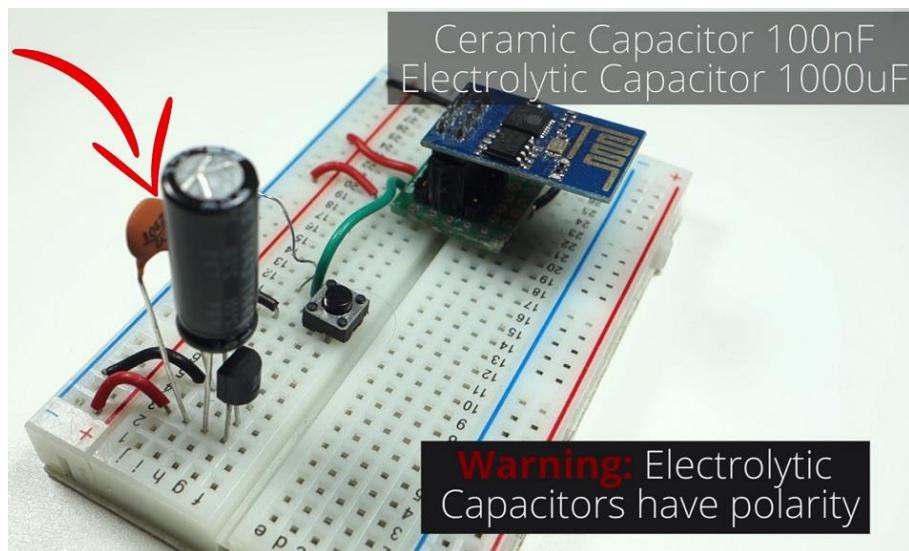


The pushbutton is connected to the RESET pin of the ESP-01. For this particular example it's not necessary, but it will be useful in the next Unit. Here's the final circuit:



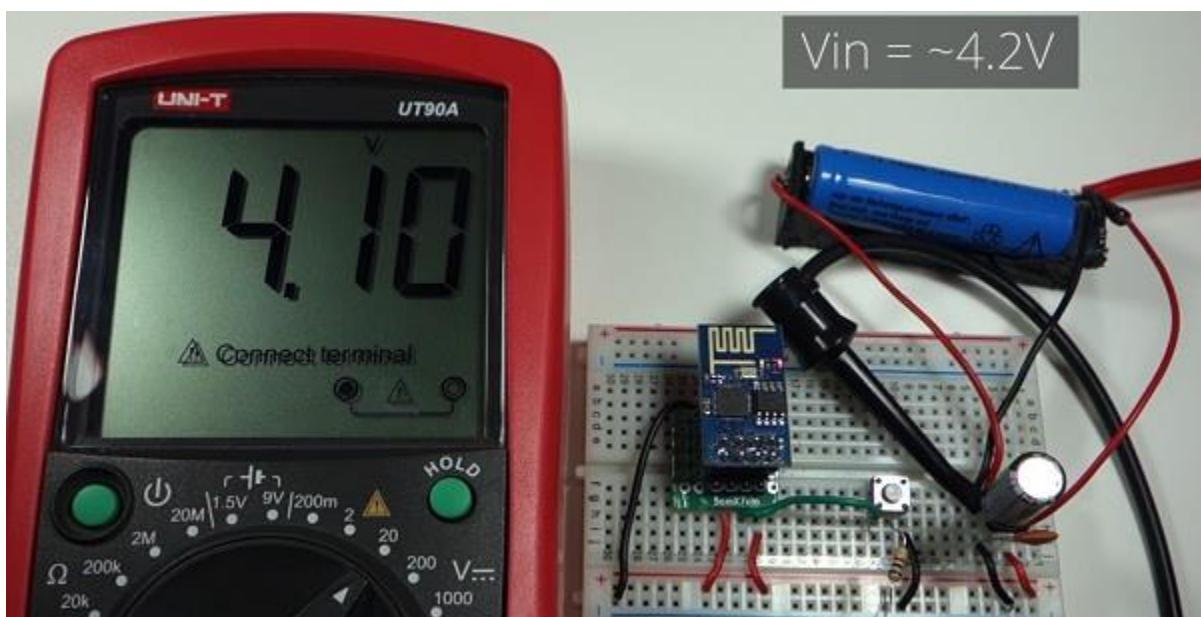
About the capacitors

The LDOs should have a ceramic capacitor and an electrolytic capacitor connected in parallel to GND and Vout to smooth the voltages peaks. The capacitors prevent unexpected resets or unstable behavior for your ESP8266.

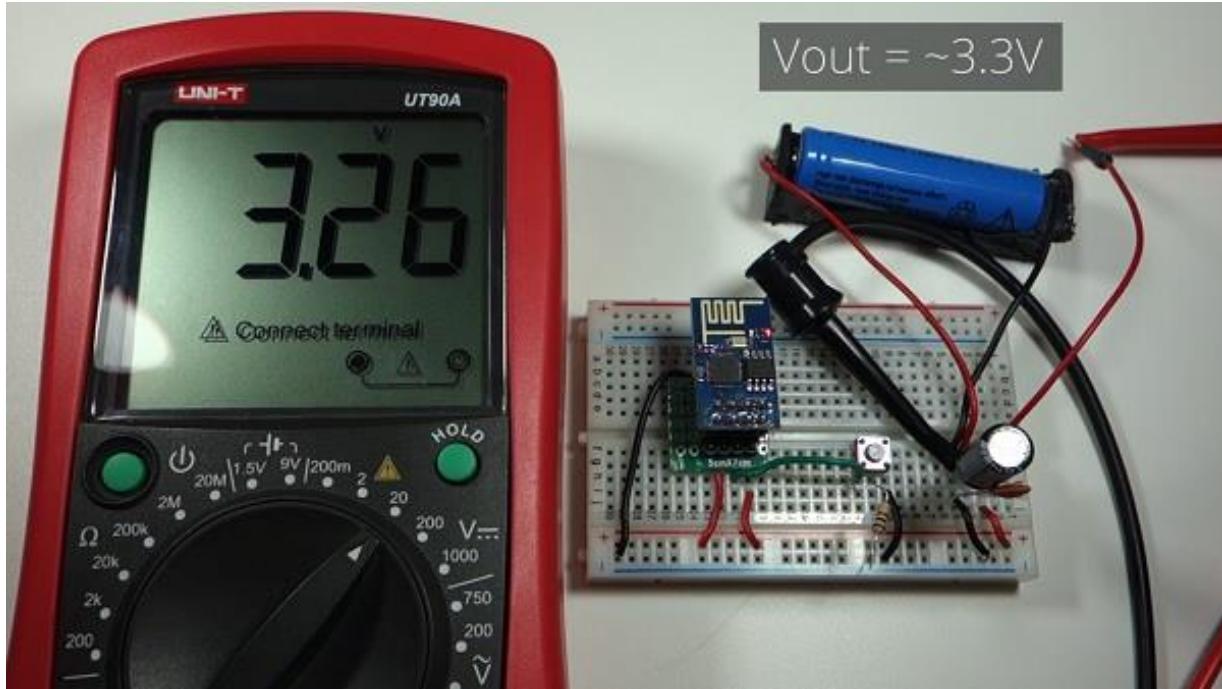


Testing

Having the multimeter measuring the Vin voltage of the LiPo battery, you can see it outputs approximately 4.2V, because the battery is currently fully charged.

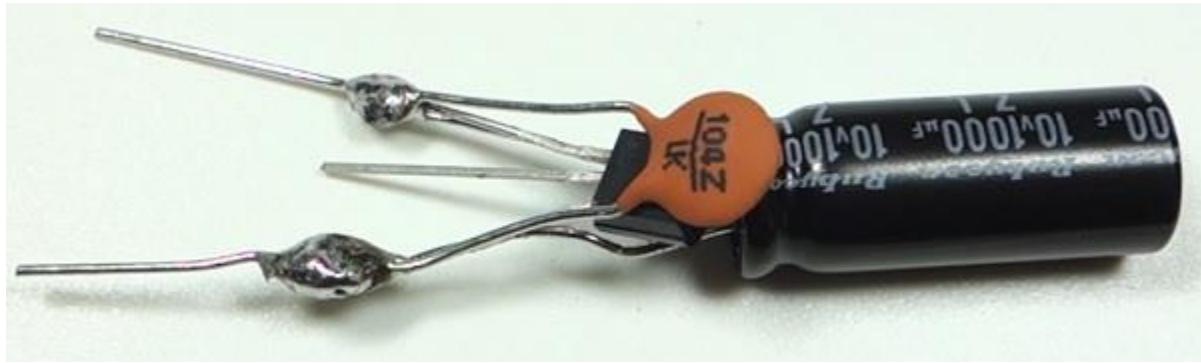


Let's place the multimeter probe on Vout. Now, the multimeter is measuring approximately 3.3V which is the recommended voltage to power the ESP8266.



Voltage Regulator

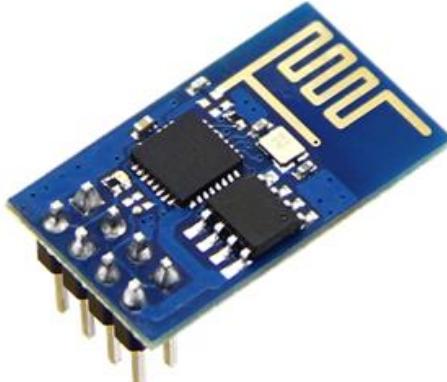
A popular voltage regulator design for the ESP8266 looks like this:



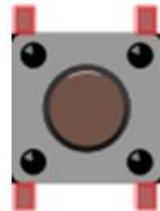
You solder the capacitors to the LDO, so in the end you have a voltage regulator in a small form factor that can be easily used in your projects.

We hope this guide was useful. This concept will be very helpful to power future projects.

Unit 4: Wi-Fi Button (DIY Dash button)



ESP-01

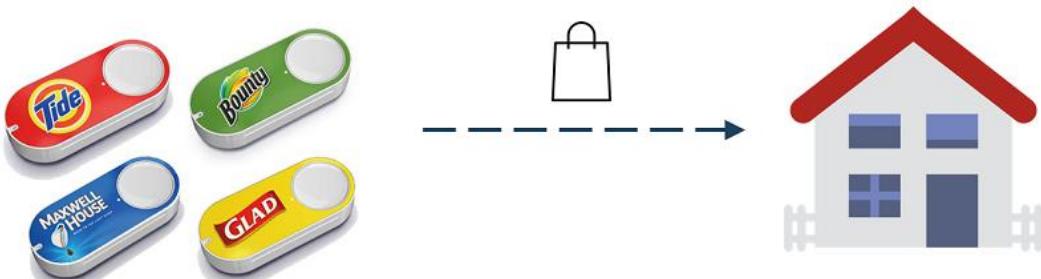


Pushbutton

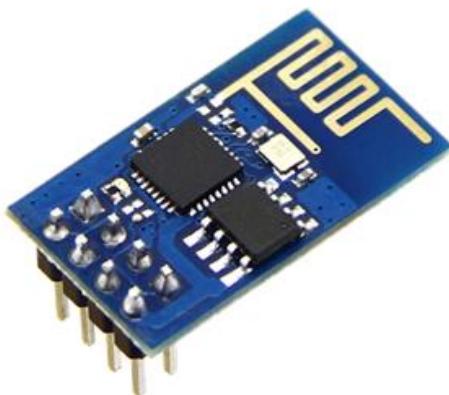
In this project you're going to build an ESP8266 Wi-Fi Button that can trigger any home automation event. This is like a remote control that you can carry in your pocket or place anywhere, that when pressed sends out an email.



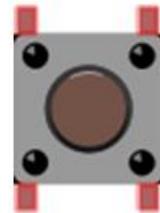
This is not a new idea and this concept was popularized by Amazon that created the Dash button, which is a small button that instantly orders a product to your home.



Since the ESP8266 boards are so inexpensive, we can make a similar project that works like the Dash button, but with our own twist.



ESP-01



Pushbutton

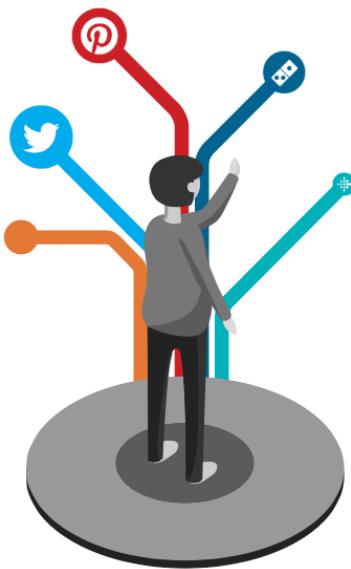
Instead of ordering a product we can turn on a light, toggle a lamp, send a value, trigger an email notification or any other task.

IFTTT

For this project we're going to use a free service called **IFTTT** that stands for If This Than That.

This service is used to automate a wide variety of tasks online. In this case, we want to send an email when the ESP8266 pushbutton is pressed.

Type in your browser <https://ifttt.com> and click the “**Get started**” button in the middle of the page. Complete the form with your details and create your account.



Every thing works better together

Get started

or

Continue with Google Continue with Facebook

Creating the Applet

Go to this URL to start creating your own applet: <https://ifttt.com/create>

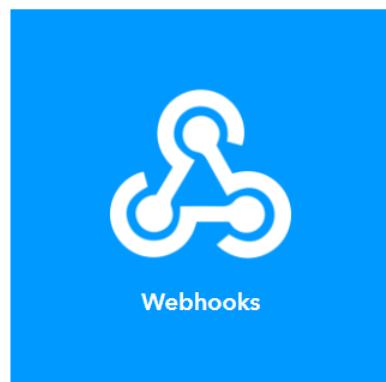
If + This Then That

Click the “+” sign. Search for “**webhooks**” in the search bar.

Choose a service

Step 1 of 6

webhooks ×



Select the “Receive a web request” option.

Choose trigger

Step 2 of 6

Receive a web request

This trigger fires every time the Maker service receives a web request to notify it of an event. For information on triggering events, go to your Maker service settings and then the listed URL (web) or tap your username (mobile)

Give a name to the event, for example “button_pressed” and click the **Create trigger** button.



Complete trigger fields

Step 2 of 6

Event Name

button_pressed

The name of the event, like "button_pressed" or "front_door_opened"

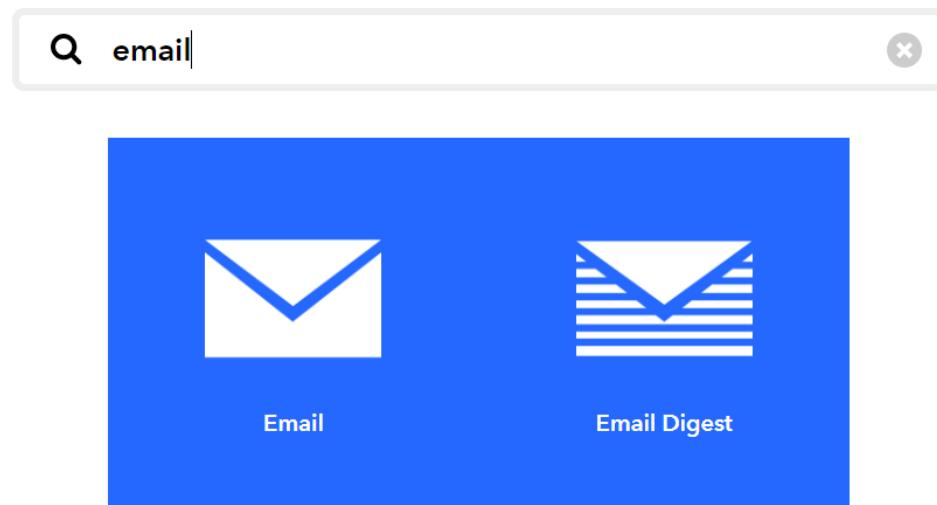
Create trigger

With the Webhooks service you can trigger an action when you make an HTTP request to a specific URL. That's what we're going to do.

Now press the "+" next to the "That" word, search and select the **Email** service.

Choose action service

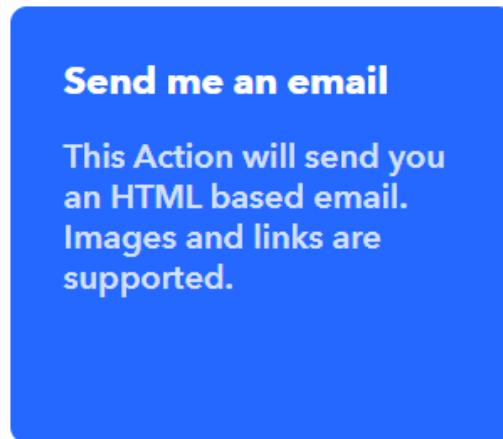
Step 3 of 6



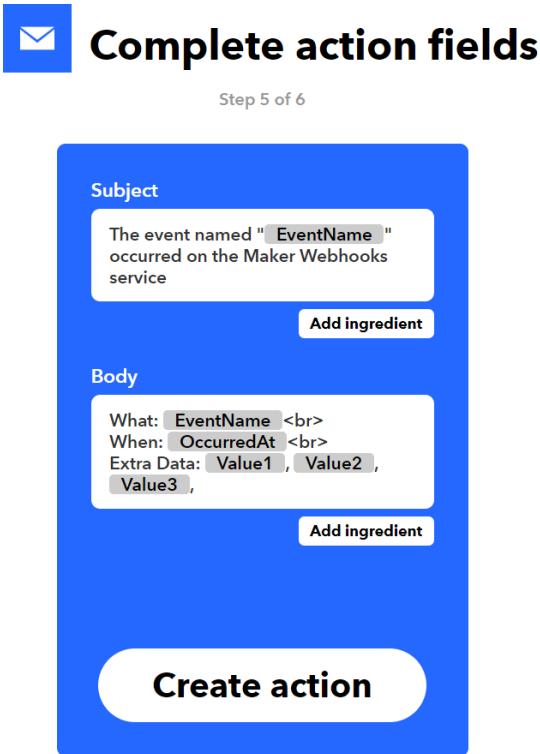
Choose the "**Send email**" option.



Step 4 of 6



Then, click the “**Create action**” button. You can leave the fields as default.



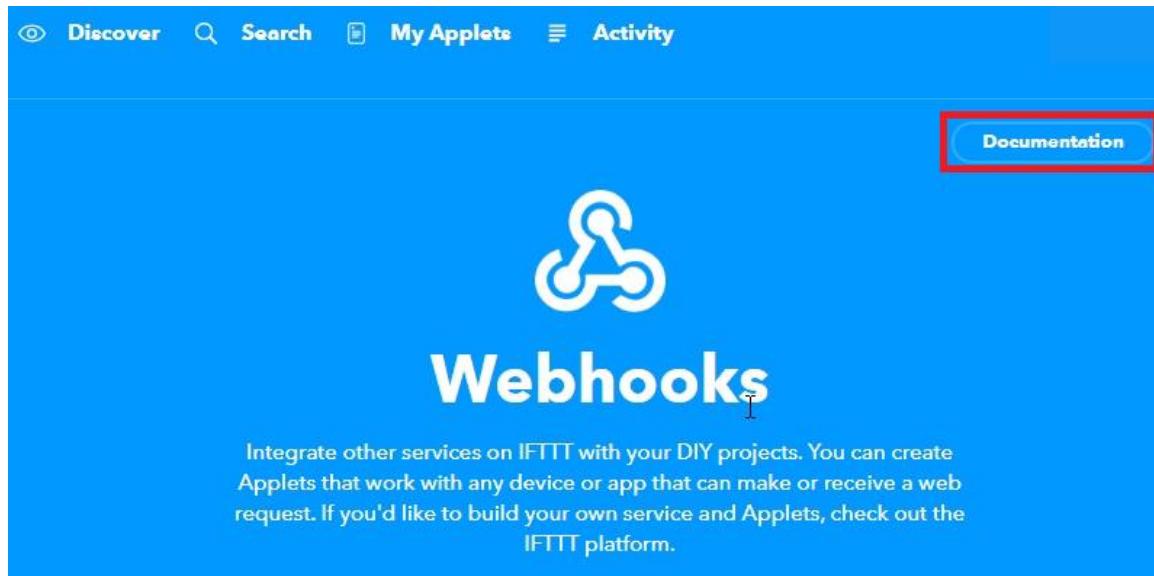
Then, click “**Finish**” to create your applet.

Review and finish

The screenshot shows the "Review and finish" step of the applet creation process. It displays the applet configuration with two icons at the top: a cloud-like icon and an envelope icon. The main area shows the trigger "If Maker Event 'button_pressed', then Send me an email at [REDACTED]". Below this, it says "79/140" and "by [REDACTED]". At the bottom, there's a light gray box with the text "Receive notifications when this Applet runs" next to a toggle switch that is currently off. A large black "Finish" button is at the very bottom.

Testing the Applet

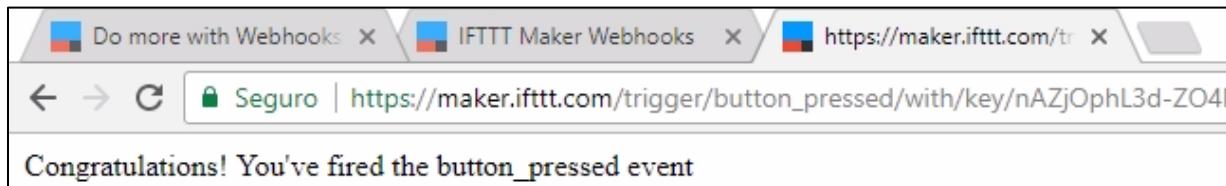
Go to the following URL: https://ifttt.com/maker_webhooks, then click the "Documentation" tab.



Here you can find your unique **API KEY** that you must keep private. Type in the event name, **button_pressed**. Your final URL should appear in the bottom of the web page. Copy that URL.

A screenshot of the IFTTT API key generation page. It shows a blue icon with three circles and the text "Your key is: nAZjOphL3d-ZO4N3k64-1A7gTINSrxMJdmqy". Below this, there's a "Back to service" link. A section titled "To trigger an Event" provides instructions: "Make a POST or GET web request to:" followed by a URL: "https://maker.ifttt.com/trigger button_pressed/with/key/nAZjOphL3d-ZO4N3k64-1A7gT1M". It also shows an optional JSON body: "{ \"value1\" : \"[]\", \"value2\" : \"[]\", \"value3\" : \"[]\" }". A note says: "The data is completely optional, and you can also pass value1, value2, and value3 as query parameters or form variables. They will be passed on to the Action in your Recipe." Finally, it shows a command-line example: "curl -X POST https://maker.ifttt.com/trigger/button_pressed/with/key/nAZjOphL3d-ZO4N3k64-1A7gT1M". The URL in the JSON body and the curl command are highlighted with red boxes.

Open a new tab in your browser and hit enter. You should see this message saying "Congratulations!".



Open your email and the new message should be there.

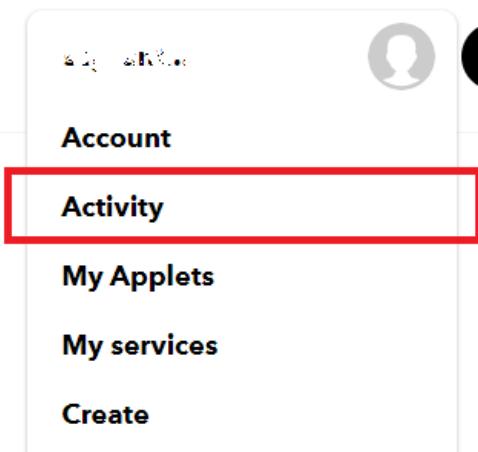
The event named "button_pressed" occurred on the Maker Webhooks service

IFTTT Webhooks via IFTTT <action@ifttt.com> [Anular subscrição](#)
para mim ▾

What: button_pressed
When: November 4, 2019 at 11:34AM
Extra Data: , , ,



In case the email didn't arrive after a few seconds, we recommend double-checking the URL and see if you're using the correct event name, both in your Applet and in your URL. Or, in your account, go to "**Activity**" and try to understand what might have happened.



If everything went out smoothly, save your unique URL in a Notepad, because you'll need it later in this project. Basically, you'll need to make a request on that URL with the ESP8266 when you press the dash button.

Code

Here's the code that you need to upload to your ESP8266 board. You need to change three variables to make it work for your: `ssid`, `password` and `resource`.

```
#include <ESP8266WiFi.h>

// Replace with your SSID and Password
const char* ssid      = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

// Replace with your unique IFTTT URL resource
const char* resource = "REPLACE_WITH_YOUR_IFTTT_URL_RESOURCE";

// How your resource variable should look like, but with your own API
// KEY (that API KEY below is just an example):
//const char* resource = "/trigger/button_pressed/with/key/nAZjOphL3d-
ZO4N3k64-1A7gTlNSrxMJdmqy";

// Maker Webhooks IFTTT
const char* server = "maker.ifttt.com";

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

  initWifi();
  makeIFTTTRequest();

  // Deep sleep mode until RESET pin is connected to a LOW signal
  // (pushbutton is pressed)
  ESP.deepSleep(0);
}

void loop() {
  // sleeping so wont get here
}

// Establish a Wi-Fi connection with your router
void initWifi() {
  Serial.print("Connecting to: ");
  Serial.print(ssid);
  WiFi.begin(ssid, password);

  int timeout = 10 * 4; // 10 seconds
  while(WiFi.status() != WL_CONNECTED && (timeout-- > 0)) {
```

```

        delay(250);
        Serial.print(".");
    }
    Serial.println("");

    if(WiFi.status() != WL_CONNECTED) {
        Serial.println("Failed to connect, going back to sleep");
    }

    Serial.print("WiFi connected in: ");
    Serial.print(millis());
    Serial.print(", IP address: ");
    Serial.println(WiFi.localIP());
}

// Make an HTTP request to the IFTTT web service
void makeIFTTTRequest() {
    Serial.print("Connecting to ");
    Serial.print(server);

    WiFiClient client;
    int retries = 5;
    while(!client.connect(server, 80) && (retries-- > 0)) {
        Serial.print(".");
    }
    Serial.println();
    if(!client.connected()) {
        Serial.println("Failed to connect, going back to sleep");
    }

    Serial.print("Request resource: ");
    Serial.println(resource);
    client.print(String("GET ") + resource +
                " HTTP/1.1\r\n" +
                "Host: " + server + "\r\n" +
                "Connection: close\r\n\r\n");

    int timeout = 5 * 10; // 5 seconds
    while(!client.available() && (timeout-- > 0)) {
        delay(100);
    }
    if(!client.available()) {
        Serial.println("No response, going back to sleep");
    }
    while(client.available()){
        Serial.write(client.read());
    }

    Serial.println("\nclosing connection");
    client.stop();
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module5/Unit4/Dash_Button_Clone/Dash_Button_Clone.ino

Here's how the code works:

- It starts the serial communication at a baud rate of 115200;
- Runs the `initWifi` function that establishes a Wi-Fi connection between your ESP8266 and your router;
- Then, it executes the `makeIFTTTRequest` function that will make a request to the IFTTT service and ultimately IFTTT will send out an email.

You just need to insert your network credentials in the following variables:

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

And copy your unique URL resource to the `resource` variable:

```
const char* resource = "REPLACE_WITH_YOUR_IFTTT_URL_RESOURCE";
```

For example:

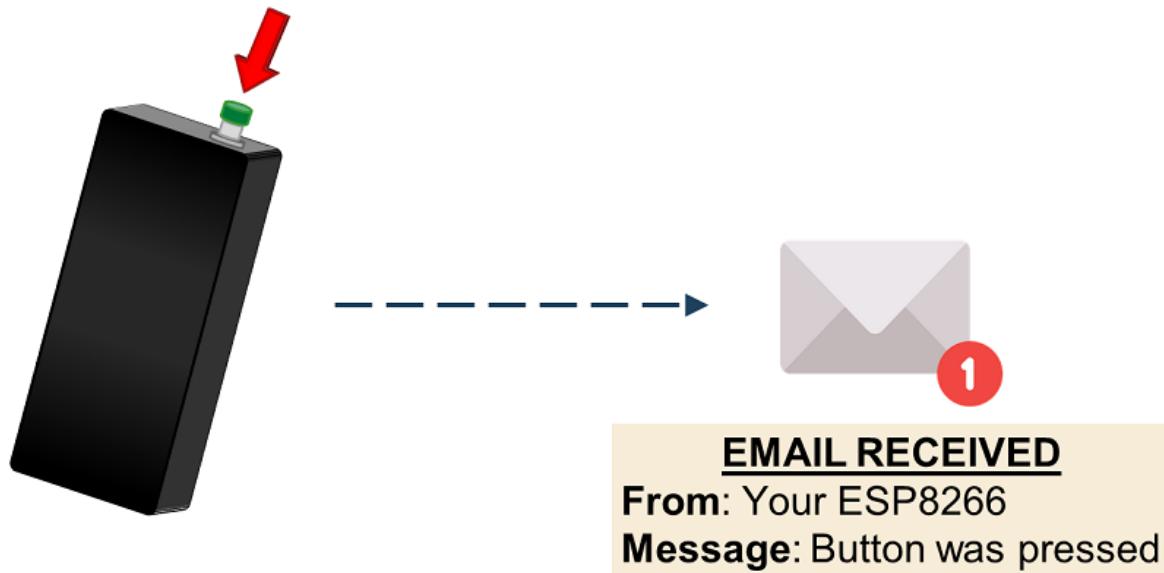
```
Const char resource = "/trigger/button_pressed/with/key/nAZjOphL3d-ZO4N3k64-1A7gTlNSrxMJdmqy";
```

Finally, we're using the Deep Sleep function, so that the ESP8266 is always off and consumes very little power. T

```
ESP.deepSleep(0);
```

The Deep Sleep function with ESP8266 was covered in greater detail in a previous Unit "[Unit 5: Deep Sleep](#)".

In summary, when you press the push button, the ESP8266 wakes up, performs an action and it goes back to Deep Sleep mode to save battery. It's simple how it works.



After adding your SSID, password and URL, upload the code to the ESP8266.

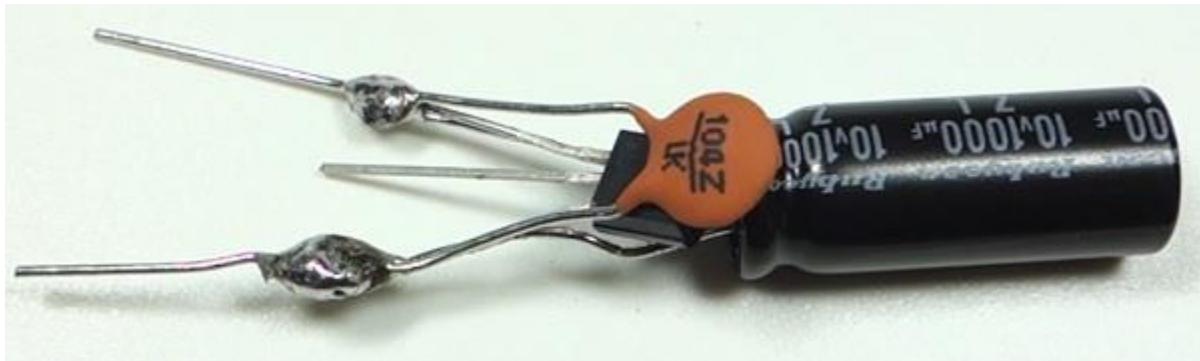
About the Circuit

We want this device to be portable and easy to make, so we're going to power the ESP8266 with a Lithium battery.



Voltage regulator

To power the ESP8266 safely with a Lithium battery you need to make a voltage regulator circuit. We've also covered that in a previous Unit called "["Voltage Regulator \(Prepared for LiPo/Li-ion Batteries\)"](#)".



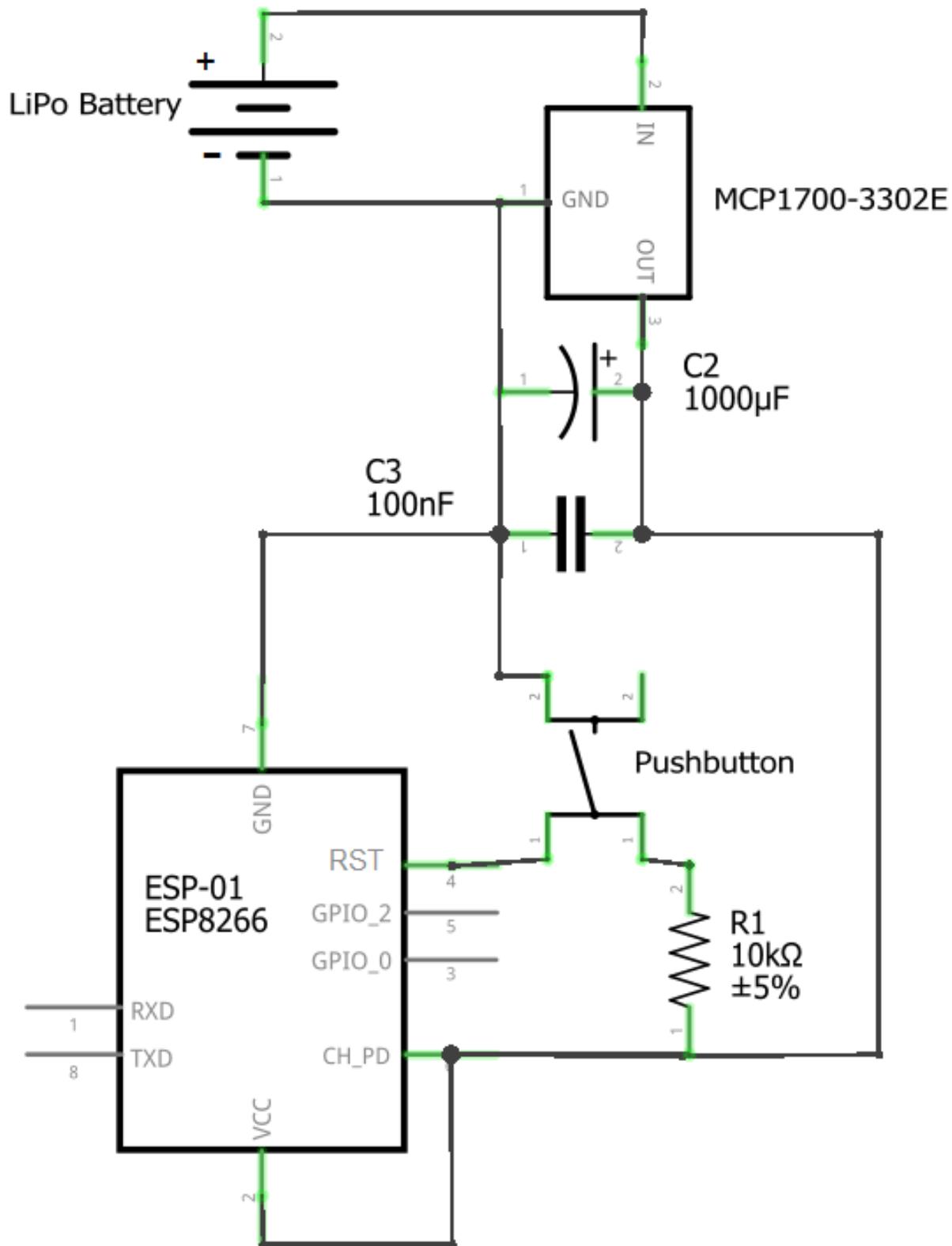
Parts required

After having the code running on the ESP-01 (or ESP8266-12E), these are the components that you need for your circuit:

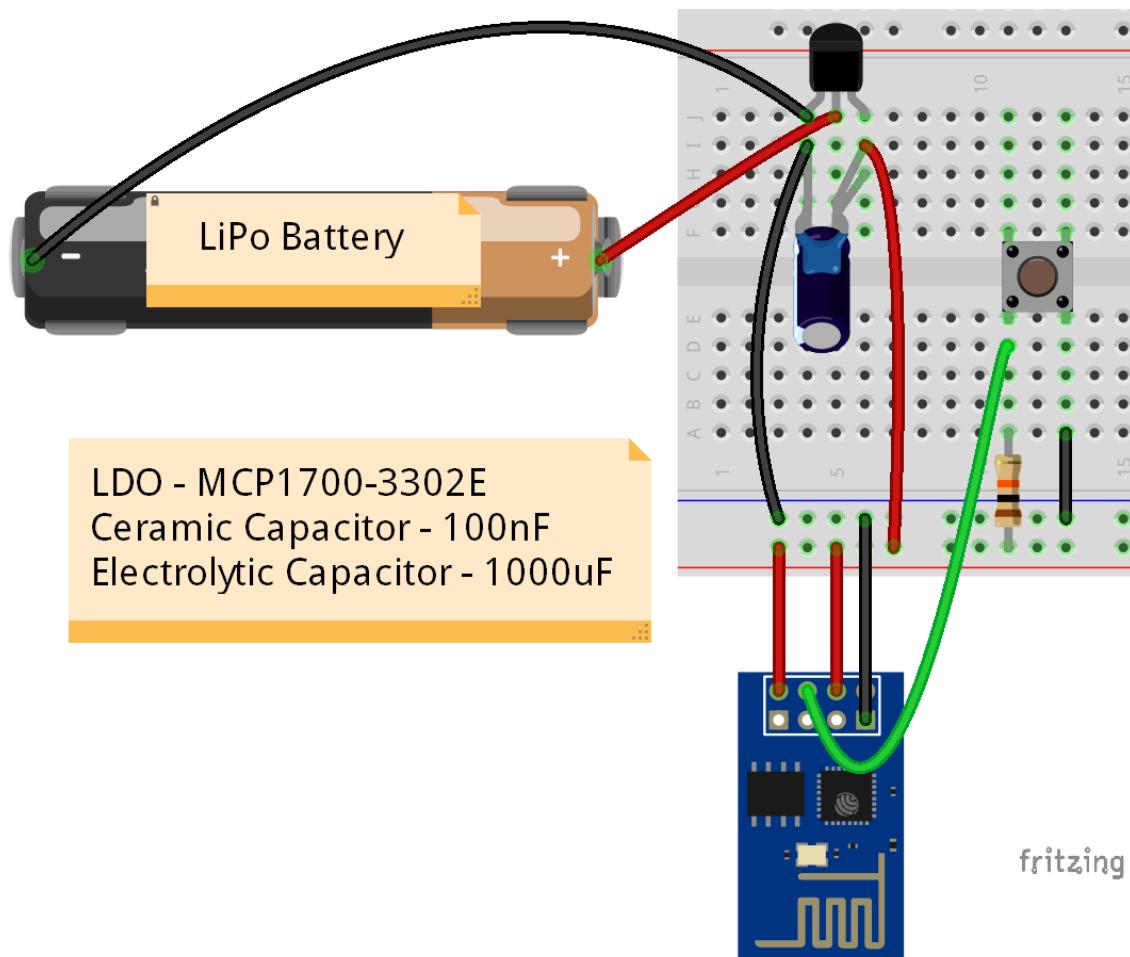
- [ESP-01](#)
- [Pushbutton](#)
- [10k ohm resistor](#)
- [Li-ion/LiPo battery](#)
- Voltage regulator:
 - [MCP1700-3302E](#)
 - [Ceramic capacitor – 100nF](#)
 - [Electrolytic capacitor – 1000uF](#)
- [Plastic box enclosure](#)

Schematic

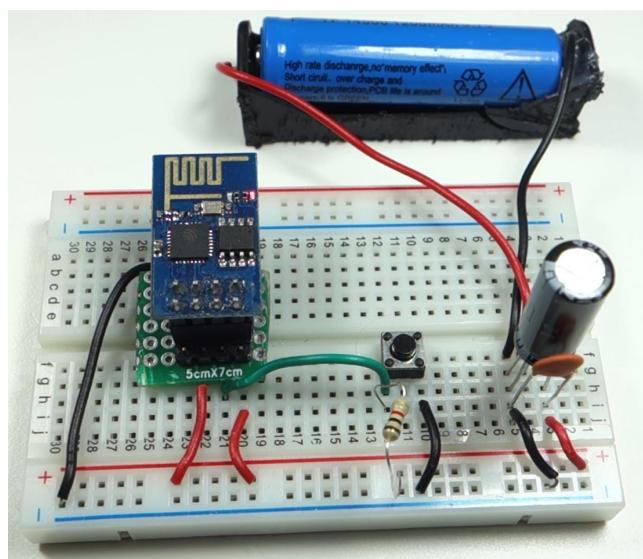
Here's the schematic diagram that you need to follow:



Here's the breadboard version:



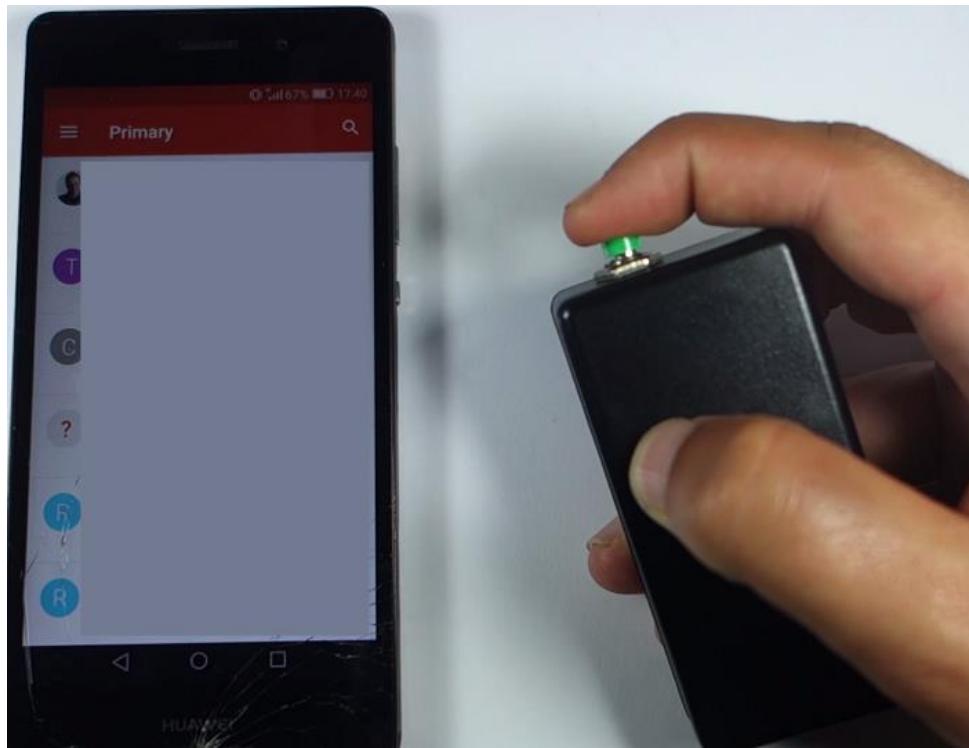
We recommend assembling the circuit first on a breadboard to test if it's working properly.



Then, you can make a permanent circuit with a small strip board, a few wires and plastic box enclosure to store everything.

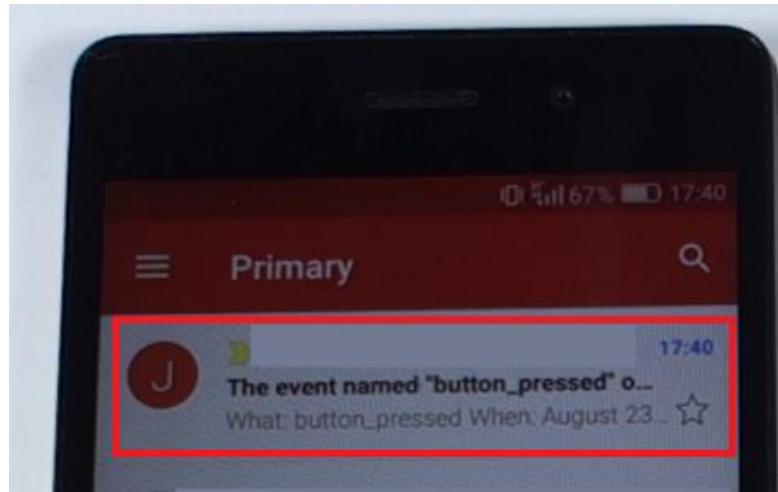


After assembling everything, here's how the final product looks like:



It kind of looks like a remote control. Now you can take it or place it anywhere.

Let's test it. When we press the pushbutton, we receive an email within a few seconds:

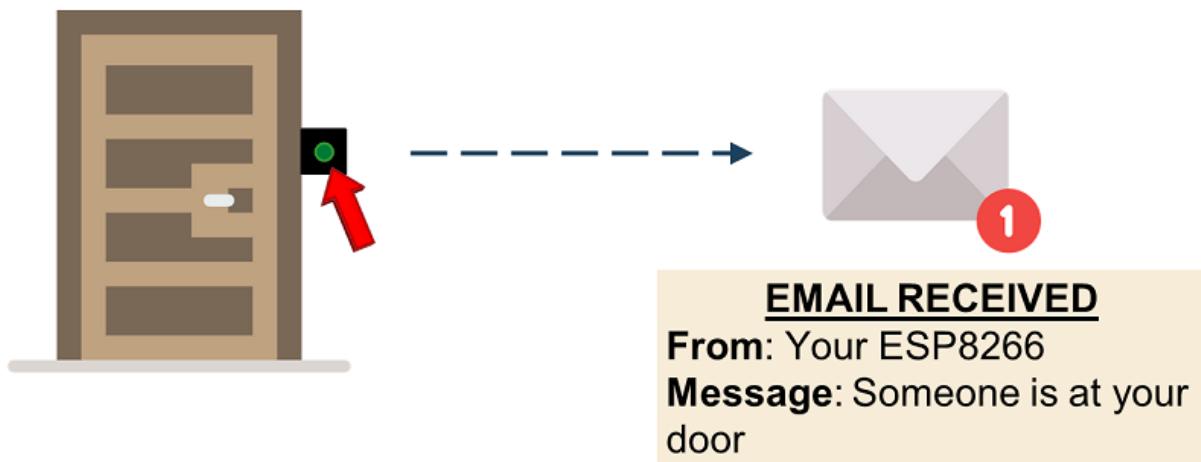


Note: even though the ESP8266 is powered with a battery, it can last weeks or even months, because it's always in Deep Sleep mode.

Wrapping Up

It's important to keep in mind that applications for this project are endless. For example, the event "**button_pressed**", depending on where you place your ESP8266 can have a different meaning.

If you place it as a doorbell button, you can use it to know if someone is at your home.



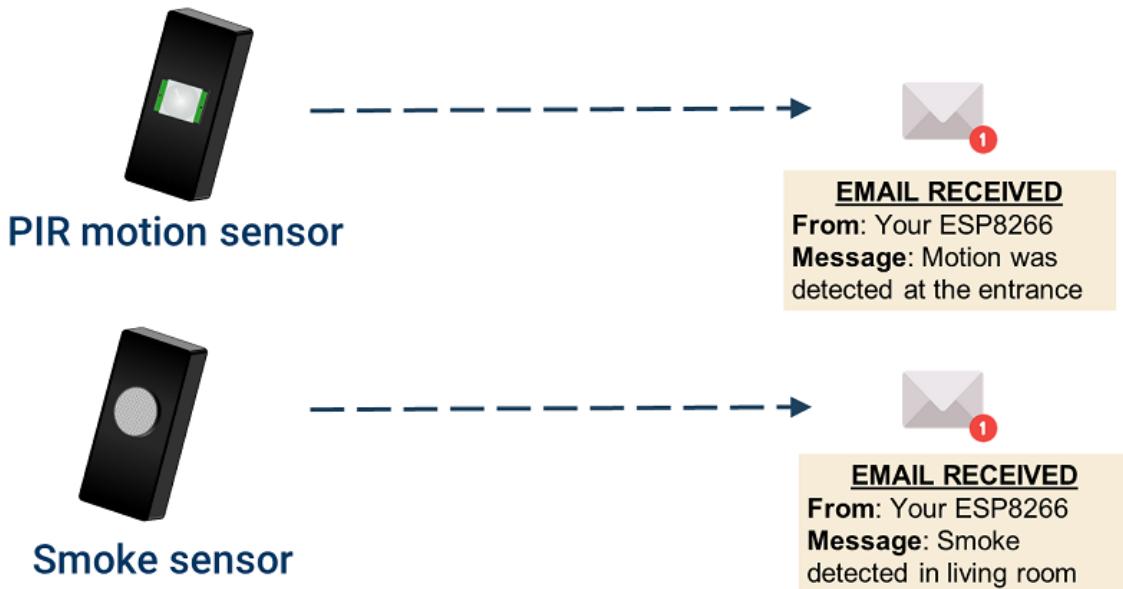
Also note that instead of using a 3rd party service like IFTTT, you can:

- Turn on relay that is connected to another ESP8266;
- Send a request to another device in your network;
- Make an HTTP request to Node-RED to trigger an action;
- Publish an MQTT message (you'll learn about MQTT in the next Module);
- Connect to any other home automation software.

Taking It Further

You can also replace the pushbutton with other sensors or actuators. For instance, if you replace the pushbutton with a PIR motion sensor, you can be notified when someone enters a room in your house. Or you can use it to detect smoke in a room:

Replace the pushbutton with a:



Replacing the pushbutton with a magnetic reed switch allows you to detect if someone opened a door or window. You can even attach it to a mailbox to see when you receive actual letters in the mail or other packages.

Replace the pushbutton with a:



Magnetic reed switch



EMAIL RECEIVED

From: Your ESP8266
Message: Entrance door was opened

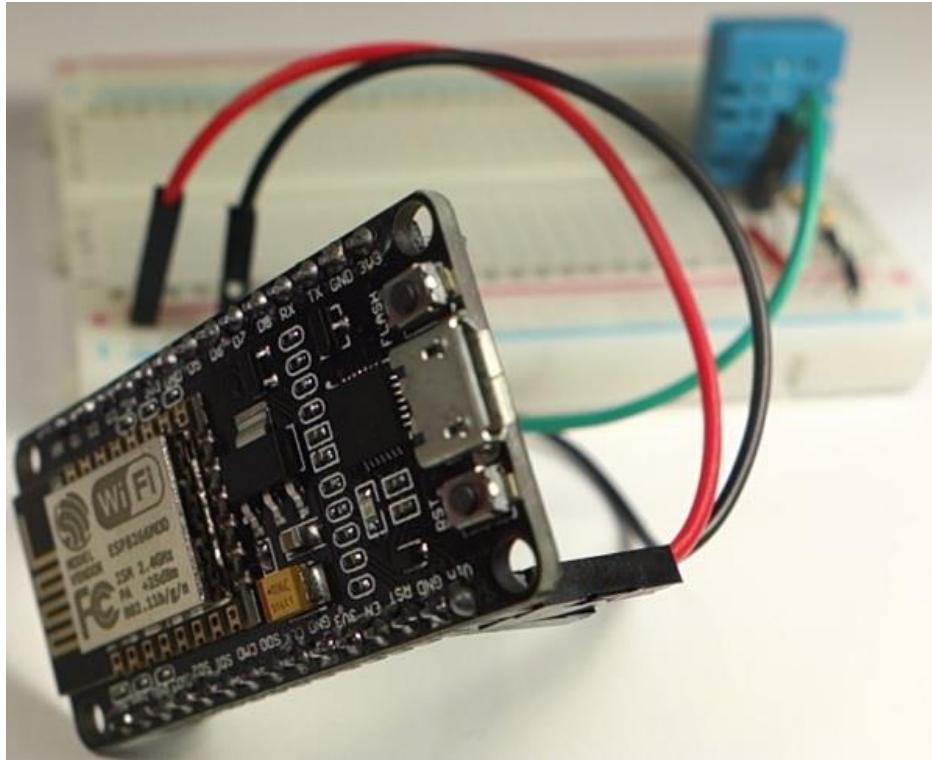


EMAIL RECEIVED

From: Your ESP8266
Message: New letter or package in your mail box

We hope you found this project interesting and you can apply these concepts to your own home automation projects.

Unit 5: ESP8266 Daily Task



In this project you're going to learn how to build a sensor node with an ESP8266 that publishes the temperature and humidity once a day to a free service called **ThingSpeak**. We'll be using a DHT11 temperature and humidity sensor and the ESP8266 will be in deep sleep mode, except to publish the readings.

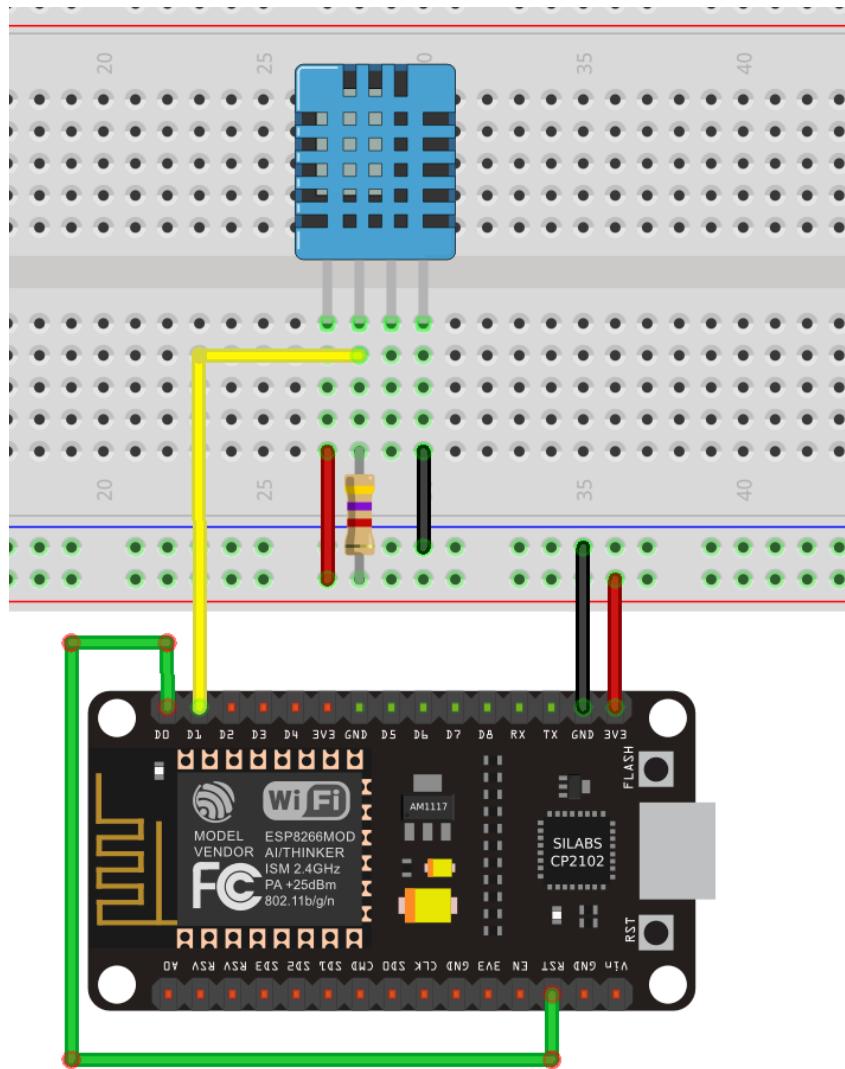
Parts Required

For this project, you need these components:

- [ESP8266](#)
- [DHT11 temperature and humidity sensor](#)
- [4.7k Ohm resistor](#)
- [Breadboard](#)
- [Jumper wires](#)

Schematic

Connect a DHT11 sensor to the ESP8266 as shown in the following schematic diagram. The data pin is connected to GPIO 5. Also, you need to add a wire to connect GPIO 16 and RST so that the ESP8266 is able to wake up from deep sleep.

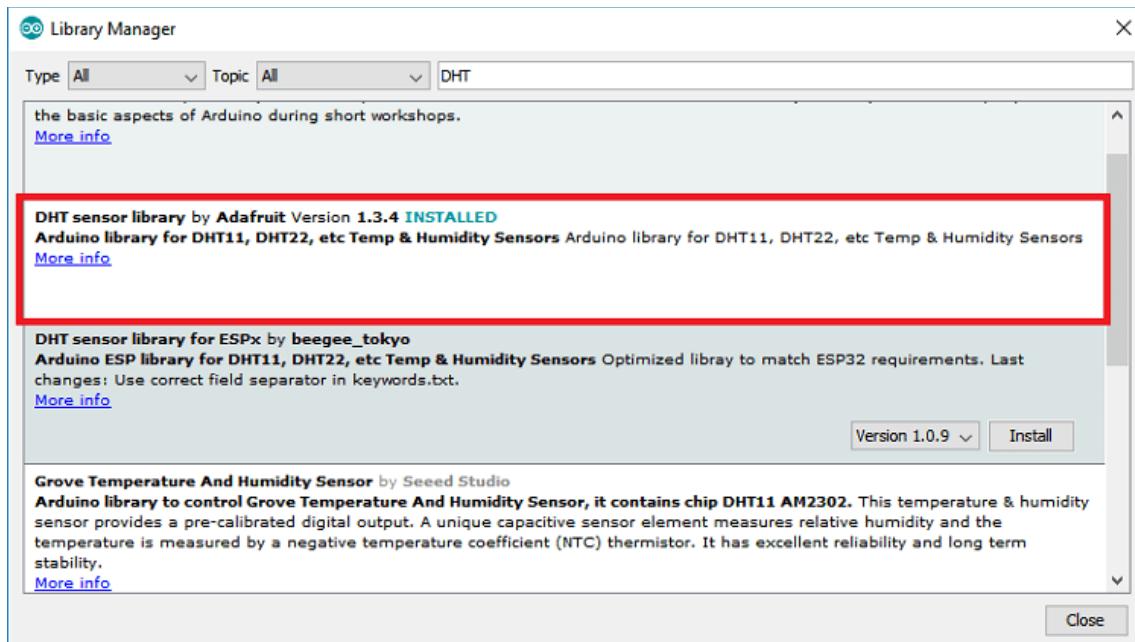


Preparing the Arduino IDE

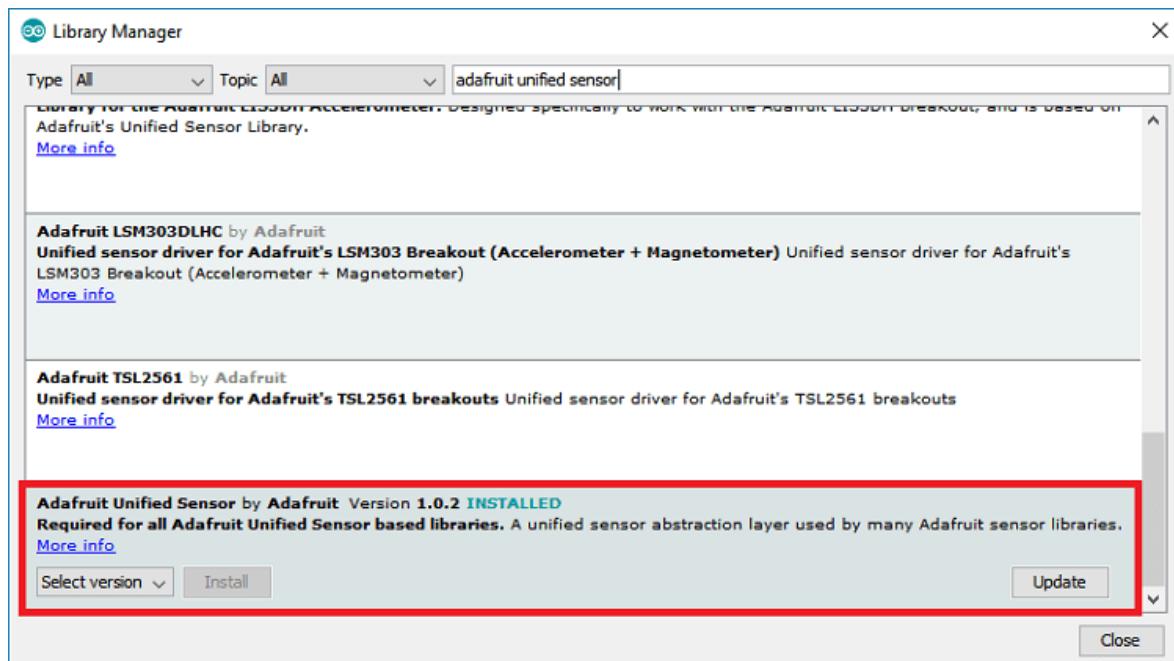
For this project you need to install the libraries to interface with the DHT sensor and the daily task library. If you've followed previous Units, you should already have the DHT libraries installed. If don't, follow the next steps to install them:

Go to **Sketch > Include library > Manage libraries...**

Search for “**DHT**” on the Search box and install the DHT library from Adafruit.



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.

Then, you also need to install the **ESP Daily Task** library. Follow the next instructions to install this library.

- 1) [Click here to download the ESP Daily Task library](#). You should have a .zip folder in your *Downloads* folder.
 - 1) Unzip the .zip folder and you should get *ESPDailyTask-master* folder.
 - 2) Rename your folder from *ESPDailyTask-master* to **ESPDailyTask**.
 - 3) Move the *ESPDailyTask* folder to your Arduino IDE installation libraries folder
 - 4) Finally, re-open your Arduino IDE.

Alternatively, after downloading the library, you can go to **Sketch ▶ Include Library ▶ Add .ZIP library** and select the library you've just downloaded.

ThingSpeak

For this project we'll be using [ThingSpeak.com](#). ThingSpeak allows you to publish your sensor readings to their website and display them in a plot with time stamps. Then, you can access your readings from anywhere in the world.



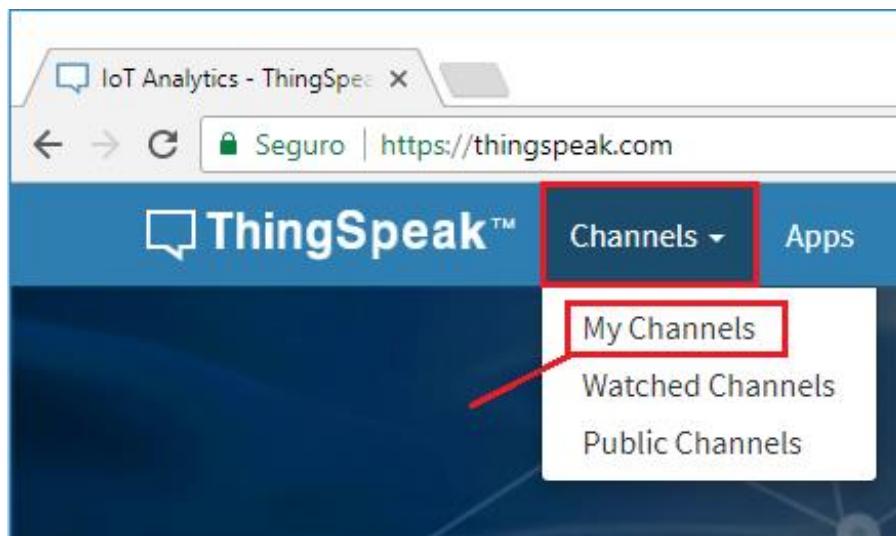
It's worth saying that this project can be easily modified to publish the values to your home automation hub or another application instead of ThingSpeak.

The main concept we want to show you with this project is how you can put your ESP8266 constantly in deep sleep mode, publishing a value every 24 hours.

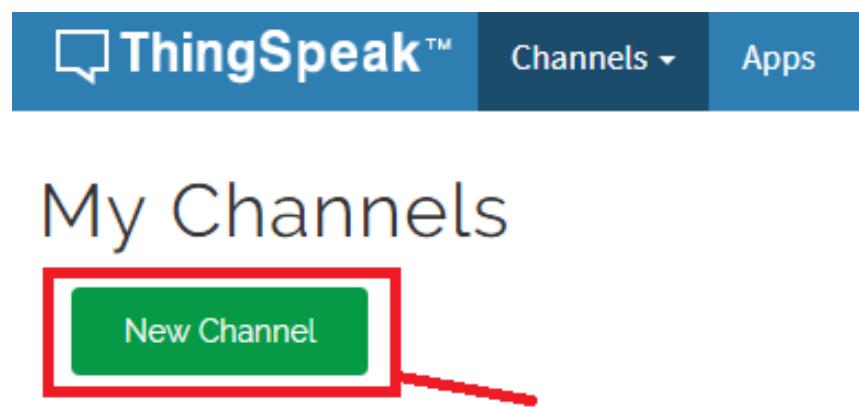
In the ThingSpeak webpage, click the “**Get Started For Free**” button to create a new account. This account is linked to a Mathworks account. So, if you already have a Mathworks account, you should login with that account.

Creating new channel

After your account is ready, sign in and open the “**Channels**” tab.



Press the create “**New Channel**” button:



Type a name for your channel, add a description, and enable a second field. Field 1 will receive the **Humidity** readings and Field 2 the **Temperature** readings. Once you've completed all these fields, save your channel.

New Channel

Name 

Description

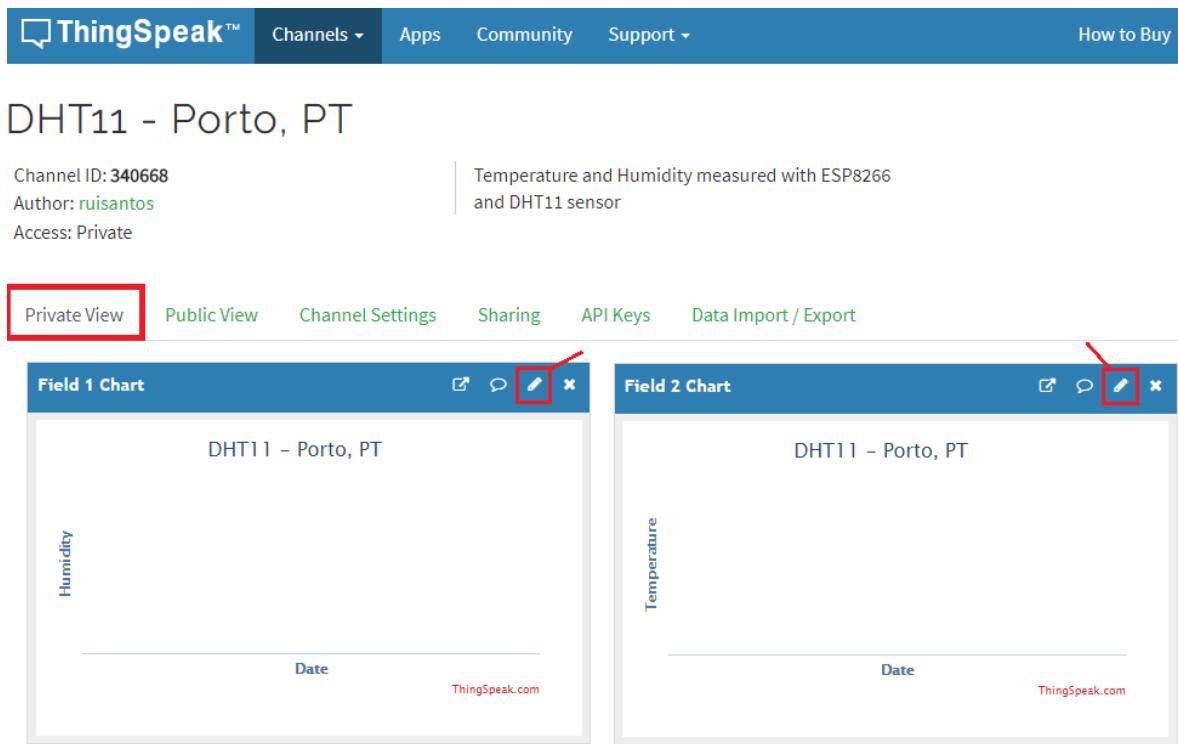
Field 1

Field 2

Save Channel 

Customizing chart

These charts can be customized, go to your Private View tab and press the edit icon:



You can give each chart a title, for example the first chart will be called **Humidity %** and the second chart will be called **Temperature °C**.

Field 1 Chart Options

Title:	Humidity %	Timescale:
X-Axis:		Average:
Y-Axis:		Median:
Color:	#d62020	Sum:
Background:	#ffffff	Rounding:
Type:	line	Data Min:
Dynamic?:	true	Data Max:
Days:		Y-Axis Min:
Results:	60	Y-Axis Max:

Save **Cancel**

You can also customize the background color, x and y axis, and much more. When you're done, press the "**Save**" button.

Field 2 Chart Options

Title:	Temperature °C	Timescale:
X-Axis:		Average:
Y-Axis:		Median:
Color:	#d62020	Sum:
Background:	#ffffff	Rounding:
Type:	line	Data Min:
Dynamic?:	true	Data Max:
Days:		Y-Axis Min:
Results:	60	Y-Axis Max:

Save **Cancel**

Write API Key

To publish values with the ESP8266, you need the **Write API Key**. Open the “**API Keys**” tab and copy the Write API Key to a safe place because you’ll need it in a moment.

The screenshot shows a ThingSpeak channel page for "DHT11 - Porto, PT". At the top, there's a navigation bar with "ThingSpeak™", "Channels", "Apps", "Community", and "Support". Below the navigation, the channel title "DHT11 - Porto, PT" is displayed. To the left, channel details are shown: "Channel ID: 340668", "Author: ruisantos", and "Access: Private". To the right, a description states "Temperature and Humidity measured with ESP8266 and DHT11 sensor". Below the title, there are tabs: "Private View", "Public View", "Channel Settings", "Sharing", "API Keys" (which is highlighted with a red box), and "Data Import / Export". Under the "API Keys" tab, the "Key" field contains the value "DDJFAGMAXNUXGUKS", which is also highlighted with a red box. A button labeled "Generate New Write API Key" is visible. On the right side, there's a "Help" section with a brief description of API keys and a "API Keys S" section with a bulleted list:

- Write API Keys: These keys have been compromised and are auto-generated.
- Read API Keys: These keys are used for reading feeds and can be disabled.

Code

Open your Arduino IDE and copy the following code but don't upload it yet. You need to change some variables to make it work for you.

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include "DHT.h"

// Uncomment one of the lines below for whatever DHT sensor type you're
// using!
#define DHTTYPE DHT11    // DHT 11
// #define DHTTYPE DHT21    // DHT 21 (AM2301)
// #define DHTTYPE DHT22    // DHT 22  (AM2302), AM2321

// Replace with your SSID and Password
const char* ssid      = "REPLACE_WITH_YOUR_SSID";
```

```

const char* password = "REPLACE_WITH_YOUR_PASSWORD";

// Replace with your unique Thing Speak WRITE API KEY
const char* apiKey = "REPLACE_WITH_YOUR_ThingSpeak_WRITE_API_KEY";

const char* resource = "/update?api_key=";

// Thing Speak API server
const char* server = "api.thingspeak.com";

// Set this for what time your daily code should run
ESP8266Task dailyTask(11*60 + 15); // 11:15am

// DHT Sensor
const int DHTPin = 5;
// Initialize DHT sensor.
DHT dht(DHTPin, DHTTYPE);

// Temporary variables
static char temperatureTemp[7];
static char humidityTemp[7];

void setup() {
    // Initializing serial port for debugging purposes
    Serial.begin(115200);
    delay(10);

    dailyTask.sleep1Day();

    // put your daily code here...
    dht.begin();

    initWifi();
    makeHTTPRequest();

    // and back to sleep once daily code is done
    dailyTask.backToSleep();
}

void loop() {
    // sleeping so wont get here
}

// Establish a Wi-Fi connection with your router
void initWifi() {
    Serial.print("Connecting to: ");
    Serial.print(ssid);
    WiFi.begin(ssid, password);

    int timeout = 10 * 4; // 10 seconds
    while(WiFi.status() != WL_CONNECTED && (timeout-- > 0)) {
        delay(250);
        Serial.print(".");
    }
    Serial.println("");
}

```

```

if(WiFi.status() != WL_CONNECTED) {
    Serial.println("Failed to connect, going back to sleep");
}

Serial.print("WiFi connected in: ");
Serial.print(millis());
Serial.print(", IP address: ");
Serial.println(WiFi.localIP());
}

// Make an HTTP request to the Node-RED software
void makeHTTPRequest() {
    // Sensor readings may also be up to 2 seconds 'old' (its a very slow
sensor)
    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);
    // Check if any reads failed and exit early (to try again).
    if (isnan(h) || isnan(t) || isnan(f)) {
        Serial.println("Failed to read from DHT sensor!");
        strcpy(temperatureTemp, "Failed");
        strcpy(humidityTemp, "Failed");
        return;
    }
    else {
        // Computes temperature values in Celsius + Fahrenheit and Humidity
        float hic = dht.computeHeatIndex(t, h, false);
        // Comment the next line, if you prefer to use Fahrenheit
        dtostrf(hic, 6, 2, temperatureTemp);

        float hif = dht.computeHeatIndex(f, h);
        // Uncomment the next line, if you want to use Fahrenheit
        //dtostrf(hif, 6, 2, temperatureTemp);

        dtostrf(h, 6, 2, humidityTemp);
        // You can delete the following Serial.print's, it's just for
debugging purposes
        Serial.print("Humidity: ");
        Serial.print(h);
        Serial.print(" %\t Temperature: ");
        Serial.print(t);
        Serial.print(" *C ");
        Serial.print(f);
        Serial.print(" *F\t Heat index: ");
        Serial.print(hic);
        Serial.print(" *C ");
        Serial.print(hif);
        Serial.print(" *F");
        Serial.print("Humidity: ");
        Serial.print(h);
        Serial.print(" %\t Temperature: ");
        Serial.print(t);
    }
}

```

```

    Serial.print(" *C ");
    Serial.print(f);
    Serial.print(" *F\t Heat index: ");
    Serial.print(hic);
    Serial.print(" *C ");
    Serial.print(hif);
    Serial.println(" *F");
}

Serial.print("Connecting to ");
Serial.print(server);

WiFiClient client;
int retries = 5;
while(!client.connect(server, 80) && (retries-- > 0)) {
    Serial.print(".");
}
Serial.println();
if(!client.connected()) {
    Serial.println("Failed to connect, going back to sleep");
}

Serial.print("Request resource: ");
Serial.println(resource);
client.print(String("GET ") + resource + apiKey + "&field1=" +
humidityTemp + "&field2=" + temperatureTemp +
" HTTP/1.1\r\n" +
"Host: " + server + "\r\n" +
"Connection: close\r\n\r\n");

int timeout = 5 * 10; // 5 seconds
while(!client.available() && (timeout-- > 0)){
    delay(100);
}
if(!client.available()) {
    Serial.println("No response, going back to sleep");
}
while(client.available()){
    Serial.write(client.read());
}

Serial.println("\nclosing connection");
client.stop();
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module5/Unit5/ESP8266_Daily_Task_ThingSpeak/ESP8266_Daily_Task_ThingSpeak.ino

Preparing your code

There are 3 variables that you need to change. Add your SSID and password so that your ESP8266 can establish a communication with your router:

```
// Replace with your SSID and Password
const char* ssid      = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Enter your ThingSpeak Write API key in the following variable:

```
// Replace with your unique Thing Speak WRITE API KEY
const char* apiKey = "REPLACE_WITH_YOUR_ThingSpeak_WRITE_API_KEY";
```

How the Code Works

The `setup()` function starts by initializing the Serial port.

```
void setup() {
    // Initializing serial port for debugging purposes
    Serial.begin(115200);
    delay(10);
```

Then, it calls the ESP8266 daily task function to determine whether 24 hours have passed or not since the last reading. If not, the ESP8266 goes back to deep sleep mode.

```
dailyTask.sleep1Day();
```

However, if 24 hours have passed since the last reading, the code will proceed. So, it will initialize the DHT sensor, connect to the Wi-Fi router and make an HTTP GET request to ThingSpeak in order to publish the readings.

```
// put your daily code here...
dht.begin();
initWifi();
makeHTTPRequest();
```

Once the readings are published, the ESP8266 goes back to deep sleep mode.

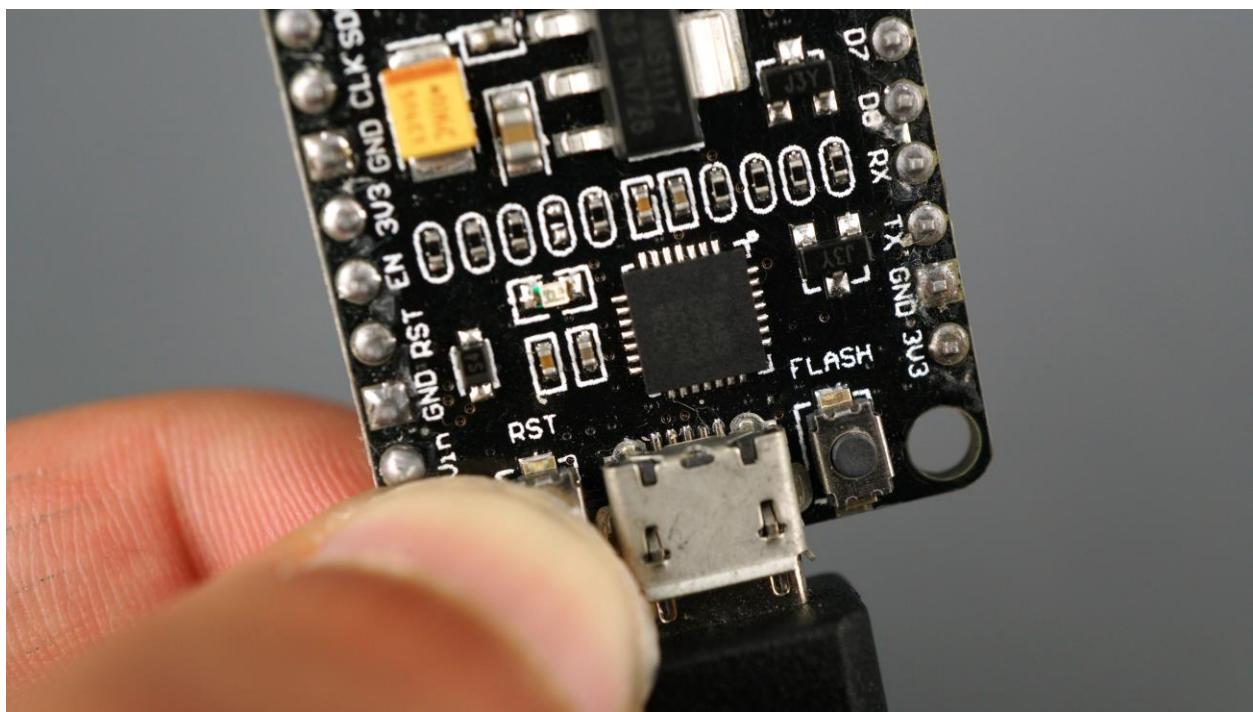
```
dailyTask.backToSleep();
```

There's nothing in the `loop()` function, because your ESP8266 is constantly in deep sleep mode and wakes itself up.

```
void loop() {
    // sleeping so wont get here
}
```

Testing the Project

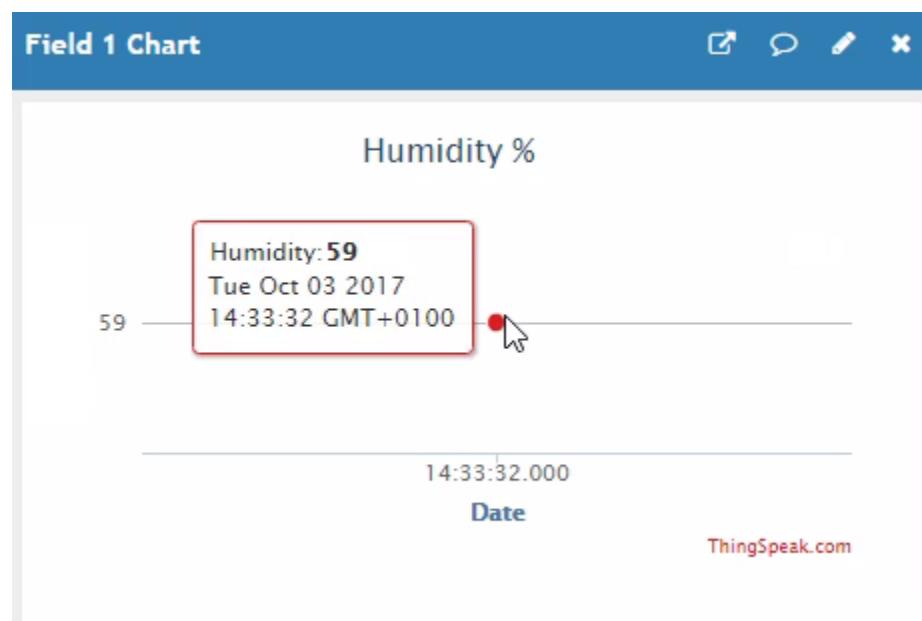
Finally, open the Arduino IDE serial monitor at a baud rate of 115200. You can test your project and simulate each hour passing. Simply remove the wire that connects the RESET pin to D0 (GPIO 16), and then you press the RESET button to simulate each hour passing.

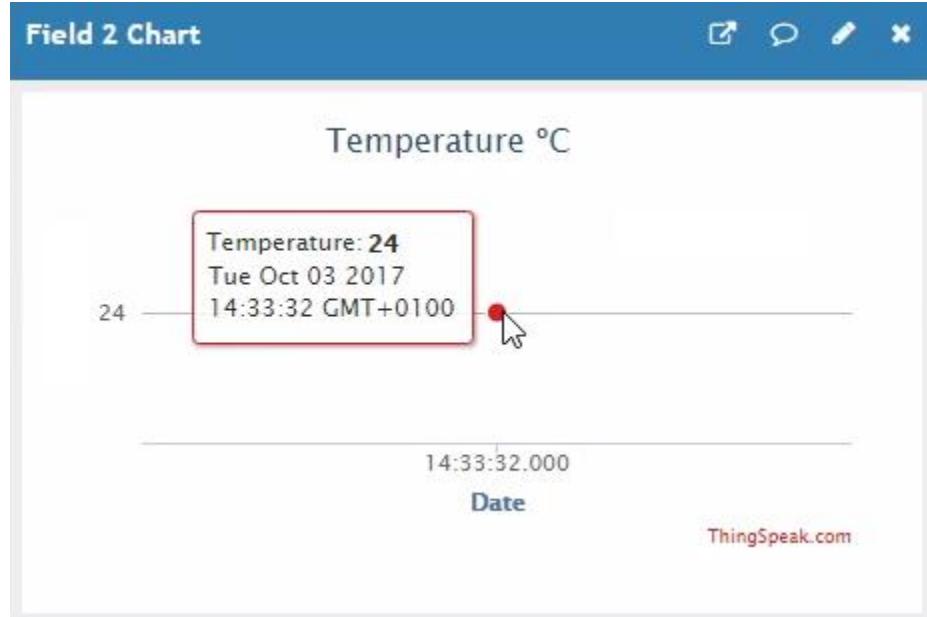


So, you need to press 24 times to publish a reading. The counter variable increases every time you press the reset button. Once you reset the ESP8266 24 times, it will publish the readings to ThingSpeak.

rtc marker: 126, counter: 20 sleepTime: 3600000000, thisSleepTime: 3600000000, fir
*** Up time: 107, deep sleeping for 3600000000 microseconds with WiFi on 0...
rl1Üž|ædà|øñññøeliç|ž, ñä"s>#çøB, òNNñlooüää#çp, ñ\$rlrlpòNåøñññf1løøøøBññ|ñ\$ññ
rtc marker: 126, counter: 21 sleepTime: 3600000000, thisSleepTime: 3600000000, fir
*** Up time: 108, deep sleeping for 3600000000 microseconds with WiFi on 0...
rlležø, \$à|øñññødä#|žfñi2'Ý|øbçøpøøoNÝdNNÜäiøcøpøødrlrlpúøäøññf1lø, øøøøbññ|ñ\$ññ
rtc marker: 126, counter: 22 sleepTime: 3600000000, thisSleepTime: 3600000000, fir
*** Up time: 105, deep sleeping for 3600000000 microseconds with WiFi on 0...
rllež|, lñ|øñññøeliøB|ž, ñipøÙr'øøøøbøp õNÝlnnÜää#|øpøžcl`ø{øpûoøøøø, ø1øøøøBññ|ñ\$ññ
rtc marker: 126, counter: 23 sleepTime: 3600000000, thisSleepTime: 3600000000, fir
*** Up time: 107, deep sleeping for 3600000000 microseconds with WiFi on 0...
r\$øøß|, lñ|øñññøsi#|øfñiø'rÛb, ñb, ñonožlNoüäiøcøp, ž#1 ø{øpønåøøøø, ø1ø, üñññ"ñøñ|ñ\$ññ
rtc marker: 126, counter: 24 sleepTime: 3600000000, thisSleepTime: 3600000000, fir
*** Up time: 105, waking up...
{l1Üž|ædà|øñññøeliç|ž, ñä"s>#çøB, ñ\$rlrlpòNåøñññf1løøøøBññ|ñ\$ññ
Connecting to: MEO-620B4B.....
<
 Autoscroll

As you can see, the Humidity and Temperature have been successfully published and plotted in the chart in real time.





After testing the circuit, you should reconnect the wire from RESET to D0. The ESP will wake up every hour, but it will only publish new readings every 24 hours.

I hope this project was useful. Now, you know how to apply the daily task concept to any other project.

Unit 6: SONOFF - \$5 WiFi Wireless Smart Switch

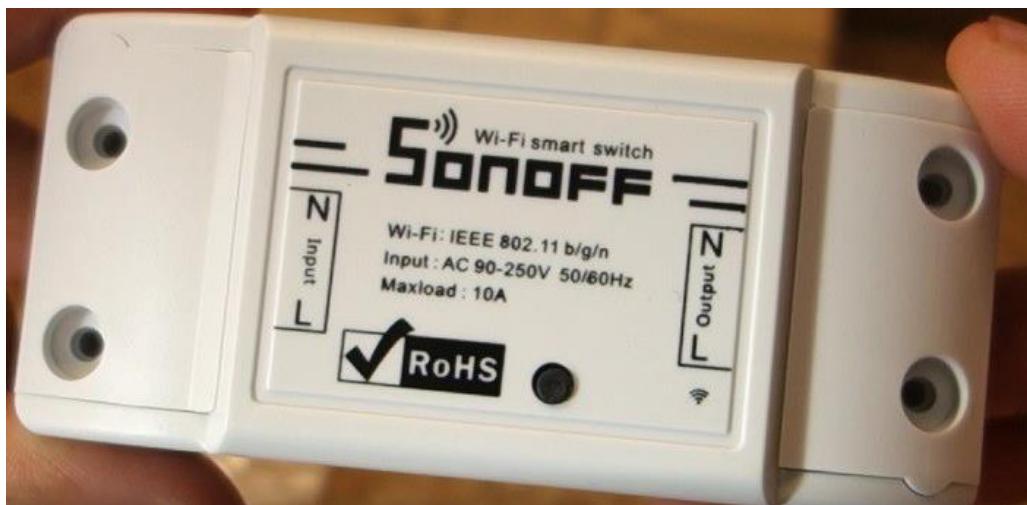


This is an extra Unit that shows how to use the SONOFF device with a local web server. The SONOFF is a device that has a built-in ESP8266. It is meant to put in series with your power lines allowing you to turn any device on and off.

First, you're going to install the default app that comes with the SONOFF device. Later, you'll learn how to reprogram the SONOFF with custom firmware.

SONOFF Overview

The following figure shows the [SONOFF WiFi smart switch](#) (costs approximately \$5).



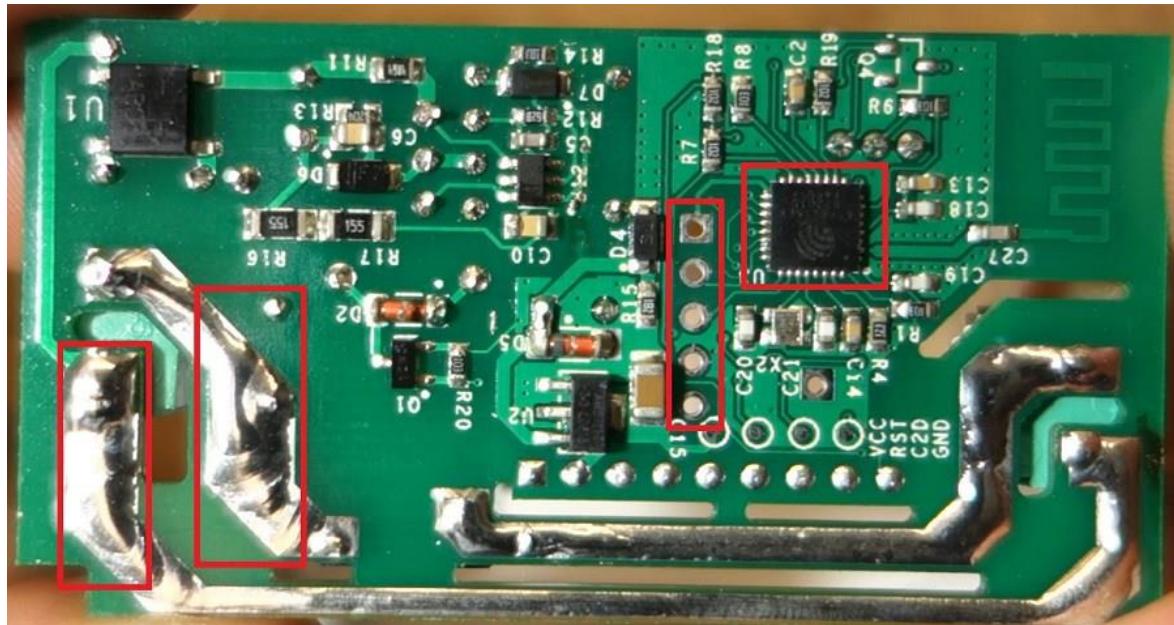
Its functioning is very simple, it has an input in one side and an output on the other side.

Then, you can simply send commands via WiFi to turn it on and off. That's pretty much how it works.

Opening the SONOFF

Let's look inside the SONOFF device. These are the main sections:

- There are **two powerlines** and they are isolated from the rest of the circuit;
- The **active line** goes to the **relay** (that's on the other side of the PCB);
- The **ESP8266**, which is the processor that provides WiFi and receives the control commands;
- The SONOFF is meant to be hacked and you can see clearly **5 connections** were left out, so that you can solder some pins and upload a custom firmware

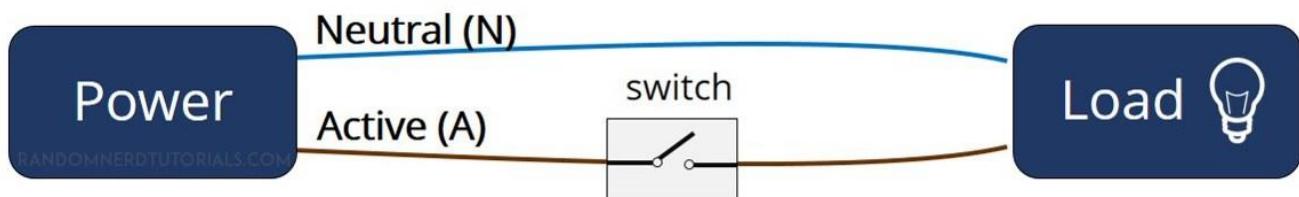


In the first part of this Unit you're going to use the standard firmware that comes with the SONOFF. Later, we're going to show how to flash a custom firmware into the SONOFF device.

SONOFF Example

Let's see how the SONOFF would fit in a normal circuit. Basically, you cut the wire that goes to the device, and you put the SONOFF in the middle, so that you can control any device that is connected on the other end.

Normally, what you have is a power source that has an active and neutral line that goes to a load, your load can be lamp for example. In the middle, you usually have a switch.



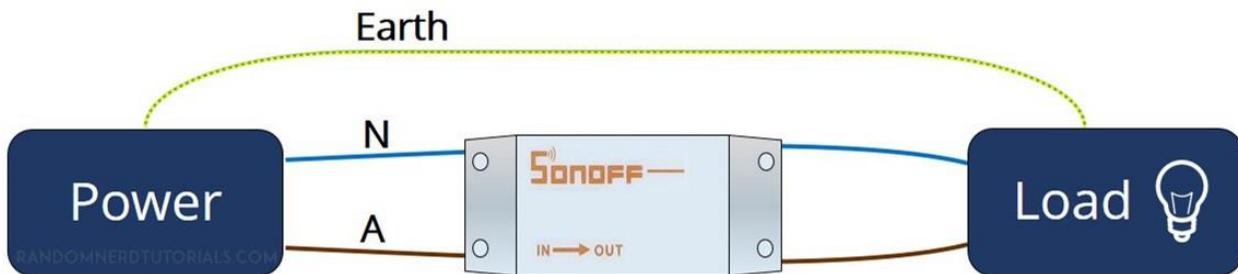
With the SONOFF, you cut that connection...



And you place the SONOFF in the middle. The SONOFF acts as a switch that is controlled via WiFi.



Note: if you have an earth line, it must go outside the SONOFF.



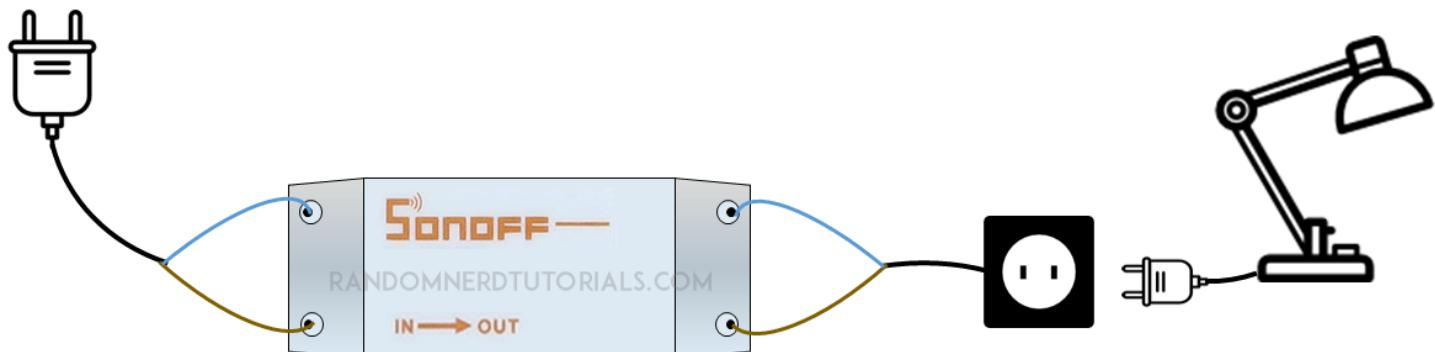
Safety Warning

Before proceeding with this project, we want to let you know that you're dealing with mains voltage. Please read the safety warning below carefully.



SONOFF Usage

Let's hook up the SONOFF. On the left side, you connect the active and neutral accordingly to the pinout. Active and neutral come out on the right.



On the left side, you have the input that connects to the outlet. The right side is the part that goes to your lamp/load.



Use your screwdriver to tighten the screws and have secure wire connection:



Place the two plastic protections and screw them.



After carefully checking all the connections, plug the male socket to the outlet.



On the other end, connect the female socket to the lamp.



Installing the App

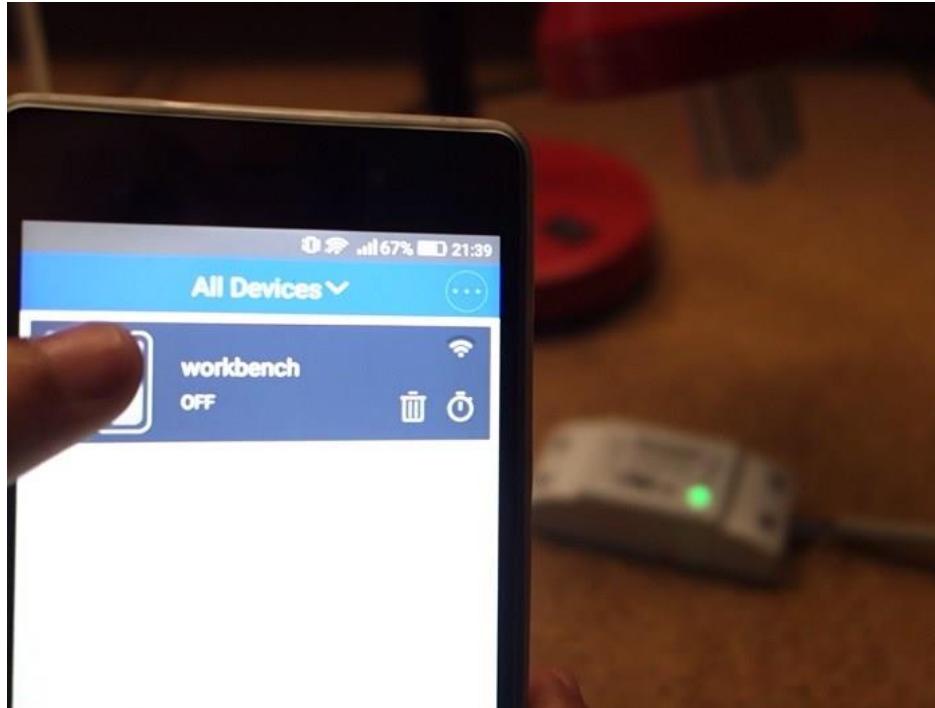
Now you have everything in place to install the app to control the light with your smartphone, follow these next instructions:

- Search for the app **eWeLink** (on the Play Store or App Store) and install it;
- Open the app and create an account;
- Power up the SONOFF device and connect the appliance that you want to control (in my case, it's a desktop lamp);
- Press and hold the SONOFF button for 5 seconds, so the green LED starts blinking;



- Go to the app and press the next button;
- Enter your network credentials and choose a name for your device;
- Add it to your dashboard.

Refresh the dashboard and you should see your device. Press the on button to turn it on. If you press off, the lamps turns off.

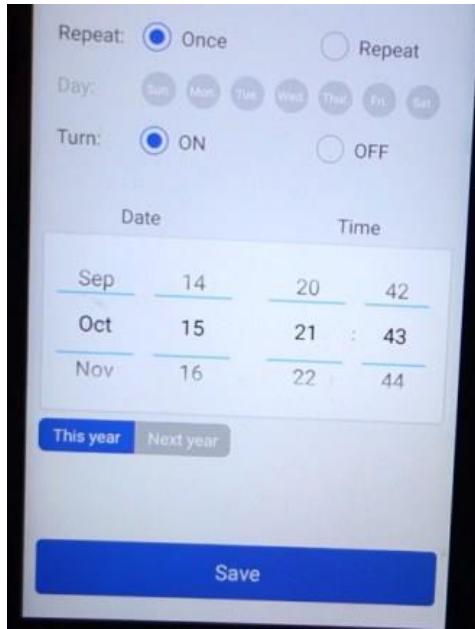


Watch the video to see a live demo of the SONOFF device:

https://youtu.be/mX97u_pQYnU.

Keep in mind that with this app you can control any device on and off from anywhere in the world, because it's controlled through the eWeLink cloud servers.

The app also comes with a nice set of features. You can add a timer that can be activated on a certain date and time. I've tested this feature and it has been working flawlessly.



Creating a Web Server for the SONOFF

To use the SONOFF with your own web server, you must flash custom firmware in the SONOFF device.

Safety Warning

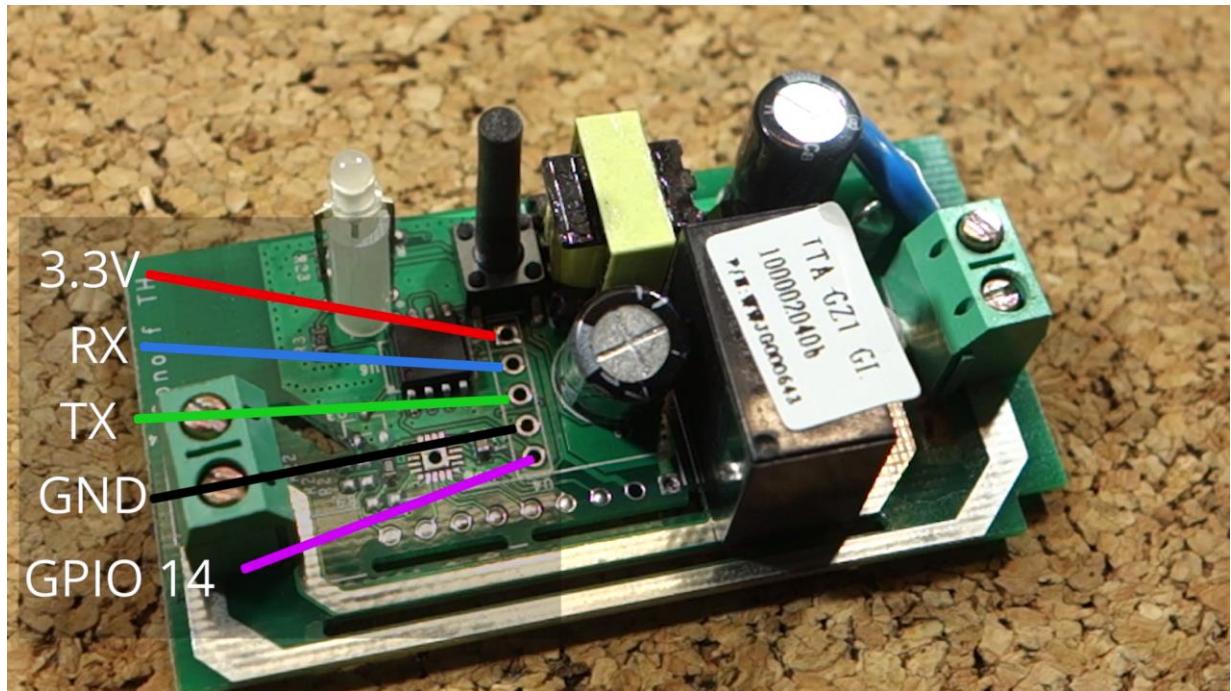
Make sure you disconnect your SONOFF from mains voltage while flashing a custom firmware. Don't touch any wires that are connected to mains voltage.



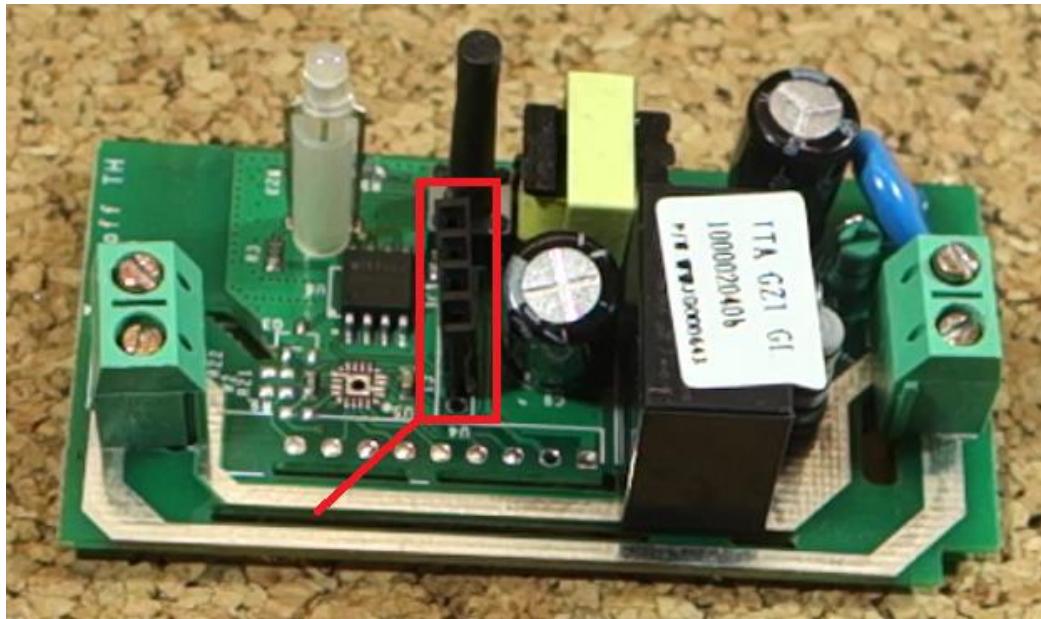
SONOFF Pinout

Open the SONOFF enclosure. As mentioned earlier, the SONOFF is meant to be hacked, and you can see clearly that some connections were left out, so that you can solder some pins and upload a custom firmware.

The following figure shows the pinout that you need to worry about.



We soldered 4 header pins, so that we can easily connect and disconnect wire cables to the SONOFF device.

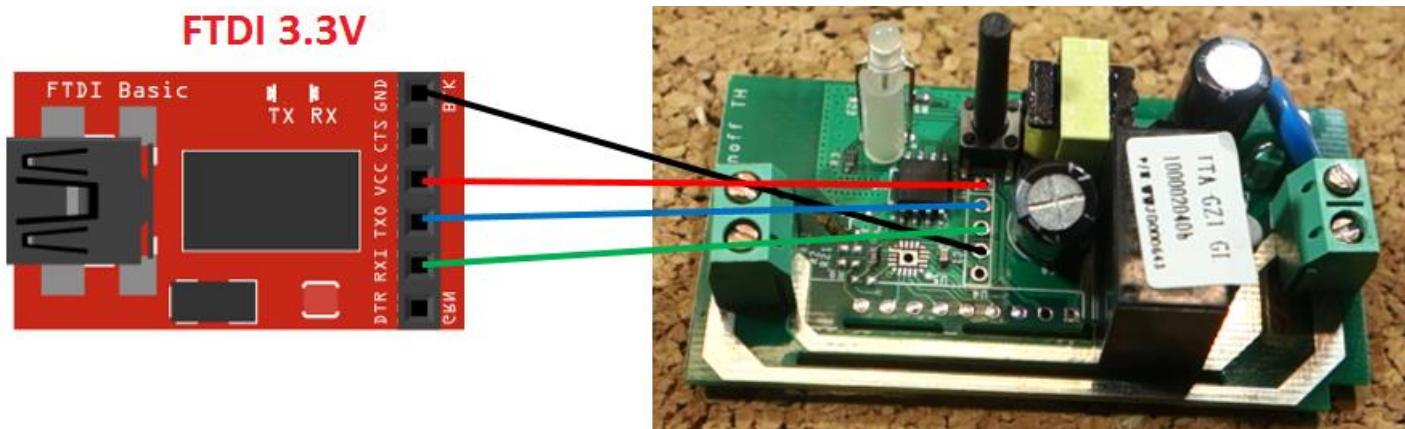


You need an FTDI module to upload a new firmware to your SONOFF.

Note: uploading a custom firmware is irreversible and you'll no longer be able to use the app eWeLink.

Connecting the FTDI to Your SONOFF

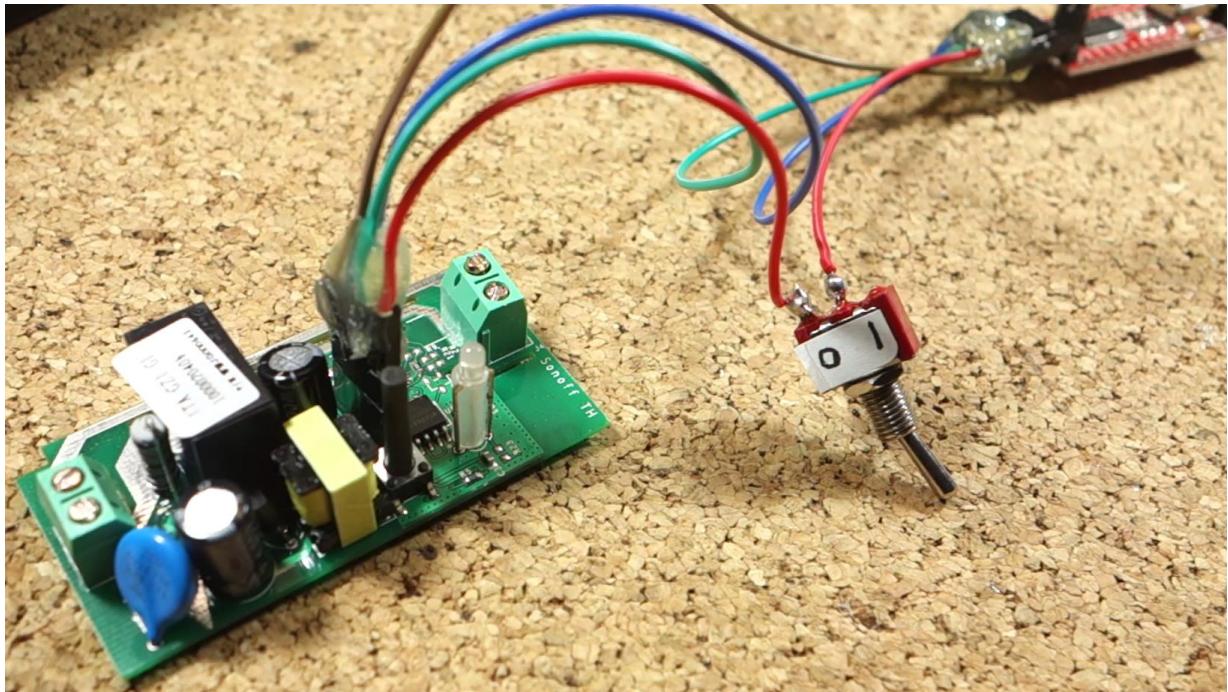
Follow the next schematic diagram to connect the FTDI programmer to the SONOFF.



- 3.3V → 3.3V
- TX → RX
- RX → TX
- GND → GND

We've added a toggle switch in the power line (3.3V), so that we can easily turn the SONOFF on and off to flash a new firmware without having to unplug the FTDI module.

Finally, connect the SONOFF to the FTDI, and connect them via USB to your computer to upload the new firmware.

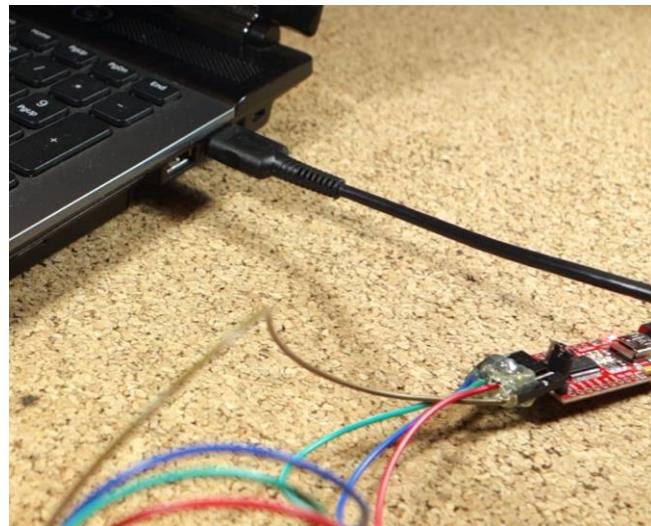


If you look closely to the previous figure, we used hot glue to glue the ends of the wires together. This prevents you to make wrong connections between the FTDI and the SONOFF in the future.

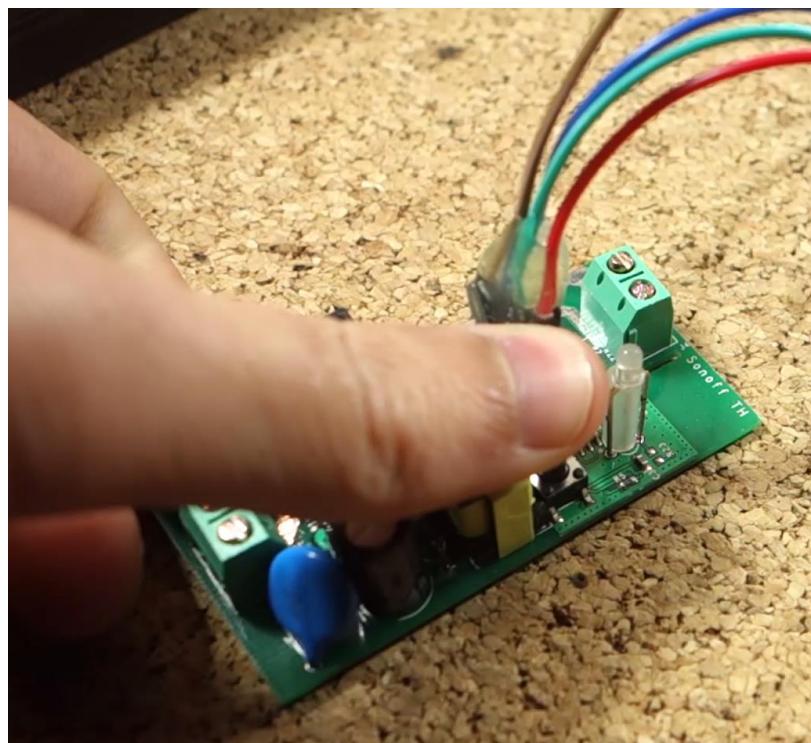
Boot Your SONOFF in Flashing Mode

To flash a new firmware to your SONOFF, you have to boot your SONOFF in flashing mode. Follow this 4-step process:

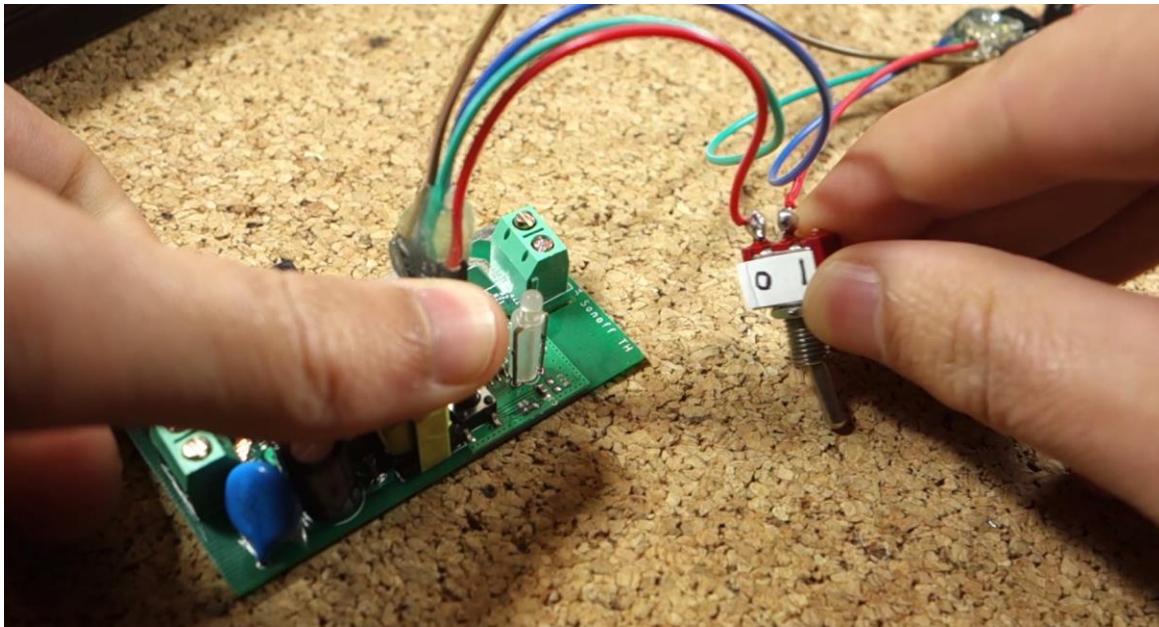
- 1) Connect your 3.3V FTDI programmer to your computer;



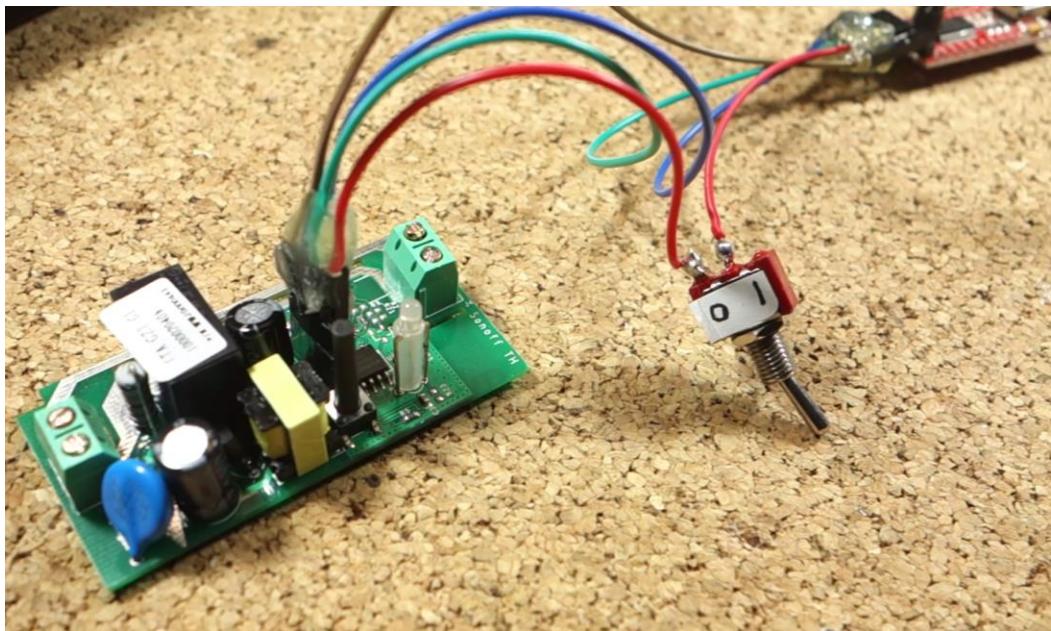
- 2) Hold down the SONOFF button;



3) Toggle the switch to apply power to the SONOFF circuit;



4) Then, you can release the SONOFF button.



Now, your SONOFF should be in flashing mode.

Code

Copy the following code to your Arduino IDE.

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>

MDNSResponder mdns;

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

ESP8266WebServer server(80);

String webPage = "";

int gpio13Led = 13;
int gpio12Relay = 12;

void setup(void) {
    webPage += "<h1>SONOFF      Web      Server</h1><p><a href=\"on\"><button>ON</button></a>&nbsp;<a href=\"off\"><button>OFF</button></a></p>";
    // preparing GPIOs
    pinMode(gpio13Led, OUTPUT);
    digitalWrite(gpio13Led, HIGH);

    pinMode(gpio12Relay, OUTPUT);
    digitalWrite(gpio12Relay, HIGH);

    Serial.begin(115200);
    delay(5000);
    WiFi.begin(ssid, password);
    Serial.println("");

    // Wait for connection
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.print("Connected to ");
    Serial.println(ssid);
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());

    if (mdns.begin("esp8266", WiFi.localIP())) {
        Serial.println("MDNS responder started");
    }
}
```

```

server.on("/", [](){
    server.send(200, "text/html", webPage);
});

server.on("/on", [](){
    server.send(200, "text/html", webPage);
    digitalWrite(gpio13Led, LOW);
    digitalWrite(gpio12Relay, HIGH);
    delay(1000);
});

server.on("/off", [](){
    server.send(200, "text/html", webPage);
    digitalWrite(gpio13Led, HIGH);
    digitalWrite(gpio12Relay, LOW);
    delay(1000);
});

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

void loop(void) {
    server.handleClient();
}

```

SOURCE CODE

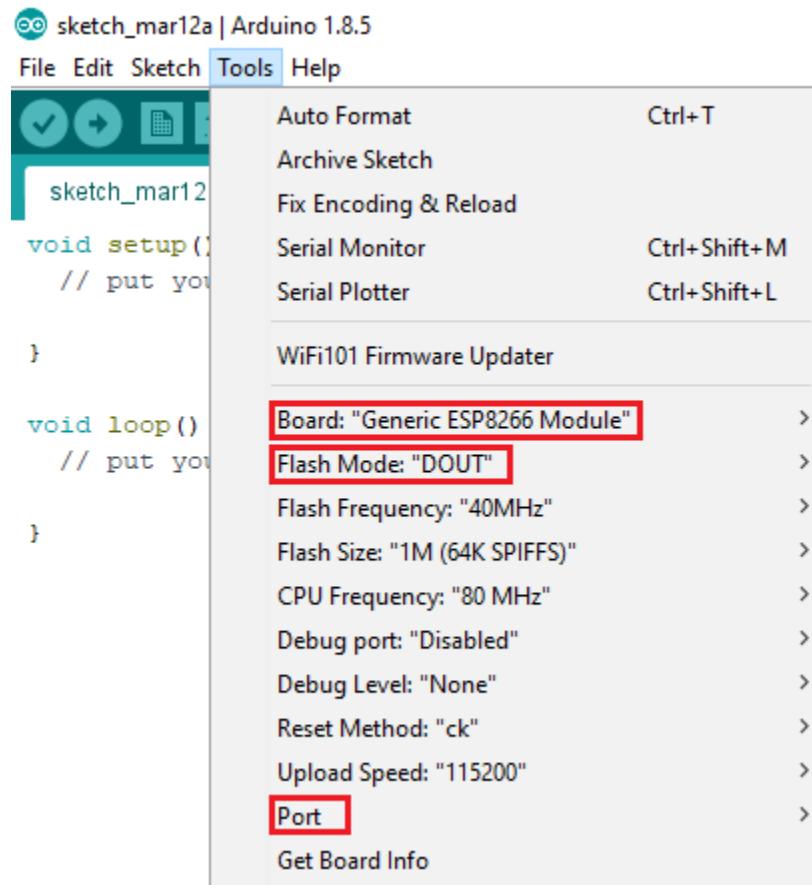
https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module5/Unit6/SONOFF_Web_Server/SONOFF_Web_Server.ino

Uploading Code

Upload the full sketch to your SONOFF (replace with your SSID and password):

Having your SONOFF device still in flashing mode, follow these next steps:

1. Select your FTDI port number under the **Tools > Port > COM14** (in our case);
2. Choose your ESP8266 board from **Tools > Board > Generic ESP8266 Module**;
3. Select **Flash Mode: “DOUT”**;
4. Press the Upload button in the Arduino IDE.



Wait a few seconds while the code is uploading. You should see a message saying "Done Uploading".

Troubleshooting

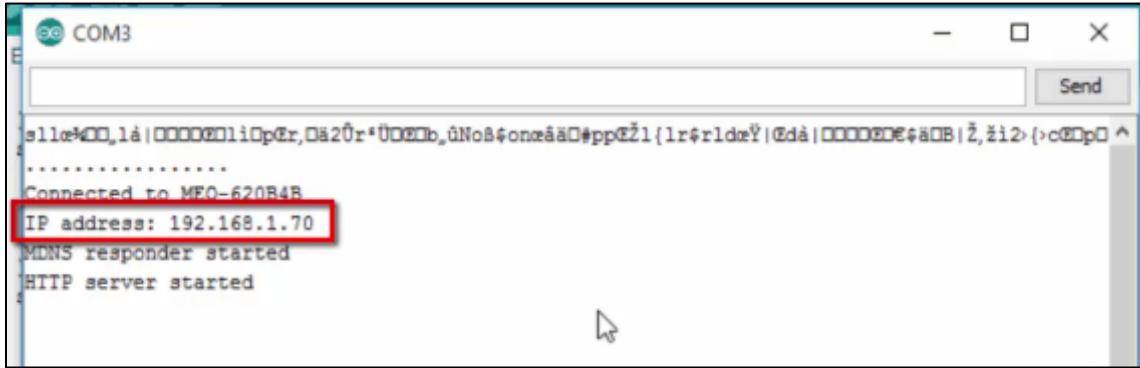
If you try to upload the sketch and it prompts the following error message:

```
warning: espcomm_sync failed
error: espcomm_open failed
```

Your SONOFF is not in flashing mode and you have to repeat the process described in section "Boot Your SONOFF in Flashing Mode" described earlier in this Unit.

ESP8266 IP Address

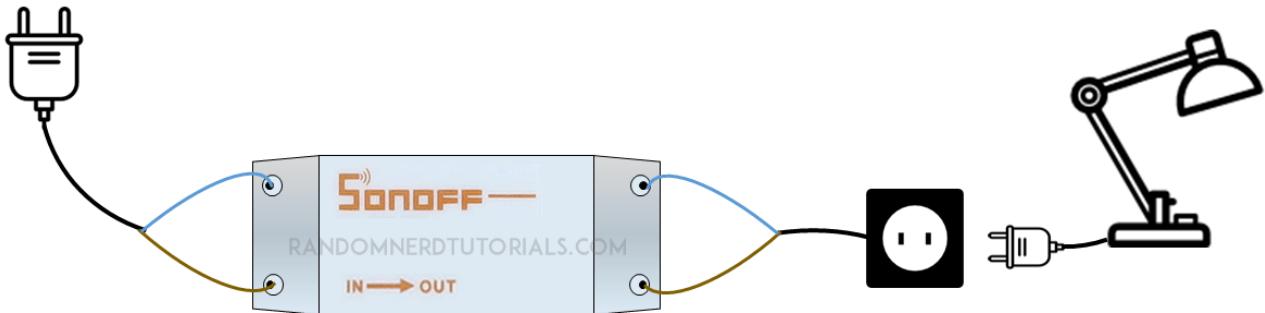
Open the Arduino serial monitor at a baud rate of 115200 and reset your board. After a few seconds your IP address should appear. In my case, it's **192.168.1.70**.



Serial monitor window showing the following text:
Connected to MFO-620B4B
IP address: 192.168.1.70
MDNS responder started
HTTP server started

Final Circuit

After uploading the code, re-assemble your SONOFF. Be very careful with the mains voltage connections. It's the exact same procedure as shown earlier in this Unit:



Demonstration

For the final demonstration open any browser from a device that is connected to the same router that your SONOFF is.

Then type the IP address and click Enter!



If you press the on switch, the lamp or any device that is connected to your SONOFF should turn on:



You can upload any other web server code to your Sonoff device.

MODULE 6:

MQTT

MQTT stands for Message Queuing Telemetry Transport and it is widely used as a communication protocol between IoT devices. In this Module, you'll learn how to exchange data between different ESP8266 boards: sensor readings, commands to control outputs, etc.

Unit 1: Introducing MQTT

This Unit is an introduction to MQTT and how to use it with the ESP8266.

MQTT stands for Message Queuing Telemetry Transport. It is a lightweight publish and subscribe system where you can publish and receive messages as a client.



MQTT is a simple messaging protocol, designed for constrained devices with low-bandwidth. It's the perfect solution for Internet of Things applications. MQTT allows you to send commands to control outputs, read and publish data from sensor nodes and much more (you'll understand how powerful MQTT is, after making the first projects). Therefore, it makes it really easy to establish a communication between multiple devices.

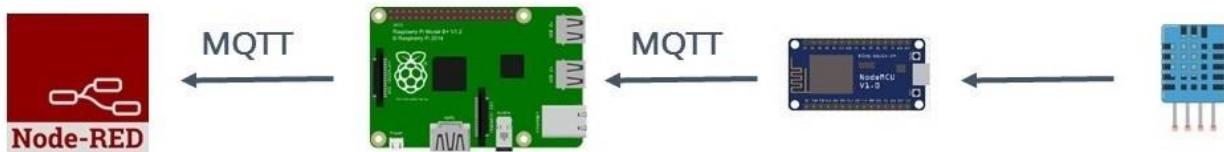
High Level Overview

Here's a quick high-level overview of what MQTT allows you to do.

You can send a command with a client (like Node-RED or another ESP8266) to control an output:



Or you can read data from a sensor and publish it to a client (like Node-RED or another ESP8266):



MQTT Basic Concepts

In MQTT there are a few basic concepts that you need to understand:

- Publish/Subscribe
- Messages
- Topics
- Broker

Publish/subscribe

The first concept is the ***publish and subscribe*** system. In a publish and subscribe system, a device can publish a message on a topic, or it can be subscribed to a topic to receive messages.



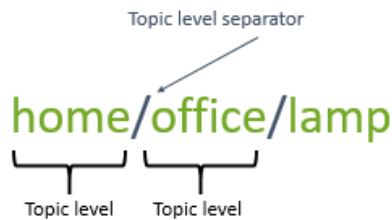
- For example, Device 1 publishes on a topic;
- Device 2 is subscribed to the same topic that Device 1 is publishing in;
- So, Device 2 receives the message.

Messages are pieces of information exchanged between your devices: whether it's a command or data.

Topics

Another important concept are the **topics**. Topics are the way you register interest for incoming messages or how you specify where you want to publish the messages.

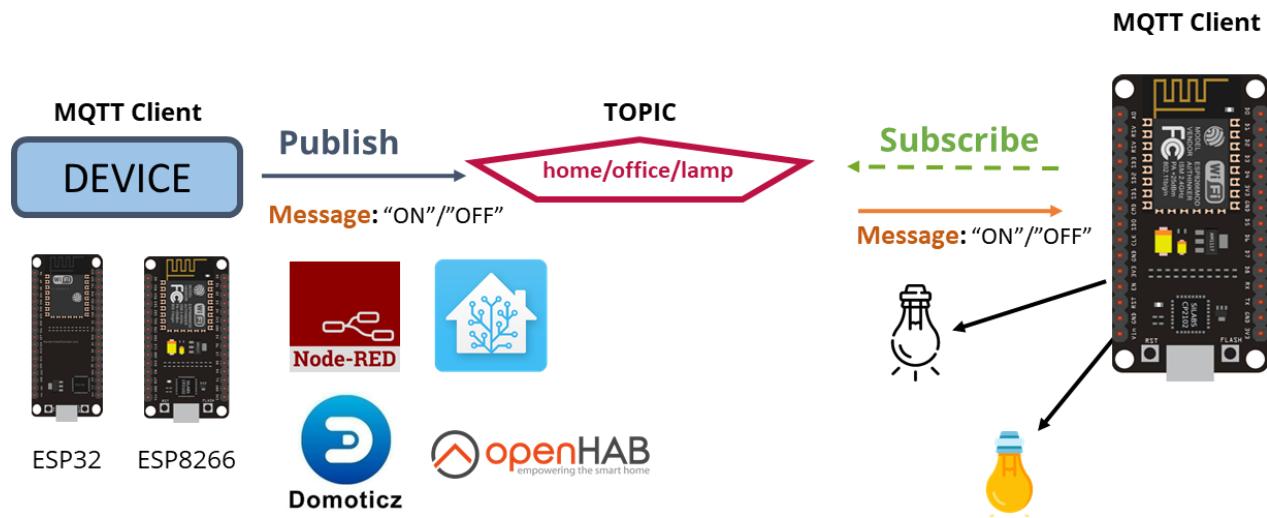
Topics are represented with strings separated by a forward slash. Each forward slash indicates the topic level. Here's an example on how you would create a topic for a lamp in your home office:



Note: topics are case-sensitive, which makes the following topics different.

home/office/lamp
≠
Home/Office/Lamp

If you would like to turn on a lamp in your home office using MQTT and the ESP8266 you can imagine the following scenario:

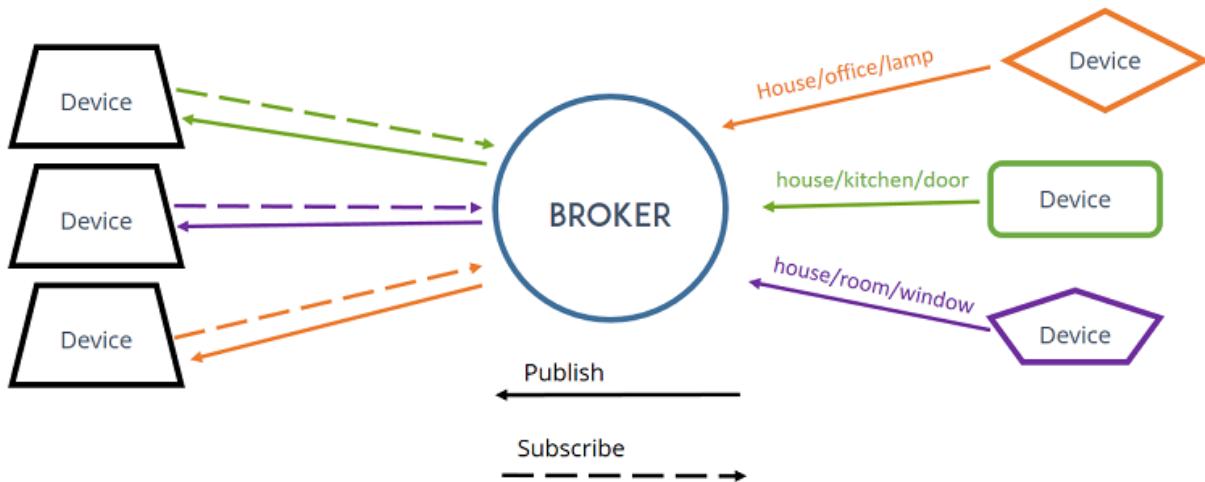


- You have a device that publishes “on” and “off” messages on the **home/office/lamp** topic.
- The ESP8266 that controls your lamp, is subscribed to that topic.
- Therefore, when a new message is published on that topic, the ESP8266 receives the “on” or “off” message and turns the lamp on or off.
- The first device can be an ESP32, an ESP8266, or a Home Automation controller platform like Node-RED, Home Assistant, Domoticz, or OpenHAB, for example.



Broker

Finally, you also need to be familiar with the term **broker**. The broker is primarily responsible for receiving all messages, filtering the messages, decide who is interested in them and then send the messages to all subscribed clients.



You can use several brokers. For example, you can use the **Mosquitto** broker hosted on a Raspberry Pi, or you can use a cloud MQTT broker.



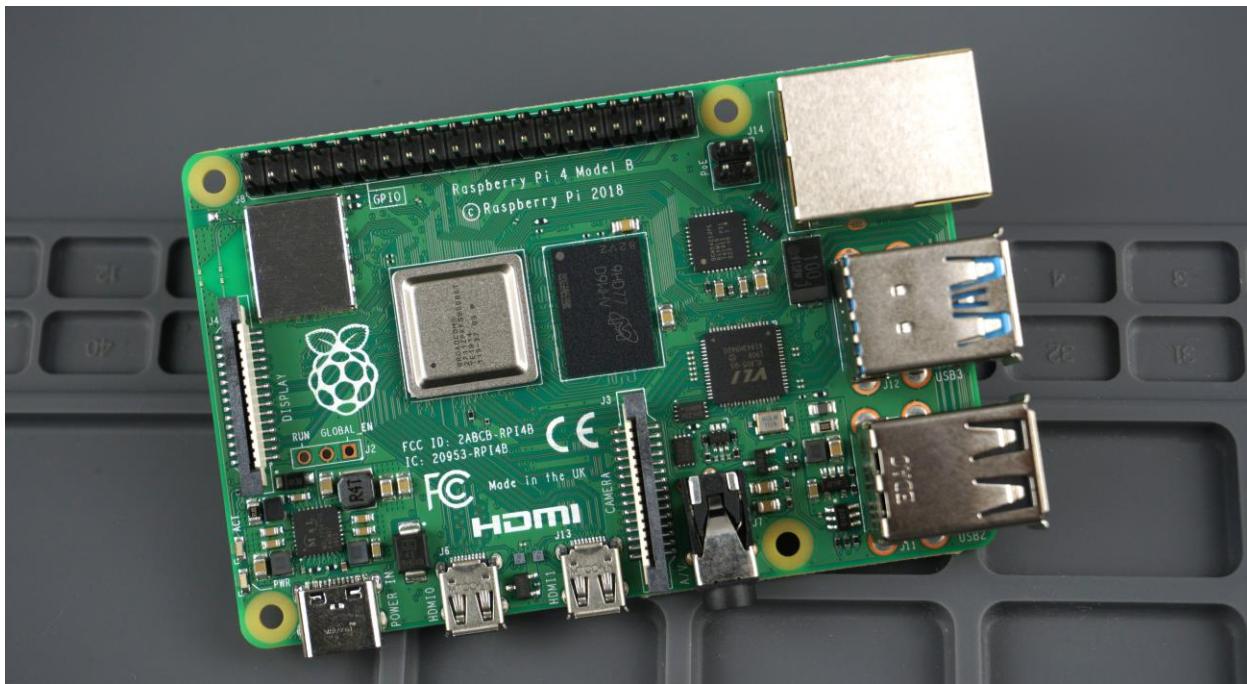
We're going to use the Mosquitto broker installed in a Raspberry Pi.

Wrapping Up

In summary:

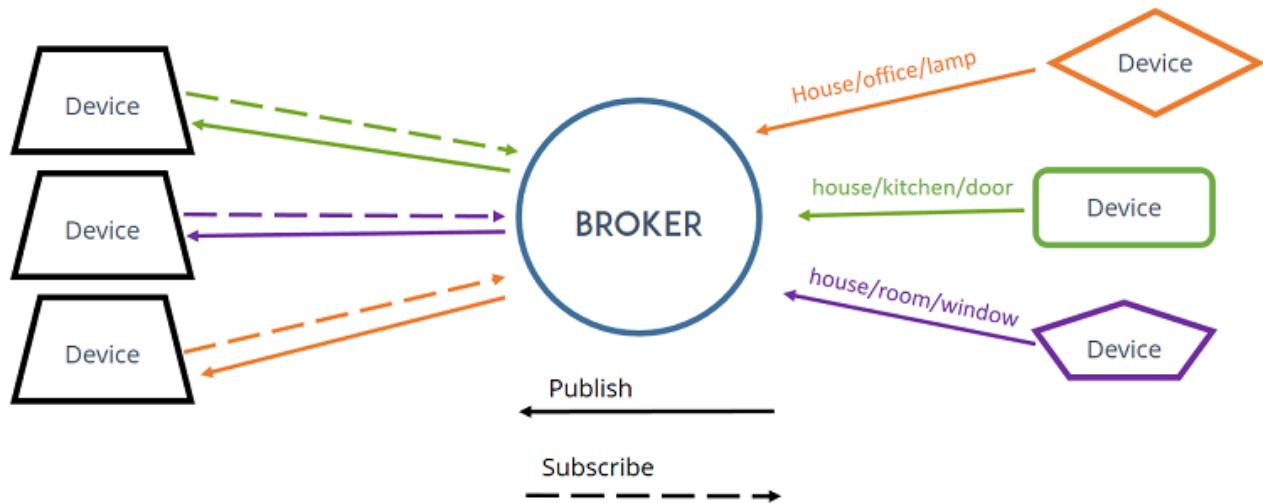
- MQTT is a communication protocol very useful in Internet of Things projects;
- In MQTT, devices can publish messages on specific topics and can be subscribed to other topics to receive messages;
- You need a broker when using MQTT. It receives all the messages and sends them to the subscribed devices.

Unit 2: Installing Mosquitto MQTT Broker on a Raspberry Pi



In this Unit, you're going to install the Mosquitto Broker on a Raspberry Pi.

The broker is primarily responsible for receiving all messages, filtering the messages, decide who is interested in them and then, publishing the message to all subscribed clients.



You can use several brokers. Throughout this eBook, we're going to use the [Mosquitto Broker](#) installed on a Raspberry Pi.

Note: you can also use a free [Cloud MQTT](#) broker (for a maximum of 5 connected devices).

Prerequisites

Before continuing with this tutorial:

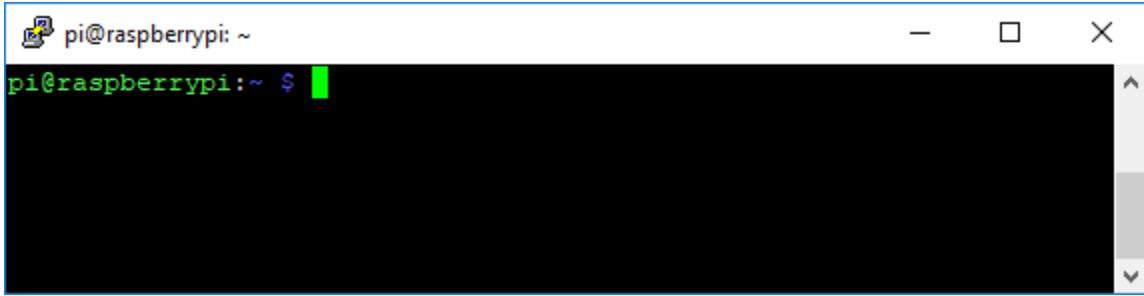
- You should be familiar with the Raspberry Pi board – read [Getting Started with Raspberry Pi](#);
- You should have the Raspbian or Raspbian Lite operating system installed in your Raspberry Pi – read [Installing Raspbian Lite, Enabling and Connecting with SSH](#);
- You also need the following hardware:
 - [Raspberry Pi board](#) – read [Best Raspberry Pi Starter Kits](#)
 - [MicroSD Card – 16GB Class10](#)
 - [Raspberry Pi Power Supply \(5V 2.5A\)](#)

After having your Raspberry Pi board prepared with Raspbian OS, you can continue with this Unit.

Installing Mosquitto Broker on Raspbian OS



Open a new Raspberry Pi terminal window:



To install the Mosquitto Broker enter these next commands:

```
pi@raspberry:~ $ sudo apt update  
pi@raspberry:~ $ sudo apt install -y mosquitto mosquitto-clients
```

To make Mosquitto auto start on boot up (this means Mosquitto will start automatically when you power your Raspberry Pi) enter:

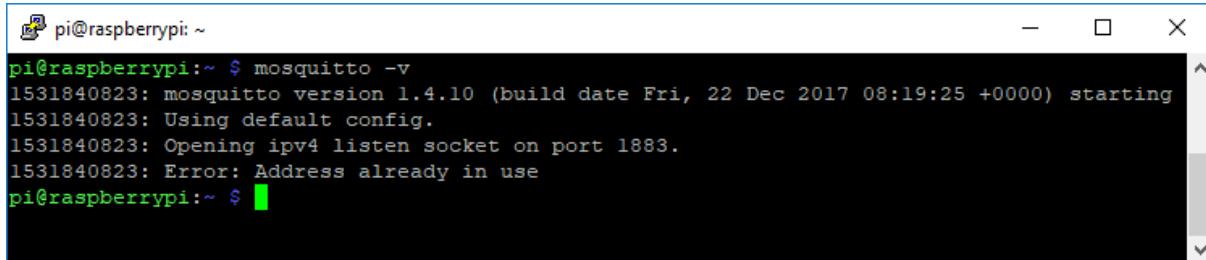
```
pi@raspberry:~ $ sudo systemctl enable mosquitto.service
```

Testing Installation

Send the command:

```
pi@raspberry:~ $ mosquitto -v
```

This returns the Mosquitto version that is currently running in your Raspberry Pi. It should be 1.4.X or above.



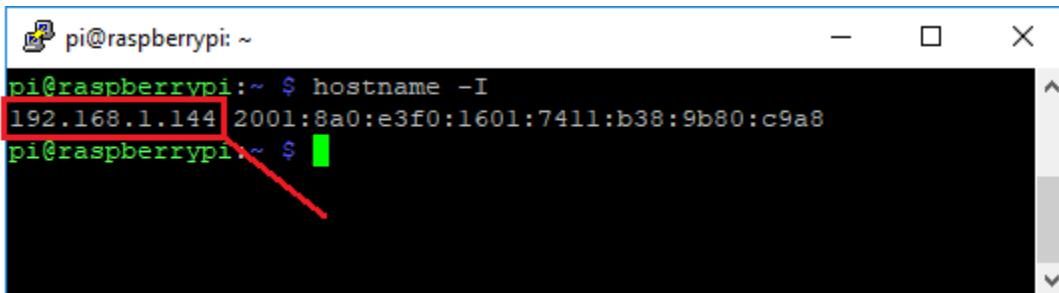
```
pi@raspberrypi:~ $ mosquitto -v  
1531840823: mosquitto version 1.4.10 (build date Fri, 22 Dec 2017 08:19:25 +0000) starting  
1531840823: Using default config.  
1531840823: Opening ipv4 listen socket on port 1883.  
1531840823: Error: Address already in use  
pi@raspberrypi:~ $
```

Note: sometimes the command `mosquitto -v` prompts a warning message saying "Error: Address already in use". That warning message means that your Mosquitto Broker is already running, so don't worry about that.

Raspberry Pi IP Address

To retrieve your Raspberry Pi IP address, type the next command in your Terminal window:

```
pi@raspberrypi:~ $ hostname -I
```



```
pi@raspberrypi:~ $ hostname -I
192.168.1.144 2001:8a0:e3f0:1601:7411:b38:9b80:c9a8
pi@raspberrypi:~ $
```

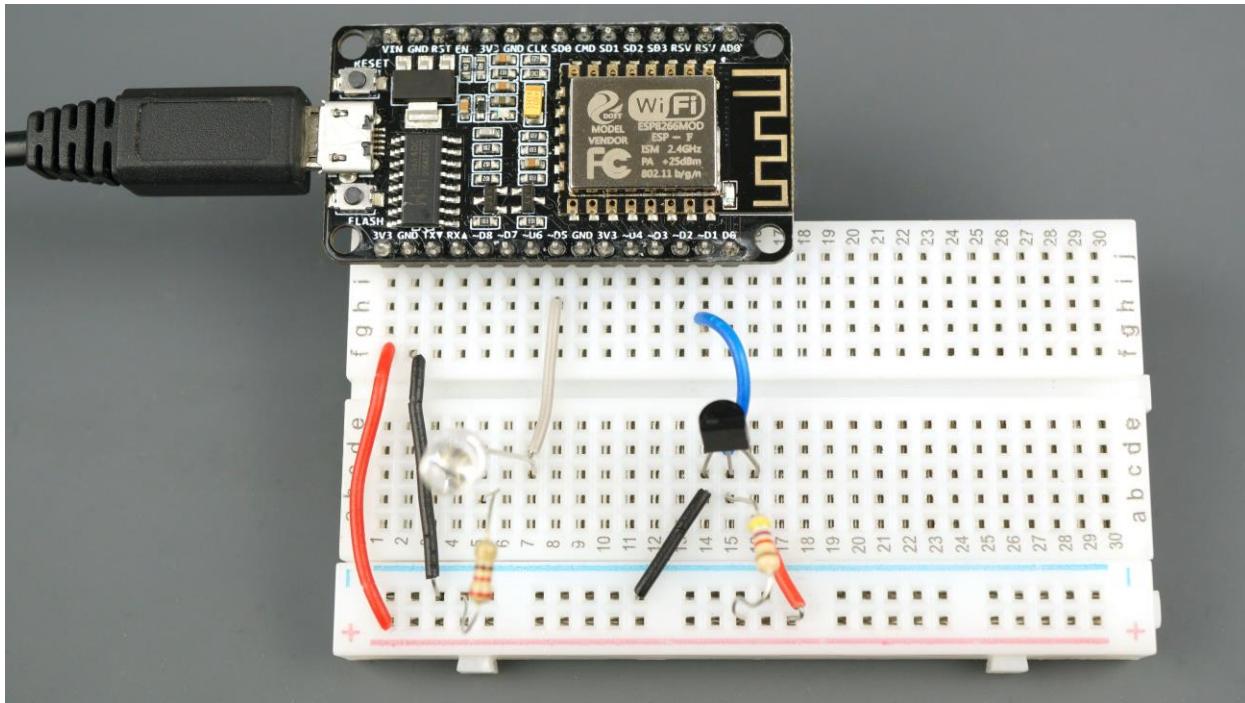
In our case, the Raspberry Pi IP address is **192.168.1.144**. Save your IP address. You'll need it in the next Units to connect the ESP8266 with the broker.

Wrapping Up

These were the steps to install the Mosquitto broker on a Raspberry Pi. In the next Units, we'll set up two ESP8266 boards as MQTT clients and you'll see how everything ties together with practical examples.

Unit 3: MQTT Project – MQTT Client

ESP8266 #1



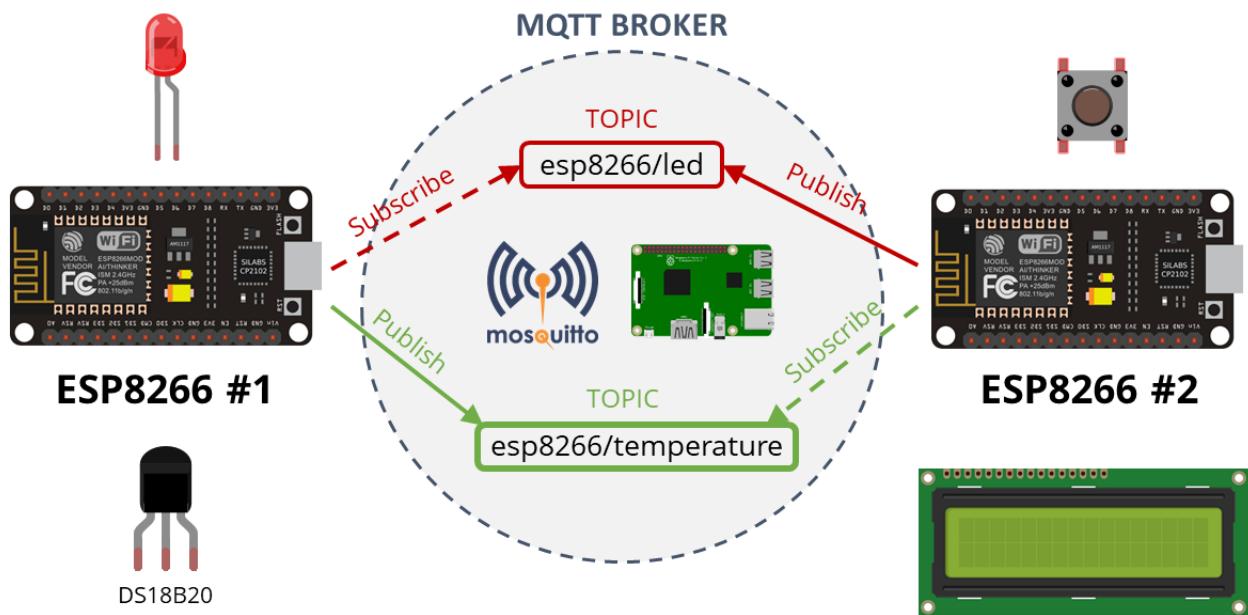
In this Unit we're going to demonstrate how you can use MQTT to exchange data between two ESP8266 boards. We're going to build a simple project to illustrate the most important concepts.

Here's a quick overview of the project.



- There are two ESP8266 boards – ESP8266 #1 and ESP8266 #2;
- ESP8266 #1 is connected to an LED and takes temperature readings with the DS18B20 sensor;
- ESP8266 #2 is attached to a pushbutton that when pressed toggles the LED of the ESP8266 #1;
- ESP8266 #2 is connected to an I2C LCD to display temperature readings received from ESP8266 #1.

Here's the MQTT diagram of this setup.



- ESP8266 #1 is subscribed to the topic **esp8266/led** and publishes temperature readings on topic **esp8266/temperature**;
- When you press the ESP8266 #2 pushbutton, it publishes a message on the topic **esp8266/led** to control the LED attached to ESP8266 #1;
- ESP8266 #2 is subscribed to **esp8266/temperature** topic to receive temperature readings and displays them on the LCD;
- We're using the Mosquitto broker installed on a Raspberry Pi to distribute the messages between the MQTT clients.

In this Unit, we're going to prepare ESP8266 #1 (in the next Unit, it's the ESP8266 #2).

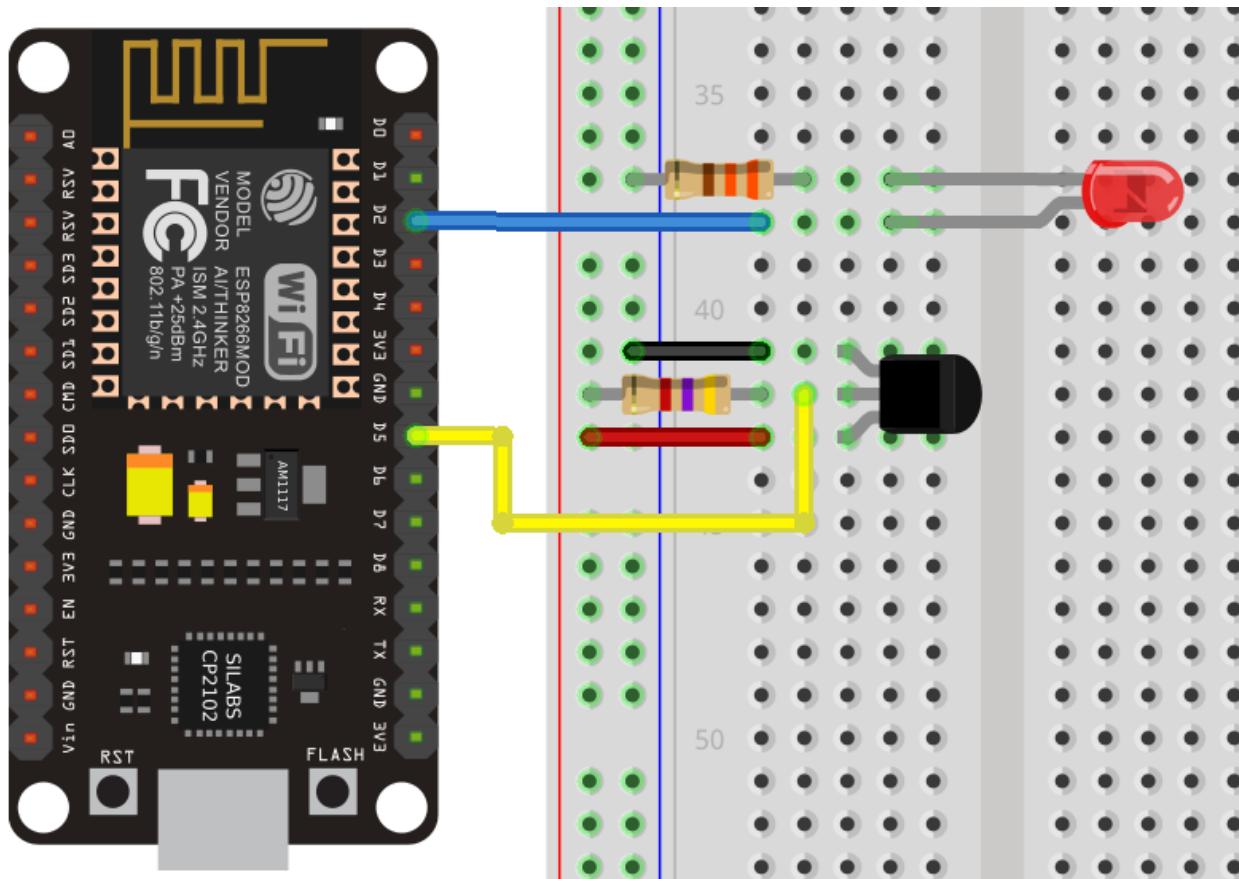
Parts Required

To complete this Unit, you need the following parts:

- [ESP8266](#)
- [5mm LED](#)
- [330 Ohm resistor](#)
- [DS18B20 temperature sensor](#)
- [4.7k Ohm resistor](#)
- [Jumper wires](#)
- [Breadboard](#)

Schematic

Start by assembling the circuit by following the next schematic diagram:



- Connect an LED with a 330 ohm resistor to GPIO 4 (D2);
- Wire the DS18B20 sensor with a 4.7K pull up resistor to GPIO 14 (D5).

Installing Libraries

There are several libraries to use MQTT with the ESP8266. We'll use the [Async MQTT Client library](#). That library also requires the [ESPASyncTCP library](#).

These libraries aren't available to install through the Arduino IDE library manager, so you need to include them in your Arduino IDE installation folder.

Installing the ESPASync TCP Library

1. [Click here to download the ESPASync TCP client library](#). You should have a *.zip* folder in your *Downloads* folder
2. Unzip the *.zip* folder and you should get *AsyncTCP-master* folder
3. Rename your folder from *AsyncTCP-master* to *AsyncTCP*
4. Move the *AsyncTCP* folder to your Arduino IDE installation libraries folder

Alternatively, in your Arduino IDE, you can go to **Sketch** ▶ **Include Library** ▶ **Add .ZIP Library** and select the library you've just downloaded.

Installing the Async MQTT Client Library

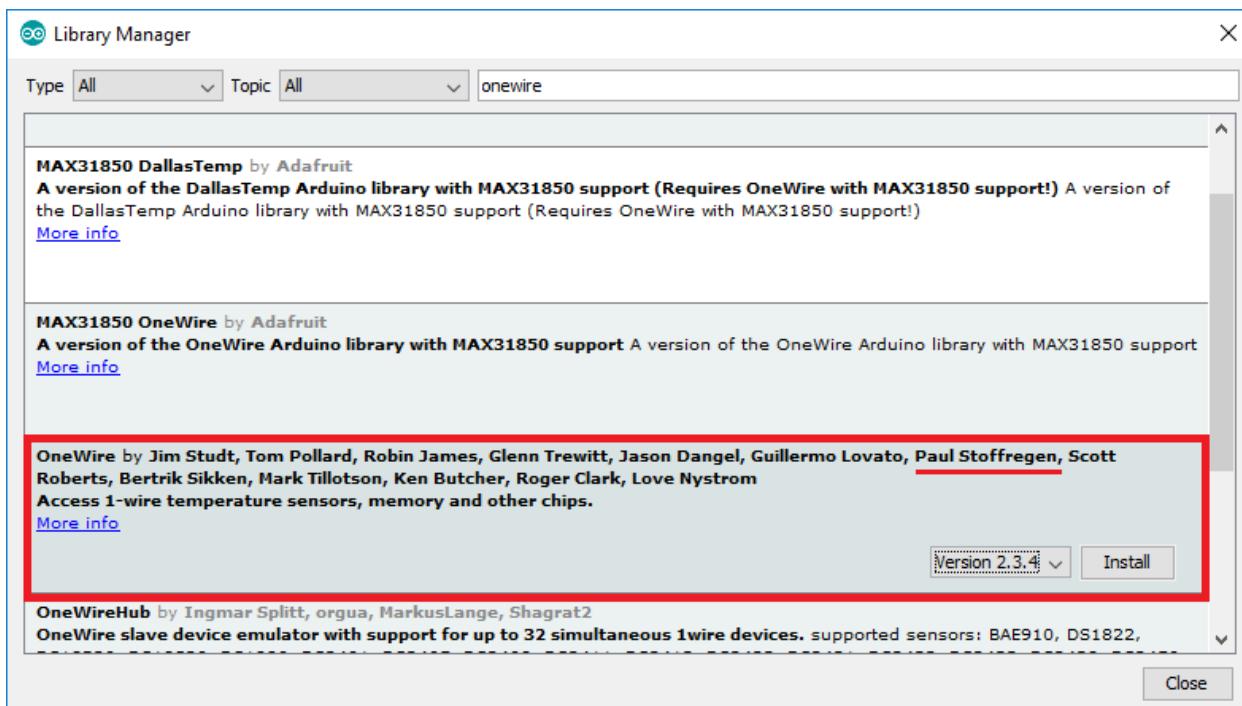
1. [Click here to download the Async MQTT client library](#). You should have a *.zip* folder in your *Downloads* folder
2. Unzip the *.zip* folder and you should get *async-mqtt-client-master* folder
3. Rename your folder from *async-mqtt-client-master* to *async_mqtt_client*
4. Move the *async_mqtt_client* folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Alternatively, in your Arduino IDE, you can go to **Sketch** ▶ **Include Library** ▶ **Add .ZIP Library** and select the library you've just downloaded.

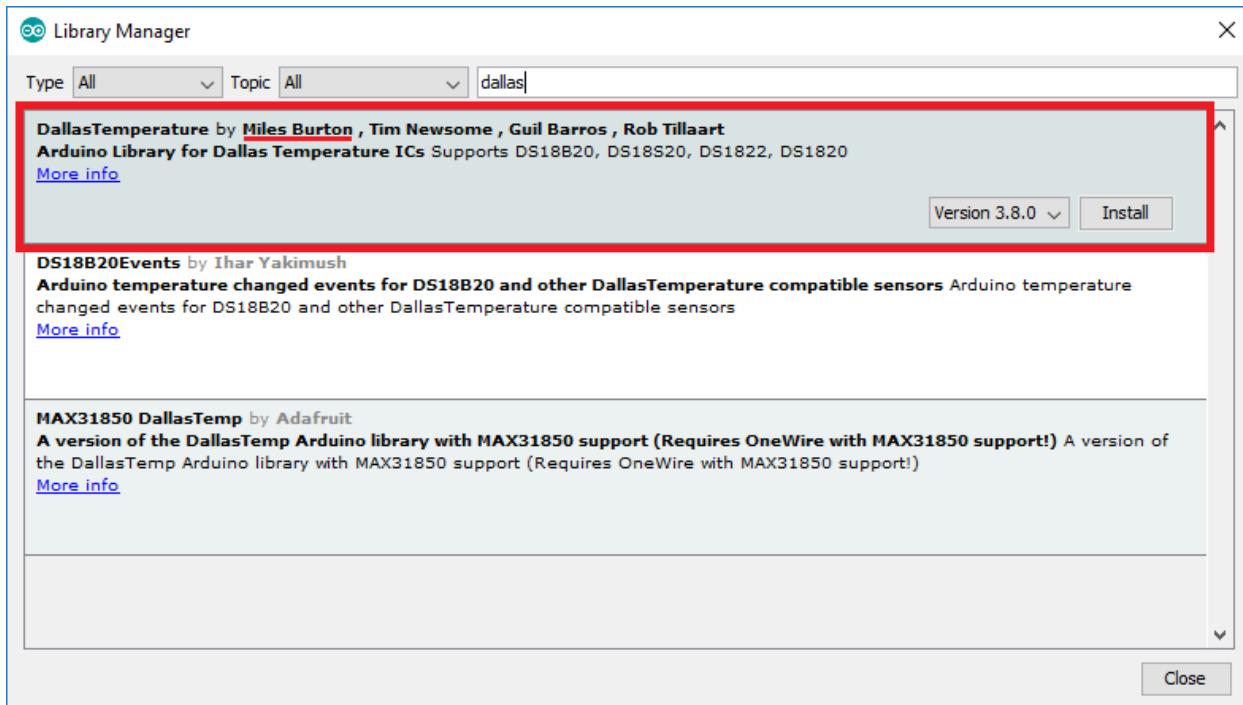
Installing the One Wire Library and Dallas Temperature Library

To interface with the DS18B20 temperature sensor, you need to install the [One Wire library by Paul Stoffregen](#) and the [Dallas Temperature library](#). Follow the next steps to install those libraries.

1. Open your Arduino IDE and go to **Sketch** > **Include Library** > **Manage Libraries**. The Library Manager should open;
2. Type “**onewire**” in the search box and install the OneWire library by Paul Stoffregen;



3. Then, search for “**Dallas**” and install the Dallas Temperature library by Miles Burton.



After installing the libraries, restart your Arduino IDE.

Code

With the libraries installed, open your Arduino IDE and copy the code provided.

```
#include <ESP8266WiFi.h>
#include <Ticker.h>
#include <AsyncMqttClient.h>

#include <OneWire.h>
#include <DallasTemperature.h>

// Change the credentials below, so your ESP8266 connects to your router
#define WIFI_SSID " REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD " REPLACE_WITH_YOUR_PASSWORD"

// Change the MQTT_HOST variable to your Raspberry Pi IP address,
// so it connects to your Mosquitto MQTT broker
#define MQTT_HOST IPAddress(192, 168, 1, XXX)
#define MQTT_PORT 1883

// Create objects to handle MQTT client
AsyncMqttClient mqttClient;
Ticker mqttReconnectTimer;
Ticker wifiReconnectTimer;
WiFiEventHandler wifiConnectHandler;
WiFiEventHandler wifiDisconnectHandler;
```

```

String temperatureString = "";
unsigned long previousMillis = 0;
const long interval = 5000;

const int ledPin = 14;
int ledState = LOW;

// GPIO where the DS18B20 is connected to
const int oneWireBus = 4;
// Setup a OneWire instance to communicate with any OneWire devices
OneWire oneWire(oneWireBus);
// Pass our oneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);

void connectToWifi() {
    Serial.println("Connecting to Wi-Fi...");
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}

void onWifiConnect(const WiFiEventStationModeGotIP& event) {
    Serial.println("Connected to Wi-Fi.");
    connectToMqtt();
}

void onWifiDisconnect(const WiFiEventStationModeDisconnected& event) {
    Serial.println("Disconnected from Wi-Fi.");
    mqttReconnectTimer.detach(); // ensure we don't reconnect to MQTT
while reconnecting to Wi-Fi
    wifiReconnectTimer.once(2, connectToWifi);
}

void connectToMqtt() {
    Serial.println("Connecting to MQTT...");
    mqttClient.connect();
}

// Add more topics that want your ESP8266 to be subscribed to
void onMqttConnect(bool sessionPresent) {
    Serial.println("Connected to MQTT.");
    Serial.print("Session present: ");
    Serial.println(sessionPresent);
    // ESP8266 subscribed to esp8266/led topic
    uint16_t packetIdSub = mqttClient.subscribe("esp8266/led", 0);
    Serial.print("Subscribing at QoS 0, packetId: ");
    Serial.println(packetIdSub);
}

void onMqttDisconnect(AsyncMqttClientDisconnectReason reason) {
    Serial.println("Disconnected from MQTT.");

    if (WiFi.isConnected()) {
        mqttReconnectTimer.once(2, connectToMqtt);
    }
}

```

```

void onMqttSubscribe(uint16_t packetId, uint8_t qos) {
    Serial.println("Subscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
    Serial.print(" qos: ");
    Serial.println(qos);
}

void onMqttUnsubscribe(uint16_t packetId) {
    Serial.println("Unsubscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}

void onMqttPublish(uint16_t packetId) {
    Serial.println("Publish acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}

// You can modify this function to handle what happens when you receive
// a certain message in a specific topic
void onMqttMessage(char* topic, char* payload,
AsyncMqttClientMessageProperties properties, size_t len, size_t index,
size_t total) {
    String messageTemp;
    for (int i = 0; i < len; i++) {
        //Serial.print((char)payload[i]);
        messageTemp += (char)payload[i];
    }
    // Check if the MQTT message was received on topic esp8266/led
    if (strcmp(topic, "esp8266/led") == 0) {
        // If the LED is off turn it on (and vice-versa)
        if (ledState == LOW) {
            ledState = HIGH;
        } else {
            ledState = LOW;
        }
        // Set the LED with the ledState of the variable
        digitalWrite(ledPin, ledState);
    }

    Serial.println("Publish received.");
    Serial.print(" message: ");
    Serial.println(messageTemp);
    Serial.print(" topic: ");
    Serial.println(topic);
    Serial.print(" qos: ");
    Serial.println(properties.qos);
    Serial.print(" dup: ");
    Serial.println(properties.dup);
    Serial.print(" retain: ");
    Serial.println(properties.retain);
    Serial.print(" len: ");
    Serial.println(len);
}

```

```

    Serial.print(" index: ");
    Serial.println(index);
    Serial.print(" total: ");
    Serial.println(total);
}

void setup() {
    // Start the DS18B20 sensor
    sensors.begin();
    // Define LED as an OUTPUT and set it LOW
    pinMode(ledPin, OUTPUT);
    digitalWrite(ledPin, LOW);

    Serial.begin(115200);

    wifiConnectHandler = WiFi.onStationModeGotIP(onWifiConnect);
    wifiDisconnectHandler
    WiFi.onStationModeDisconnected(onWifiDisconnect);
    =
}

mqttClient.onConnect(onMqttConnect);
mqttClient.onDisconnect(onMqttDisconnect);
mqttClient.onSubscribe(onMqttSubscribe);
mqttClient.onUnsubscribe(onMqttUnsubscribe);
mqttClient.onMessage(onMqttMessage);
mqttClient.onPublish(onMqttPublish);
mqttClient.setServer(MQTT_HOST, MQTT_PORT);

connectToWifi();
}

void loop() {
    unsigned long currentMillis = millis();
    // Every X number of seconds (interval = 5 seconds)
    // it publishes a new MQTT message on topic esp8266/temperature
    if (currentMillis - previousMillis >= interval) {
        // Save the last time a new reading was published
        previousMillis = currentMillis;
        // New temperature readings
        sensors.requestTemperatures();
        temperatureString = " " + String(sensors.getTempCByIndex(0)) + "C "
+
            String(sensors.getTempFByIndex(0)) + "F";
        Serial.println(temperatureString);
        // Publish an MQTT message on topic esp8266/temperature with Celsius
        // and Fahrenheit temperature readings
        uint16_t packetIdPub2 = mqttClient.publish("esp8266/temperature",
2, true, temperatureString.c_str());
        Serial.print("Publishing on topic esp8266/temperature at QoS 2,
packetId: ");
        Serial.println(packetIdPub2);
    }
}

```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module6/Unit3/MQTT_Client_1/MQTT_Client_1.ino

If you want to make the code work straight away, just type your Wi-Fi SSID and password in the following variables.

```
#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"  
#define WIFI_PASSWORD "REPLACE_WITH_YOUR_PASSWORD"
```

You also need to enter the Mosquitto MQTT broker IP address. Go to the previous Unit to find your Raspberry Pi IP address.

```
#define MQTT_HOST IPAddress(192, 168, 1, 74)
```

You can upload the code as it is and it will work. But we recommend continuing reading this Unit to learn how it actually works.

How the Code Works

The following section imports all the required libraries.

```
#include <ESP8266WiFi.h>  
#include <Ticker.h>  
#include <AsyncMqttClient.h>  
#include <OneWire.h>  
#include <DallasTemperature.h>
```

As said before, you need to include your SSID, password, and Raspberry Pi IP address.

```
// Change the credentials below, so your ESP8266 connects to your router  
#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"  
#define WIFI_PASSWORD "REPLACE_WITH_YOUR_PASSWORD"  
  
// Change the MQTT_HOST variable to your Raspberry Pi IP address,  
// so it connects to your Mosquitto MQTT broker  
#define MQTT_HOST IPAddress(192, 168, 1, 74)
```

Create an object to handle the MQTT client and timers to reconnect to your MQTT broker and router when it disconnects.

```
// Create objects to handle MQTT client
AsyncMqttClient mqttClient;
Ticker mqttReconnectTimer;
Ticker wifiReconnectTimer;
WiFiEventHandler wifiConnectHandler;
WiFiEventHandler wifiDisconnectHandler;
```

After that, declare some variables to hold the temperature and create auxiliary timer variables to publish readings every 5 seconds.

```
String temperatureString = "";
unsigned long previousMillis = 0;
const long interval = 5000;
```

Then, define the LED pin and store its initial state.

```
const int ledPin = 14;           // GPIO where the LED is connected to
int ledState = LOW;           // the current state of the output pin
```

Finally, define the DS18B20 sensor pin and create objects to make it work.

```
// GPIO where the DS18B20 is connected to
const int oneWireBus = 4;
// Setup a OneWire instance to communicate with any OneWire devices
OneWire oneWire(oneWireBus);
// Pass our oneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);
```

MQTT functions: connect to Wi-Fi, connect to MQTT, Wi-Fi events

We haven't added any comments to the functions defined in the next code section. Those functions come with the AsyncMqttClient library. The function's names are pretty self-explanatory.

For example, the `connectToWifi()` connects your ESP8266 to your router:

```
void connectToWifi() {
  Serial.println("Connecting to Wi-Fi...");
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}
```

The `connectToMqtt()` connects your ESP8266 to your MQTT broker:

```
void connectToMqtt() {
    Serial.println("Connecting to MQTT...");
    mqttClient.connect();
}
```

Subscribe to MQTT topic

The `onMqttConnect()` function is responsible to subscribe your ESP8266 to topics. You can modify this function and add more topics that you want your ESP8266 to be subscribed to. In this case, the ESP8266 is only subscribed to **esp8266/led** topic.

```
void onMqttConnect(bool sessionPresent) {
    Serial.println("Connected to MQTT.");
    Serial.print("Session present: ");
    Serial.println(sessionPresent);
    // ESP8266 subscribed to esp8266/led topic
    uint16_t packetIdSub = mqttClient.subscribe("esp8266/led", 0);
    Serial.print("Subscribing at QoS 0, packetId: ");
    Serial.println(packetIdSub);
}
```

The important part in this snippet is the following line that subscribes to an MQTT topic using the `.subscribe()` method:

```
uint16_t packetIdSub = mqttClient.subscribe("esp8266/led", 0);
```

This method accepts as arguments the MQTT topic you want the ESP8266 to be subscribed to, and the Quality of Service (QoS). You can [read this article](#) for more information about MQTT QoS.

MQTT functions: disconnect, subscribe, unsubscribe, and publish

If the ESP8266 loses connection with the MQTT broker, it prints that message in the serial monitor and tries to reconnect.

```
void onMqttDisconnect(AsyncMqttClientDisconnectReason reason) {
    Serial.println("Disconnected from MQTT.");
    if (WiFi.isConnected()) {
```

```
    mqttReconnectTimer.once(2, connectToMqtt);
}
}
```

When the ESP8266 subscribes to an MQTT topic, it prints the packet id and quality of service (QoS).

```
void onMqttSubscribe(uint16_t packetId, uint8_t qos) {
    Serial.println("Subscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
    Serial.print(" qos: ");
    Serial.println(qos);
}
```

If you unsubscribe from a topic, it also prints a message with some information about that.

```
void onMqttUnsubscribe(uint16_t packetId) {
    Serial.println("Unsubscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}
```

And when you publish a message to an MQTT topic, it prints the packet id in the Serial Monitor.

```
void onMqttPublish(uint16_t packetId) {
    Serial.println("Publish acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}
```

Receiving MQTT messages

When a message is received on a topic that the ESP8266 is subscribed to, in this case the **esp8266/led** topic, it executes the `onMqttMessage()` function. In this function you should add what happens when a message is received on a specific topic.

```
void onMqttMessage(char* topic, char* payload, AsyncMqttClientMessageProperties properties, size_t len, size_t index, size_t total) {
    String messageTemp;
    for (int i = 0; i < len; i++) {
        //Serial.print((char)payload[i]);
        messageTemp += (char)payload[i];
    }
}
```

```
}

// Check if the MQTT message was received on topic esp8266/led
if (strcmp(topic, "esp8266/led") == 0) {
    // If the LED is off turn it on (and vice-versa)
    if (ledState == LOW) {
        ledState = HIGH;
    } else {
        ledState = LOW;
    }
    // Set the LED with the ledState of the variable
    digitalWrite(ledPin, ledState);
}
```

In this previous snippet, the following if statement, checks if a new message was published on the **esp8266/led** topic.

```
if (strcmp(topic, "esp8266/led") == 0) {
```

Then, you can add your logic inside that if statement to make the ESP8266 do something. In this case, we're toggling the LED every time we receive a message on that topic.

```
if (ledState == LOW) {
    ledState = HIGH;
} else {
    ledState = LOW;
}
// Set the LED with the ledState of the variable
digitalWrite(ledPin, ledState);
```

Basically, all these functions that we've just mentioned are callback functions. So, they are executed asynchronously.

setup()

Now, let's proceed to the `setup()`. Start the DS18B20 sensor, define the LED as an OUTPUT, set it to LOW and start the serial communication.

```
// Start the DS18B20 sensor
sensors.begin();
// Define LED as an OUTPUT and set it LOW
pinMode(ledPin, OUTPUT);
digitalWrite(ledPin, LOW);
Serial.begin(115200);
```

Assign all the callbacks functions. These are all callback functions, which means that they will be executed automatically when needed. For example, when the ESP8266 connects to the broker, it automatically calls the `onMqttConnect()` function, and so on.

```
wifiConnectHandler = WiFi.onStationModeGotIP(onWifiConnect);
wifiDisconnectHandler = WiFi.onStationModeDisconnected(onWifiDisconnect);
mqttClient.onConnect(onMqttConnect);
mqttClient.onDisconnect(onMqttDisconnect);
mqttClient.onSubscribe(onMqttSubscribe);
mqttClient.onUnsubscribe(onMqttUnsubscribe);
mqttClient.onMessage(onMqttMessage);
mqttClient.onPublish(onMqttPublish);
mqttClient.setServer(MQTT_HOST, MQTT_PORT);

connectToWifi();
```

Finally, connect to Wi-Fi:

```
connectToWifi();
```

loop()

In the `loop()`, create a timer that will allow you to publish new temperature readings in the **esp8266/temperature** topic every 5 seconds.

```
void loop() {
    unsigned long currentMillis = millis();
    // Every X number of seconds (interval = 5 seconds)
    // it publishes a new MQTT message on topic esp8266/temperature
    if (currentMillis - previousMillis >= interval) {
        // Save the last time a new reading was published
        previousMillis = currentMillis;
        // New temperature readings
        sensors.requestTemperatures();
        temperatureString = " " + String(sensors.getTempCByIndex(0)) + "C " +
            String(sensors.getTempFByIndex(0)) + "F";
        Serial.println(temperatureString);
        // Publish an MQTT message on topic esp8266/temperature with Celsius and
        // Fahrenheit temperature readings
        uint16_t packetIdPub2 = mqttClient.publish("esp8266/temperature", 2, true,
        temperatureString.c_str());
        Serial.print("Publishing on topic esp8266/temperature at QoS 2, packetId: ");
        Serial.println(packetIdPub2);
    }
}
```

Publishing/Subscribing to more topics

This is a basic example, but illustrates how MQTT works with publish and subscribe.

If you would like to publish more readings on different topics, you can duplicate these next three lines in the `loop()`. Basically, use the `.publish()` method to publish data on a topic.

```
uint16_t packetIdPub2 = mqttClient.publish("esp8266/temperature", 2,  
true, temperatureString.c_str());  
Serial.print("Publishing on topic esp8266/temperature at QoS 2, packetId: ");  
Serial.println(packetIdPub2);
```

On the other hand, if you want to subscribe to more topics, go to the `onMqttConnect()` function, duplicate the following line and replace the topic with another topic that you want to be subscribed to.

```
uint16_t packetIdSub = mqttClient.subscribe("esp8266/led", 0);
```

Finally, in the `onMqttMessage()` function, you can add logic to determine what happens when you receive a message in a specific topic.

Uploading the Code

With your Raspberry Pi powered on and running the Mosquitto MQTT broker, upload the code to your ESP8266.

Note: complete the previous Unit if you haven't prepared the Mosquitto MQTT broker yet.

Open the serial monitor at the 115200 baud rate and check if your ESP8266 is being successfully connected to your router and MQTT broker.

```
Connecting to Wi-Fi...
[WiFi-event] event: 2
[WiFi-event] event: 0
[WiFi-event] event: 4
[WiFi-event] event: 7
WiFi connected
IP address:
192.168.1.147
Connecting to MQTT...
Connected to MQTT.
```

Autoscroll No line ending 115200 baud Clear output

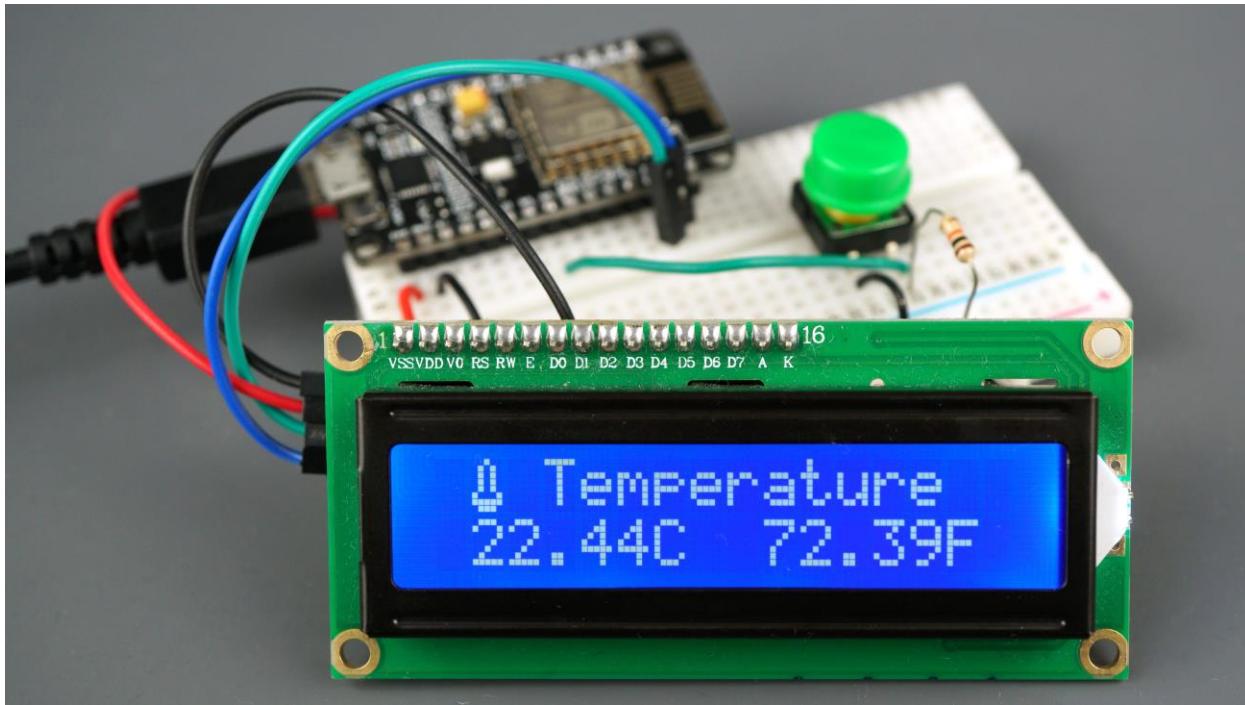
As you can see, it's working as expected.

Continue To The Next Unit

Go to the next Unit to prepare ESP8266 #2 and continue this project.

Unit 4: MQTT Project – MQTT Client

ESP8266 #2



This is part 2 of the ESP8266 MQTT project example. If you've followed the previous Unit, your ESP8266 #1 is ready. In this Unit, you're going to prepare ESP8266 #2. To recap:

- ESP8266 #2 receives the temperature readings from ESP8266 #1 and displays them on an LCD;
- ESP8266 #2 has a pushbutton that controls the LED of the ESP8266 #1.

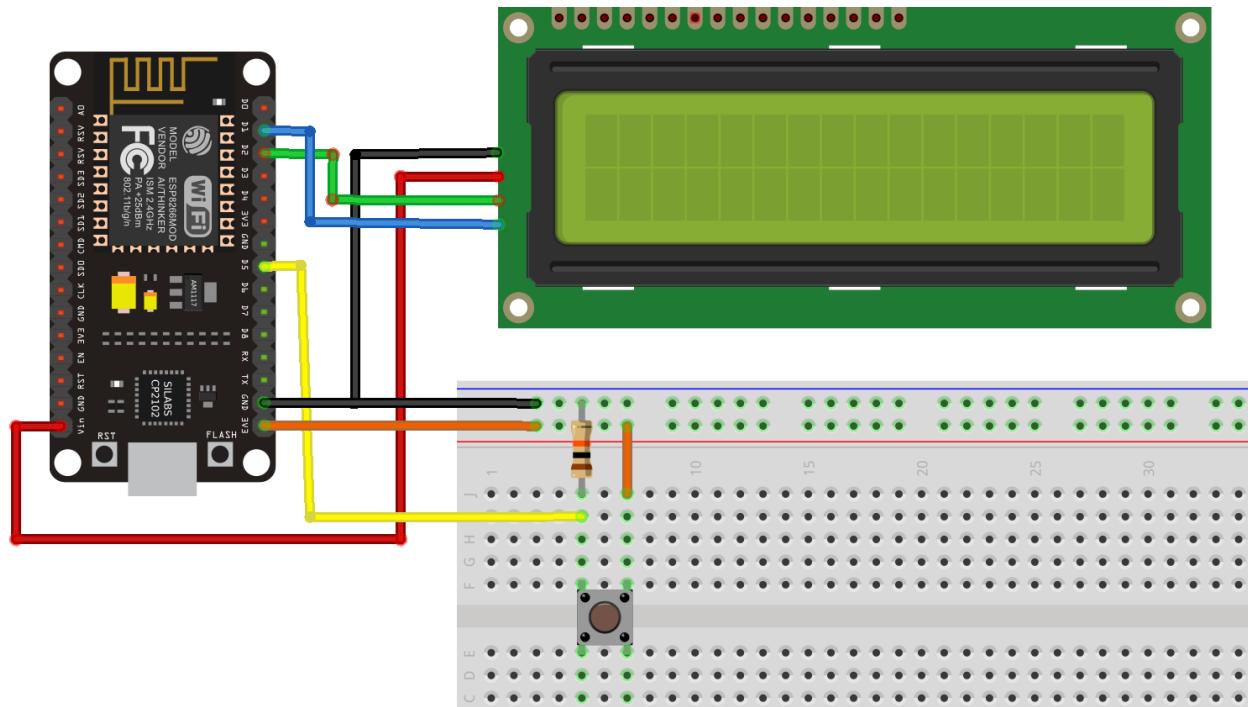
Parts Required

For this Unit, you need the following parts:

- [ESP8266](#)
- [Pushbutton](#)
- [10k Ohm resistor](#)
- [16x2 I2C LCD](#)
- [Jumper wires](#)
- [Breadboard](#)

Schematic

Start by assembling the circuit:



- Wire a pushbutton to the ESP8266: one lead to 3.3V and the other lead connected to GPIO 14 (D5) using a 10k Ohm pull down resistor.
- Wire the I2C LCD: connect SDA to GPIO 4 (D2) and SCL to GPIO 5 (D1). The LCD operates at 5V, so connect it to Vin and GND.

Installing Libraries

After having your circuit ready, it's time to install the necessary libraries in your Arduino IDE.

If you've followed the previous Unit, you already have the [ESPAsyncTCP library](#) and [Async MQTT Client library](#) installed. So, you only need to install the [Liquid Crystal I2C library](#). Follow the next steps to install that library.

Installing the Liquid Crystal I2C Library

1. [Click here to download the LiquidCrystal_I2C library](#). You should have a .zip folder in your *Downloads*
2. Unzip the .zip folder and you should get *LiquidCrystal_I2C-master* folder
3. Rename your folder from *LiquidCrystal_I2C-master* to *LiquidCrystal_I2C*
4. Move the *LiquidCrystal_I2C* folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Alternatively, in your Arduino IDE, you can go to **Sketch** ▶ **Include Library** ▶ **Add .ZIP Library** and select the library you've just downloaded.

Code

After having both libraries installed, open your Arduino IDE and copy the code provided.

```
#include <ESP8266WiFi.h>
#include <Ticker.h>
#include <AsyncMqttClient.h>
#include <LiquidCrystal_I2C.h>

// Change the credentials below, so your ESP8266 connects to your router
#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD "REPLACE_WITH_YOUR_PASSWORD"
```

```

// Change the MQTT_HOST variable to your Raspberry Pi IP address,
// so it connects to your Mosquitto MQTT broker
#define MQTT_HOST IPAddress(192, 168, 1, XX)
#define MQTT_PORT 1883

// Create objects to handle MQTT client
AsyncMqttClient mqttClient;
Ticker mqttReconnectTimer;
Ticker wifiReconnectTimer;
WiFiEventHandler wifiConnectHandler;
WiFiEventHandler wifiDisconnectHandler;

// Set the LCD number of columns and rows
const int lcdColumns = 16;
const int lcdRows = 2;

// Set LCD address, number of columns and rows
// if you don't know your display address, run an I2C scanner sketch
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);

// Thermometer icon
byte thermometerIcon[8] = {
    B00100,
    B01010,
    B01010,
    B01010,
    B01010,
    B10001,
    B11111,
    B01110
};

// Define GPIO where the pushbutton is connected to
const int buttonPin = 14;
int buttonState;
int lastButtonState = LOW;
unsigned long lastDebounceTime = 0;
unsigned long debounceDelay = 50;

void connectToWifi() {
    Serial.println("Connecting to Wi-Fi...");
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}

void onWifiConnect(const WiFiEventStationModeGotIP& event) {
    Serial.println("Connected to Wi-Fi.");
    connectToMqtt();
}

void onWifiDisconnect(const WiFiEventStationModeDisconnected& event) {
    Serial.println("Disconnected from Wi-Fi.");
    mqttReconnectTimer.detach(); // ensure we don't reconnect to MQTT while
reconnecting to Wi-Fi
    wifiReconnectTimer.once(2, connectToWifi);
}

void connectToMqtt() {
    Serial.println("Connecting to MQTT...");
    mqttClient.connect();
}

```

```

// Add more topics that want your ESP8266 to be subscribed to
void onMqttConnect(bool sessionPresent) {
    Serial.println("Connected to MQTT.");
    Serial.print("Session present: ");
    Serial.println(sessionPresent);
    uint16_t packetIdSub = mqttClient.subscribe("esp8266/temperature", 0);
    Serial.print("Subscribing at QoS 0, packetId: ");
    Serial.println(packetIdSub);
}

void onMqttDisconnect(AsyncMqttClientDisconnectReason reason) {
    Serial.println("Disconnected from MQTT.");

    if (WiFi.isConnected()) {
        mqttReconnectTimer.once(2, connectToMqtt);
    }
}

void onMqttSubscribe(uint16_t packetId, uint8_t qos) {
    Serial.println("Subscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
    Serial.print(" qos: ");
    Serial.println(qos);
}

void onMqttUnsubscribe(uint16_t packetId) {
    Serial.println("Unsubscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}

void onMqttPublish(uint16_t packetId) {
    Serial.println("Publish acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}

// You can modify this function to handle what happens when you receive a
// certain message in a specific topic
void onMqttMessage(char* topic, char* payload, AsyncMqttClientMessageProperties properties, size_t len, size_t index, size_t total) {
    String messageTemp;
    for (int i = 0; i < len; i++) {
        //Serial.print((char)payload[i]);
        messageTemp += (char)payload[i];
    }
    if (strcmp(topic, "esp8266/temperature") == 0) {
        lcd.clear();
        lcd.setCursor(1, 0);
        lcd.write(0);
        lcd.print(" Temperature");
        lcd.setCursor(0, 1);
        lcd.print(messageTemp);
    }

    Serial.println("Publish received.");
    Serial.print(" message: ");
    Serial.println(messageTemp);
}

```

```

    Serial.print(" topic: ");
    Serial.println(topic);
    Serial.print(" qos: ");
    Serial.println(properties.qos);
    Serial.print(" dup: ");
    Serial.println(properties.dup);
    Serial.print(" retain: ");
    Serial.println(properties.retain);
    Serial.print(" len: ");
    Serial.println(len);
    Serial.print(" index: ");
    Serial.println(index);
    Serial.print(" total: ");
    Serial.println(total);
}

void setup() {
    Serial.begin(115200);
    Serial.print(" 1");

    // Initialize LCD
    lcd.init();
    // Turn on LCD backlight
    lcd.backlight();
    // Create thermometer icon
    lcd.createChar(0, thermometerIcon);

    // Define buttonPin as an INPUT
    pinMode(buttonPin, INPUT);
    Serial.print("2");

    wifiConnectHandler = WiFi.onStationModeGotIP(onWifiConnect);
    wifiDisconnectHandler = WiFi.onStationModeDisconnected(onWifiDisconnect);

    mqttClient.onConnect(onMqttConnect);
    mqttClient.onDisconnect(onMqttDisconnect);
    mqttClient.onSubscribe(onMqttSubscribe);
    mqttClient.onUnsubscribe(onMqttUnsubscribe);
    mqttClient.onMessage(onMqttMessage);
    mqttClient.onPublish(onMqttPublish);
    mqttClient.setServer(MQTT_HOST, MQTT_PORT);

    connectToWifi();
}

void loop() {
    // Read the state of the pushbutton and save it in a local variable
    int reading = digitalRead(buttonPin);

    // If the pushbutton state changed (due to noise or pressing it), reset the
    // timer
    if (reading != lastButtonState) {
        // Reset the debouncing timer
        lastDebounceTime = millis();
    }

    // If the button state has changed, after the debounce time
    if ((millis() - lastDebounceTime) > debounceDelay) {
        // And if the current reading is different than the current buttonState
}

```

```

    if (reading != buttonState) {
        buttonState = reading;
        // Publish an MQTT message on topic esp8266/led to toggle the LED (turn
        the LED on or off) P
        if (buttonState == HIGH) {
            mqttClient.publish("esp8266/led", 0, true, "toggle");
            Serial.println("Publishing on topic esp8266/led topic at QoS 0");
        }
    }
}
// Save the reading. Next time through the loop, it'll be the lastButtonState
lastButtonState = reading;
}

```

SOURCE CODE

[https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module6/Unit4/MQTT Client 2.ino](https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART1_Arduino/Module6/Unit4/MQTT%20Client%202.ino)

Similar to the previous example, if you type your SSID, password and MQTT broker IP address the code will work straight away.

```

// Change the credentials below, so your ESP8266 connects to your router
#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD "REPLACE_WITH_YOUR_PASSWORD"

// Change the MQTT_HOST variable to your Raspberry Pi IP address,
// so it connects to your Mosquitto MQTT broker
#define MQTT_HOST IPAddress(192, 168, 1, XX)

```

However, we recommend continuing reading this Unit to learn how it works.

How the Code Works

We'll skip most code sections, because they were already explained before.

Add your SSID, password and MQTT broker IP address.

```

// Change the credentials below, so your ESP8266 connects to your router
#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD "REPLACE_WITH_YOUR_PASSWORD"

// Change the MQTT_HOST variable to your Raspberry Pi IP address,
// so it connects to your Mosquitto MQTT broker
#define MQTT_HOST IPAddress(192, 168, 1, XX)

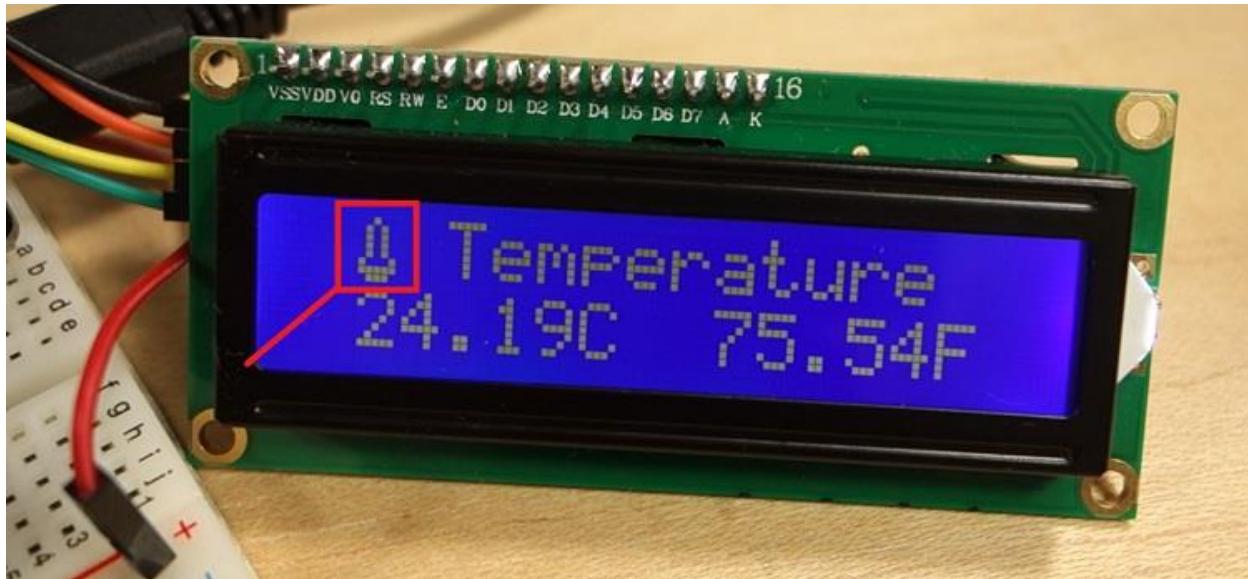
```

Define the LCD number of columns and rows. In this case, we're using a 16×2 character LCD.

```
const int lcdColumns = 16;  
const int lcdRows = 2;
```

The following byte array is used to display a thermometer icon on the LCD.

```
byte thermometerIcon[8] = {  
    B00100,  
    B01010,  
    B01010,  
    B01010,  
    B01010,  
    B10001,  
    B11111,  
    B01110  
};
```



Define the button Pin, a variable to store the current button state, last button state and declare auxiliary variables to create a debounce timer to avoid false pushbutton presses.

```
const int buttonPin = 14;  
int buttonState;  
int lastButtonState = LOW;  
unsigned long lastDebounceTime = 0;  
unsigned long debounceDelay = 50;
```

MQTT functions

The `connectToWiFi()`, `connectToMQTT()` and other functions were already explained in the previous Unit.

The `onMqttConnect()` function is where you add the topics you want your ESP8266 to be subscribed to.

```
void onMqttConnect(bool sessionPresent) {
    Serial.println("Connected to MQTT.");
    Serial.print("Session present: ");
    Serial.println(sessionPresent);
    uint16_t packetIdSub = mqttClient.subscribe("esp8266/temperature", 0);
    Serial.print("Subscribing at QoS 0, packetId: ");
    Serial.println(packetIdSub);
}
```

In this case, the ESP8266 is only subscribed to the **esp8266/temperature** topic, but you can change the `onMqttConnect()` function to subscribe to more topics.

Let's skip to the `onMqttMessage()` function.

```
void onMqttMessage(char* topic, char* payload, AsyncMqttClientMessageProperties properties, size_t len, size_t index, size_t total) {
    String messageTemp;
    for (int i = 0; i < len; i++) {
        //Serial.print((char)payload[i]);
        messageTemp += (char)payload[i];
    }
    if (strcmp(topic, "esp8266/temperature") == 0) {
        lcd.clear();
        lcd.setCursor(1, 0);
        lcd.write(0);
        lcd.print(" Temperature");
        lcd.setCursor(0, 1);
        lcd.print(messageTemp);
    }
}
```

As explored in the previous Unit, this is where you implement what happens when you receive a message in a subscribed topic. You can create more if statements to check other topics, or to check the message content.

In this case, when we receive a message in the **esp8266/temperature** topic we display the message in the LCD.

setup()

In the `setup()`, start the LCD, turn on the back light and create the thermometer icon, so we can display it on the LCD.

```
Serial.begin(115200);

// Initialize LCD
lcd.init();
// Turn on LCD backlight
lcd.backlight();
// Create thermometer icon
lcd.createChar(0, thermometerIcon);

// Define buttonPin as an INPUT
pinMode(buttonPin, INPUT);
```

And set all the callback functions for the Wi-Fi and MQTT events.

```
wifiConnectHandler = WiFi.onStationModeGotIP(onWifiConnect);
wifiDisconnectHandler = WiFi.onStationModeDisconnected(onWifiDisconnect);

mqttClient.onConnect(onMqttConnect);
mqttClient.onDisconnect(onMqttDisconnect);
mqttClient.onSubscribe(onMqttSubscribe);
mqttClient.onUnsubscribe(onMqttUnsubscribe);
mqttClient.onMessage(onMqttMessage);
mqttClient.onPublish(onMqttPublish);
mqttClient.setServer(MQTT_HOST, MQTT_PORT);
```

loop()

In the `loop()`, we publish an MQTT message on the **esp8266/led** topic when the pushbutton is pressed. We're using the button Debounce example code that ensures that you don't get false pushbutton presses.

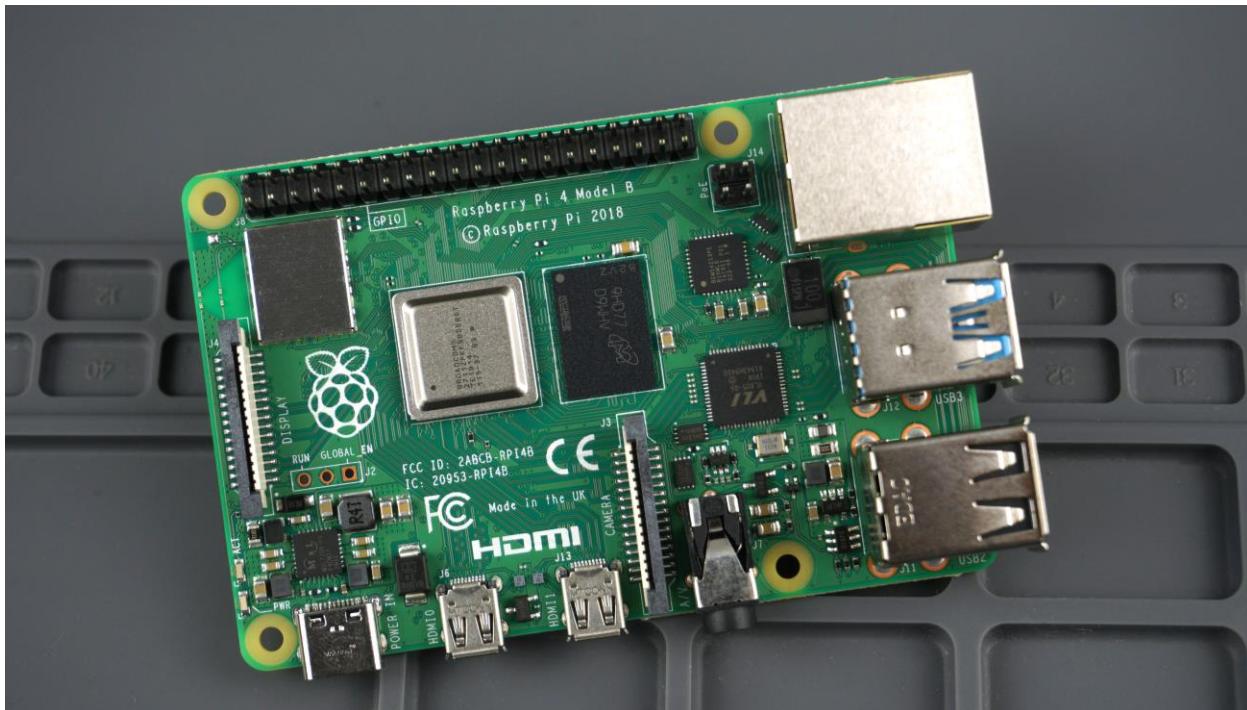
When the button is pressed the following if statement is true and the message "toggle" is published in the **esp8266/led** topic.

```
if (buttonState == HIGH) {
    mqttClient.publish("esp8266/led", 0, true, "toggle");
    Serial.println("Publishing on topic esp8266/led topic at QoS 0");
}
```

That's it for the code explanation. You can upload the code to your ESP8266.

Demonstration

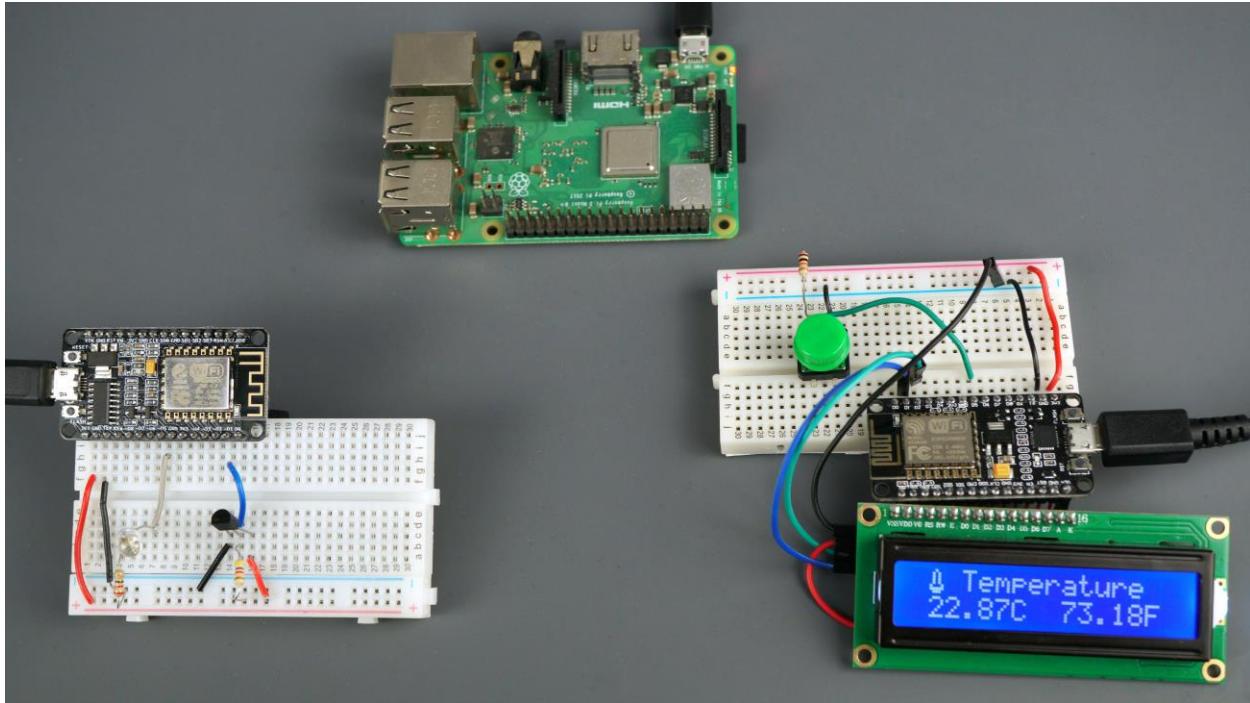
With your Raspberry Pi powered on and running the Mosquitto MQTT broker, open the serial monitor at the 115200 baud rate.



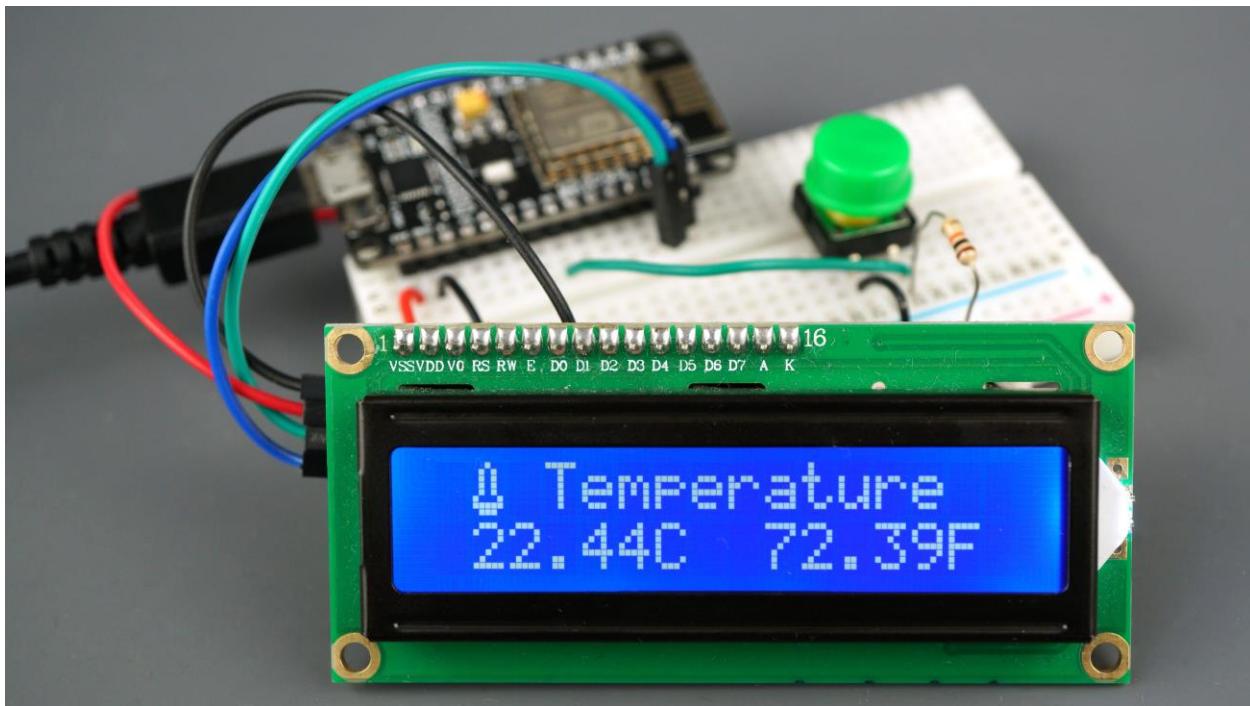
Check if your ESP8266 is being successfully connected to your router and MQTT broker. As you can see in the following screenshot, it's working properly:

```
Connecting to Wi-Fi...
[WiFi-event] event: 2
[WiFi-event] event: 0
[WiFi-event] event: 4
[WiFi-event] event: 7
WiFi connected
IP address:
192.168.1.148
Connecting to MQTT...
Connected to MQTT.
```

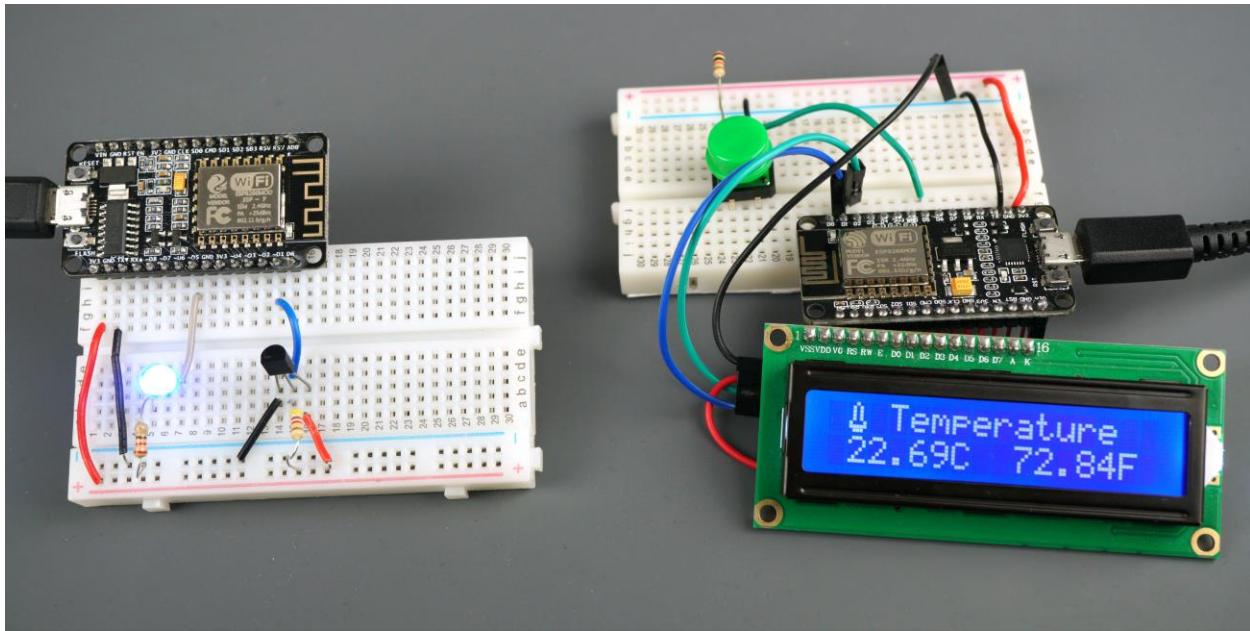
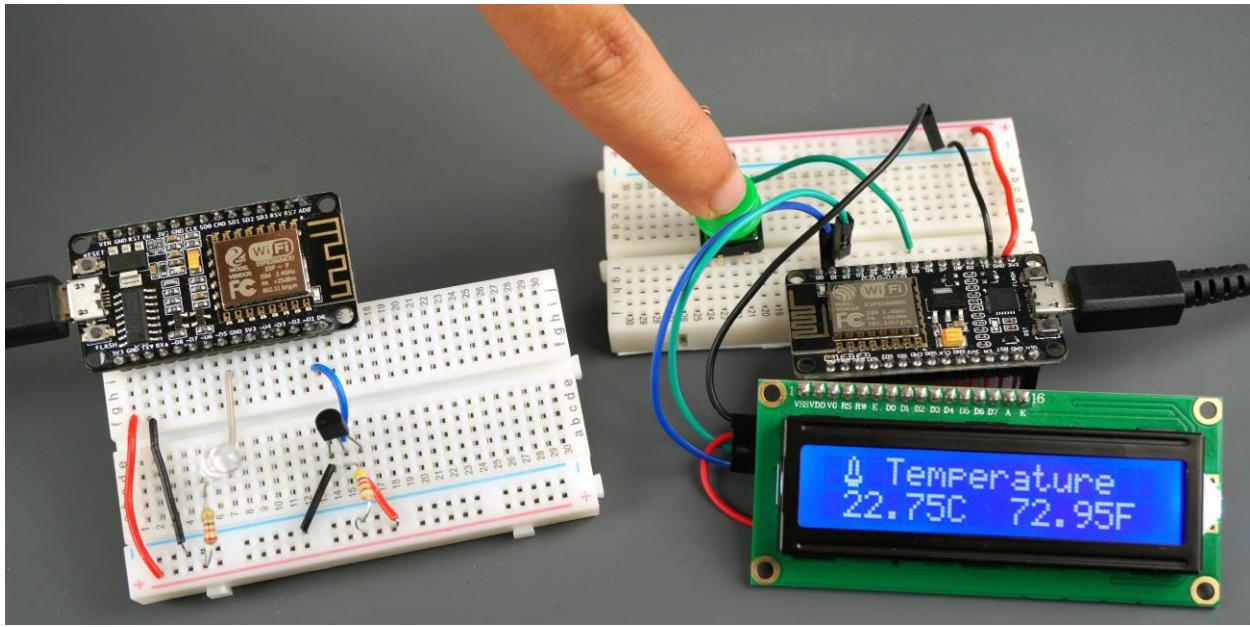
Now, power both ESP8266 boards and leave the mosquitto MQTT broker running.



The ESP8266 #2 receives the temperature readings from ESP8266 #1 in both Celsius and Fahrenheit degrees.



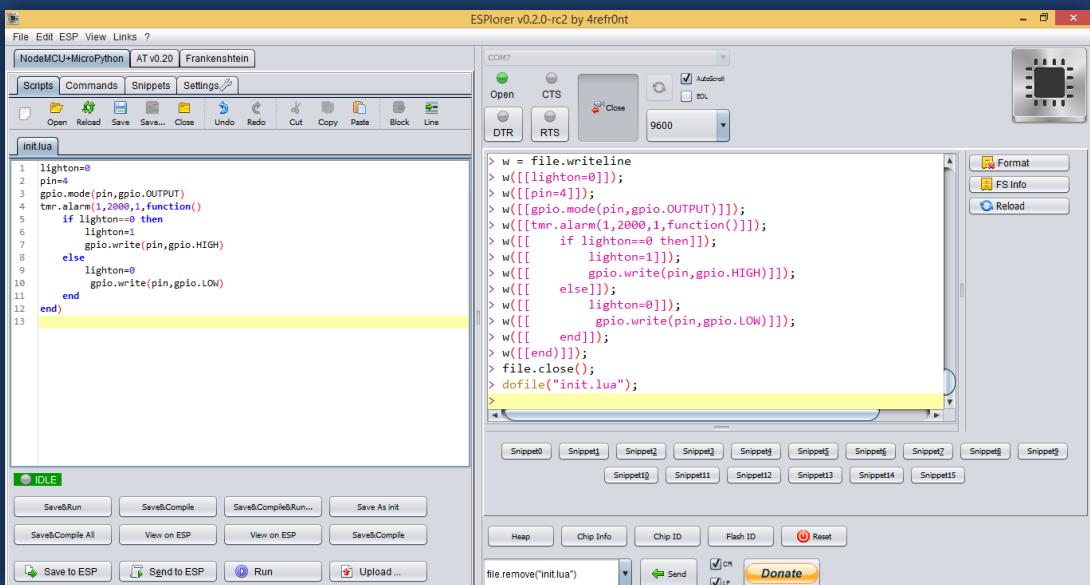
If you press the pushbutton, it instantly toggles the LED attached to the ESP8266 #1.



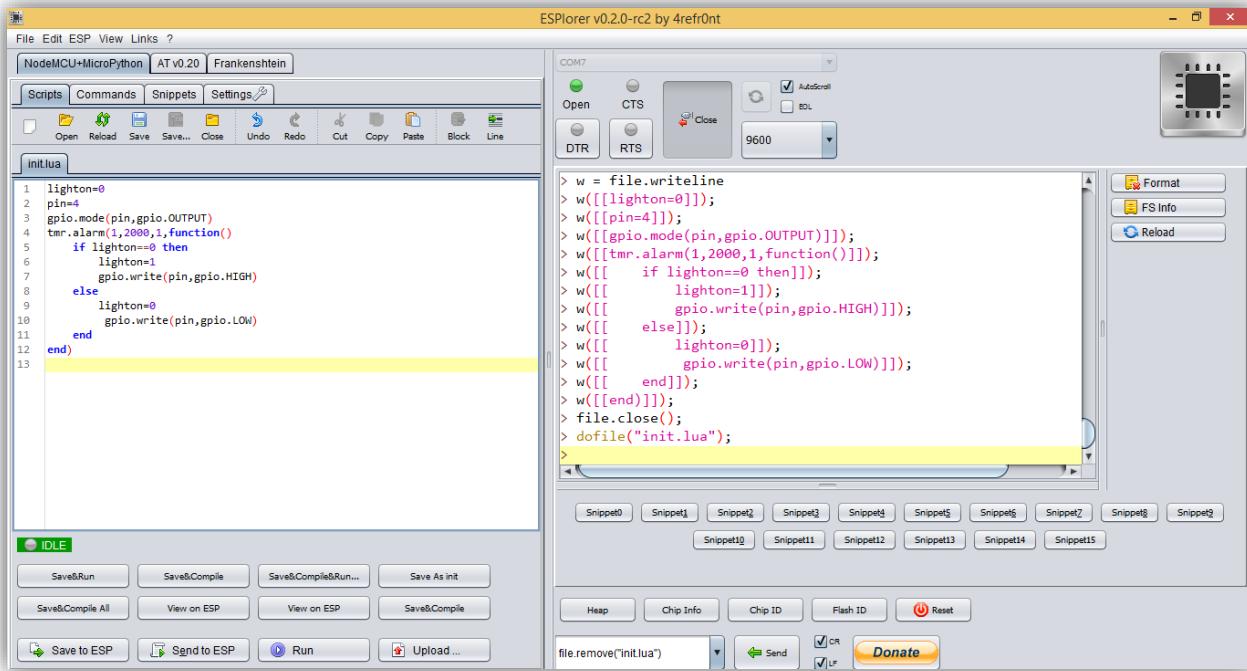
This system can easily be expanded to add more ESP8266 boards to your network. You can create more MQTT topics to send commands and receive other sensor readings. To make this process more practical, you can also use a home automation platform to add a dashboard to your system, like Node-RED, Home Assistant, Domoticz, or OpenHAB, for example (this is not covered in this course).

PART 2

ESP8266 with NodeMCU Firmware



Unit 1: ESP8266 with NodeMCU Firmware



In this Unit, you're going to download, install and prepare your ESP8266 to work with NodeMCU firmware.

Why Flashing Your ESP8266 with NodeMCU?

NodeMCU is a firmware package that allows you to program the ESP8266 modules with Lua scripts. You will find it very similar to the way you program your Arduino.

With just a few lines of code you can establish a WiFi connection, control the ESP8266 GPIOs, set your ESP8266 as a web server and a lot more.

With this firmware, your ESP8266 obtains functionalities of the ESP8266's internal microcontroller, such as, SPI, UART, I2C, PWM, GPIO and more...

External resources

- NodeMCU: http://nodemcu.com/index_en.html
- Firmware: <https://github.com/nodemcu/nodemcu-firmware>
- Firmware Flasher: <https://github.com/RuiSantosdotme/nodemcu-flasher>
- NodeMCU Documents: <http://nodemcu.readthedocs.io/en/master>

Downloading NodeMCU Firmware Binaries

To download the latest binaries with the modules required for this eBook, I've created a GitHub repository with the latest binaries for the NodeMCU firmware.

Open the following link:

- <https://github.com/RuiSantosdotme/NodeMCU-Binaries>

Click the URL that ends with “**-integer.bin**”.

RuiSantosdotme / NodeMCU-Binaries

Code Issues 0 Pull requests 0 Wiki Pulse Graphs Settings

Unwatch 1 Star 0 Fork 0

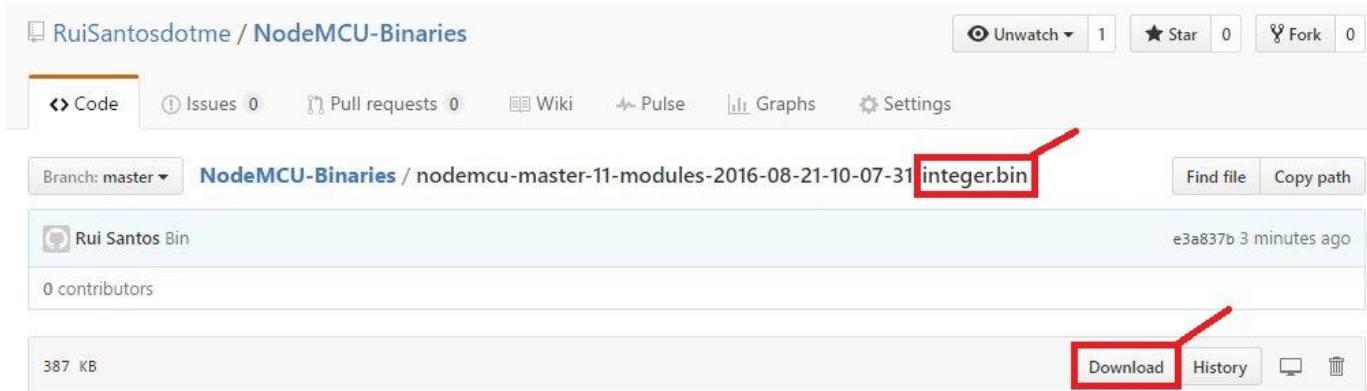
NodeMCU Binaries <http://randomnerdtutorials.com/> — Edit

2 commits 1 branch 0 releases 0 contributors

Branch: master ▾ New pull request Create new file Upload files Find file Clone or download ▾

File	Action	Time
README.md	Create README.md	a minute ago
nodemcu-master-11-modules-2016-08-21-10-07-31-float.bin	Bin	3 minutes ago
nodemcu-master-11-modules-2016-08-21-10-07-31-integer.bin	Bin	3 minutes ago

Press the **Download** button.



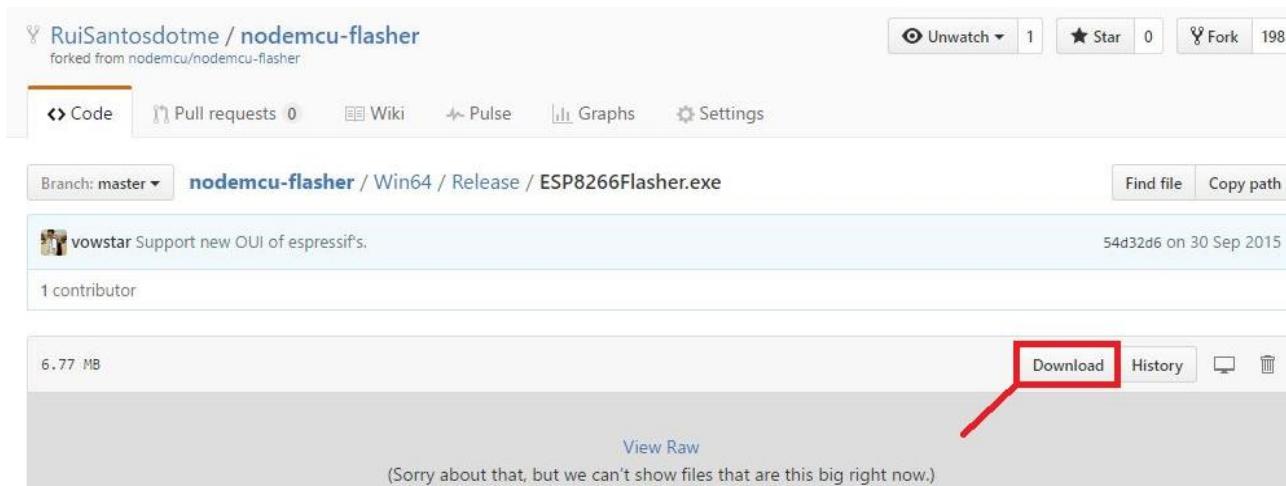
Downloading NodeMCU Flasher For Windows

At this point the official place to download the latest firmware was outdated. I've forked the project and I'll keep the firmware links updated.

Go to this link to download the NodeMCU Flasher for your Windows version:

- Windows 32 bits: <https://github.com/RuiSantosdotme/nodemcu-flasher/blob/master/Win32/Release/ESP8266Flasher.exe>
- Windows 64 bits: <https://github.com/RuiSantosdotme/nodemcu-flasher/blob/master/Win64/Release/ESP8266Flasher.exe>

To download the NodeMCU firmware flasher go to the preceding URL and press the "Download" button (as shown in the following figure).



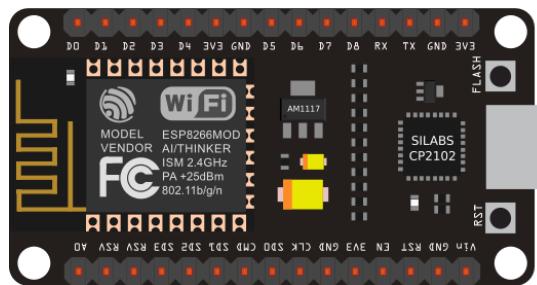
Flashing ESP8266

There are two different sections to upload code to your ESP8266. If you're using an ESP-12E that has built-in programmer read Option A.

If you're using the ESP-01 or ESP-07, you need an FTDI programmer - read Option B.

Option A - Flashing firmware to ESP-12E

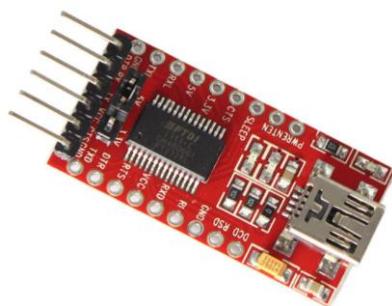
Flashing new firmware to your [ESP8266-12E NodeMCU Kit](#) is very simple, since it has built-in programmer. You plug your board to your computer and you don't need to make any additional connections:



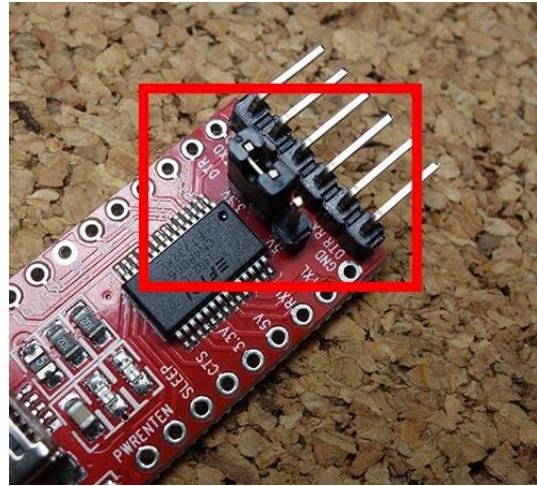
Option B – Flashing firmware to ESP-01

Uploading code to the [ESP-01](#) requires establishing a serial communication between your ESP8266 and a FTDI Programmer.

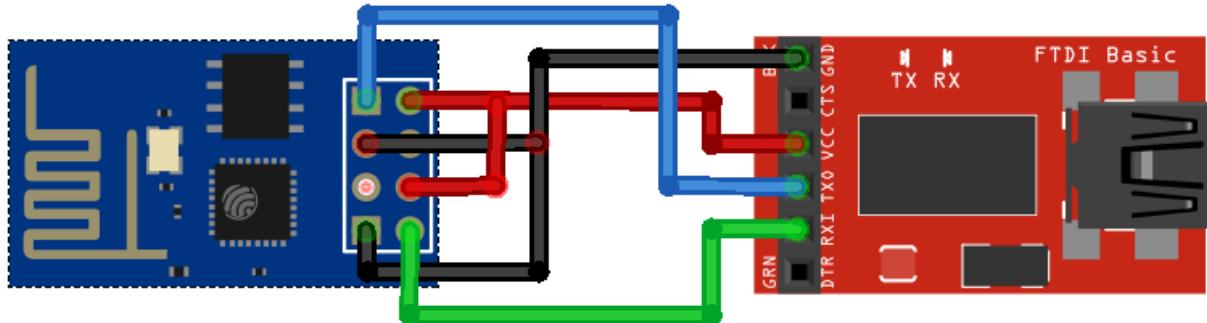
You can [click here to find the FTDI Programmer price at different stores.](#)



Important: most FTDI Programmers have a jumper to convert from 5V to 3.3V. Make sure your FTDI Programmer is set to 3.3V operation (as shown in the following figure).



Follow the circuit in the figure below to connect your ESP to your FTDI Programmer to establish a serial communication.



The following table shows the connections between the ESP-01 and the FTDI programmer:

ESP-01	FTDI Programmer
RX	TX
TX	RX
CH_PD	3.3V
GPIO 0	GND
VCC	3.3V
GND	GND

Important: when you want to upload code, GPIO 0 needs to be connected to GND because it requires the ESP8266 to flash a new firmware. In normal usage (if you're not flashing your ESP with a new firmware) it should be connected to VCC.

Download FTDI drivers

If you have a brand new FTDI Programmer and you need to install your FTDI drivers on Windows PC, visit the website in the following link to download the official drivers:

- <http://www.ftdichip.com/Drivers/VCP.htm>

Unbricking the FTDI Programmer on Windows PC

If you're having trouble installing the FTDI drivers on Windows 7/8/8.1/10 it's very likely that FTDI is bricked. Watch the video tutorial in the following link to fix that:

- <http://youtu.be/SPdSKT6KdF8>

In the previous video, it is said that you need to download the drivers from the FTDI website. Read carefully the video description to find all the links. The following link provides the drivers you need:

- <http://www.ftdichip.com/Drivers/CDM/CDM%20v2.12.00%20WHQL%20Certified.zip>

Once you have your ESP8266+FTDI Programmer connected to your computer, go to the next section.

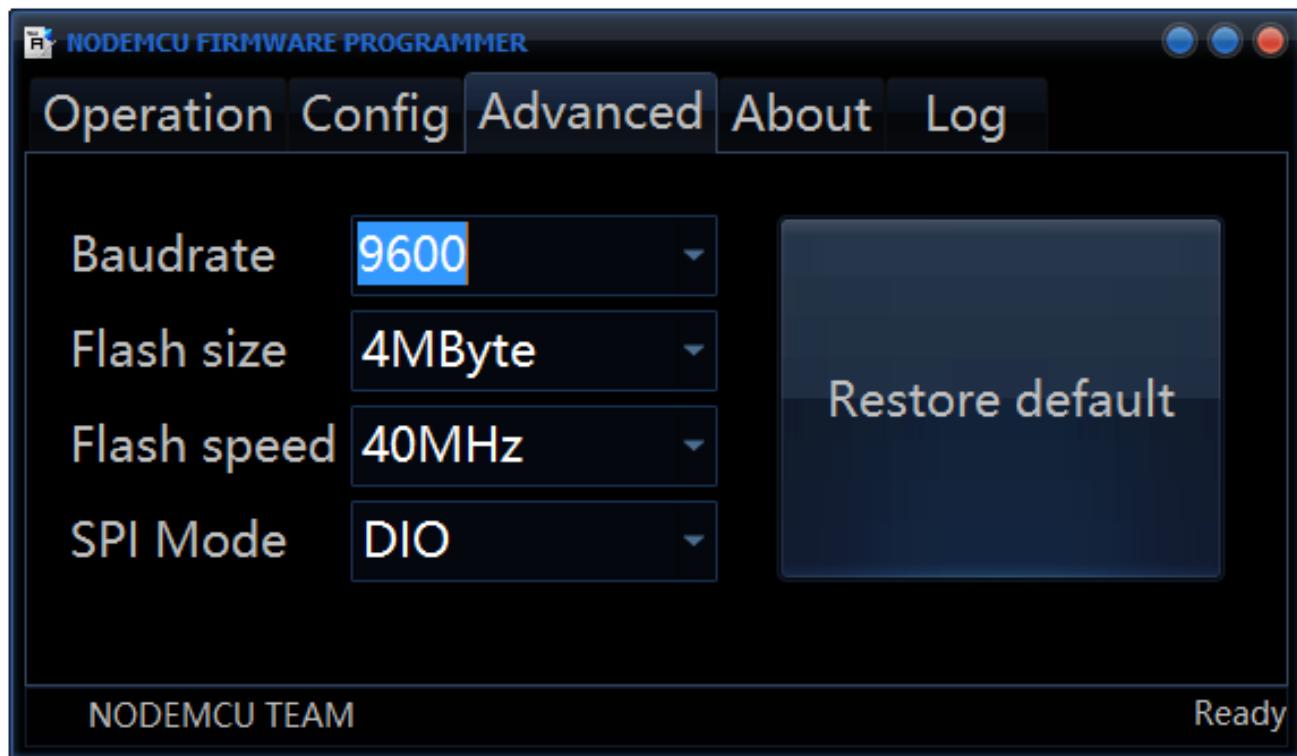
Flashing NodeMCU on Windows PC

At this point you have the circuit prepared to flash your ESP8266 and the NodeMCU firmware flasher for Windows. How do we load this firmware on the ESP8266 chip?

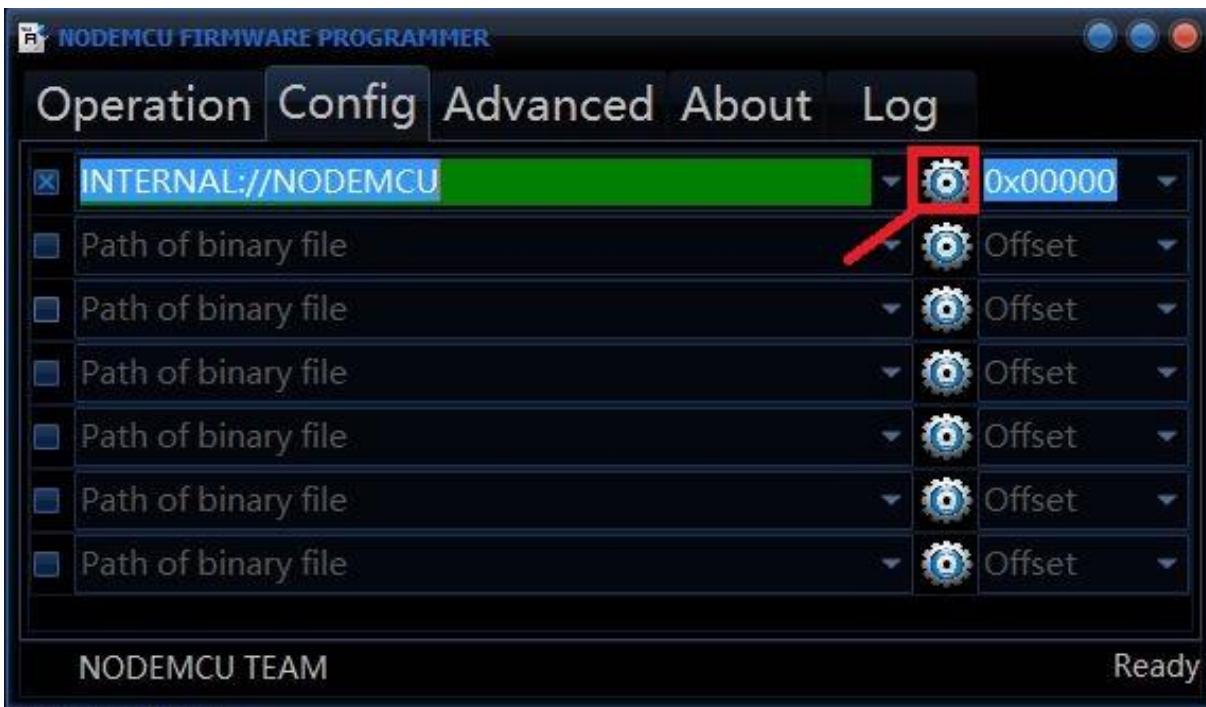
It's pretty simple. Having the serial communication established between your ESP8266 and your computer, you just need to follow these instructions:

1. Open the firmware flasher for Windows PC
2. Go to the **Advanced** tab (figure below)
3. Set your baud rate to 9600

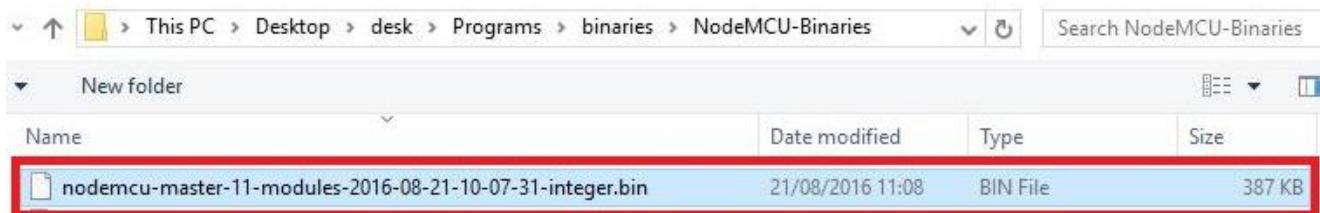
Note: some ESP8266 modules work at baud rate 115200 by default. If you're trying to flash and nothing happens, try to change the baud rate to 115200. Here's how your firmware flasher should look in the **Advanced** tab:



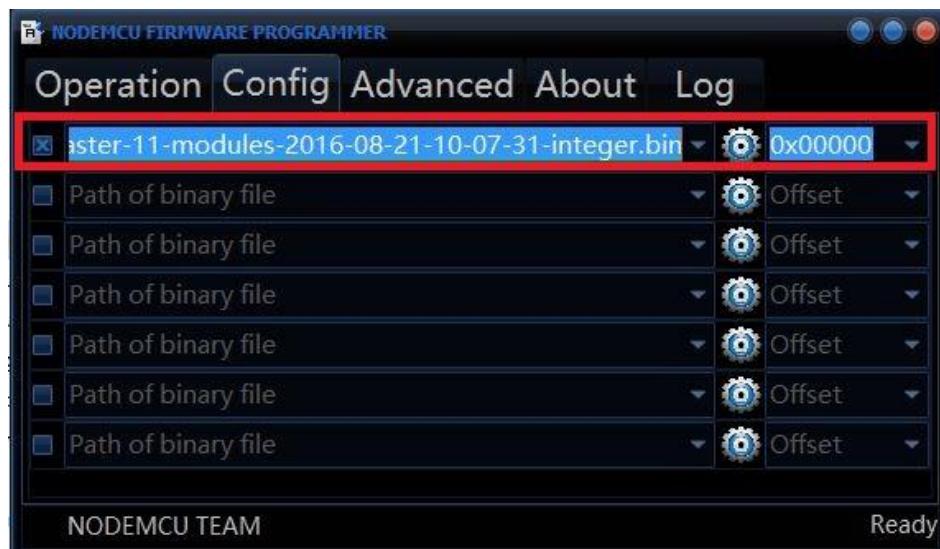
Then, open the **Config** tab and press the gear icon to change the binaries.



Select the NodeMCU binaries that you've download in the preceding section "Downloading NodeMCU Firmware Binaries".



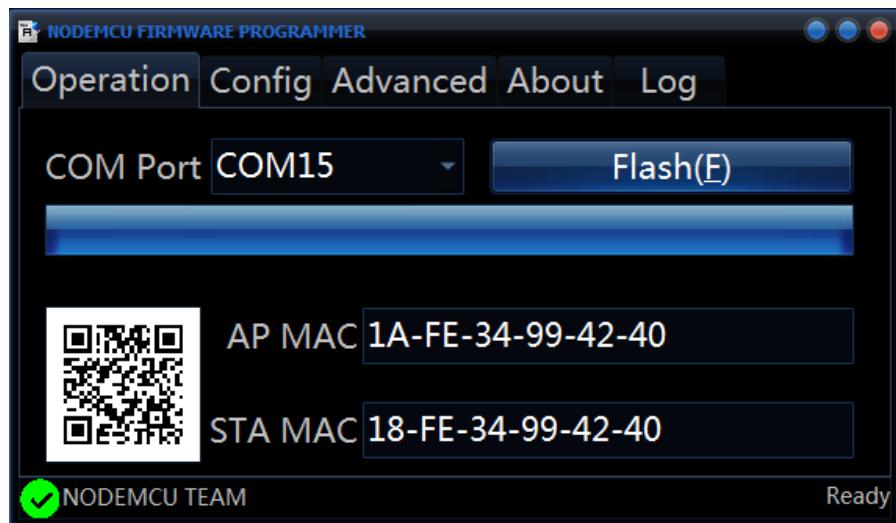
Here's how it should look like



Then, go to the **Operation** tab and follow these instructions:

1. Select the **COM Port** of your ESP12-E/FTDI Programmer
2. Press the **Flash(F)** button
3. That starts the flashing process

When it's finished you should see a green Check icon at the bottom left corner.



While the flash process is occurring, the ESP8266 blue LED blinks. If you don't see it blinking and your flasher is stuck, repeat this process again:

1. Check your connections
2. Disconnect the ESP/FTDI Programmer from the computer and reconnect it
3. Re-open your NodeMCU flasher
4. Check the NodeMCU flasher Advanced Settings
5. Try a different baud rate (9600 or 115200)
6. Try to flash your firmware again

When the flash process finishes, remove power from the ESP and disconnect GPIO 0 pin from GND.

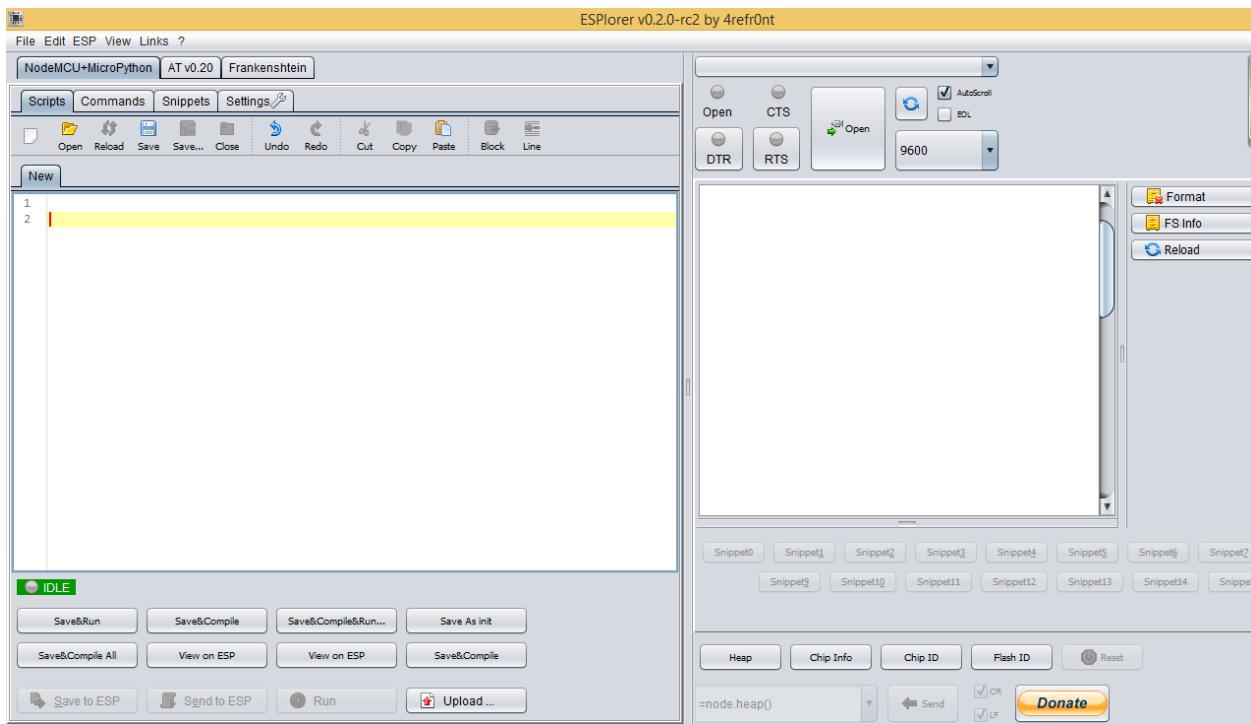
Flashing NodeMCU on Mac OS X or Linux

To flash NodeMCU in Mac OS X or Linux, here's a good tutorial on how to do that:

<http://www.whatimade.today/flashing-the-nodemcu-firmware-on-the-esp8266-linux-guide/>

If you have problems flashing the firmware in Mac OS X or Linux, please try to use a Windows PC or post a comment in the Facebook group.

Unit 2: Blinking LED with NodeMCU



In this Unit you're going to download and install ESPlorer (which is an open-source IDE for your ESP). You're also going to design a simple circuit to blink an LED with NodeMCU firmware.

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. Whether is an LED a lamp or your toaster.

What's the ESPlorer?

The ESPlorer is an IDE for ESP8266 developers. It's multiplatform, this simply means that it runs on Windows, Mac OS X or Linux (it was created in JAVA).

This software allows you to easily establish a serial communication with your ESP8266, send commands, upload code and much more.

Requirements

You need to have JAVA installed in your computer. If you don't have, go to this website: <http://java.com/download>, download and install the latest version.

External resources

- Download ESPlorer: <http://esp8266.ru/esplorer>
- GitHub Repository: <https://github.com/4refr0nt/ESPlorer>

Downloading ESPlorer

Now let's download the ESPlorer IDE, visit the following URL:
<http://esp8266.ru/esplorer/#download>

Then click the "Download ESPlorer.zip (v 0.2.0-rc6)" link.



Installing ESPlorer

Grab the folder that you just downloaded. It should be named "ESPlorer.zip". Unzip it. Inside that folder you should see the following files:

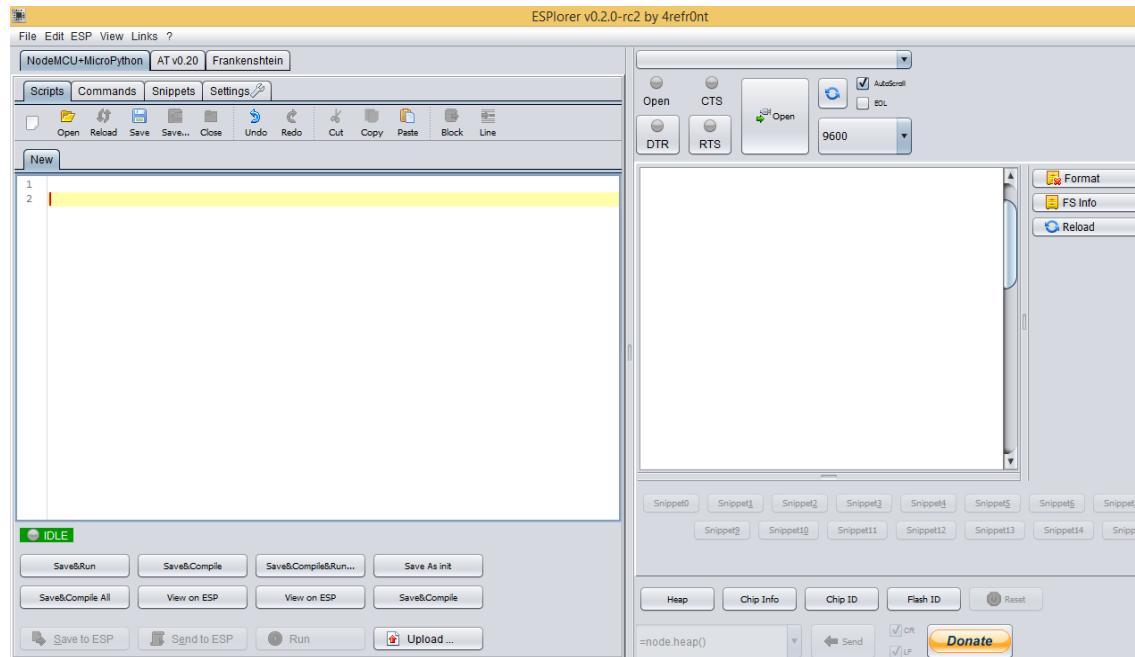
Name	Date modified	Type	Size
_lua	23/03/2015 14:02	File folder	
lib	23/03/2015 13:57	File folder	
ESPlorer	15/12/2014 23:49	Windows Batch File	1 KB
ESPlorer	26/04/2015 19:47	Executable Jar File	2 097 KB
version	26/04/2015 20:11	Text Document	1 KB

Execute the “ESPlorer.jar” file and the ESPlorer IDE should open after a few seconds (the “ESPlorer.jar” file is what you need to open every time you want to work with the ESPlorer IDE).

Note: if you’re on Mac OS X or Linux, just type this command line in your terminal to run the ESPlorer: ***sudo java -jar ESPlorer.jar***

ESPlorer IDE

When you first open the ESPlorer IDE, this is what you should see:



ESPlorer IDE has a lot of options you don’t need and you might feel overwhelmed with all those buttons and menus.

But don't worry, I'll go over the features you will need to complete all the projects in this eBook.

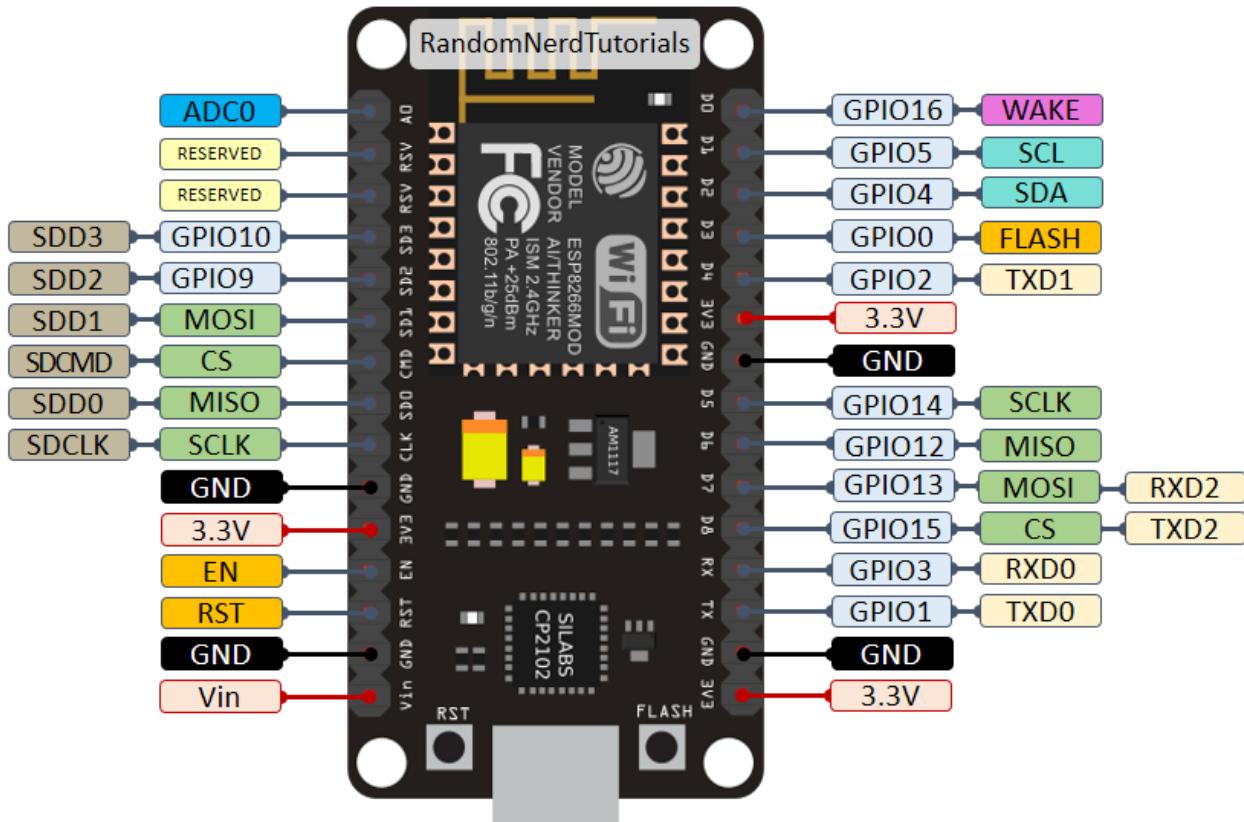
Do not close the ESPlorer IDE, you're going to use it in just a few minutes.

About GPIOs Assignment

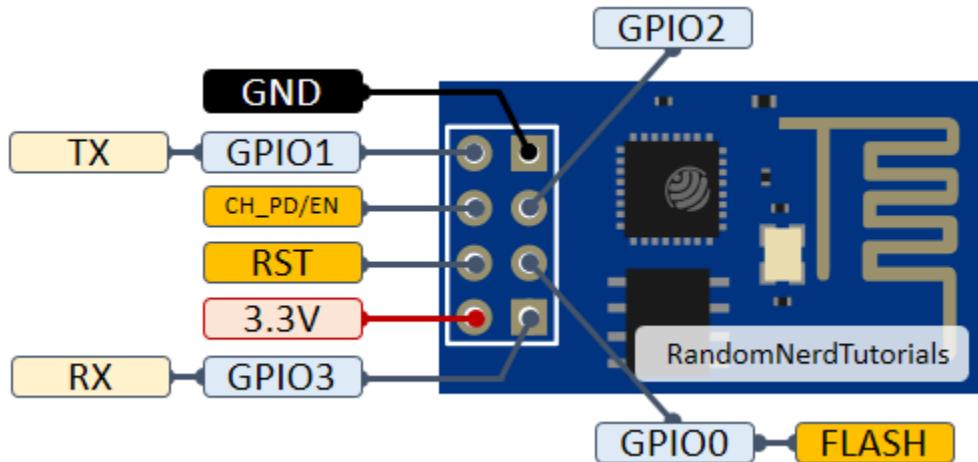
This page provides a quick pinout reference for ESP8266-12E. Use the table below for a quick reference on how to assign the ESP GPIOs in Lua code:

IO index (in code)	ESP8266 GPIO
0 [*]	GPIO 16
1	GPIO 5
2	GPIO 4
3	GPIO 0
4	GPIO 2
5	GPIO 14
6	GPIO 12
7	GPIO 13
8	GPIO 15
9	GPIO 3
10	GPIO 1
11	GPIO 9
12	GPIO 10

Here's the location for each pin in the actual board:



Just a quick recap, here's the ESP-01 pinout (all pins operate at 3.3V):



Important: in the next section called “Writing Your Lua Script” when we define:

```
pin = 3
```

we are referring to **GPIO 0**, and if we define:

```
pin = 4
```

we are referring to **GPIO 2**.

This is how this firmware is internally defined. You don't need to worry about this, simply remember that 3 refers to GPIO 0 and 4 refers to GPIO 2.

I'll explore this concept in more detail later in this eBook.

Writing Your Lua Script

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

SOURCE CODE

[https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART2_LUA\[Unit2_blink.lua\]](https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART2_LUA[Unit2_blink.lua])

```
init.lua
1 lighton=0
2 pin=4
3 gpio.mode(pin,gpio.OUTPUT)
4 tmr.alarm(1,2000,1,function()
5     if lighton==0 then
6         lighton=1
7         gpio.write(pin,gpio.HIGH)
8     else
9         lighton=0
10        gpio.write(pin,gpio.LOW)
11    end
12 end)
```

How this script works:

1. Create a variable called *lighton* to control the current state of your LED
2. Define *pin = 4* (4 refers to GPIO 2) as an *OUTPUT*

3. Next create a `tmr.alarm()` function that is executed every 2 seconds (2000 milliseconds)
4. The script checks the value of the variable `lighton`. If the variable contains 0, it means the output pin is *LOW* and the LED is off. The program then changes the variable `lighton` to 1, set the pin to *HIGH*, and the LED turns on.
5. If the variable `lighton` did not contain 0 it means the LED is on. Then the script would run the statements in the `else` section. It would first reassign the variable `lighton` to contain 0, and then change the pin to *LOW* which turns the LED off.
6. The program repeats steps 4. and 5. every two seconds, which causes the LED to blink!

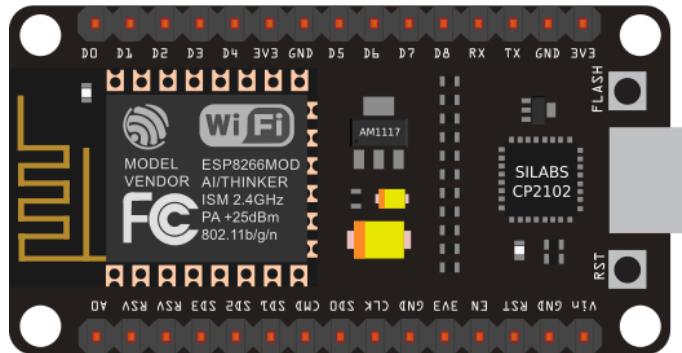
Note: you should name your Lua script “`init.lua`”, that ensures your ESP8266 executes your script every time it restarts.

Uploading Code to ESP8266

There are two different sections to upload code to your ESP8266. If you’re using an ESP-12E that has built-in programmer read Option A. If you’re using the ESP-01 or ESP-07, you need an FTDI Programmer - read Option B.

Option A - Uploading code to ESP-12E

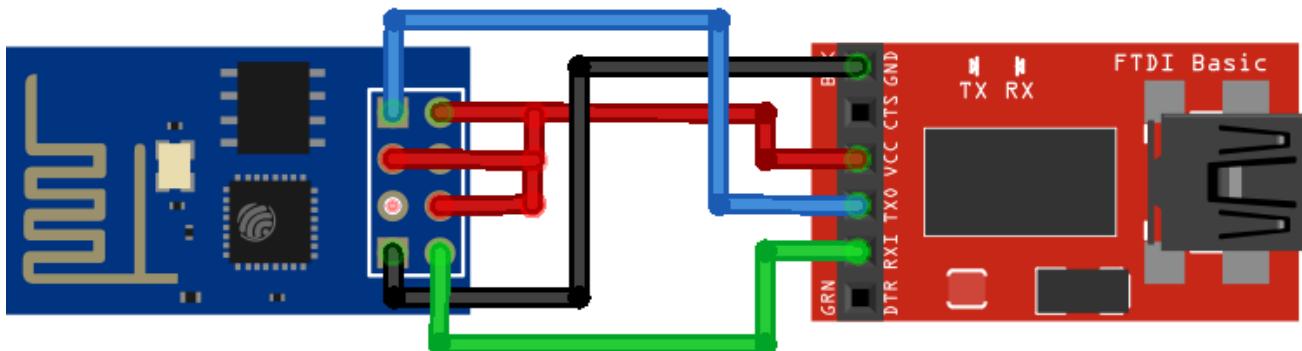
Uploading code to your ESP-12E NodeMCU Kit is simple, since it has built-in programmer. You plug your board to your computer and you don’t need to make any additional connections:



Option B - Uploading code to ESP-01

With the ESP-01 or ESP-07 you need to establish a serial communication with your ESP8266 using an FTDI Programmer to upload code.

Follow the circuit in the figure below to connect your ESP to your FTDI Programmer to establish a serial communication.



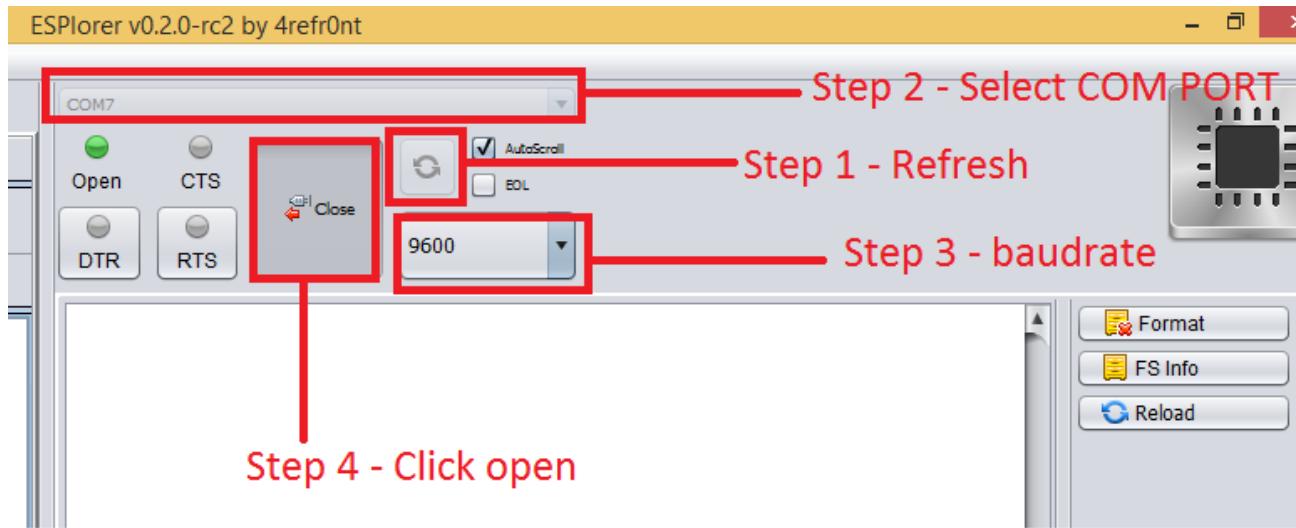
Note: the circuit above has GPIO 0 connected to VCC because we want to upload code.

Uploading init.lua Script

Having your ESP8266 connected to your computer, go to the ESPlorer IDE. Look at the top right corner of your ESPlorer IDE and follow these instructions:

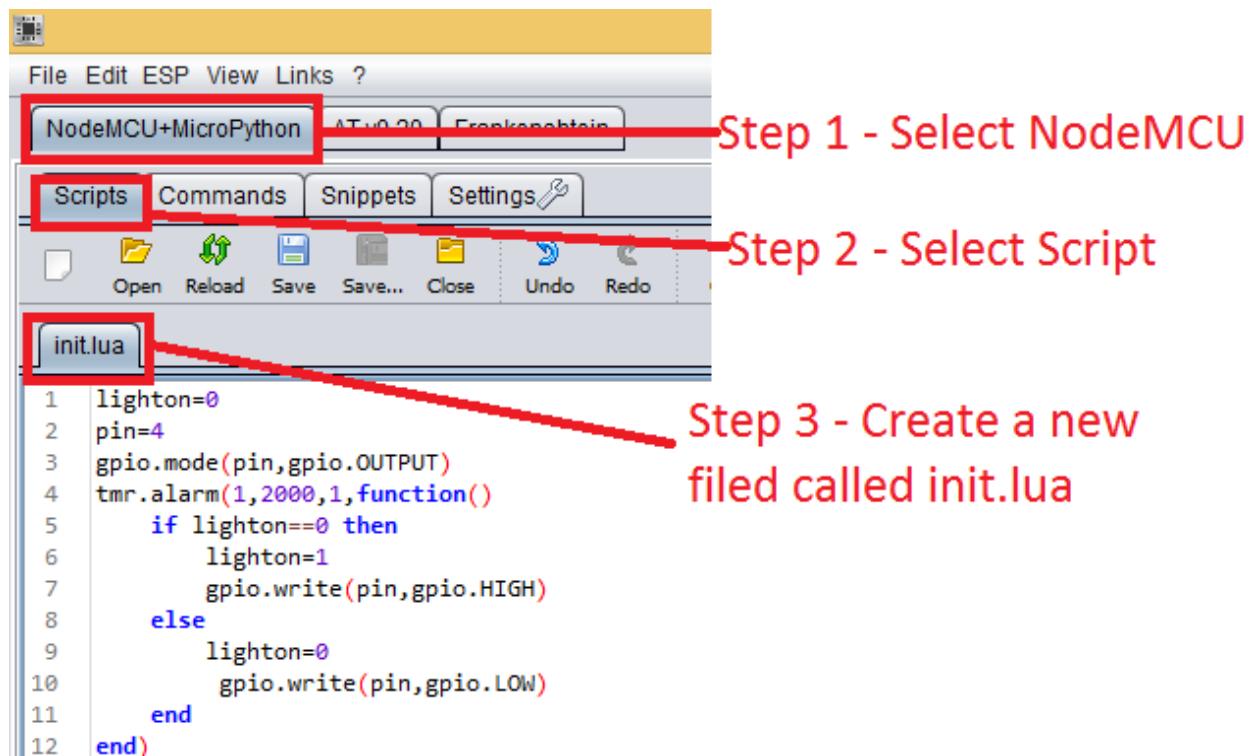
1. Press the Refresh button
2. Select the COM port for your ESP-12E/FTDI Programmer

3. Select 9600 as your baud rate
4. Click Open



Then in the top left corner of the ESPlorer IDE, follow these instructions:

1. Select NodeMCU
2. Select Scripts
3. Create a new file called "init.lua"



Copy the Lua script (which was created in the previous section) to the code window (as you can see in the figure below):

The screenshot shows the NodeMCU+MicroPython software interface. At the top, there's a menu bar with File, Edit, ESP, View, Links, and a question mark icon. Below the menu is a toolbar with tabs for NodeMCU+MicroPython, AT v0.20, and Frankenstein. The main window has tabs for Scripts, Commands, Snippets, and Settings. Below the tabs is a toolbar with icons for Open, Reload, Save, Save..., Close, Undo, Redo, Cut, Copy, and Paste. The main code editor window is titled "init.lua" and contains the following Lua code:

```
1 lighton=0
2 pin=4
3 gpio.mode(pin,gpio.OUTPUT)
4 tmr.alarm(1,2000,1,function()
5     if lighton==0 then
6         lighton=1
7         gpio.write(pin,gpio.HIGH)
8     else
9         lighton=0
10        gpio.write(pin,gpio.LOW)
11    end
12)
13
```

A red rectangular box highlights the code area. To the right of the code, the text "Step 1 - Copy your code to this window" is displayed in red.

The next step is to save your code to your ESP8266.

At the left bottom corner press the button "**Save to ESP**".

The output window displays the commands being sent to the ESP8266. It should look similar to the next figure.

```

1 lighton=0
2 pin=4
3 gpio.mode(pin,gpio.OUTPUT)
4 tmr.alarm(1,2000,1,function()
5     if lighton==0 then
6         lighton=1
7         gpio.write(pin,gpio.HIGH)
8     else
9         lighton=0
10        gpio.write(pin,gpio.LOW)
11    end
12 end)
13

```

Output Window

Click Save to ESP to upload your init.lua script

You can remove your init.lua, if you type:
file.remove("init.lua") and click "Send"

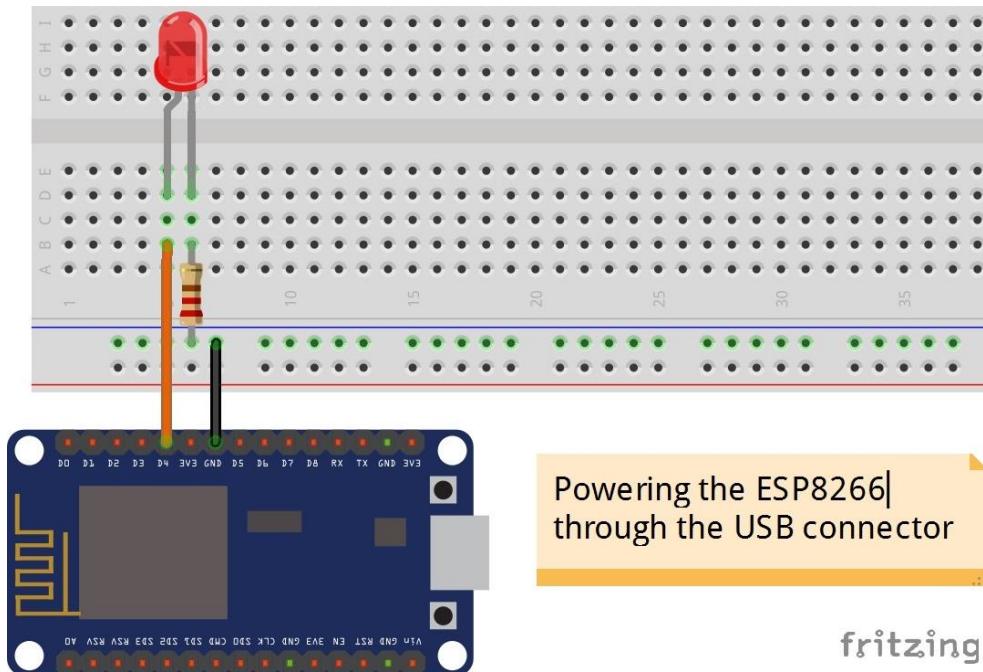
Note: you can easily delete the "init.lua" file from the ESP. Simply type `file.remove("init.lua")` and press the button "Send" (see figure above). Or you can type the command `file.format()` to remove all the files saved in your ESP8266.

Final Circuit

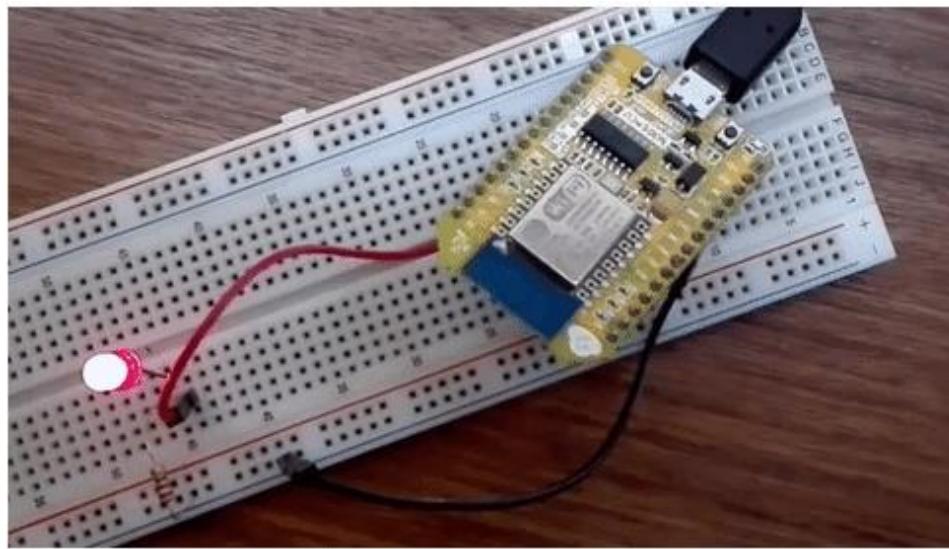
After uploading the code, unplug the ESP8266 from your computer. Next, change the wiring to match the following diagrams.

Final ESP-12E circuit

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



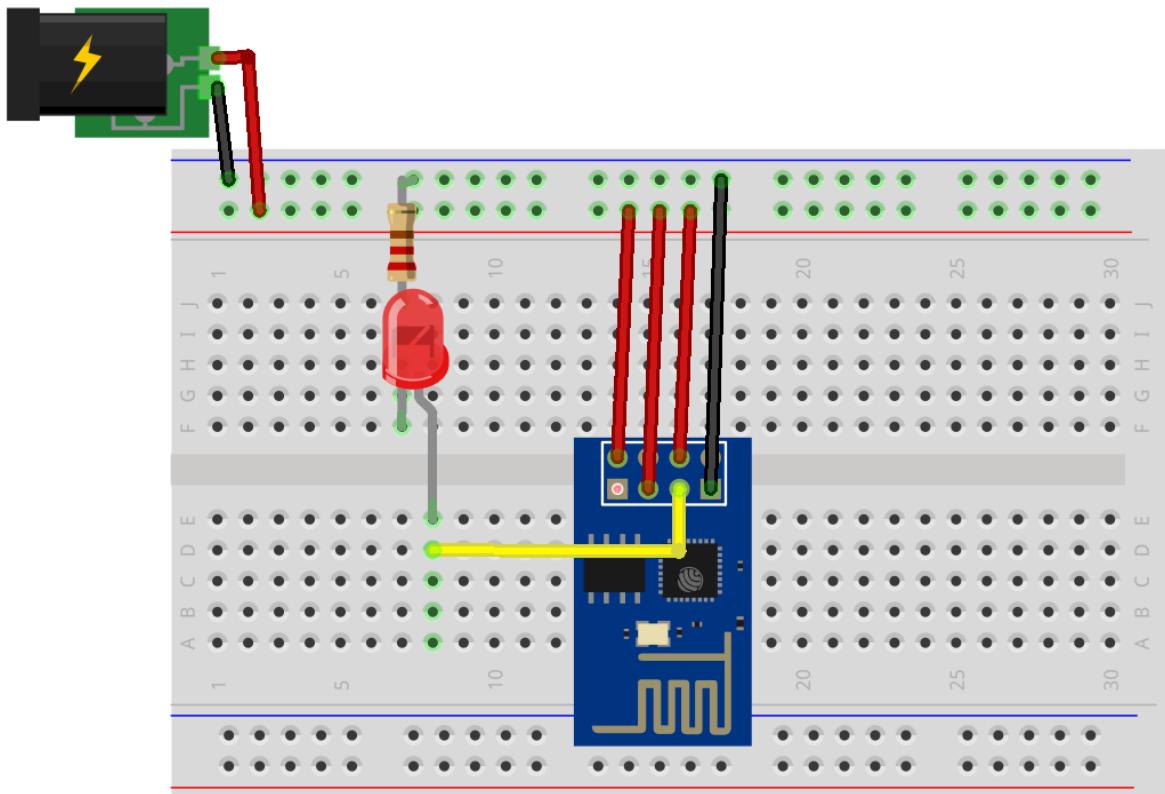
Restart your ESP8266.



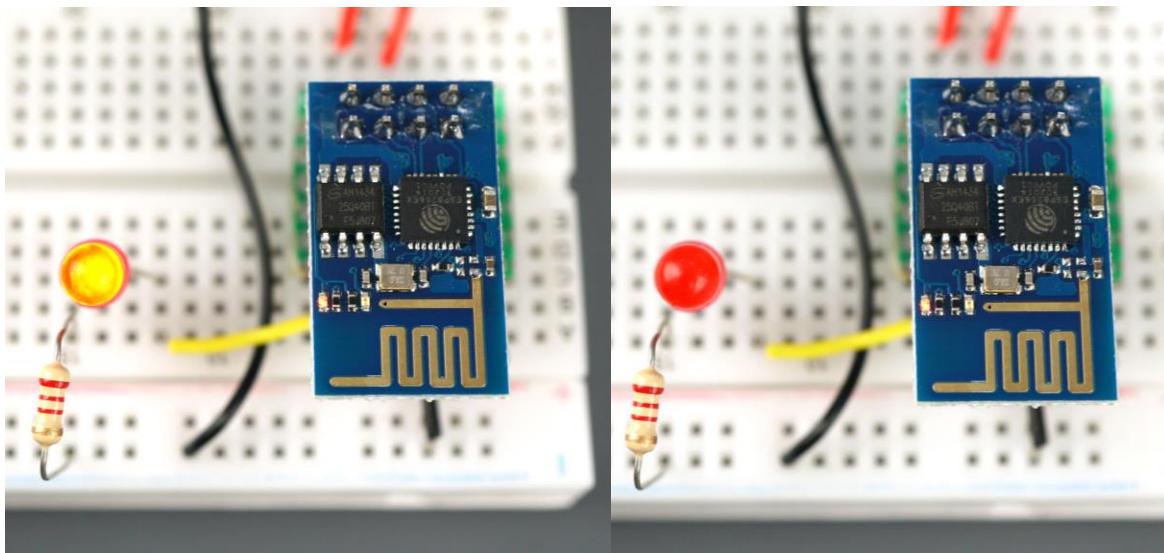
Final ESP-01 circuit

Then, apply power from a 3.3V source or use your FTDI Programmer to the ESP.

3.3V



Restart your ESP8266. Congratulations, you've made it! Your LED should be blinking every 2 seconds.



Unit 3: Lua Programming Language – The Basics



Before diving deeper into more projects with the ESP8266 I thought it would be helpful to create a Unit dedicated to Lua Programming language. Lua is a light weight programming language written in C. It started as an in-house project in 1993 by Roberto Ierusalimschy, Luiz Henrique de Figueiredo and Waldemar Celes.

More details about this program language can be found here:
[https://en.wikipedia.org/wiki/Lua_\(programming_language\)](https://en.wikipedia.org/wiki/Lua_(programming_language))

NodeMCU is a Lua based firmware package for the ESP8266, so it's important that you know the Lua basics in order to write your own scripts for the ESP8266.

Variables

In Lua, though we don't have variable data types, we have three types based on the scope of the variable. The scope means that a variable can be either of global or local scope.

- **Global variables:** All variables are considered global (unless it is declared as a local)

```
pin = 3
test = "It works!"
```

- **Local variables:** When the type is specified as local for a variable, its scope is limited with the functions inside their scope

```
local pin = 3
local test = "It works!"
```

- **Table fields:** This is a special type of variable that can hold anything except *nil* (we won't cover table fields)

Note: Lua is case-sensitive. So, a variable called *P/N* is different from *Pin* or *pin*.

Data Types (Value Types)

Lua is a dynamically typed language, so the variables don't have types, only the values have types. Values can be stored in variables, passed as parameters and returned as results.

The list of data types for values are given below.

Value Type	Description
string	Arrays of characters
number	Represents real (double precision floating point) numbers
boolean	Includes true and false as values. Generally used for condition checking.
function	A method that is written in Lua
nil	No data stored in the variable
table, userdata and thread	We won't cover these 3 value types in this eBook.

Here's a great illustration of the value types in action:

```
print(type("Hello World!"))    -- string
print(type(7))                  -- number
print(type(true))                -- boolean
print(type(print))              -- function
print(type(nil))                -- nil
```

Note: when working with NodeMCU in your ESP8266, you'll see the value *nil* come up once in a while. It simply means that a variable is not defined. Also, if you want to delete a variable, simply set that variable to the *nil* value.

Comments

Comments are plain text that explains how the code works. Anything designated as a comment is ignored by the ESP module. Comments start with two dashes: --. There are two types of comments:

- Single-line comment

```
print("Hello World!") -- That's how you make a comment
```

- Multi-line comment

```
--[[  
print("Hello World!") this is a multi line comment  
--]]
```

Operators

An operator is a symbol that tells the interpreter to perform specific mathematical or logical manipulations. Lua language is rich in built-in operators and provides following type of operators:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Misc Operators

For all the following tables and examples in this section assume that you have two variables: *A* that stores number 1 and a variable *B* that stores number 2.

A = 1
B = 2

Arithmetic operators

Operator	Example	Result
+	A + B	3
-	A - B	-1
*	A * B	2
/	B / A	2
%	B % A	0
^	B^2	4
-	-A	-1

Relational operators

Operator	Example	Result
==	(A == B)	not true
~=	(A ~= B)	true
>	(A > B)	not true
<	(A < B)	true
>=	(A >= B)	not true
<=	(A <= B)	true

Logical operators

Operator	Example	Result

and	(A and B)	false
or	(A or B)	true
not	!(A and B)	true

Concatenation operator

Now imagine that you have two new variables:

```
a = "Hello "
b = "World!"
```

Operator	Example	Result
..	a..b	"Hello World!"

Loops

A loop allows us to execute a block of code multiple times for as long as the condition (*boolean_value*) is true.

```
--While Loop
while bloean_value
do
  -- executes this code while is true
end

-- or Foor Loop
for init,max/min value, increment
do
  -- executes this code while is true
end
```

if... else statements

if... else statements are one the most important coding tools for adding control to your program. *if... else* statements are used as follows:

```
if boolean_value then
    -- if the boolean_value is true
else
    -- if the boolean_value is false
end
```

Their use is just as the words describe them: If a certain condition is met (*boolean_value=true*), then the code inside the *if* statement runs. If the condition is false (*boolean_value=false*), the code inside the *else* statement runs.

Functions

Functions are great ways to organize your code. If you want to do something multiple times, instead of repeating your code several times, you create a separate function that you can call and execute any time.

This is how you create a new function that takes one parameter (the temperature in Kelvin) and converts that temperature to both Celsius and Fahrenheit:

```
function displayTemperature(kelvin)
    celsius = kelvin - 273.15
    print("Temperature in Celsius is: ", celsius)

    fahrenheit = (celsius*9/5+32)
    print("Temperature in Fahrenheit is: ", fahrenheit)
end

k = 294 --temperature in Kelvin
displayTemperature(k) -- calls function
```

Unit 4: Interacting with the ESP8266

GPIOs using NodeMCU Firmware



GPIO stands for *general purpose input/output*, which sums up what pins in this mode can do: they can be either inputs or outputs for the vast majority of applications.

In this Unit we're going to explore the NodeMCU GPIO API. If you want learn more about this API, you can visit the NodeMCU official documentation:

- <https://nodemcu.readthedocs.io/en/master/>

This Unit explains how to set the different modes for each GPIO. I will share snippets of code that can be applied to your projects.

Since these code snippets work with the ESP8266, feel free to test them!

Pin Mode

When using a GPIO pin we need to specify its mode of operation. There are three possible modes that you can assign to each pin (one mode at a time for any pin):

Mode	Reference	Description
OUTPUT	gpio.OUTPUT	You set the pin to HIGH or LOW
INPUT	gpio.INPUT	You read the current state of the pin
INTERRUPT	gpio.INT	Similar to INPUT, you're constantly checking for a change in a pin. When a change occurs, it executes a function.

How to assign pins

The table below shows the GPIO pin index assignments for the ESP8266. The ESP-01 has only two: GPIO 0 and GPIO 2:

IO index (in code)	ESP8266 GPIO
0 [*]	GPIO 16
1	GPIO 5
2	GPIO 4
3	GPIO 0
4	GPIO 2
5	GPIO 14
6	GPIO 12
7	GPIO 13
8	GPIO 15
9	GPIO 3
10	GPIO 1
11	GPIO 9
12	GPIO 10

OUTPUT mode

Using `gpio.write()` you can set any GPIO to HIGH (3.3V) or LOW (0V). That's how you turn an LED on or off.

Here is how to make the pin GPIO 2 put out a HIGH (3.3V):

```
pin = 4
gpio.mode(pin, gpio.OUTPUT)
gpio.write(pin, gpio.HIGH)
```

Here is how to make the pin GPIO 2 put out a LOW (0V):

```
pin = 4
gpio.mode(pin, gpio.OUTPUT)
gpio.write(pin, gpio.LOW)
```

INPUT mode

Using *gpio.read()* you can read the current state of any GPIO. For example, here is how you check if a button was pressed.

```
pin = 4
gpio.mode(pin, gpio.INPUT)
print (gpio.read(pin))
```

If *print(gpio.read(pin))* = 1 the button was being pressed and if *print(gpio.read(pin))* = 0 the button was released, or was not pressed.

INTERRUPT mode

The *gpio.trig()* allows you to detect when something occurs in a pin (when its state changes from HIGH to LOW or vice-versa). When that event occurs, it executes a function.

Imagine that you have a motion sensor. When motion is detected you want to trigger a specific function that does something (send an email for example).

To create an interrupt, this is what you would do:

```

pin = 4
gpio.mode(pin, gpio.INT)

function onChange ()
    print('Motion Detected')
end

gpio.trig(pin, 'up', onChange)

```

The second parameter of the function called `gpio.trig(pin, <event_name>, <function_name>)` can take these 5 types of events:

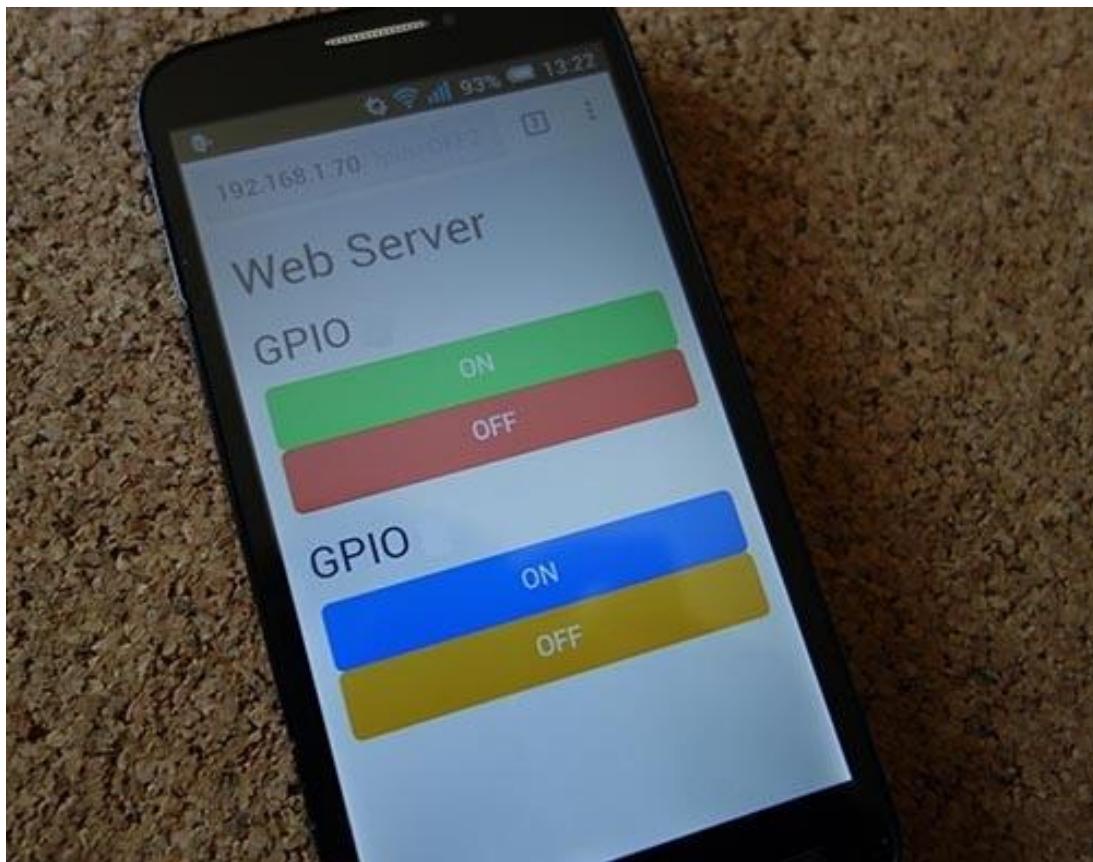
Event name	When occurs
up	Pin goes to HIGH
down	Pin goes to LOW
both	Pin goes LOW->HIGH or HIGH->LOW
low	While Pin is LOW
high	While Pin is HIGH

Unit 5: Web Server with ESP8266

In this Unit you're going to create a web server with your ESP8266 that can be accessed with any device that has a browser. This means you can control the ESP GPIOs from your laptop, smartphone, tablet and so on!

In this project you're going to control two LEDs. This is just an example; the idea is to replace those LEDs with a Power Switch Tail or a relay to control any electronic devices that you want.

Spoiler alert: this is what you're going to achieve at the end of this project!



Writing Your Lua Script

You can download the Lua script to create a web server that controls two outputs (GPIO 5 and GPIO 4) below:

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART2_LUA/Unit5_1_web_server.lua

Let's see how this code works.

The snippet of code below starts by setting the mode of your ESP8266 to a station. Then, you configure your ESP8266 with your own credentials (network name and password). You need to replace that second line with your credentials, so that your ESP can connect to your network.

The *print()* function in line 3 prints your ESP8266 IP address in the output window of the ESPlorer IDE (you need that IP to access your web server).

Next, you create two variables (*led1* and *led2*) which refer to GPIO 5 and GPIO 4 respectively and define them as *OUTPUTs*.

```
1 wifi.setmode(wifi.STATION)
2 wifi.sta.config("YOUR_NETWORK_NAME", "YOUR_NETWORK_PASSWORD")
3 print(wifi.sta.getip())
4 led1 = 1
5 led2 = 2
6 gpio.mode(led1, gpio.OUTPUT)
7 gpio.mode(led2, gpio.OUTPUT)
```

The next thing to do is creating your web server on port 80. You do it like this:

```

o
9  srv=net.createServer(net.TCP)
10 srv:listen(80,function(conn)
11
12     end)
13 end)

```

Inside your web server you tell exactly what happens when a connection is established with a client *conn:on()*. You also create a few local variables that store your web page and your current URL path.

```

11 conn:on("receive", function(client,request)
12     local buf = ""
13     local _, _, method, path, vars = string.find(request, "([A-Z]+) (.+)?(.+) HTTP");
14     if(method == nil)then
15         _, _, method, path = string.find(request, "([A-Z]+) (.+) HTTP");
16     end
17     local _GET = {}
18     if (vars ~= nil)then
19         for k, v in string.gmatch(vars, "(%w+)=(%w+)&*") do
20             _GET[k] = v
21         end
22     end

```

The *buf* variable stores your web page. It's just a basic web page that uses the Bootstrap framework (see next snippet of code).

Learn more about the Bootstrap framework: <http://getbootstrap.com>.

Your web page has four buttons to turn your LEDs *HIGH* and *LOW*. Two buttons for GPIO 5 and the other two for GPIO 4.

Your buttons are simply `` HTML tags with a CSS class that gives them the button look. So when you press a button, you open *another* web page that has a different URL. And that's how your ESP8266 knows what it needs to do (whether is to turn your LEDs *HIGH* or *LOW*).

```

23     buf = buf.."<head>";
24     buf = buf.."<meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">";
25     buf = buf.."<script src=\"https://code.jquery.com/jquery-2.1.3.min.js\"></script>";
26     buf = buf.."<link rel=\"stylesheet\" href=\"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css\">";
27     buf = buf.."</head><div class=\"container\">";
28
29     buf = buf.."<h1>Web Server</h1>";
30     buf = buf.."<h2>GPIO 0</h2>";
31     buf = buf.."<div class=\"row\">";
32     buf = buf.."<div class=\"col-md-2\"><a href=\"?pin=ON1\" class=\"btn btn-block btn-lg btn-primary\">ON1</a></div>";
33     buf = buf.."<div class=\"col-md-2\"><a href=\"?pin=OFF1\" class=\"btn btn-block btn-lg btn-primary\">OFF1</a></div>";
34     buf = buf.."</div>";
35     buf = buf.."<h2>GPIO 2</h2>";
36     buf = buf.."<div class=\"row\">";
37     buf = buf.."<div class=\"col-md-2\"><a href=\"?pin=ON2\" class=\"btn btn-block btn-lg btn-primary\">ON2</a></div>";
38     buf = buf.."<div class=\"col-md-2\"><a href=\"?pin=OFF2\" class=\"btn btn-block btn-lg btn-primary\">OFF2</a></div>";
39     buf = buf.."</div></div>";

```

This final snippet of code checks which button in your webpage was pressed. Basically, it checks the URL that you have just clicked.

Let's see an example. When you click the button OFF from the GPIO 5 you open this URL: <http://192.168.7.2/?pin=OFF1>. Your Lua code checks that URL and with some *if...else* statements it knows that you want your GPIO 5 (which is defined as *led1*) to turn *LOW*.

```

41     local _on,_off = "", ""
42     if(_GET.pin == "ON1")then
43         gpio.write(led1, gpio.HIGH);
44     elseif(_GET.pin == "OFF1")then
45         gpio.write(led1, gpio.LOW);
46     elseif(_GET.pin == "ON2")then
47         gpio.write(led2, gpio.HIGH);
48     elseif(_GET.pin == "OFF2")then
49         gpio.write(led2, gpio.LOW);
50     end
51     client:send(buf);
52     client:close();
53     collectgarbage();
54 end)
55 end)

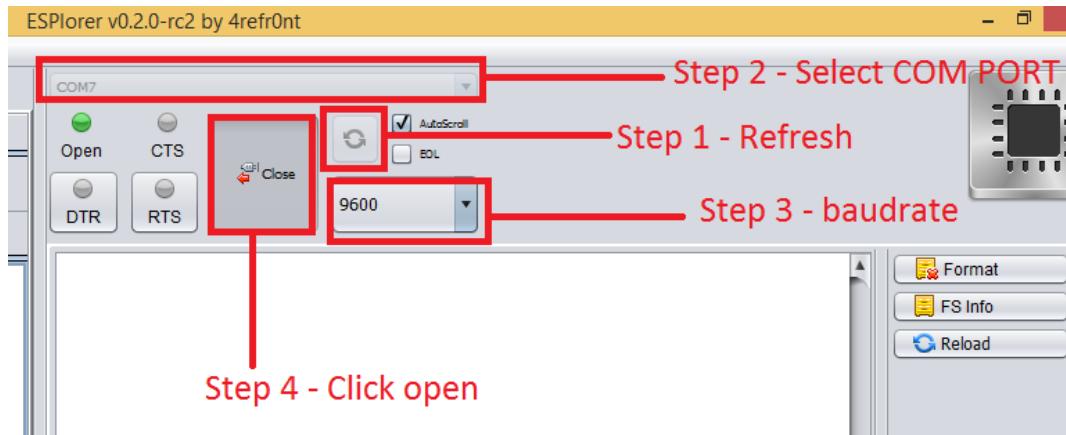
```

Uploading init.lua Script

Having your ESP8266 connected to your computer, go to the ESPlorer IDE. Look at the top right corner of your ESPlorer IDE and follow these instructions:

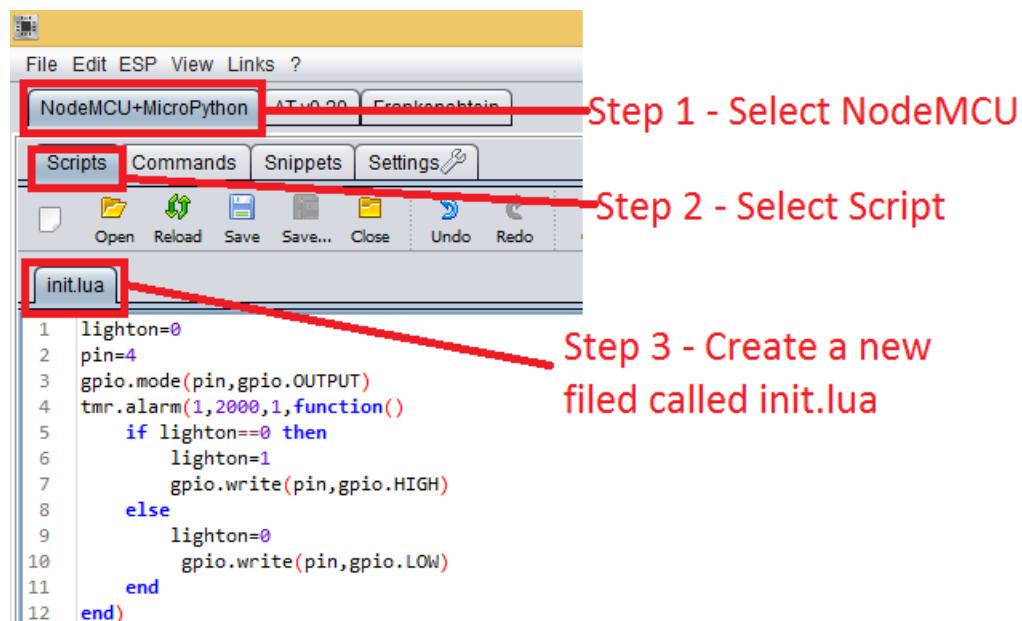
1. Press the Refresh button

2. Select the COM port for your ESP-12E/FTDI Programmer
3. Select 9600 as your baud rate
4. Click Open

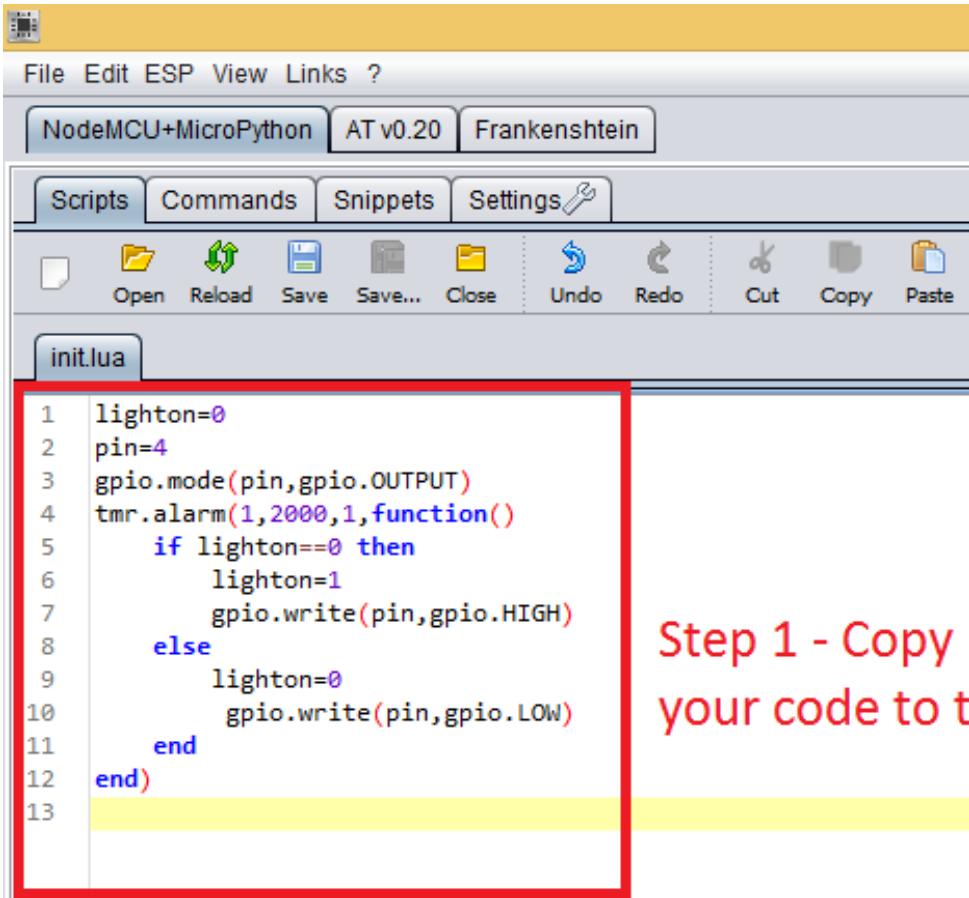


Then in the top left corner of the ESPlorer IDE, follow these instructions:

1. Select NodeMCU
2. Select Scripts
3. Create a new file called "init.lua"



Copy the Lua script (which was created in the previous section) to the code window (as you can see in the figure below):



The screenshot shows the NodeMCU+MicroPython software interface. The title bar includes "File Edit ESP View Links ?" and tabs for "NodeMCU+MicroPython", "AT v0.20", and "Frankenstein". Below the tabs are buttons for "Scripts", "Commands", "Snippets", and "Settings". A toolbar below the buttons contains icons for "Open", "Reload", "Save", "Save...", "Close", "Undo", "Redo", "Cut", "Copy", and "Paste". The main window displays a file named "init.lua" containing the following code:

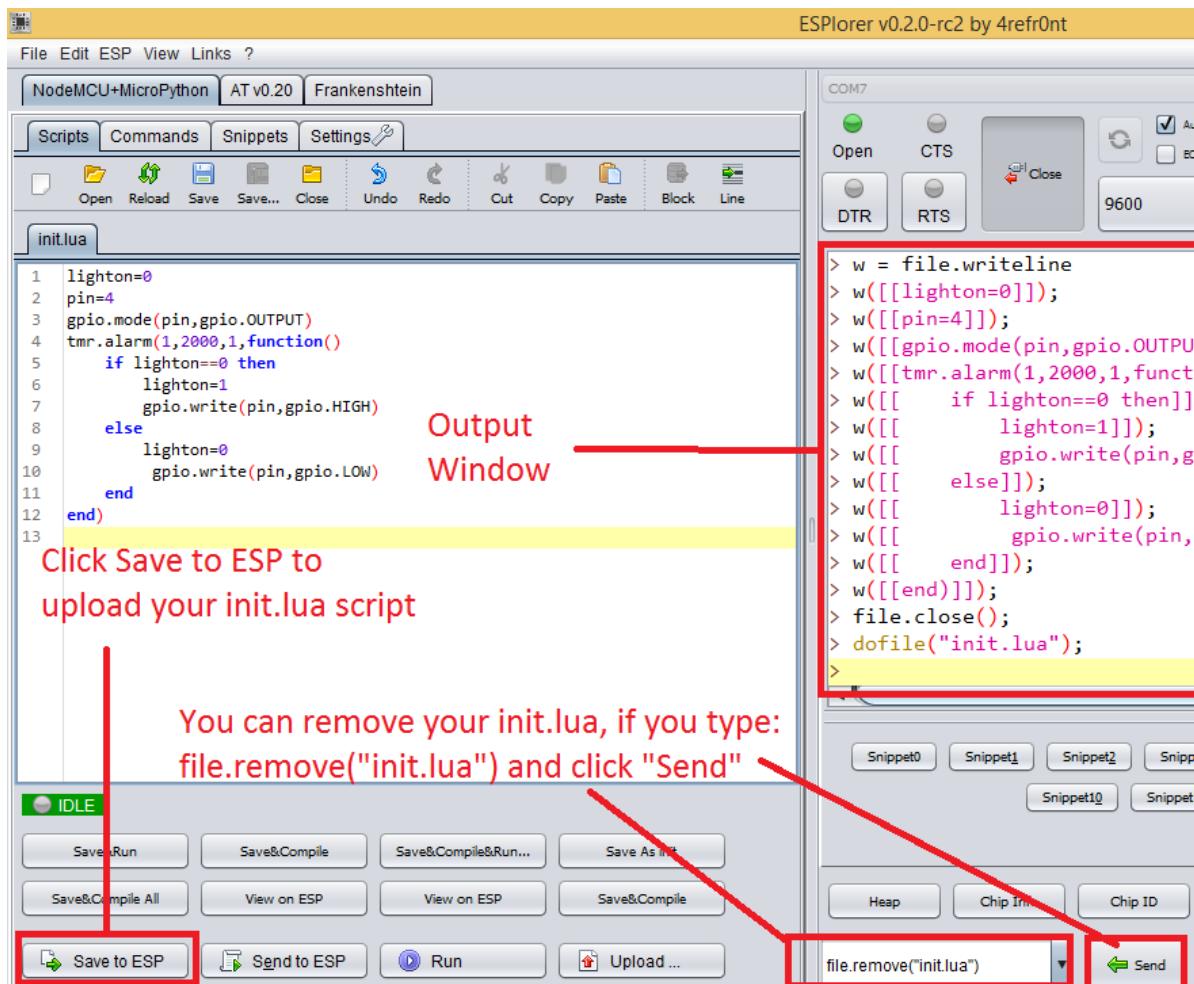
```
1 lighton=0
2 pin=4
3 gpio.mode(pin,gpio.OUTPUT)
4 tmr.alarm(1,2000,1,function()
5     if lighton==0 then
6         lighton=1
7         gpio.write(pin,gpio.HIGH)
8     else
9         lighton=0
10        gpio.write(pin,gpio.LOW)
11    end
12 end)
13
```

A red box highlights the code area in the editor. To the right of the code, the text "Step 1 - Copy your code to this window" is displayed in red.

The next step is to save your code to your ESP8266.

At the left bottom corner press the button "**Save to ESP**".

The output window displays the commands being sent to the ESP8266. It should look similar to the figure below.



Note: you can easily delete the "init.lua" file from the ESP. Simply type `file.remove("init.lua")` and press the button "Send" (see figure above). Or you can type the command `file.format()` to remove all the files saved in your ESP8266.

ESP8266 IP Address

After uploading your web server Lua script to your ESP8266, in your output window you're going to see 3 IP addresses.

The IP that that you want is the first one, in my case it's: 192.168.1.70. Your IP should be different, **save your ESP8266 IP** so you can access it later in this Unit.

```

f = buf.."<div class='row'>";
f = buf.."<div class='col-md-2'><a href=?pin=ON2? class='btn btn-primary'>ON</a></div>";
f = buf.."<div class='col-md-2'><a href=?pin=OFF2? class='btn btn-primary'>OFF</a></div>";
f = buf.."</div></div>";

_on,_off = "", ""
if(_GET.pin == "ON1")then
    gpio.write(led1, gpio.HIGH);
elseif(_GET.pin == "OFF1")then
    gpio.write(led1, gpio.LOW);
elseif(_GET.pin == "ON2")then
    gpio.write(led2, gpio.HIGH);
elseif(_GET.pin == "OFF2")then
    gpio.write(led2, gpio.LOW);

rsrui/Desktop/init.lua

```

Your IP _____

```

192.168.1.70 255.255.255.0 192.168.1.254

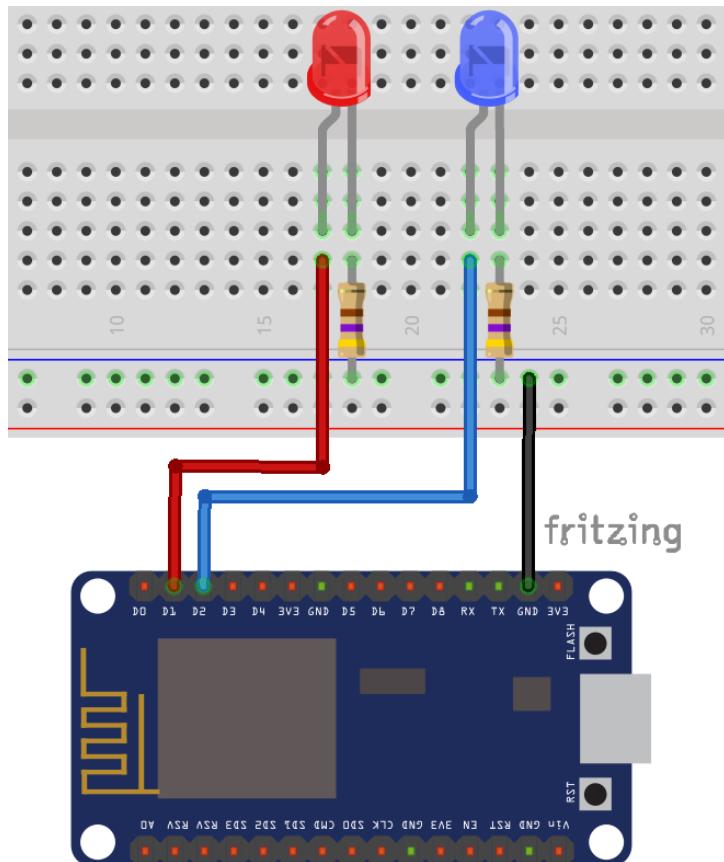
```

Save&Compile Save&Compile&Run... Save As init
View on ESP View on ESP Save&Compile
Send to ESP Run Upload ...
node.restart() Send CR LF Donate

Note: if your IP address doesn't appear in your output window you can send the command `print(wifi.sta.getip())` to print your ESP8266 IP.

Final Circuit

After uploading your code to your ESP8266, follow the next schematic diagram (you can use 270 Ohm resistors for the LEDs).

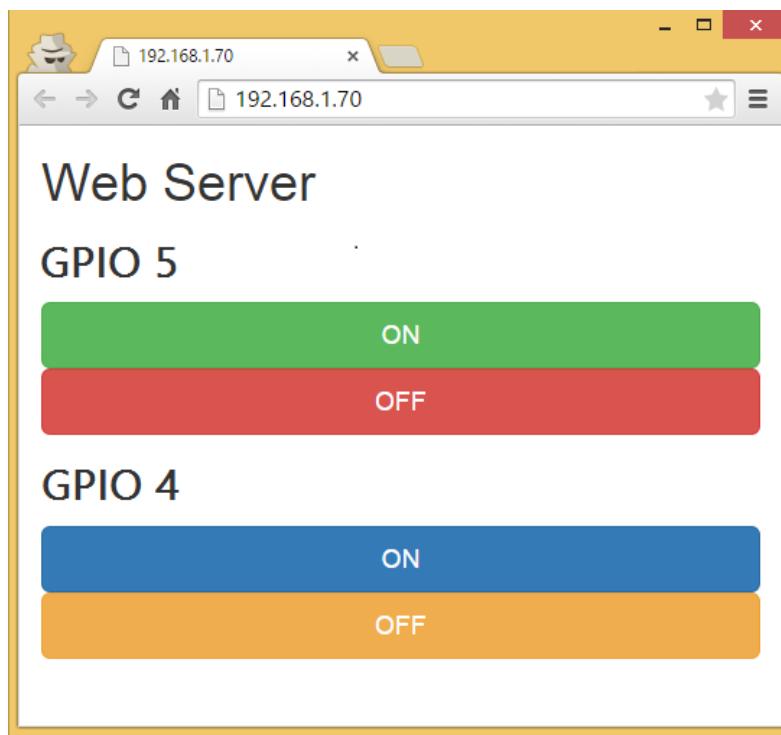


Accessing Your Web Server

Follow the next instructions before accessing your web server:

1. Restart your ESP8266
2. Open a browser
3. Type the IP address that you've previously saved (in my case: 192.168.1.70) in the URL bar

This web page loads:



Note: to access your web server, you need to be connected to the same router that your ESP8266 is.

That was fun! Having a \$4 WiFi module that can act as a web server and serves mobile responsive web pages is pretty amazing.

Making Your Web Server Password Protected

At this point your web server is running on your local network and anyone that is connected to your router can type the IP address of your ESP into their browser and access your web server.

To make your web server more secure let's add an authentication mechanism. After implementing this feature, when someone tries to access your web server they need to enter a username and a password.

You only need to add those 6 lines of code below to your existing web server:

```
local _, _, auth = string.find(request, "%cAuthorization: Basic ([%w=%+/]+)");--Authorization:  
if (auth == nil or auth ~= "dXNlcjpwYXNz")then --user:pass  
    client:send("HTTP/1.0 401 Authorization Required\r\nWWW-Authenticate: Basic realm=\"ESP8266  
    client:close();  
    return;  
end
```

Go to the following link and download the web server script with the authentication mechanism:

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART2_LUA/Unit5_2_web_server_authentication.lua

Encoding Your Username and Password

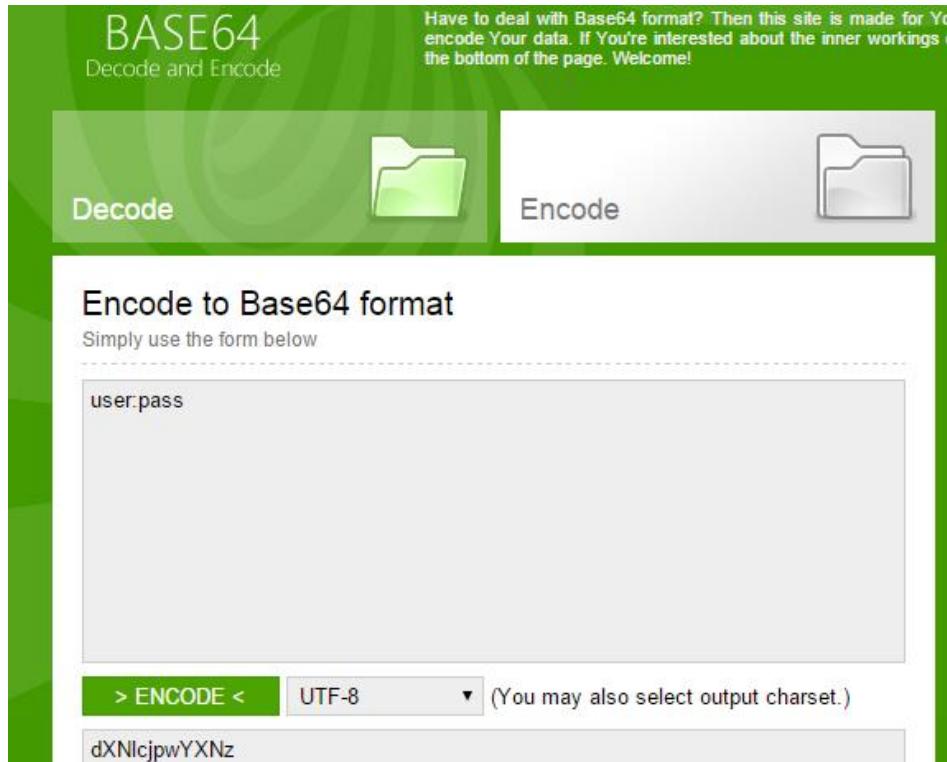
At this point if you upload the code I've mentioned in the preceding section, your username is *user* and your password is *pass*. I'm sure you want to change and customize this example with your own credentials.

Go to the following URL: <https://www.base64encode.org>. In the first field, type the following:

your_username:your_password

Note: you need to type the ":" between your username and your password.

In my example, I've entered *user:pass* (as shown in the figure below):



Press the green "Encode" button to generate your base64 encoded string. In my example is *dXNlcjpwYXNz*.

Copy your string and replace it in this line of the Lua script that you downloaded.

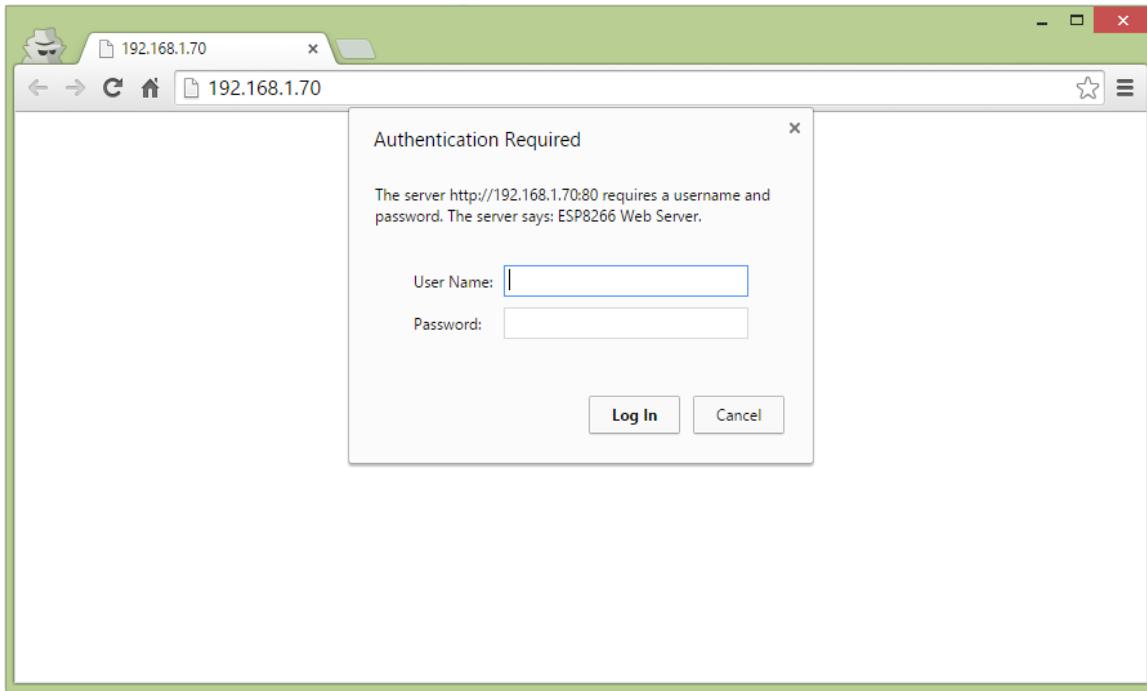
```
if (auth == nil or auth ~= "dXNlcjpwYXNz")then --user:pass
```

Uploading New Web Server Script

Now that you have your code ready, you need to upload your updated Lua script to the ESP8266.

After a successful upload, access your ESP web server by entering your ESP IP address in your browser.

It should require your username and password to access your web server. This is what you should see:



Taking It Further

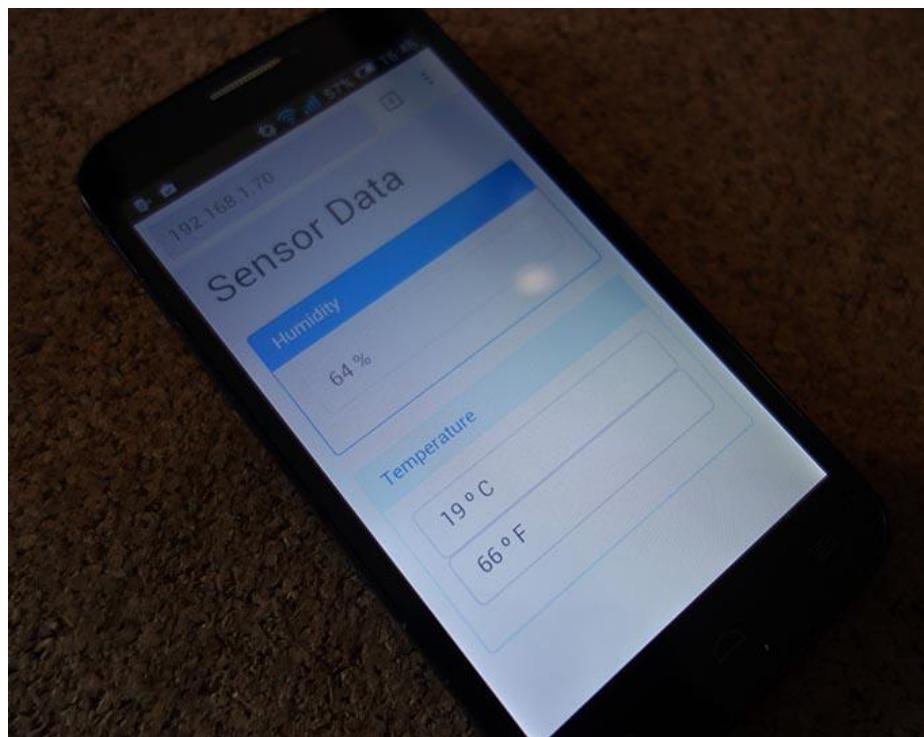
I hope you're happy about seeing that LED turning on and off! I know that it's just an LED, but creating a web server just like you did is an extremely useful concept.

Controlling some house appliances may be more exciting than lighting up an LED. You can easily and immediately replace the LED with a new component that allows you to control any device that connects directly to the sockets on the wall like a [relay module](#), for example.

Unit 6: Displaying Temperature and Humidity on a Web Page

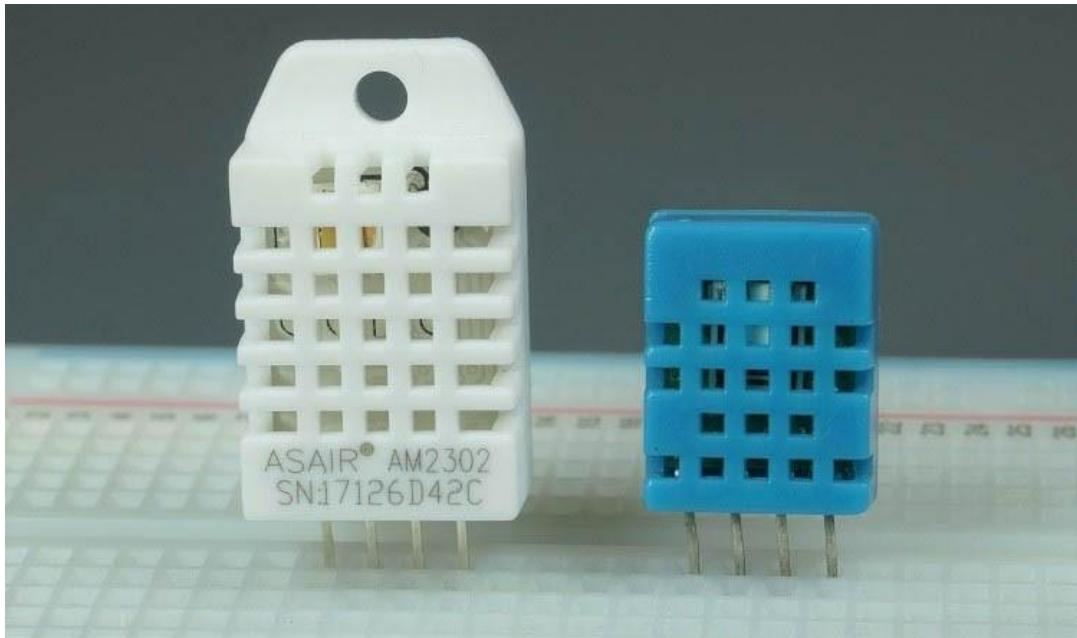
In this Unit you're going to create a web server with your ESP8266 that can be accessed with any device that has a browser. This project serves a web page with the current temperature and humidity. You'll use a DHT11 sensor to measure the temperature and humidity.

Here's the final project result:



Introducing the DHT11 and DHT22 Sensors

The DHT11 and DHT22 sensors are used to measure temperature and relative humidity. These are very popular among makers and electronics hobbyists.



These sensors contain a chip that does analog to digital conversion and spit out a digital signal with the temperature and humidity. This makes them very easy to use with any microcontroller.

DHT11 vs DHT22

The DHT11 and DHT22 are very similar, but differ in their specifications. The following table compares some of the most important specifications of the DHT11 and DHT22 temperature and humidity sensors. For a more in-depth analysis of these sensors, please check the sensors' datasheet.

	DHT11	DHT22
Temperature range	0 to 50 °C (+/- 2 °C)	-40 to 80 °C (+/- 0.5°C)
Humidity range	20 to 90% (+/-5%)	0 to 100% (+/-2%)
Resolution	Humidity: 1% Temperature: 1°C	Humidity: 1% Temperature: 1°C
Operating voltage	3 – 5.5V DC	3 – 6V DC

Current supply	0.5 – 2.5 mA	1 – 1.5 mA
Sampling period	1 second	2 seconds
Price	\$1 to \$5	\$4 to \$10
Where to buy?	Check prices	Check prices

The DHT22 sensor has a better resolution and a wider temperature and humidity measurement range. However, it is a bit more expensive, and you can only request readings with 2 seconds interval. The DHT11 has a smaller range and it's less accurate. However, you can request sensor readings every second. It's also a bit cheaper. Despite their differences, they work in a similar way, and you can use the same code to read temperature and humidity. You just need to select in the code the sensor type you're using.

DHT Pinout

DHT sensors have four pins as shown in the following figure. However, if you get your DHT sensor in a breakout board, it comes with only three pins and with an internal pull-up resistor on pin 2.



The following table shows the DHT22/DHT11 pinout. When the sensor is facing you, pin numbering starts at 1 from left to right

DHT pin	Connect to
1	3.3V
2	Any digital GPIO; also connect a 4.7k Ohm pull-up resistor (or similar)
3	Don't connect
4	GND

Writing Your init.lua Script

You can download the Lua script to create a web page that displays the temperature and humidity from a DHT11 sensor below:

SOURCE CODE

```
https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART2 LUA/Unit6\_DHT11\_module.lua
```

The first snippet of code starts by setting the mode of your ESP to a station. Then it configures the module with your own credentials (network name and password). You need to replace that second line with your credentials, so that your ESP8266 can connect to your network.

It uses a function called *tmr.delay(5000)* to delay 5 seconds your code. That gives time for the module to connect to the network.

The *print()* function in line 4 prints your ESP8266 IP address in the output window of the ESPlorer IDE (you need that IP to access your web server).

```
1 wifi.setmode(wifi.STATION)
2 wifi.sta.config("YOUR_NETWORK_NAME", "YOUR_NETWORK_PASSWORD")
3 print(wifi.sta.getip())
4 tmr.delay(5000)
```

Next, you define where the DHT module is connected, in this case, *pin* = 1 which refers to GPIO 5. You also create three other variables to store data.

```
6 pin = 1
7 bimb = 1
8 fare = 0
9 fare_dec = 0
```

Then, you create a function called *ReadDHT11()* that does exactly what it sounds like. It reads the humidity and temperature data from your sensor and stores it in the right variables. It also prints that data in your ESPlorer IDE output window. You can delete lines 30 to 32. They are just for debugging purposes.

```
11 --Read DHT Sensor
12 function ReadDHT11()
13     status, temp, humi, temp_dec, humi_dec = dht.read(pin)
14     if status == dht.OK then
15         print(string.format("DHT Temperature:%d.%03d;Humidity:%d.%03d\r\n",
16                         math.floor(temp),
17                         temp_dec,
18                         math.floor(humi),
19                         humi_dec
20                     ))
21         fare = (9 * math.floor(temp) / 5) + 32
22         fare_dec = (9 * temp / 5) % 10
23     elseif status == dht.ERROR_CHECKSUM then
24         print( "DHT Checksum error." )
25     elseif status == dht.ERROR_TIMEOUT then
26         print( "DHT timed out." )
27     end
28 end
29 ReadDHT11()
```

You also need to create a *tmr.alarm()* function that runs the *ReadDHT11()* function every 5 seconds. Remember, that function retrieves the current humidity and temperature readings every time it runs.

Next, in line 41, you create the web server on port 80. In line 46, you call the function `conn:send()`.

```
39 tmr.alarm(1,5000, 1, function() ReadDHT11() bimb=bimb+1 if bimb==5
40
41     srv=net.createServer(net.TCP) srv:listen(80,function(conn)
42         conn:on("receive",function(conn,payload)
43             --print(payload) -- for debugging only
44             --generates HTML web site
45             conn:send()
46             conn:on("sent",function(conn) conn:close()end)
47         end)
48     end)
```

Inside the function `conn:send()` you add your HTML page (see code below). It's just a basic web page that uses the Bootstrap framework. Learn more about the Bootstrap framework: <http://getbootstrap.com>.

```
conn:send("HTTP/1.1 200 OK\r\nConnection: keep-alive\r\nCache-Control: private, no-store\r\n\r\n\
<!DOCTYPE HTML>\ \
<html><head><meta charset="utf-8"><meta name="viewport" content="width=device-width, initial-scale=1"></head>\ \
<meta http-equiv="X-UA-Compatible" content="IE=edge">\ \
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">\ \
<meta http-equiv="refresh" content="6">\ \
</head><div class="container">\ \
<h1>Sensor Data</h1><br><div class="row">\ \
<div class="col-md-4"><div class="panel panel-primary"><div class="panel-heading"><h3 class="panel-title">Humidity</h3>\ \
</div><div class="panel-body">\ \
<div class="form-group form-group-lg"><input type="text" class="form-control" value="'.math.floor(humi)..'.'..humi_dec..' %>\ \
</div></div></div>\ \
<div class="col-md-4"><div class="panel panel-info"><div class="panel-heading"><h3 class="panel-title">Temperature</h3>\ \
</div><div class="panel-body">\ \
<div class="form-group form-group-lg"><input type="text" class="form-control" value="'.math.floor(temp)..'.'..temp_dec..' deg F">\ \
<input type="text" class="form-control" value="'.fare..'.'..fare_dec..' deg F">\ \
</div></div></div></div></div>\ \
</html>")
```

The figure above displays temperature and humidity in a nice looking mobile responsive web page.

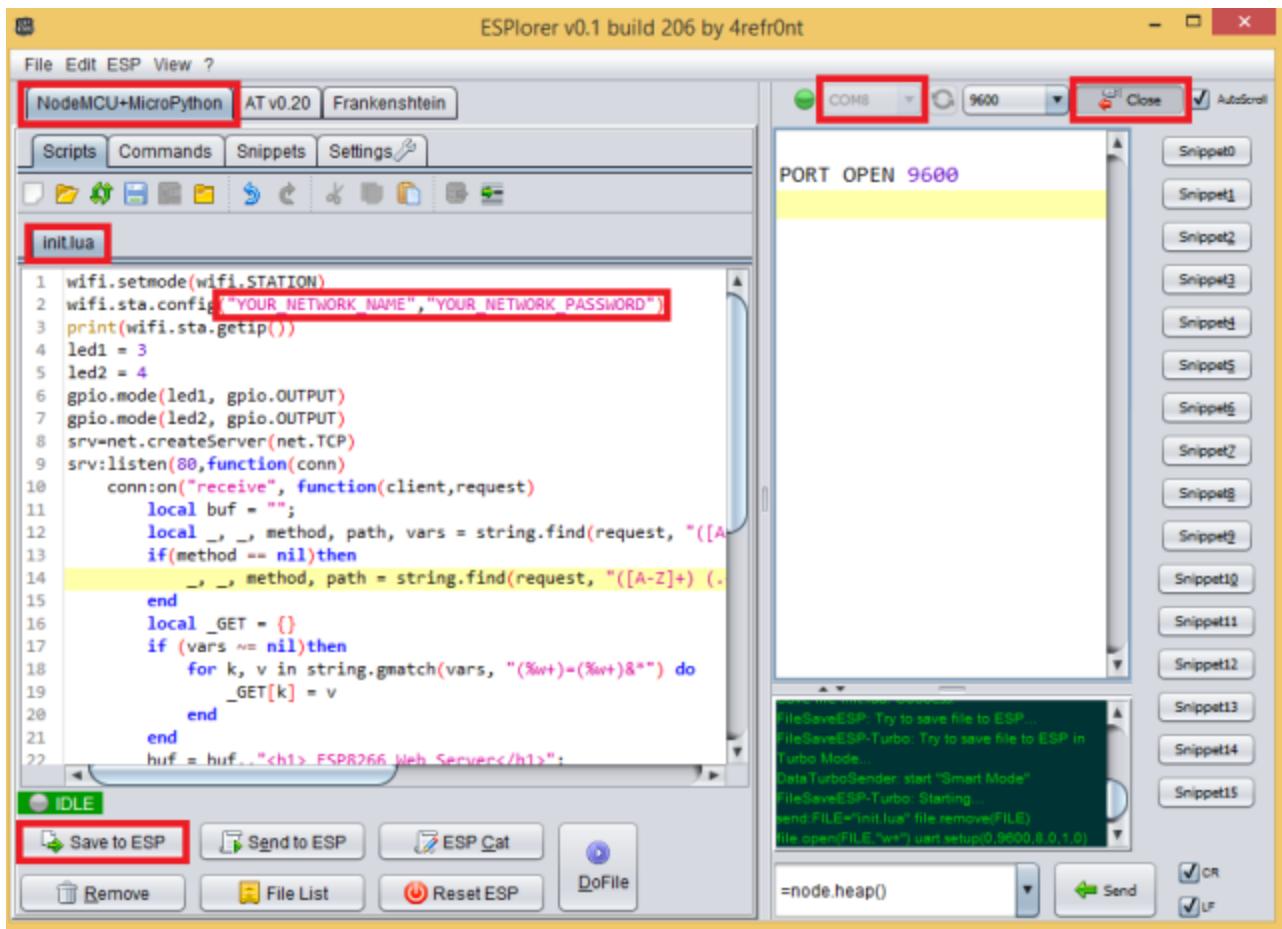
Uploading init.lua

Follow these instructions to upload a Lua script:

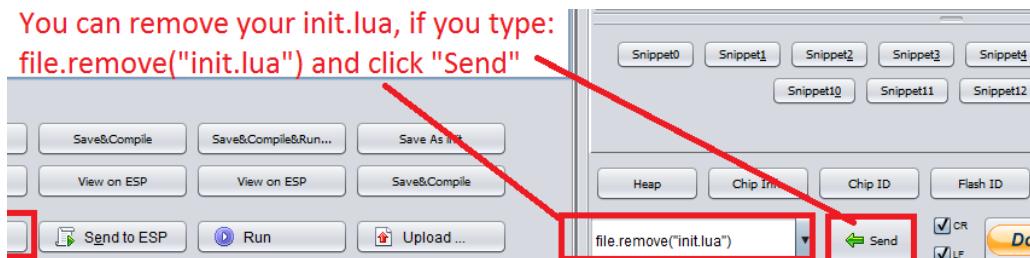
1. Connect your ESP8266-12E that has built-in programmer to your computer
2. Select your ESP8266-12E port
3. Press Open/Close

4. Select NodeMCU+MicroPython tab
5. Create a new file called init.lua
6. Press Save to ESP

Everything that you need to worry about or change is highlighted in red box.



In your output window, it starts showing exactly which commands are being sent to your ESP8266 and it should look similar to the figure below.



Note: you can easily delete the “init.lua” file from the ESP. Simply type `file.remove("init.lua")` and press the button “Send” (see figure above). Or you can type the command `file.format()` to remove all the files saved in your ESP8266.

ESP8266 IP Address

After uploading your web server Lua script to your ESP8266, in your output window you’re going to see 3 IP addresses.

The IP that you want is the first one, in my case it’s: 192.168.1.70. Your IP should be different, **save your ESP8266 IP** so you can access it later in this Unit.

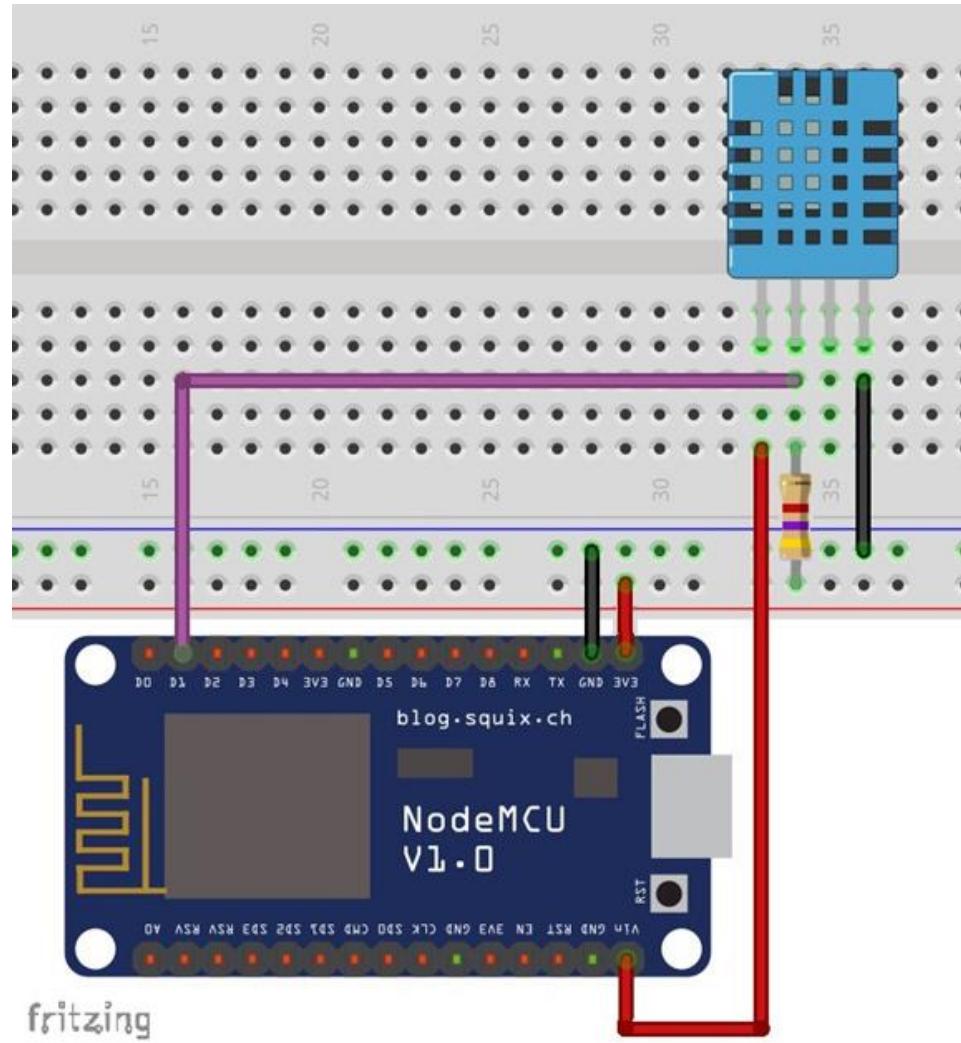
```
f = buf.."<div class='row'>";
f = buf.."<div class='col-md-2'><a href=?pin=ON2? class='btn btn-primary'>ON</a>
f = buf.."<div class='col-md-2'><a href=?pin=OFF2? class='btn btn-primary'>OFF</a>
f = buf.."</div></div>";
cal_on,_off = "", ""
if _GET.pin == "ON1" then
  gpio.write(led1, gpio.HIGH);
elseif _GET.pin == "OFF1" then
  gpio.write(led1, gpio.LOW);
elseif _GET.pin == "ON2" then
  gpio.write(led2, gpio.HIGH);
elseif _GET.pin == "OFF2" then
  gpio.write(led2, gpio.LOW);
end
w\LL      enuJJ,
> w([[ client:send(buf);]]));
> w([[ client:close();]]));
> w([[ collectgarbage();]]);
> w([[ end]]]);
> w([[end]]));
> file.close();
> dofile("init.lua");
192.168.1.70  255.255.255.0  192.168.1.254
```

Note: if your IP address doesn’t appear in your output window you can send the command `print(wifi.sta.getip())` to print your ESP8266 IP.

Final Circuit

After uploading your code to your ESP8266, follow the next schematic diagram. You need to use a 4700 ohm resistor or higher with your DHT11 sensor as pull-up resistor.

Warning: the 3.3V power supply should supply 250mA to ensure that your circuit works.

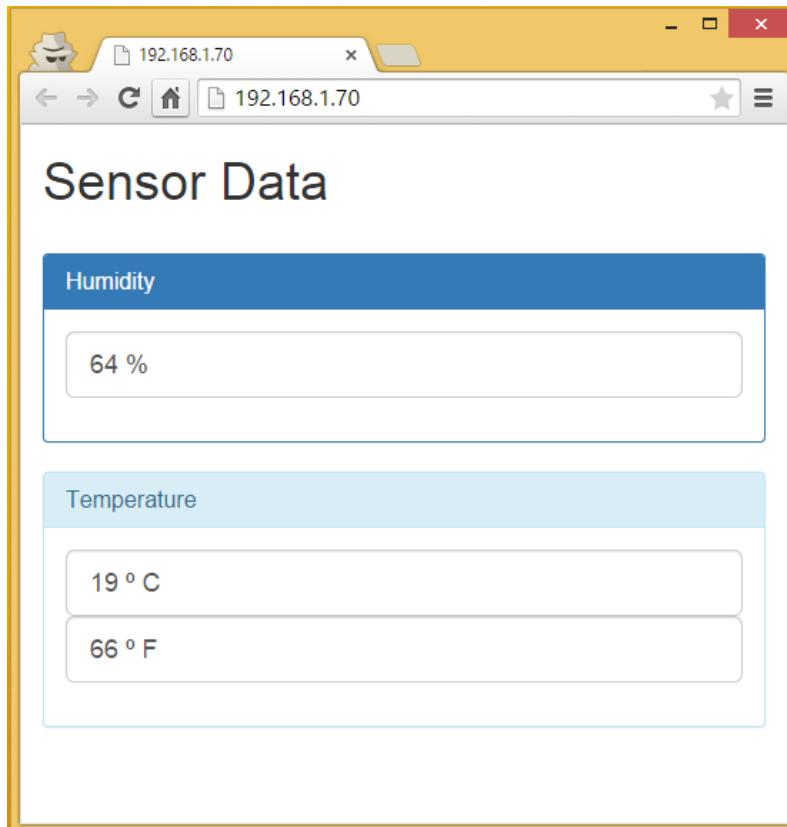


Accessing Your Web Server

Follow the next instructions before accessing your web server:

1. Restart your ESP8266
 2. Open a browser
 3. Type the IP address that you've previously saved (in my case: 192.168.1.70) in the URL bar

A web page like the one below loads:



Pretty cool, huh?! Feel free to modify or add your own twist to this code. And remember this page updates automatically every 6 seconds.

Additionally, this page is mobile responsive so it looks great in any device.

Taking It Further

The ESP-12E comes with just one analog pin which is very limiting if you want to add analog sensors or modules to your project...

So, what can you do to overcome that problem? I have an idea.

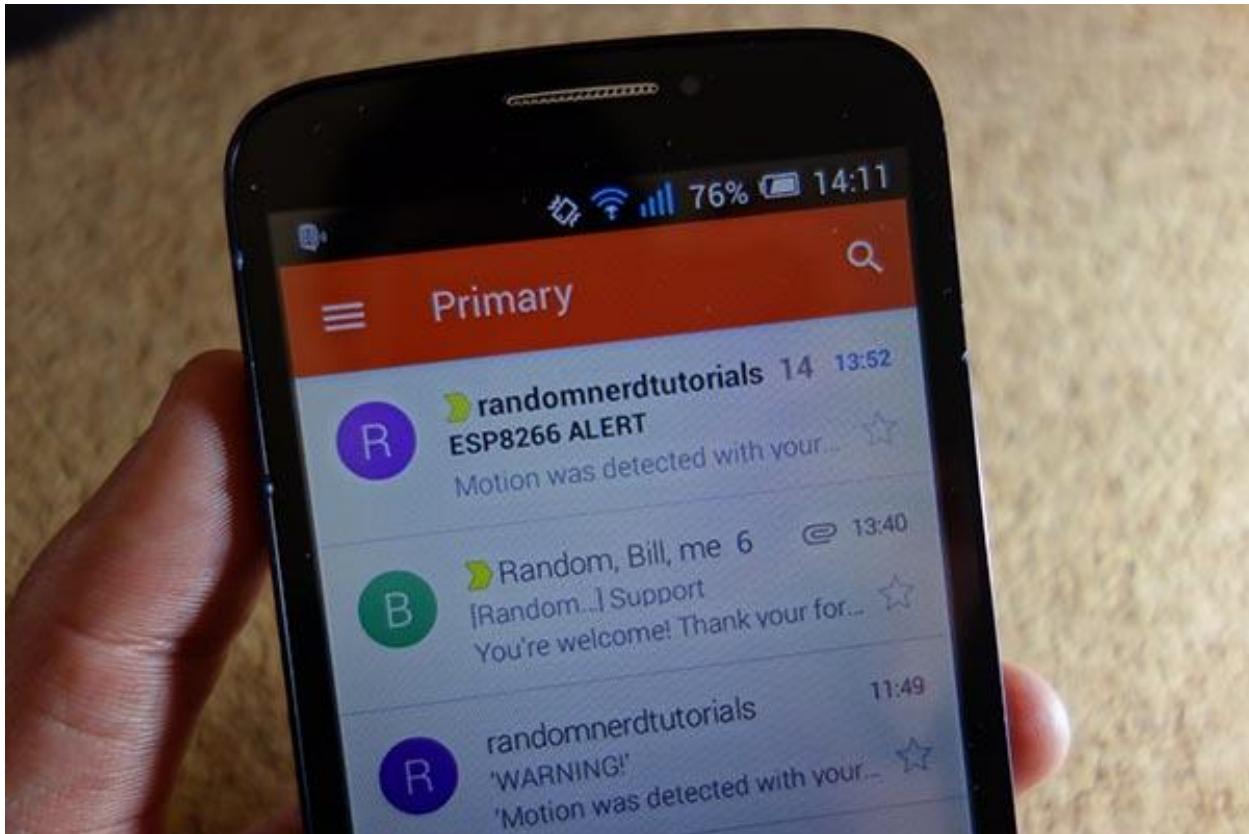
You know how to establish a serial communication with the ESP8266. You can attach a bunch of sensors to your Arduino and connect it to your ESP. Your Arduino does all the work and sends the readings to your ESP via serial. Finally, your ESP grabs all the data and displays it in your neat web server!

Here's a good tutorial on how to send serial data from an Arduino to an ESP:

<http://randomnerdtutorials.com/sending-data-from-an-arduino-to-the-esp8266-via-serial/>

You can re-purpose some of the pieces in that tutorial to read your sensors with an Arduino.

Unit 7: Email Notifier with ESP8266 and PIR Motion Sensor



In this Unit you're going to create an email notifier with an ESP8266 and a cheap PIR Motion sensor. You can set this project in a strategic place and when someone walks by, it sends you an email alert. We'll basically show how to build a home surveillance system for \$6.

IFTTT

For this project we're going to use a free service called **IFTTT** that stands for If This Than That. This service is used to automate a wide variety of tasks online. In this case, we want to send an email when the ESP8266 detects motion with a PIR motion sensor.

Type in your browser <https://ifttt.com> and click the “**Get started**” button in the middle of the page. Complete the form with your details and create your account.



Every thing works better together

Get started

or

G Continue with Google

F Continue with Facebook

Creating the Applet

Go to this URL to start creating your own applet: <https://ifttt.com/create>

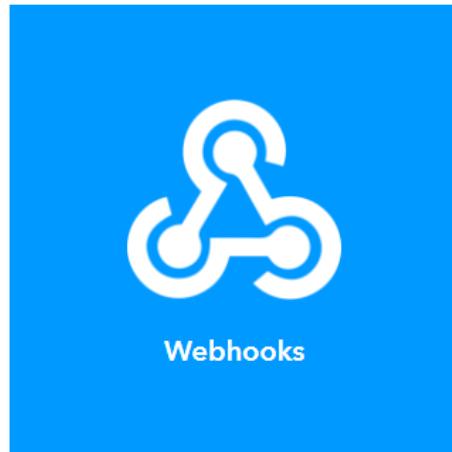
If + This Then That

Click the “+” sign. Search for “**webhooks**” in the search bar.

Choose a service

Step 1 of 6

Q webhooks X



Select the “Receive a web request” option.



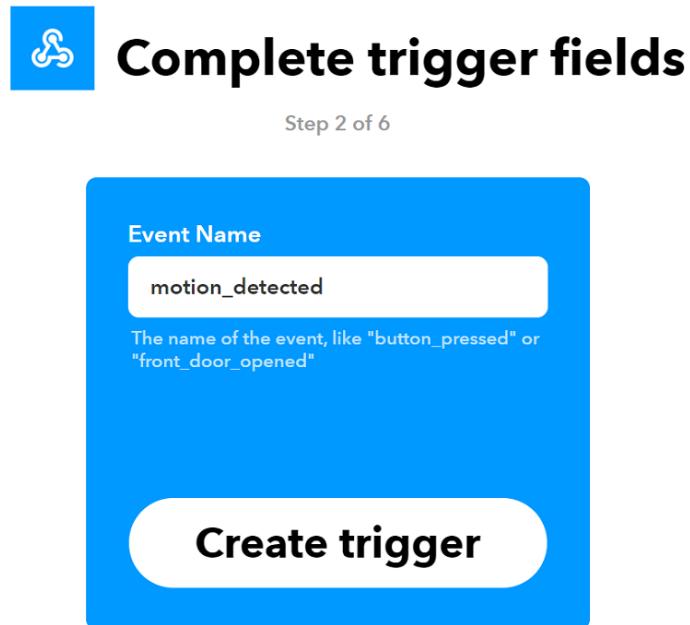
Choose trigger

Step 2 of 6

Receive a web request

This trigger fires every time the Maker service receives a web request to notify it of an event. For information on triggering events, go to your Maker service settings and then the listed URL (web) or tap your username (mobile)

Give a name to the event, for example “motion_detected” and click the **Create trigger** button.

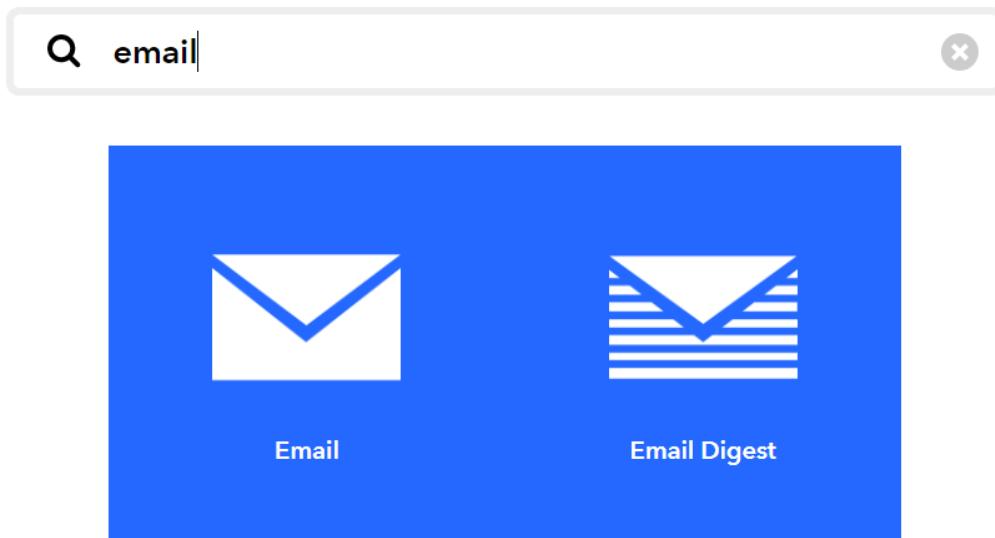


With the Webhooks service you can trigger an action when you make an HTTP request to a specific URL. That's what we're going to do.

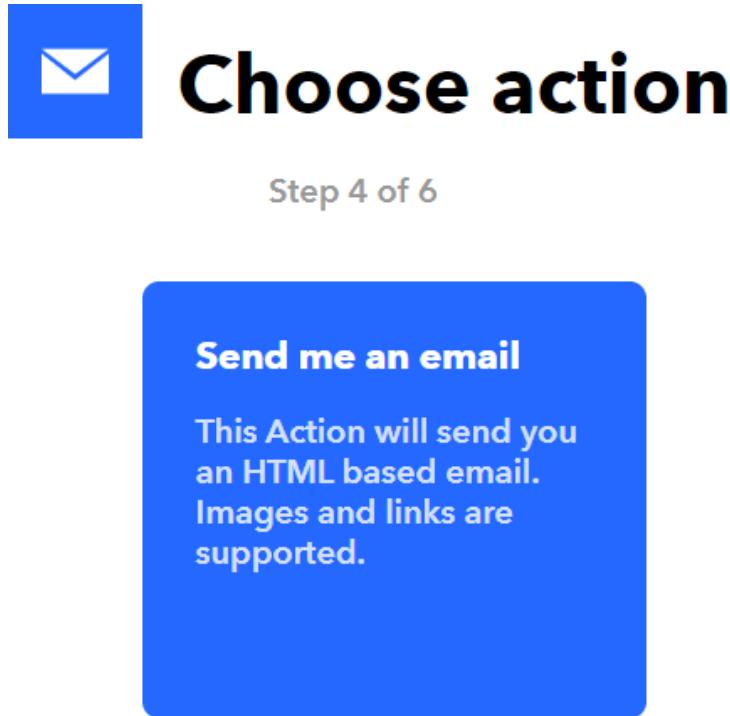
Now press the “+” next to the “**That**” word, search and select the **Email** service.

Choose action service

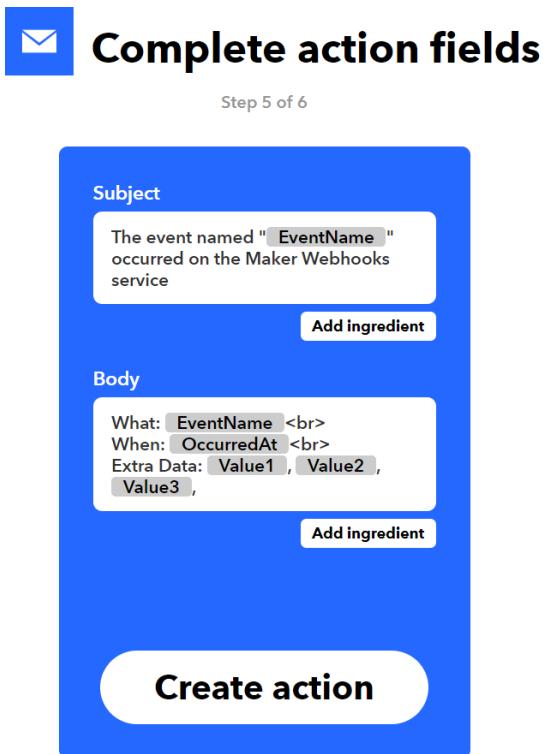
Step 3 of 6



Choose the “**Send email**” option.



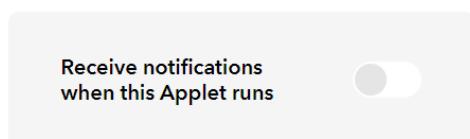
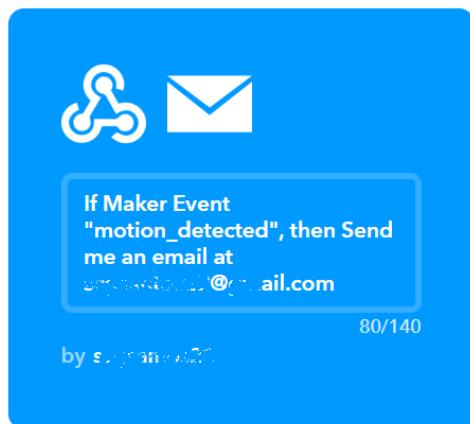
Then, click the “**Create action**” button. You can leave the fields as default.



Then, click “**Finish**” to create your applet.

Review and finish

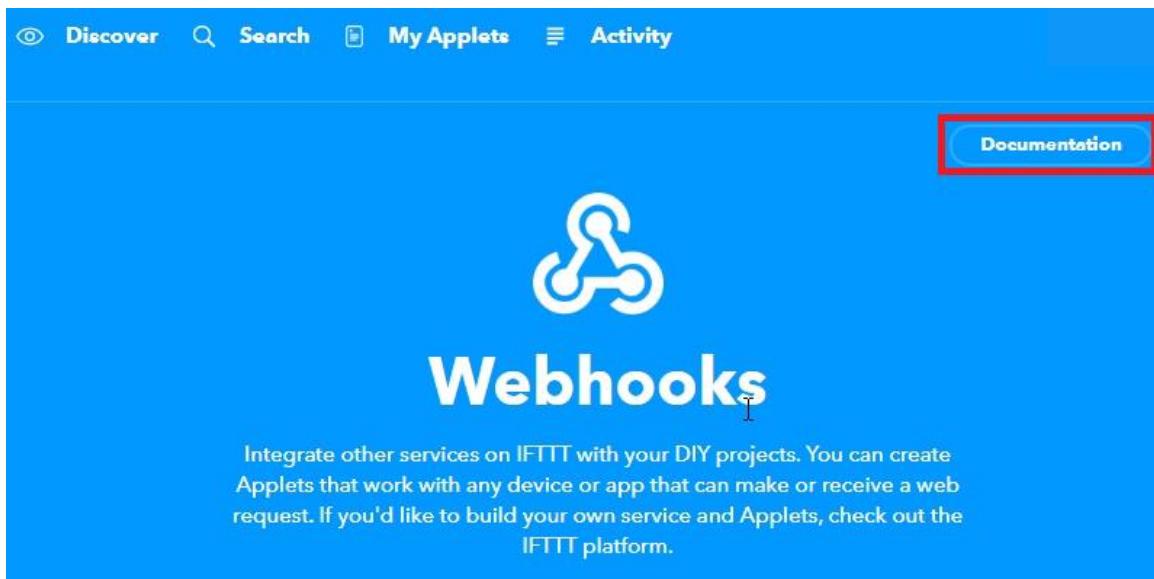
Step 6 of 6



Finish

Testing the Applet

Go to the following URL: https://ifttt.com/maker_webhooks, then click the "Documentation" tab.



Here you can find your unique **API KEY** that you must keep private. Type in the event name, **motion_detected**. Your final URL should appear in the bottom of the web page. Copy that URL.



A screenshot of a web browser showing the IFTTT API key generation page. The URL is https://maker.ifttt.com/trigger/motion_detected/with/key/bIlr99vLOPj_RPdKf... . The key itself is highlighted with a red box. A blue icon of three interconnected circles is visible in the top left corner.

Your key is: bIlr99vLOPj_RPdKf...

◀ Back to service

To trigger an Event

Make a POST or GET web request to:

```
https://maker.ifttt.com/trigger/motion_detected/with/key/bIlr99vLOPj_RPdKf...
```

With an optional JSON body of:

```
{ "value1" : "_____", "value2" : "_____", "value3" : "_____"}
```

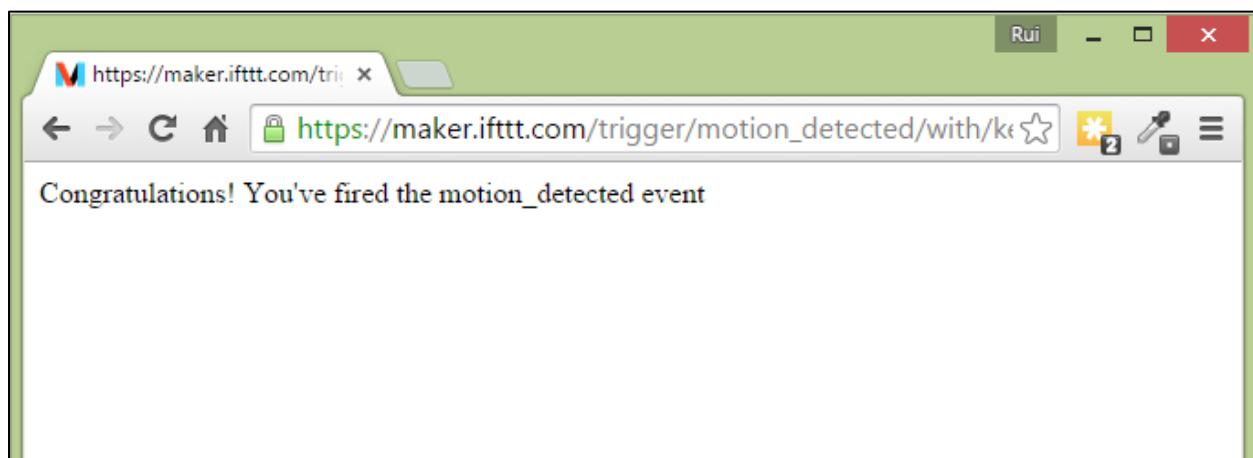
The data is completely optional, and you can also pass `value1`, `value2`, and `value3` as query parameters or fo passed on to the Action in your Recipe.

You can also try it with `curl` from a command line.

```
curl -X POST https://maker.ifttt.com/trigger/motion_detected/with/key/bIlr99vLOPj_RPdKf...
```

Test It

Open a new tab in your browser, paste that URL and hit enter or simply click the “**Test It**” button. You should see this message saying “Congratulations! (...)”.



Open your email and the new message should be there.

The event named "motion_detected" occurred on the Maker Webhooks service

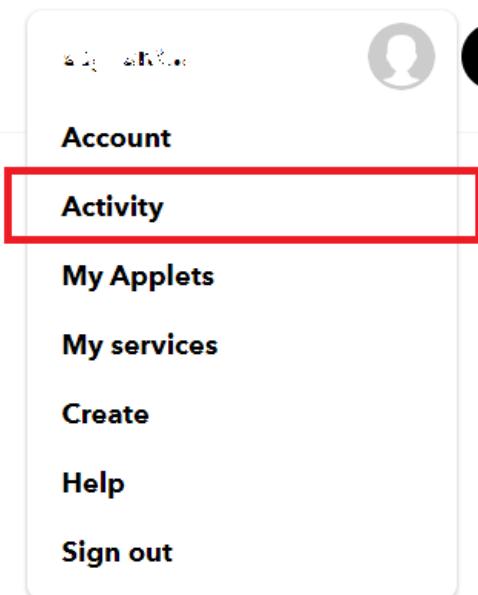
 **Webhooks via IFTTT** <action@ifttt.com> [Angular subscrição](#)
para mim ▾

What: motion_detected
When: November 5, 2019 at 11:24AM
Extra Data: , , ,


If Maker Event "motion_detected", then
Send me an email at
action@ifttt@gmail.com

[Unsubscribe](#) from these notifications or sign in to manage your Email service.

In case the email didn't arrive after a few seconds, we recommend double-checking the URL and see if you're using the correct event name, both in your Applet and in your URL. Or, in your account, go to "**Activity**" and try to understand what might have happened.



If everything went out smoothly, save your unique URL in a Notepad, because you'll need it later in this project. Basically, you'll need to make a request on that URL with the ESP8266 when motion is detected.

Writing Your init.lua Script

You can download the Lua script to create your email notifier project below:

SOURCE CODE

```
https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART2\_LUA/Unit7\_email\_notifier\_ifttt.lua
```

The next snippet of code starts by setting the mode of your ESP8266 to a station. Then, you configure your ESP8266 with your own credentials (network name and password). You need to replace that second line with your credentials, so your ESP8266 can connect to your network.

Create one variable *pin* = 2 which refers to GPIO 4 and define it as *INTERRUPT*. That's where you're going to attach your PIR Motion sensor.

```
1 wifi.setmode(wifi.STATION)
2 wifi.sta.config("YOUR_NETWORK_NAME", "YOUR_NETWORK_PASSWORD")
3 pin = 2
4 gpio.mode(pin, gpio.INT)
```

For this project we need to use an *INTERRUPT* that occurs when the PIR Motion sensor goes from LOW to HIGH, so we use the event “*up*” in the *gpio.trig(pin, 'up', onChange)* function. So when motion is detected it executes the function called *onChange()* which creates a TCP connections to IFTTT and makes your HTTP POST request.

```

6  function onChange ()
7      -- A simple http client
8      print('Motion Detected')
9      conn = nil
10     conn=net.createConnection(net.TCP, 0)
11     conn:on("receive", function(conn, payload) end)
12     conn:connect(80,"maker.ifttt.com")
13     conn:on("connection", function(conn, payload)
14         conn:send("POST /trigger/motion_detected/with/key/YOUR_API_KEY HTTP/1.1\r\n"
15         conn:close()
16         print('Email Sent')
17     end
18     gpio.trig(pin, 'up', onChange)

```

You have to replace the line 14 with **your URL** from <https://ifttt.com/maker>.

Replace line 14:

```

conn:send("POST      /trigger/motion_detected/with/key/YOUR_API_KEY
HTTP/1.1\r\nHost:      maker.ifttt.com\r\nConnection:      keep-
alive\r\nAccept:  */*\r\n\r\n") end)

```

With your API KEY:

```

conn:send("POST      /trigger/motion_detected/with/key/b6eDdHYb1Ev2Sy32qLwe
HTTP/1.1\r\nHost:      maker.ifttt.com\r\nConnection:      keep-
alive\r\nAccept:  */*\r\n\r\n") end)

```

Uploading init.lua

Follow these instructions to upload a Lua script:

1. Connect your ESP8266-12E that has built-in programmer to your computer
2. Select your ESP8266-12E port
3. Press Open/Close
4. Select NodeMCU+MicroPython tab
5. Create a new file called init.lua
6. Press Save to ESP

Everything that you need to worry about or change is highlighted in red box.

```

1 wifi.setmode(wifi.STATION)
2 wifi.sta.config("YOUR NETWORK NAME", "YOUR NETWORK PASSWORD")
3 print(wifi.sta.getip())
4 led1 = 3
5 led2 = 4
6 gpio.mode(led1, gpio.OUTPUT)
7 gpio.mode(led2, gpio.OUTPUT)
8 srv=net.createServer(net.TCP)
9 srv:listen(80,function(conn)
10     conn:on("receive", function(client,request)
11         local buf = ""
12         local _, _, method, path, vars = string.find(request, "([A-Z]+)([A-Z][a-z]*)([A-Z][a-z]*)")
13         if(method == nil)then
14             _, _, method, path = string.find(request, "([A-Z]+)([A-Z][a-z]*)")
15         end
16         local _GET = {}
17         if (vars ~= nil)then
18             for k, v in string.gmatch(vars, "(%w+)=(%w+)&*") do
19                 _GET[k] = v
20             end
21         end
22         huf = huf..<h1> FSPR266 Web Server</h1>;

```

In your output window, it should start showing exactly which commands are being sent to your ESP8266 and it should look similar to the figure below.

Modifying PIR Motion Sensor to Work at 3.3V

For this project, you're going to use a PIR Motion sensor. This type of sensor is used to detect movement from humans or pets. You can read a guide on my website on [how to use this sensor with an Arduino](#).

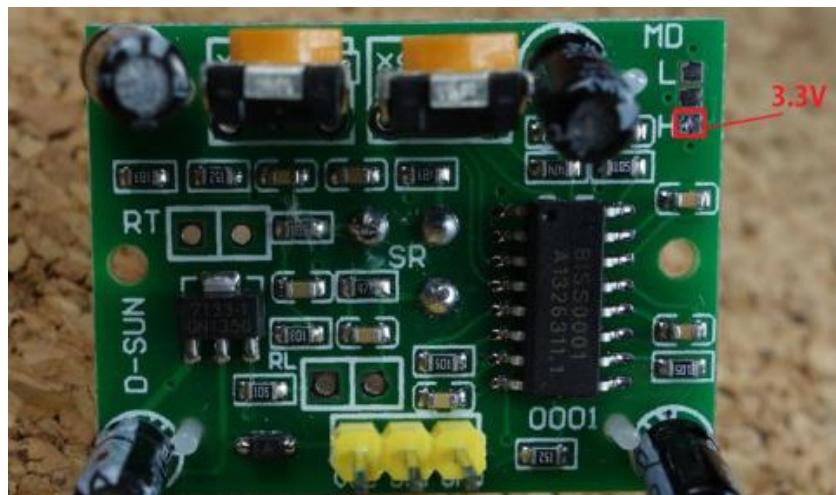


By default, this module runs at 5V, but it has an on board voltage regulator that drops that voltage to 3.3V. With a quick online search I've found a blog [post](#) that explains how you can bypass the voltage regulator and use this module at 3.3V, which is exactly what you need.

You can [click here to find the PIR motion sensor](#) price at different stores.

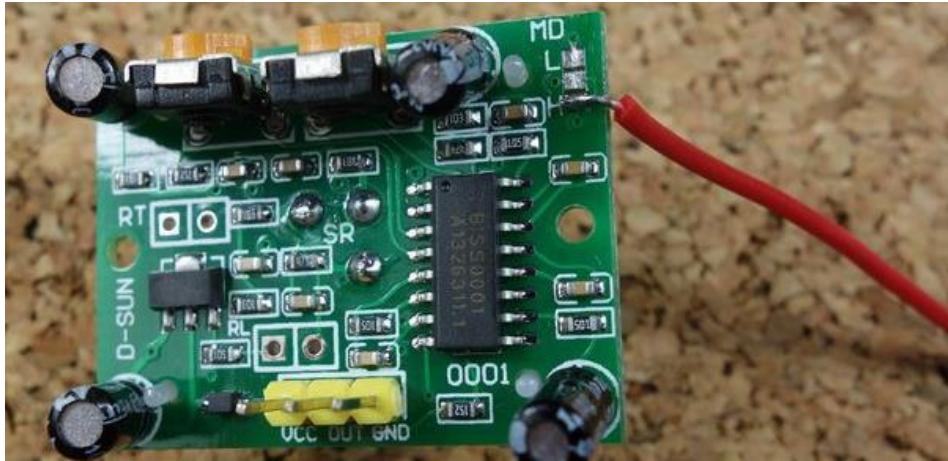
Before soldering

Some of these modules come with pins soldered in that top right corner, so you don't have to solder anything. You simply connect a jumper wire to that pin that is highlighted in red (see figure below). With my particular sensor I had to solder a small wire.



After soldering

Here's how it looks, now if you supply 3.3V through that red wire your module works at 3.3V.

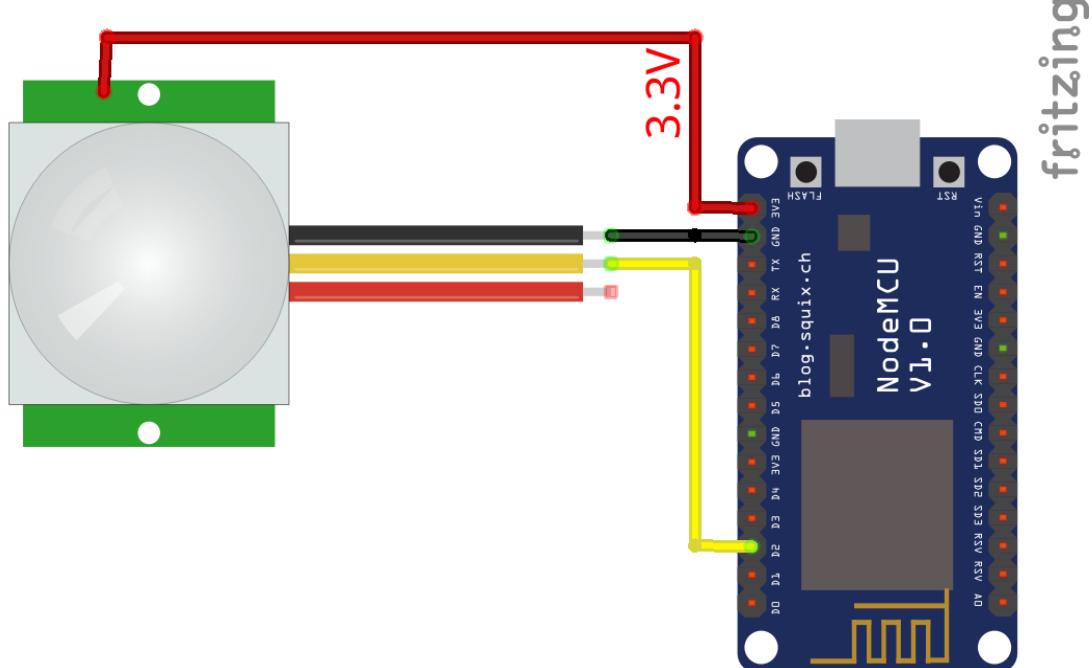


Testing

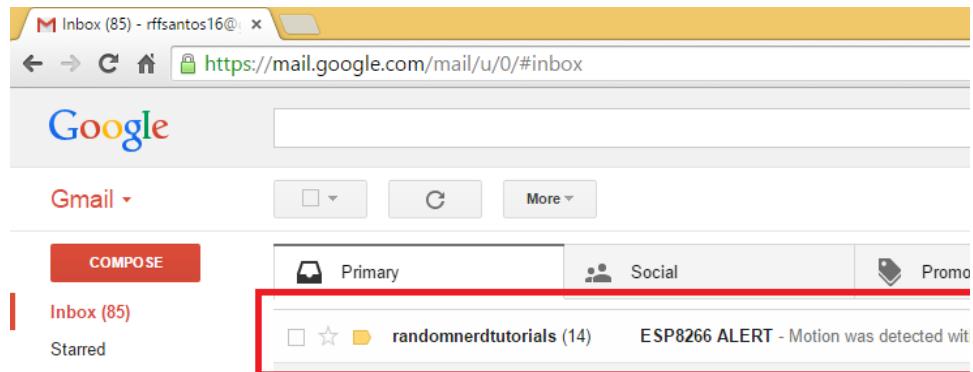
You can test this sensor with an [Arduino](#) using this sketch example.

Final Circuit

This circuit is very quick to assemble, simply follow this schematic diagram:



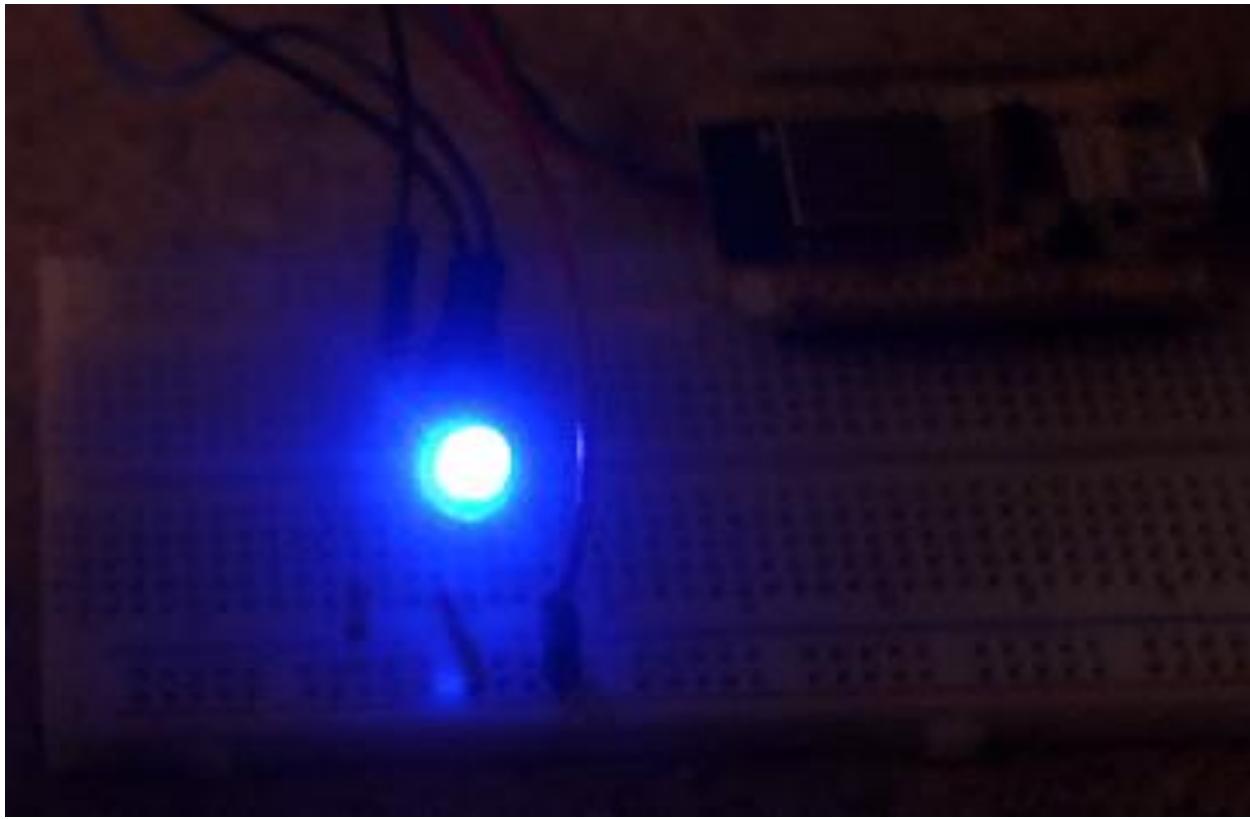
When you move your hand in front of the PIR Motion sensor, or when someone passes in front of your sensor, an email is immediately sent to your inbox! How cool is that?



Taking It Further

The IFTTT platform is very powerful and you can integrate your ESP8266 projects with more than 200 apps. Here's a list of apps that you can connect to your ESP:
<https://ifttt.com/search/services>.

Unit 8: ESP8266 RGB Color Picker



In this Unit, you're going to build a web server with an ESP8266 to remotely control an RGB LED. This project is called ESP8266 RGB Color Picker.

Let's get started!

Parts Required

Here's the hardware that you need to complete this project:

- [ESP8266-12E](#)
- [RGB LED Common Anode](#)
- [3x 470Ω Resistors](#)
- [Breadboard](#)
- [Jumper Wires](#)

Writing Your init.lua Script

Upload the following code into your ESP8266 using the ESPlorer IDE. Your file should be named “init.lua”.

SOURCE CODE

https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART2_LUA/Unit8_ESP_RGB_Color_Picker.lua

Don't forget to add your network name (SSID) and password to the script.

```
wifi.setmode(wifi.STATION)
wifi.sta.config("YOUR_NETWORK_NAME", "YOUR_NETWORK_PASSWORD")

print(wifi.sta.getip())

function led(r, g, b)
    pwm.setduty(1, r)
    pwm.setduty(2, g)
    pwm.setduty(3, b)
end

pwm.setup(1, 1000, 1023)
pwm.setup(2, 1000, 1023)
pwm.setup(3, 1000, 1023)
pwm.start(1)
pwm.start(2)
pwm.start(3)

srv=net.createServer(net.TCP)
srv:listen(80,function(conn)
    conn:on("receive", function(client,request)
        local buf = "";
        buf = buf.."HTTP/1.1 200 OK\r\n\r\n"
    end
end)
```

```

    local _, _, method, path, vars = string.find(request, "([A-Z]+)
(.+)?(.+) HTTP");
    if(method == nil)then
        _, _, method, path = string.find(request, "([A-Z]+) (.+) HTTP");
    end
    local _GET = {}
    if (vars ~= nil)then
        for k, v in string.gmatch(vars, "(%w+)=(%w+)&*") do
            _GET[k] = v
        end
    end
    buf = buf.."<!DOCTYPE html><html><head>";
    buf = buf.."<meta charset=\"utf-8\">";
    buf = buf.."<meta http-equiv=\"X-UA-Compatible\""
content=\"IE=edge\">";
    buf = buf.."<meta name=\"viewport\" content=\"width=device-width,
initial-scale=1\">";
    buf = buf.."<link rel=\"stylesheet\""
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
">";
    buf = buf.."<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"><
/script>";
    buf = buf.."<script
src="https://cdnjs.cloudflare.com/ajax/libs/jscolor/2.0.4/jscolor.min.js"><
/script>";
    buf = buf.."</head><body><div class=\"container\"><div
class=\"row\"><h1>ESP Color Picker</h1>";
    buf = buf.."<a type=\"submit\" id=\"change_color\" type=\"button\""
class="btn btn-primary\">Change Color</a> ";
    buf = buf.."<input class=\"jscolor {onFineChange:'update(this)'}\""
id="rgb\"></div></div>";
    buf = buf.."<script>function update(picker)
{document.getElementById('rgb').innerHTML = Math.round(picker.rgb[0]) + ', '
+ Math.round(picker.rgb[1]) + ', ' + Math.round(picker.rgb[2]);";
    buf = buf.."document.getElementById(\"change_color\").href=?r=" +
Math.round(picker.rgb[0]*4.0117) + "&g=" +

```

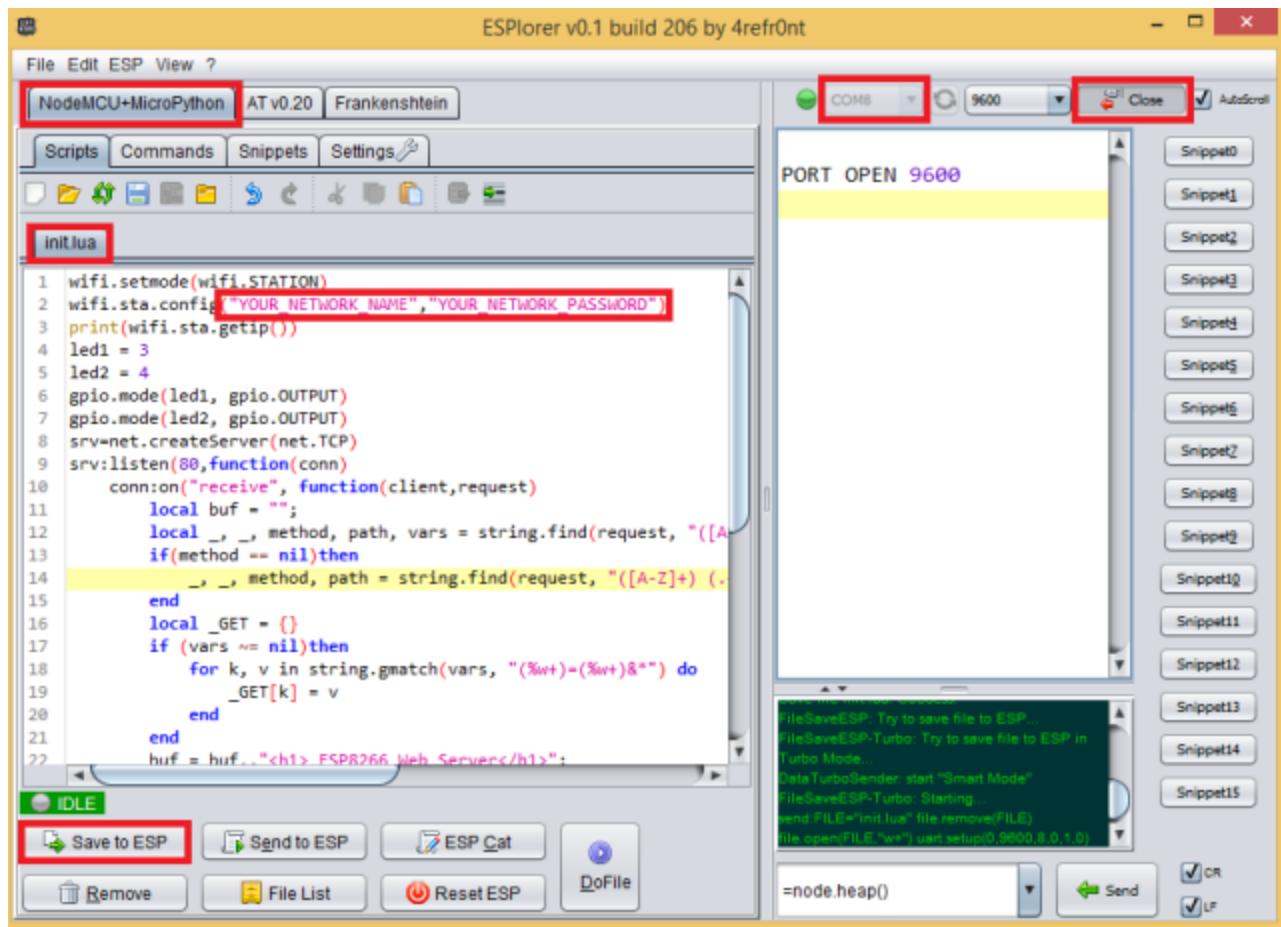
```
Math.round(picker.rgb[1]*4.0117) + \&b=\\" +  
Math.round(picker.rgb[2]*4.0117); }</script></body></html>";  
  
    if(_GET.r or _GET.g or _GET.b) then  
        -- This is for RGB Common Cathode  
        -- led(_GET.r, _GET.g,_GET.b)  
  
        -- This is for RGB Common Anode  
        led(1023-_GET.r, 1023-_GET.g,1023-_GET.b)  
    end  
    client:send(buf);  
    client:close();  
    collectgarbage();  
end)  
end)
```

Uploading init.lua

Follow these instructions to upload a Lua script:

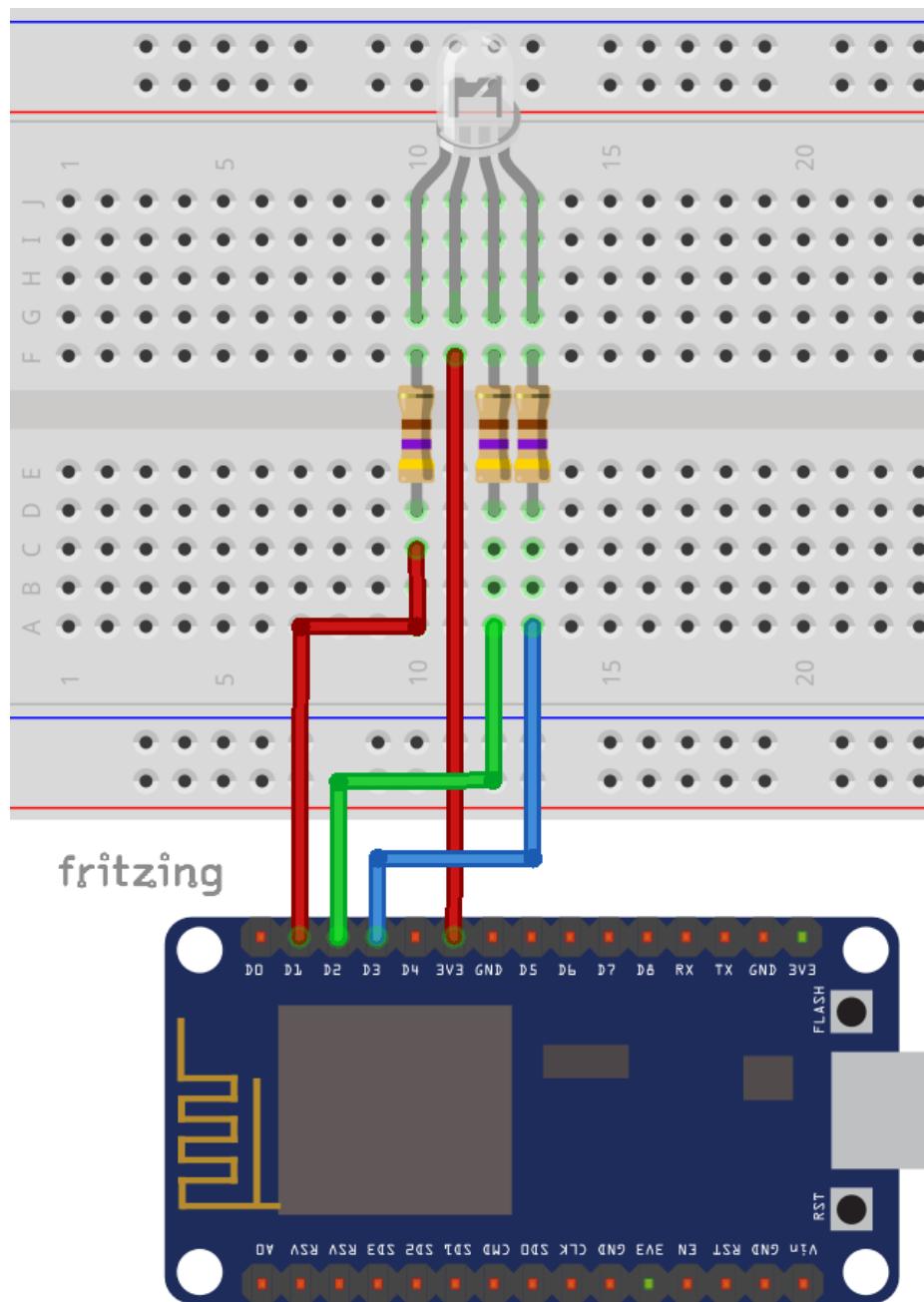
1. Connect your ESP8266-12E that has built-in programmer to your computer
2. Select your ESP8266-12E port
3. Press Open/Close
4. Select NodeMCU+MicroPython tab
5. Create a new file called init.lua
6. Press Save to ESP

Everything that you need to worry about or change is highlighted in red box.



Final Circuit

Now, follow the next schematic diagram to create the circuit for the RGB LED common anode.



Important: if you're using an RGB LED common cathode, you need to connect the longer lead to GND.

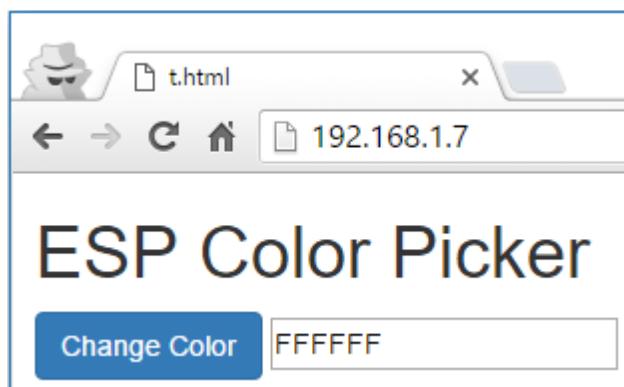
ESP8266 IP Address

When your ESP8266 restarts, it prints the ESP IP address in the serial monitor. Save that IP address, because you'll need it later.

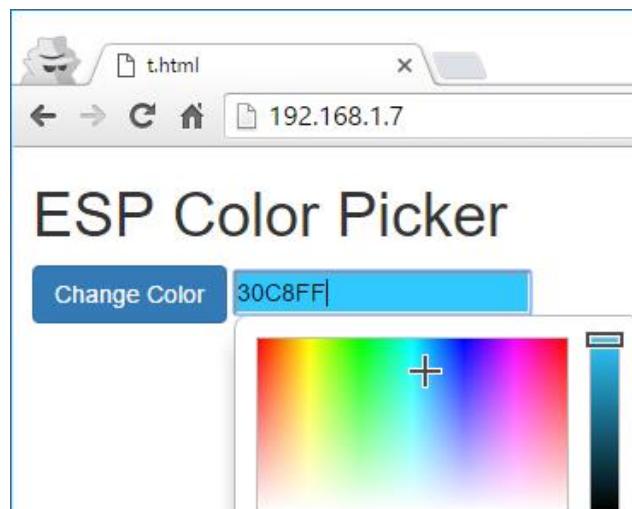
In my case, the ESP IP address is 192.168.1.7. You're all set!

Opening Your Web Server

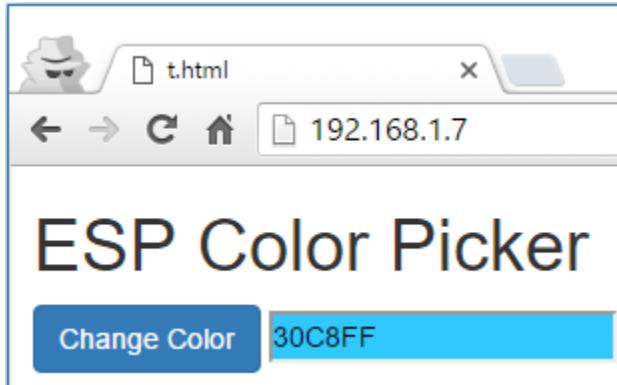
Go to any browser and enter the IP address of your ESP8266. This is what you should see:



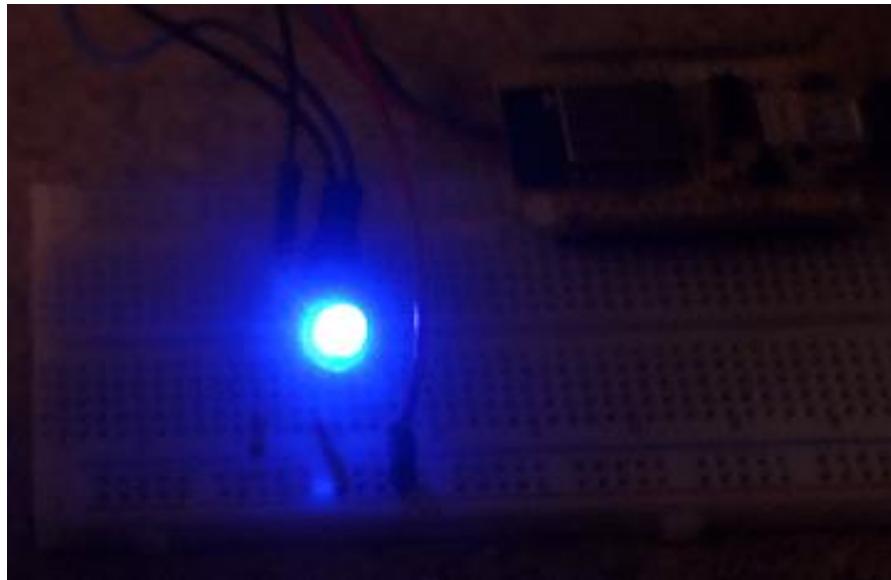
Click the field and a small window opens with a color picker. Simply drag your mouse or finger and select the color for your RGB LED:



Then simply click the "Change Color" button:



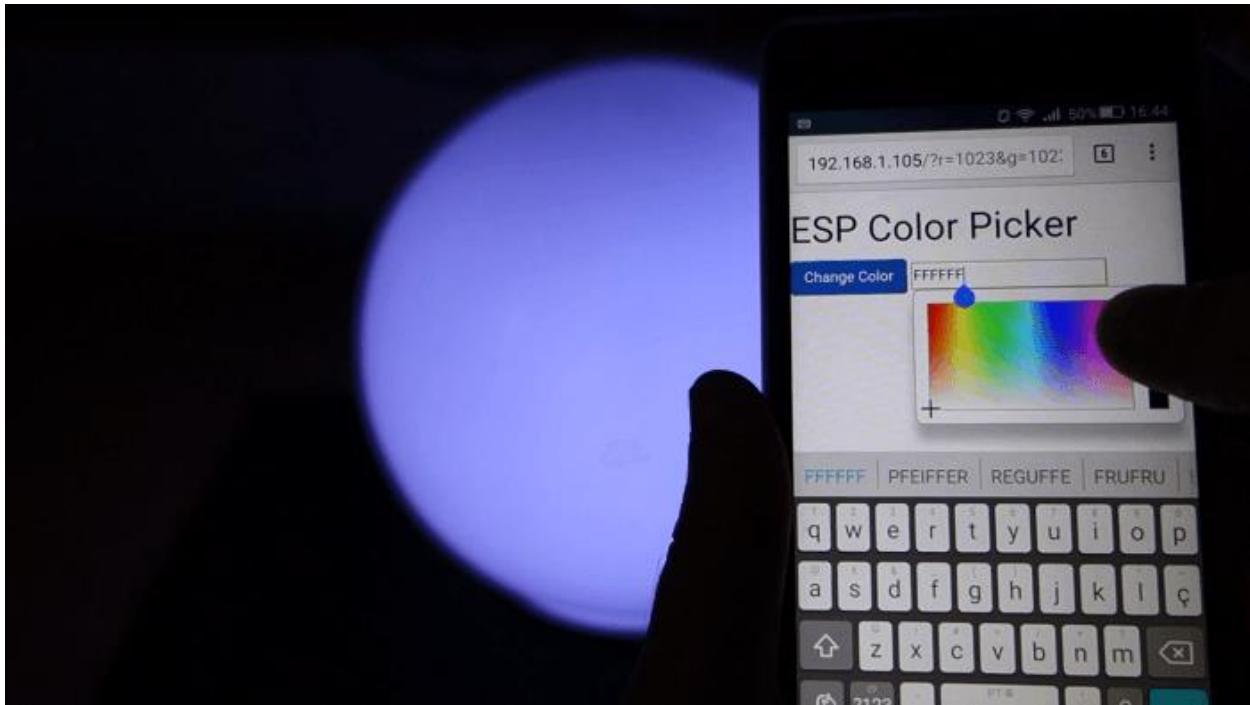
Now your RGB LED changes to the blue color:



Taking It Further

This is a basic example that shows you how easy it is to remotely control an RGB LED with an ESP8266. You can take this example and modify it to control an actual lamp as shown in the next Unit.

Unit 9: DIY WiFi RGB LED Mood Light



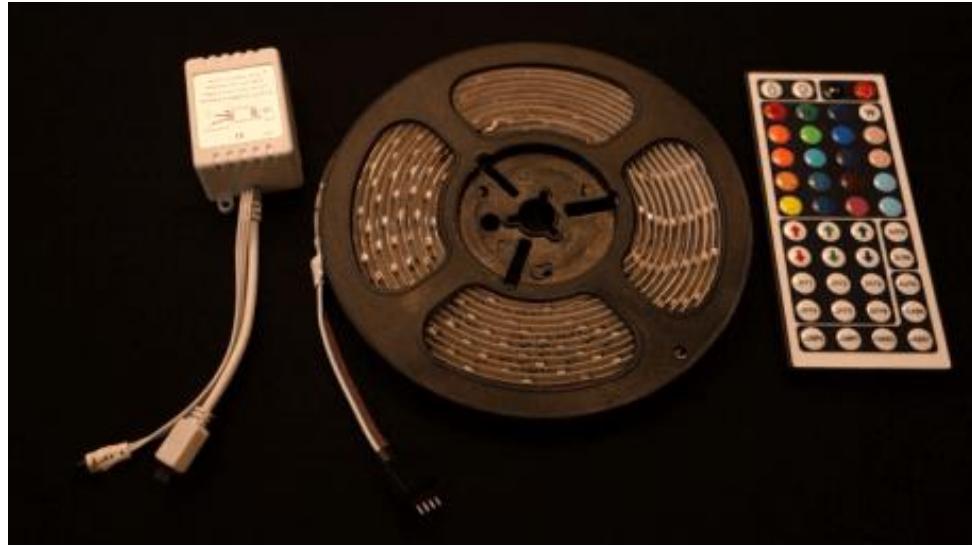
In this project, I'll show you how you can build a mood light. You'll use an ESP8266 to remotely control the color of your light using your smartphone or any other device that has a browser. This project is called \$10 DIY WiFi RGB LED Mood Light.

To learn more about RGB LEDs use the following tutorial as a reference: [How do RGB LEDs work?](#)

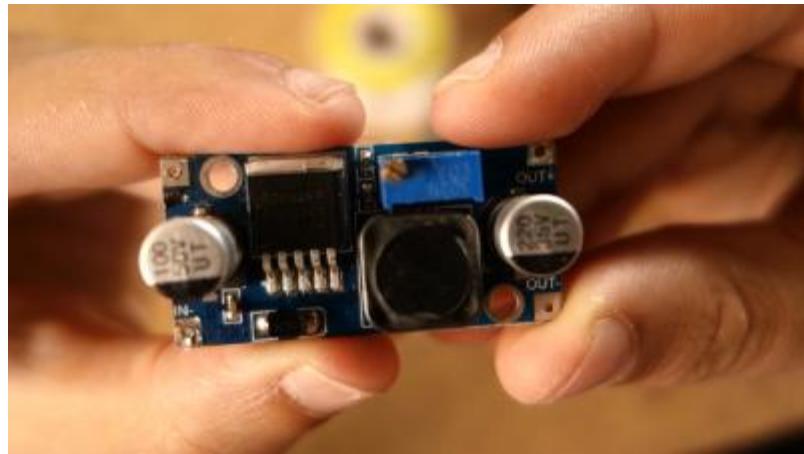
Parts Required

Here's the hardware that you need to complete this project:

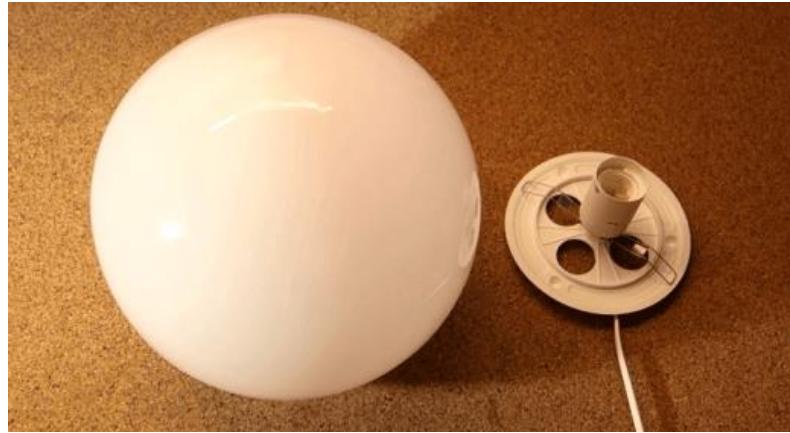
- [ESP8266](#)
- [RGB LED Strip](#)



- [DC12V Power Supply](#)
- Device to reduce voltage from 12V to 5V
 - Alternative – LM7805 with heat sink
 - **Recommended** – [Step down buck converter module](#)



- [3x NPN Transistors 2N2222 or equivalent](#)
- [3x 1k Ohm Resistors](#)
- [Breadboard](#)
- [Jumper Wires](#)
- Table Lamp with Mood Light Look



Writing Your init.lua Script

Upload the following code into your ESP8266 using the ESPlorer IDE. Your file should be named "init.lua".

SOURCE CODE

```
https://github.com/RuiSantosdotme/ESP8266-eBook/blob/master/Code/PART2\_LUA/Unit9\_Mood\_Light\_with\_ESP8266.lua
```

Don't forget to add your network name (SSID) and password to the script.

```
wifi.setmode(wifi.STATION)
wifi.sta.config("REPLACE_WITH_YOUR_SSID","REPLACE_WITH_YOUR_PASSWORD")

print(wifi.sta.getip())

function led(r, g, b)
    pwm.setduty(5, r)
    pwm.setduty(6, g)
    pwm.setduty(7, b)
end

pwm.setup(5, 1000, 1023)
pwm.setup(6, 1000, 1023)
pwm.setup(7, 1000, 1023)
```

```

pwm.start(5)
pwm.start(6)
pwm.start(7)
srv=net.createServer(net.TCP)
srv:listen(80,function(conn)
    conn:on("receive", function(client,request)
        local buf = "";
        buf = buf.."HTTP/1.1 200 OK\r\n"
        local _, _, method, path, vars = string.find(request, "([A-Z]+) (.+)?(.+) HTTP");
        if(method == nil)then
            _, _, method, path = string.find(request, "([A-Z]+) (.+) HTTP");
        end
        local _GET = {}
        if (vars ~= nil)then
            for k, v in string.gmatch(vars, "(%w+)=(%w+)&*") do
                _GET[k] = v
            end
        end
        buf = buf.."<!DOCTYPE html><html><head>";
        buf = buf.."<meta charset=\"utf-8\">";
        buf = buf.."<meta http-equiv=\"X-UA-Compatible\" content=\"IE=edge\">";
        buf = buf.."<meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">";
        buf = buf.."<link rel=\"stylesheet\" href=\"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css\">";
        buf = buf.."<script src=\"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js\"></script>";
        buf = buf.."<script src=\"https://cdnjs.cloudflare.com/ajax/libs/jscolor/2.0.4/jscolor.min.js\"></script>";
        buf = buf.."</head><body><div class=\"container\"><div class=\"row\"><h1>ESP Color Picker</h1>;
        buf = buf.."<a type=\"submit\" id=\"change_color\" type=\"button\" class=\"btn btn-primary\">Change Color</a> ";
        buf = buf.."<input class=\"jscolor {onFineChange:'update(this)'}\" id=\"rgb\"></div></div>";
        buf = buf.."<script>function update(picker) {document.getElementById('rgb').innerHTML = Math.round(picker.rgb[0]) + ', ' + Math.round(picker.rgb[1]) + ', ' + Math.round(picker.rgb[2]);";
        buf = buf.."document.getElementById(\"change_color\").href=?r=" + Math.round(picker.rgb[0]*4.0117) + "&g=" + Math.round(picker.rgb[1]*4.0117) + "&b=" + Math.round(picker.rgb[2]*4.0117);}</script></body></html>";

        if(_GET.r or _GET.g or _GET.b) then
            led(_GET.r, _GET.g,_GET.b)
    end
end)

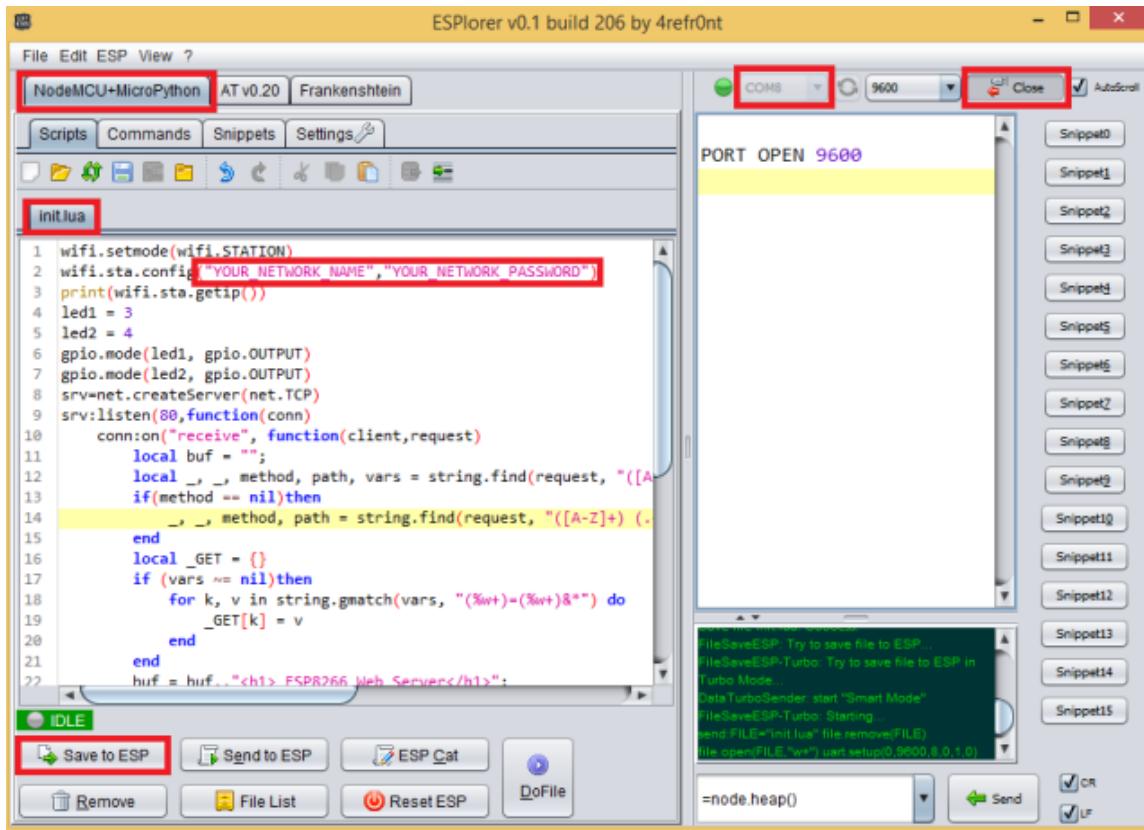
```

```
    end  
    client:send(buf);  
    client:close();  
    collectgarbage();  
end)  
end)
```

Uploading init.lua

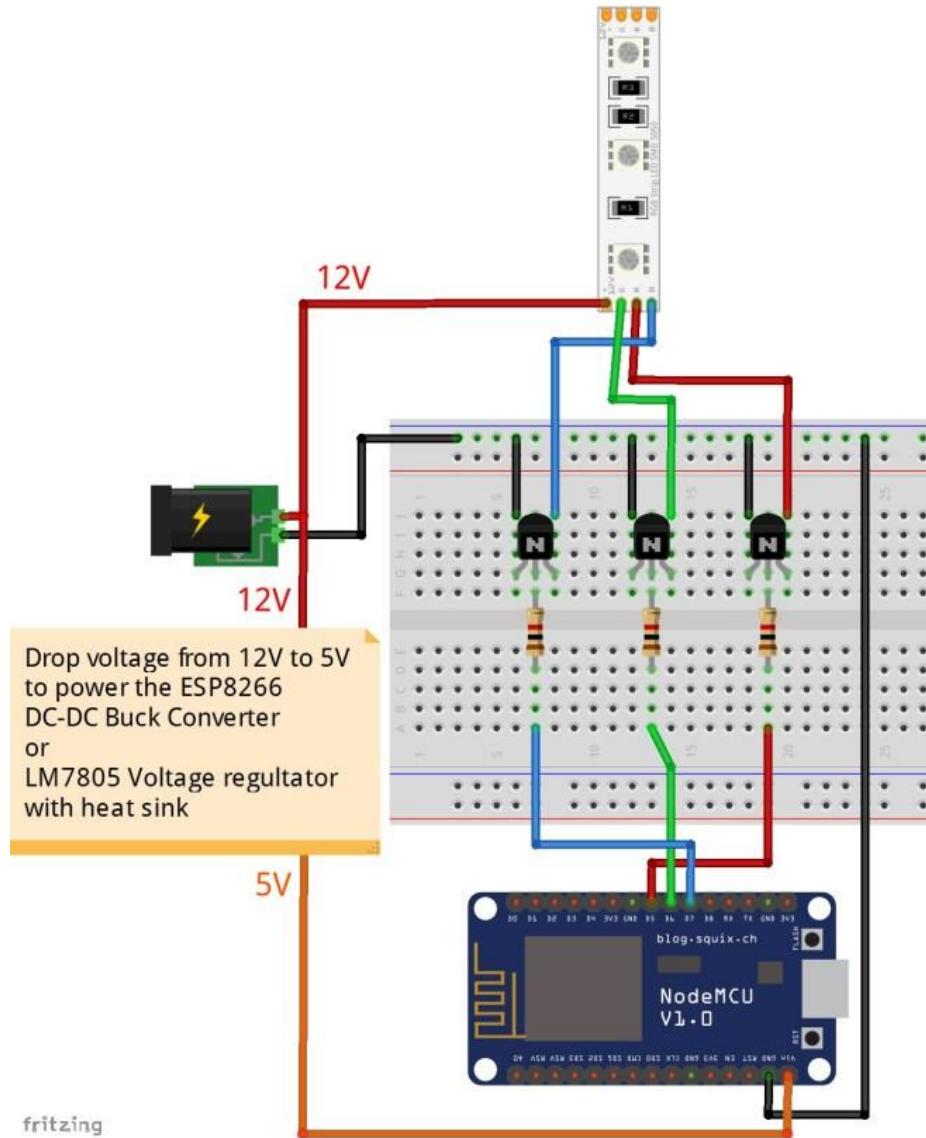
Follow these instructions to upload a Lua script:

1. Connect your ESP8266-12E that has built-in programmer to your computer
 2. Select your ESP8266-12E port
 3. Press Open/Close
 4. Select NodeMCU+MicroPython tab
 5. Create a new file called init.lua
 6. Press Save to ESP



Final Circuit

Now follow this schematic to create the final circuit.



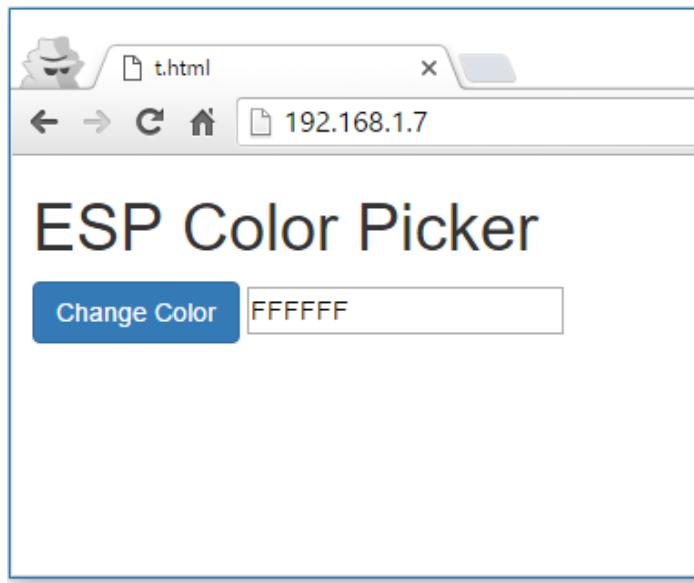
ESP8266 IP Address

When your ESP8266 restarts, it prints ESP IP address in the serial monitor the. Save that IP address, because you'll need it later.

In my case, the ESP IP address is 192.168.1.105. You're all set!

Opening Your Web Server

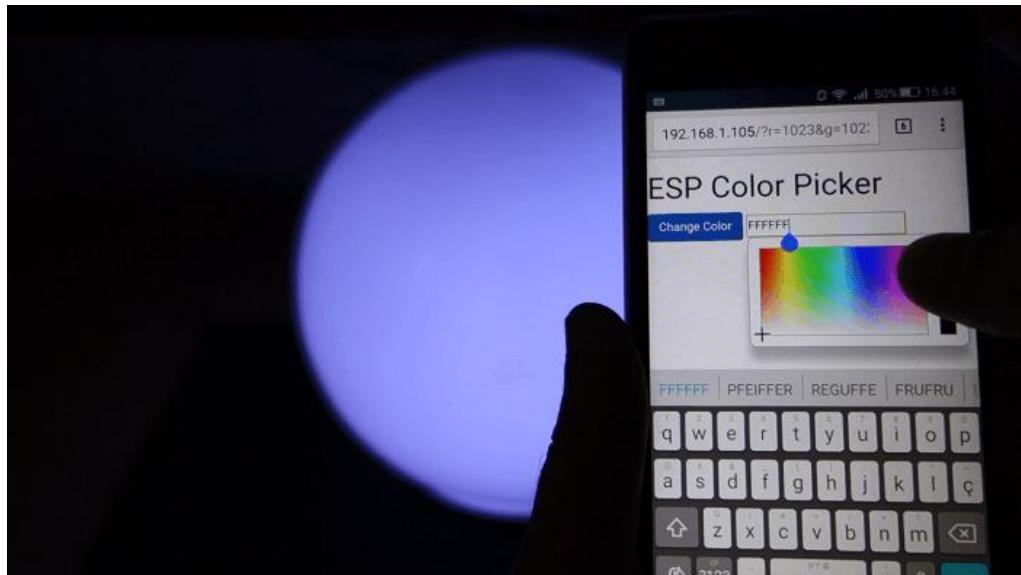
Go to any browser and enter the IP address of your ESP8266. This is what you should see:



Click the input field and a small window opens with a color picker. Simply drag your mouse or finger and select the color for your RGB LED strip:



Finally, press the “Change Color” button:



Now, your mood light can be placed in your living room:



Final Thoughts

Congratulations for completing this eBook!

If you followed all the projects presented in this eBook, you now have the knowledge to build your Home Automation system using the ESP8266.

Let's see the most important concepts that you've learned. You know how to:

- Use the ESP8266 with Arduino IDE and NodeMCU firmware;
- Create a password protected web server to control any output;
- Make your web server accessible from anywhere in the world;
- Create a web page to display and monitor sensor data;
- Build an email alert system;
- Use MQTT to communicate between two ESP boards;
- And much more...

Now, feel free to add multiple ESP8266 boards to your projects and build up on the snippets of code presented in this eBook to fit your own project requirements.

I hope you had fun following all these projects! If you have something that you would like to share let me know in the Facebook group ([Join the Facebook group here](#)).

Good luck with all your projects,

Rui Santos

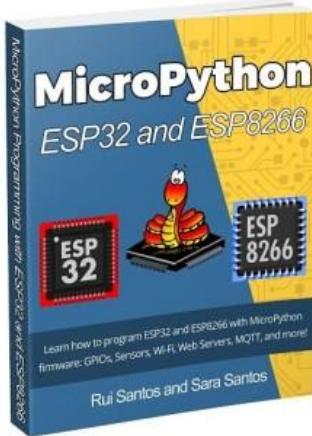
Download Other RNT Products

[Random Nerd Tutorials](#) is an online resource with electronics projects, tutorials and reviews. Creating and posting new projects takes a lot of time. At this moment, Random Nerd Tutorials has more than 200 free blog posts with complete tutorials using open-source hardware that anyone can read, remix and apply to their own projects.

To keep free tutorials coming, there's also paid content or as we like to call "premium content". To support Random Nerd Tutorials, you can [download premium content here](#). If you enjoyed this eBook/course, make sure you [check all the others](#).

MicroPython Programming with ESP32 and ESP8266

Learn how to program the ESP32 and ESP8266 with MicroPython, a re-implementation of Python 3 programming language targeted for microcontrollers. This is one of the easiest ways to program your ESP32/ESP8266 boards! [Read product description.](#)



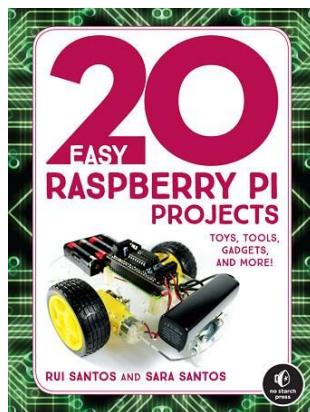
Learn ESP32 with Arduino IDE

If you like the ESP8266, you'll love the ESP32. This 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! [Read product description.](#)



20 Easy Raspberry Pi Projects

20 Easy Raspberry Pi Projects book is a beginner-friendly collection of electronics projects using the Raspberry Pi. Projects are explained with full-color visuals and simple step-by-step instructions. This book was a collaboration with the NoStarch Press Publisher and it is available in paperback format. [Read product description.](#)



Build a Home Automation System for \$100

Learn Raspberry Pi, ESP8266, Arduino and Node-RED. This is a premium step-by-step course to get you building a real world home automation system using open-source hardware and software. [Read product description](#).



Arduino Step-by-step Projects

Our step-by-step course to get you building cool Arduino projects even with no prior experience! This Arduino Course is a compilation of 25 projects divided in 5 Modules that you can build by following clear step-by-step instructions with schematic diagrams and downloadable code. [Read product description](#).



Android Apps for Arduino with MIT App Inventor 2

This eBook is our step-by-step guide designed to get you building cool Android applications for Arduino, even with no prior experience! *Android Apps for Arduino with MIT App Inventor 2* is a practical course in which you're going to build 8 Android applications to interact with the Arduino. [Read product description.](#)

