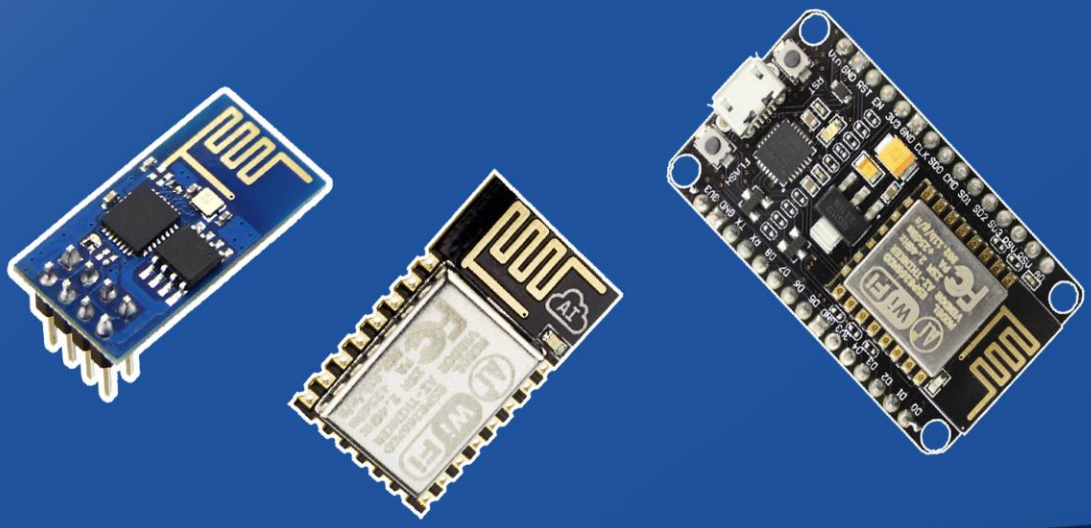
HOME AUTOMATION USING ESP8266

2nd Edition

Complete guide for ESP8266



How to control devices and read sensors using the low cost ESP8266 WiFi module with Arduino IDE and NodeMCU firmware

Written by Rui Santos

HOME AUTOMATION USING ESP8266

Security Notice

This is the kind of thing I hate to have 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 actions.**

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

- http://randomnerdtutorials.com/home-automation-using-esp8266

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

What I really want to say is thanking your purchasing this eBook and I hope you have fun 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 author (Rui Santos) does not warrant that the information contained in this eBook is fully complete and shall not be responsible for any errors or omissions.

The author (Rui Santos) 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 will find some links and some of them are affiliate links. This means the author (Rui Santos) earns a small commission from each purchase with that link. Please understand that the author has experience with all of these products, and he recommends them because they are useful, not because of the small commissions he makes if you decide to buy something. Please do not spend any money on these products unless you feel you need them.

Other Helpful Links

- Join Private Facebook Group
- Contact Support Via Email
- Terms and Conditions

Table of Contents

About the Author

Hey there!

Thank you for purchasing my eBook "Home Automation Using ESP8266".

I'm Rui Santos, founder of the Random Nerd Tutorials blog and author of BeagleBone For Dummies.

If you're new to the world of ESP8266, this eBook is perfect for you! If you already used the ESP8266 before, I'm sure you'll also learn something new.

This eBook contains the information you need to get up to speed quickly and start your own venture with the ESP8266 using both the Arduino IDE and the NodeMCU firmware!

Thanks for reading,

Rui Santos

P.S. If you would like the longer version of my story, you can find it over here.

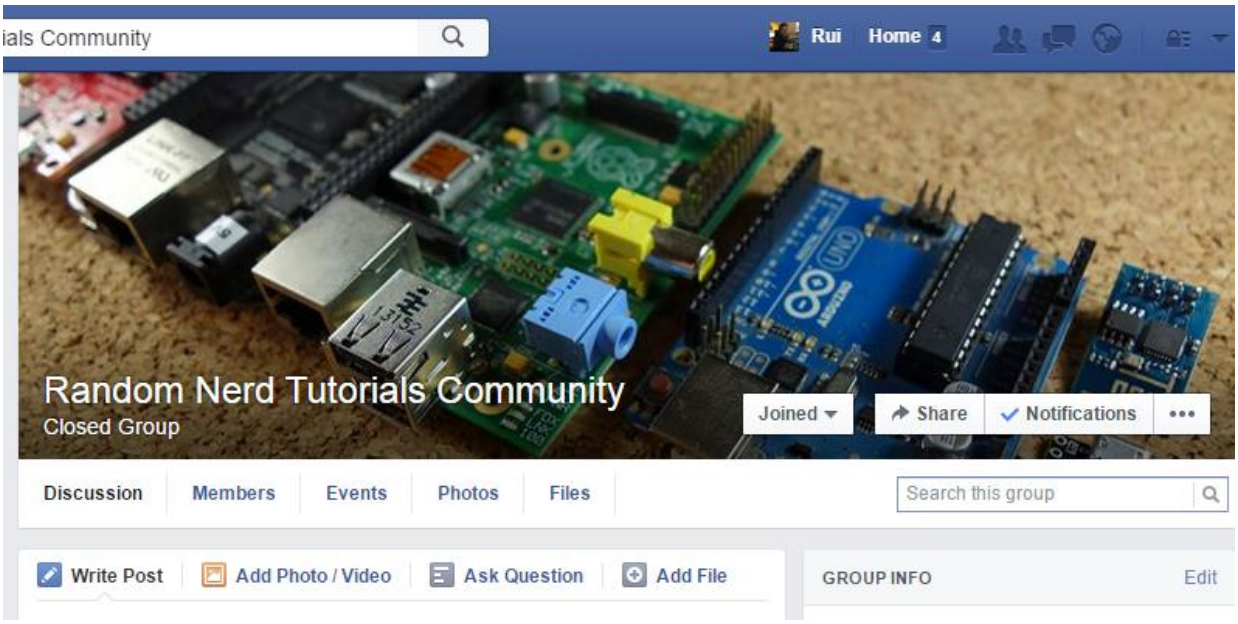
Join the Private Facebook Group

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, 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

Unit 1 - Getting Started with ESP8266

Hello and thank you for purchasing this eBook!

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

This eBook covers:

- Technical specifications of the ESP8266
- Where to buy the ESP8266
- How to install the Arduino IDE and how it works
- How to flash the ESP8266 with NodeMCU firmware
- How to install ESPlorer IDE and how it works
- How to establish a serial communication with the ESP8266
- How to blink an LED with ESP8266
- How to interact with the ESP8266 GPIOs
- How to create a web server
- How to access your web server from anywhere
- Lua programming language
- How to send emails with the ESP8266
- How to create an email notifier with a PIR Motion Sensor
- And a lot more…

Let's Begin!

Part 0 gives you an overview of what you can do with your ESP8266, the ESP technical specifications and where you can buy one.

About the ESP8266

The ESP8266 is a $4 (up to $10) WiFi module with an ARM processor that is great to extend the functionality of a microcontroller such as an Arduino. It can communicate with your microcontroller via serial.

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

Comparing the ESP with other WiFi solutions in the market, this is definitely 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?

You can create a web server, send HTTP requests, control outputs, read inputs and interrupts, send emails, post tweets, etc.

ESP8266 specifications

- 802.11 b/g/n protocol
- Wi-Fi Direct (P2P), soft-AP
- Integrated TCP/IP protocol stack
- Built-in low-power 32-bit CPU
- SDIO 2.0, SPI, UART

Finding Your ESP8266

The ESP8266 comes in a wide variety of versions (as shown in the figure below). The ESP-12E or often called ESP-12E NodeMCU Kit is currently the most practical version and that's the module we'll be using most throughout this eBook.

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

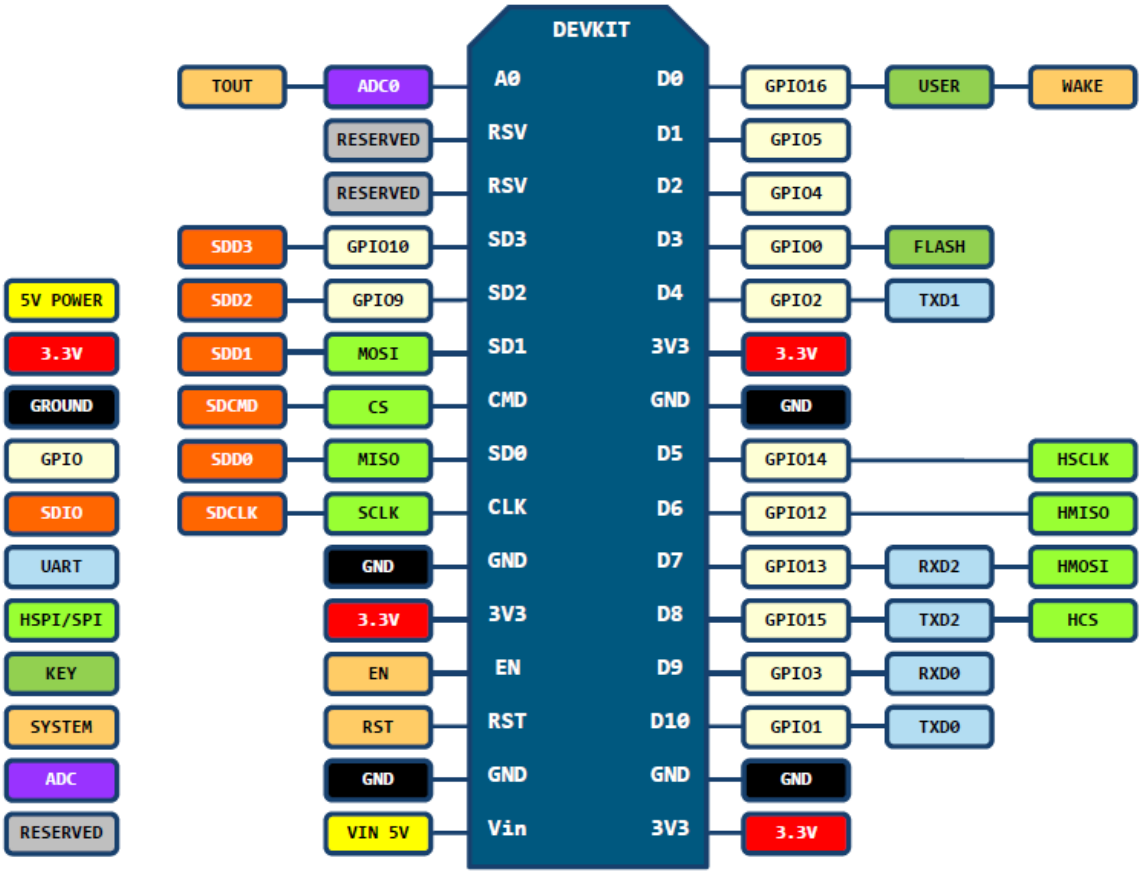
Recommended ESP8266 Board

This eBook was tested with ESP-01, ESP-07, ESP-12 and ESP-12E. So you can make all the projects presented in this eBook using any of those boards.

I highly recommend using the ESP8266-12E NodeMCU Kit, the one that has built-in programmer. The built-in programmer makes it easy to prototype and upload your programs. Click here to buy this module on eBay.

ESP-12E NodeMCU Kit Pinout

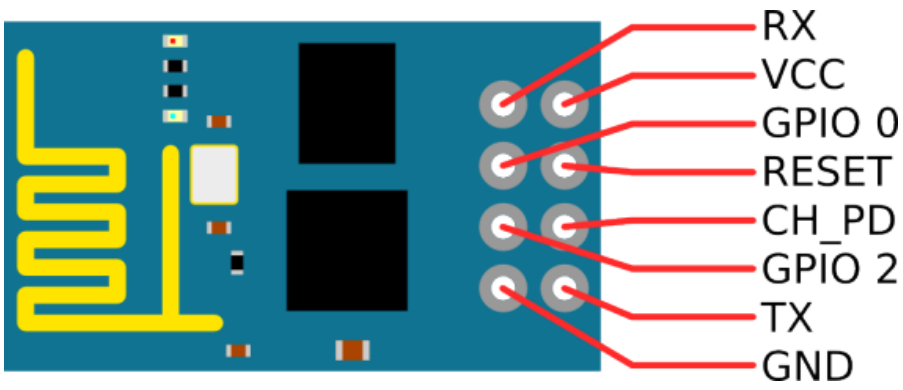
Here's a quick overview of the ESP-12E NodeMCU Kit pinout:



D0(GPIO16) can only be used as gpio read/write, no interrupt supported, no pwm/i2c/ow supported.

ESP-01 Pinout

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

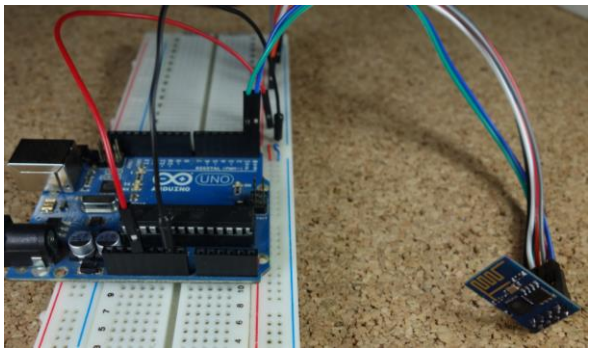


Warning: before applying power to your module, please note that this module operates at 3.3V. If you plug it up to 5V, it will fry.

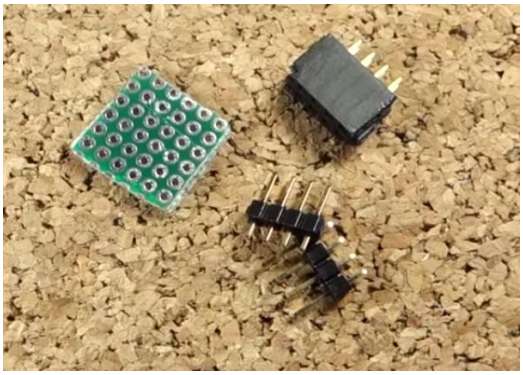
ESP-01 is Not Breadboard Friendly…

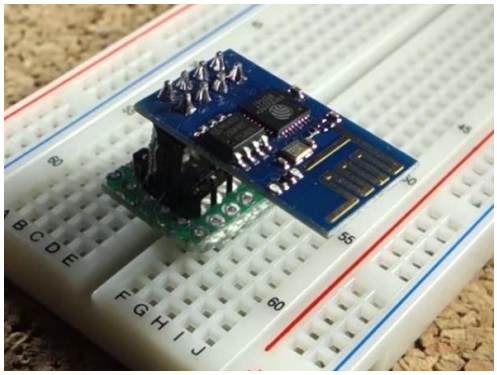
Sadly, out of the box your ESP-01 is not breadboard friendly. You can either use some female to male jumper wires (described below as Option A); or you can take an extra step and design a small PCB that fits nicely in your breadboard (described below as Option B). Both ways work fine!

Option A – jumper wires



Option #2 – small PCB





Choosing Your Programming Environment

This eBook is divided in two distinct parts and each part covers a different programming environment/firmware for the ESP8266:

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

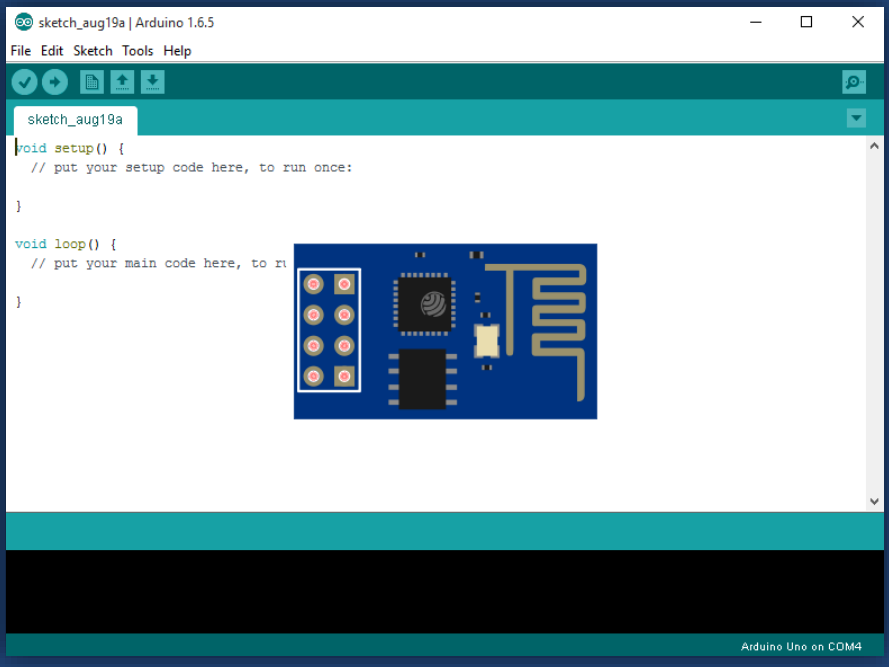
There are various firmware available to program the ESP8266 board. This eBook covers how to use the Arduino IDE and NodeMCU firmware.

I 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.

I encourage you to read this book linearly and try the Arduino IDE first, then you can try to program the ESP with NodeMCU firmware.

PART 1

ESP8266 with Arduino IDE



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. This means you can program your ESP using the friendly Arduino programming language.

What's the Arduino IDE?

The Arduino IDE is an open-source software that makes it easy to write code and upload it to the Arduino board. This GitHub repository added support for the ESP 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.

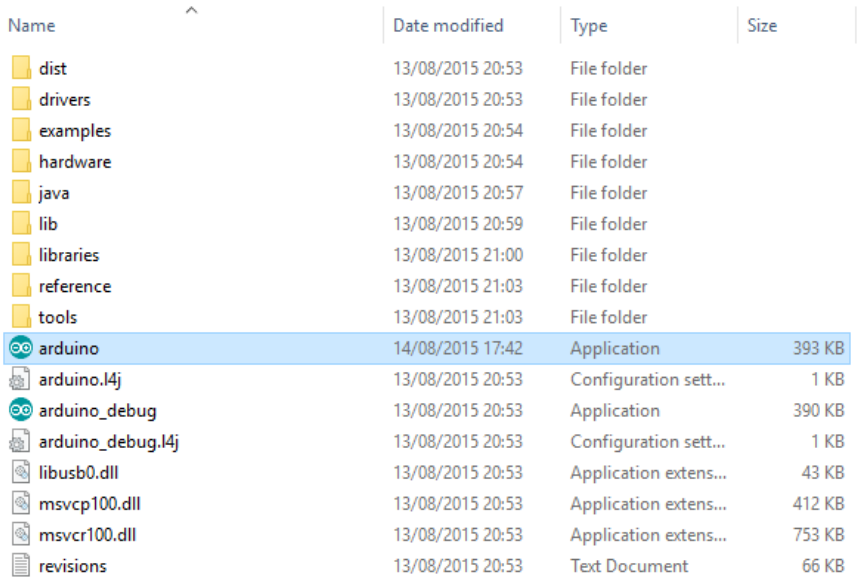
Downloading Arduino IDE

To download the Arduino IDE, visit the following URL: https://www.arduino.cc/en/Main/Software.

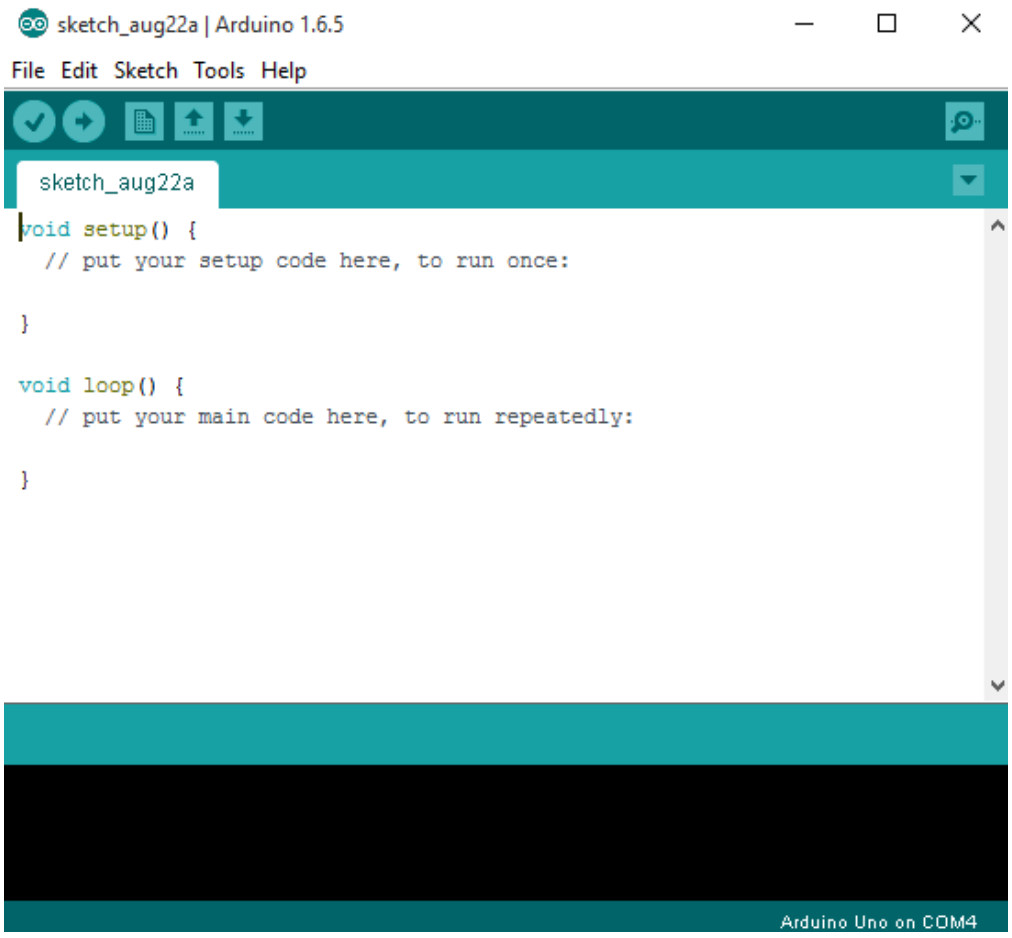
Then, select your operating system and download the software (as shown below).

Installing Arduino IDE

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



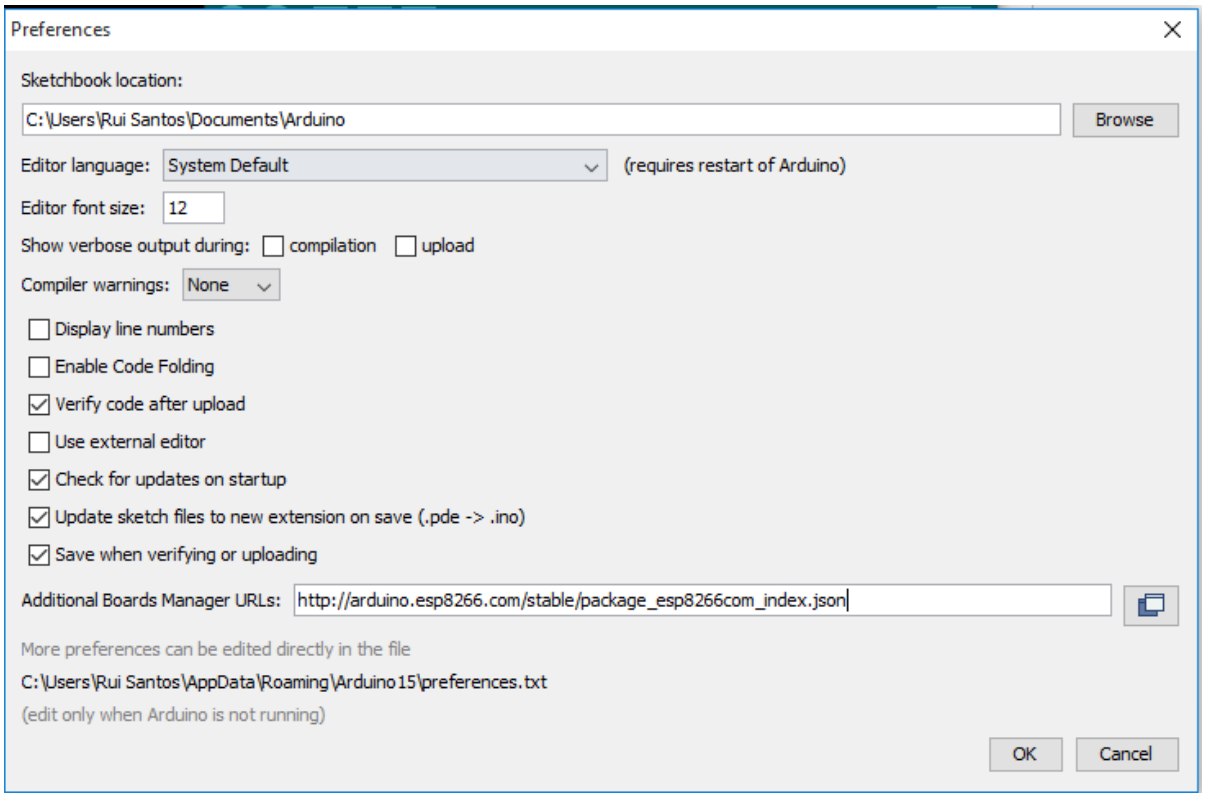
When the Arduino IDE first opens, this is what you should see:



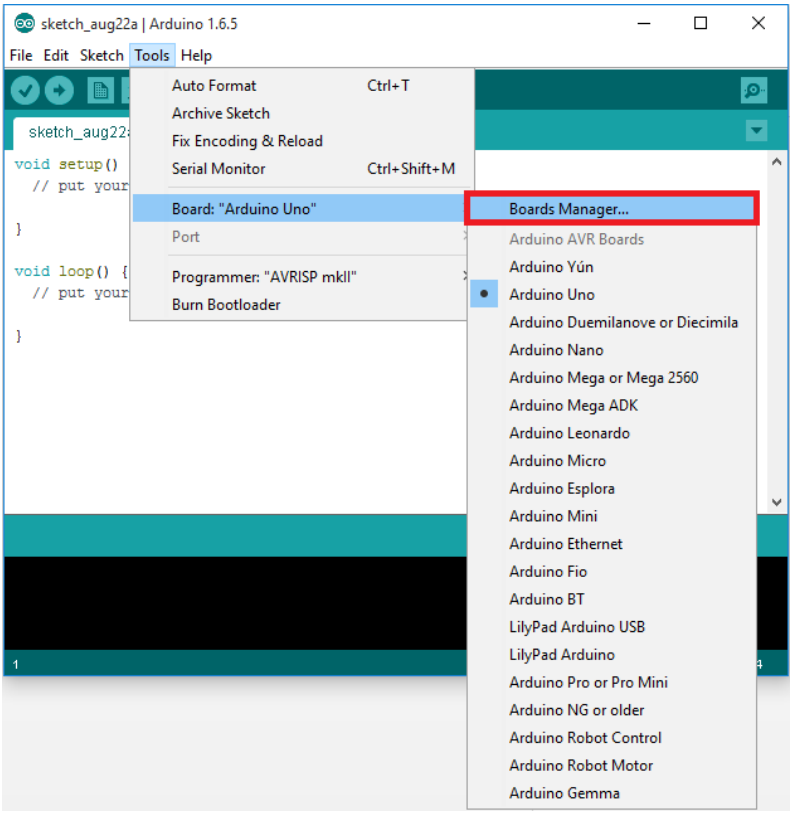
Installing ESP8266 Board

To install the ESP8266 board in your Arduino IDE, follow these next instructions:

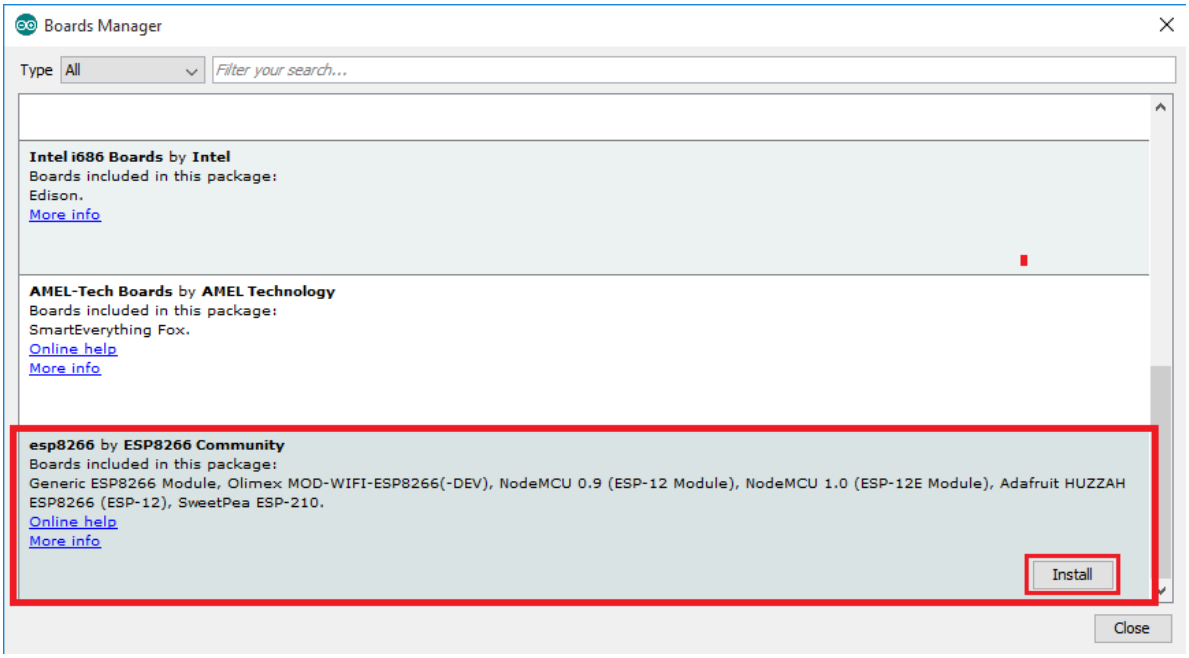
1. Open the preferences window from the Arduino IDE. Go to **File** > **Preferences**
2. Enter http://arduino.esp8266.com/stable/package_esp8266com_index.json into **Additional Board Manager URLs** field and press the "**OK**" button



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

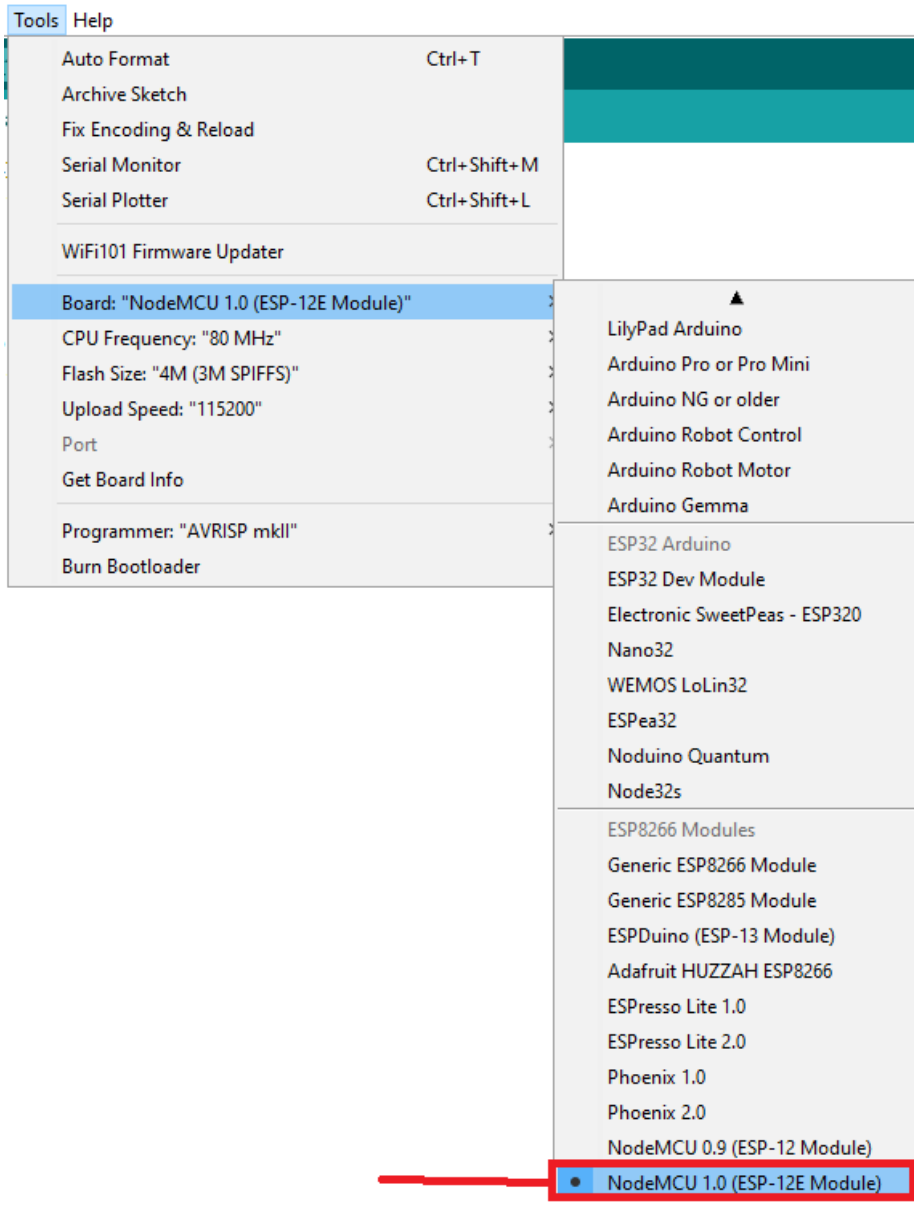


4. Scroll down, select the ESP8266 board menu and Install "**esp8266 by ESP8266 Community**"



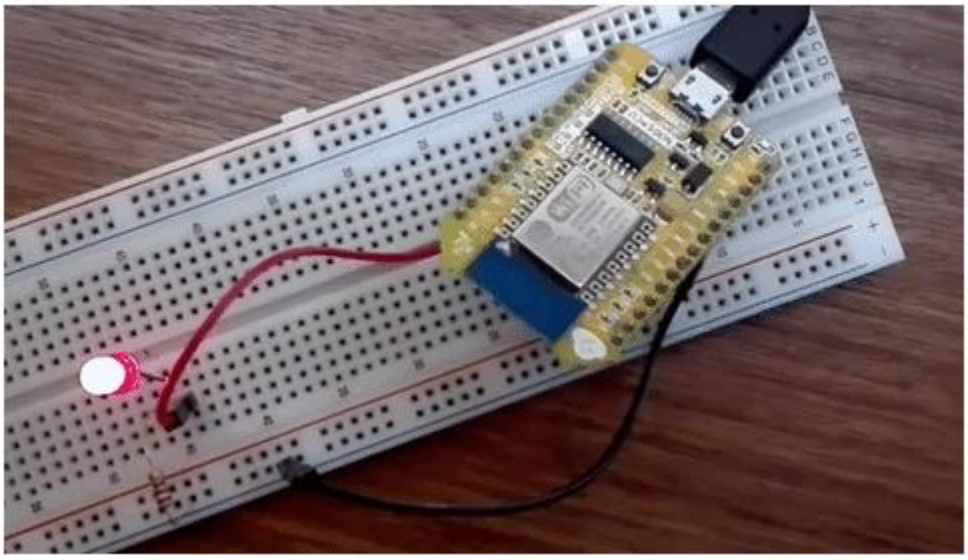
5. Open the Arduino **Tools** menu

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



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

Unit 2
Blinking LED with Arduino IDE



Unit 2 - Blinking LED with Arduino IDE

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

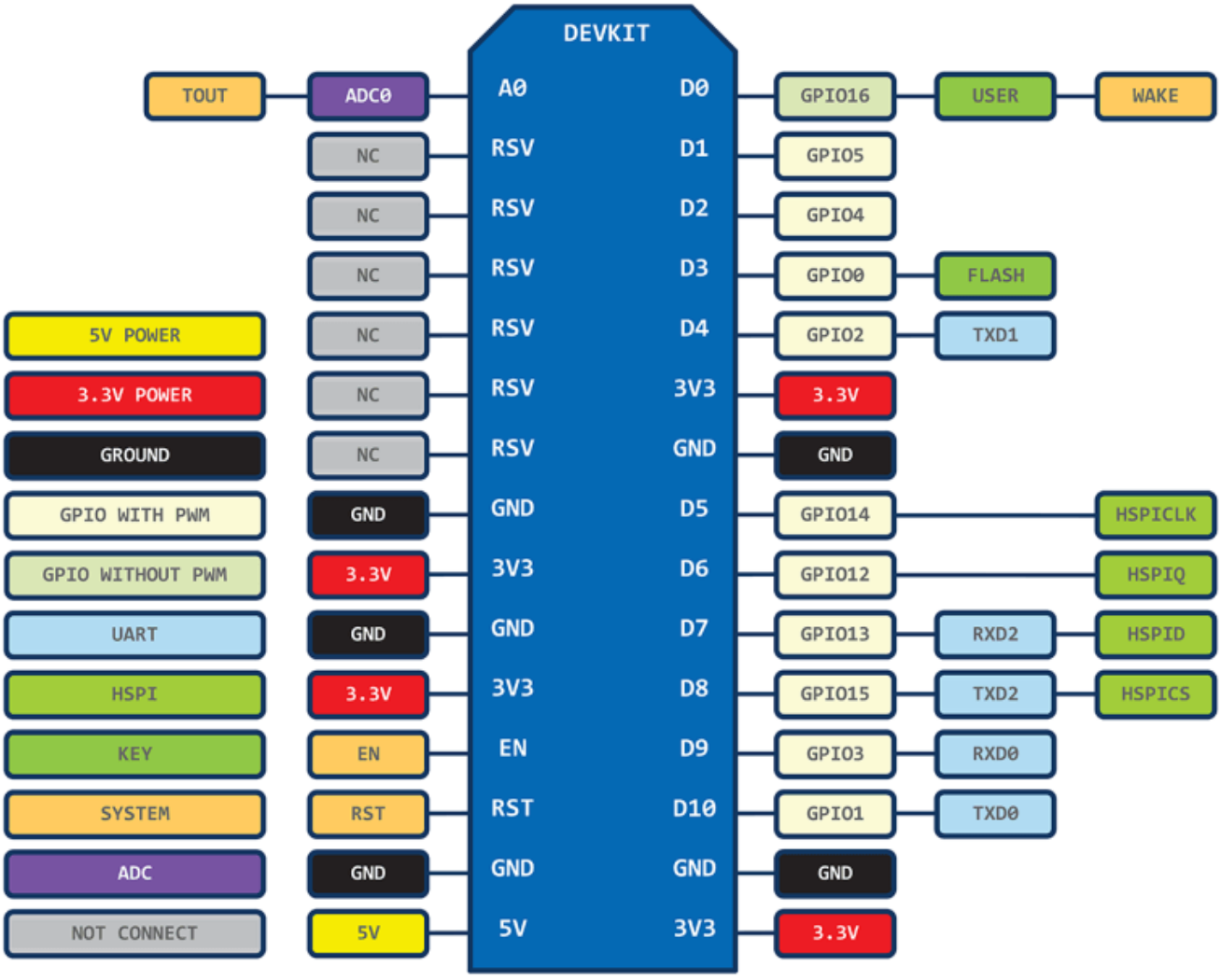
Why do we always blink an LED first? That's a great question! If you can blink an LED you can pretty much say that you can turn any electronic device on or off. Whether is an LED, a lamp or your toaster.

About GPIOs Assignment

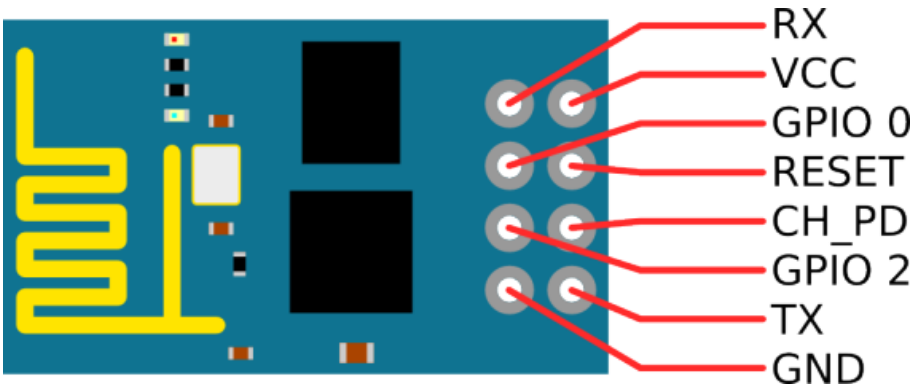
Use this next table as a quick reference on how to assign the ESP8266 GPIOs in the Arduino code.

IO index (Arduino code)	ESP8266 GPIO	ESP-12E
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	ADC 0	A0

Here's the location of 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 Arduino Sketch” when we define:

pin = 0

we are referring to **GPIO 0**, and if we define:

pin = 2

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 0 refers to GPIO 0 and 2 refers to GPIO 2. I’ll explore this concept in more detail later in this eBook.

Writing Your Arduino Sketch

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

SOURCE CODE

https://gist.github.com/RuiSantosdotme/789861277da9680e9cfb

```
int pin = 2;

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

// the loop function runs over and over again forever
void loop() {
  digitalWrite(pin, HIGH);   // turn the LED on (HIGH is the voltage level)
```

```
  delay(1000);               // wait for a second
  digitalWrite(pin, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

How this sketch works:

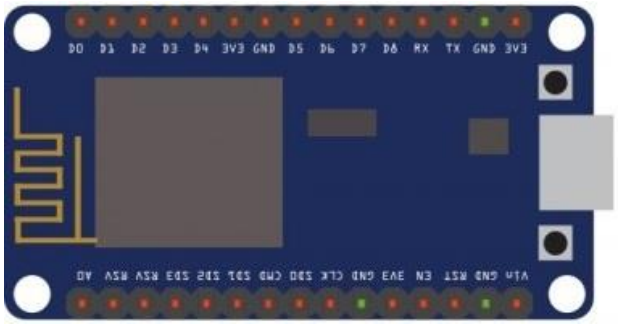
1. Create a *integer (int)* variable called ***pin* = 2** which refers to **GPIO 2**
2. In ***setup()***, you use the function ***pinMode(pin, OUTPUT)*** to set your **GPIO 2** as an ***OUTPUT*** . This code only runs once.
3. Next, in the ***loop()***, you use two functions ***digitalWrite()*** and ***delay()***. This section of code will run over and over again until you unplug your ESP.
4. You turn the LED on for 1 second (1000 milliseconds) using ***digitalWrite(pin, HIGH)*** and ***delay(1000)***.
5. Then you turn the LED off using ***digitalWrite(pin, LOW)*** and wait 1 second with ***delay(1000)***.
6. The program keeps repeating steps 4. and 5. which makes the LED blinking!

Uploading Code to ESP8266

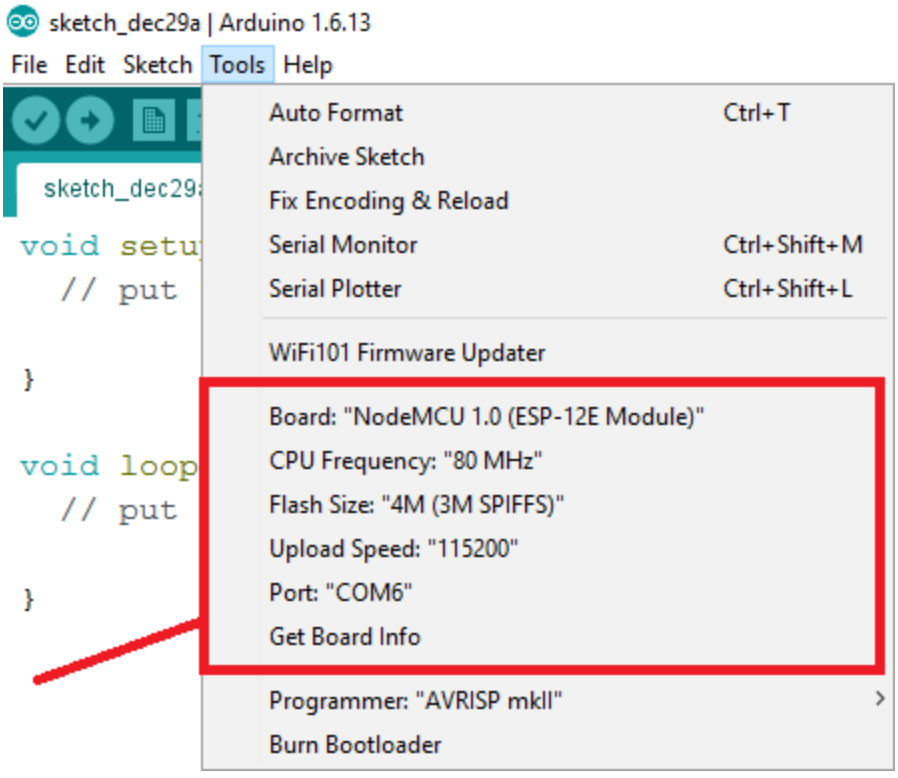
There are two different sections to upload code to your ESP8266. If you're using an ESP-12E that has built-in programmer read Option A. If you're using the ESP-01 or ESP-07, you need an FTDI programmer - read Option B.

Option A - Uploading code to ESP-12E

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

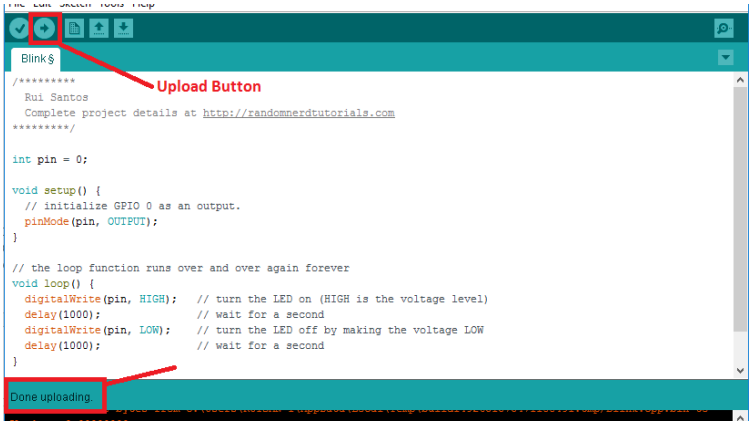


Look at the **Tools** menu, select Board "**NodeMCU 1.0 (ESP-12E Module)**" and all the configurations, by default, should look like this:



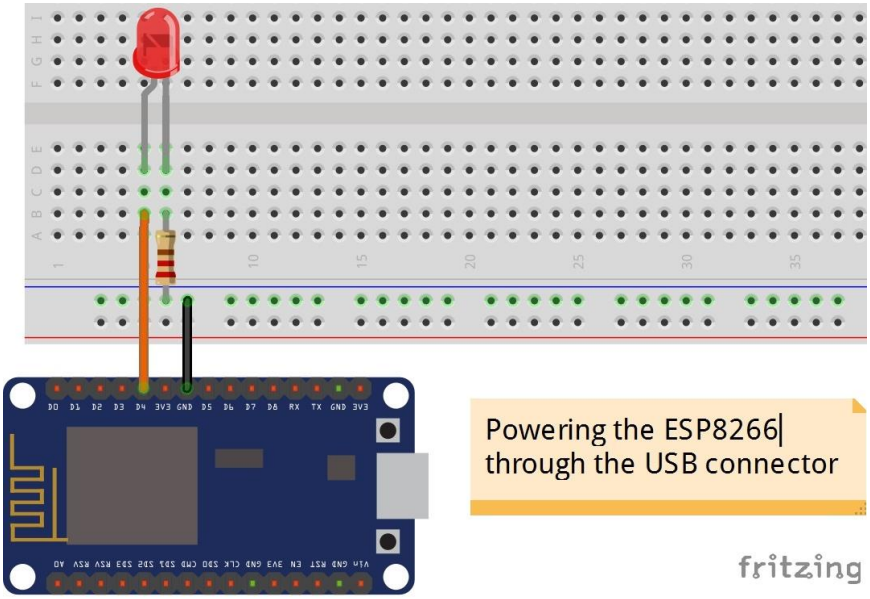
Important: your COM port is very likely to be different from the preceding screenshot (Port: “COM6”). That’s fine, because it doesn’t interfere with anything. On the other hand, all the other configurations should look exactly like mine.

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

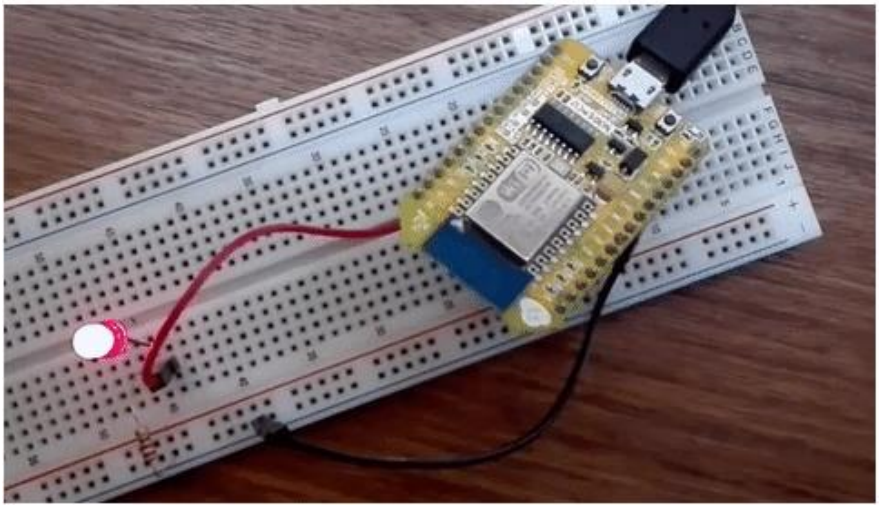


Final ESP-12E circuit

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



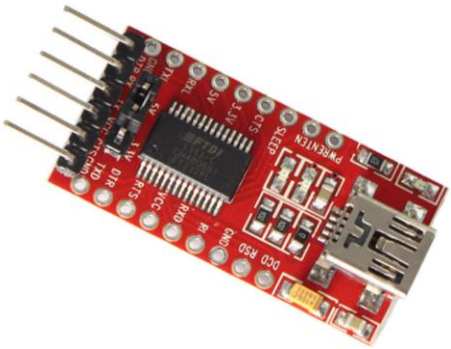
Restart your ESP8266. Congratulations, you've made it! Your LED should be blinking every 1 second!



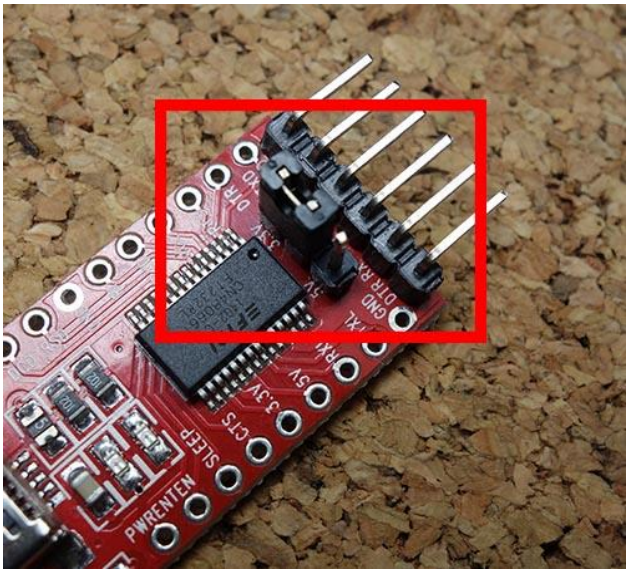
Option B - Uploading code to ESP-01

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

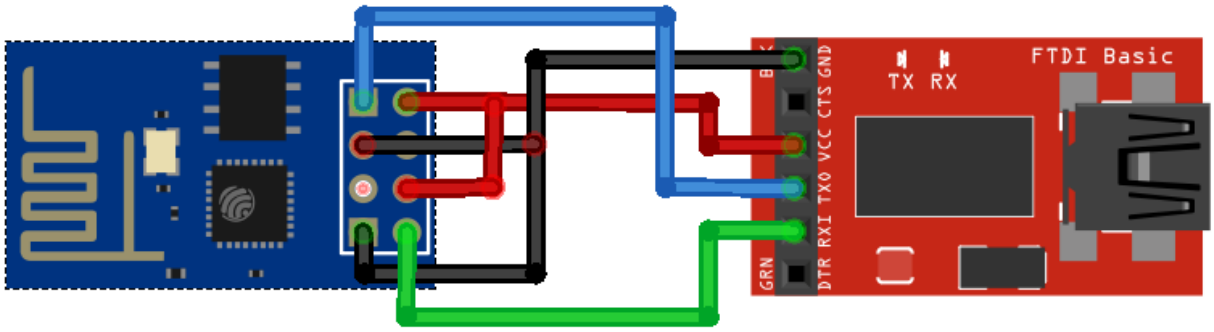
You can click here to purchase one FTDI Programmer on eBay for $3 (http://randomnerdtutorials.com/ebay-ftdi-programmer).



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.



Here's the connections:

- RX -> TX
- TX -> RX
- CH_PD -> 3.3V
- GPIO 0 -> GND
- VCC -> 3.3V
- GND -> GND

Note: the circuit above has GPIO 0 connected to GND, that's because we want to upload code. When you upload a new sketch into your ESP it requires the ESP to flash a new firmware. In normal usage (if you're not flashing your ESP with a new firmware) it would be connected to VCC.

If you have a brand new FTDI Programmer and you need to install your FTDI drivers on Windows PC, visit this website for the official drivers: http://www.ftdichip.com/Drivers/VCP.htm. In alternative, you can contact the seller that sold you the FTDI Programmer.

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 this video tutorial to fix that: http://youtu.be/SPdSKT6KdF8.

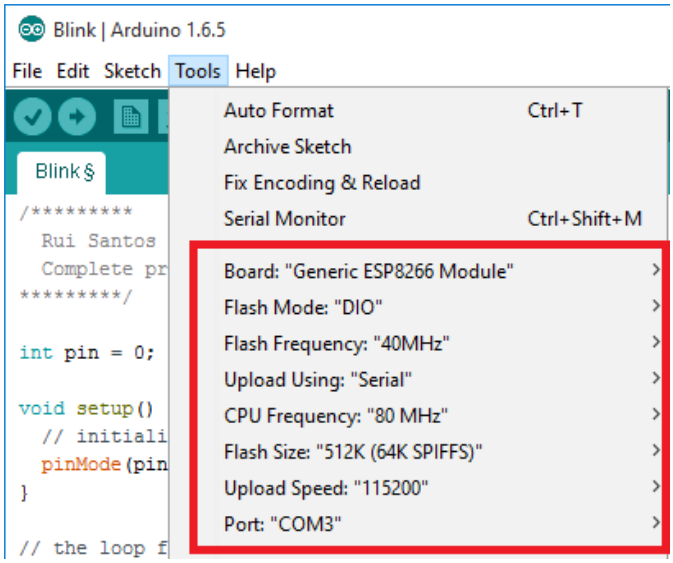
In the previous video, it is said for you to download the drivers from the FTDI website. Read carefully the YouTube description to find all the links. Here's the drivers you need:

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

Preparing your Arduino IDE

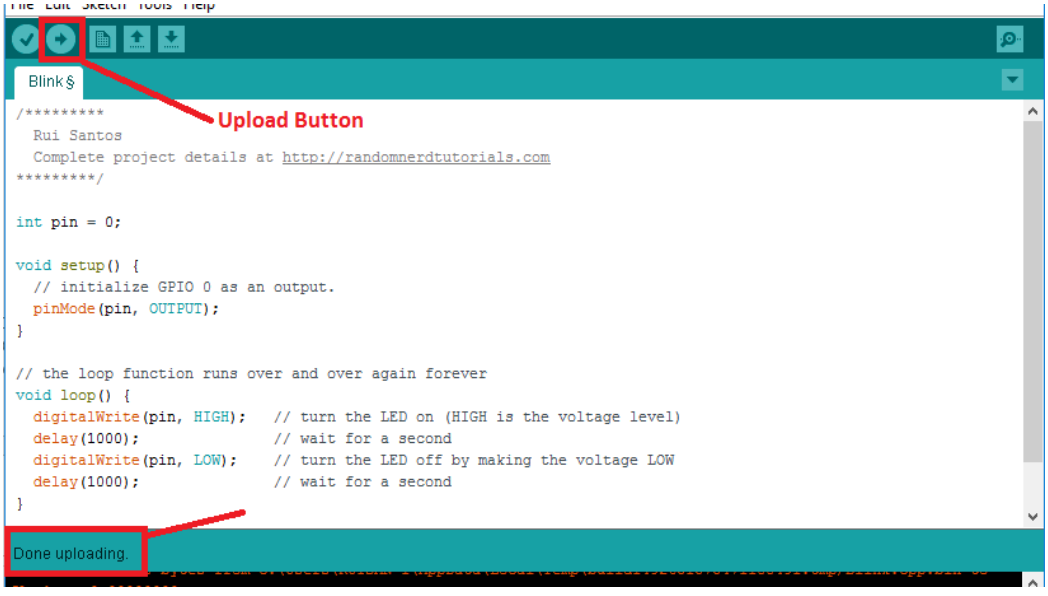
Once you have your ESP8266+FTDI Programmer connected to your computer, open the Arduino IDE.

Look at the **Tools** menu, select the **Board: "Generic ESP8266 Module"** and all the configurations, by default, should look like this:



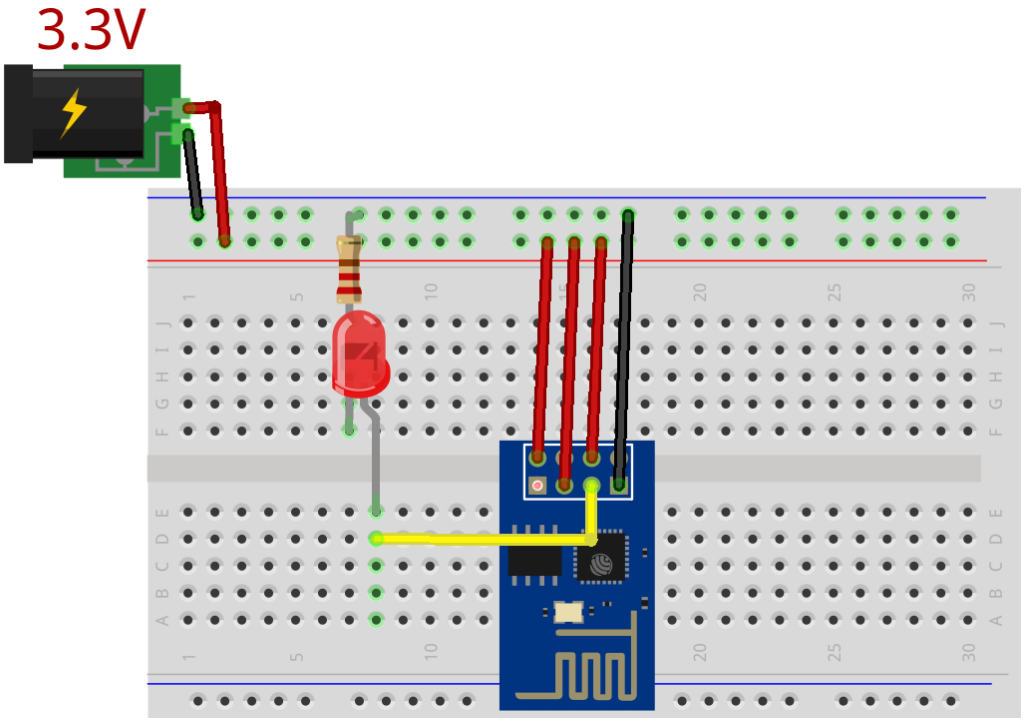
Important: your COM port is very likely to be different from the preceding screenshot (Port: "COM3"). That's fine, because it doesn't interfere with anything. On the other hand, all the other configurations should look exactly like mine.

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



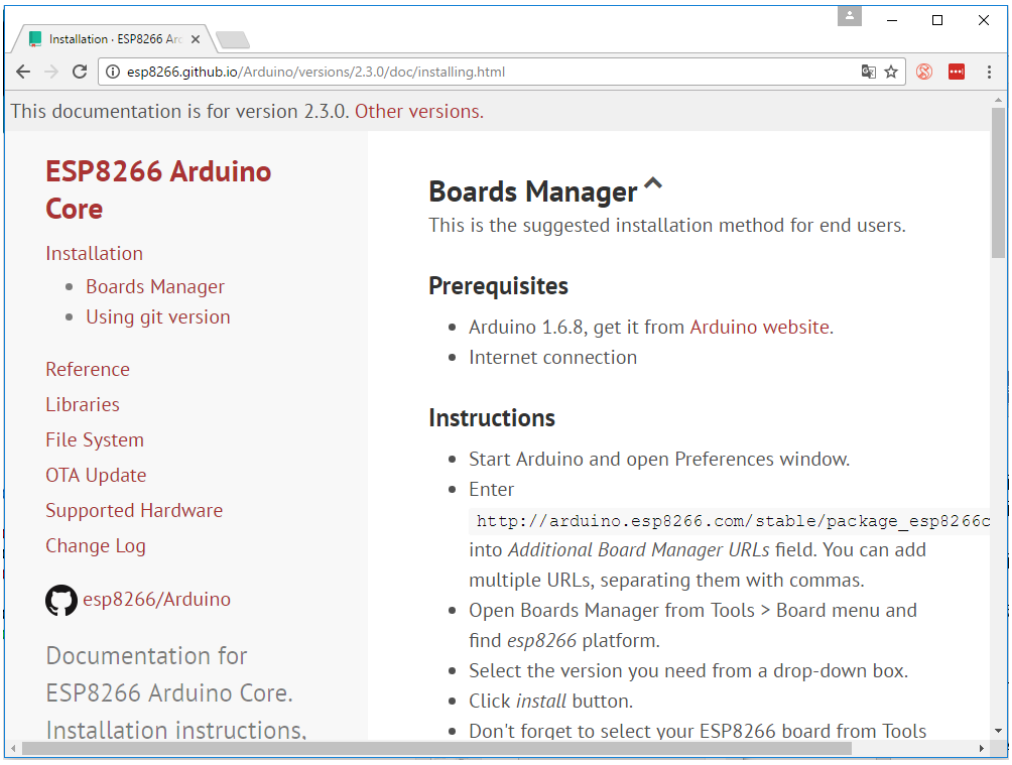
Final ESP-01 circuit

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



Apply power from a 3.3V source to the ESP. Congratulations, you've made it! Your LED should be blinking every 1 second!

Unit 3
Reference for ESP8266 using Arduino IDE



Unit 3 - Reference for ESP8266 using Arduino IDE

Before diving deeper into the main projects, it may be helpful taking a look at this unit to understand how to use the Arduino IDE with the ESP.

I encourage you to read the "Digital IO" section. The rest of this Unit is optional, so feel free to skip it and use it as a reference guide for your future projects.

More documentation and details about this reference can be found here: http://esp8266.github.io/Arduino/versions/2.3.0/

Note: this Unit might be subject to change in the future since I don't control updates and changes made by the authors of this ESP integration with the Arduino IDE. Please check the preceding URL for the latest information. This eBook will be updated if necessary.

Digital IO

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.

When using a GPIO pin, we first need to specify it's mode of operation. There are five possible modes that you can assign to each pin (one mode at a time for any pin):

Mode	Description
OUTPUT	You set the pin to HIGH or LOW

INPUT	You read the current state of the pin
INPUT_PULLUP	Similar to INPUT, but you use internal pull-up resistors
INPUT_PULLDOWN	Similar to INPUT, but you use internal pull-down resistors
INTERRUPT	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. Pin numbers in Arduino IDE correspond directly to the ESP8266 GPIO pin numbers. The ESP-01 has only two: GPIO 0 and GPIO 2. Use the table below as a reference if you are either using ESP-12E or ESP-01.

IO index (in code)	ESP8266 GPIO	Available Modes
0	**GPIO 0**	**INPUT, OUTPUT, INPUT_PULLUP or INTERRUPT**
1	GPIO 1	INPUT, OUTPUT, INPUT_PULLUP or INTERRUPT
2	**GPIO 2**	**INPUT, OUTPUT, INPUT_PULLUP or INTERRUPT**
3	GPIO 3	INPUT, OUTPUT, INPUT_PULLUP or INTERRUPT
4	GPIO 4	INPUT, OUTPUT, INPUT_PULLUP or INTERRUPT
5	GPIO 5	INPUT, OUTPUT, INPUT_PULLUP or INTERRUPT
6	GPIO 6	INPUT, OUTPUT, INPUT_PULLUP or INTERRUPT
7	GPIO 7	INPUT, OUTPUT, INPUT_PULLUP or INTERRUPT
8	GPIO 8	INPUT, OUTPUT, INPUT_PULLUP or INTERRUPT
9	GPIO 9	INPUT, OUTPUT, INPUT_PULLUP or INTERRUPT
10	GPIO 10	INPUT, OUTPUT, INPUT_PULLUP or INTERRUPT

11	GPIO 11	INPUT, OUTPUT, INPUT_PULLUP or INTERRUPT
12	GPIO 12	INPUT, OUTPUT, INPUT_PULLUP or INTERRUPT
13	GPIO 13	INPUT, OUTPUT, INPUT_PULLUP or INTERRUPT
14	GPIO 14	INPUT, OUTPUT, INPUT_PULLUP or INTERRUPT
15	GPIO 15	INPUT, OUTPUT, INPUT_PULLUP or INTERRUPT
16	GPIO 16	INPUT, OUTPUT, INPUT_PULLDOWN

Important: at startup, pins are configured as INPUT.

Note: pins may also serve other functions, like Serial, I2C, SPI. These functions are normally activated by the corresponding library.

OUTPUT mode

Using ***digitalWrite()*** 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):

```
int pin = 2;
pinMode(pin, OUTPUT);
digitalWrite(pin, HIGH);
```

Here is how to make the pin GPIO 2 put out a LOW (0V):

```
int pin = 2;
pinMode(pin, OUTPUT);
digitalWrite(pin, LOW);
```

INPUT mode

Using *digitalRead()* you can read the current state of any GPIO. For example, here is how you would check if a button was pressed.

```
int pin = 2;
pinMode(pin, INPUT);
digitalRead(pin);
```

If *digitalRead(pin)* = 1 the button was being pressed and if *digitalRead(pin)* = *0* the button was released, or was not pressed.

INTERRUPT mode

Pin interrupts are supported in the ESP through *attachInterrupt()* and *detachInterrupt()* functions.

Similar to the Arduino. Interrupts may be attached to any GPIO pin, except GPIO 16. Standard Arduino interrupt types are supported: CHANGE, RISING and FALLING. More information can be found here:

- https://www.arduino.cc/en/Reference/AttachInterrupt
- https://www.arduino.cc/en/Reference/DetachInterrupt

Analog Input

The ESP-01 doesn't offer any analog inputs.

On the other hand, other versions such as the ESP-12 has a single ADC channel available to users. It may be used either to read voltage at ADC pin, or to read module supply voltage (VCC).

To read external voltage applied to ADC pin, use *analogRead(A0)*.

Important: Input voltage range is 0 — 1.0V.

To read VCC voltage, ADC pin must be kept unconnected. Additionally, the following line has to be added to the sketch:

ADC_MODE(ADC_VCC);

This line has to appear outside of any functions, for instance right after the #include lines of your sketch.

Analog Output

analogWrite(pin, value) enables software PWM on the given pin. PWM may be used on pins 0 to 16. Call *analogWrite(pin, 0)* to disable PWM on the pin. value may be in range from 0 to PWMRANGE, which is equal to 1023 by default. PWM range may be changed by *callinganalogWriteRange(new_range)*.

PWM frequency is 1kHz by default. Call *analogWriteFreq(new_frequency)* to change the frequency.

Timing and Delays

millis() and *micros()* return the number of milliseconds and microseconds elapsed after reset, respectively.

delay(ms) pauses the sketch for a given number of milliseconds and allows WiFi and TCP/IP tasks to run. *delayMicroseconds(us)* pauses for a given number of microseconds.

Remember that there is a lot of code that needs to run on the chip besides the sketch when WiFi is connected. WiFi and TCP/IP libraries get a chance to handle any pending events each time the *loop()* function completes, OR when delay is called. If you have a loop somewhere in your sketch that takes a lot of time (>50ms) without calling *delay*(), you might consider adding a call to delay function to keep the WiFi stack running smoothly.

There is also a *yield()* function which is equivalent to *delay(0)*. The *delayMicroseconds* function, on the other hand, does not yield to other tasks, so using it for delays more than 20 milliseconds is not recommended.

Serial

Serial object works much the same way as on a regular Arduino. Apart from hardware FIFO (128 bytes for TX and RX) HardwareSerial has additional 256-byte TX and RX buffers. Both transmit and receive is interrupt-driven. Write and read functions only block the sketch execution when the respective FIFO/buffers are full/empty.

Serial uses UART 0, which is mapped to pins GPIO 1 (TX) and GPIO 3 (RX). Serial may be remapped to GPIO 15 (TX) and GPIO 13 (RX) by calling *Serial.swap()* after *Serial.begin()*. Calling swap again maps UART 0 back to GPIO 1 and GPIO 3.

Serial1 uses UART 1, TX pin is GPIO 2. UART 1 can not be used to receive data because normally it's RX pin is occupied for flash chip connection. To use *Serial1*, call *Serial1.begin(baudrate)*.

By default the diagnostic output from WiFi libraries is disabled when you call *Serial.begin()*. To enable debug output again,

call *Serial.setDebugOutput(true)*. To redirect debug output to *Serial1* instead, call *Serial1.setDebugOutput(true)*.

You also need to use *Serial.setDebugOutput(true)* to enable output from *printf()* function.

Both Serial and Serial1 objects support 5, 6, 7, 8 data bits, odd (O), even (E), and no (N) parity, and 1 or 2 stop bits. To set the desired mode, call *Serial.begin(baudrate,SERIAL_8N1)*, *Serial.begin(baudrate,SERIAL_6E2)*, etc.

Progmem

The Program memory features work much the same way as on a regular Arduino; placing read only data and strings in read only memory and freeing heap for your application. The important difference is that on the ESP8266 the literal strings are not pooled. This means that the same literal string defined inside aF("") and/or PSTR("") will take up space for each instance in the code. So you will need to manage the duplicate strings yourself.

WiFi (ESP8266WiFI library)

This is mostly similar to WiFi shield library. Differences include:

- *WiFi.mode(m)*: set mode to *WIFI_AP, WIFI_STA,* or *WIFI_AP_STA*.
- call *WiFi.softAP(ssid)* to set up an open network
- call *WiFi.softAP(ssid, password)* to set up a WPA2-PSK network (password should be at least 8 characters)
- *WiFi.macAddress(mac)* is for STA, *WiFi.softAPmacAddress(mac)* is for AP.

- *WiFi.localIP()* is for STA, *WiFi.softAPIP()* is for AP.
- *WiFi.RSSI()* doesn't work
- *WiFi.printDiag(Serial)* will print out some diagnostic info
- *WiFiUDP* class supports sending and receiving multicast packets on STA interface. When sending a multicast packet, replace *udp.beginPacket(addr,port*) with *udp.beginPacketMulticast(addr, port, WiFi.localIP()).* When listening to multicast packets, replace *udp.begin(port)* with *udp.beginMulticast(WiFi.localIP(), multicast_ip_addr,port).* You can *useudp.destinationIP()* to tell whether the packet received was sent to the multicast or unicast address. Also note that multicast doesn't work on softAP interface.

WiFiServer, *WiFiClient* and *WiFiUDP* behave mostly the same way as with WiFi shield library. Four samples are provided for this library. You can see more commands here:http://www.arduino.cc/en/Reference/WiFi

Ticker

Library for calling functions repeatedly with a certain period. Two examples included.

It is currently not recommended to do blocking IO operations (network, serial, file) from Ticker callback functions. Instead, set a flag inside the ticker callback and check for that flag inside the loop function.

EEPROM

This is a bit different from standard EEPROM class. You need to call *EEPROM.begin(size)* before you start reading or writing, size being the

number of bytes you want to use. Size can be anywhere between 4 and 4096 bytes.

EEPROM.write() does not write to flash immediately, instead you must call *EEPROM.commit()* whenever you wish to save changes to *flash*. *EEPROM.end()* will also commit, and will release the RAM copy of EEPROM contents.

EEPROM library uses one sector of flash located at 0x7b000 for storage.

I2C (Wire library)

Wire library currently supports master mode up to approximately 450KHz. Before using I2C, pins for SDA and SCL need to be set by calling *Wire.begin(int sda, int scl)*, i.e. *Wire.begin(0, 2)* on ESP-01, else they default to pins 4(SDA) and 5(SCL).

SPI

SPI library supports the entire Arduino SPI API including transactions, including setting phase (CPHA). Setting the Clock polarity (CPOL) is not supported, yet (SPI_MODE2 and SPI_MODE3 not working).

ESP-specific APIs

APIs related to deep sleep and watchdog timer are available in the ESP object, only available in Alpha version.

- *ESP.deepSleep(microseconds, mode)* will put the chip into deep sleep. mode is one of

WAKE_RF_DEFAULT, WAKE_RFCAL, WAKE_NO_RFCAL, WAKE_RF_DISABLED. (GPIO16 needs to be tied to RST to wake from deepSleep.)

- *ESP.restart()* restarts the CPU.
- *ESP.getFreeHeap()* returns the free heap size.
- *ESP.getChipId()* returns the ESP8266 chip ID as a 32-bit integer.

Several APIs may be used to get flash chip info:

- *ESP.getFlashChipId()* returns the flash chip ID as a 32-bit integer.
- *ESP.getFlashChipSize()* returns the flash chip size, in bytes, as seen by the SDK (may be less than actual size).
- *ESP.getFlashChipSpeed(void)* returns the flash chip frequency, in Hz.
- *ESP.getCycleCount()* returns the cpu instruction cycle count since start as an unsigned 32-bit. This is useful for accurate timing of very short actions like bit banging.
- *ESP.getVcc()* may be used to measure supply voltage. ESP needs to reconfigure the ADC at startup in order for this feature to be available. Add the following line to the top of your sketch to use *getVcc*:

ADC_MODE(ADC_VCC);

Important: TOUT pin has to be disconnected in this mode.

Note: that by default ADC is configured to read from TOUT pin using *analogRead(A0)* and *ESP.getVCC()* is not available.

OneWire

from https://www.pjrc.com/teensy/td_libs_OneWire.html

Library was adapted to work with ESP8266 by including register definitions into OneWire.h Note that if you already have OneWire library in your Arduino/libraries folder, it will be used instead of the one that comes with this package.

mDNS and DNS-SD responder (ESP8266mDNS library)

Allows the sketch to respond to multicast DNS queries for domain names like "foo.local", and DNS-SD (service discovery) queries. Currently the library only works on STA interface, AP interface is not supported. See attached example for details.

SSDP responder (ESP8266SSDP)

SSDP is another service discovery protocol, supported on Windows out of the box. See attached example for reference.

DNS server (DNSServer library)

Implements a simple DNS server that can be used in both STA and AP modes. The DNS server currently supports only one domain (for all other domains it will reply with NXDOMAIN or custom status code). With it clients can open a web server running on ESP8266 using a domain name, not an IP address. See attached example for details.

Servo

This library exposes the ability to control RC (hobby) servo motors. It will support upto 24 servos on any available output pin. By defualt the first 12 servos will use Timer0 and currently this will not interfere with any other support. Servo counts above 12 will use Timer1 and features that use it will be effected. While many RC servo motors will accept the 3.3V IO data pin from a ESP8266, most will not be able to run off 3.3v and will require another power source that matches their specifications. Make sure to connect the grounds between the ESP8266 and the servo motor power supply.

Other supported libraries (not included with the Arduino IDE)

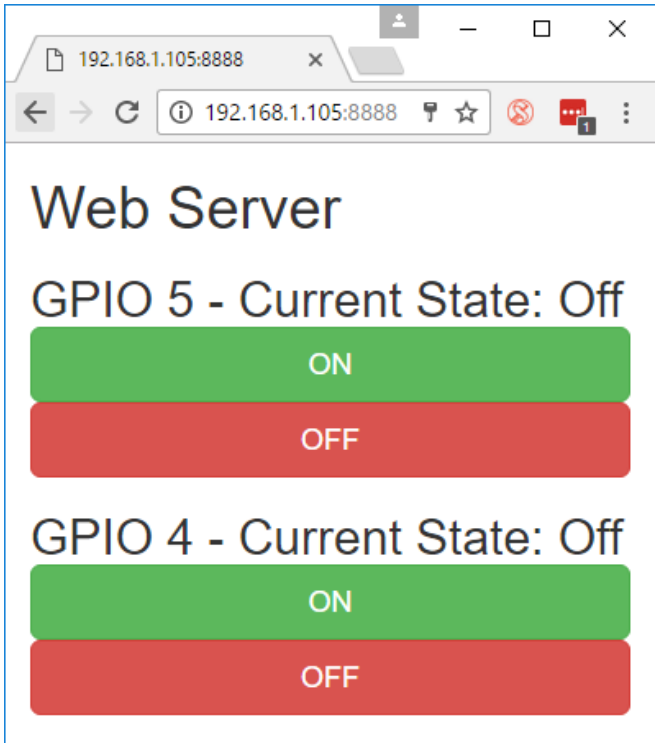
Libraries that don't rely on low-level access to AVR registers should work well. Here are a few libraries that were verified to work:

- Adafruit_ILI9341 - Port of the Adafruit ILI9341 for the ESP8266
- arduinoVNC - VNC Client for Arduino
- arduinoWebSockets - WebSocket Server and Client compatible with ESP8266 (RFC6455)
- aREST - REST API handler library.
- Blynk - easy IoT framework for Makers (check out the Kickstarter page).
- DallasTemperature
- DHT-sensor-library - Arduino library for the DHT11/DHT22 temperature and humidity sensors. Download latest v1.1.1 library and no changes are necessary. Older versions should initialize DHT as follows: DHT dht(DHTPIN, DHTTYPE, 15)
- DimSwitch - Control electronic dimmable ballasts for fluorescent light tubes remotely as if using a wall switch.
- Encoder - Arduino library for rotary encoders. Version 1.4 supports ESP8266.
- esp8266_mdns - mDNS queries and responses on esp8266. Or to describe it another way: An mDNS Client or Bonjour Client library for the esp8266.
- ESPAsyncTCP - Asynchronous TCP Library for ESP8266 and ESP32/31B

- ESPAsyncWebServer - Asynchronous Web Server Library for ESP8266 and ESP32/31B
- Homie for ESP8266 - Arduino framework for ESP8266 implementing Homie, an MQTT convention for the IoT.
- NeoPixel - Adafruit's NeoPixel library, now with support for the ESP8266 (use version 1.0.2 or higher from Arduino's library manager).
- NeoPixelBus - Arduino NeoPixel library compatible with ESP8266. Use the "DmaDriven" or "UartDriven" branches for ESP8266. Includes HSL color support and more.
- PubSubClient - MQTT library by @Imroy.
- RTC - Arduino Library for Ds1307 & Ds3231 compatible with ESP8266.
- Souliss, Smart Home - Framework for Smart Home based on Arduino, Android and openHAB.
- ST7735 - Adafruit's ST7735 library modified to be compatible with ESP8266. Just make sure to modify the pins in the examples as they are still AVR specific.
- Task - Arduino Nonpreemptive multitasking library. While similiar to the included Ticker library in the functionality provided, this library was meant for cross Arduino compatibility.
- TickerScheduler - Library provides simple scheduler for Ticker to avoid WDT reset
- Teleinfo - Generic French Power Meter library to read Teleinfo energy monitoring data such as consuption, contract, power, period, ... This library is cross platform, ESP8266, Arduino, Particle, and simple C++. French dedicated post on author's blog and all related information about Teleinfo also available.
- UTFT-ESP8266 - UTFT display library with support for ESP8266. Only serial interface (SPI) displays are supported for now (no 8-bit parallel mode, etc). Also includes support for the hardware SPI controller of the ESP8266.
- WiFiManager - WiFi Connection manager with web captive portal. If it can't connect, it starts AP mode and a configuration portal so you can choose and enter WiFi credentials.
- OneWire - Library for Dallas/Maxim 1-Wire Chips.
- Adafruit-PCD8544-Nokia-5110-LCD-Library - Port of the Adafruit PCD8544 - library for the ESP8266.
- PCF8574_ESP - A very simplistic library for using the PCF8574/PCF8574A I2C 8-pin GPIO-expander.
- Dot Matrix Display Library 2 - Freetronics DMD & Generic 16 x 32 P10 style Dot Matrix Display Library
- SdFat-beta - SD-card library with support for long filenames, software- and hardware-based SPI and lots more.

- FastLED - a library for easily & efficiently controlling a wide variety of LED chipsets, like the Neopixel (WS2812B), DotStar, LPD8806 and many more. Includes fading, gradient, color conversion functions.
- OLED - a library for controlling I2C connected OLED displays. Tested with 0.96 inch OLED graphics display.
- MFRC522 - A library for using the Mifare RC522 RFID-tag reader/writer.
- Ping - lets the ESP8266 ping a remote machine.

Unit 4
Password Protected Web Server



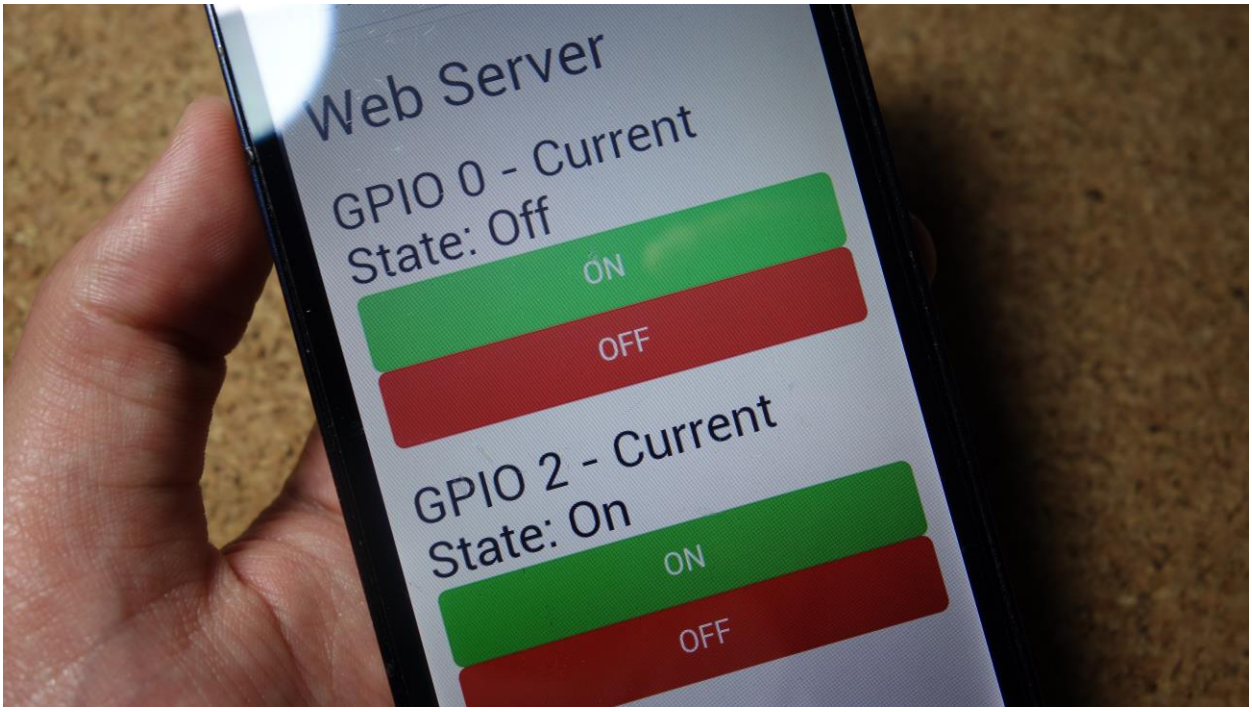
Unit 4 - Password Protected Web Server

In this Unit you're going to create a password protected web server with your ESP8266 that can be accessed with any device 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.

This web server will be password protected and accessible from anywhere in the world.

Spoiler Alert: this is what you're going to achieve at the end of this project!



Writing Your Arduino Sketch

Let's create a web server that controls two outputs (GPIO 5 and GPIO 4).

You can download the sketch for this project by opening the following link:

SOURCE CODE
https://gist.github.com/RuiSantosdotme/1e1f0f415a8da5fc901257296a9a0b68

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)* which refer to GPIO 5 and GPIO 4 respectively.

```
// variables
String header;
String gpio5_state = "Off";
String gpio4_state = "Off";
int gpio5_pin = 5;
int gpio4_pin = 4;
```

Now, let's go ahead and create your *setup()* function (it only runs once when your ESP first boots):

```
// only runs once
void setup() {

}
```

Start a serial communication at a 115200 baud rate for debugging purposes. Define your GPIOs as *OUTPUTs* and set them *LOW*.

```
// 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);
```

With the following code snippet: begins the WiFi connection, waits for a sucessful connection and prints the ESP 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(10000);

  // Printing the ESP IP address
  Serial.println(WiFi.localIP());
}
```

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.

```
// runs over and over again
void loop() {
  // Listening 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.

```
if (client) {
    Serial.println("New client");
    // boolean to locate when the http request ends
    boolean blank_line = true;
    while (client.connected()) {
      if (client.available()) {
```

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.

By default, your username is *user* and your password is *pass*. I'll show you how to change that in just a moment. If the user enters the correct username and password it shows the web page that controls the ESP.

```
// 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.

Let's see an example. When you click the button OFF from the GPIO 5 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*.

```
// 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);
  }
```

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/.

Your web page has four buttons to turn your LEDs *HIGH* and *LOW*. 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. 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*).

```
// 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>");
```

If you've entered the wrong credentials, it prints a text message in your browser saying "Authentication failed".

```
// wrong user or pass, 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>");
}
```

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

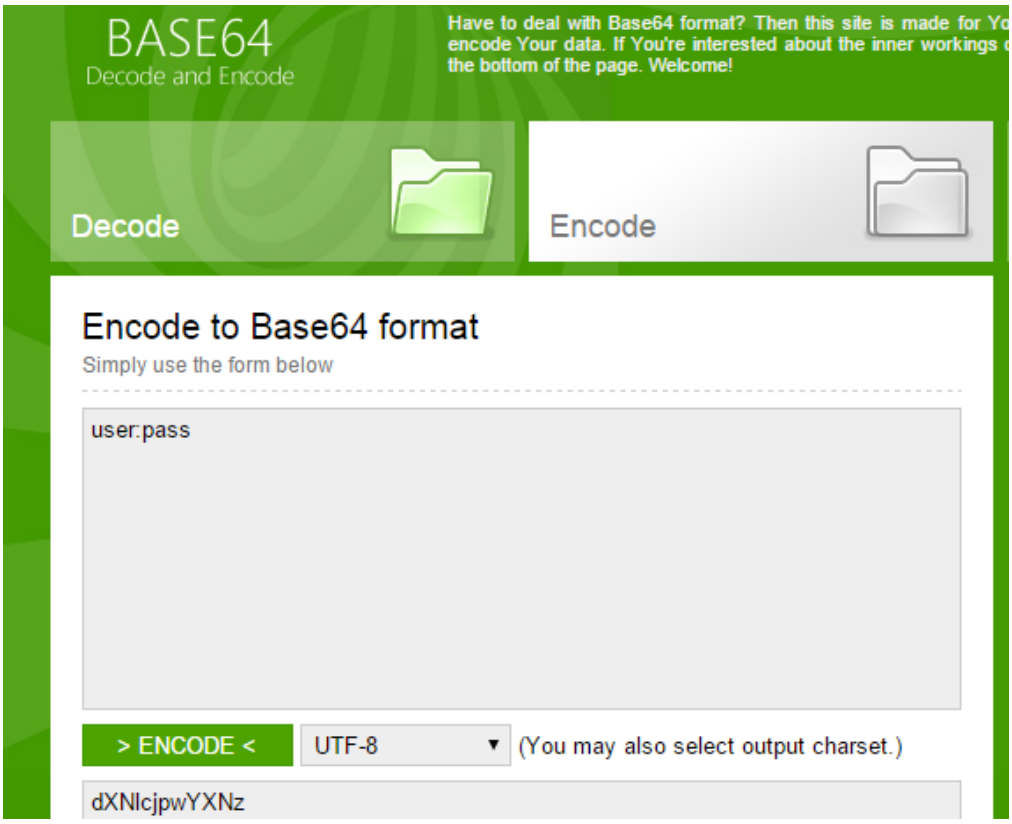
At this point if you upload the code written in the preceding section, your username is *user* and your password is *pass*. I'm sure you want to change and customize this example with your own credentials.

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 green "Encode" button to generate your base64 encoded string. In my example is *dXNlcjpwYXNz*.

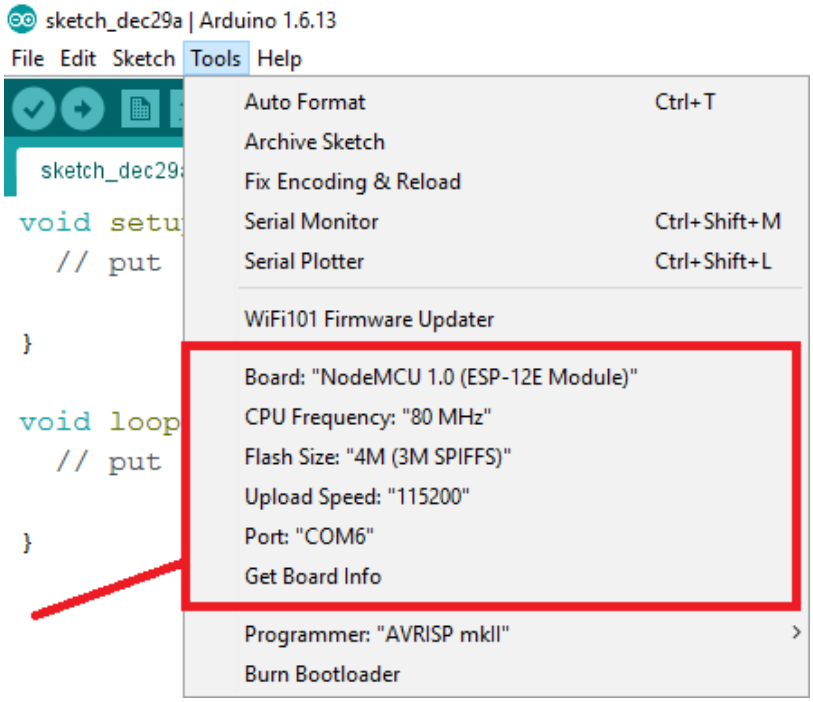
Copy your encoded string and replace it in your sketch:

```
if(header.indexOf("dXNlcjpwYXNz") >= 0)
```

Tip: you can find that *if* statement above on line 80 from this Gist: *https://gist.github.com/RuiSantosdotme/1e1f0f415a8da5fc901257296a9a0b68*

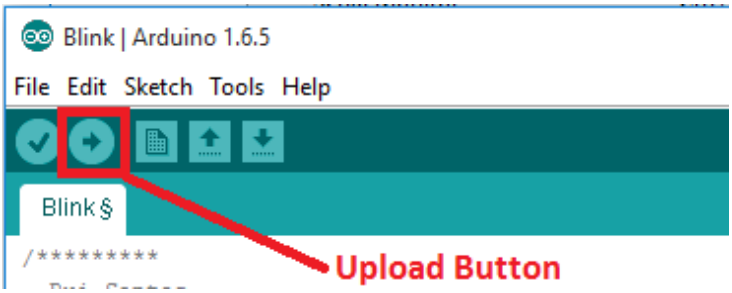
Uploading Code

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 look like this:

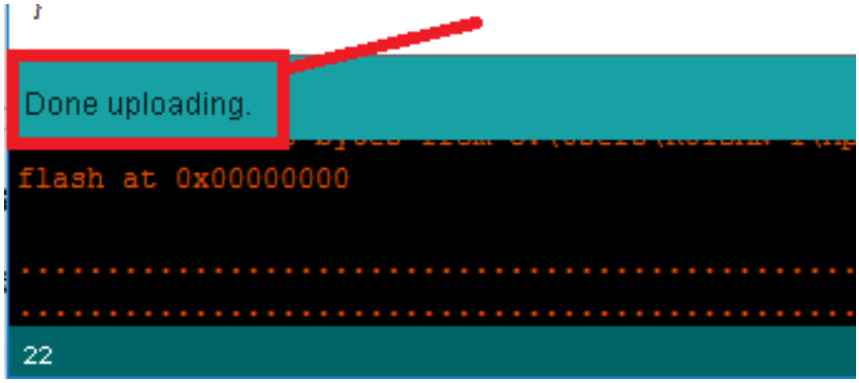


Important: your COM port is very likely to be different from the preceding screenshot (Port: "COM6"). That's fine, because it doesn't interfere with anything. On the other hand, all the other configurations should look exactly like mine.

After checking the configurations click the "**Upload Button**".

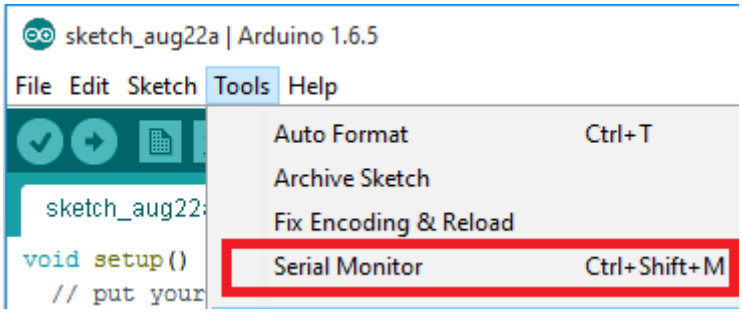


Wait a few seconds until you see the message "**Done uploading.**" in the bottom left corner.

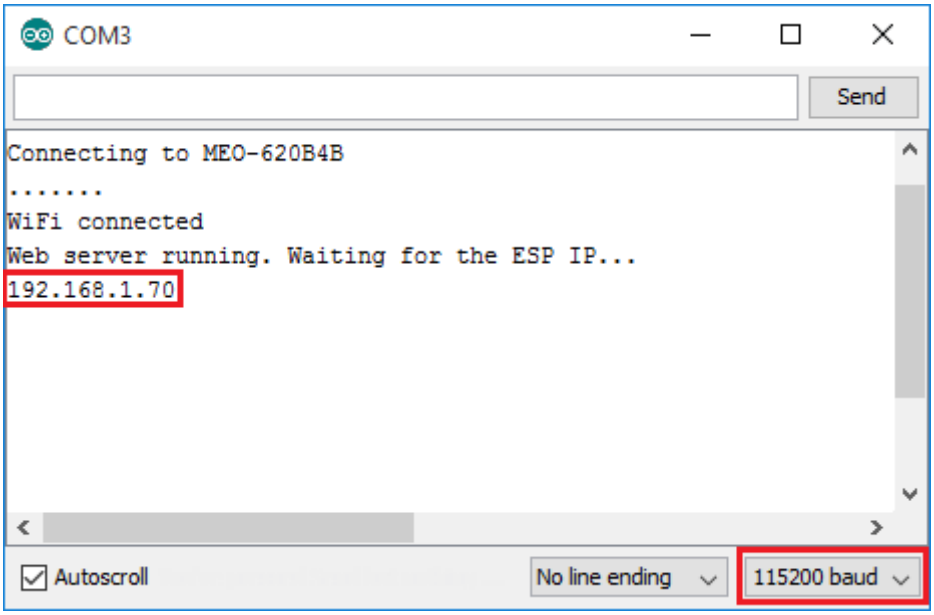


ESP8266 IP Address

After uploading your web server Sketch to your ESP, go to **Tools** > **Serial Monitor**. In your Serial Monitor window you're going to see your ESP IP address appearing when your ESP first boots.



My IP address is 192.168.1.70 (as shown in the figure below). Your IP should be different, **save your ESP8266 IP** so that you can access it later in this Unit.



Important: set the Serial Monitor baud rate to 115200, otherwise you won't see anything.

Troubleshooting: if your IP address doesn't show up in your Serial Monitor Here's what you need to do:

1. Let the Serial Monitor stay opened
2. The ESP-12E/FTDI Programmer should remain connected to your computer
3. Restart your ESP-12E with the on-board button

The IP address should appear in your Serial Monitor screen after 10 seconds.

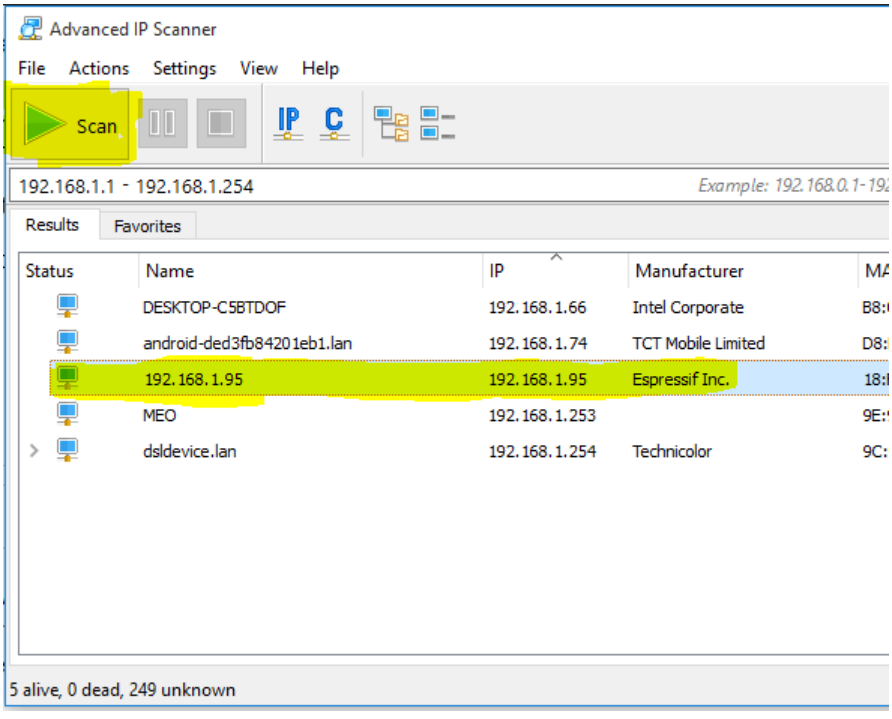
Install an IP Scanner Software

If you still can't see the ESP IP address, follow these next instructions to install an IP Scanner software:

1. An IP Scanner software searches for all the devices in your network
2. Download this free software:
    - Windows PC: www.advanced-ip-scanner.com
    - MAC OS X, Windows or Linux: http://angryip.org
3. Install one of these software (while having your ESP running with that web server script)
4. Open the IP Scanner software and click "Scan"

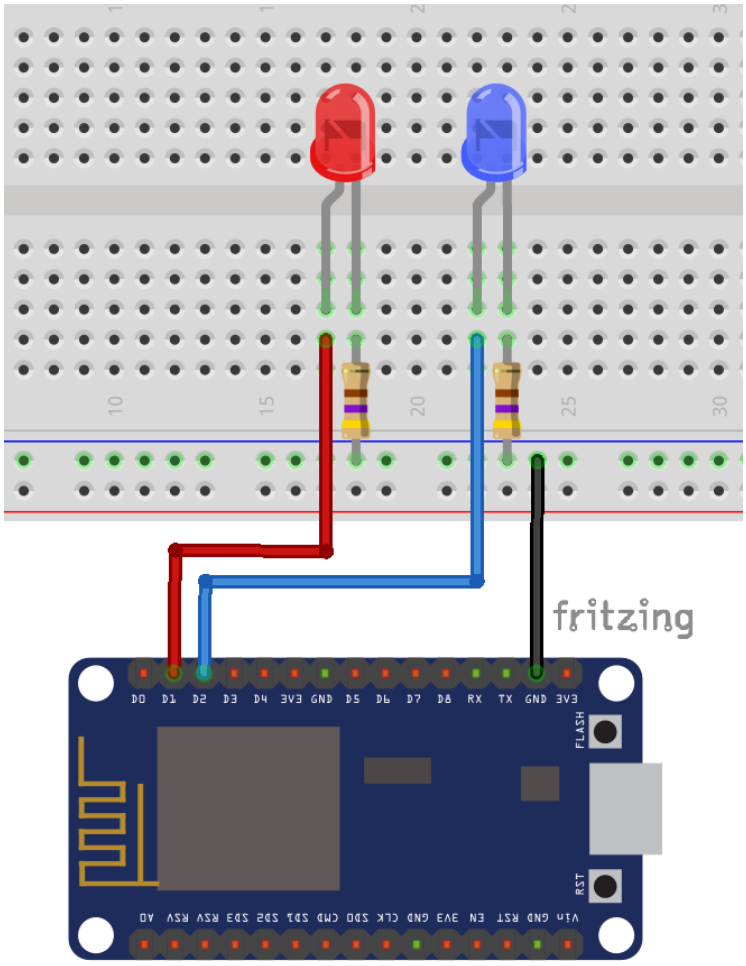
Let that process finish (it can take a couple of minutes).

In my case, it found my ESP. If I type 192.168.1.95:8888 in my browser, I can see the ESP web server.



Final Circuit

After uploading your code to your ESP8266, follow the next schematics (you can use resistors between 270 and 470 ohm for the LEDs).

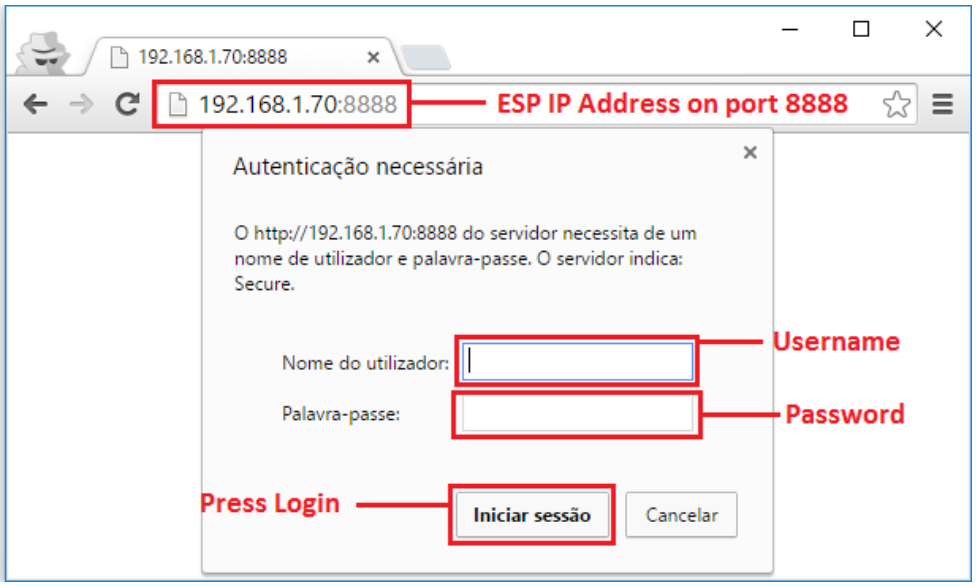


Accessing Your Web Server

Now follow the next instructions before accessing your web server:

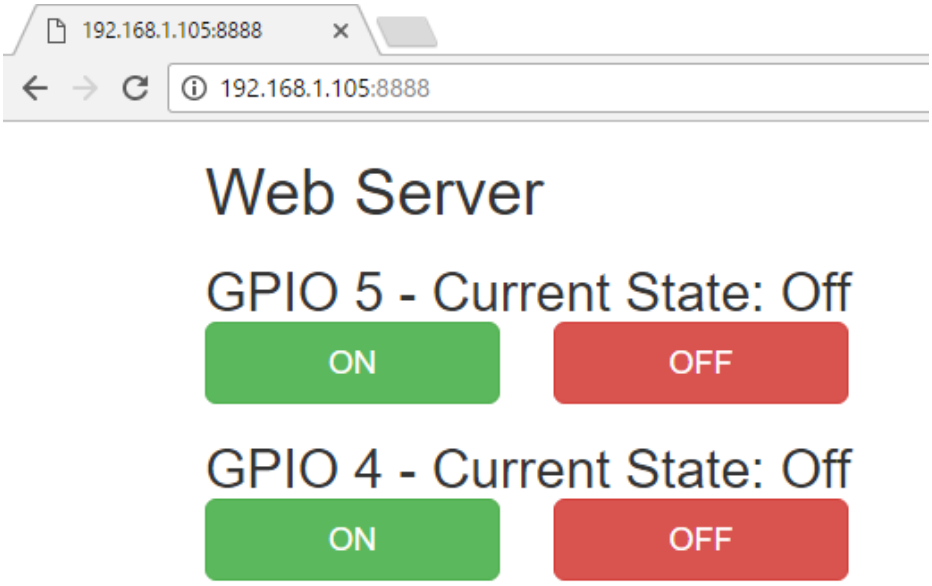
1. Restart your ESP8266 module
2. Open a browser
3. Type the IP address that you've previously saved in the URL bar followed by :8888 (in my case: http://192.168.1.70:8888)

It should require that you enter your username and password in order to open your web server. This is what you should see:



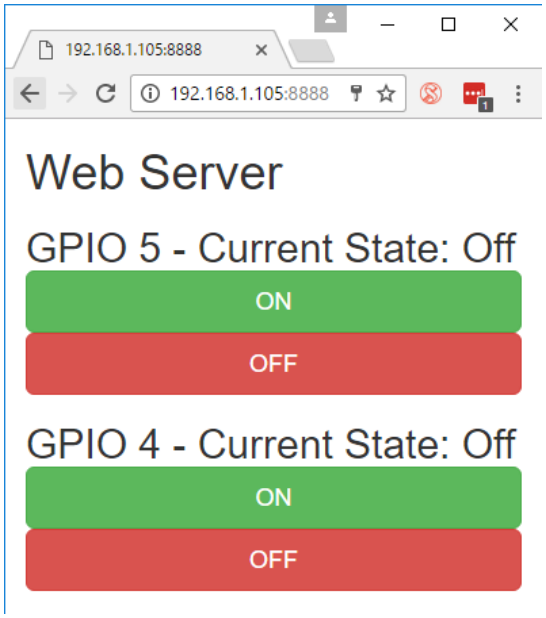
4. Enter your username and password
5. Press Login

A web page like the one below should load:



Note: at this point, 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 page is amazing.



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. You have three options…

Option A – PowerSwitch Tail II

The easiest route is to get yourself a PowerSwitch Tail II (www.powerswitchtail.com), which provides a safe way of dealing with high-voltage devices

The way this bulky component works is quite straightforward. Rather than connecting a house appliance directly to the wall, you connect it to the PowerSwitch Tail II which plugs into the wall.

The PowerSwitch Tail II has three pins that enable it to behave like a simple digital logic device. You connect the PowerSwitch Tail to an output GPIO of the ESP8266.

Your output pin will send a signal that's either HIGH or LOW. Whenever the signal is HIGH, there's a connection to the wall socket; when it's LOW, the connection is broken, as though the device were unplugged.

Here's how you should connect your PowerSwitch Tail II

PowerSwitch Tail II Pin Number	Signal Name	ESP-12E Pins
1	+in	GPIO 5 or GPIO 4 for example
2	-in	GND
3	GND	Not used

Search for the PowerSwitch Tail II's instruction sheet that comes with the device for more details on how to wire it up.

Option B – Relay

There's another way to have your ESP8266 control a house appliance, but that method is a bit more complicated. It requires a bit of extra knowledge and wariness because you're dealing with alternating current (AC), and it involves relay modules.

I won't discuss this project in great detail, but here's a good tutorial: http://randomnerdtutorials.com/guide-for-relay-module-with-arduino/

The preceding link takes you to my blog and shows how to control relays using an Arduino, but you can apply the same concepts to your ESP module.

WARNING: always be safe when dealing with high voltages, if you don't know what you're doing ask someone who does.

Option C – Remote controlled sockets

With this option you're going to build a web server with an ESP8266 that can control remotely any sockets (safely!).

Parts required

- 1x ESP8266 eBay: http://ebay.to/1HkFXB1
- 1x FTDI Programmer eBay: http://ebay.to/1EQQWjA
- 1x Arduino UNO eBay: http://ebay.to/1EJbhZE
- 1x 433MHz Receiver/Transmitter module eBay: http://ebay.to/1LHyroq
- Remote and sockets that work at 433MHz eBay: http://ebay.to/1KkYpvp

Remote controlled sockets (433MHz)

You can buy remote controlled sockets in any store or you can buy them on eBay. Keep in mind that they need to communicate via RF at 433MHz.

Here's my setup:

- Remote control – channel I
- Socket 1 – channel I and mode 1
- Socket 2 – channel I and mode 3

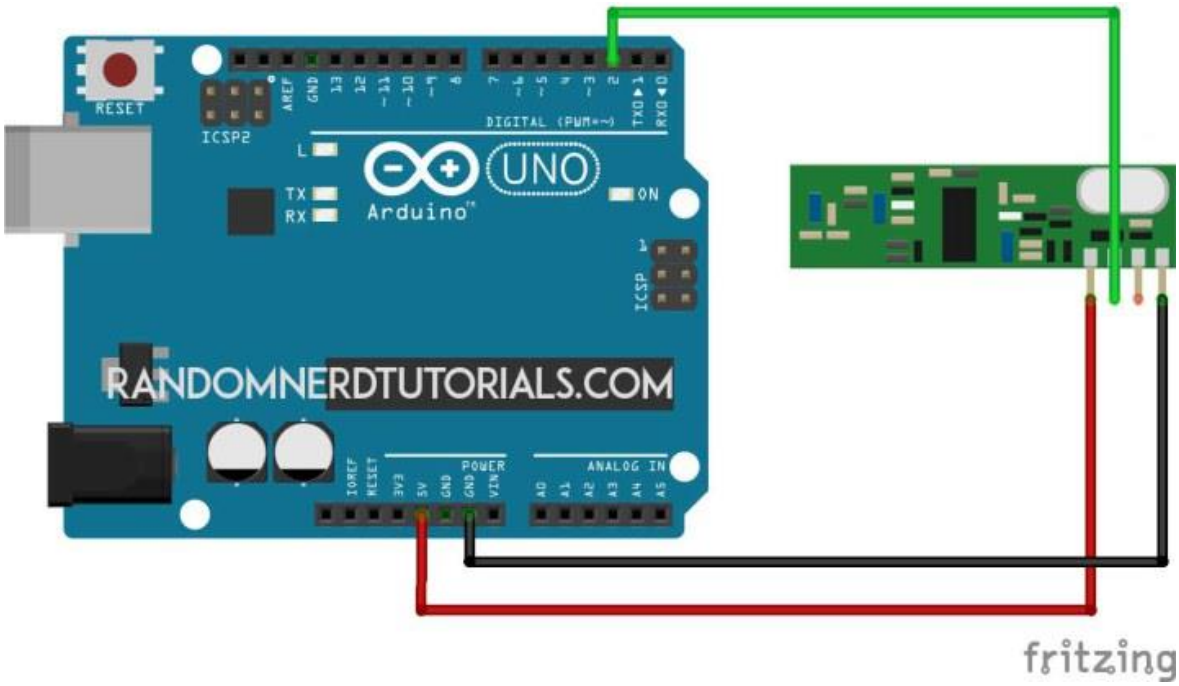
RC Switch library download

Here's the Arduino library you need for this project:

1. Download the RC Switch library
2. Unzip the RC Switch library
3. Remove the "-" from the folder name, otherwise your Arduino IDE won't recognize your library
4. Install the RC Switch library in your Arduino IDE
5. Restart your Arduino IDE

The RC Switch library is great and it works with almost all remote controlled sockets in the market.

Receiver circuit



Follow the circuit above for your receiver. Then upload the code below or you can go to **File** > **Examples** > **RC Switch** > **ReceiveDemo_Advanced**.

SOURCE CODE

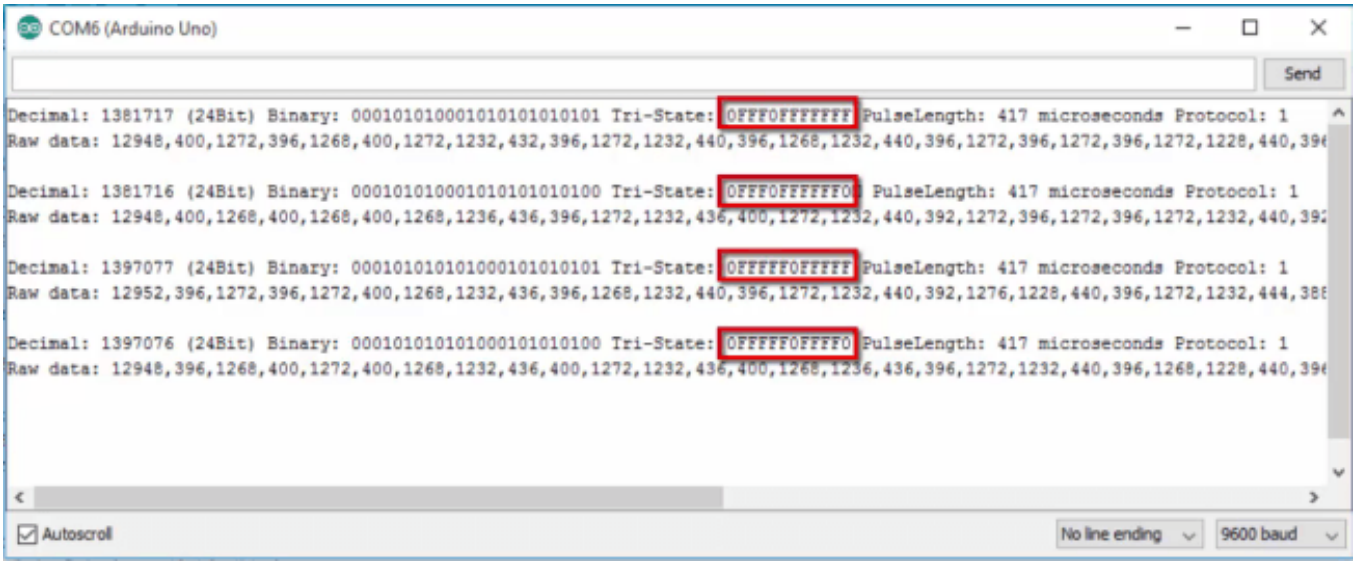
https://gist.github.com/RuiSantosdotme/007053c5bb307829ff81a29fd3be0453

```
#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();
  }
}
```

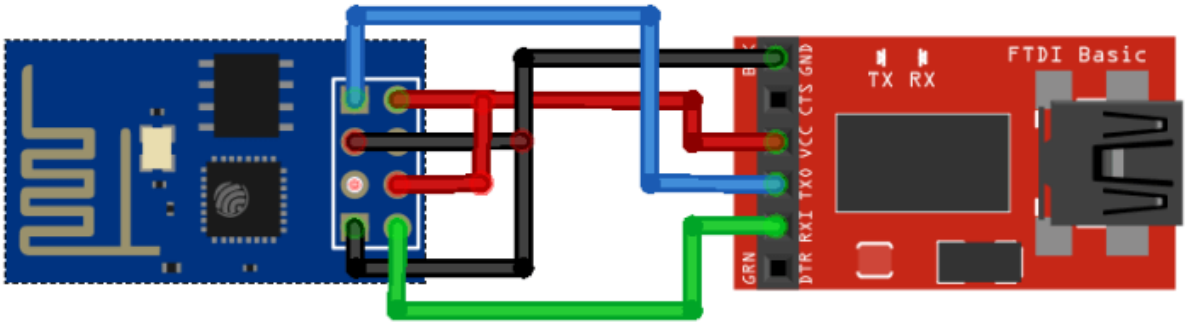
Save the TriState values

Open your Arduino 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.



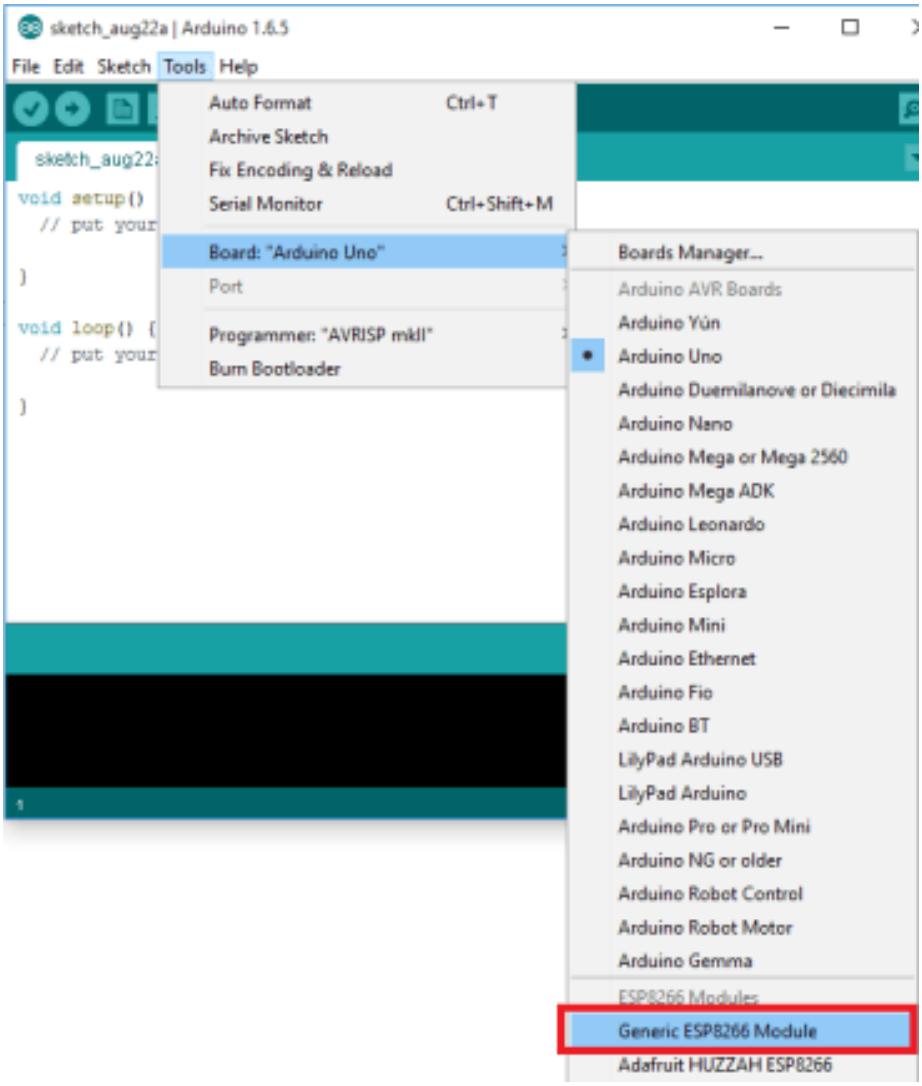
Schematics (3.3V FTDI Programmer)

To upload code to your ESP-01 you need to establish a serial communication between your FTDI programmer and your ESP-01 (as explained in previous Units).



Uploading code

In your Arduino IDE, go to **Tools** and select **"Board: Generic ESP8266 Module"**.



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.

SOURCE CODE

https://gist.github.com/RuiSantosdotme/415cadae73f877bb6a08f76f9a21ca07

```
#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 = "0FFF0FFFFFFF";
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> <a
href=\"socket1Off\"><button>OFF</button></a></p>";
  webPage += "<p>Socket #2 <a
href=\"socket2On\"><button>ON</button></a> <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();
}
```

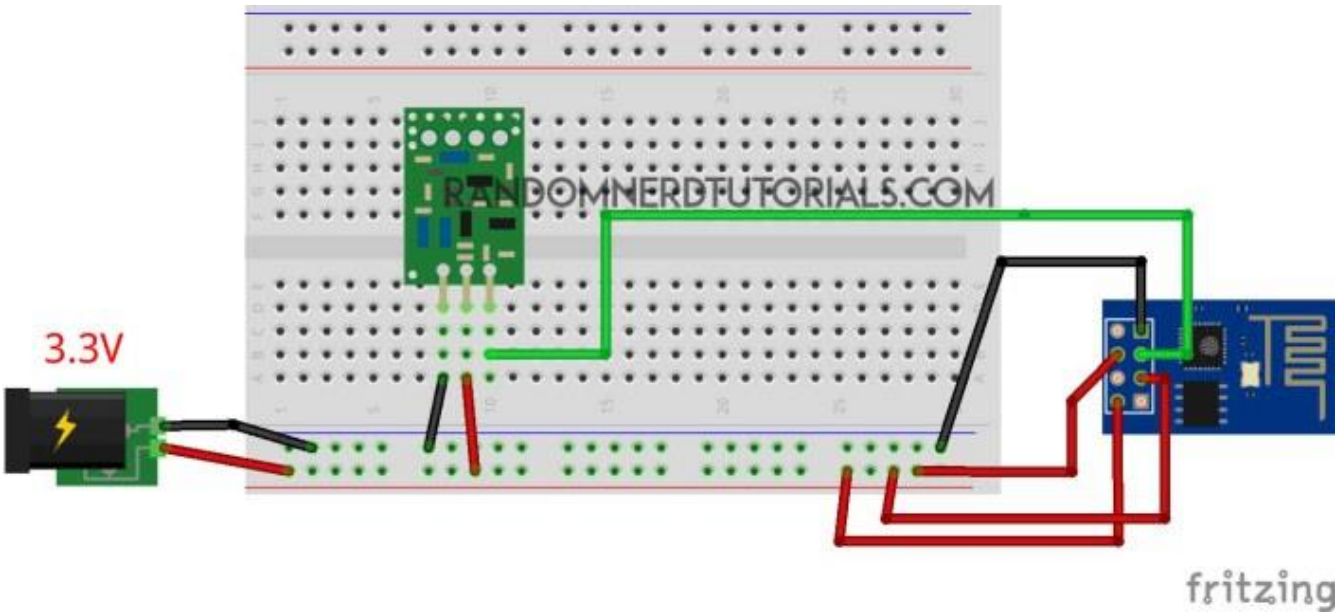
ESP8266 IP address

Open the Arduino IDE Serial monitor at a baud rate of 115200. Connect GPIO 0 of your ESP8266 to VCC and reset your board.

After a few seconds your IP address should appear. In my case, it's **192.168.1.70:8888**.

Final circuit

This is the final circuit for your ESP8266 that hosts a web server and transmits RF signals to control your sockets.

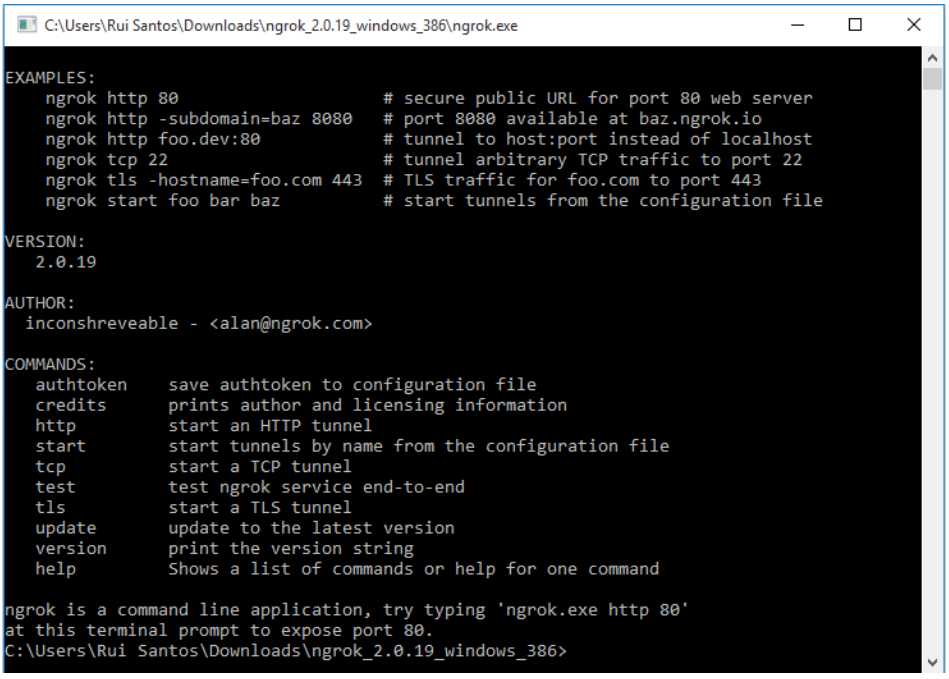


Demonstration

For the final demonstration open any browser from a device that is connected to the same router that your ESP is. Type the IP address and press Enter.

When you press the buttons in your web server, you can control both sockets on and off.

Unit 5
Making Your Web Server Accessible from Anywhere in the World

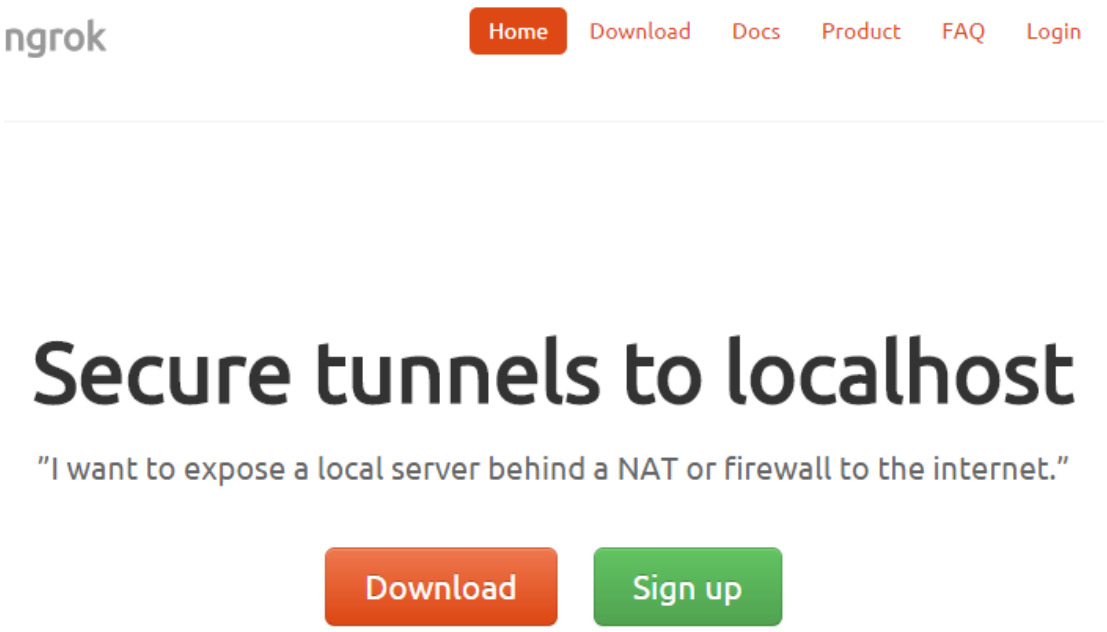


Unit 5 - Making Your Web Server Accessible from Anywhere in the World

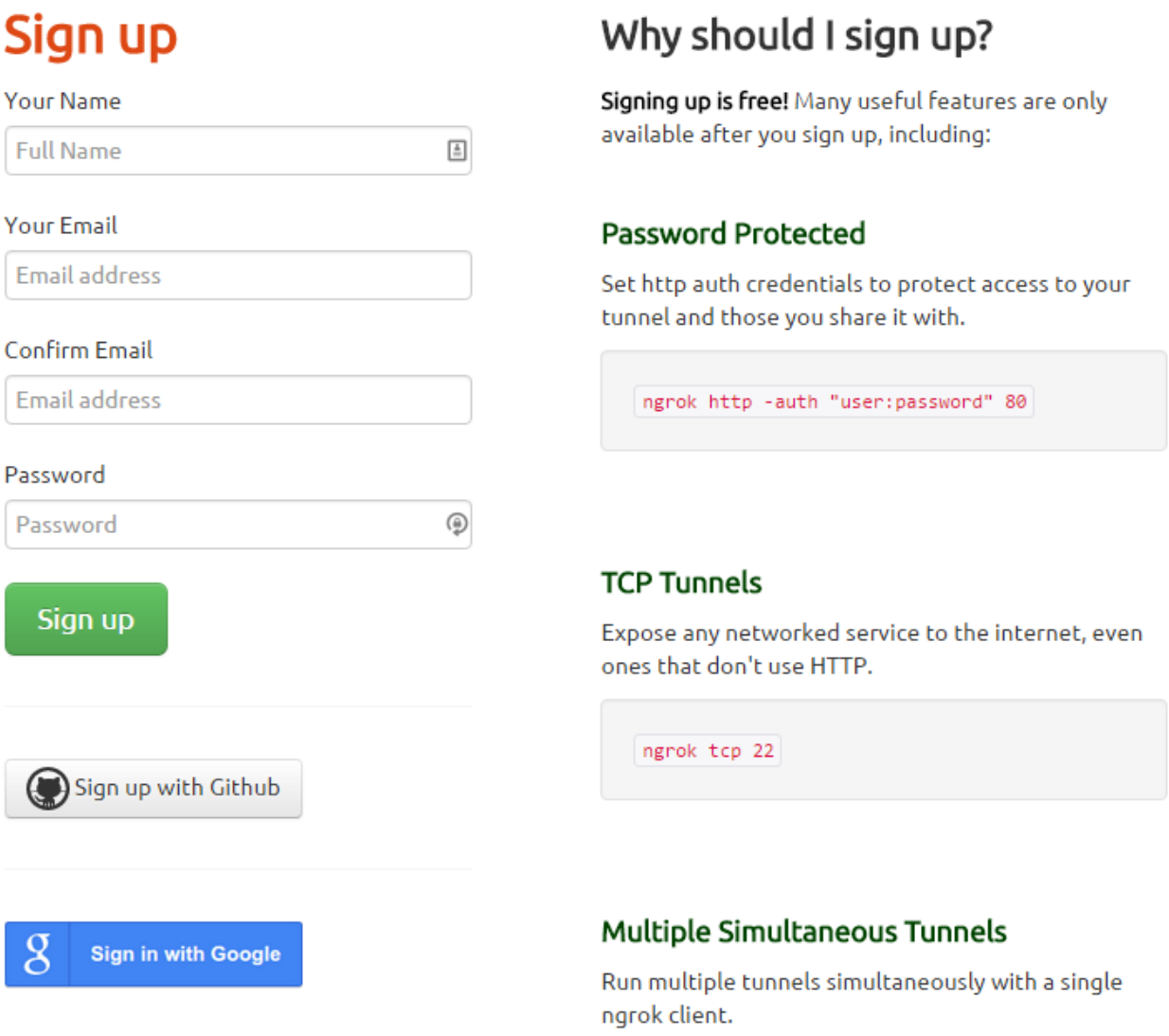
In this Unit you're going to make your web server accessible from anywhere in the world. You'll be using a free service to create a secure tunnel to your ESP which is running in your localhost.

Creating nkgrok Account

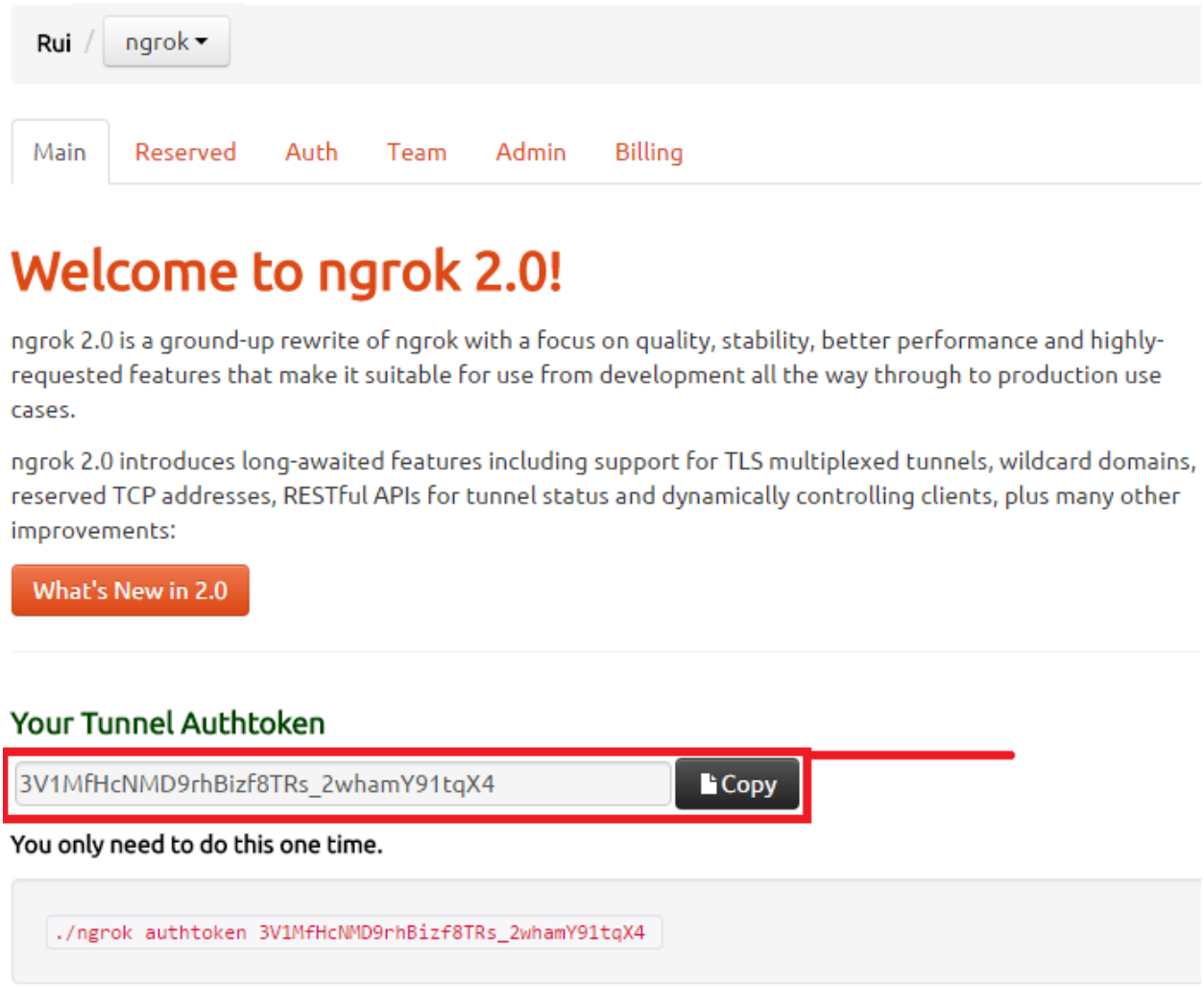
This service is called ngrok and it's free. Go to https://ngrok.com to create your account. Click the green "Sign up" button:



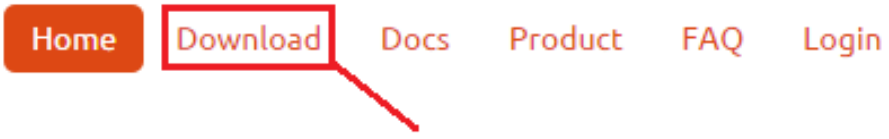
Enter your details in the forms in the left box (as shown in the figure below).



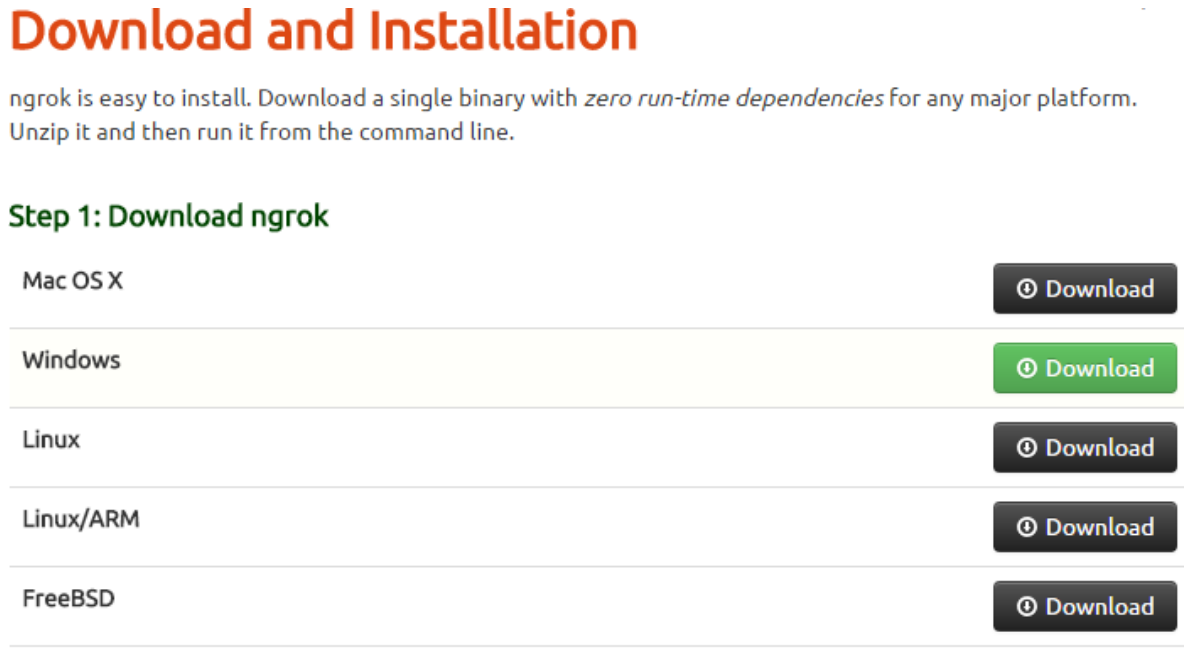
After creating your account, login and go to the main dashboard to find your Tunnel Authtoken. Copy your unique Authtoken to a safe place (you'll need it later in this Unit).



Go to the Download tab in the navigation bar:

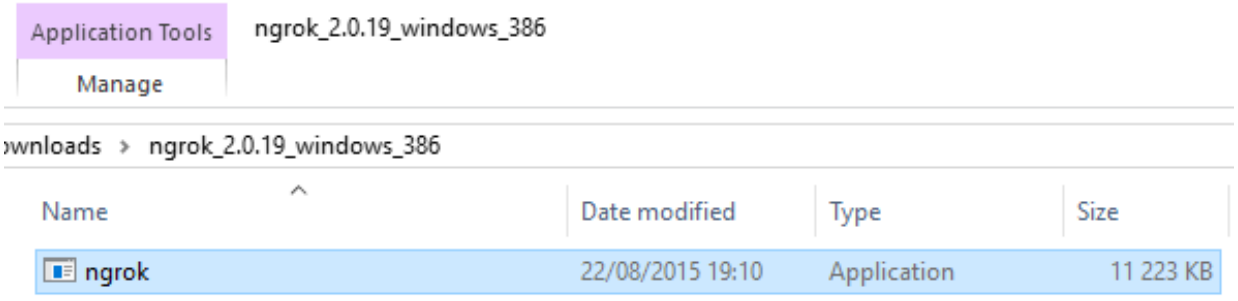


Then, choose your operating system and download the ngrok software.

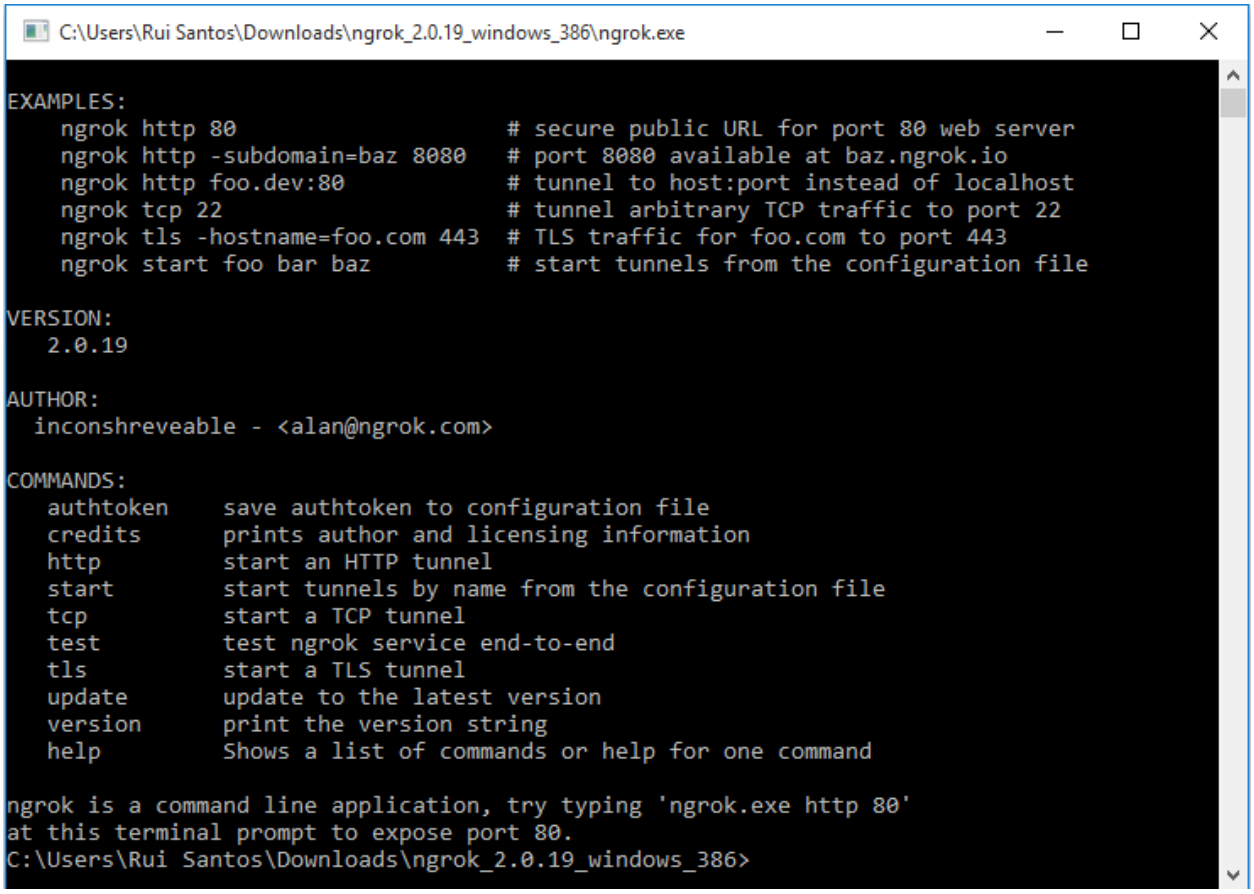


Unzip the folder that you have just downloaded and open the ngrok software.

Note: if you're on Linux open the terminal and type *./ngrok –help* to open the ngrok software.



You should see a window that looks like this:

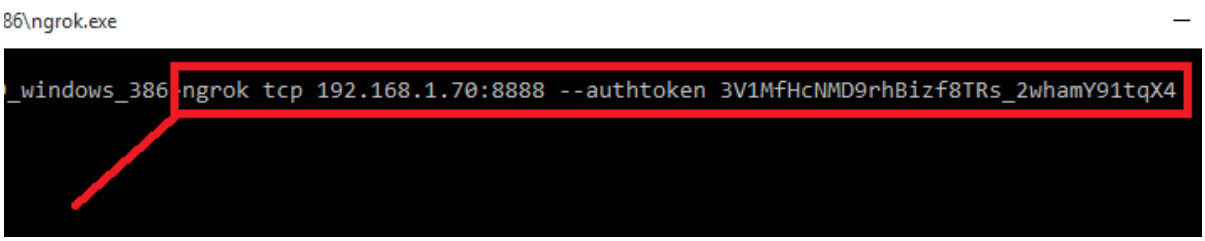


Launching ngrok Secure Tunnel

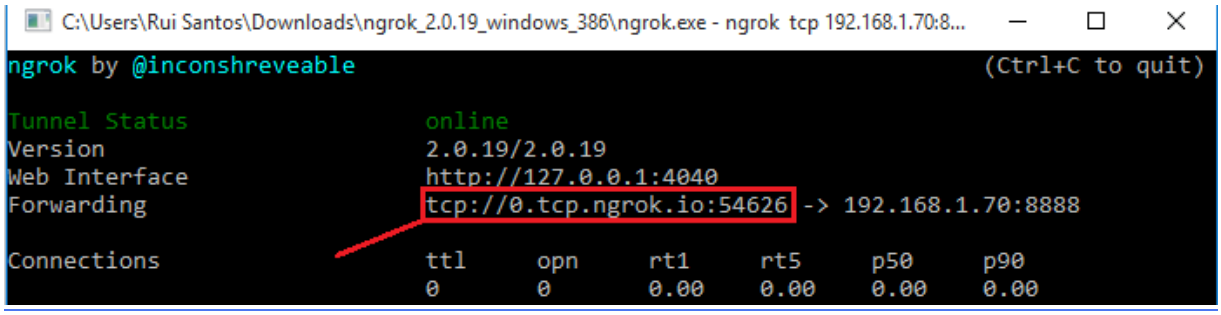
Now in your terminal enter the following command and replace the red text with your own IP address and ngrok's tunnel Authtoken:

ngrok tcp 192.168.1.70:8888 --authtoken 3V1MfHcNMD9rhBizf8TRs_2whamY91tqX4

Here's how it looks like, 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.



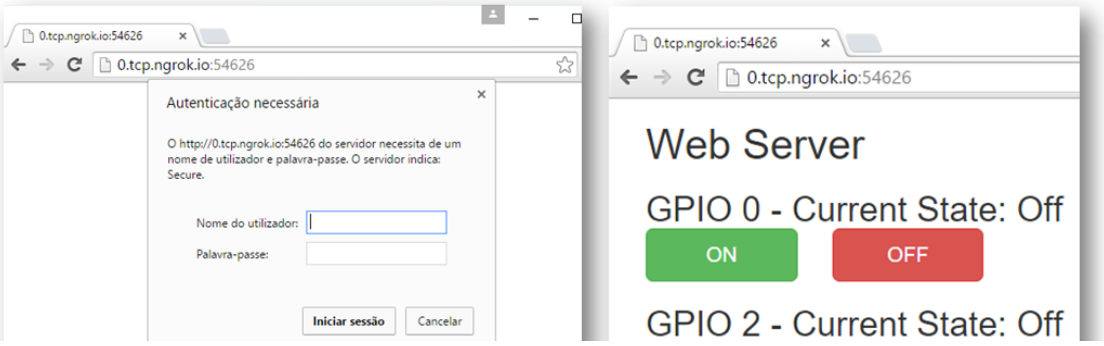
You can access your web server from anywhere in the world by typing your unique URL (in my case http://0.tcp.ngrok.io:54626/) in a browser.

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 ESP 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
Sonoff the $5 WiFi Wireless Smart Switch

Unit 6 - Sonoff the $5 WiFi Wireless Smart Switch

In this Unit you're going to learn 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

In the figure below, you can see the basic Sonoff.

You can get a Sonoff for $5 on eBay: http://ebay.to/2eNzHf2.

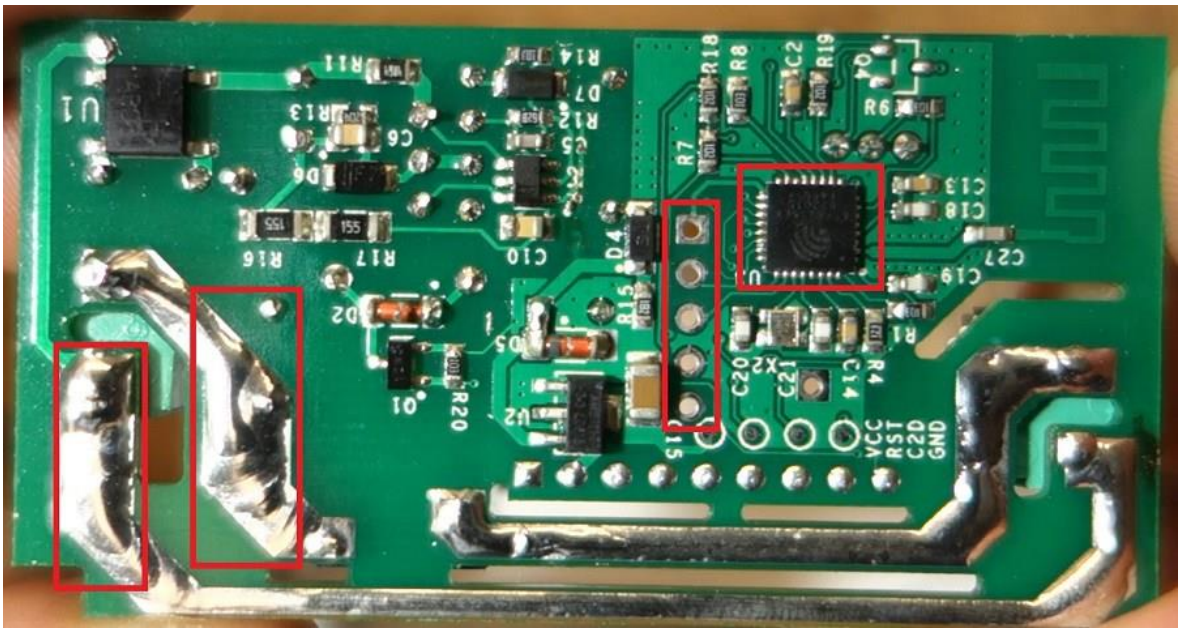
It's very simple, it has an input in one side and an output on the other side.

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 take a 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 that those **5 connections** were left out, so that you can solder some pins and upload a custom firmware

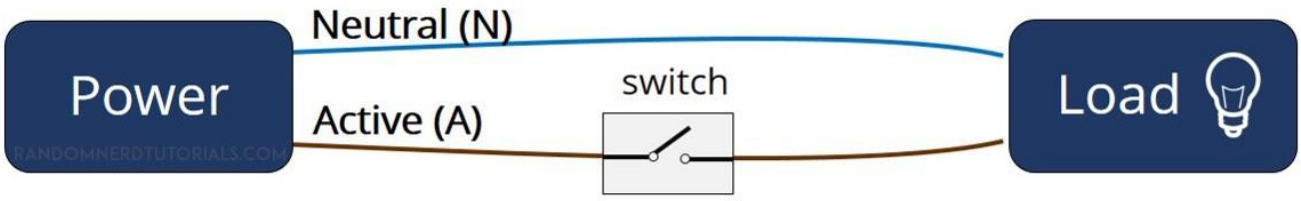


In the first part of this Unit you're going to use the standard firmware that comes with the Sonoff. Later, I'm going to show how to flash a custom firmware into the Sonoff device.

Sonoff Example

Let's take a look at 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.

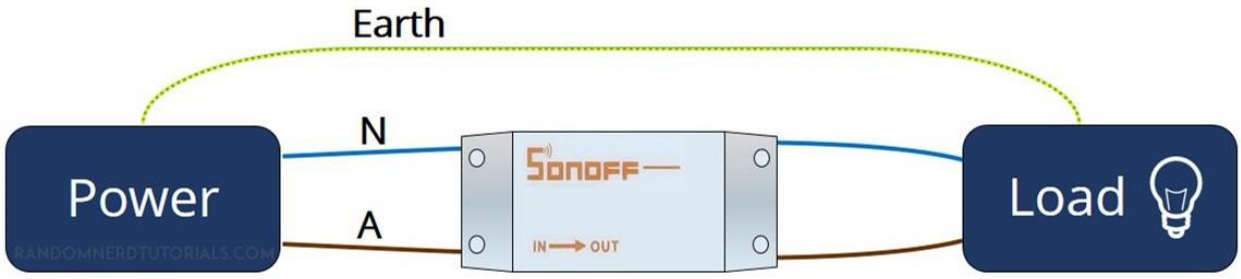
Normally, what you have is a power source that has an active and neutral line that goes to a load, your load can be lamp for example. In the middle, you usually have a switch.



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 has to go outside the Sonoff. In my case, I don't have earth in my home.

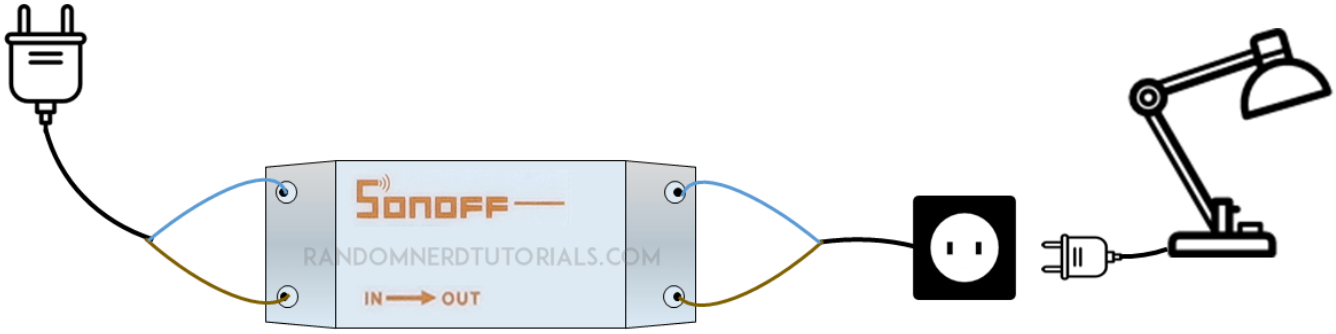


Safety Warning

Before proceeding with this project, I 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.

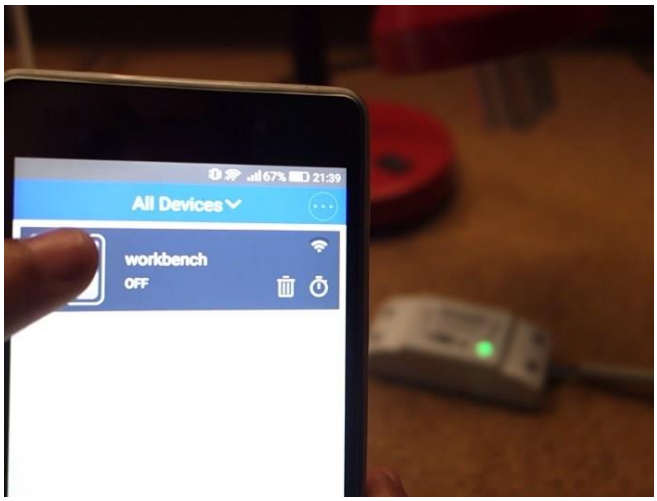
Installing the App

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.



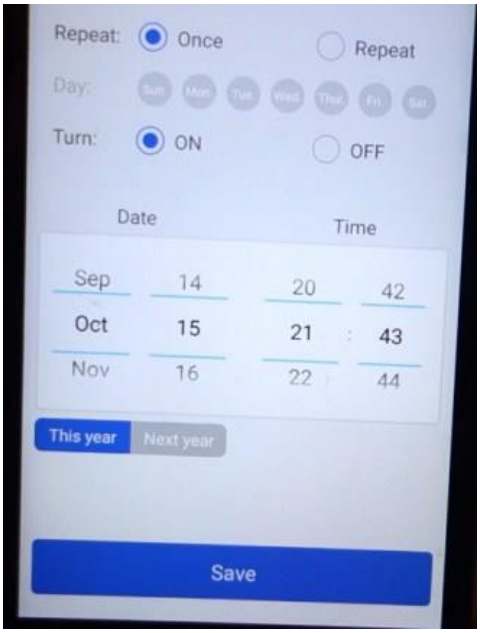
Watch the video to see a live demo of the Sonoff device:

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, click the timer button. 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

In order to use the Sonoff with your own web server, you have to flash custom firmware in the Sonoff device.

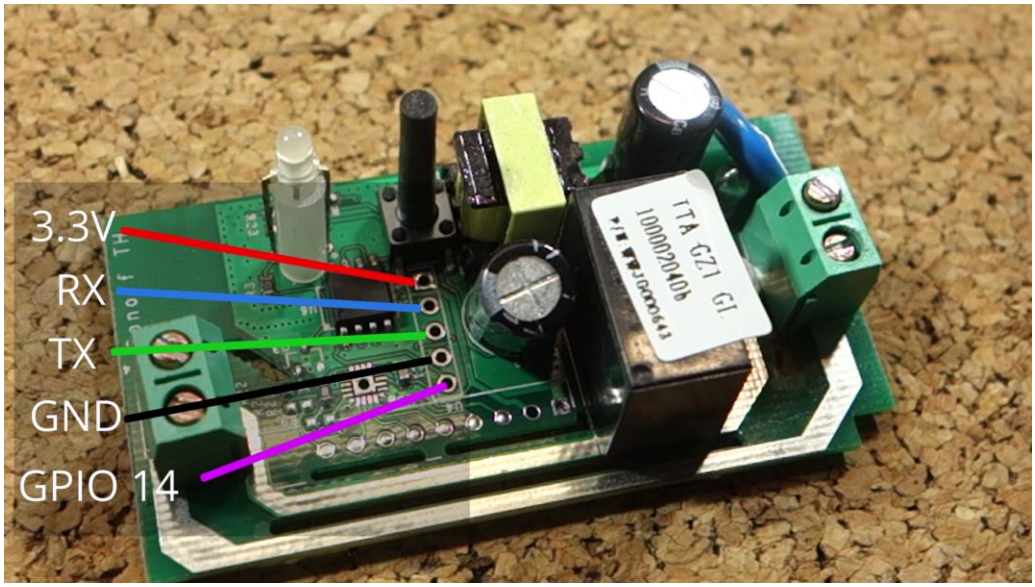
Safety Warning

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.

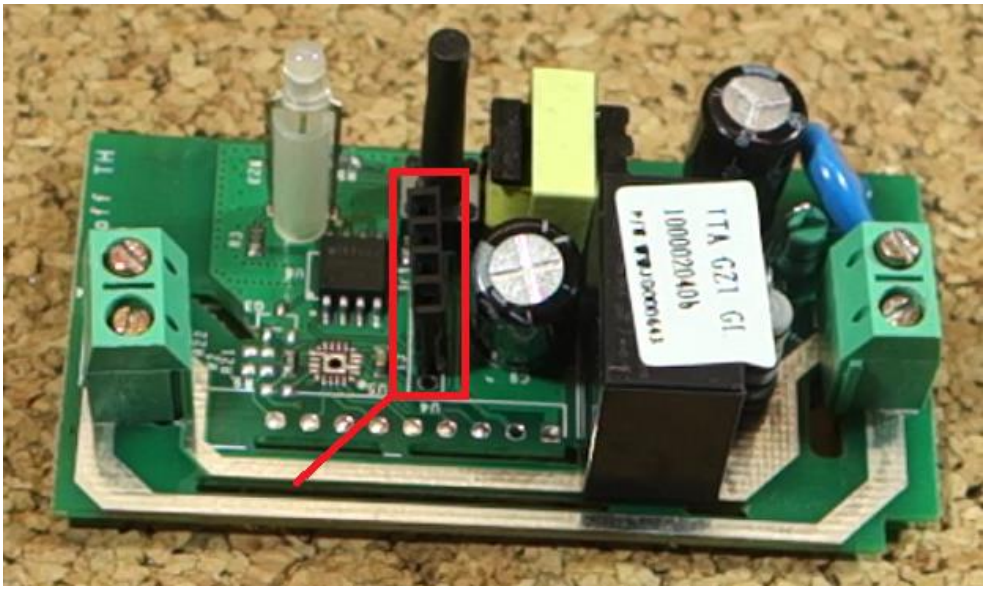
Sonoff Pinout

Open the Sonoff enclosure. As I mentioned earlier, the Sonoff is meant to be hacked, and you can see clearly that these connections were left out, so that you can solder some pins and upload a custom firmware.

That's the pinout that you need to worry about.



I soldered 4 header pins, so that I can easily connect and disconnect wire cables to my Sonoff device.



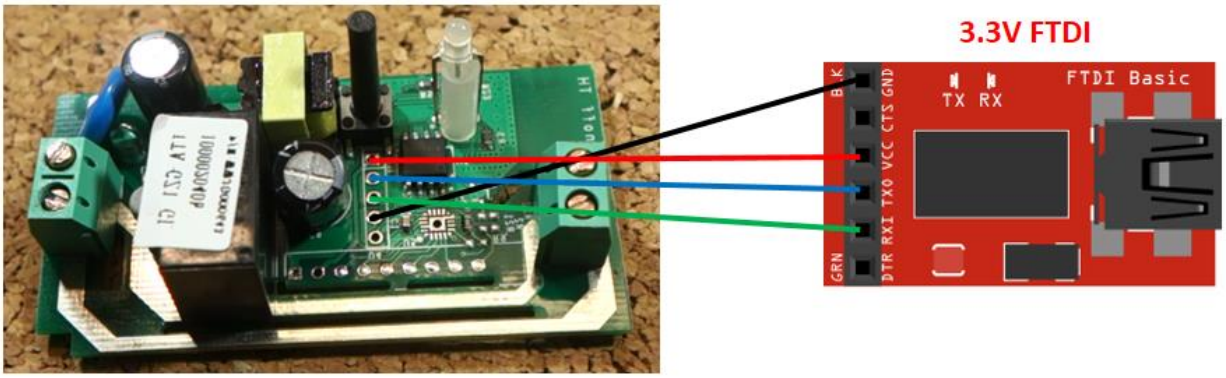
You need an FTDI module to upload a new firmware to your Sonoff.

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

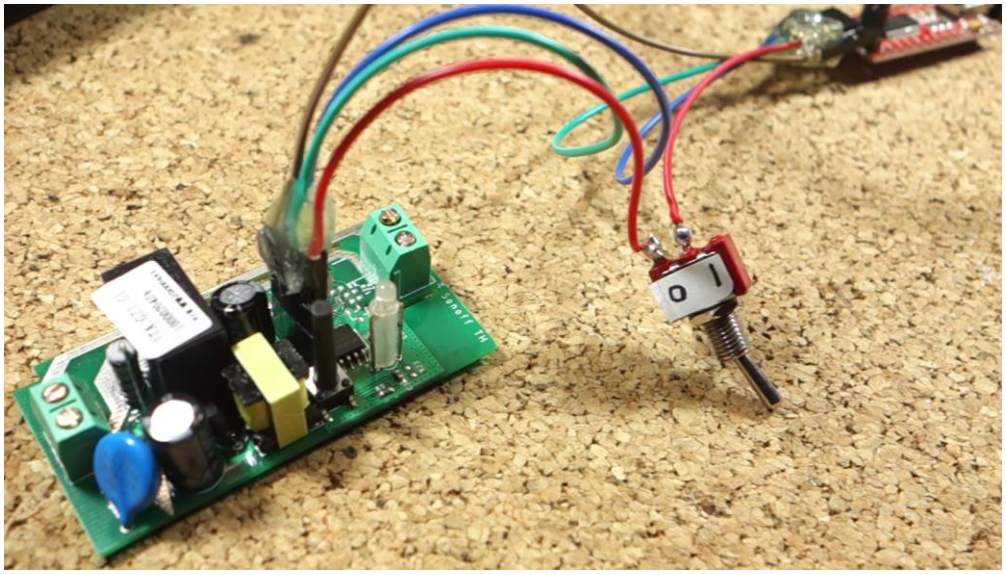
You should follow these next schematics to connect your FTDI programmer to your Sonoff device.

- 3.3V –> 3.3V
- TX -> RX
- RX -> TX
- GND -> GND



I've added a toggle switch in the power line (3.3V), so that I 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.

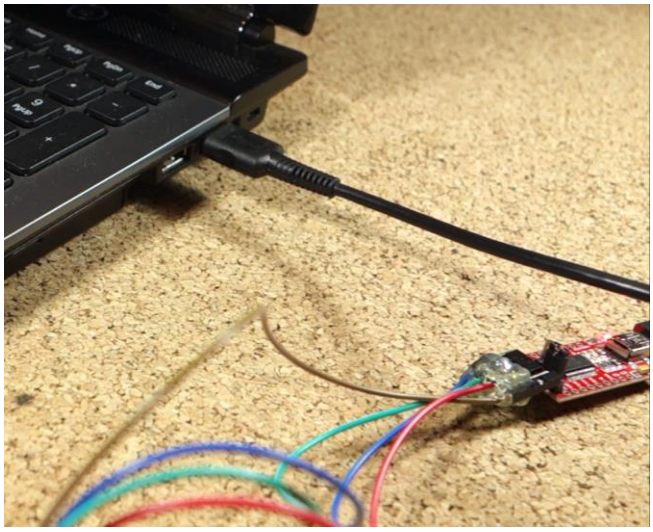


If you look closely to the previous figure, I used hot glue to glue the ends of the wires together. This prevents you to make wrong connections between the FTDI and the Sonoff in the future.

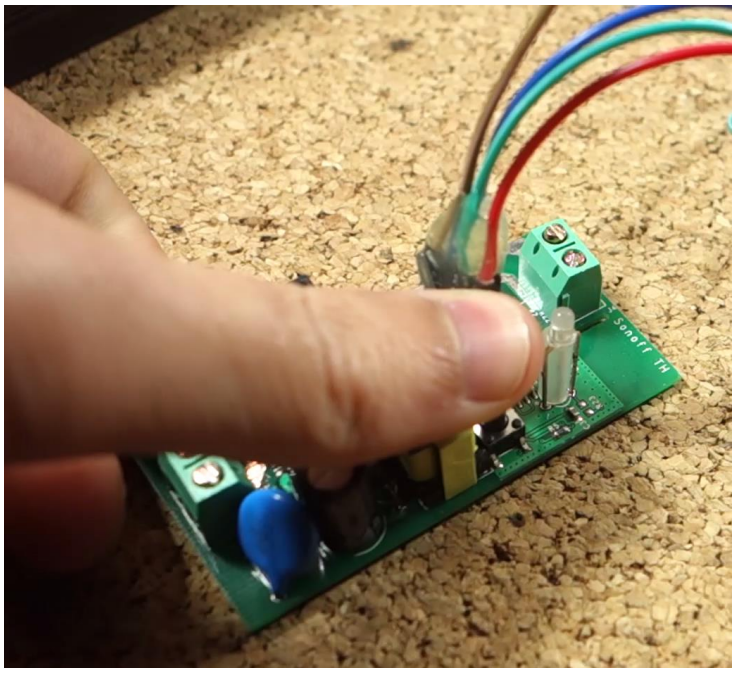
Boot Your Sonoff in Flashing Mode

To flash a new firmware to your Sonoff, you have to boot your Sonoff in flashing mode. Follow this 4 step process:

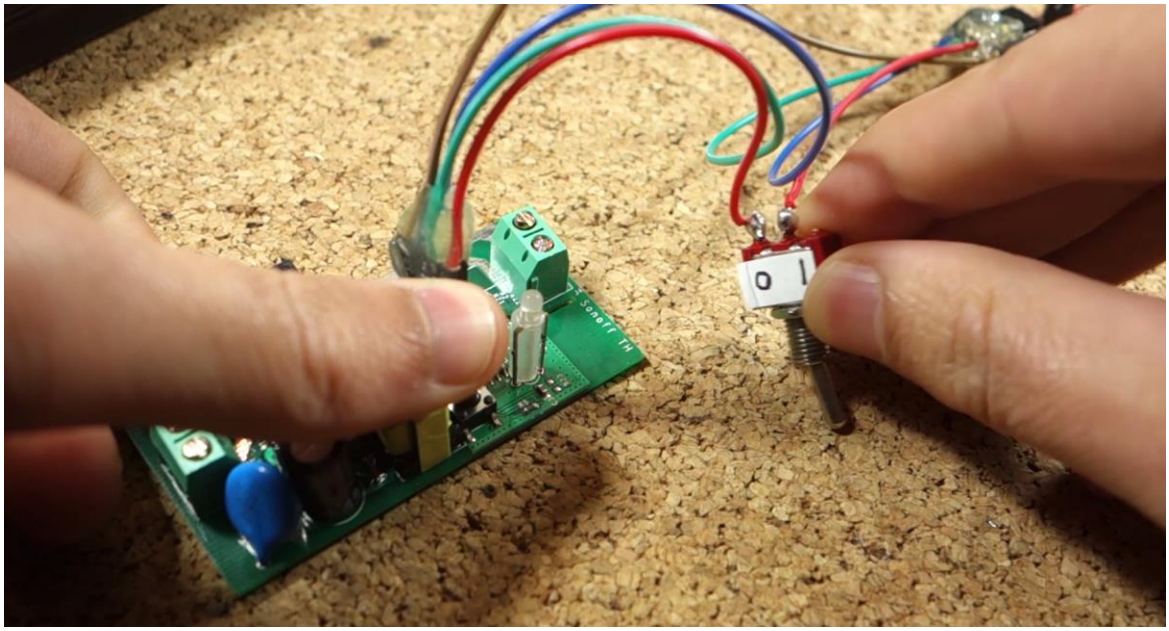
1) Connect your 3.3V FTDI programmer to your computer



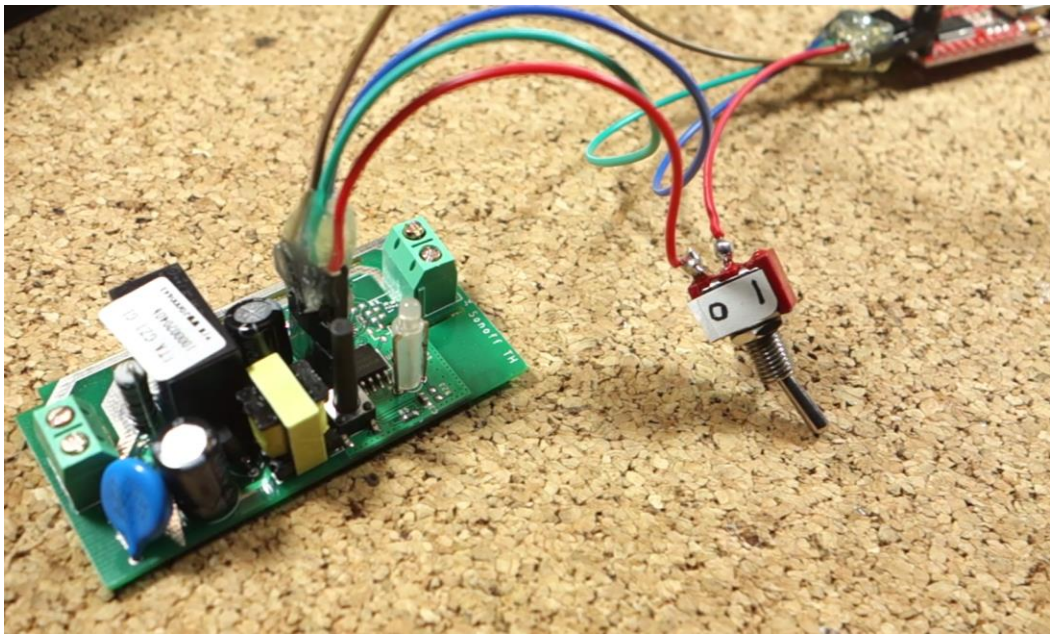
2) Hold down the Sonoff button



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.

Writing Your Arduino Sketch

At this point, you should have the ESP8266 add-on installed in the Arduino IDE. Open the next link and copy the code to your Arduino IDE.

SOURCE CODE
https://gist.github.com/RuiSantosdotme/ff82adac944150d336dce38710552124

```
#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> <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());
```

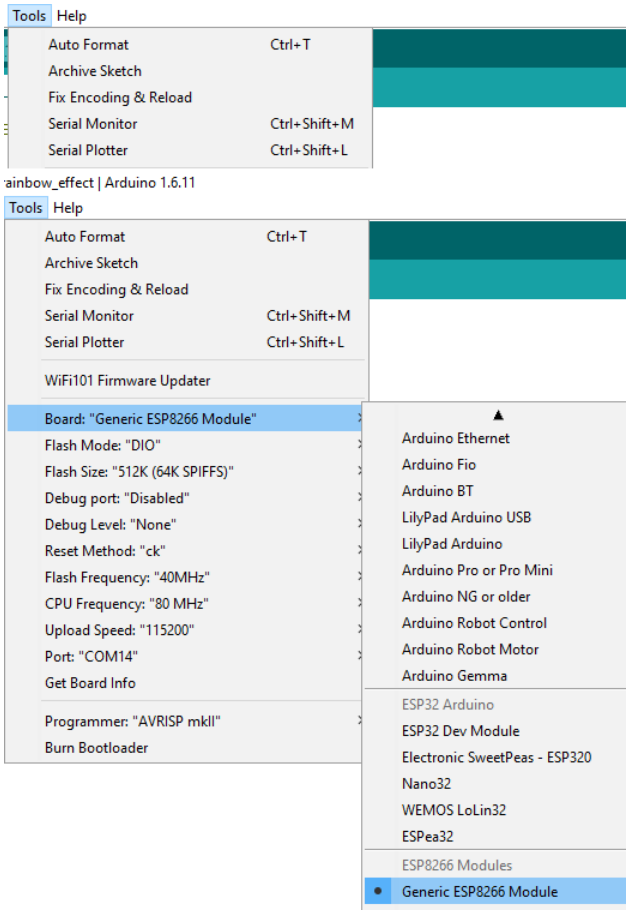
```
  if (mdns.begin("esp8266", WiFi.localIP())) {
    Serial.println("MDNS responder started");
  }

  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();
}
```

Uploading Code

Finally, open your Arduino IDE. You can 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 my case)
2. Choose your ESP8266 board from **Tools** > **Board** > **Generic ESP8266 Module**
3. 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:

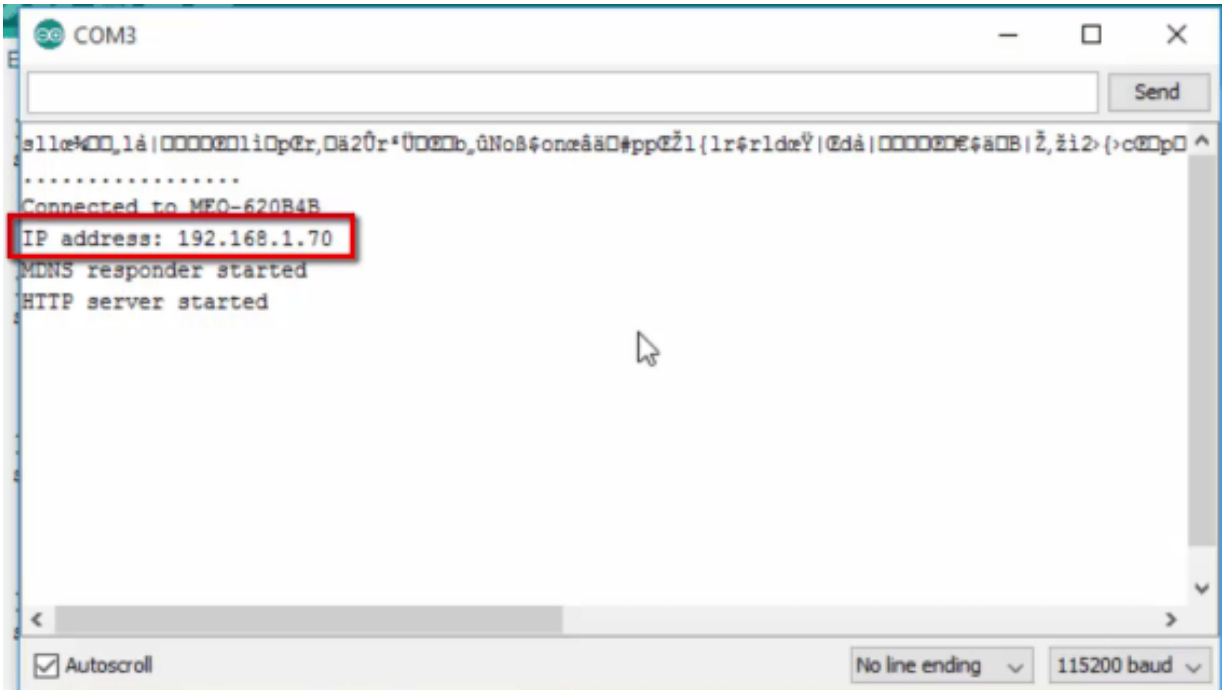
```
warning: espcomm_sync failed
error: espcomm_open failed
```

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. Connect GPIO 0 of your ESP8266 to VCC and reset your board.

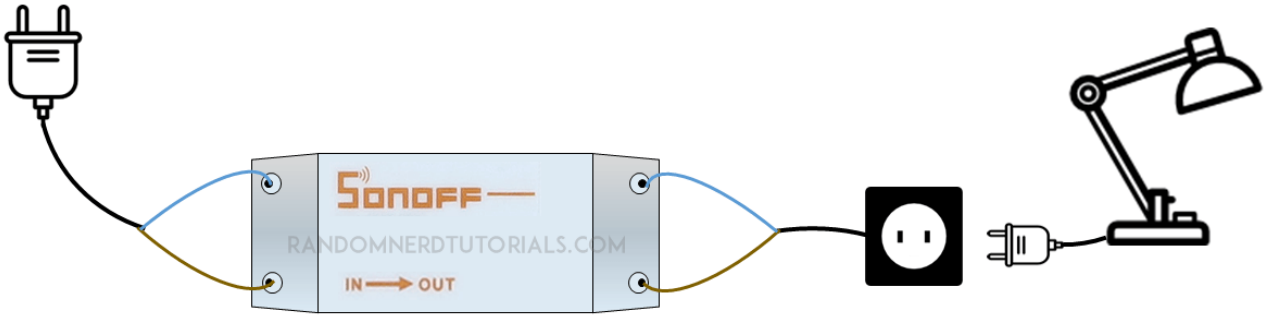
After a few seconds your IP address should appear. In my case, it’s **192.168.1.70**.



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:



How it should look:

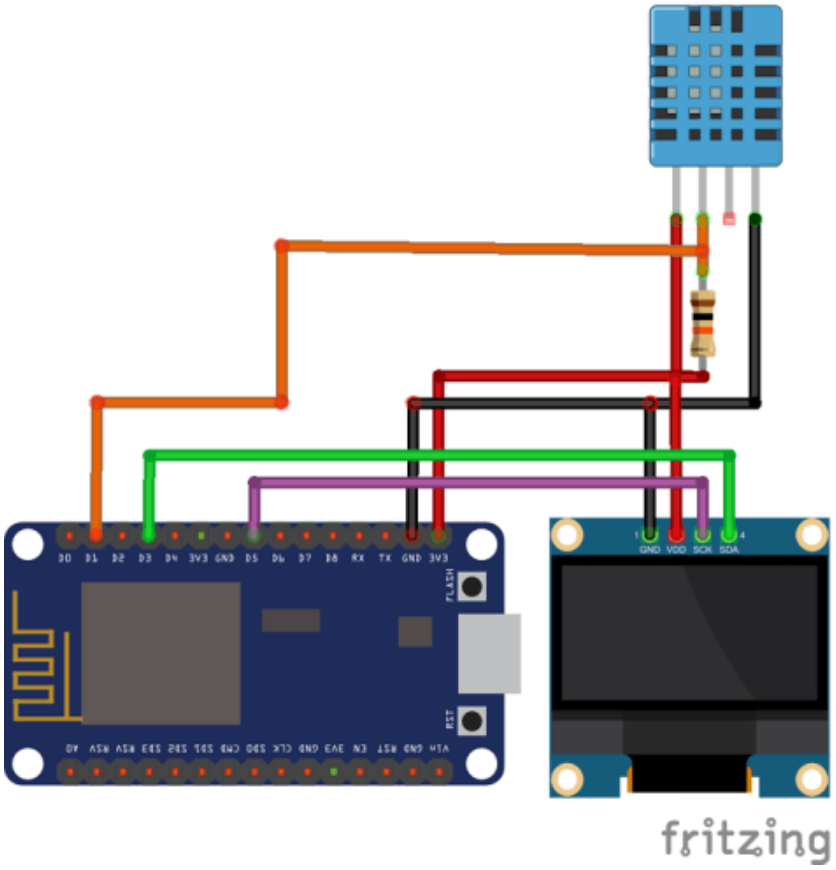
Demonstration

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:

Unit 7
Display Temperature and Humidity on OLED Display



Unit 7 – Display Temperature and Humidity on OLED Display

This Unit shows how to use the 0.96 inch OLED display with the ESP8266 to display the temperature and humidity.

This OLED display can be often a great addition to your ESP8266 projects.

Introducing the 0.96 inch OLED display

The OLED display is the one in the following figure:

This is a very small display that is made of 128 by 64 individual OLED pixels and no backlight is required. This display uses I2C communication. This means that it can communicate with the ESP8266 using just 2 pins.

Libraries

To control the OLED display, you need to install the esp8266 oled ssd1306 library.

Installing the esp8266 oled ssd1306 library:

1. Click here to download the esp8266 oled ssd1306 library. You should have a .zip folder in your Downloads folder
2. Unzip the .zip folder and you should get esp8266-oled-ssd1306-master folder
3. Rename your folder from ~~esp8266-oled-ssd1306-master~~ to esp8266_oled_ssd1306
4. Move the esp8266_oled_ssd1306 folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Temperature and Humidity in the OLED Display

In this example, you will display the temperature and humidity in the OLED display. This is just an example that shows how to integrate the OLED display with your ESP8266 using the Arduino IDE.

The idea of using the OLED display with the ESP8266 is to illustrate how you can create a physical user display for your ESP projects.

The temperature and humidity will be measured using the DHT11 temperature and humidity sensor.

If you're not familiar with the DHT11 sensor I recommend that you read the following blog post: Complete Guide for DHT11/DHT22 Humidity and Temperature Sensor With Arduino.

Parts Required

For this project, you need these components:

- 1x ESP-12E (view on eBay)
- 1x 0.96 inch OLED display (view on eBay)
- 1x DHT11 temperature and humidity sensor (view on eBay)
- 1x Breadboard
- 1x 10k Ohm resistor
- Jumper wires

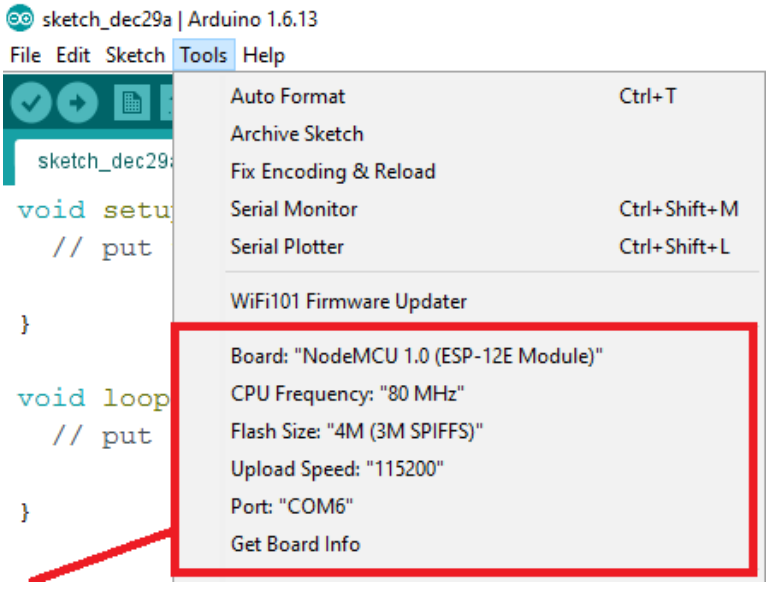
Uploading Code

Before uploading the code, make sure you've installed all the necessary libraries. Note that for this example, you also need to install the DHT library.

Installing the DHT sensor library:

1. Click here to download the DHT-sensor-library. You should have a .zip folder in your Downloads folder
2. Unzip the .zip folder and you should get DHT-sensor-library-master folder
3. Rename your folder from ~~DHT-sensor-library-master~~ to DHT_sensor_library (you really need to replace those "-" by "_")
4. Move the DHT_sensor_library folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

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



Visit the link below to download the code that you need to upload to your ESP8266.

SOURCE CODE

https://gist.github.com/RuiSantosdotme/761d99dfc24dd1b4486177e3c9b64171

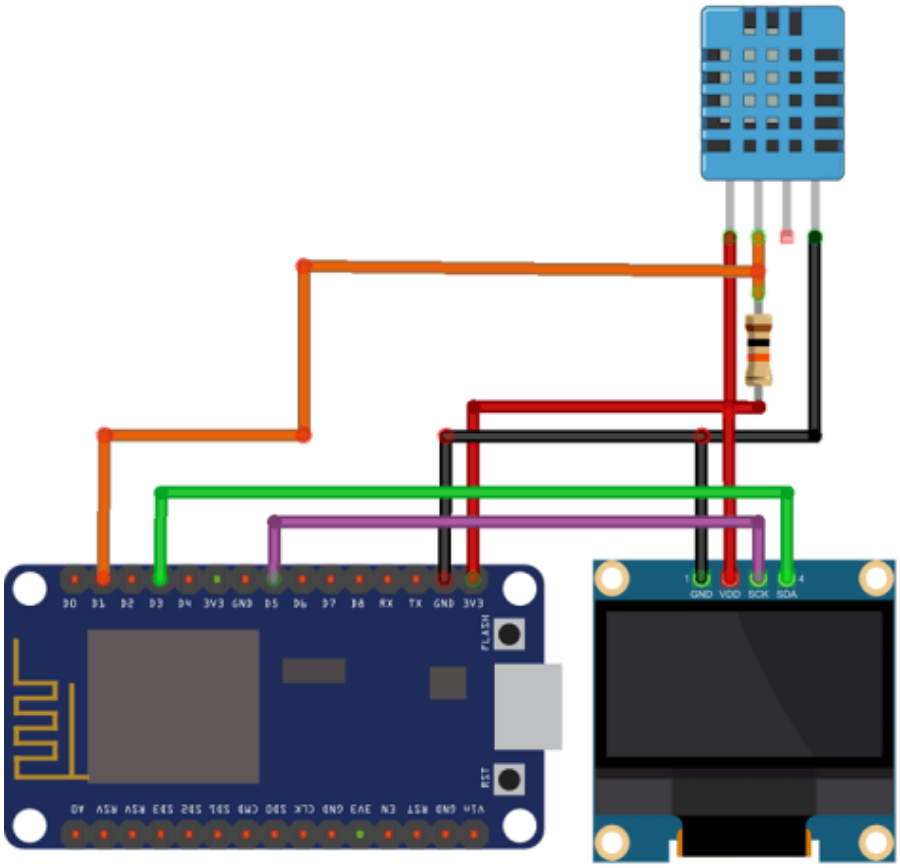
```
// Include the correct display library
// For a connection via I2C using Wire include
#include <Wire.h>  // Only needed for Arduino 1.6.5 and earlier
#include "SSD1306.h" // alias for `#include "SSD1306Wire.h"`
// or #include "SH1106.h" alis for `#include "SH1106Wire.h"`
// For a connection via I2C using brzo_i2c (must be installed) include
// #include <brzo_i2c.h> // Only needed for Arduino 1.6.5 and earlier
// #include "SSD1306Brzo.h"
// #include "SH1106Brzo.h"
// For a connection via SPI include
// #include <SPI.h> // Only needed for Arduino 1.6.5 and earlier
// #include "SSD1306Spi.h"
// #include "SH1106SPi.h"

#include <DHT.h>

#define DHTPIN 5     // what pin we're connected to
#define DHTTYPE DHT11   // DHT 11

// Initialize the OLED display using SPI
// D5 -> CLK
// D7 -> MOSI (DOUT)
// D0 -> RES
// D2 -> DC
// D8 -> CS
// SSD1306Spi        display(D0, D2, D8);
// or
```

```
// SH1106Spi         display(D0, D2);

// Initialize the OLED display using brzo_i2c
// D3 -> SDA
// D5 -> SCL
// SSD1306Brzo display(0x3c, D3, D5);
// or
// SH1106Brzo  display(0x3c, D3, D5);

// Initialize the OLED display using Wire library
SSD1306  display(0x3c, D3, D5);
// SH1106 display(0x3c, D3, D5);

// Initialize DHT sensor for normal 16mhz Arduino
DHT dht(DHTPIN, DHTTYPE);

void setup(){
  // Initialising the UI will init the display too.
  display.init();

  display.flipScreenVertically();
  display.setFont(ArialMT_Plain_16);
  display.setTextAlignment(TEXT_ALIGN_LEFT);
  dht.begin(); // initialize dht
}

void displayTempHumid(){
  // 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
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit
  float f = dht.readTemperature(true);

  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t) || isnan(f)){
```

```
    display.clear(); // clearing the display
    display.drawString(5,0, "Failed DHT");
    return;
  }
  display.clear();
  display.drawString(0, 0, "Humidity: " + String(h) + "%\t");
  display.drawString(0, 16, "Temp: " + String(t) + "C");
  display.drawString(0, 32, "Temp: " + String(f) + "F");
}
void loop(){
  displayTempHumid();
  display.display();
  delay(2000);
}
```

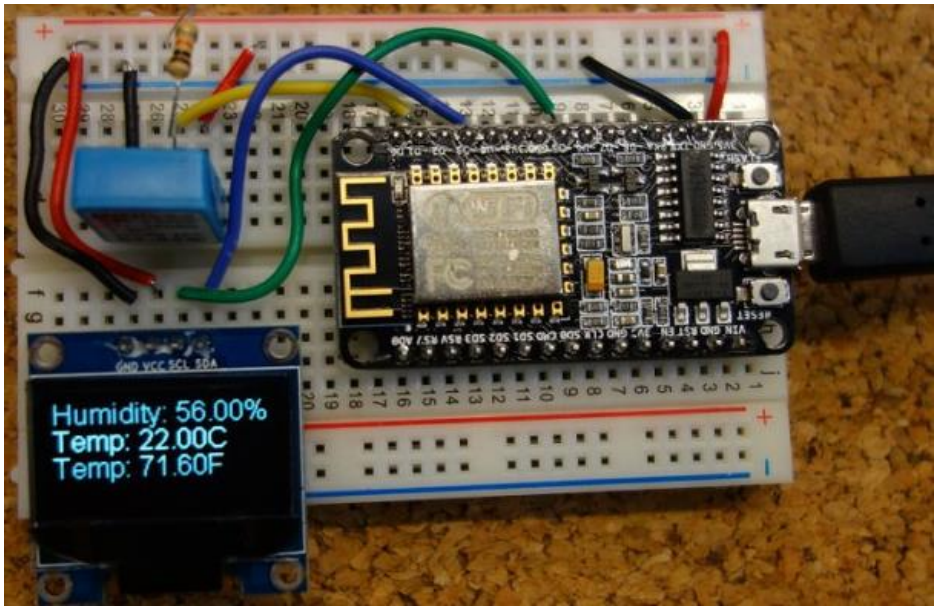
Final Circuit

Assemble the circuit by following the schematics below.



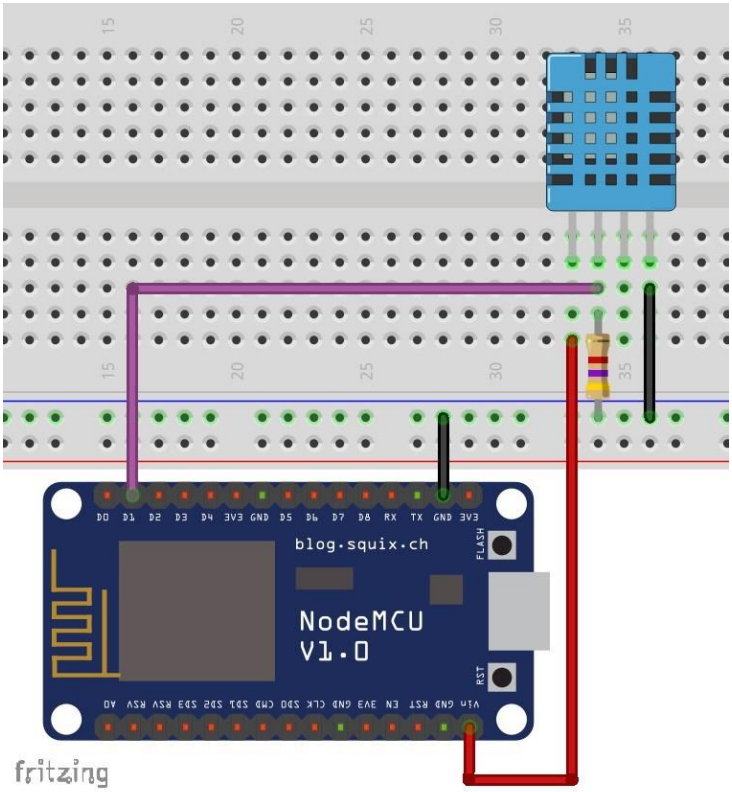
Demonstration

By the end of this project this is what you should have:



You could easily add web server capabilities to this project to display the sensor readings both on your smartphone and OLED display.

Unit 8
DHT11/DHT22 Temperature and Humidity Web Server



Unit 8 - DHT11/DHT22 Temperature and Humidity Web Server

In this project, you'll create a standalone web server with an ESP8266 that displays the temperature and humidity with a DHT11 or DHT22 sensor.

Note: if you've followed the previous Unit, you are already familiar with DHT sensor library. You can skip "Installing the DHT Sensor Library" and go directly to the "Uploading Code" section.

Installing the DHT Sensor Library

The DHT sensor library provides an easy way of using any DHT sensor to read temperature and humidity with your ESP8266 or Arduino boards.

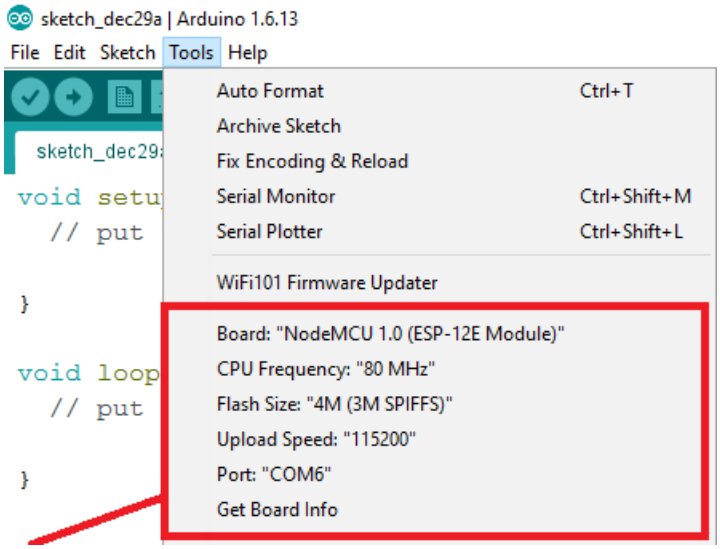
Installing the DHT sensor library:

1. Click here to download the DHT-sensor-library. You should have a .zip folder in your Downloads folder
2. Unzip the .zip folder and you should get DHT-sensor-library-master folder
3. Rename your folder from ~~DHT-sensor-library-master~~ to DHT_sensor_library (you really need to replace those "-" by "_")
4. Move the DHT_sensor_library folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Uploading Code

Having the ESP8266 add-on for the Arduino IDE installed.

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



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.

SOURCE CODE

https://gist.github.com/RuiSantosdotme/74eb202d83131e818aa5d3a866955556

```
// Including the ESP8266 WiFi library
#include <ESP8266WiFi.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 network details
const char* ssid = "YOUR_NETWORK_NAME";
const char* password = "YOUR_NETWORK_PASSWORD";

// Web Server on port 80
WiFiServer server(80);

// DHT Sensor
const int DHTPin = 5;
// Initialize DHT sensor.
DHT dht(DHTPin, DHTTYPE);

// Temporary variables
static char celsiusTemp[7];
static char fahrenheitTemp[7];
static char humidityTemp[7];

// only runs once on boot
void setup() {
  // Initializing serial port for debugging purposes
  Serial.begin(115200);
  delay(10);

  dht.begin();

  // 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());
}

// runs over and over again
void loop() {
  // Listenning for new clients
  WiFiClient client = server.available();

  if (client) {
    Serial.println("New client");
    // bolean to locate when the http request ends
    boolean blank_line = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();

        if (c == '\n' && blank_line) {
            // 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(celsiusTemp,"Failed");
```

```
            strcpy(fahrenheitTemp, "Failed");
            strcpy(humidityTemp, "Failed");
          }
          else{
            // Computes temperature values in Celsius + Fahrenheit and
Humidity
            float hic = dht.computeHeatIndex(t, h, false);
            dtostrf(hic, 6, 2, celsiusTemp);
            float hif = dht.computeHeatIndex(f, h);
            dtostrf(hif, 6, 2, fahrenheitTemp);
            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");
          }
          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 and humidity
            client.println("<!DOCTYPE HTML>");
            client.println("<html>");
            client.println("<head></head><body><h1>ESP8266 - Temperature and
Humidity</h1><h3>Temperature in Celsius: ");
            client.println(celsiusTemp);
            client.println("*C</h3><h3>Temperature in Fahrenheit: ");
            client.println(fahrenheitTemp);
            client.println("*F</h3><h3>Humidity: ");
            client.println(humidityTemp);
            client.println("%</h3><h3>");
            client.println("</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.");
  }
}
```

Note: you can comment/uncomment the sketch, if you prefer to use the DHT22 sensor.

Parts Required

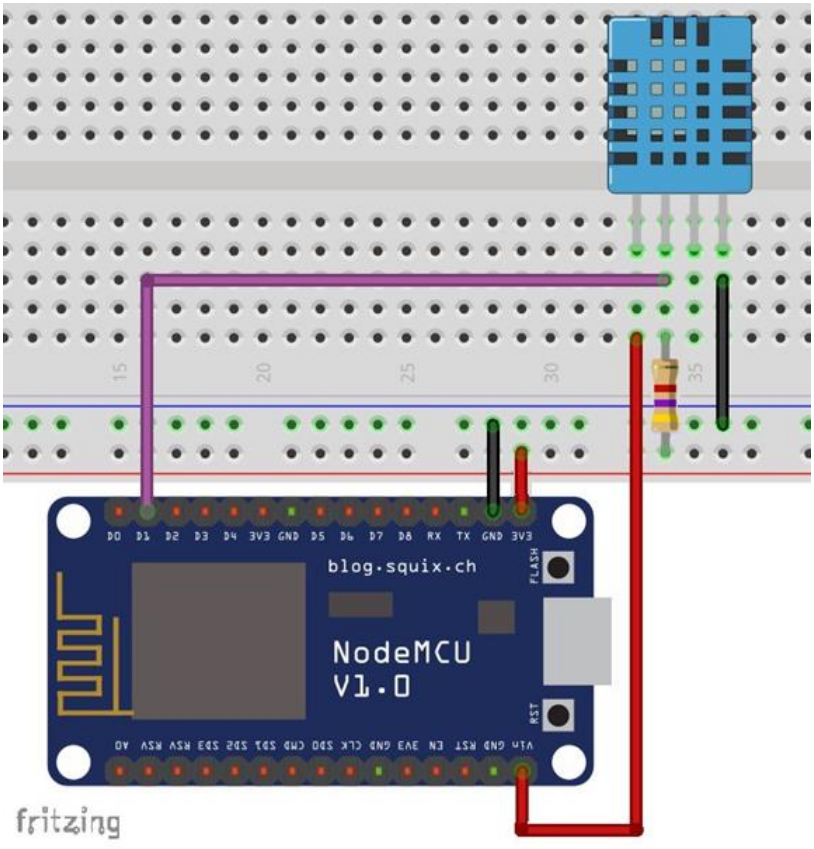
To complete this project you need the following components:

- 1x ESP-12E (see on eBay)
- 1x DHT11 Temperature and Humidity Sensor (see on eBay)
- 1x 4700 Ohm Resistor (see on eBay)
- 1x Breadboard (see on eBay)

Note: other DHT sensor types will also work with a small change in the code (as described in the sketch comments).

Final Circuit

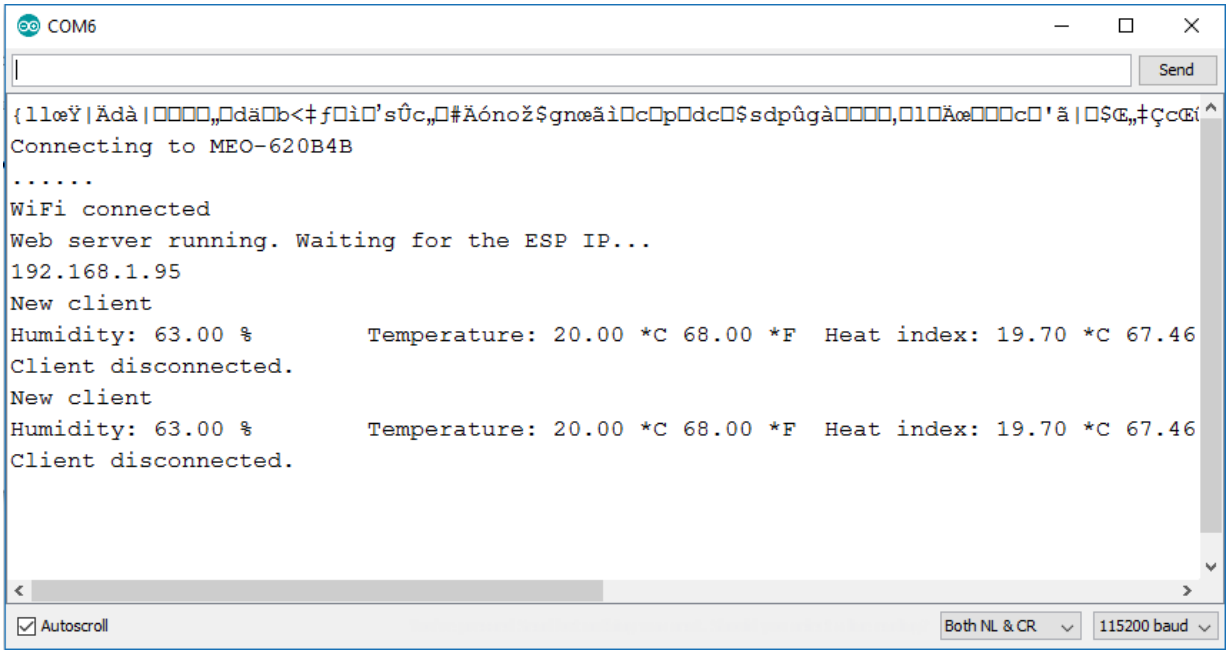
Here's the schematics:



Important: the DHT sensor requires 5V to operate properly, so make sure you use the Vin pin from your ESP8266 that outputs 5V.

ESP8266 IP Address

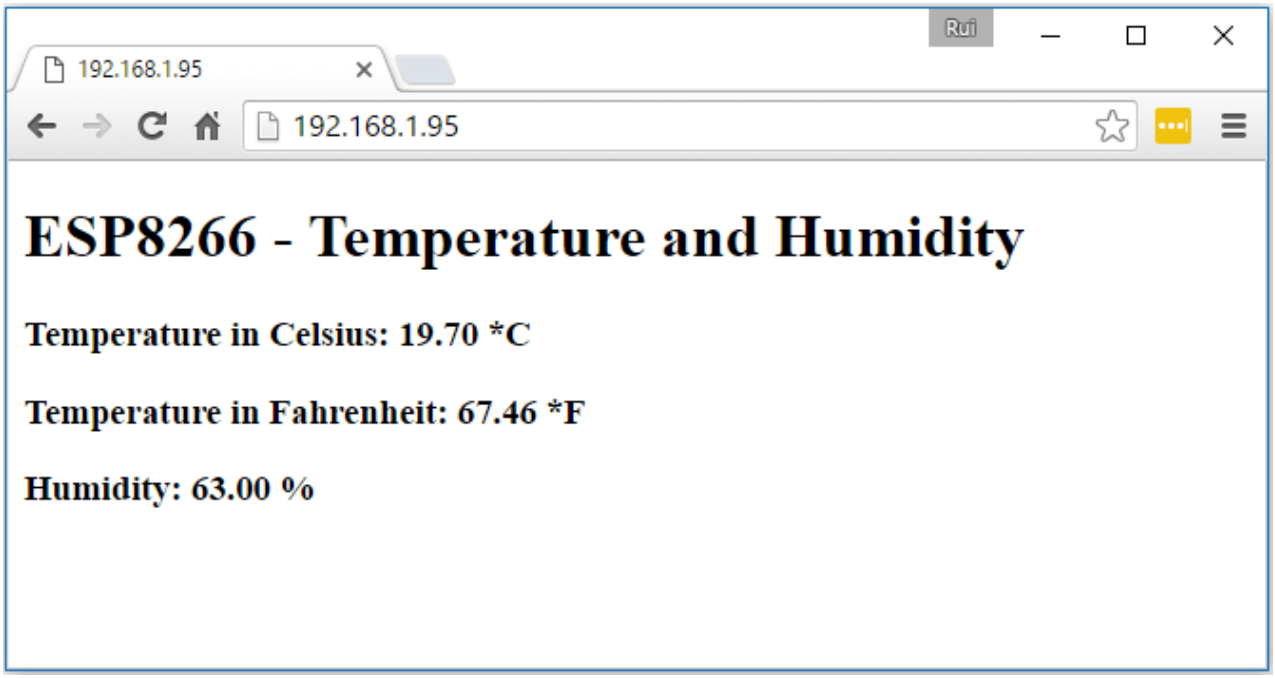
Open the Arduino IDE serial monitor at a baud rate of 115200. After a few seconds your IP address should appear. In my case it's 192.168.1.95.



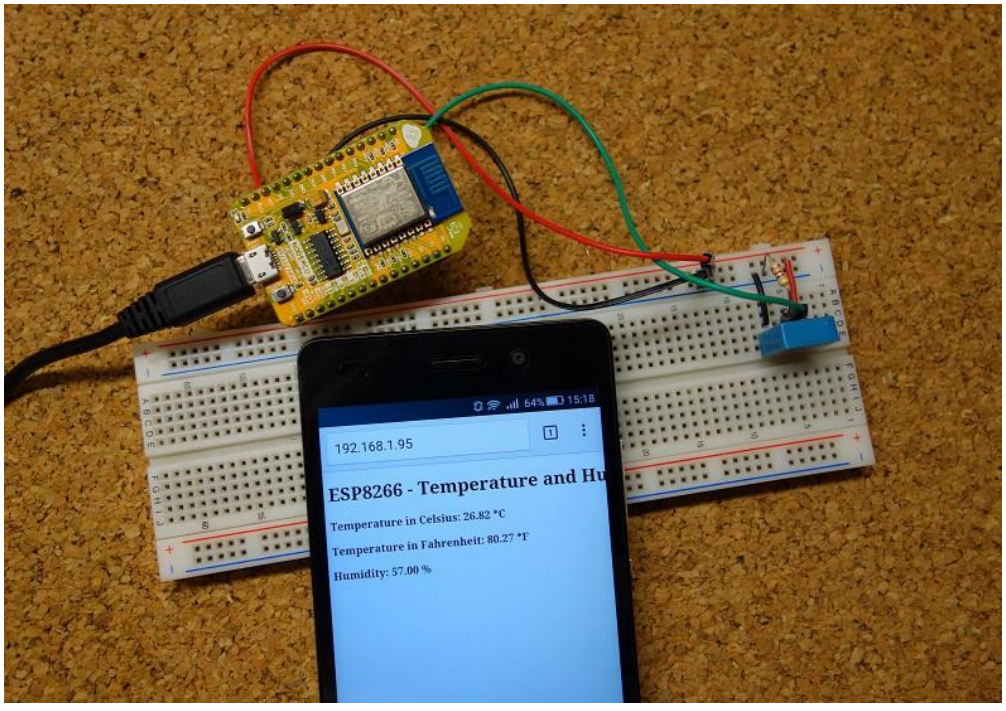
Demonstration

For the final demonstration open any browser from a device that is connected to the same router that your ESP.

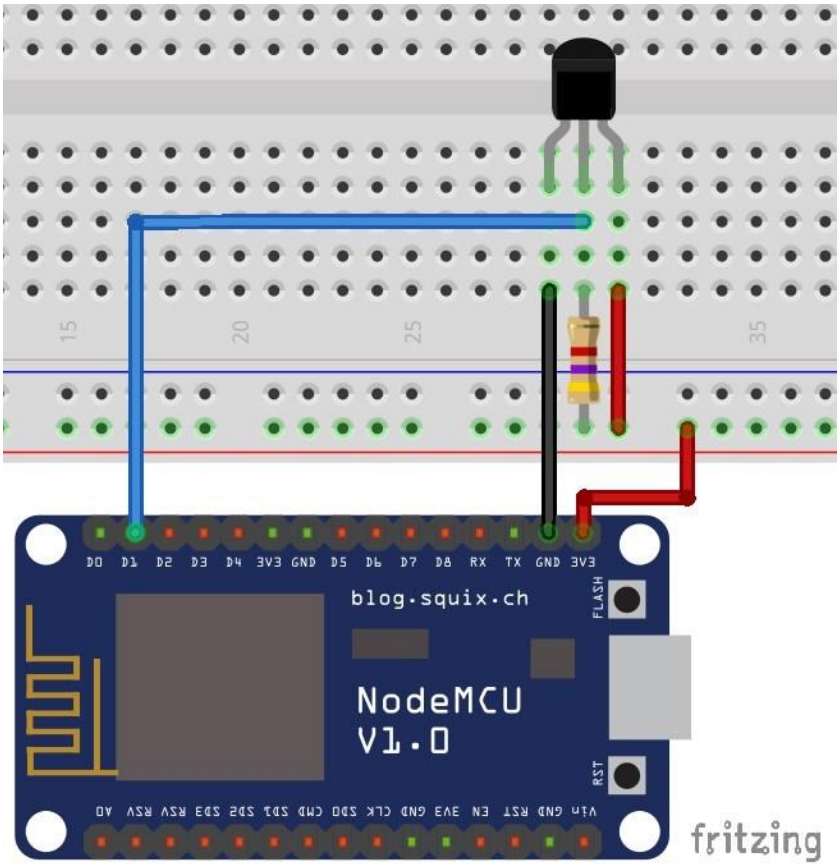
Then, type the IP address and press Enter!



Now you can see the latest temperature and humidity readings with your smartphone.



Unit 9
DS18B20 Temperature Sensor Web Server



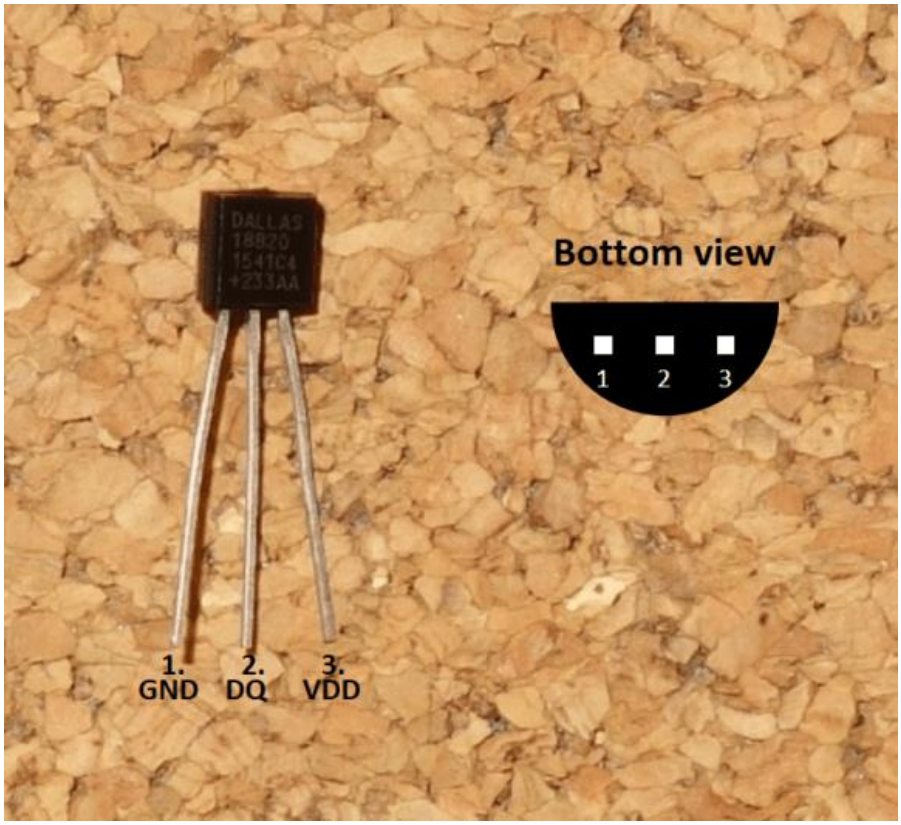
Unit 9 - DS18B20 Temperature Sensor Web Server

In this Unit, you'll create a standalone web server with an ESP8266 that displays the temperature with a DS18B20 temperature sensor.

DS18B20 Temperature Sensor

The DS18B20 temperature sensor is a 1-wire digital temperature sensor. This means that you can read the temperature with a very simple circuit setup. It communicates on common bus, which means that you can connect several devices and read their values using just one digital pin of the ESP8266.

The sensor has just three pins as you can see in the following figure:



The DS18B20 is also available in waterproof version:

Features

Here's the main features of the DS18B20 temperature sensor:

- Communicates over 1-wire bus communication
- Operating range temperature: -55ºC to 125ºC
- Accuracy +/-0.5 ºC (between the range -10ºC to 85ºC)

Installing the Libraries

You need to install the OneWire library and DallasTemperature library.

Installing the OneWire library

1. Click here to download the OneWire library. You should have a .zip folder in your Downloads
2. Unzip the .zip folder and you should get OneWire-master folder

3. Rename your folder from ~~OneWire-master~~ to OneWire
4. Move the OneWire folder to your Arduino IDE installation libraries folder

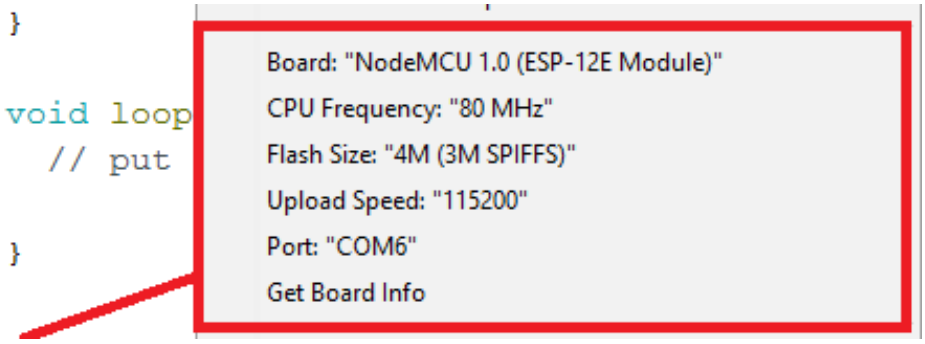
Installing the DallasTemperature library

1. Click here to download the DallasTemperature library. You should have a .zip folder in your Downloads
2. Unzip the .zip folder and you should get Arduino-Temperature-Control-Library-master folder
3. Rename your folder from ~~Arduino-Temperature-Control-Library-master~~ to DallasTemperature
4. Move the DallasTemperature folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Uploading Code

Having the ESP8266 add-on for the Arduino IDE installed.

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



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.

SOURCE CODE

https://gist.github.com/RuiSantosdotme/d66091acc126dfa4ed176add1e6b8502

```
// Including the ESP8266 WiFi library
#include <ESP8266WiFi.h>
#include <OneWire.h>
#include <DallasTemperature.h>

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

// 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);

// 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) {
    Serial.println("New client");
    // bolean to locate when the http request ends
    boolean blank_line = true;
    while (client.connected()) {
      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>");
            client.println("<head></head><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;
        }
```

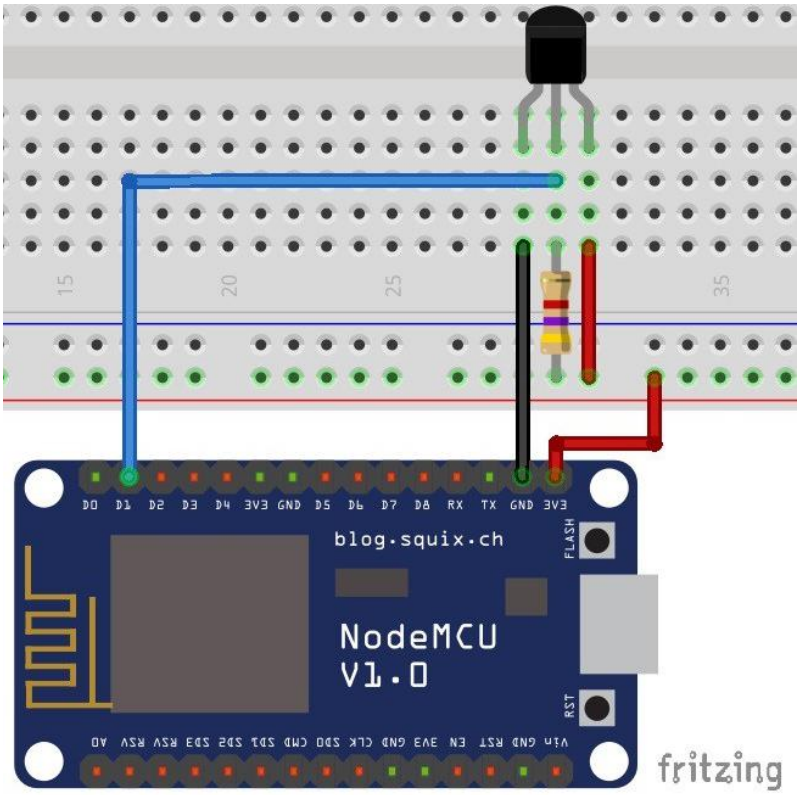
```
        else if (c != '\r') {
          // when finds a character on the current line
          blank_line = false;
        }
      }
    }
    // closing the client connection
    delay(1);
    client.stop();
    Serial.println("Client disconnected.");
  }
}
```

Parts Required

To complete this project you need the following components:

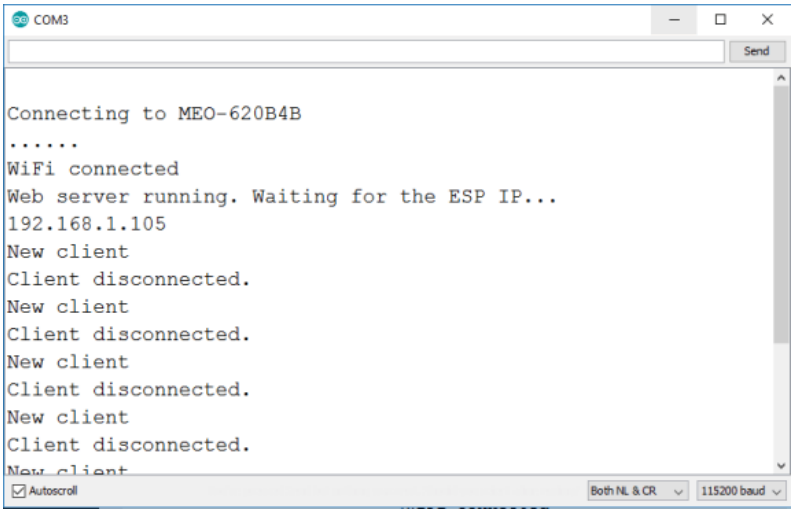
- 1x ESP-12E (see on eBay)
- 1x DS18B20 temperature sensor (view on eBay)
- 1x 4700 Ohm Resistor (see on eBay)
- 1x Breadboard (see on eBay)

Final Circuit



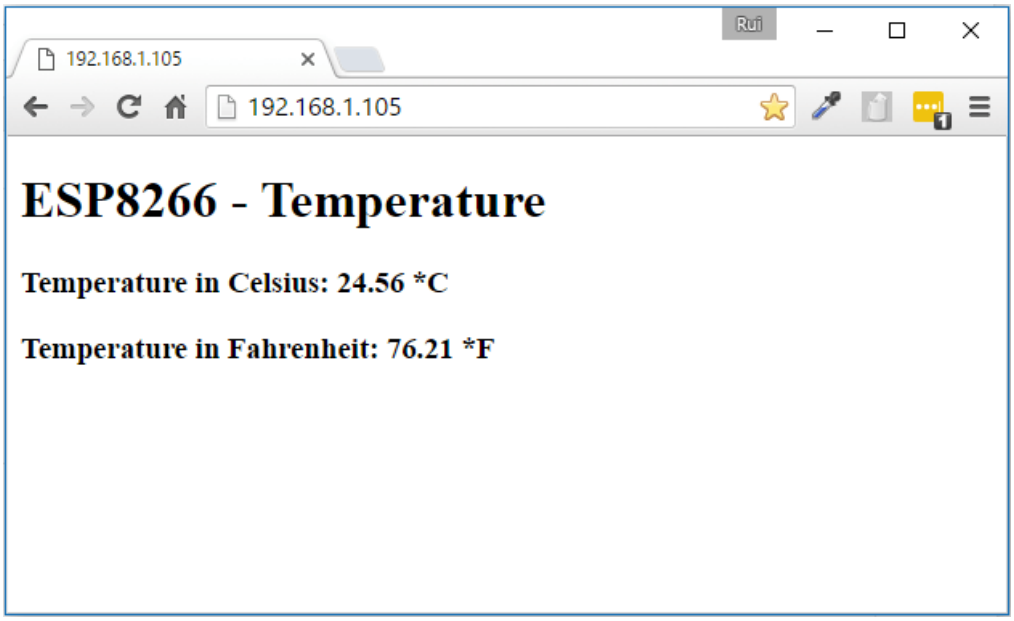
ESP8266 IP Address

Open the Arduino IDE serial monitor at a baud rate of 115200. After a few seconds your IP address should appear. In my case, it's, 192.168.1.105.



Demonstration

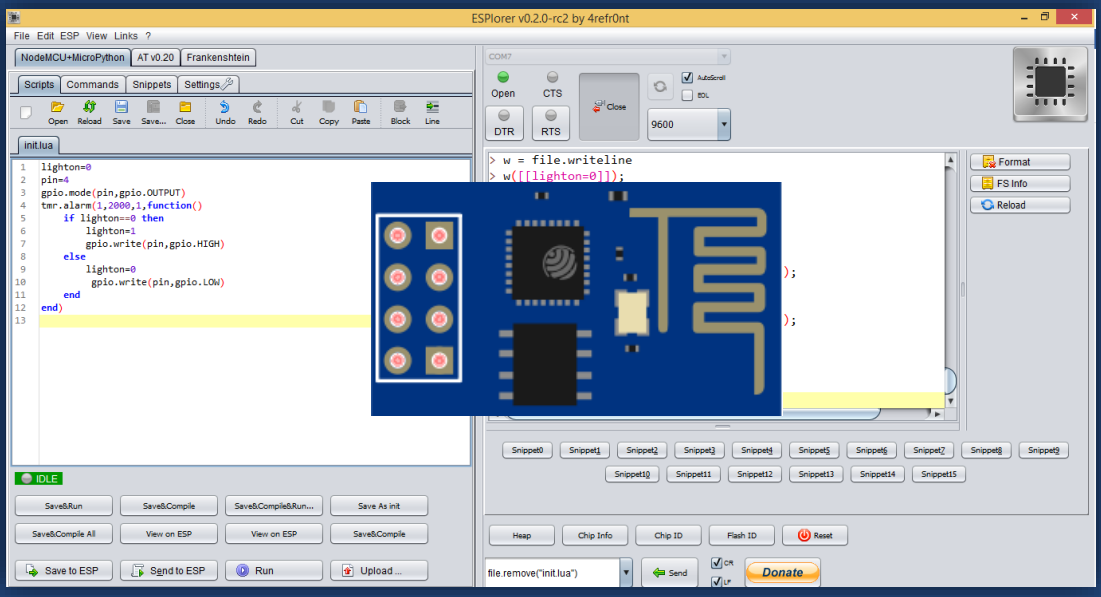
For the final demonstration open any browser from a device that is connected to the same router that your ESP.



Now, you can see temperature in Celsius and Fahrenheit in your web server. To see the latest readings simply refresh the web page.

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

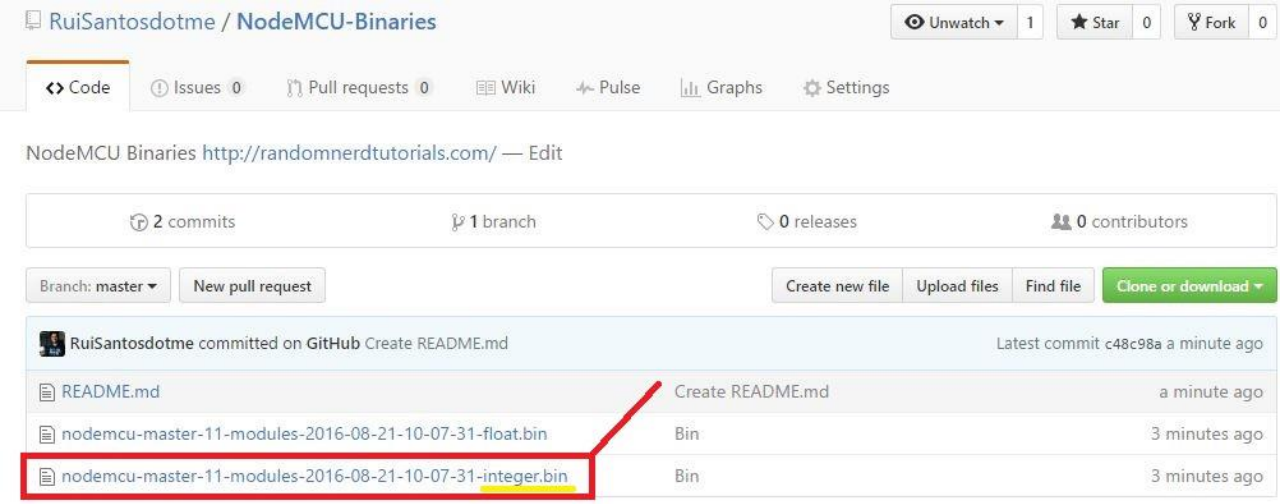
Downloading NodeMCU Firmware Binaries

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.

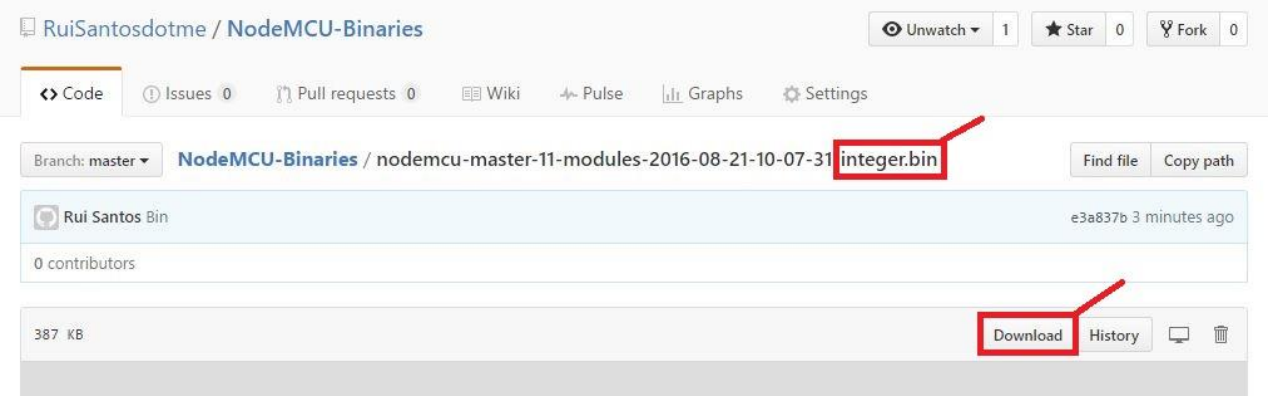
Open the following link:

- https://github.com/RuiSantosdotme/NodeMCU-Binaries

Click the URL that ends with "**-integer.bin**".



Press the **Download** button.



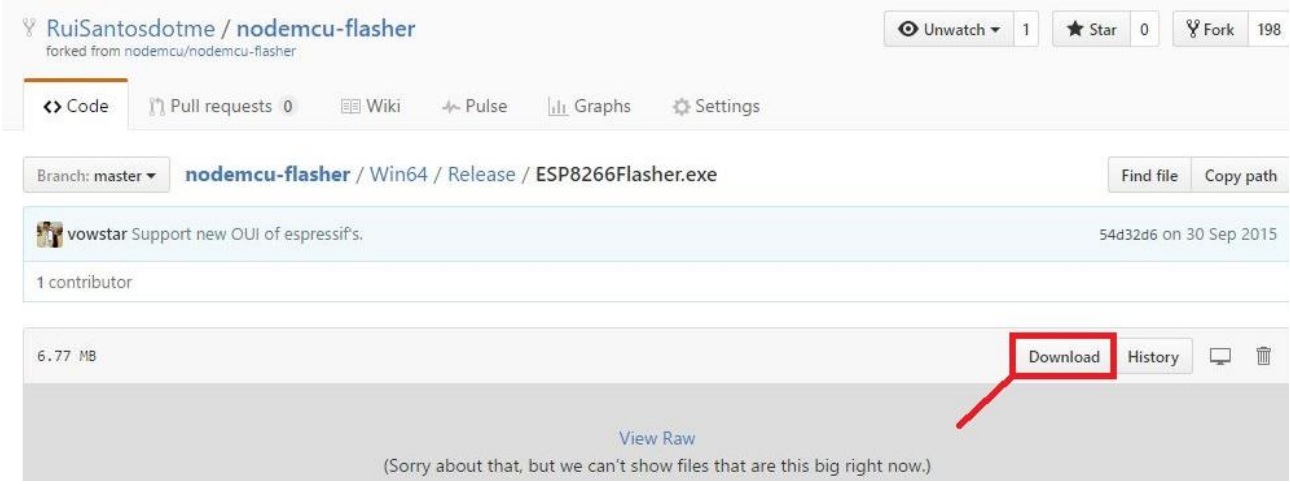
Downloading NodeMCU Flasher For Windows

At this point (when I updated this eBook) 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).



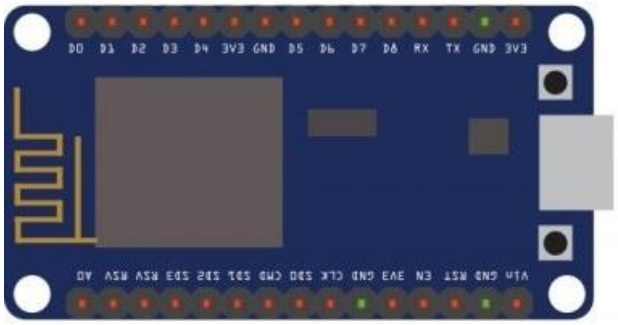
Flashing ESP8266

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

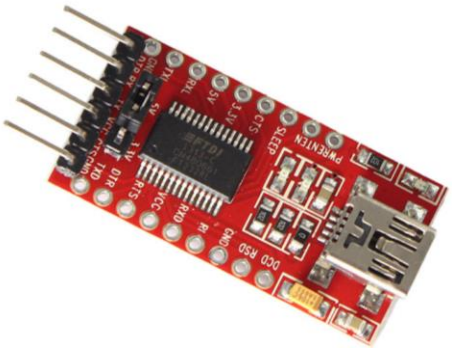
Flashing new firmware to your ESP-12E NodeMCU Kit is very simple, since it has built-in programmer. You plug your board to your computer and you don't need to make any additional connections:



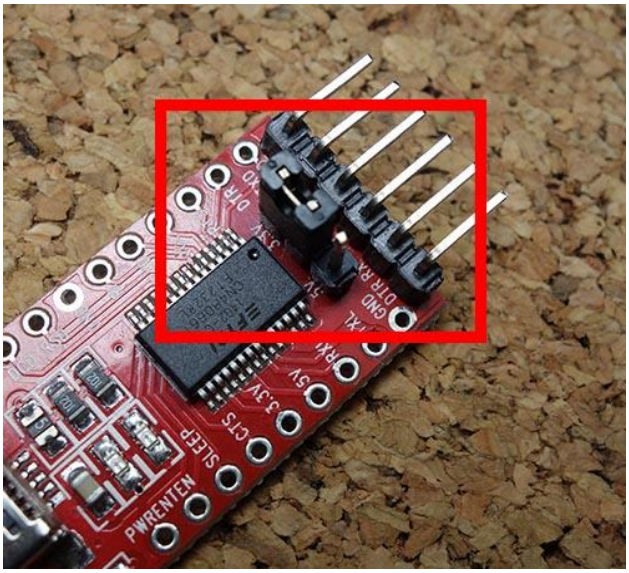
Option B – Flashing firmware to ESP-01

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

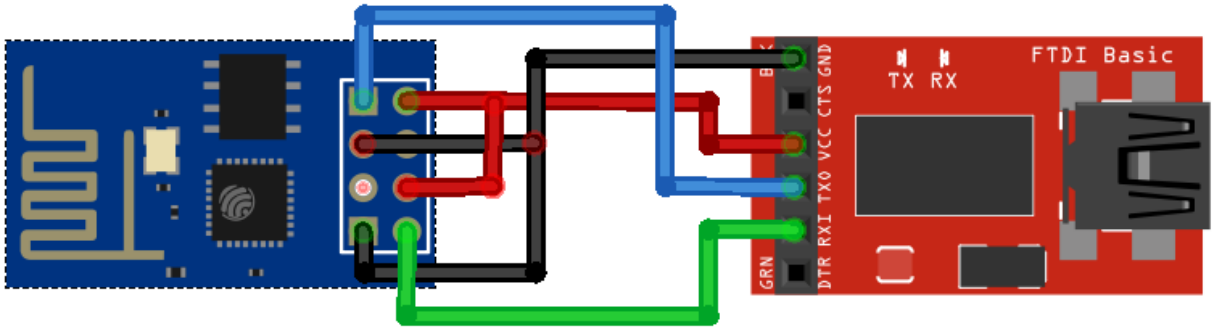
You can click here to purchase one FTDI Programmer on eBay for $3 (http://randomnerdtutorials.com/ebay-ftdi-programmer).



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.



Here's the connections:

- RX -> TX
- TX -> RX
- CH_PD -> 3.3V
- GPIO 0 -> GND
- VCC -> 3.3V
- GND -> GND

Note: the circuit above has GPIO 0 connected to GND, that's because we want to upload code. When you upload a new sketch into your ESP it requires the ESP to flash a new firmware. In normal usage (if you're not flashing your ESP with a new firmware) it would be connected to VCC.

If you have a brand new FTDI Programmer and you need to install your FTDI drivers on Windows PC, visit this website for the official drivers: http://www.ftdichip.com/Drivers/VCP.htm. In alternative, you can contact the seller that sold you the FTDI Programmer.

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 this video tutorial to fix that: http://youtu.be/SPdSKT6KdF8.

In the previous video, it is said for you to download the drivers from the FTDI website. Read carefully the YouTube description to find all the links. Here's 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.

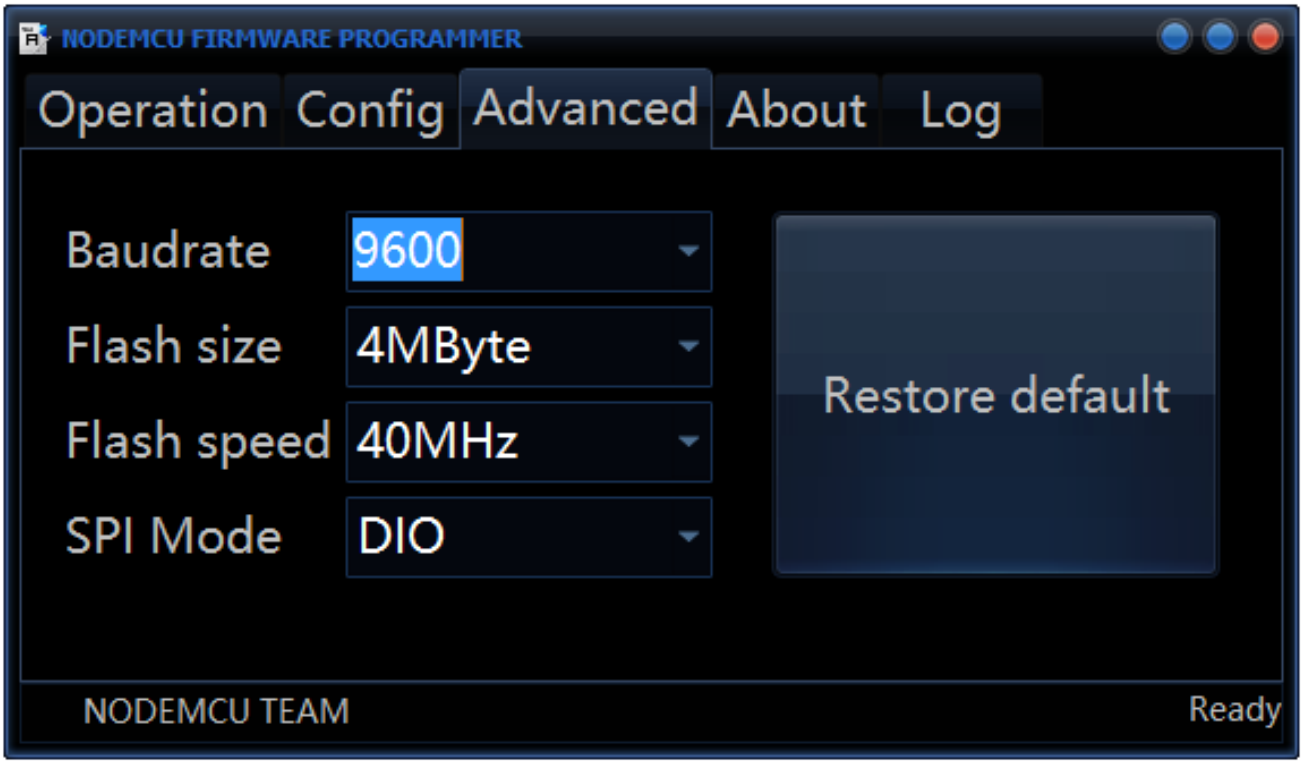
Flashing NodeMCU on Windows PC

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?

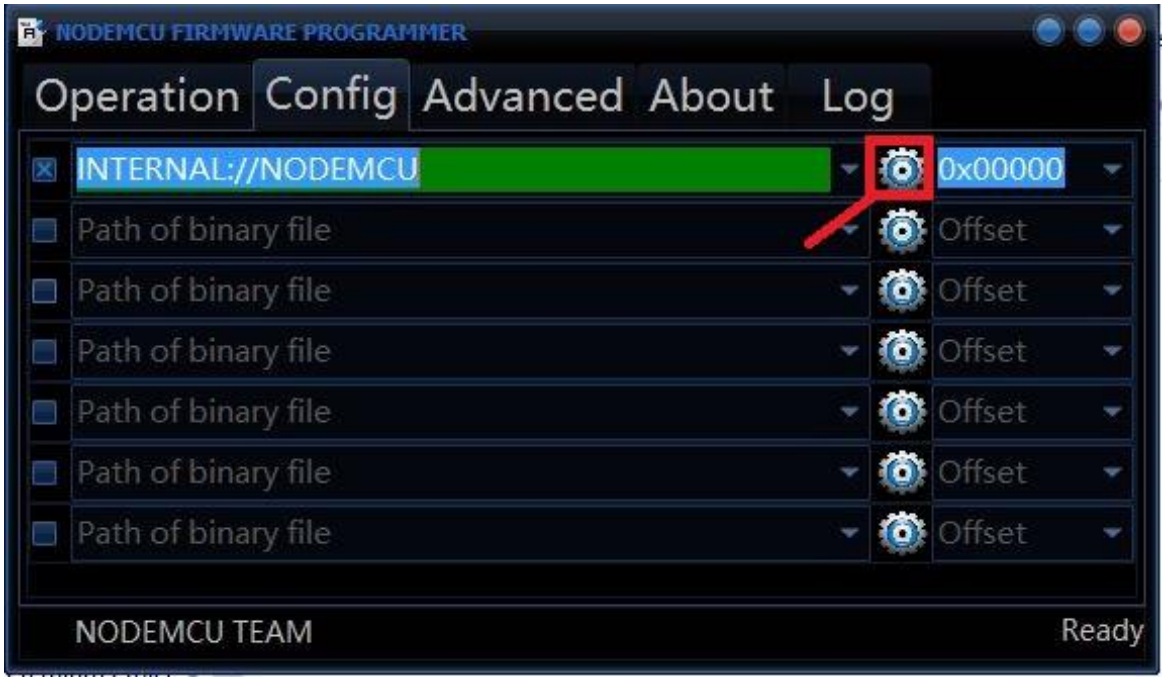
It's pretty simple. Having the serial communication established between your ESP8266 and your computer, you just need to follow these instructions:

1. Open the firmware flasher for Windows PC
2. Go to the **Advanced** tab (figure below)
3. Set your baud rate to 9600

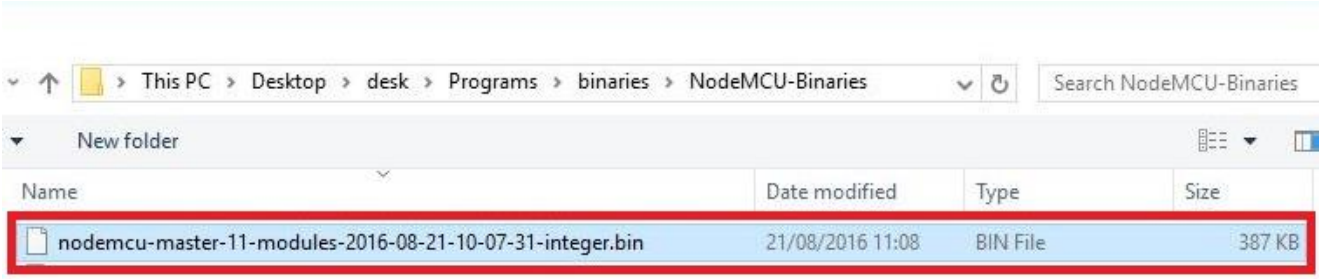
Note: some ESP8266 modules work at baud rate 115200 by default. If you're trying to flash and nothing happens, try to change the baud rate to 115200. Here's how your firmware flasher should look in the **Advanced** tab:



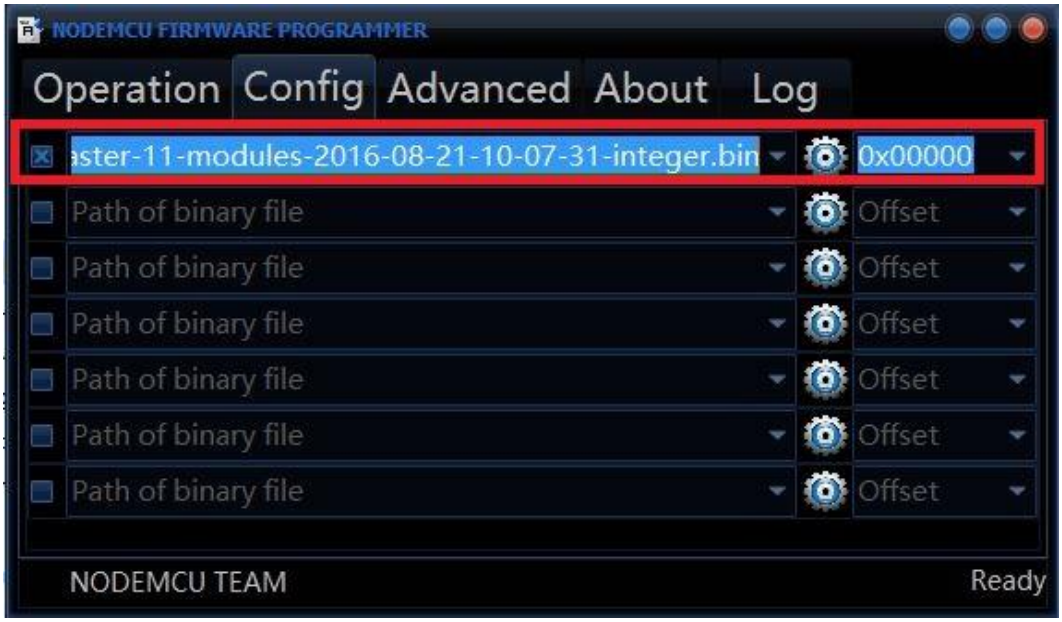
Then, open the **Config** tab and press the **gear** icon to change the binaries.



Select the NodeMCU binaries that you've download in the preceding section "Downloading NodeMCU Firmware Binaries".



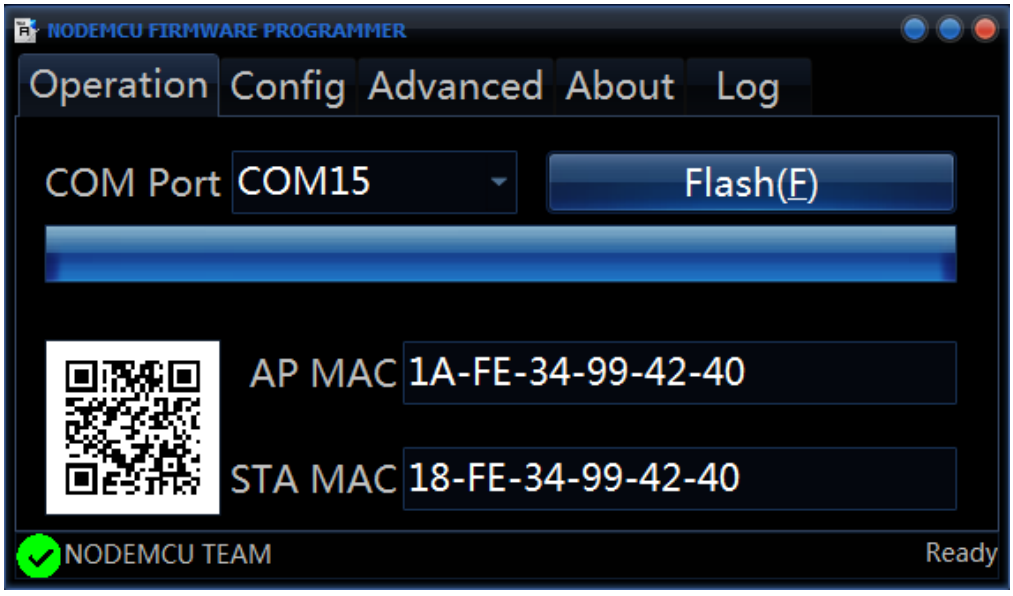
Here's how it should look like:



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-12E/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.

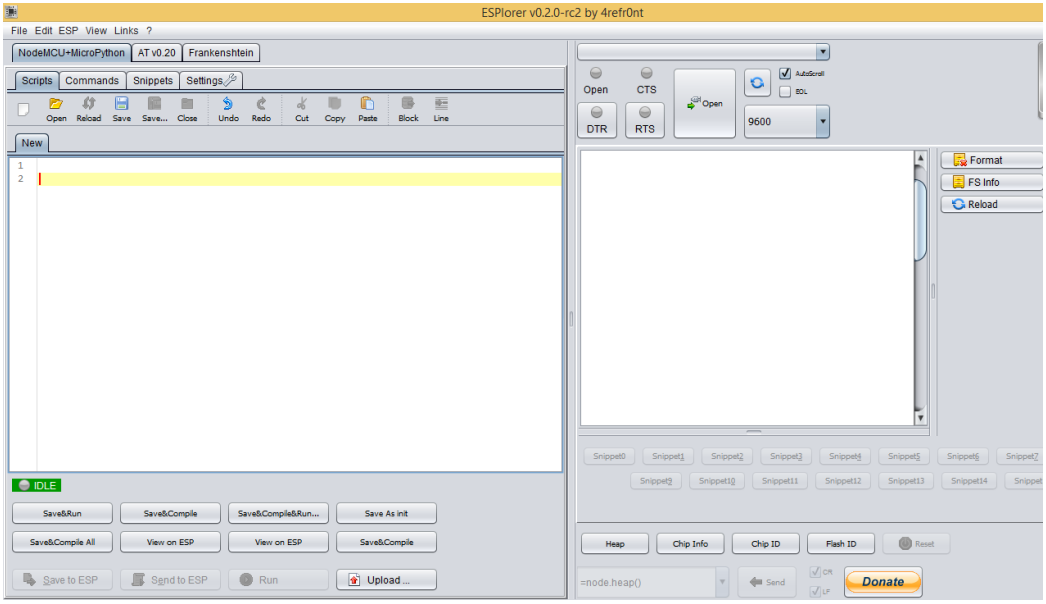
Flashing NodeMCU on Mac OS X or Linux

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



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 communications 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

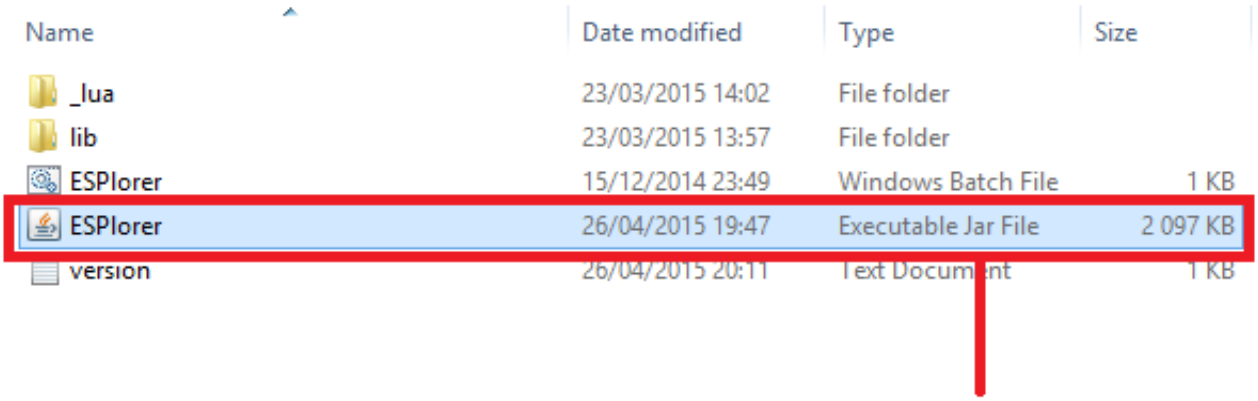
Downloading ESPlorer

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:

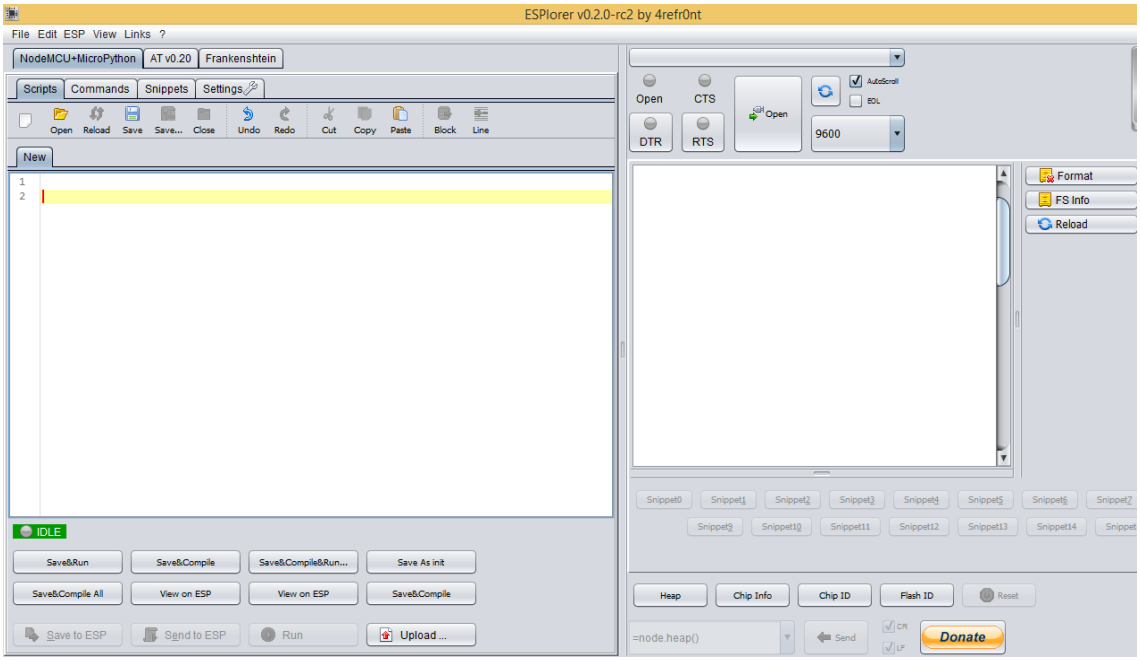


Execute the "ESPlorer.jar" file and the ESPlorer IDE should open after a few seconds (the "ESPlorer.jar" file is what you need to open every time you want to work with the ESPlorer IDE).

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 your 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.

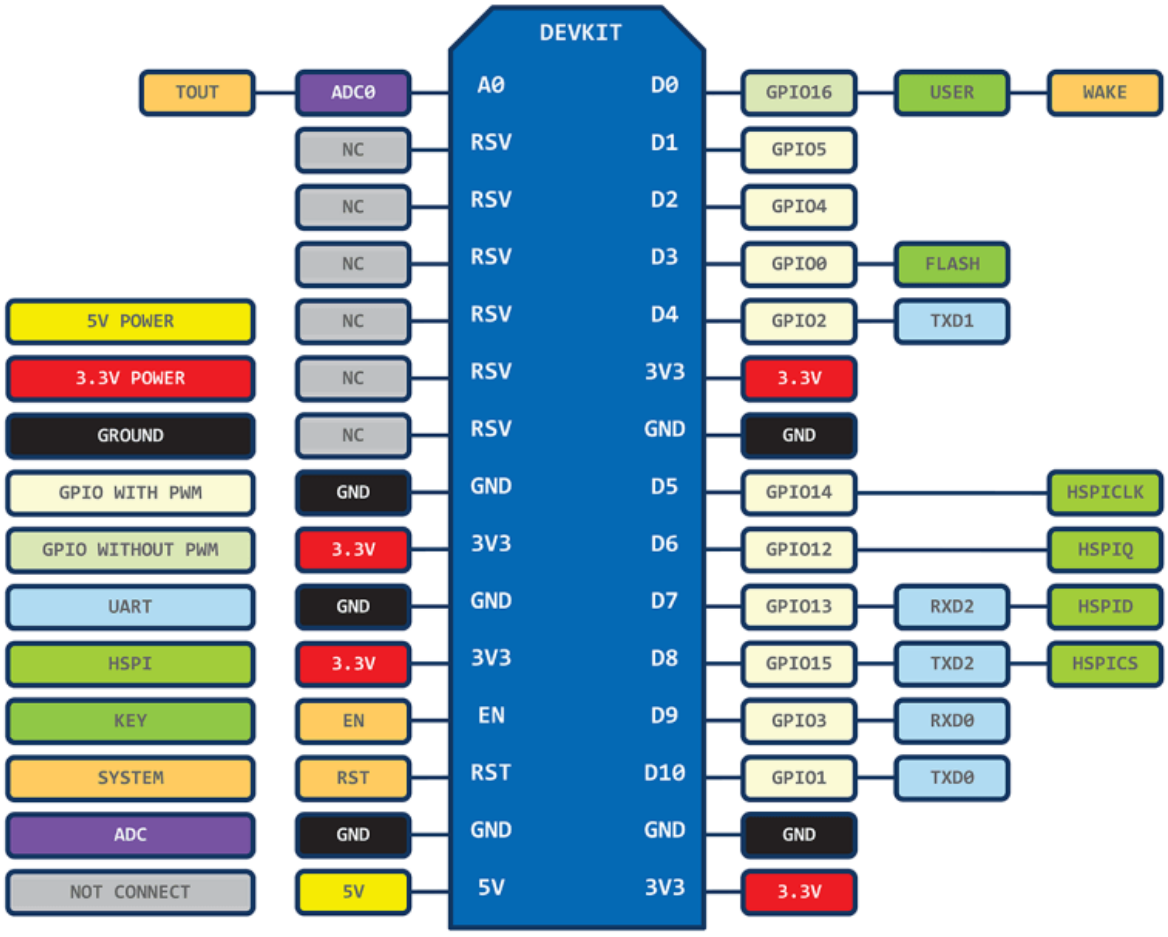
About GPIOs Assignment

This page provides a quick pinout reference for ESP8266-12E that has built-in programmer.

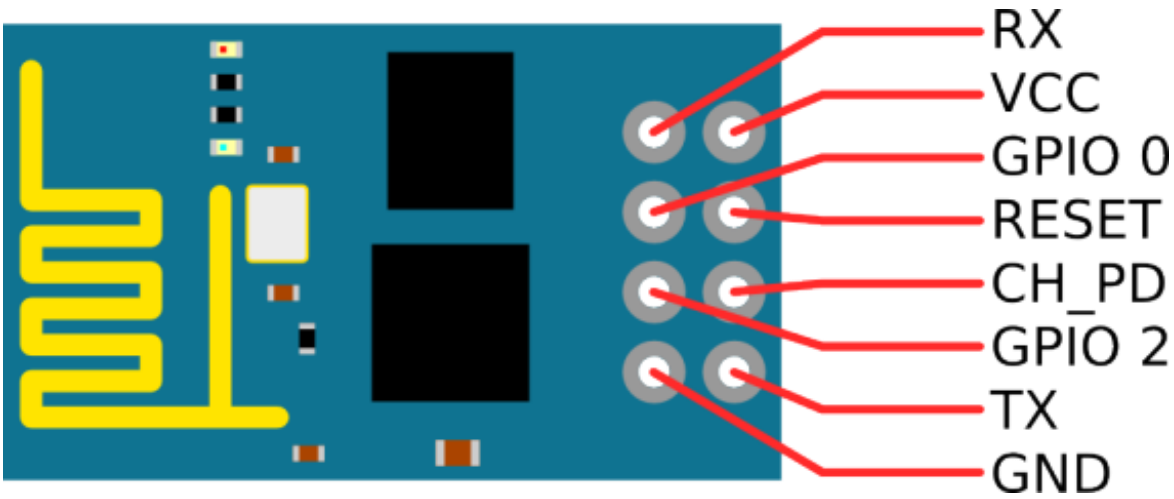
Use this table below for a quick reference on how to assign the ESP GPIOs in Lua code:

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://gist.github.com/RuiSantosdotme/e615217effb468ac270828bbf8fcddb1

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!

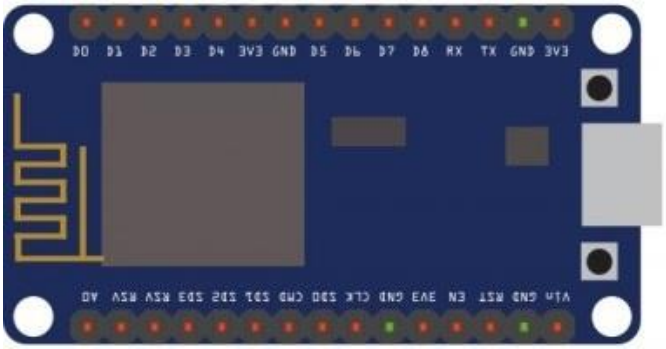
Note: you should name your Lua script “init.lua”, that ensures your ESP8266 executes your script every time it restarts.

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

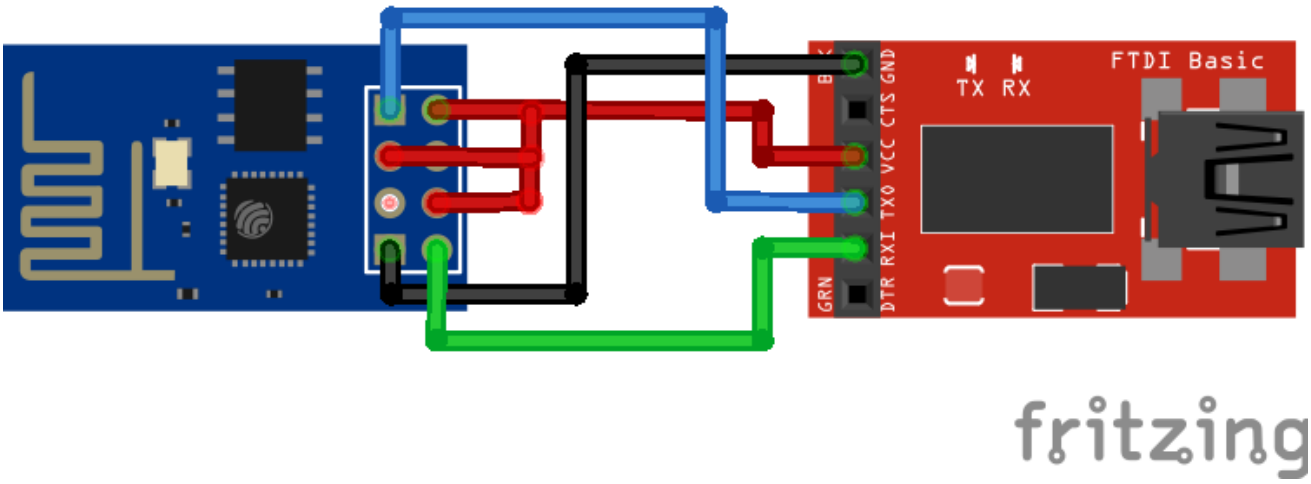
Uploading code to your ESP-12E NodeMCU Kit is simple, since it has built-in programmer. You plug your board to your computer and you don’t need to make any additional connections:



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.



Here's the connections:

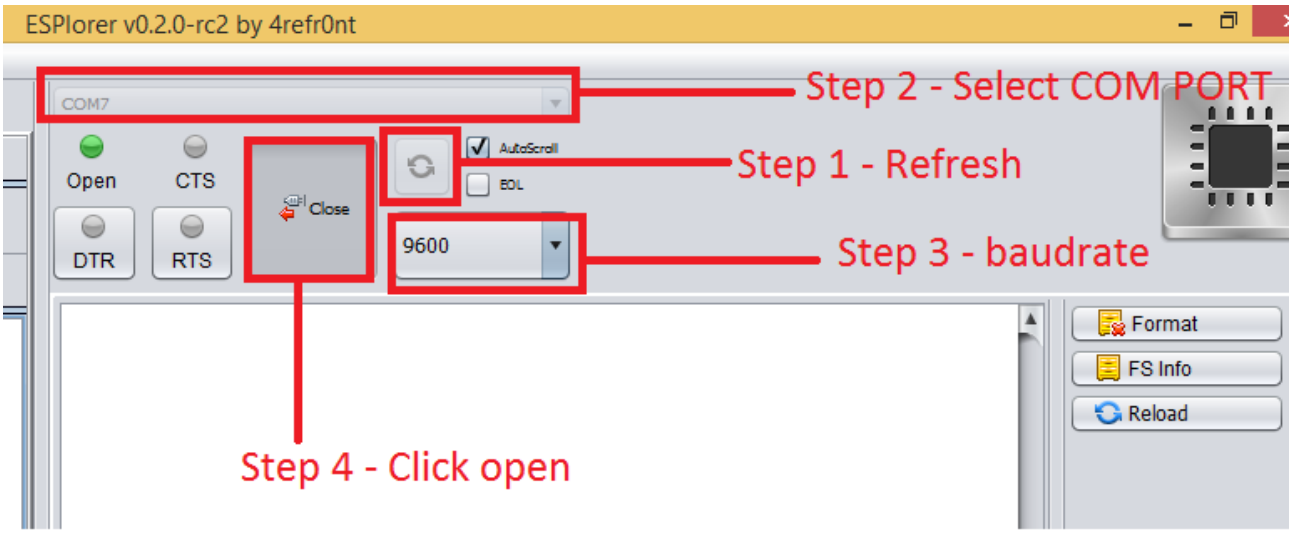
- RX -> TX
- TX -> RX
- CH_PD -> 3.3V
- GPIO 0 -> VCC
- VCC -> 3.3V
- GND -> GND

Note: the circuit above has GPIO 0 connected to VCC because we want to upload code.

Uploading init.lua script

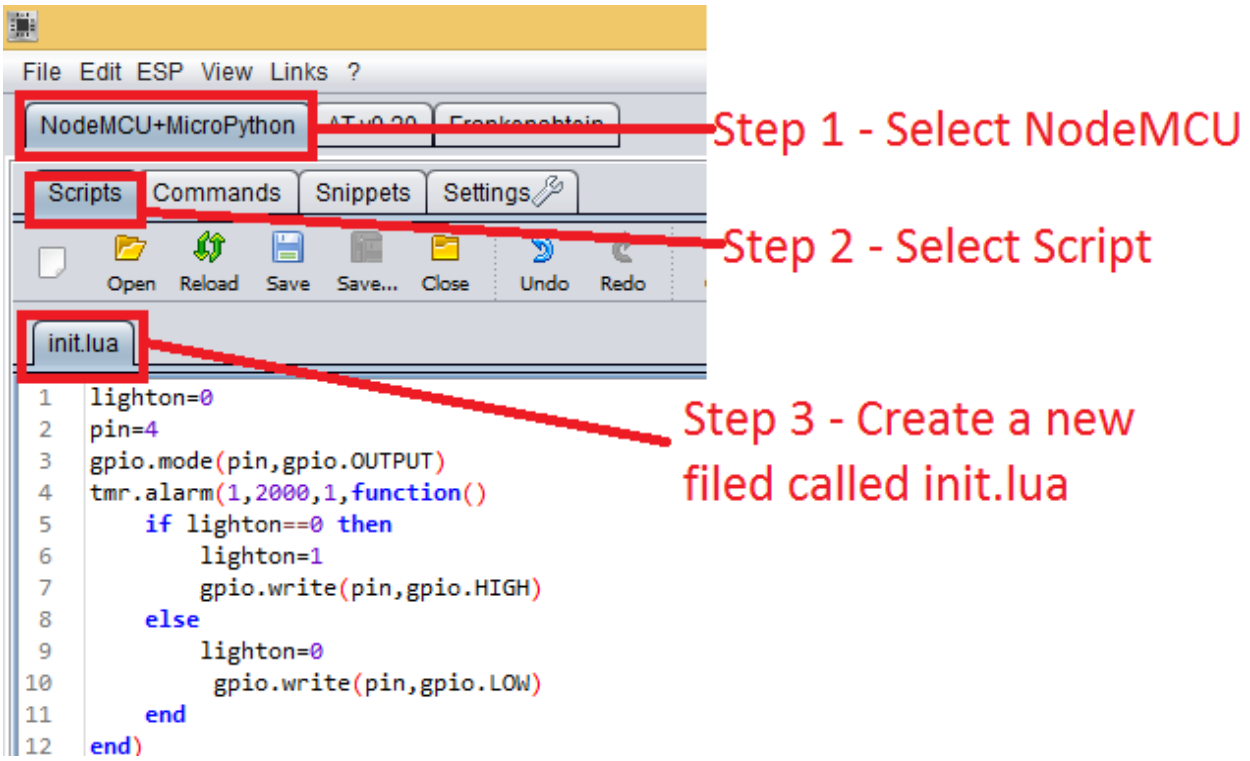
Having your ESP8266 connected to your computer, go to the ESPlorer IDE. Look at the top right corner of your ESPlorer IDE and follow these instructions:

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

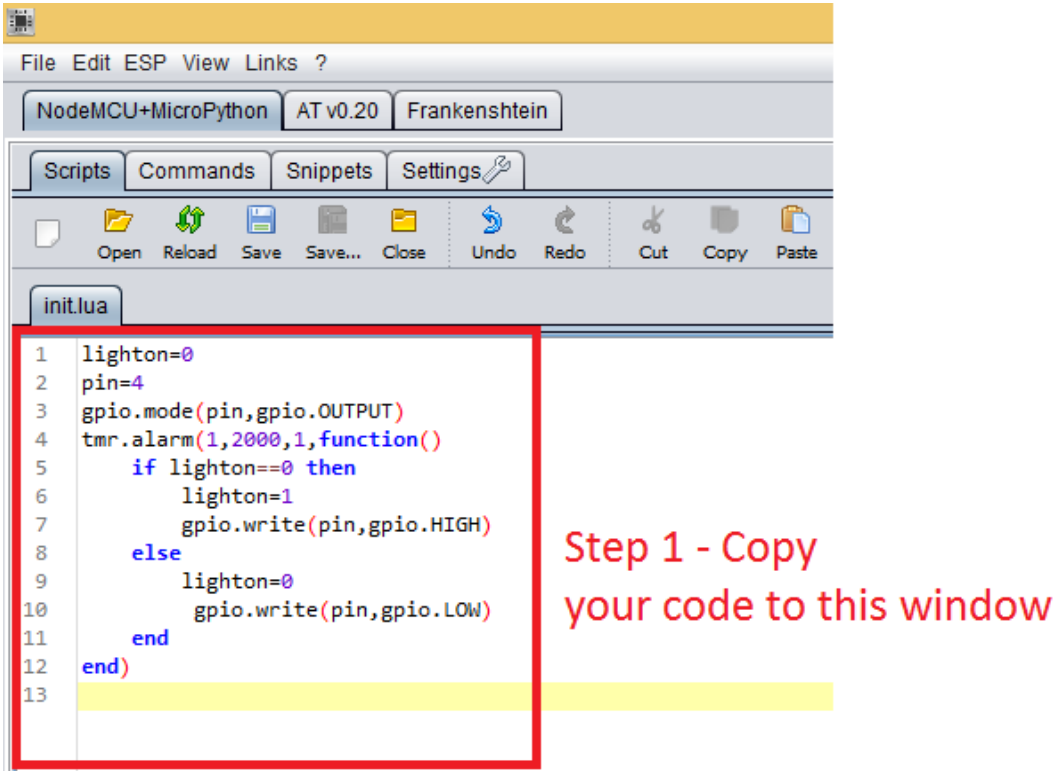


Then in the top left corner of the ESPlorer IDE, follow these instructions:

1. Select NodeMCU
2. Select Scripts
3. Create a new file called “init.lua”



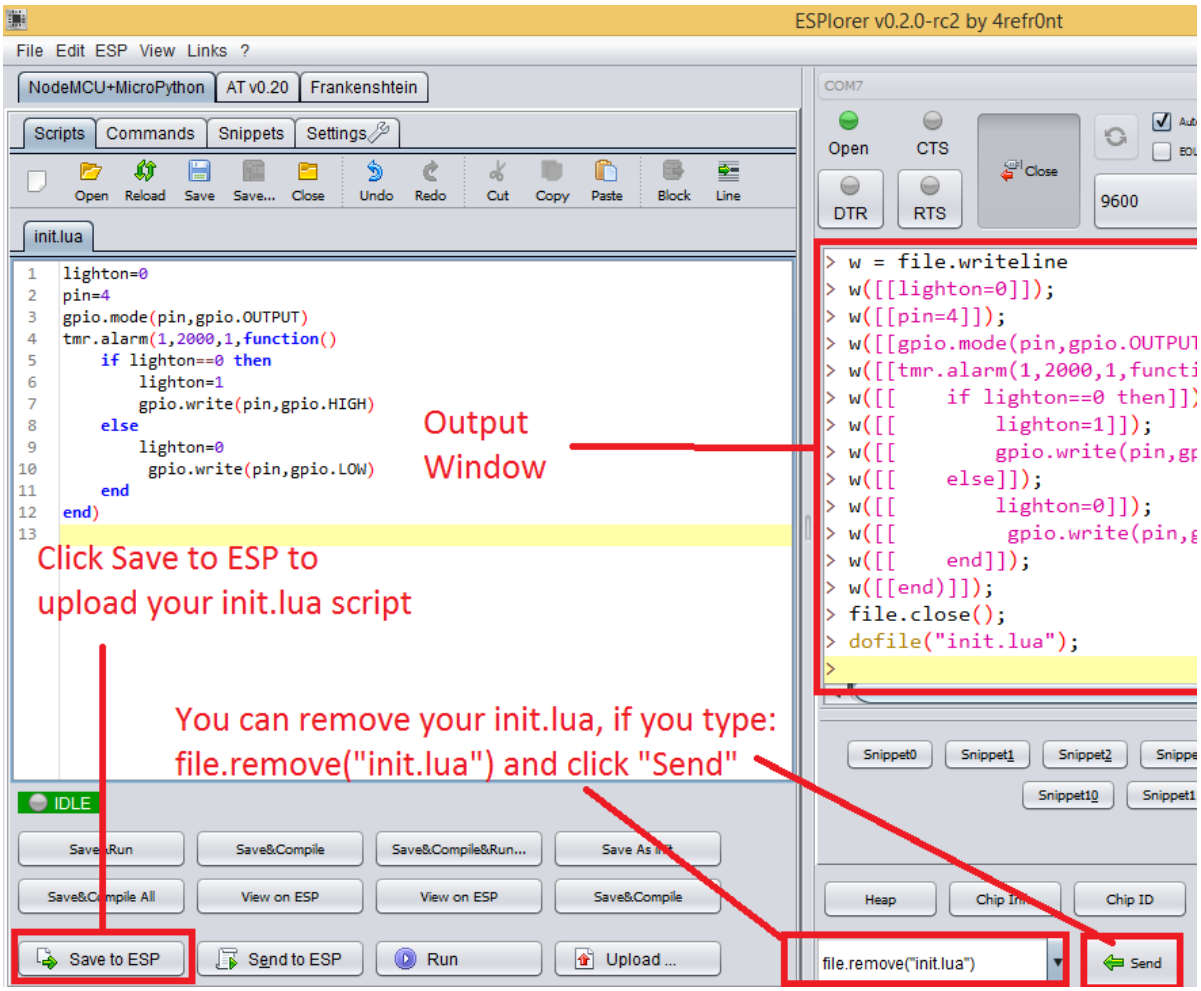
Copy the Lua script (which was created in the previous section) to the code window (as you can see in the figure below):



The 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.



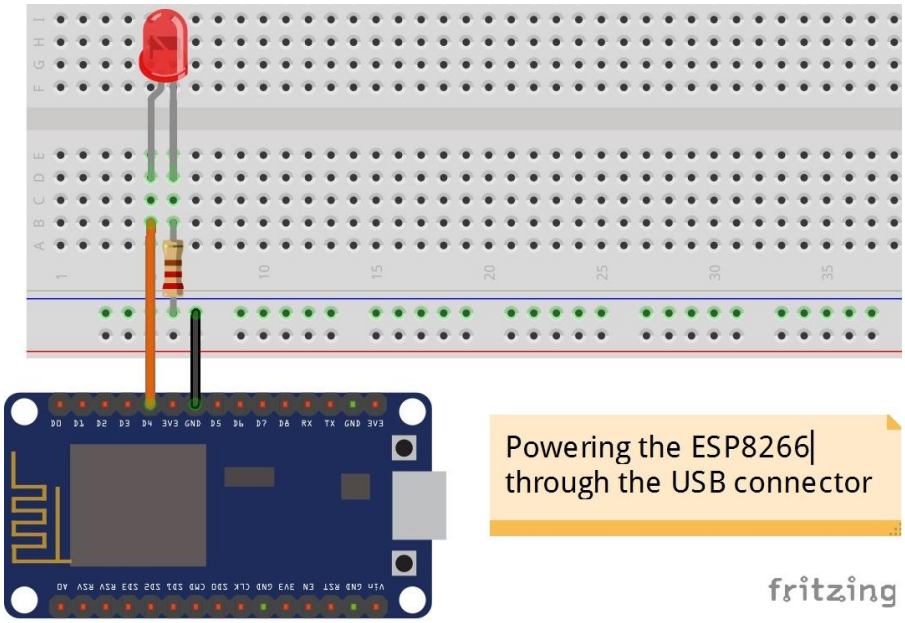
Note: you can easily delete the "init.lua" file from the ESP. Simply type *file.remove("init.lua")* and press the button "Send" (see figure above). Or you can type the command *file.format()* to remove all the files saved in your ESP8266.

Final Circuit

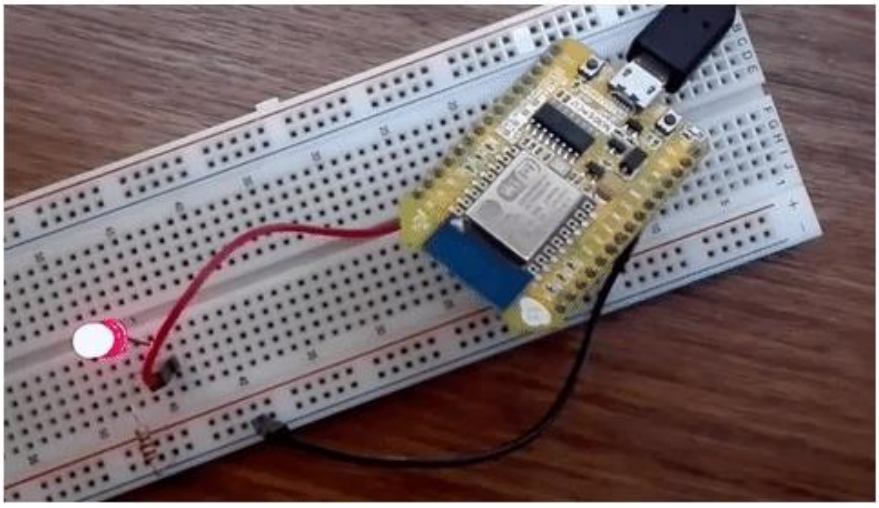
After uploading the code, unplug the ESP from your computer. Next, change the wiring to match the following diagrams.

Final ESP-12E circuit

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

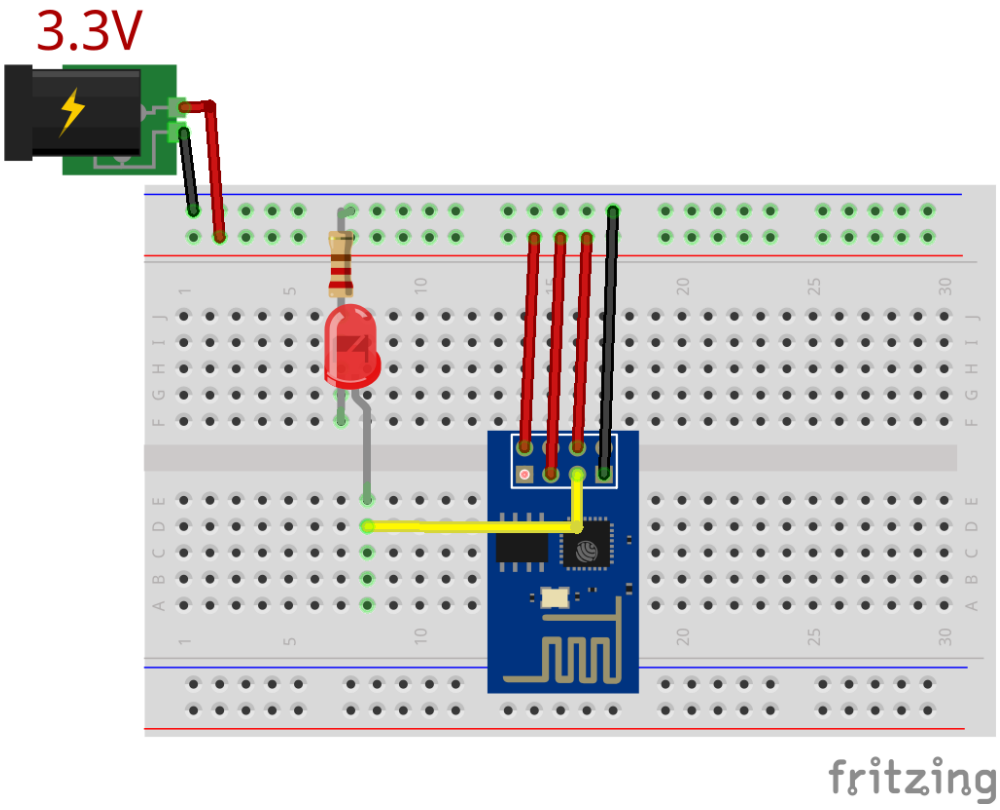


Restart your ESP8266.



Final ESP-01 circuit

Then, apply power from a 3.3V source or use your FTDI Programmer to the ESP.



Restart your ESP8266.

Congratulations, you've made it! Your LED should be blinking every 2 seconds.

Unit 3
Lua Programming Language –The Basics

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 Cele.

More details about this program language can be found here: http://en.wikipedia.org/wiki/Lua_%28programming_language%29

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 *PIN* 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 bolean_value then
    -- if the bolean_value is true
else
    -- if the bolean_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

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:

- http://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 it's 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).

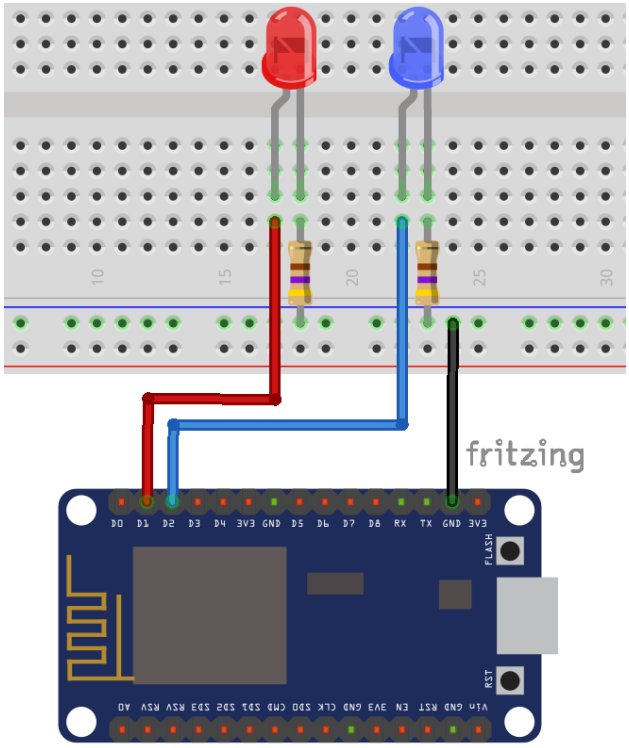
To create an interrupt, this is what you would do:

```
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

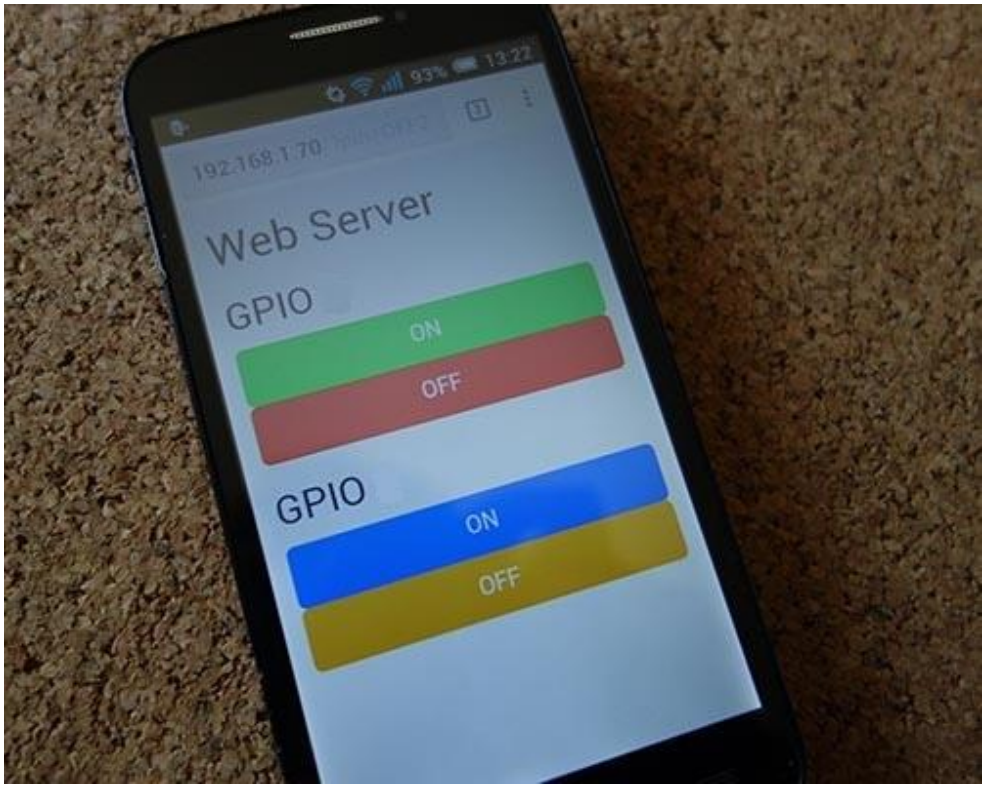


Unit 5 - Web Server with ESP8266

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://gist.github.com/RuiSantosdotme/86d6175c438687043707eff7b2143d25

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:

```
 8
 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          but = but.."<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.
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-
33          buf = buf.."<div class=\"col-md-2\"><a href=\"?pin=OFF1\" class=\"btn btn-block btn-lg btn
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-
38          buf = buf.."<div class=\"col-md-2\"><a href=\"?pin=OFF2\" class=\"btn btn-block btn-lg btn
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.

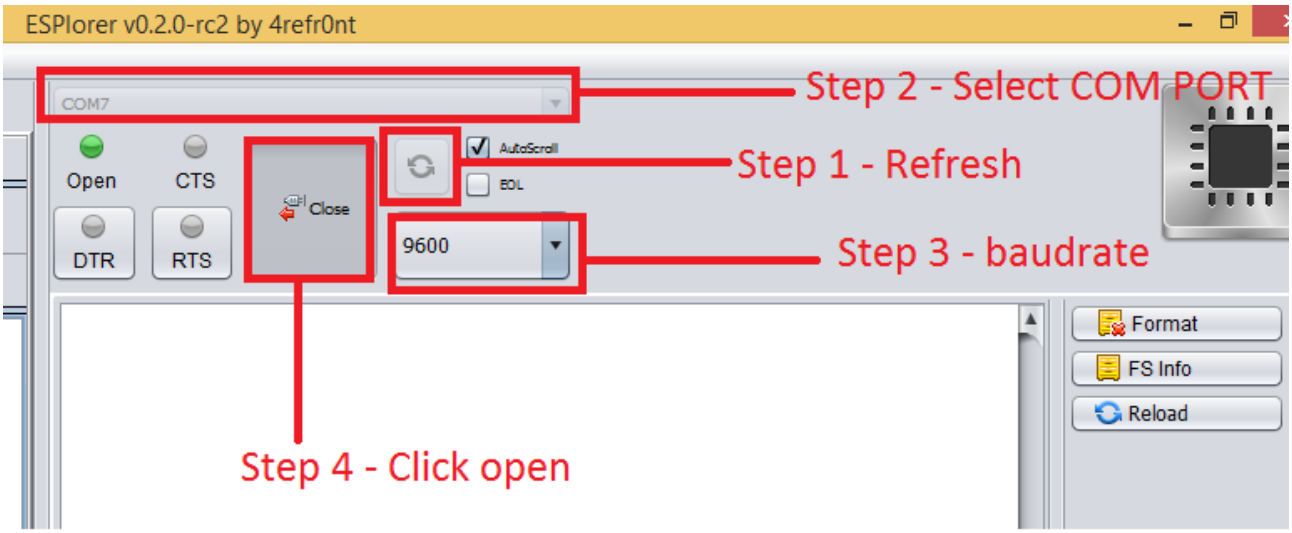
Let's see an example. When you click the button OFF from the GPIO 5 you open this URL: http://192.168.7.2/?pin=OFF1. Your Lua code checks that URL and with some *if… else* statements it knows that you want your GPIO 5 (which is defined as *led1*) to turn *LOW*.

```
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

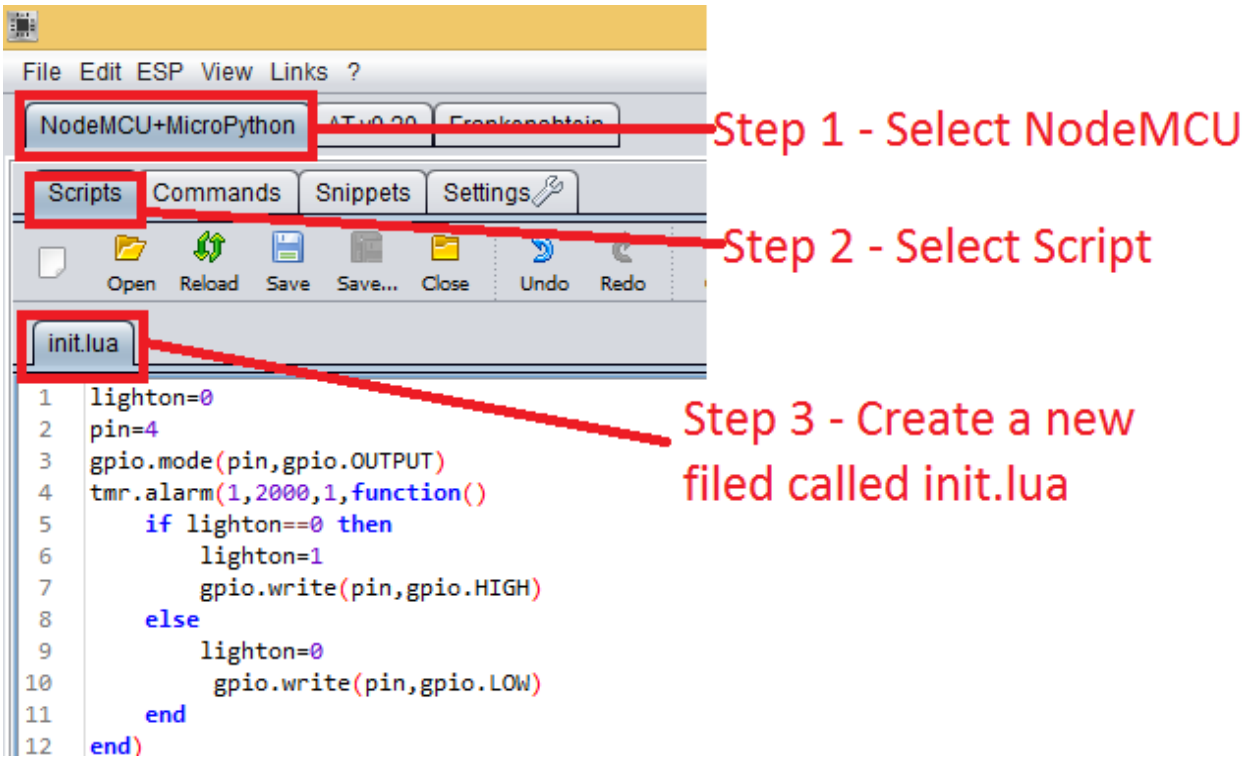
Having your ESP8266 connected to your computer, go to the ESPlorer IDE. Look at the top right corner of your ESPlorer IDE and follow these instructions:

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

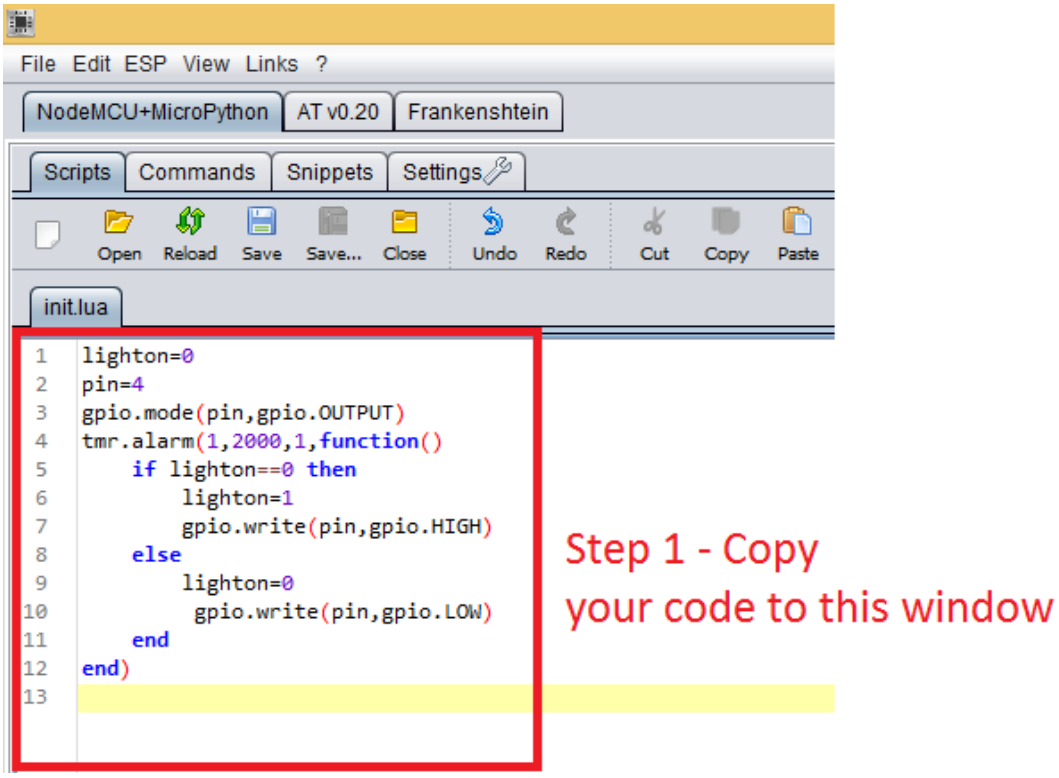


Then in the top left corner of the ESPlorer IDE, follow these instructions:

1. Select NodeMCU
2. Select Scripts
3. Create a new file called “init.lua”



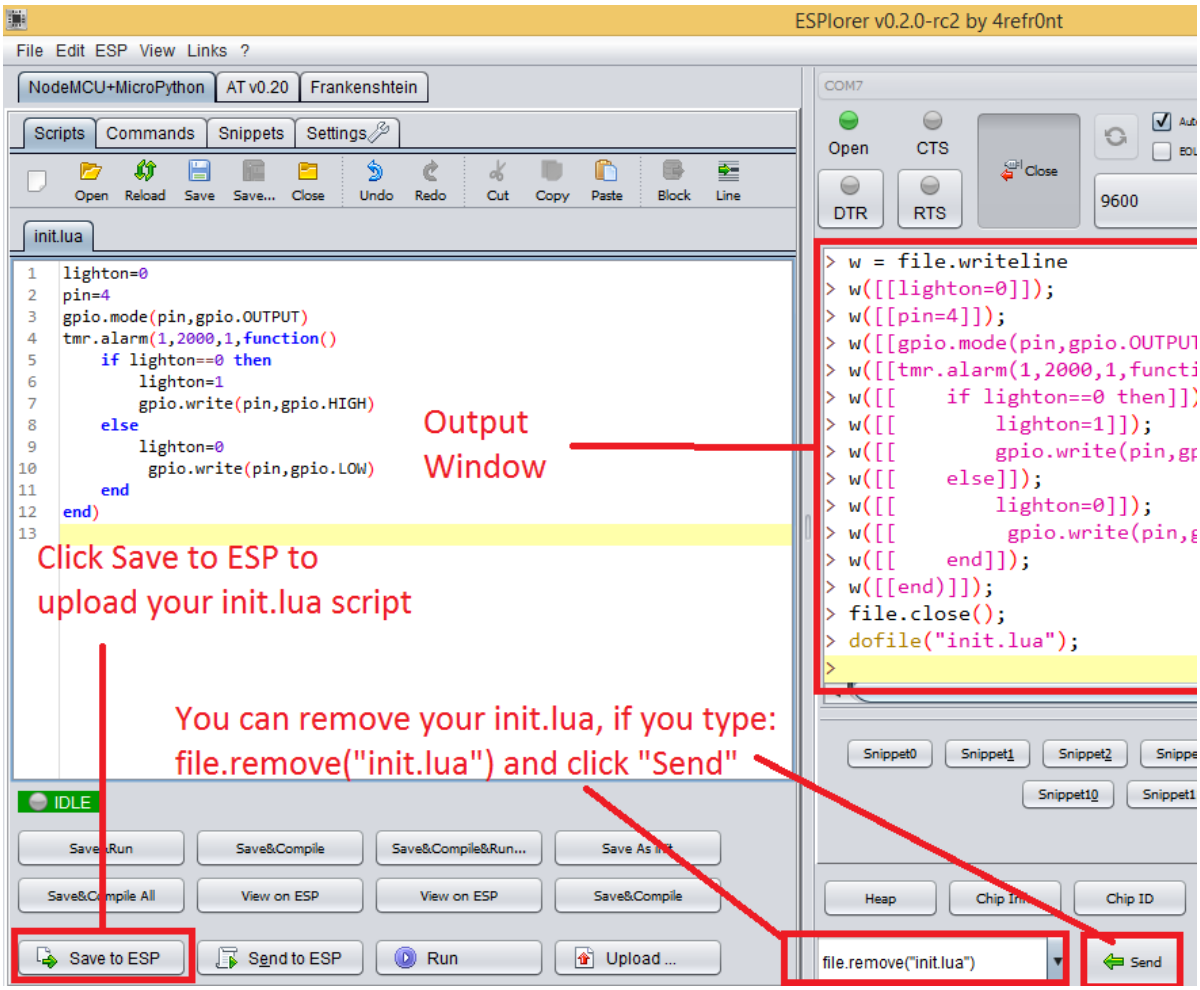
Copy the Lua script (which was created in the previous section) to the code window (as you can see in the figure below):



The 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.

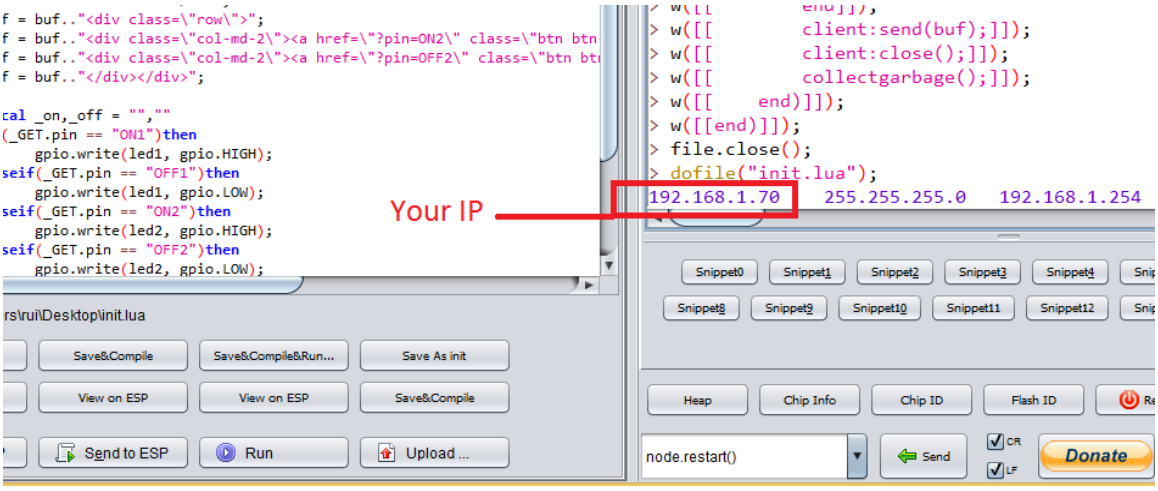


Note: you can easily delete the "init.lua" file from the ESP. Simply type *file.remove("init.lua")* and press the button "Send" (see figure above). Or you can type the command *file.format()* to remove all the files saved in your ESP8266.

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.

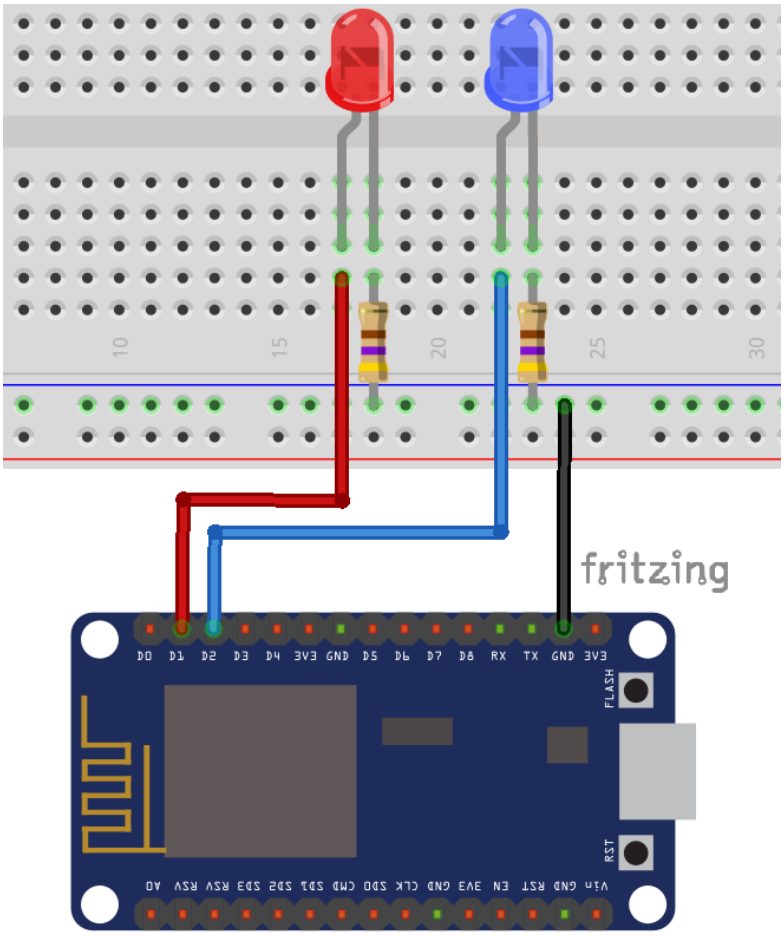
The IP that that you want is the first one, in my case it's: 192.168.1.70. Your IP should be different, **save your ESP8266 IP** so you can access it later in this Unit.



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 schematics (you can use 270 Ohm resistors for the LEDs).

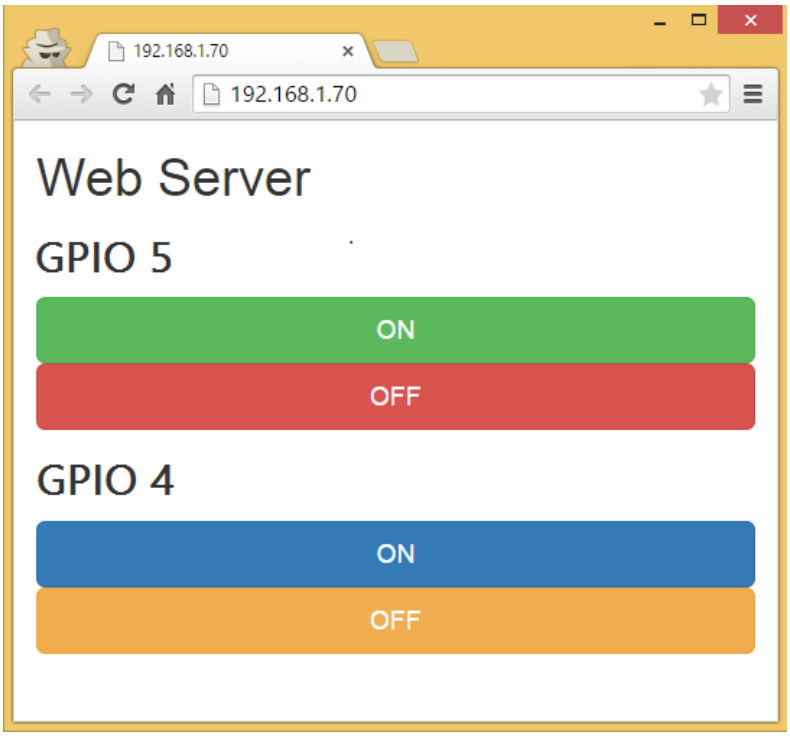


Accessing Your Web Server

Follow the next instructions before accessing your web server:

1. Restart your ESP8266 module
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 this 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://gist.github.com/RuiSantosdotme/f84545fc834fbc788ae19e41ac3a1910

Encoding Your Username and Password

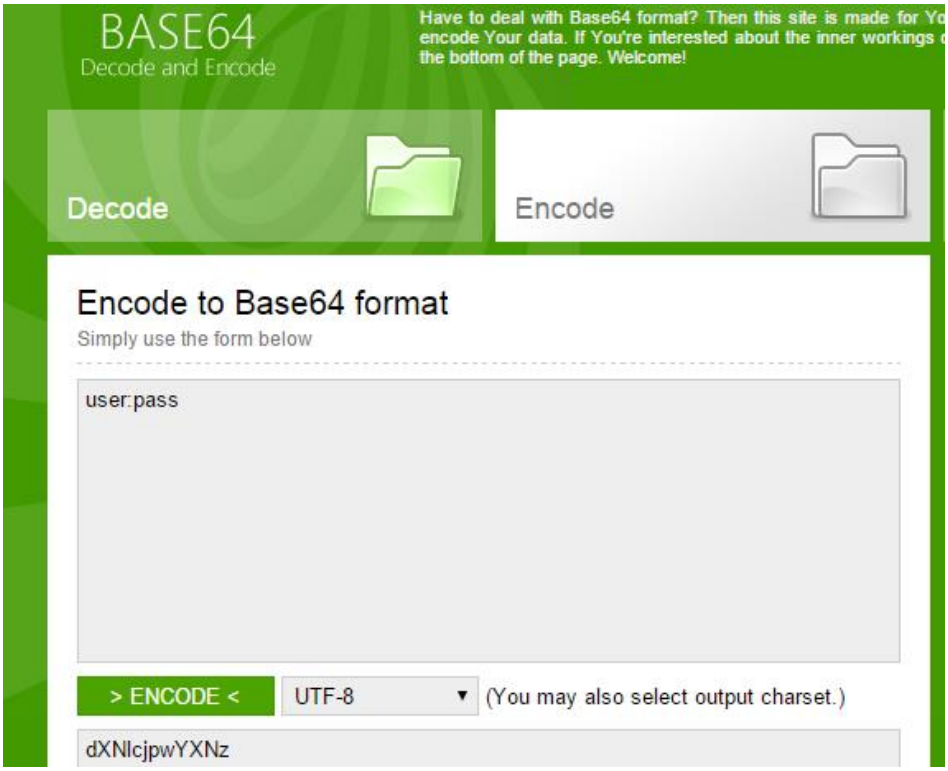
At this point if you upload the code I've mentioned in the preceding section, your username is *user* and your password is *pass*. I'm sure you want to change and customize this example with your own credentials.

Go to the following URL: https://www.base64encode.org. In the first field, type the following:

your_username:your_password

Note: you actually 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.

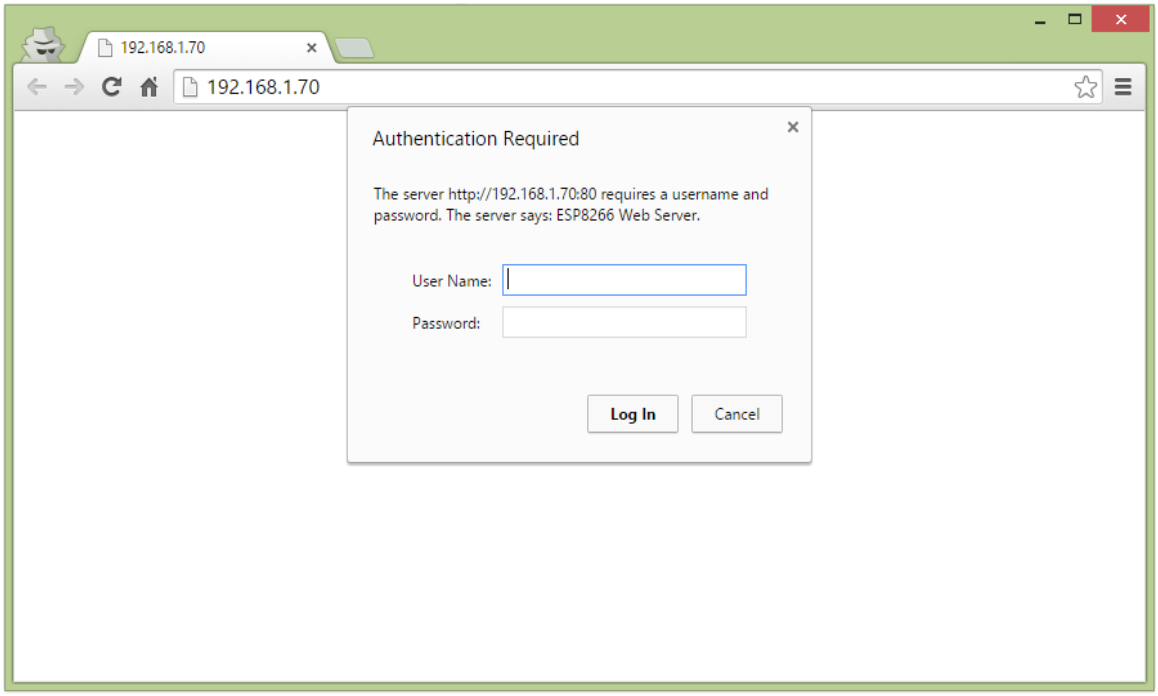
```
if (auth == nil or auth ~= "dXNlcjpwYXNz")then --user:pass
```

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. You have two options...

Option A – PowerSwitch Tail II

The easiest route is to get yourself a PowerSwitch Tail II (www.powerswitchtail.com), which provides a safe way of dealing with high-voltage devices

The way this bulky component works is quite straightforward. Rather than connecting a house appliance directly to the wall, you connect it to the PowerSwitch Tail II which plugs into the wall.

The PowerSwitch Tail II has three pins that enable it to behave like a simple digital logic device. You connect the PowerSwitch Tail to an output GPIO of the ESP8266.

Your output pin will send a signal that's either HIGH or LOW. Whenever the signal is HIGH, there's a connection to the wall socket; when it's LOW, the connection is broken, as though the device were unplugged.

Here's how you should connect your PowerSwitch Tail II

PowerSwitch Tail II Pin Number	Signal Name	ESP8266 Pins
1	+in	GPIO 5 or GPIO 4
2	-in	GND
3	GND	Not used

Search for the PowerSwitch Tail II's instruction sheet for more details on how to wire it up.

Option B – Relay

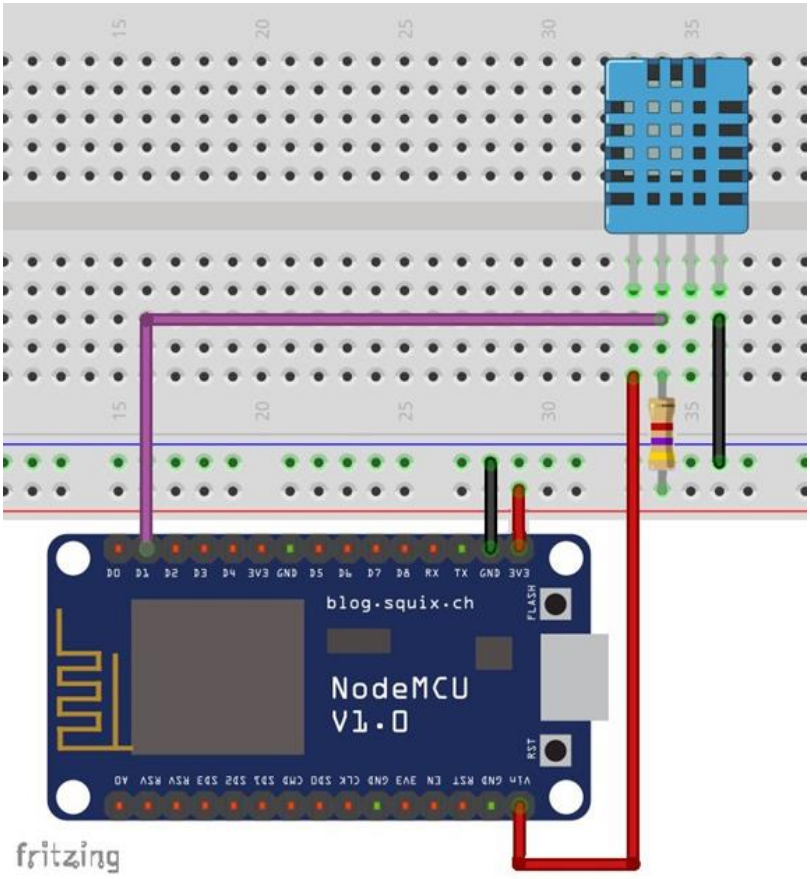
There's another way to have your ESP8266 control a house appliance, but that method is a bit more complicated. It requires a bit of extra knowledge and wariness because you're dealing with alternating current (AC), and it involves relay modules.

I won't discuss this project in great detail, but here's a good tutorial: http://randomnerdtutorials.com/guide-for-relay-module-with-arduino/

The preceding link takes you to my blog and shows how to control relays using an Arduino, but you can apply the same concepts to your ESP module.

WARNING: always be safe when dealing with high voltages, if you don't know what you're doing ask someone who does.

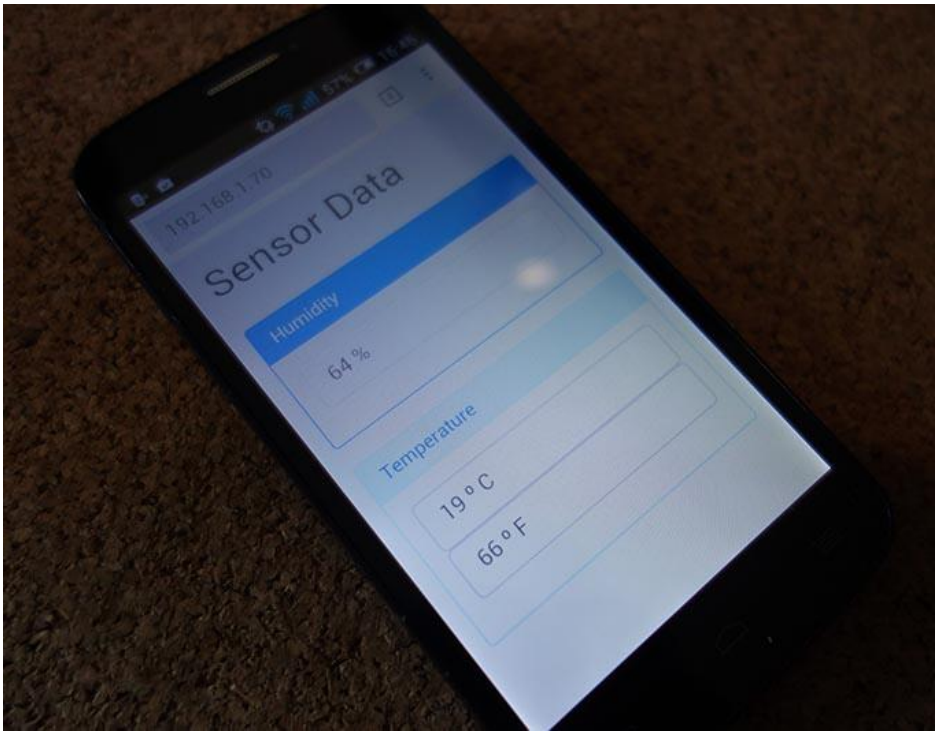
Unit 6
Displaying Temperature and Humidity on a Web Page



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:

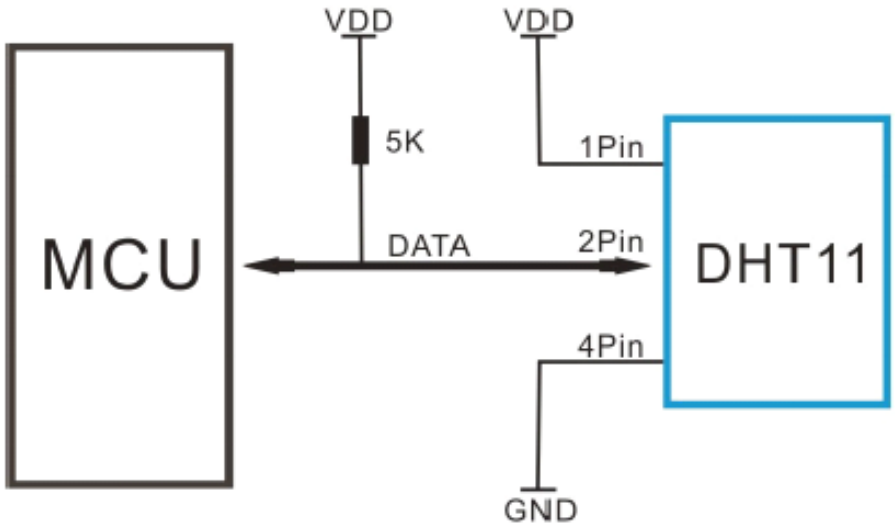


Using DHT11 Temperature and Humidity Sensor

The DHT sensors are relative cheap sensors for measuring temperature and humidity.

These sensors contain a chip inside that does analog to digital conversion and spits out a digital signal with the temperature and humidity.

With any microcontroller (MCU) these signals are fairly easy to read.



DHT11 specifications

- Absolut accuracy: ±5%
- Repeatability: ±1%
- Long term stability: ±1% per year
- Price: $1 to $5

Pins

- VCC (3.3V)
- Data OUT
- Don’t connect
- 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://gist.github.com/RuiSantosdotme/078ab06f6bf63b290c1f100a8c5e729a

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()*.

```
38
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>\
    <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..' d
    <input type="text" class="form-control" value="'..fare..'.'..fare_dec..' deg F">\
    </div></div></div></div></div></div></html>')
```

The figure above displays temperature and humidity in a nice looking mobile responsive web page.

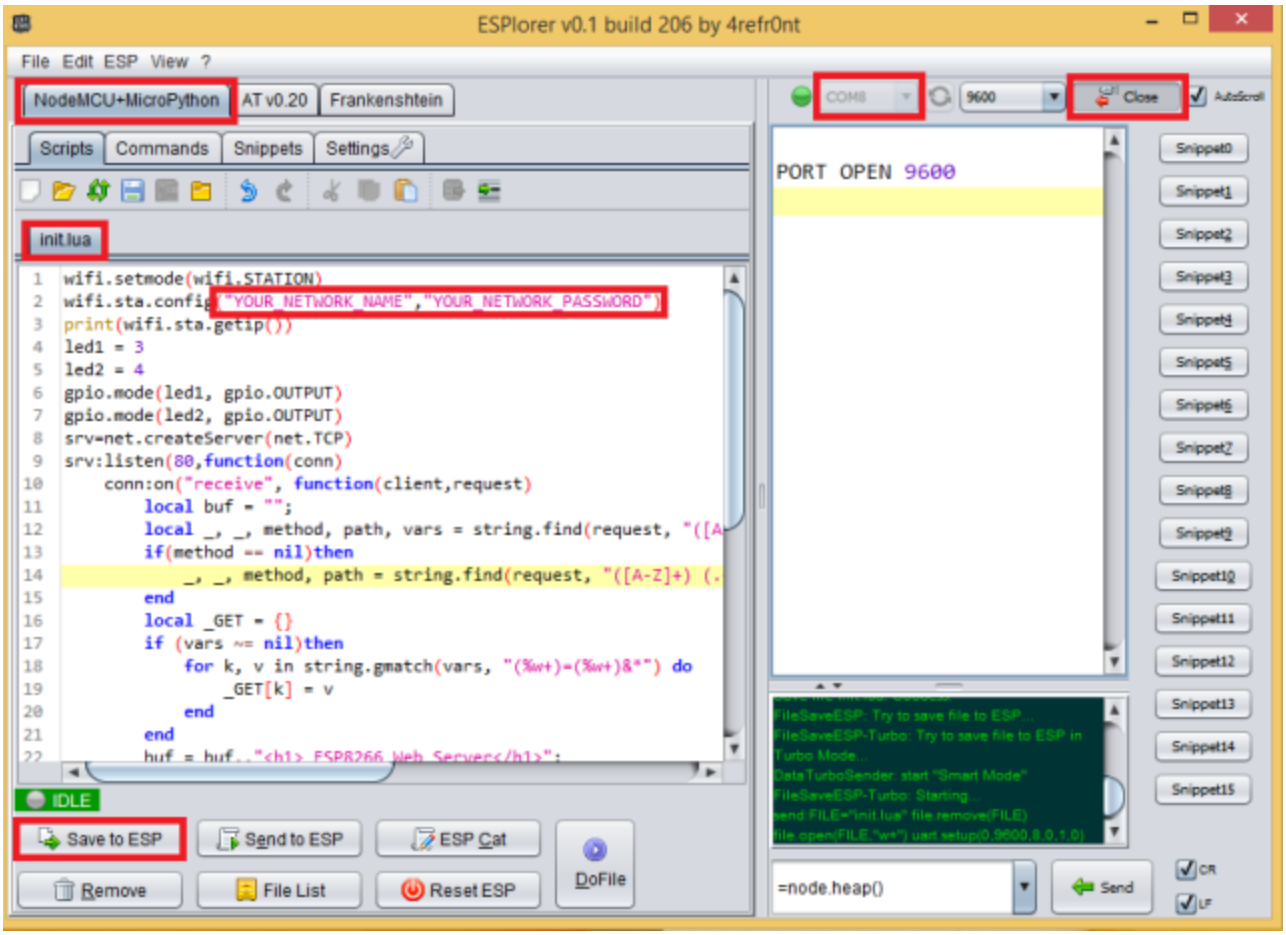
Uploading init.lua

Follow these instructions to upload a Lua script:

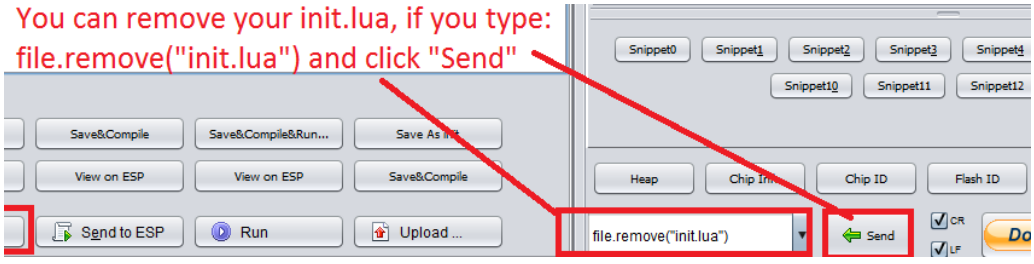
1. Connect your ESP8266-12E that has built-in programmer to your computer
2. Select your ESP8266-12E port

3. Press Open/Close
4. Select NodeMCU+MicroPtyhon 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.

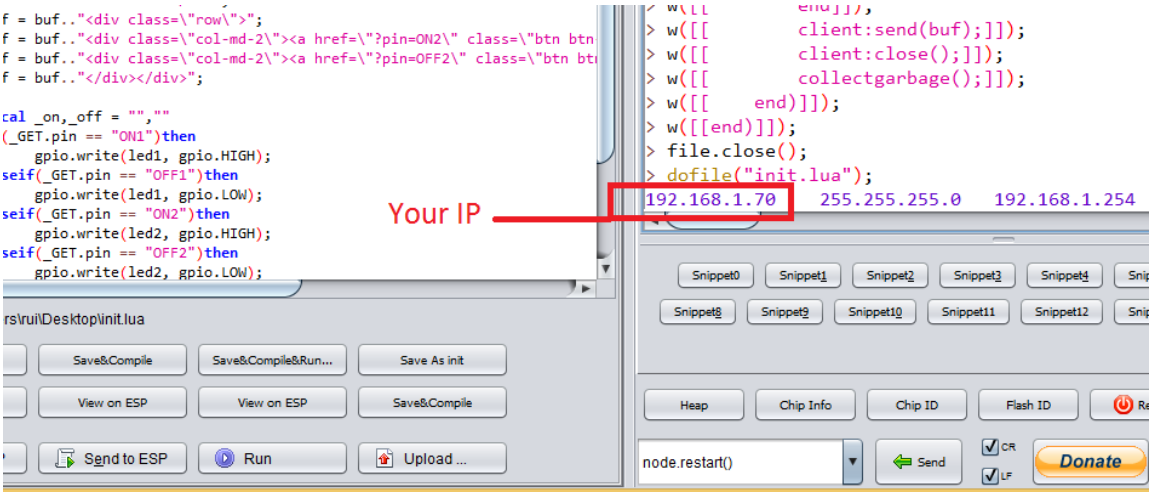


Note: you can easily delete the "init.lua" file from the ESP. Simply type *file.remove("init.lua")* and press the button "Send" (see figure above). Or you can type the command *file.format()* to remove all the files saved in your ESP8266.

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.

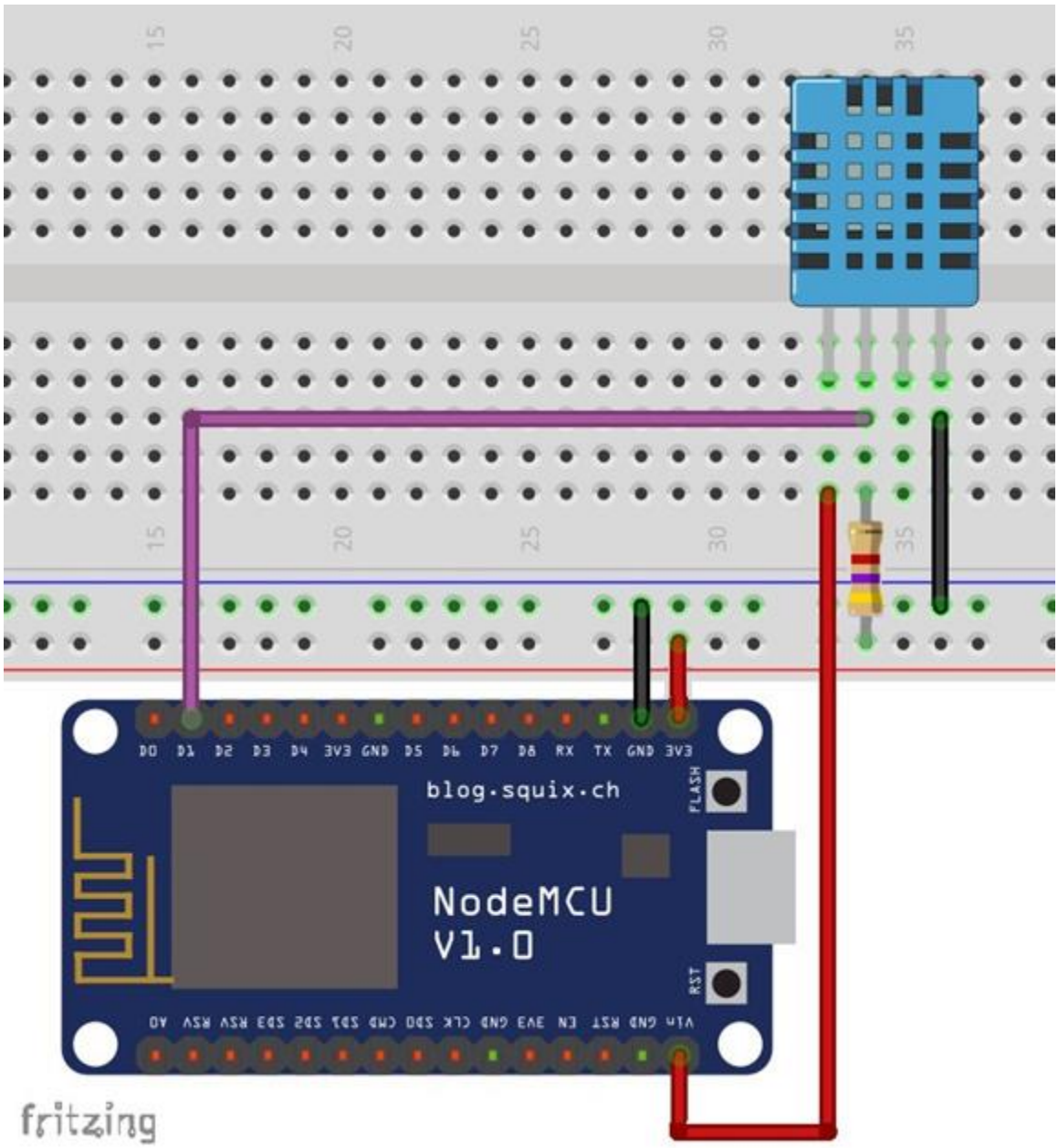


Note: if your IP address doesn't appear in your output window you can send the command *print(wifi.sta.getip())* to print your ESP8266 IP.

Final Circuit

After uploading your code to your ESP8266, follow the next schematics. 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.

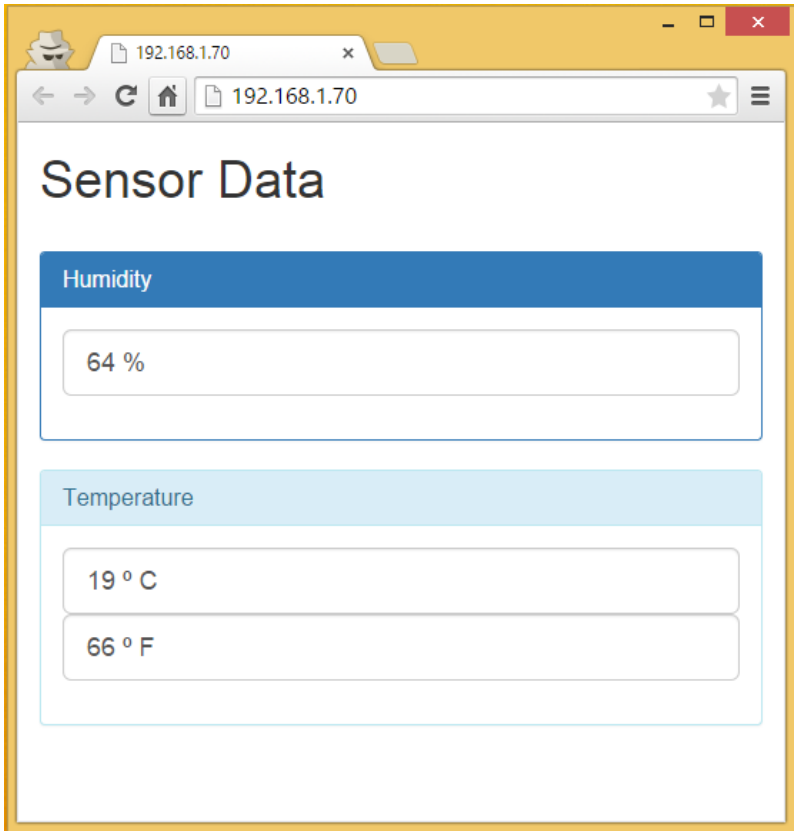


Accessing Your Web Server

Follow the next instructions before accessing your web server:

1. Restart your ESP8266 module
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.

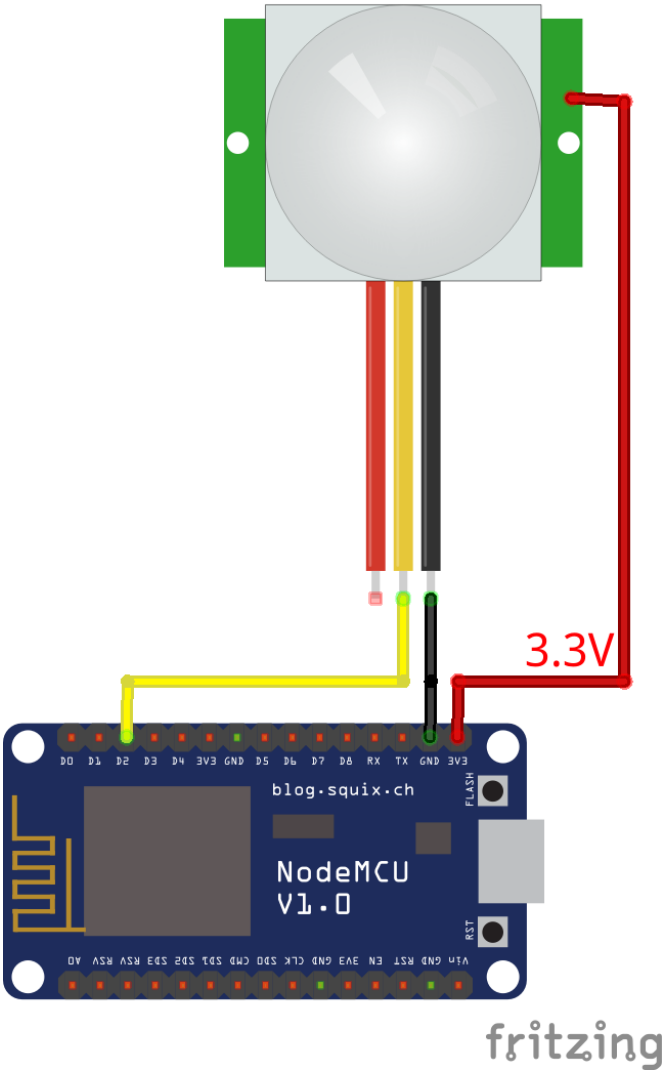
You know how to establish a serial communication with the ESP8266. You can attach a bunch of sensors to your Arduino and connect it to your ESP.

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



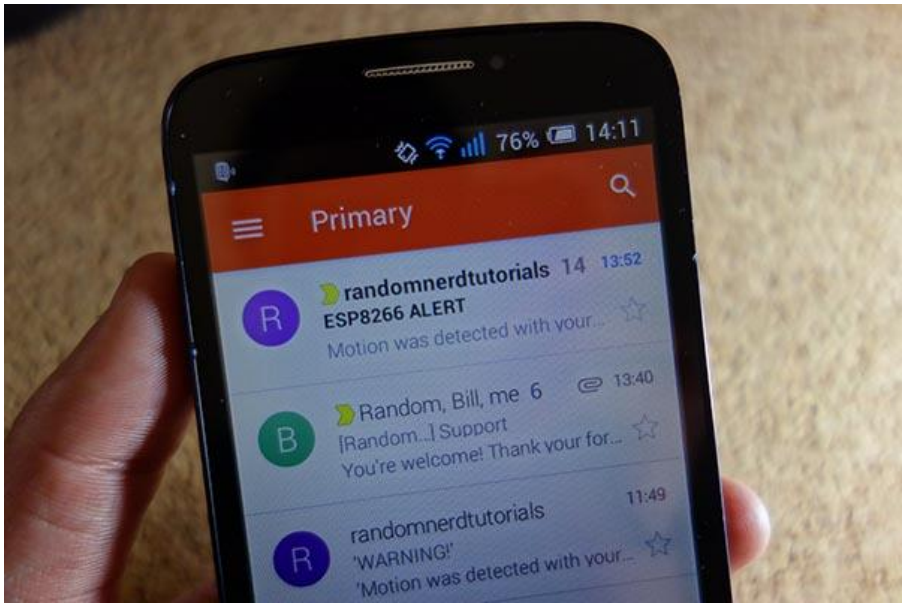
Unit 7 - Email Notifier with ESP8266 and PIR Motion Sensor

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. I'll basically show how to build a home surveillance system for $6.

In order to accomplish this task, you have to sign up for one free service called IFTTT which stands for "If This Then That".

IFTTT is a platform that gives you creative control over dozens of products and apps. You can make apps work together. For example when you send a request to IFTTT, it triggers a recipe that sends you an email alert.

Here's the end result:

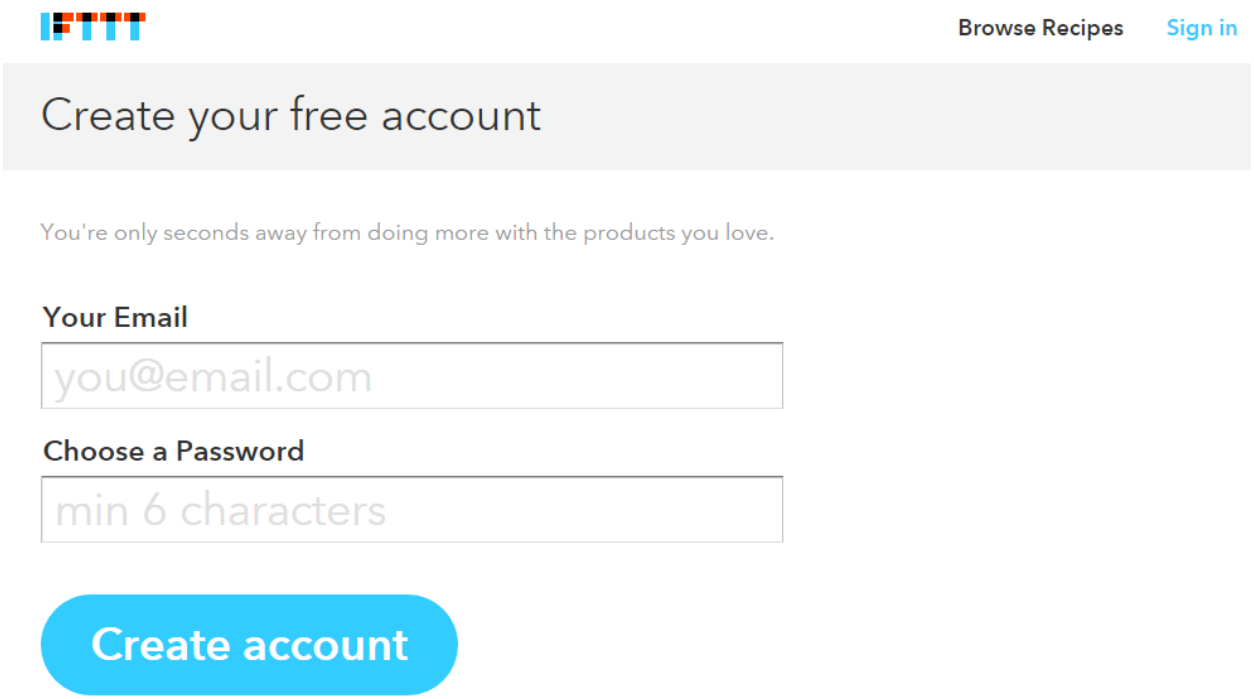


Creating Your IFTTT Account

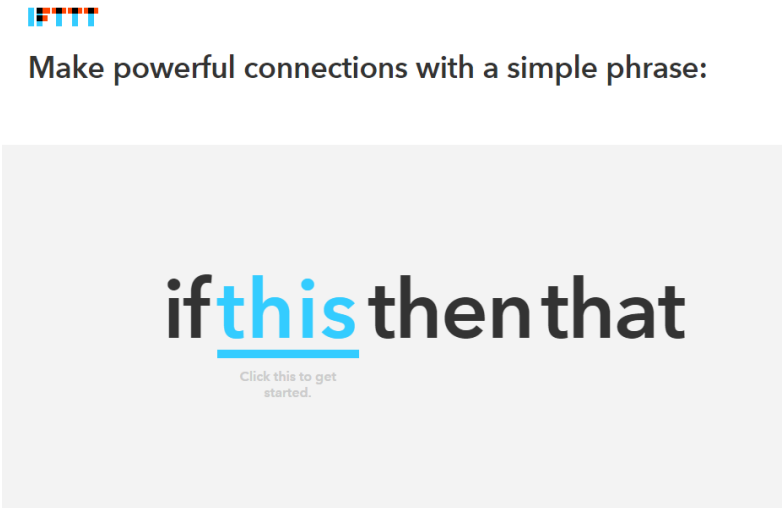
Creating an account on IFTTT is free.

Go the official site: https://ifttt.com and click the “Sign Up” button in the middle of the page.

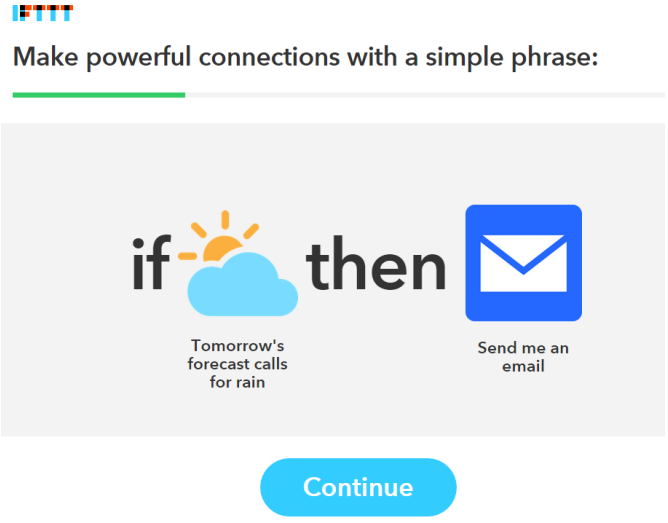
Complete the form with your personal information (see figure below) and create your account.



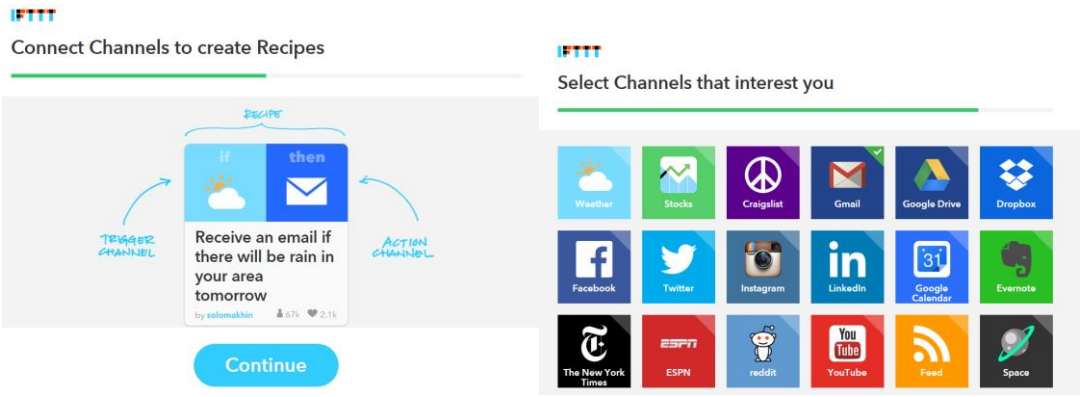
After creating your account, follow their getting started tutorial by clicking the word “this”:



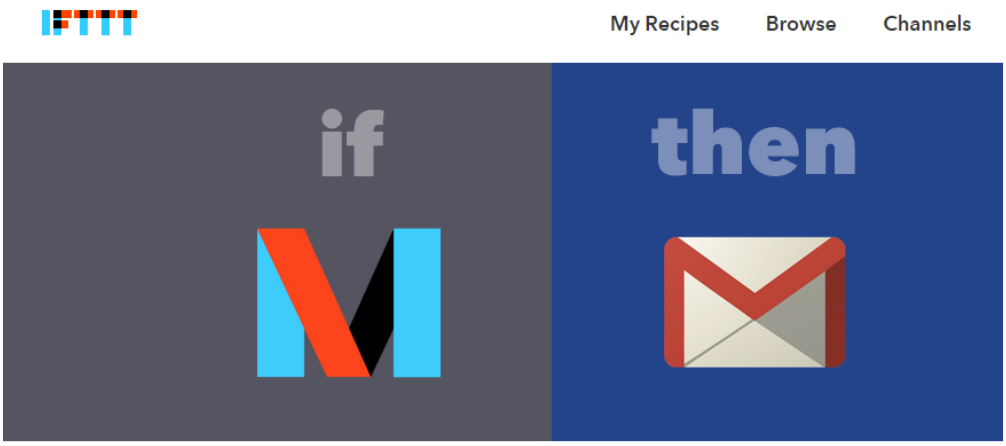
Then wait a few seconds and click the big blue button “Continue”:



Click “Continue” a few more times to finish their introductory tutorial:



I've created a recipe that you can use that integrates perfectly in this project. If you're logged in at IFTTT and you open this URL: https://ifttt.com/recipes/315426-esp8266-email-notifier you can use my recipe instantly.

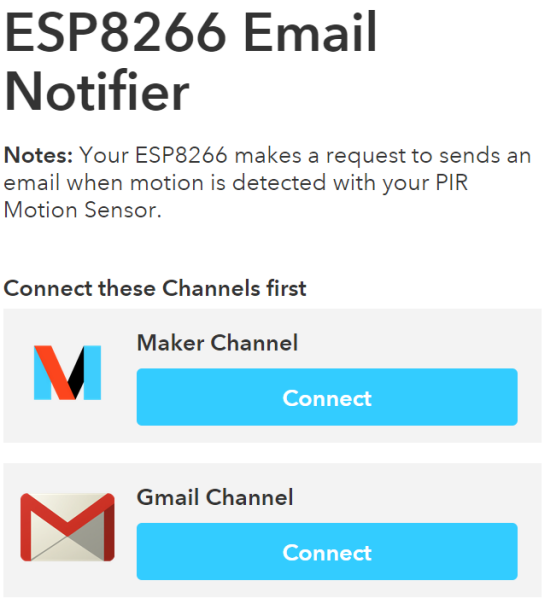


ESP8266 Email Notifier

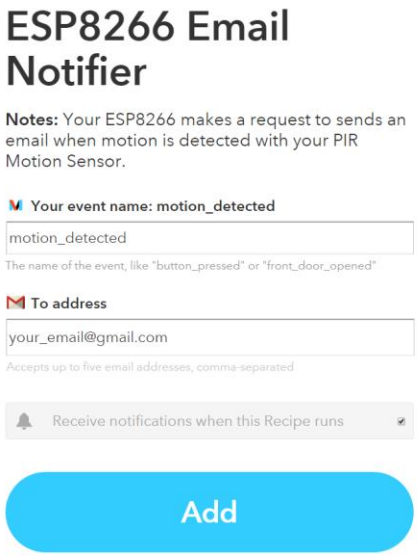
Notes: Your ESP8266 makes a request to sends an email when motion is detected with your PIR Motion Sensor.

Next scroll down that page and follow these instructions to make it work for you:

1. Connect your account to the Maker Channel
2. Then connect your account to your Gmail Channel



A new page loads (see figure below) when you finish connecting your account to Maker Channel and Gmail Channel.

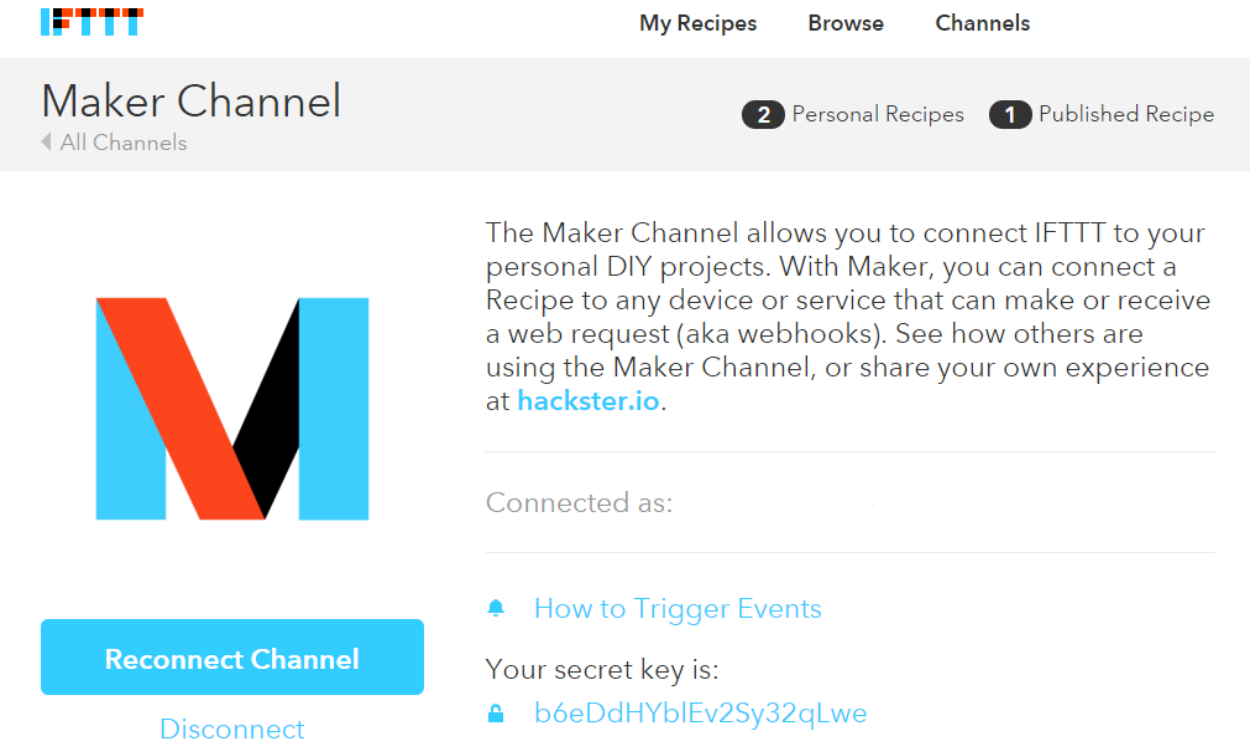


Fill the recipe with your own information. Follow these instructions:

1. Type “motion_detected” without the quotes in your event name
2. Type your email address in the “To address” field (that’s the email address where you’re going to receive the email alerts)
3. Press the “Add” button

Go to this URL: https://ifttt.com/maker. Copy your secret key to a safe place (you’ll need them later in this Unit).

My secret key is: *b6eDdHYblEv2Sy32qLwe*



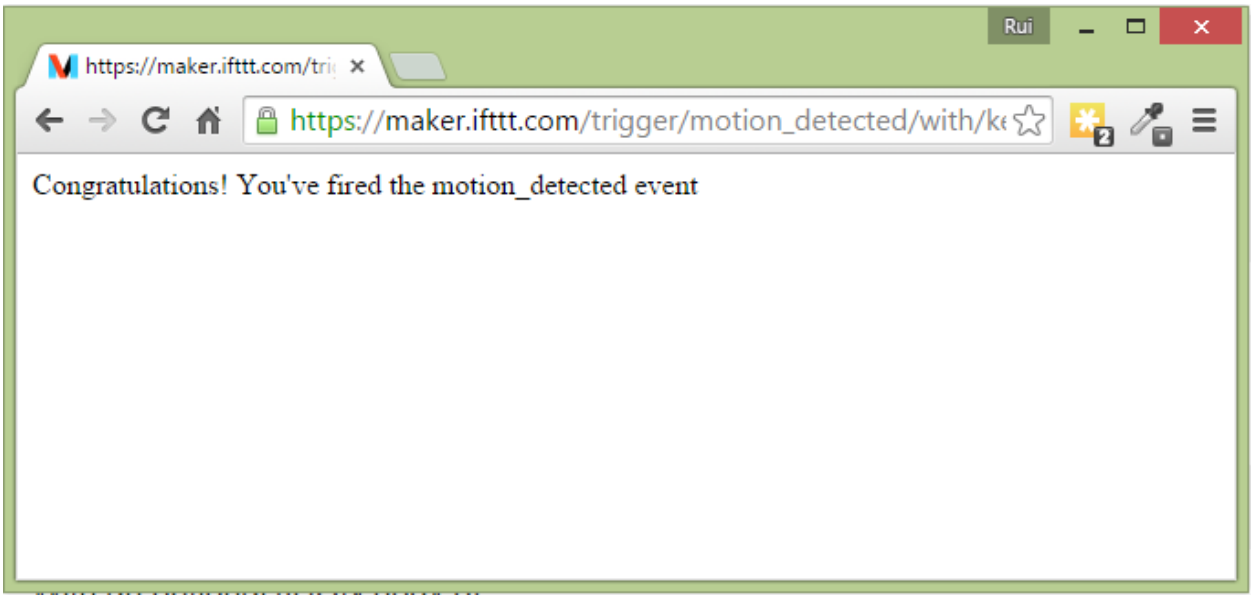
Let’s test if your request is working properly. Replace **YOUR_API_KEY** in the following URL:

https://maker.ifttt.com/trigger/motion_detected/with/key/**YOUR_API_KEY**

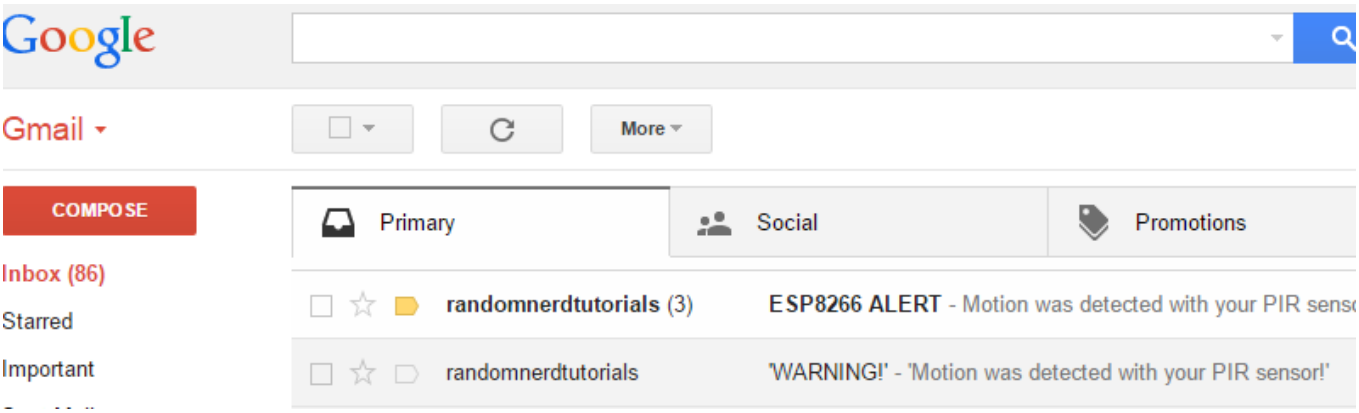
With your API KEY:

https://maker.ifttt.com/trigger/motion_detected/with/key/**b6eDdHYblEv2Sy32qLwe**

Copy the URL with your API KEY to your browser.



You should see something like the preceding figure. Then, go to your email and a new email should be in the inbox.



If you got the email in your inbox, lucky you! Most things don't work right at the first time. If you didn't receive the email, check your spam folder or re-check your IFTTT recipe.

Now, let's make your ESP8266 send emails for you 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://gist.github.com/RuiSantosdotme/093389a650666b98cb312971c3c30be9

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 *INTERRUP*. That's where you're going to attach your PIR Motion sensor.

```
wifi.setmode(wifi.STATION)
wifi.sta.config("YOUR_NETWORK_NAME","YOUR_NETWORK_PASSWORD")
pin = 2
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/b6eDdHYblEv2Sy32qLwe** HTTP/1.1\r\nHost: maker.ifttt.com\r\nConnection: keep-alive\r\nAccept: */*\r\n\r\n") end)

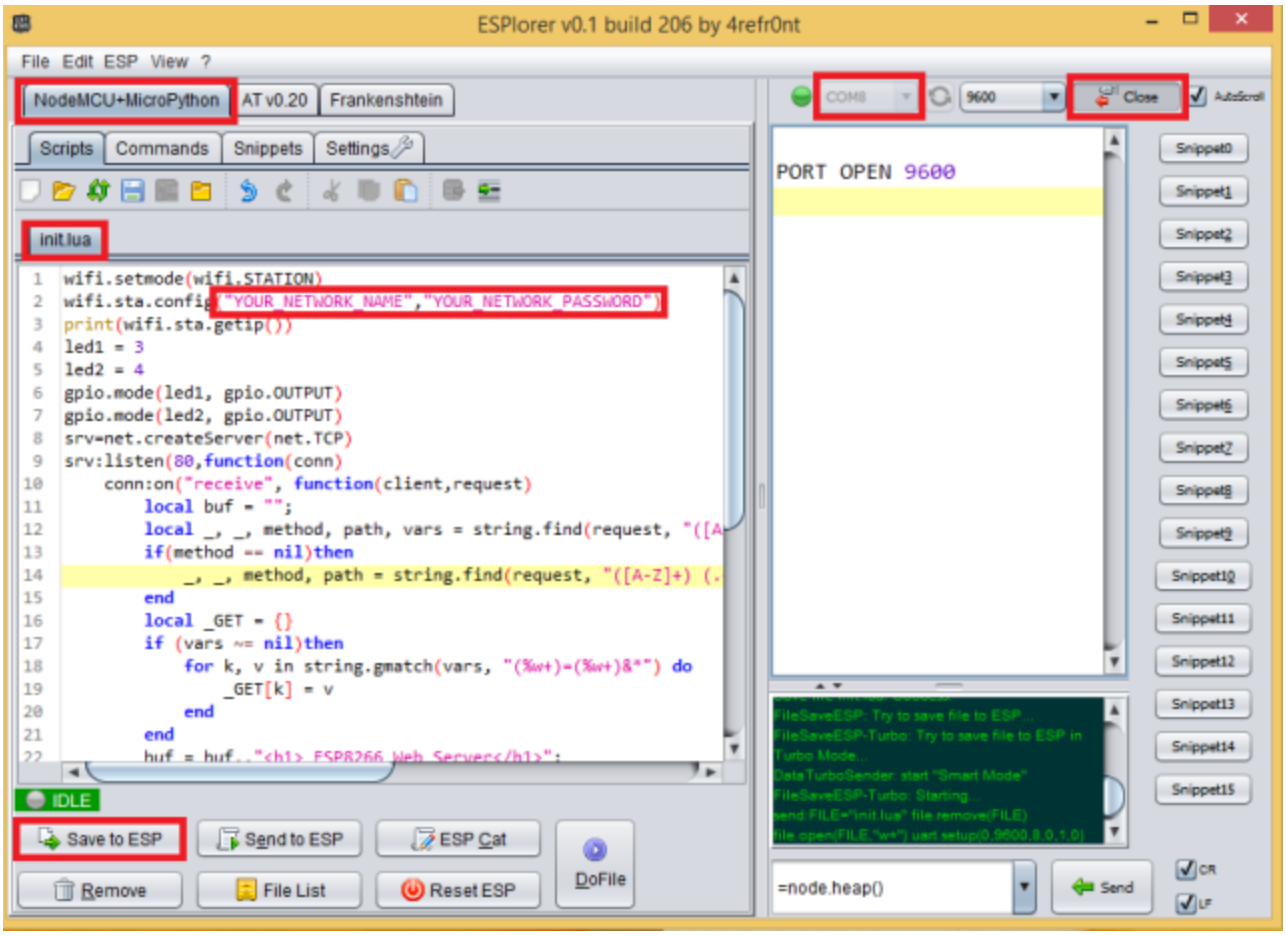
Uploading init.lua

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+MicroPtyhon 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 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

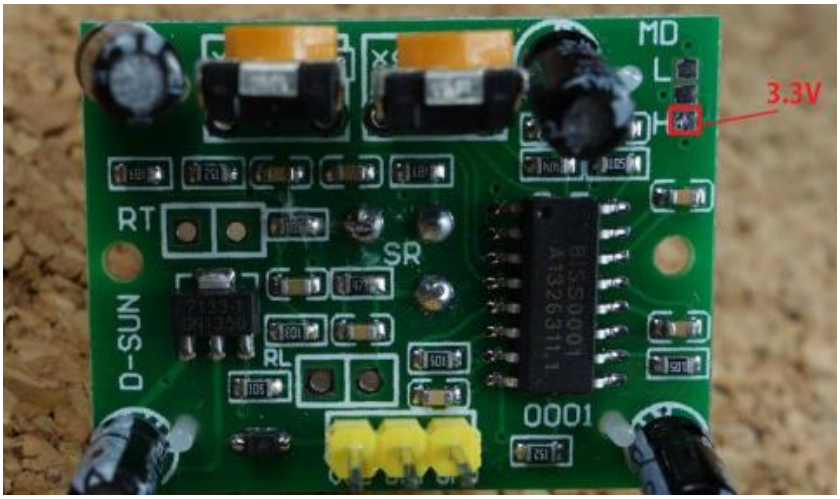
For this project, you're going to use a PIR Motion sensor. This type of sensor is used to detect movement from humans or pets. You can read a guide on my website on how to use this sensor with an Arduino.

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.

Go to this link: http://randomnerdtutorials.com/ebay-pir to purchase one of these modules on eBay for less than $2.

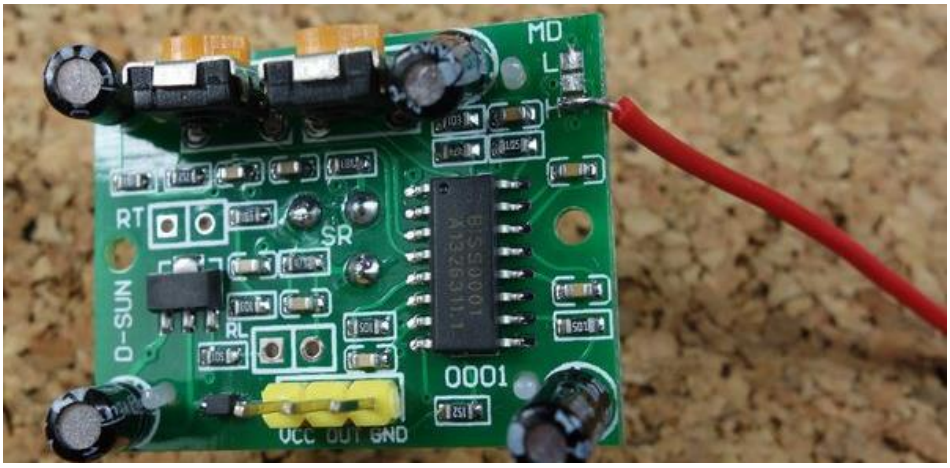
Before soldering

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.

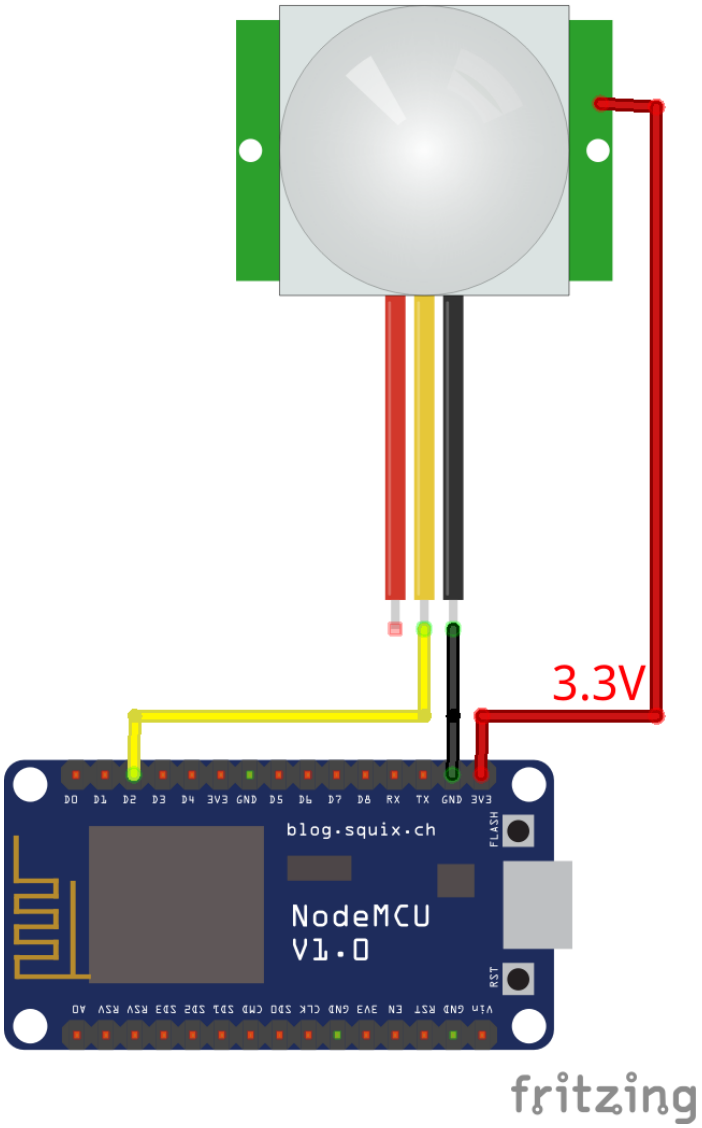


Testing

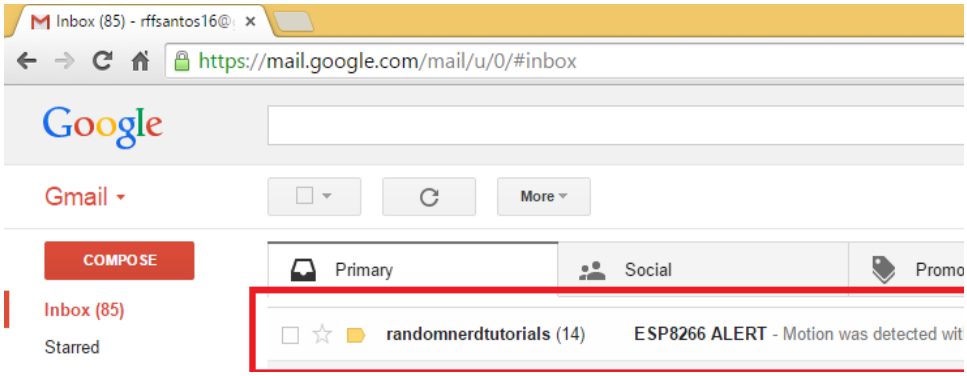
You can test this sensor with an Arduino using this sketch example.

Final Circuit

This circuit is very quick to assemble, simply follow these schematics:



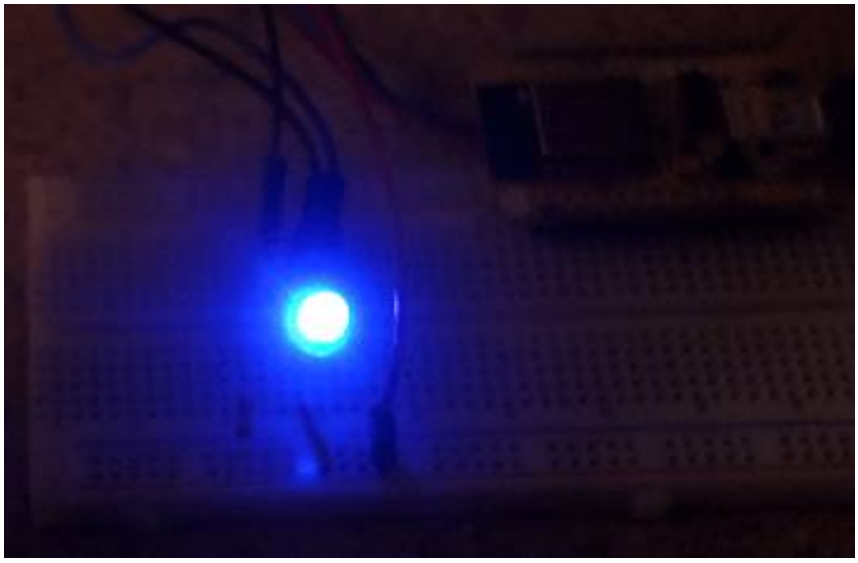
When you move your hand in front of the PIR Motion sensor, or when someone passes in front of your sensor, an email is immediately sent to your inbox! How cool is that?



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/channels.

Unit 8
ESP8266 RGB Color Picker



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:

- 1x ESP8266-12E
- 1x RGB LED Common Anode
- 3x 470Ω Resistors
- 1x Breadboard

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://gist.github.com/RuiSantosdotme/6ac55d00918c6ae043ba3c49471a0d5a

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\n\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\
">";
```

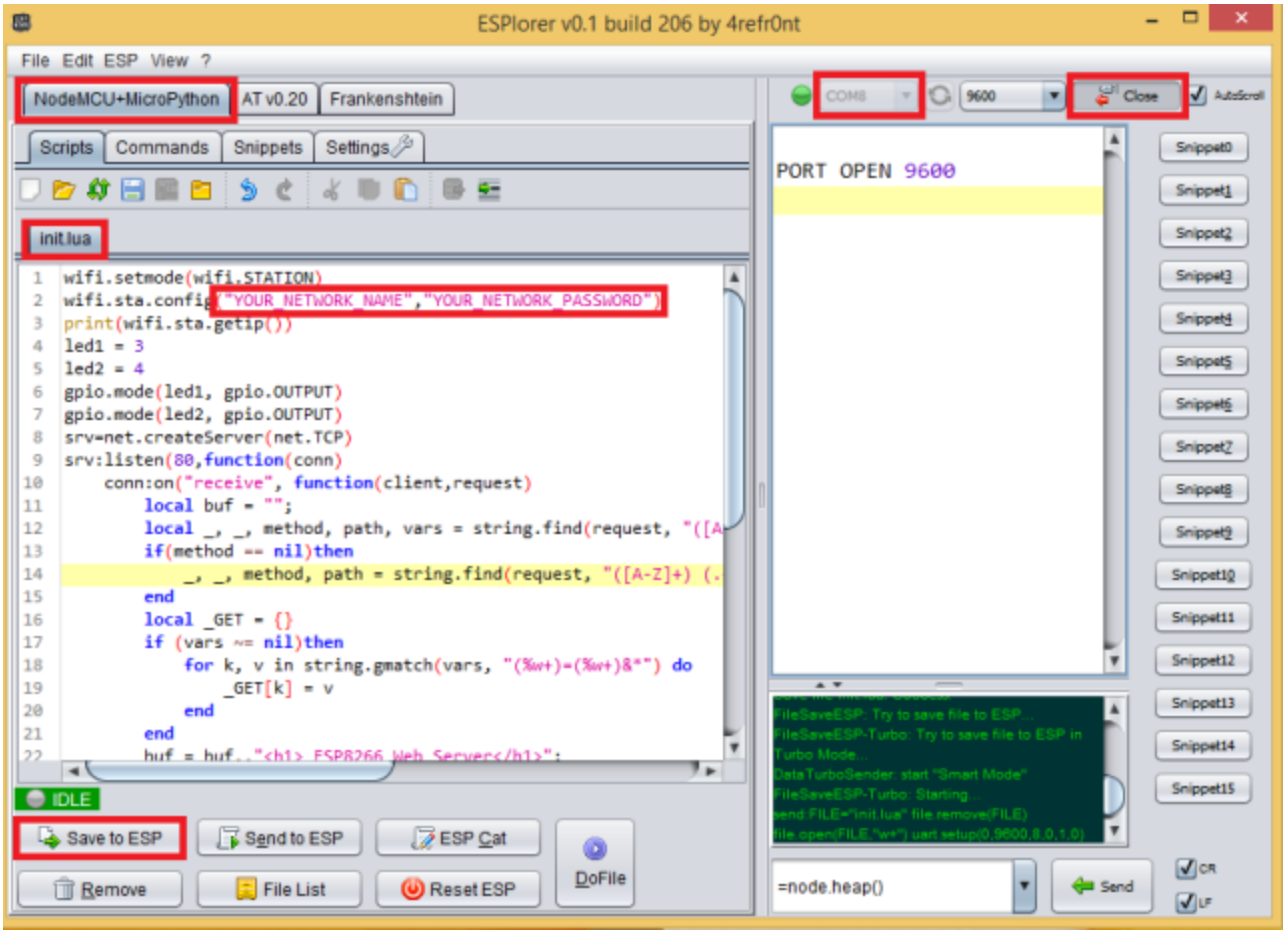
```
        buf = buf.."<script
src=\"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js\"><
/script>";
        buf = buf.."<script
src=\"https://cdnjs.cloudflare.com/ajax/libs/jscolor/2.0.4/jscolor.min.js\"><
/script>";
        buf = buf.."</head><body><div class=\"container\"><div
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
            -- 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:

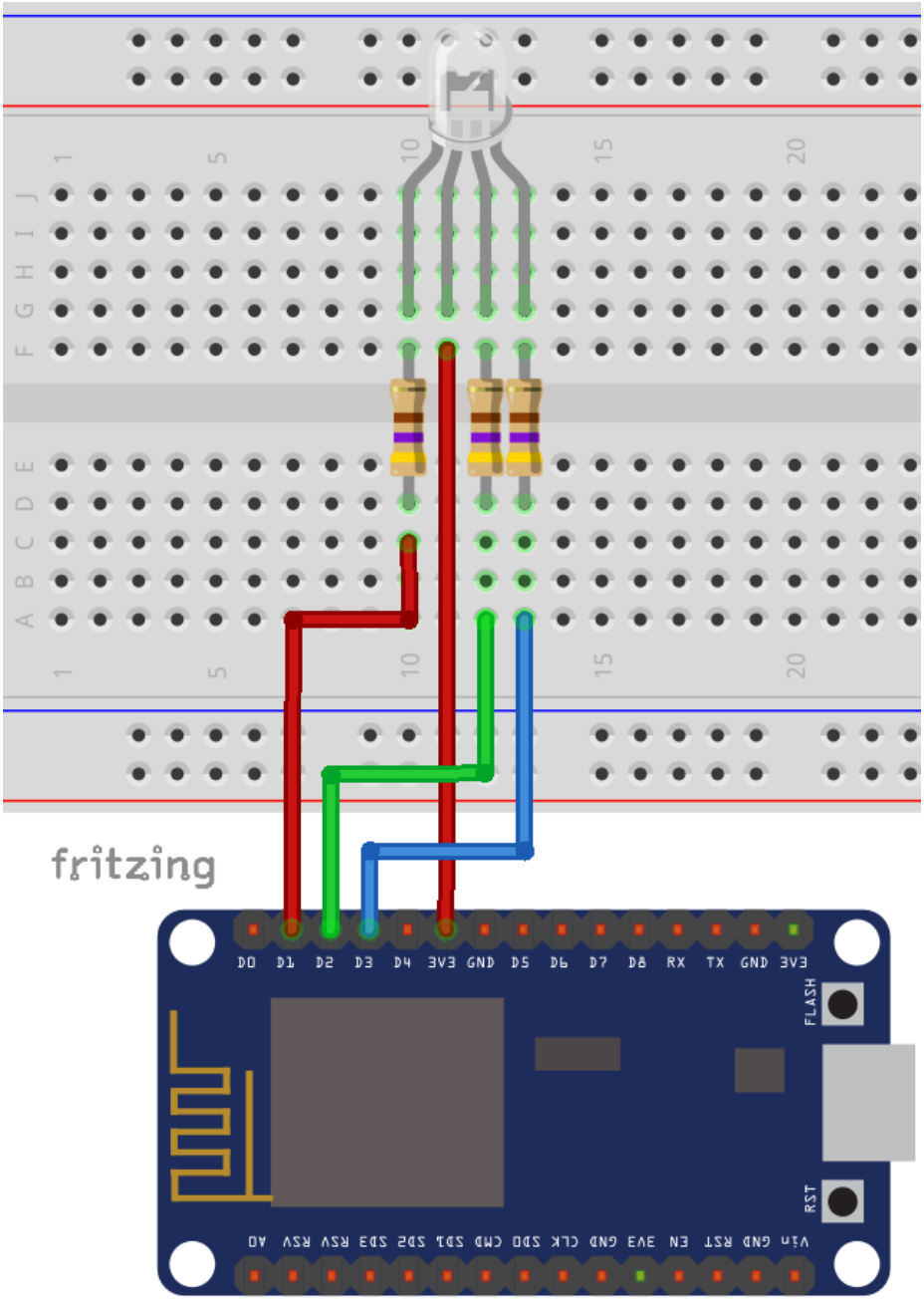
1. Connect your ESP8266-12E that has built-in programmer to your computer
2. Select your ESP8266-12E port
3. Press Open/Close
4. Select NodeMCU+MicroPtyhon tab
5. Create a new file called init.lua
6. Press Save to ESP

Everything that you need to worry about or change is highlighted in red box.



Final Circuit

Now, follow these schematics 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.

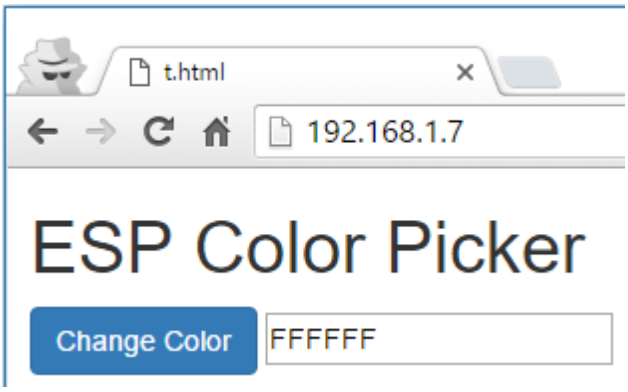
ESP8266 IP Address

When your ESP8266 restarts, it prints the ESP IP address in the serial monitor . Save that IP address, because you'll need it later.

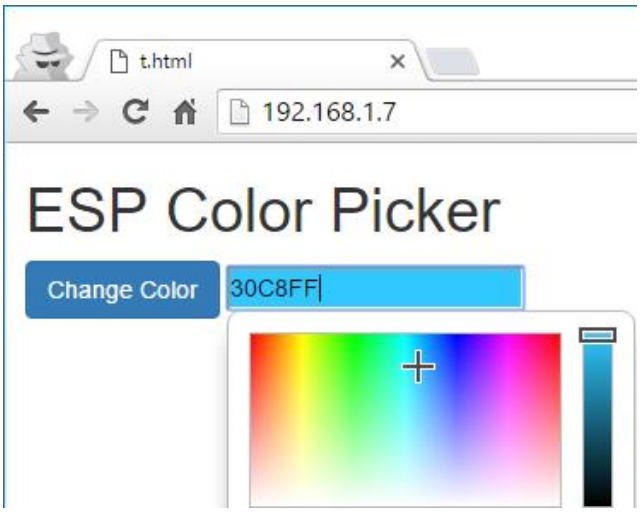
In my case, the ESP IP address is 192.168.1.7. You're all set!

Opening Your Web Server

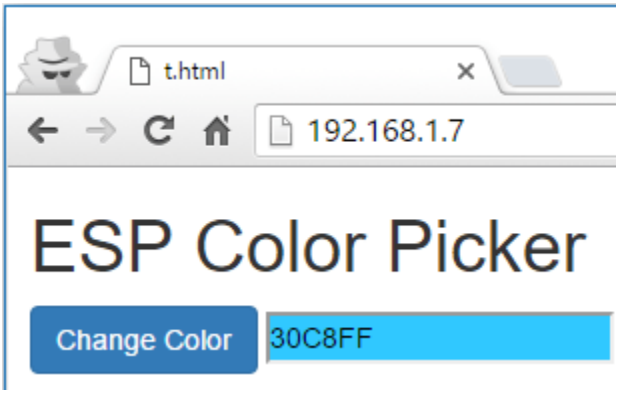
Go to any browser and enter the IP address of your ESP8266. This what you should see:



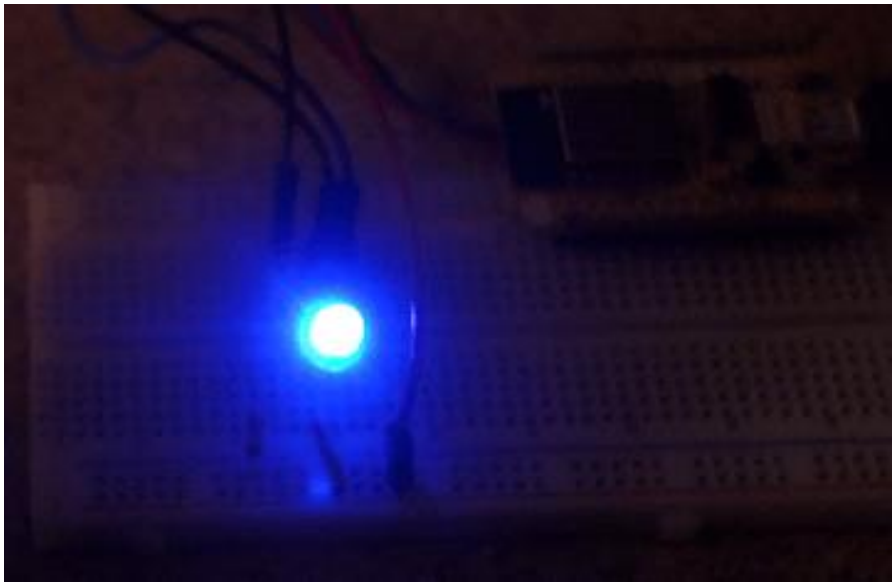
Click the field and a small window opens with a color picker. Simply drag your mouse or finger and select the color for your RGB LED:



Then simply click the “Change Color” button:



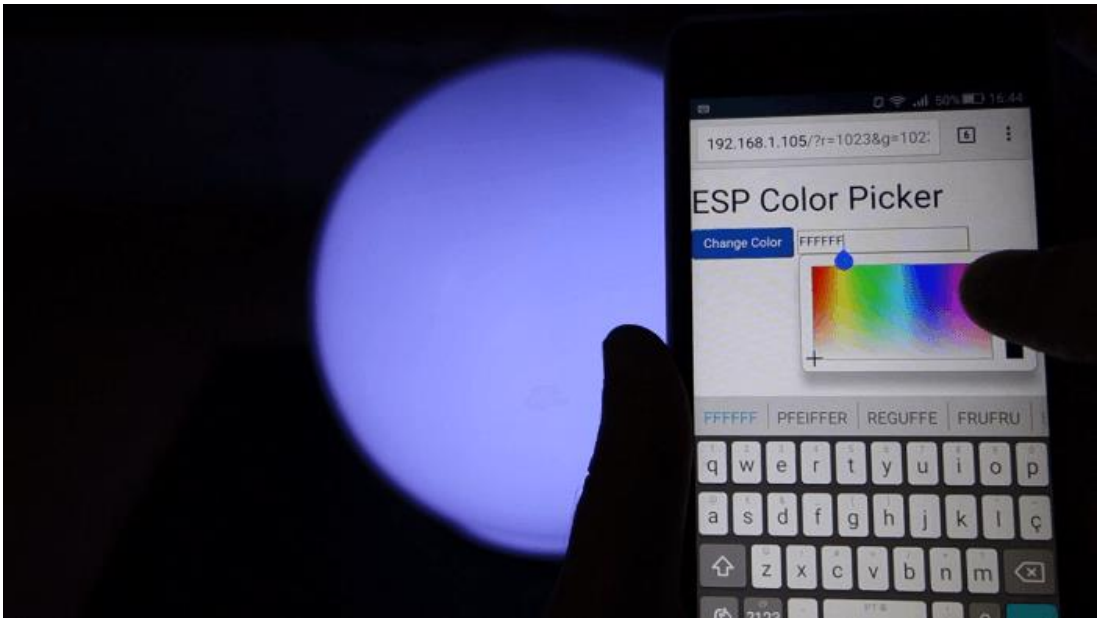
Now your RGB LED changes to the blue color:



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
$10 DIY WiFi RGB LED Mood Light



Unit 9 - $10 DIY WiFi RGB LED Mood Light

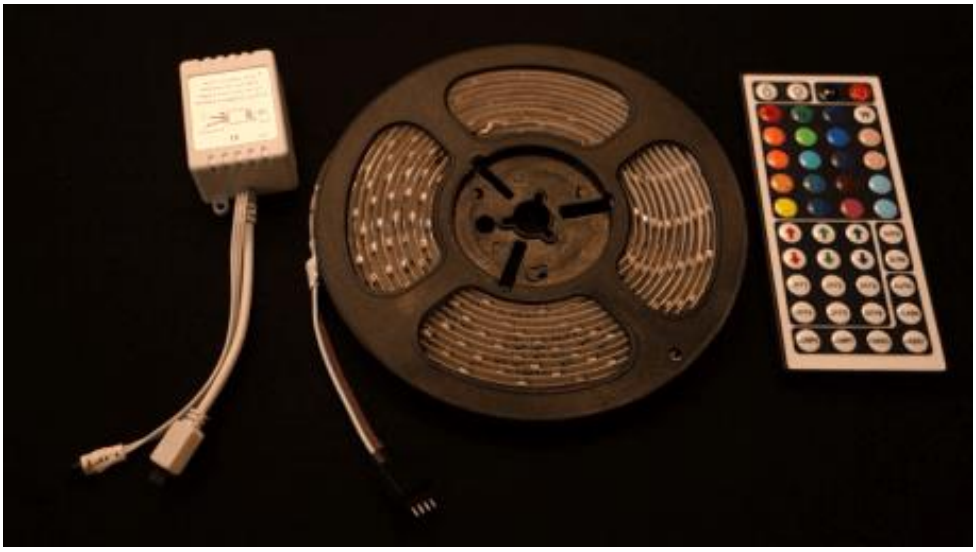
In this project, I'll show you how you can build a mood light. You'll use an ESP8266 to remotely control the color of your light using your smartphone or any other device that has a browser. This project is called $10 DIY WiFi RGB LED Mood Light.

To learn more about RGB LEDs use the following tutorial as a reference: How do RGB LEDs work?

Parts Required

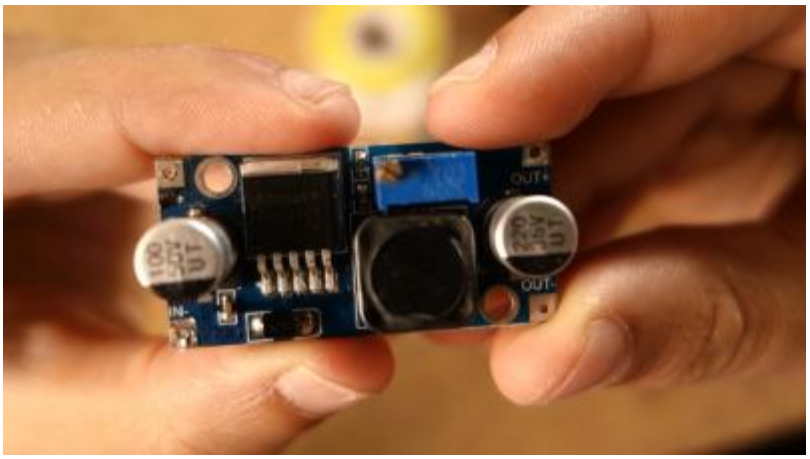
Here's the hardware that you need to complete this project:

- 1x ESP8266-12E (see on eBay)
- 1x RGB LED Strip (see on eBay)

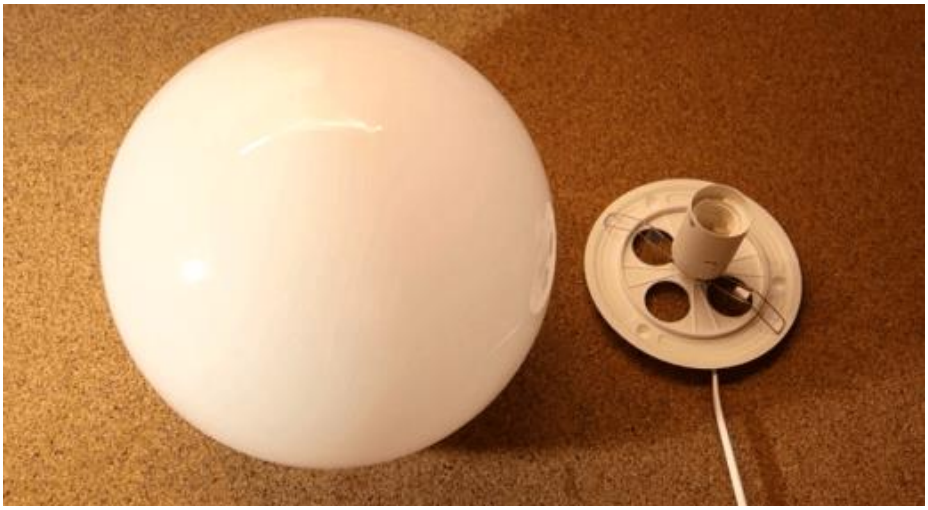


- 1x 12V Power Supply (see on eBay)

- Device to reduce voltage from 12V to 5V
    - Alternative – LM7805 with heat sink
    - Recommended – Step down buck converter module (see on eBay)



- 3x NPN Transistors 2N2222 or equivalent (see on eBay)
- 3x 1k Ohm Resistors (see on eBay)
- 1x Breadboard (see on eBay)
- Wire Cables (see on eBay)
- 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://gist.github.com/RuiSantosdotme/295328687bdf9f492847b3d260cf8d08

Don’t forget to add your network name (SSID) and password to the script.

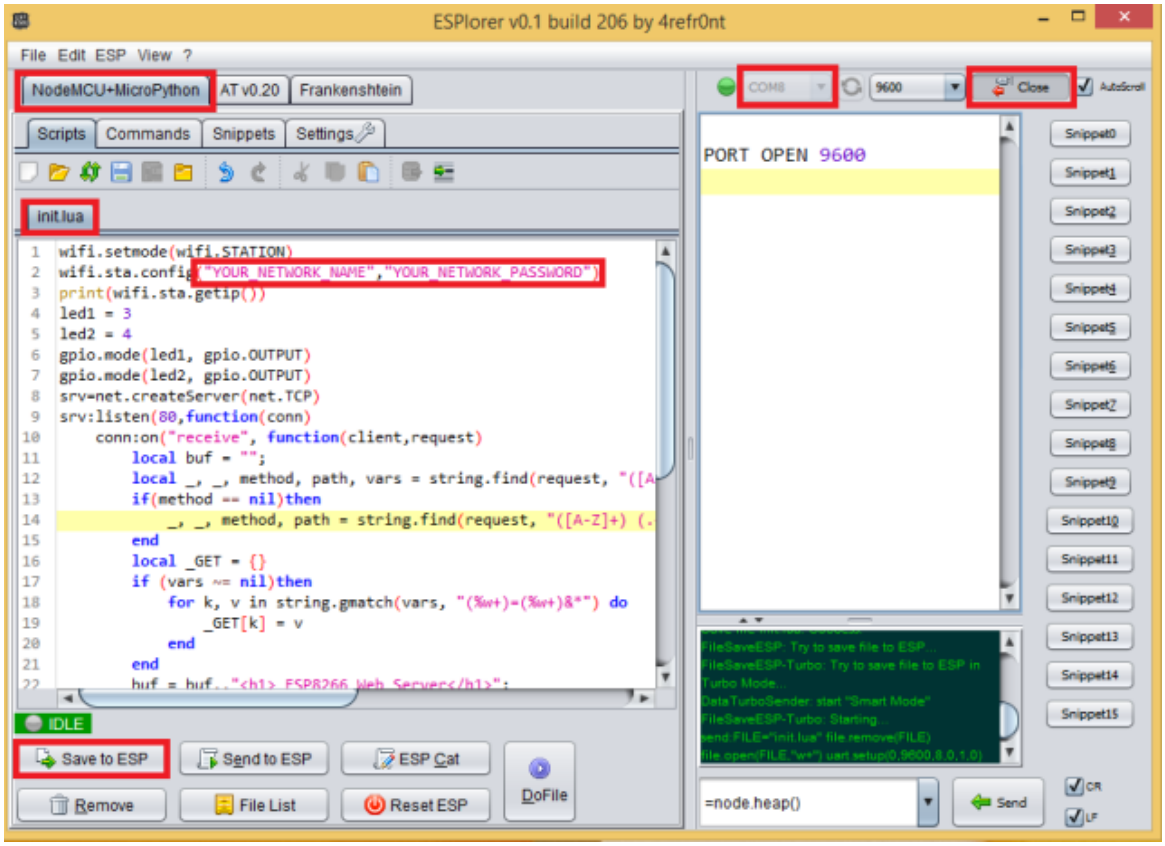
```
wifi.setmode(wifi.STATION)
wifi.sta.config("REPLACE_WITH_YOUR_SSID","REPLACE_WITH_YOUR_PASSWORD")

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\n\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
        client:send(buf);
        client:close();
        collectgarbage();
    end)
end)
```

Uploading init.lua

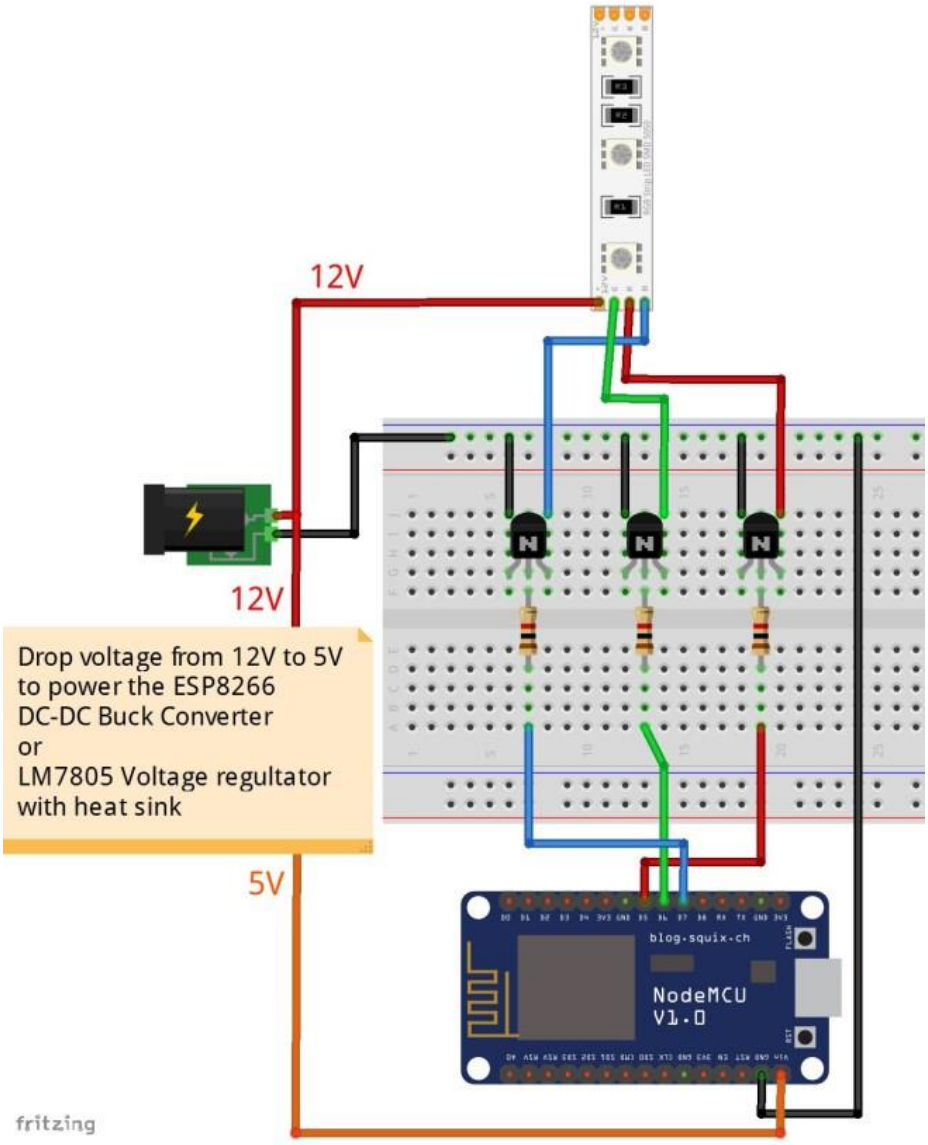
Follow these instructions to upload a Lua script:

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+MicroPtyhon tab
5. Create a new file called init.lua
6. Press Save to ESP



Final Circuit

Now follow these schematics to create the final circuit.



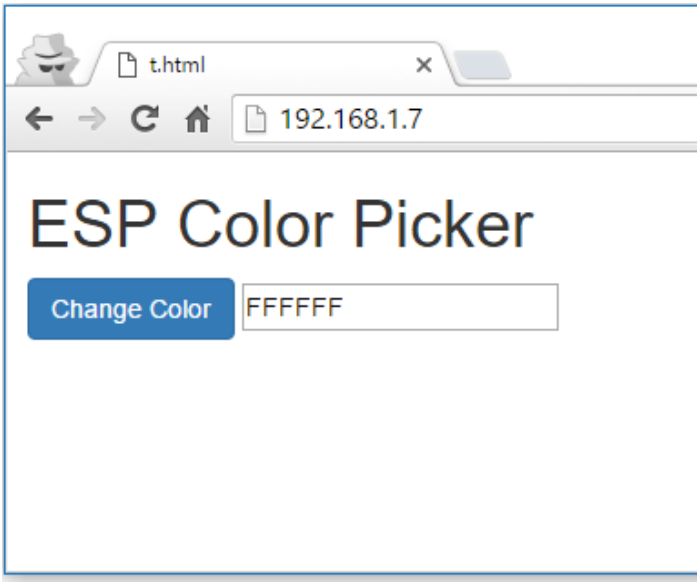
ESP8266 IP Address

When your ESP8266 restarts, it prints ESP IP address in the serial monitor the. Save that IP address, because you'll need it later.

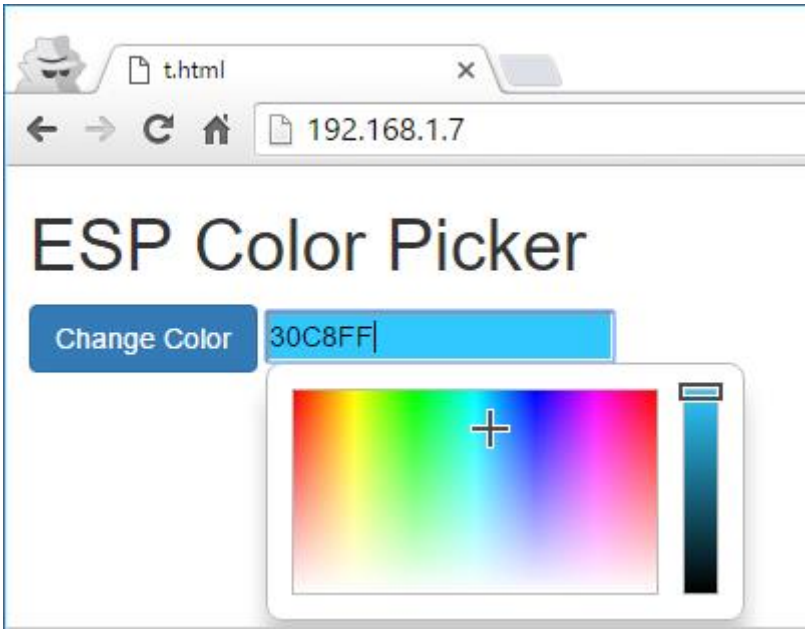
In my case, the ESP IP address is 192.168.1.105. You're all set!

Opening Your Web Server

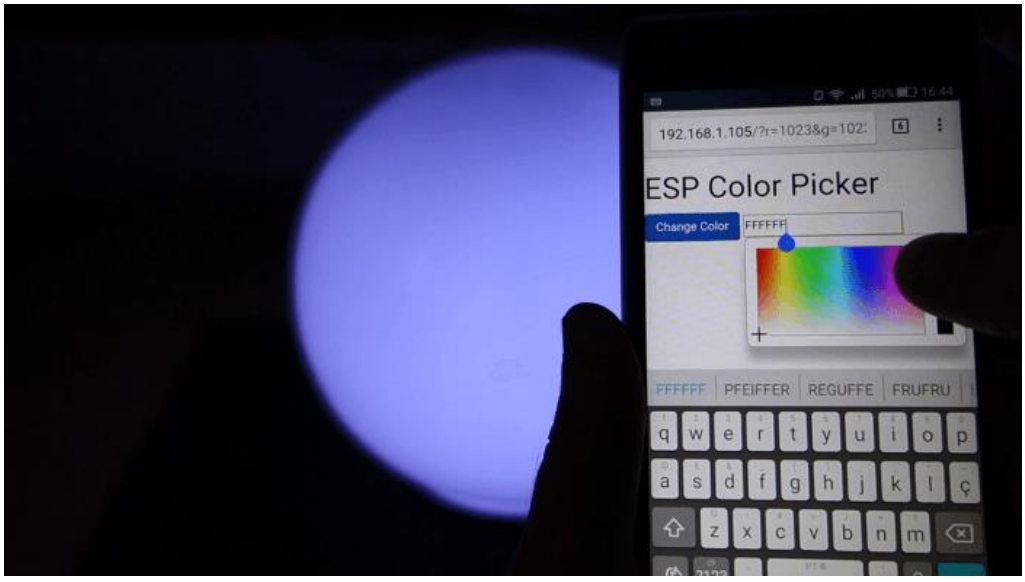
Go to any browser and enter the IP address of your ESP8266. This is what you should see:



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 sensor data
- Build an email alert system

Now, feel free to add multiple ESP8266s 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

P.S. If you haven't already, you can download a FREE eBook with all my Arduino Projects by visiting -> http://randomnerdtutorials.com/ebook

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 nearly 150 free blog posts with complete tutorials using open-source hardware that anyone can read, remix and apply to their own projects: http://randomnerdtutorials.com

To keep free tutorials coming, there's also paid content or as I like to call "premium content".

To support Random Nerd Tutorials you can download premium content here. If you enjoyed this eBook make sure you check all the others.

Thanks for taking the time to read my work!

Good luck with all your projects,

-Rui Santos

P.S. Click here for more courses and eBooks like this one.

Click here to Download other Courses and eBooks

http://randomnerdtutorials.com/products