Contents

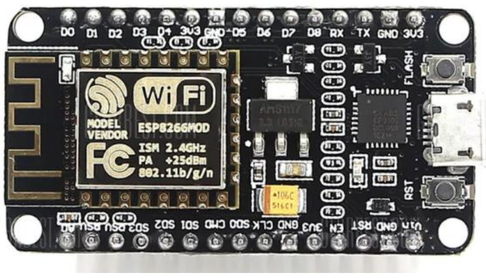
First steps with: ESP8266 & Arduino IDE on Windows 2
Wifi connecting and android application Example 1.0 8
Code without the external screen 12
Analyzing the code 15
Wifi connecting with search engine only Example 1.1 20
Example 2.0 controlling two leds from a browser 28

Tutorial by: Christianidis Vasileios

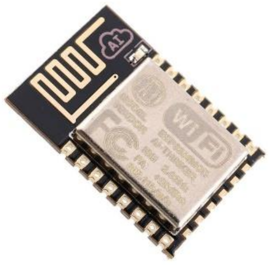
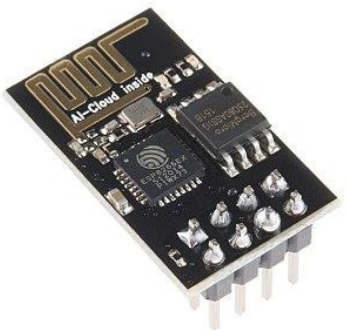
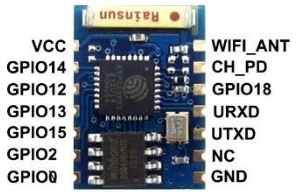
You can contact me by email: basilisvirus@hotmail.com

First steps with: ESP8266 & Arduino IDE on Windows

From now on, in this tutorial, I will refer to **ESP8266** as **ESP**.
I will refer to **NodeMCU ESP8266 Development Board** as **Full dev board**
I will refer to **Microcontrollers** as **Mcu** or **µcu**
I will refer to **General Purpose Input Output** as GPIO
Note that I am not using the ESP8266 on its own, because these are the first steps, it is good to buy the full development board that NodeMCU - NodeMCU is the name of the company that made the dev board-. So that I wont burn my ESP by mistake, cause some pins use 3.3volt and not 5v. (I think even if you need to program it, the tx and rx pins use 3.3v instead of 5v, but I wont cover this information on this tutorial)
So the name to google it, is: (NodeMCU ESP8266 Development Board) and this is what you want to buy for a start, as a newbie. So, the difference of the ESP and the Full dev board is, first of all, how it looks.
The Full dev board looks like this/or like this (3 images). the pcb can also be blue-colored instead of black:

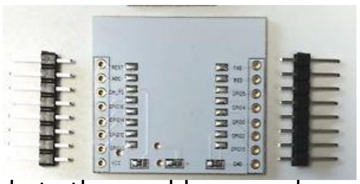


and the ESP looks like this:

As you see, the ESP as well as the full dev board, comes in some modifications and different versions. now, you may also notice that on the full dev board, there actually is the ESP itself embed/soldered on the full dev board.
Now, if you buy the full dev board as I recommend you, you will spend some more money (I think its +4 euros), but the advantages will be:

1) If you use the full dev board, you wont interacting directly to the ESP, thus avoiding any kind of damage to the board because of wrong power supply or if you try to solder it you may burn it etc.
2) the full dev board is breadboard friendly, while the ESP as you see doesn't have any pins to connect it to the breadboard, you will have to solder it to a breakout board for the ESP. this is how a breakout board for the esp looks like:

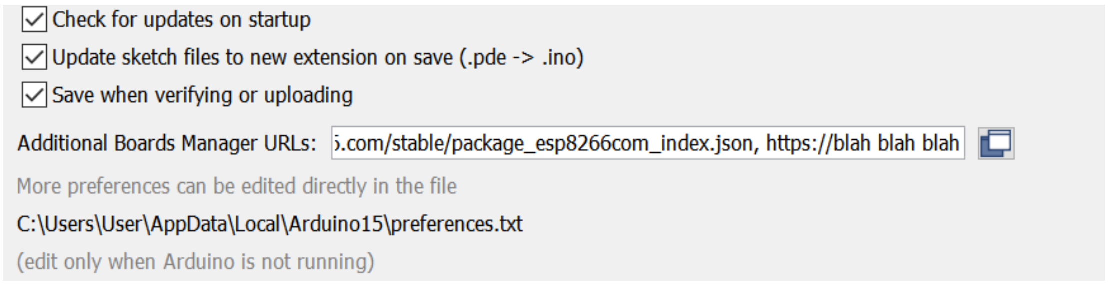


3) I don't know what other problems can be caused if a beginner uses the ESP, but you get the idea.

So, my version of Full dev board looks like this:

Ok a few things to know about it (either you buy the ESP, or the full dev board): it has its own µcu, meaning that it is actually a microcontroller like the atmega of the Arduino, it can run software on its own and it doesn't need lets say another mcu to work properly.
If you are asking yourself: "then why do I need the Arduino/atmega or any other mcu and I just don't use the ESP (of the full dev board) only?"
The answer is: it doesn't have much memory to run bigger software and it doesn't have many GPIOs , it also has only one Analog read pin.
Also, you don't need to use any usb drivers to upload the program on the full dev board, but I don't know if it goes the same with ESP.

So, first of, we will make the build-in led to light up. so,
1) Just connect it to your computer using a usb cable.
2) (kind of optional) Use the device manager to find out in which com port it is connected. (com1, com7 etc)
3) download and install Arduino IDE.
4) Open Arduino ide, go to File/preferences/ and paste this link:
https://arduino.esp8266.com/stable/package_esp8266com_index.json where it says "Additional Boards Manager urls. if there is something already pasted there, separate it with a comma and a space like this:

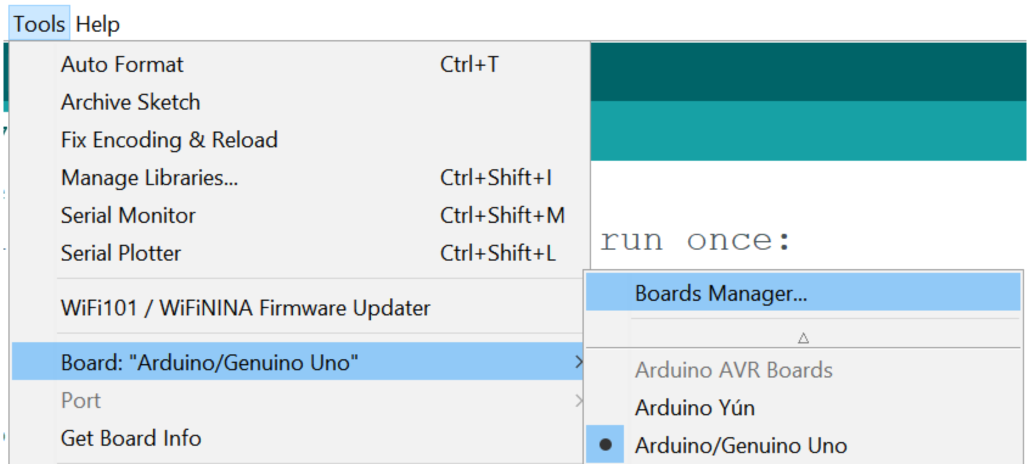


☑ Check for updates on startup
☑ Update sketch files to new extension on save (.pde -> .ino)
☑ Save when verifying or uploading

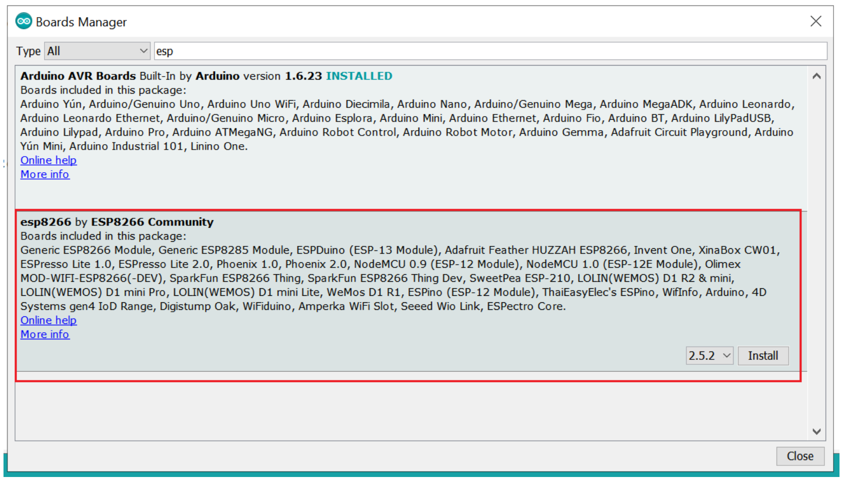
Additional Boards Manager URLs: 5.com/stable/package_esp8266com_index.json, https://blah blah blah

More preferences can be edited directly in the file
C:\Users\User\AppData\Local\Arduino15\preferences.txt
(edit only when Arduino is not running)

Then click ok,
5) Go here :



Select Esp… by esp community, click install and restart Arduino ide.


Boards Manager

Type All esp

Arduino AVR Boards Built-In by **Arduino** version **1.6.23** **INSTALLED**
Boards included in this package:
Arduino Yún, Arduino/Genuino Uno, Arduino Uno WiFi, Arduino Diecimila, Arduino Nano, Arduino/Genuino Mega, Arduino MegaADK, Arduino Leonardo, Arduino Leonardo Ethernet, Arduino/Genuino Micro, Arduino Esplora, Arduino Mini, Arduino Ethernet, Arduino Fio, Arduino BT, Arduino LilyPadUSB, Arduino Lilypad, Arduino Pro, Arduino ATMegaNG, Arduino Robot Control, Arduino Robot Motor, Arduino Gemma, Adafruit Circuit Playground, Arduino Yún Mini, Arduino Industrial 101, Linino One.
Online help
More info

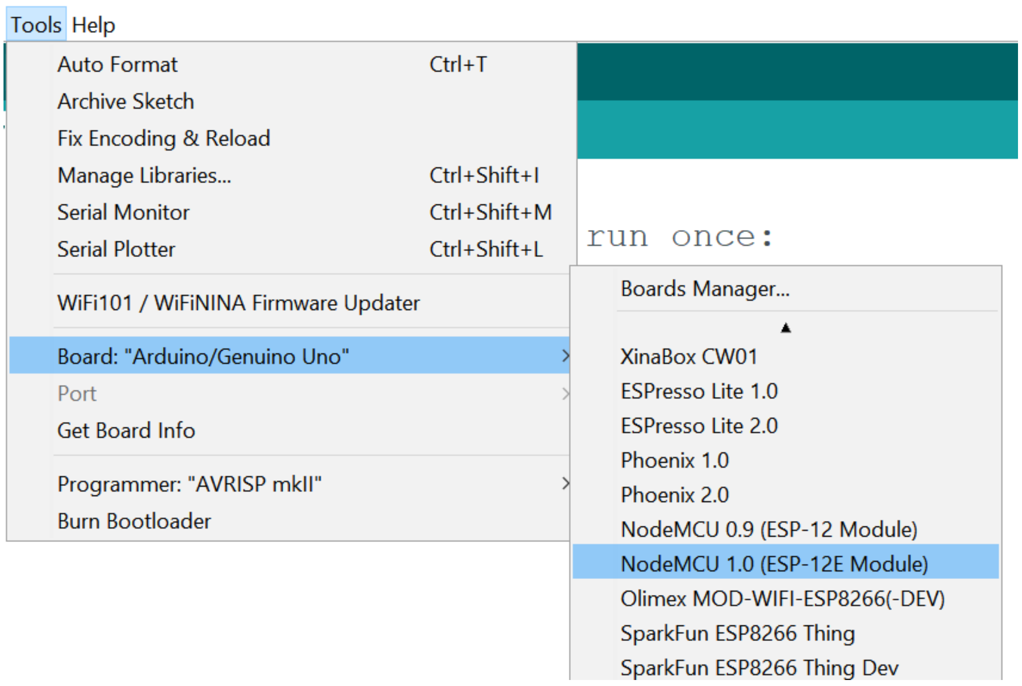
esp8266 by **ESP8266 Community**
Boards included in this package:
Generic ESP8266 Module, Generic ESP8285 Module, ESPDuino (ESP-13 Module), Adafruit Feather HUZZAH ESP8266, Invent One, XinaBox CW01, ESPresso Lite 1.0, ESPresso Lite 2.0, Phoenix 1.0, Phoenix 2.0, NodeMCU 0.9 (ESP-12 Module), NodeMCU 1.0 (ESP-12E Module), Olimex MOD-WIFI-ESP8266(-DEV), SparkFun ESP8266 Thing, SparkFun ESP8266 Thing Dev, SweetPea ESP-210, LOLIN(WEMOS) D1 R2 & mini, LOLIN(WEMOS) D1 mini Pro, LOLIN(WEMOS) D1 mini Lite, WeMos D1 R1, ESPino (ESP-12 Module), ThaiEasyElec's ESPino, WifInfo, Arduino, 4D Systems gen4 IoD Range, Digistump Oak, WiFiduino, Amperka WiFi Slot, Seeed Wio Link, ESPectro Core.
Online help
More info

2.5.2 Install

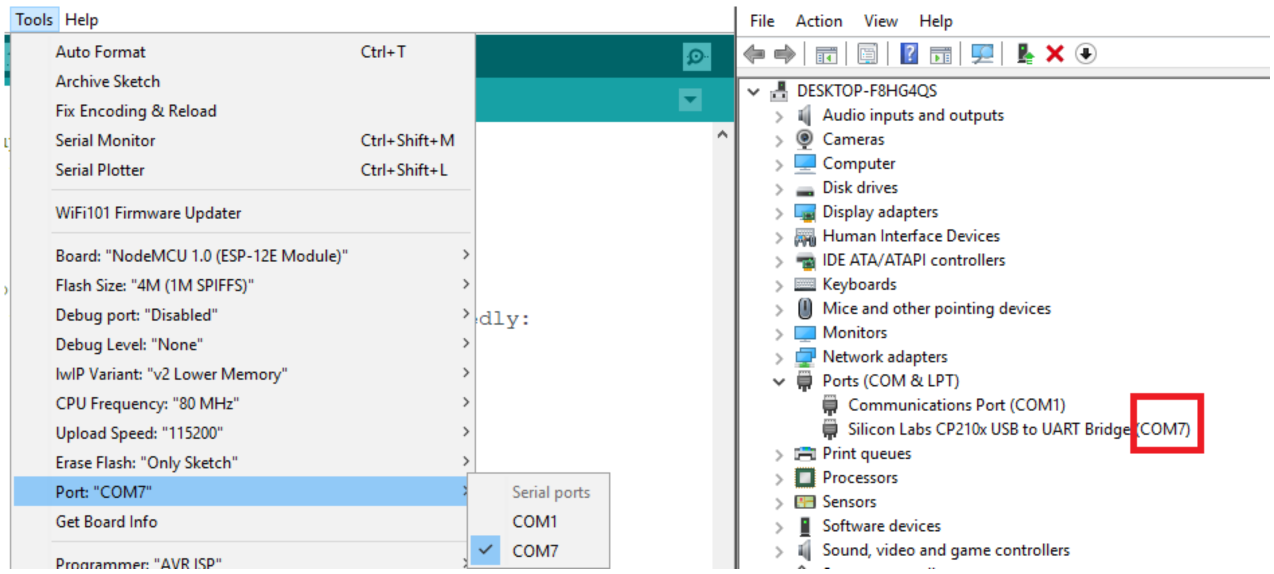
Close

If this (installing the community library) doesn't work, you can find more help here: https://github.com/esp8266/Arduino or you can google it some more for more info.

6) Select NodeMCU 1.0 (ESP-12E Module):

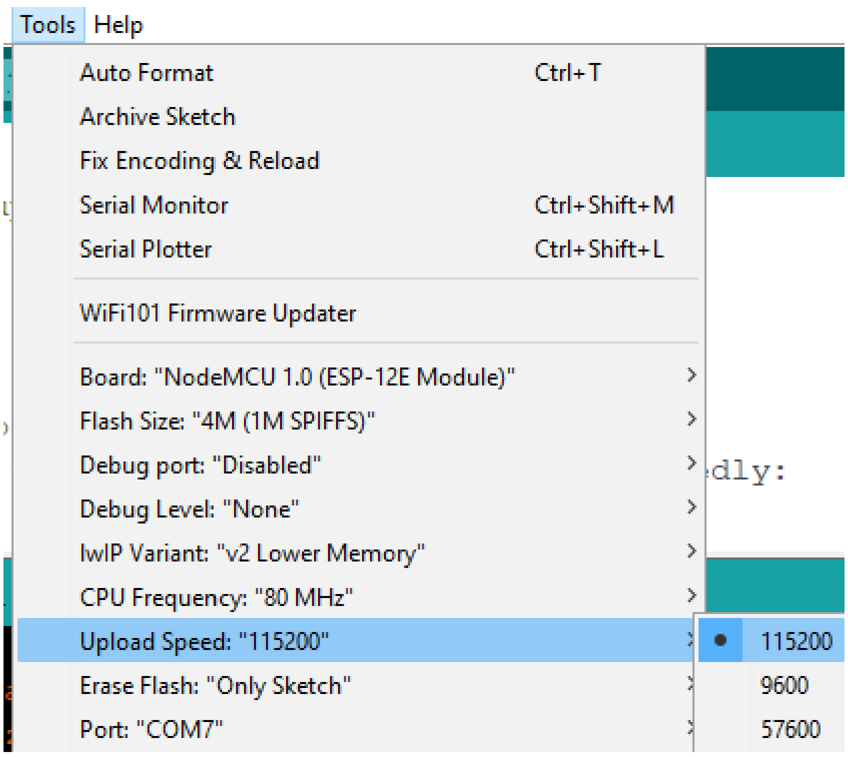

Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino/Genuino Uno"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
run once:
Boards Manager...
XinaBox CW01
ESPresso Lite 1.0
ESPresso Lite 2.0
Phoenix 1.0
Phoenix 2.0
NodeMCU 0.9 (ESP-12 Module)
NodeMCU 1.0 (ESP-12E Module)
Olimex MOD-WIFI-ESP8266(-DEV)
SparkFun ESP8266 Thing
SparkFun ESP8266 Thing Dev

along with the com port you connected the ESP. on this image I have the device manager side to side just to see how it looks like.


Board: "NodeMCU 1.0 (ESP-12E Module)"
Flash Size: "4M (1M SPIFFS)"
Debug port: "Disabled"
Debug Level: "None"
IwIP Variant: "v2 Lower Memory"
CPU Frequency: "80 MHz"
Upload Speed: "115200"
Erase Flash: "Only Sketch"
Port: "COM7"
Programmer: "AVR ISP"

Try to upload a empty schech (with only void setup and void loop on it)
void setup() {}
void loop() {}

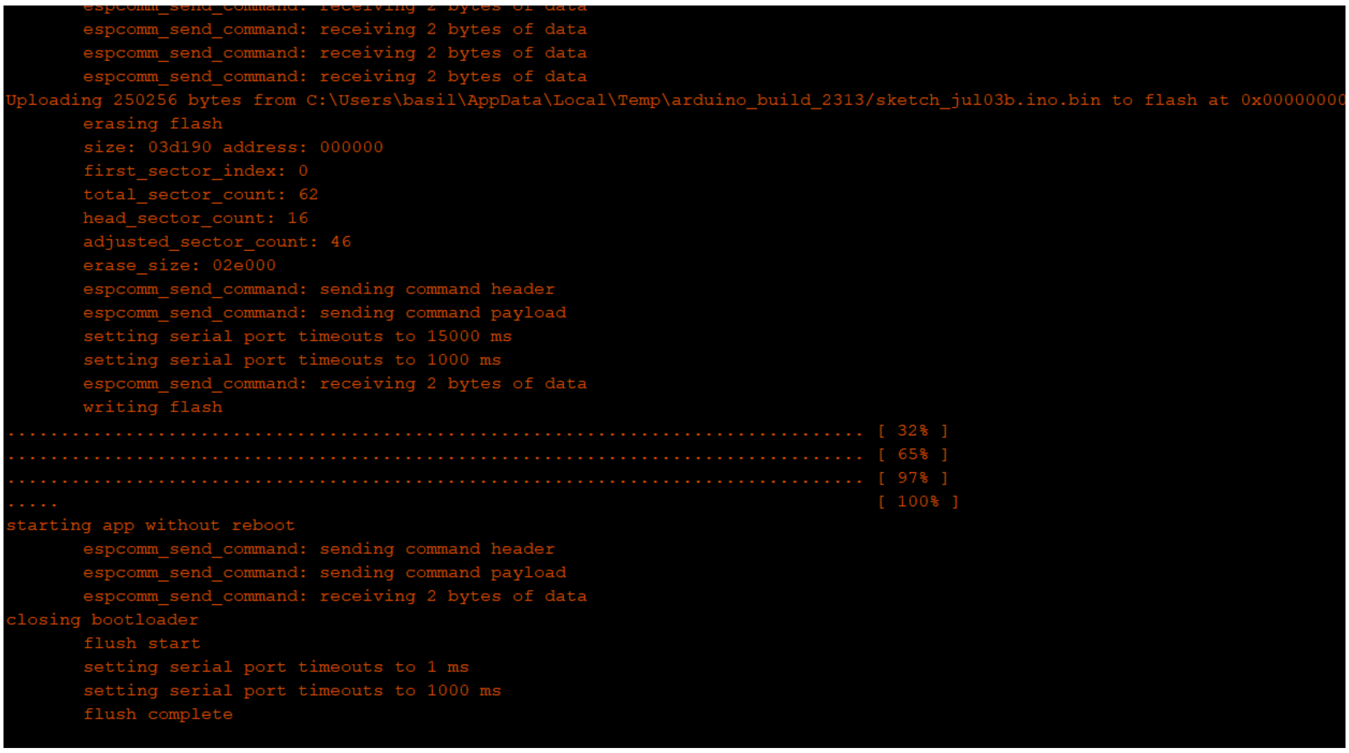
If something goes wrong, try to lower the upload speed



```
Tools  Help
Auto Format                                   Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor                                Ctrl+Shift+M
Serial Plotter                                Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "NodeMCU 1.0 (ESP-12E Module)"              >
Flash Size: "4M (1M SPIFFS)"                       >
Debug port: "Disabled"                             >
Debug Level: "None"                                >
lwIP Variant: "v2 Lower Memory"                    >
CPU Frequency: "80 MHz"                            >
Upload Speed: "115200"                             >   ● 115200
Erase Flash: "Only Sketch"                         >     9600
Port: "COM7"                                       >     57600
```

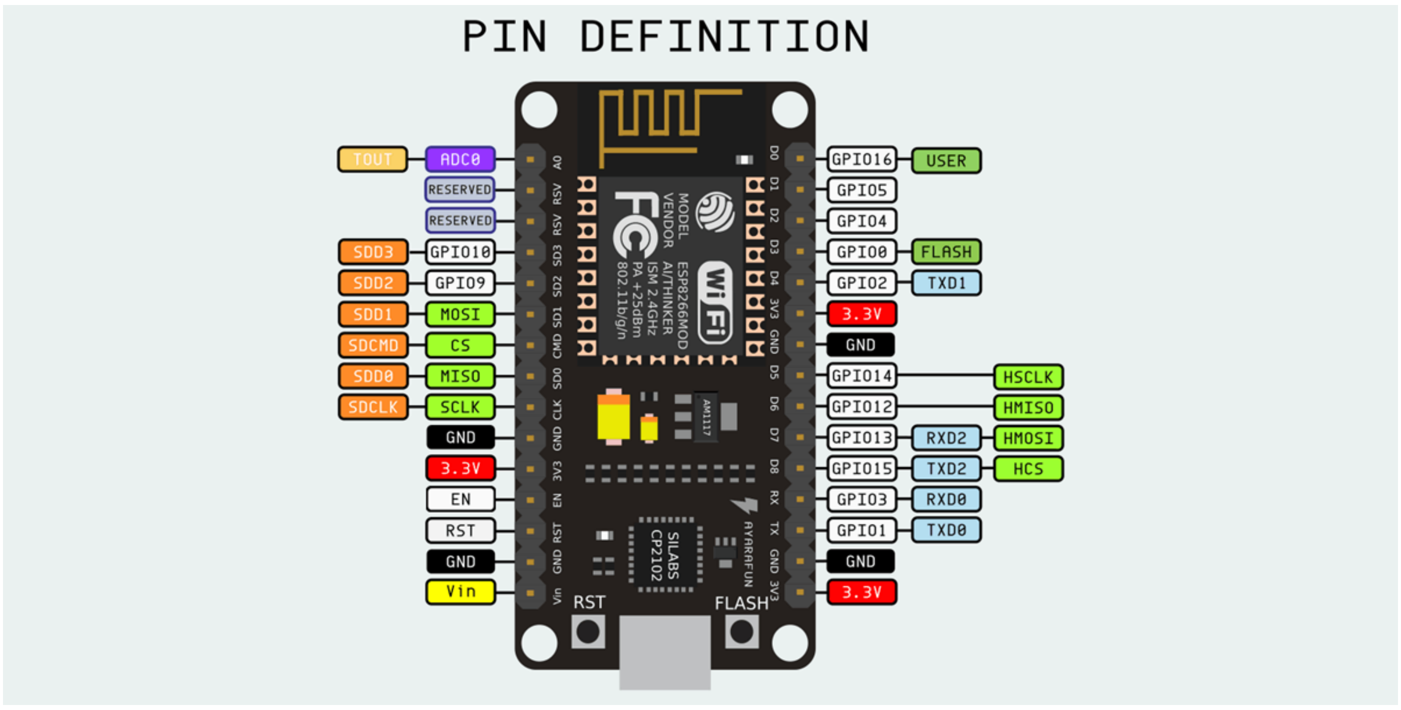
This is what I use, higher upload speeds may let the board become unstable.

That's how a successful upload should looks like:

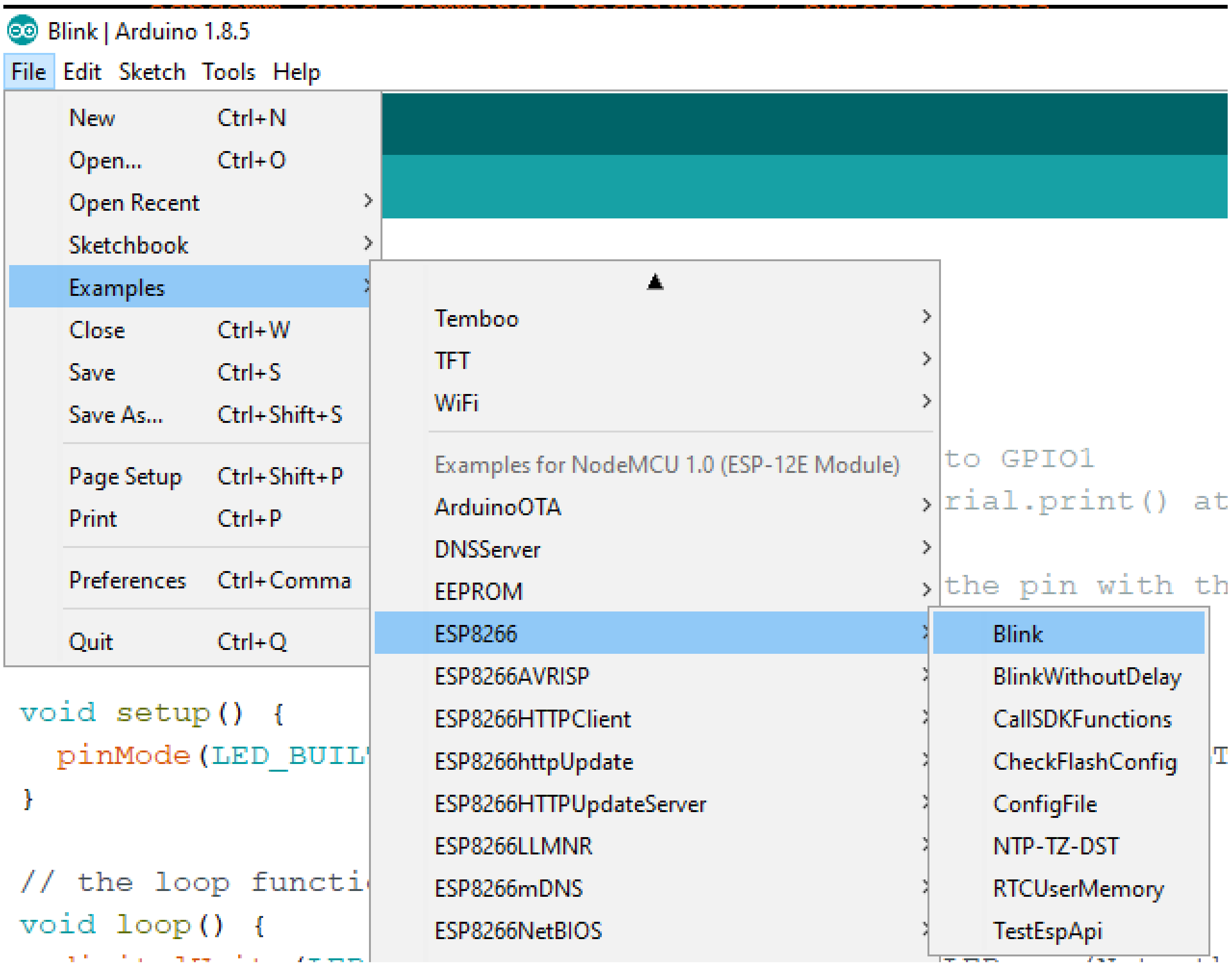


```
	espcomm_send_command: receiving 2 bytes of data
	espcomm_send_command: receiving 2 bytes of data
	espcomm_send_command: receiving 2 bytes of data
	espcomm_send_command: receiving 2 bytes of data
Uploading 250256 bytes from C:\Users\basil\AppData\Local\Temp\arduino_build_2313/sketch_jul03b.ino.bin to flash at 0x00000000
	erasing flash
	size: 03d190 address: 000000
	first_sector_index: 0
	total_sector_count: 62
	head_sector_count: 16
	adjusted_sector_count: 46
	erase_size: 02e000
	espcomm_send_command: sending command header
	espcomm_send_command: sending command payload
	setting serial port timeouts to 15000 ms
	setting serial port timeouts to 1000 ms
	espcomm_send_command: receiving 2 bytes of data
	writing flash
................................................................................ [ 32% ]
................................................................................ [ 65% ]
................................................................................ [ 97% ]
.....                                                                            [ 100% ]
starting app without reboot
	espcomm_send_command: sending command header
	espcomm_send_command: sending command payload
	espcomm_send_command: receiving 2 bytes of data
closing bootloader
	flush start
	setting serial port timeouts to 1 ms
	setting serial port timeouts to 1000 ms
	flush complete
```

So, if the above worked correctly, lets get going.
Note that: build in led is on pin 16, but it may (or may not) change according to the full dev board you are using.



Follow these steps to get the ready-example code for blinking:

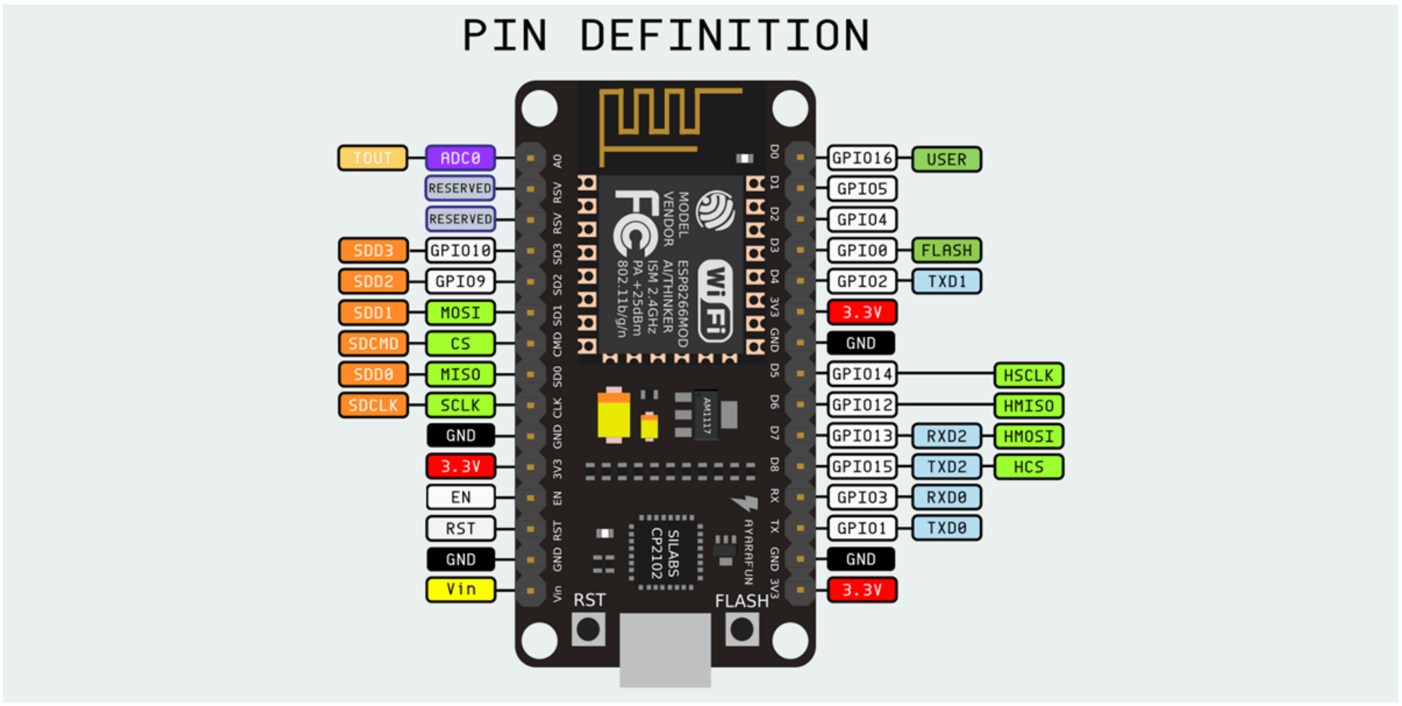


```
to GPIO1
rial.print() at

the pin with th

void setup() {
  pinMode(LED_BUIL
}

// the loop functi
void loop() {
```

As you can see, the build in led may change its pin depending on the board, just look it up a little bit, or put an external led (use one of your leds) to make it blink.

```
/*
ESP8266 Blink by Simon Peter
Blink the blue LED on the ESP-01 module
This example code is in the public domain

The blue LED on the ESP-01 module is connected to GPIO1
(which is also the TXD pin; so we cannot use Serial.print() at the same time)

Note that this sketch uses LED_BUILTIN to find the pin with the internal LED
*/
```

Remember: We don't need any extra libraries to control the GPIOs of the esp full dev board. Just note that you may not be able to control some pins, like all the TX, RX if you use the serial monitor at the same time, and some others like the FLASH pin and pins 9 and 10 are also for special uses.

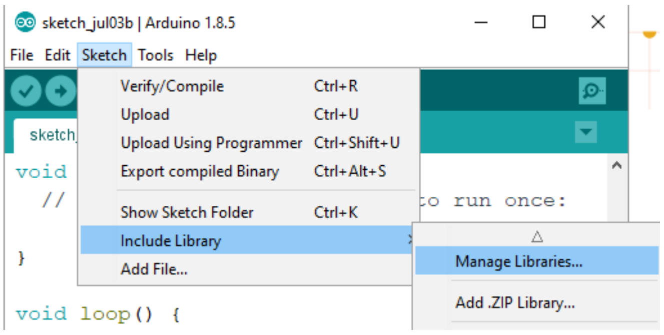


Okay we are almost there. you know how to control the GPIOs now, all you need to do is a bit of practicing and testing, but lets get to the wifi thing, right?

Wifi connecting and android application Example 1.0

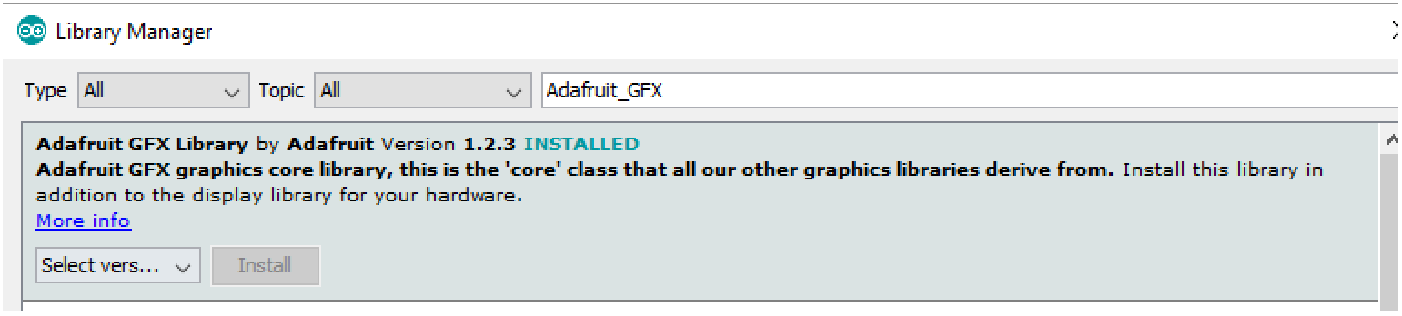
So I followed this tutorial:
https://www.youtube.com/watch?v=fVpAN3adK9A
which is on this website: http://androidcontrol.blogspot.com/2016/05/esp8266-wifi-control-relay.html
with the source code here: https://github.com/amphancm/ESP8266WiFiControl (file esp8266Server.ino)
and the android app: https://play.google.com/store/apps/details?id=esp8266.wifi.control

Ok **if you cannot follow the tutorial:**, I will show you what to do. first of, install these libraries: Adafruit_GFX.h (Version 1.0.2 only) and ESP_Adafruit_SSD1306.h like this:
The **Adafruit_GFX.h** can be found here:


sketch_jul03b | Arduino 1.8.5
File Edit Sketch Tools Help
Verify/Compile Ctrl+R
Upload Ctrl+U
Upload Using Programmer Ctrl+Shift+U
Export compiled Binary Ctrl+Alt+S
Show Sketch Folder Ctrl+K
Include Library
Add File...
Manage Libraries...
Add .ZIP Library...

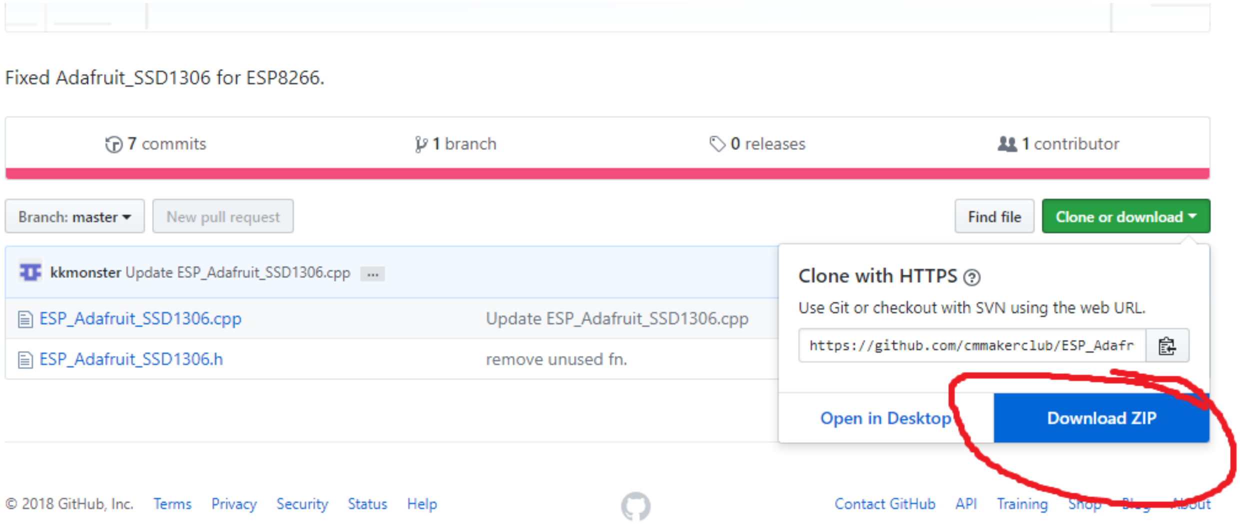
```
void
  //
}

void loop() {
```

And search for it: (mine works with the latest version and the 1.0.2, I recommend you test 1.0.2 first and if it doesn't work, you can update it.) **From now on, at this tutorial I will use the lastest version of Adafruit_GFX.h.**



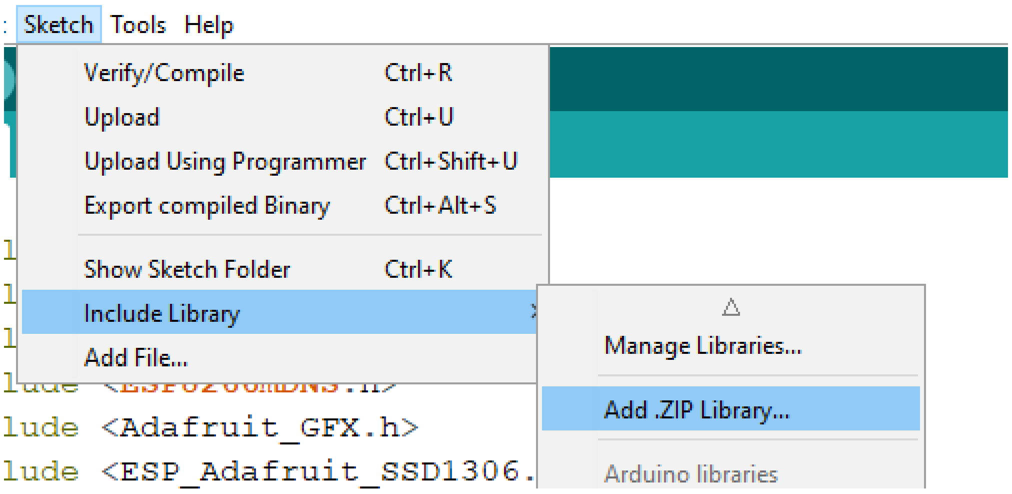
Adafruit GFX Library by **Adafruit** Version **1.2.3** INSTALLED
Adafruit GFX graphics core library, this is the 'core' class that all our other graphics libraries derive from. Install this library in addition to the display library for your hardware.
More info

the **ESP_Adafruit_SSD1306.h** you need to google it and download the github repo, and install it like so:
go here https://github.com/cmmakerclub/ESP_Adafruit_SSD1306

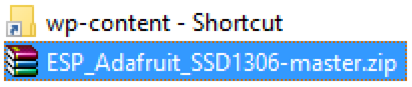

Fixed Adafruit_SSD1306 for ESP8266.
7 commits
1 branch
0 releases
1 contributor
Branch: master
New pull request
Find file
Clone or download
kkmonster Update ESP_Adafruit_SSD1306.cpp
ESP_Adafruit_SSD1306.cpp Update ESP_Adafruit_SSD1306.cpp
ESP_Adafruit_SSD1306.h remove unused fn.
Clone with HTTPS
Use Git or checkout with SVN using the web URL.
https://github.com/cmmakerclub/ESP_Adafr
Open in Desktop
Download ZIP
© 2018 GitHub, Inc. Terms Privacy Security Status Help
Contact GitHub API Training Shop Blog About

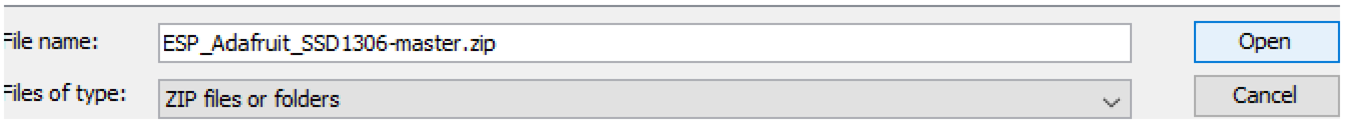
download as zip

And install it like so:



select it, click open and it will be installed almost right away, reinstall Arduino ide.

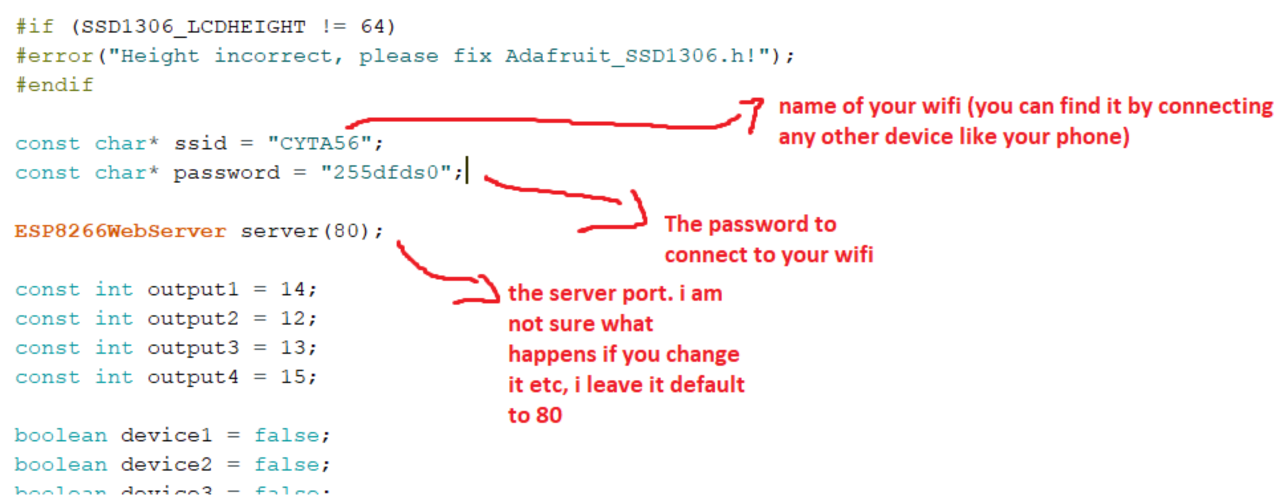




File name: ESP_Adafruit_SSD1306-master.zip

Files of type: ZIP files or folders

You are ready, you have the libraries to help the IDE compile the program, all you need to do is to get the sketch and change these:

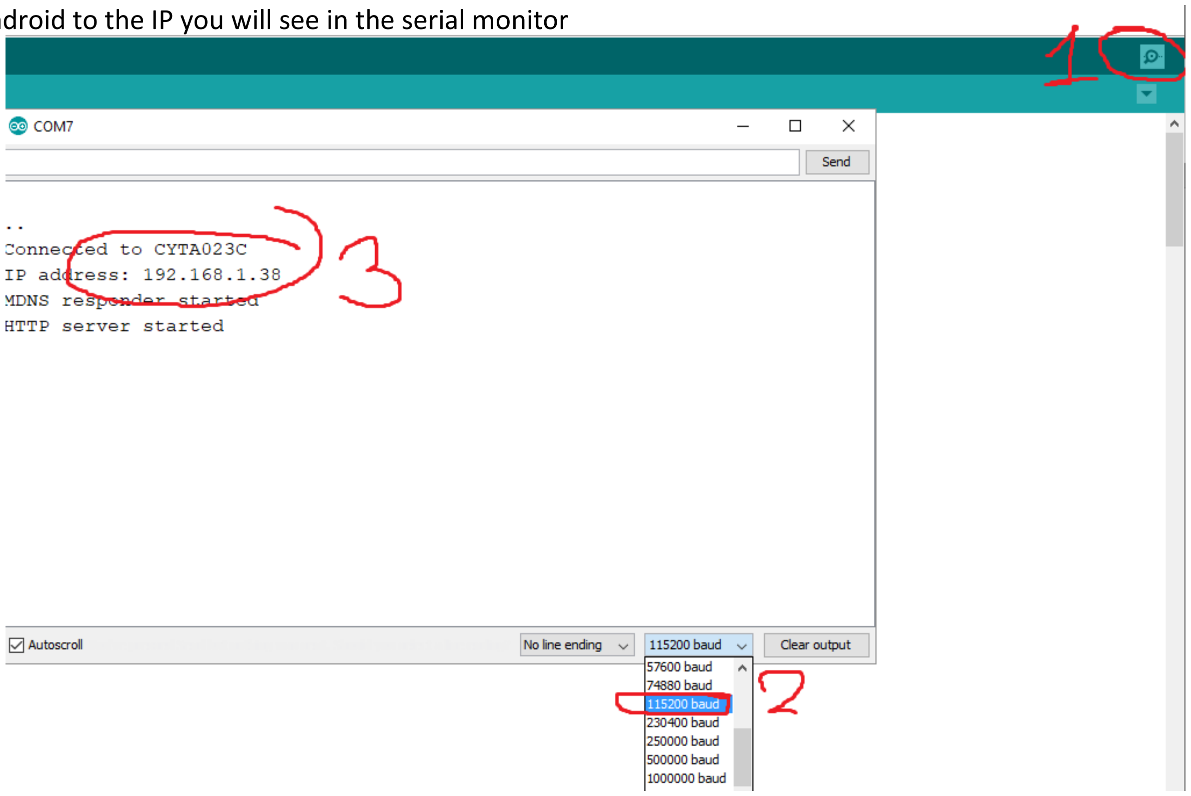


```
#if (SSD1306_LCDHEIGHT != 64)
#error("Height incorrect, please fix Adafruit_SSD1306.h!");
#endif

const char* ssid = "CYTA56";
const char* password = "255dfds0";

ESP8266WebServer server(80);

const int output1 = 14;
const int output2 = 12;
const int output3 = 13;
const int output4 = 15;

boolean device1 = false;
boolean device2 = false;
```

name of your wifi (you can find it by connecting any other device like your phone)
The password to connect to your wifi
the server port. i am not sure what happens if you change it etc, i leave it default to 80

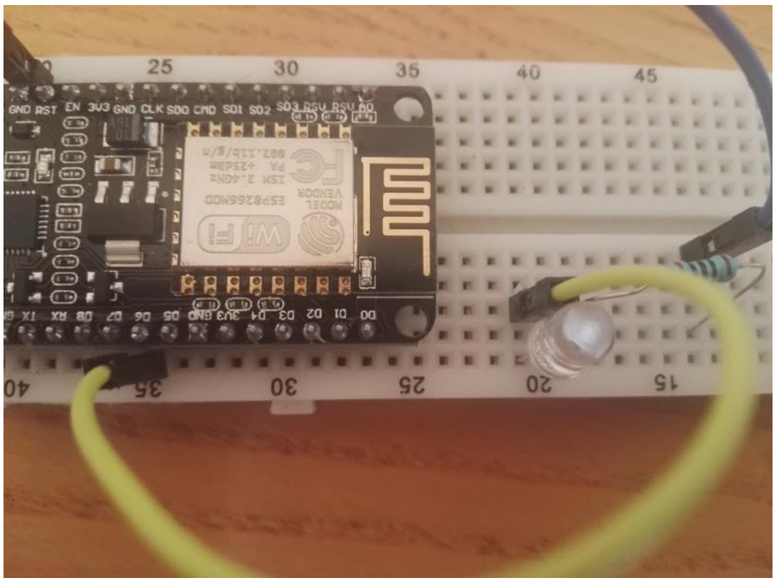
note that the ‘server port’ may needs to be changed if you are encountering issues and nothing else works for you. --The ESP8266WebServer server(θ) picks the θ port (endpoint of communication, not to be confused with com port) of our wifi to be connected to and communicate through the θ port of your router/modem. In theory, you can use any port you want from 0 to 1023, but some ports are usually/already in use from your computer by default and are in use from your internet or modem. Here you can check some of them: https://en.wikipedia.org/wiki/Port_(computer_networking)#Common_port_numbers

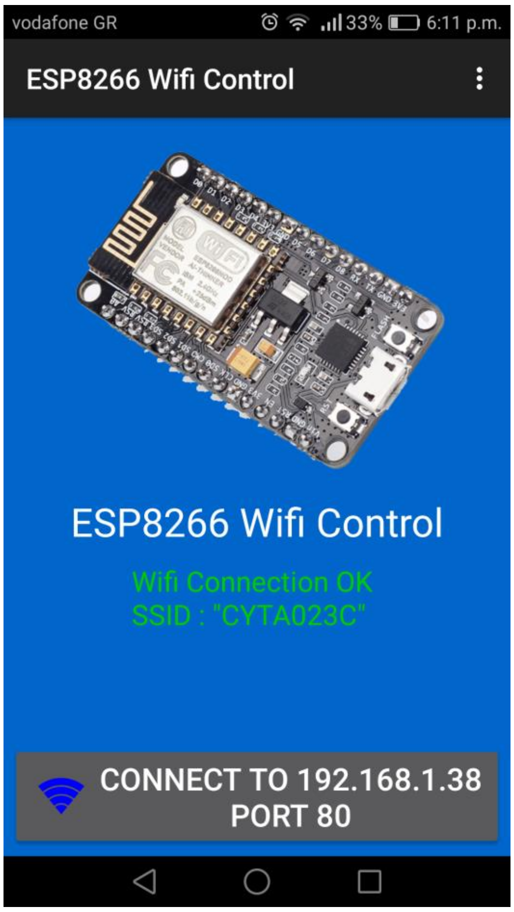
ok, upload it, it may not upload from the first try, happens to me usually on this project… try to compile first, and then upload..
Ok things are straight forward from now on, we wont use relays, but only leds.
connect the corresponding pins:
const int output1 = 14;
const int output2 = 12;
const int output3 = 13;
const int output4 = 15;
to some leds, follow the video I linked above https://www.youtube.com/watch?v=fVpAN3adK9A to connect your android to the IP you will see in the serial monitor



```
..
Connected to CYTA023C
IP address: 192.168.1.38
MDNS responder started
HTTP server started
```

Here are my settings/ I used D6 pin (output 2) and blinked the led/ controlled the led using my cell phone.

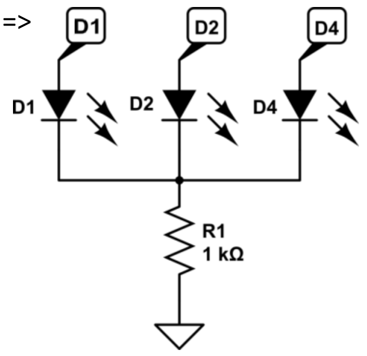




You can also paste this IP address (192.168…) to a search engine from toyr computer and see the condition (on/off) of each pin.

After you manage connecting and controlling the leds (and if the online project is the same and the play store app hasn't change, and you have(probably) the same ESP full dev board version as mine), you will notice some things:

1) if you connect all the GPIO pins of the ESP full dev board to leds just to check whats going on exactly, you will notice that pins 9 and 10 are always active. Don't mind them for now, this is normal.
2) There is a chance that your code wont work, because you connected more leds to the board. Do not connect more leds than the ones you need. some GPIO pins are used when the ESP full dev board boots and connecting anything there will cause its boot to fail
3) There is a chance you used the same resistor to protect all the leds like so: => if you did that, make sure all your LEDs have the same voltage drop, aka are similar leds. If one of them (for example D1 Led) has smaller voltage drop than the others, the D2 and D4 leds will not light.



--Now, I am checking the code to see whats exactly going on, for now I have managed to "throw away" the part of the code that outputs to the screen connected to the ESP full dev board. I don't use the screen anyway.

--you also **don't have to** #include <ESP8266WiFi.h>, cause if you check the #include <ESP8266WebServer.h> (which we include), documentary:
https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WebServer/src/ESP8266WebServer.h
you will see that it includes the ESP8266WiFi.h on its own anyway.
-- We don't need #include <Adafruit_GFX.h> since its used for graphics (aka to display to screen, which I don't use)
--#include <ESP_Adafruit_SSD1306.h> is also used for the screen, I wont need it.
--Im also preety sure #define OLED_RESET 4 is used for the screen, so ill comment it out for now.
--#include <ESP8266mDNS.h> is used upon calling Wifi.begin(,) so we need it.
Side note: if you don't already know, a **multicast DNS** (**mDNS**) is a protocol that when you are visiting/searching for a website, it translates this website's ip into a name (the name of the website), thus shoing us the name of the website and not its ip address. this makes it easier for us to look up for websites (we just look for the name, not the ip, and the **multicast DNS** (**mDNS**) translates the name).

Note: *GPIO pins 9 and 10 are not used for GPIO. They are 'extra pins'. Ignore them for now and don't use them in your code. You may as well not connect leds to them.*

Also, Pin TXD1 outputs the same info (transmission) as the TXD0. Note that you cannot use TX, TXD1 RX, RXD2, TXD2 if you are using the serial monitor, since they send and receive data from the serial monitor to the computer. That's right, if you want to use these five pins you will have to NOT use the serial monitor.

Don't use pins 0, 2 and 15 as well, and don't connect leds to them for now, until you get some code to run. They are 'special' pins and define how the ESP full dev board starts/boots. So if you connect anything wrong (or program these pins the wrong way) your board will boot faulty.

So, the code now looks like this:

Code without the external screen

```
//#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
//#include <Adafruit_GFX.h>
//#include <ESP_Adafruit_SSD1306.h> //used for the screen as well
//#define OLED_RESET 4
//Adafruit_SSD1306 display(OLED_RESET);
//#if (SSD1306_LCDHEIGHT != 64)
//#error("Height incorrect, please fix Adafruit_SSD1306.h!");
//#endif
const char* ssid = "CHOLC";
const char* password = "DFSSA";
ESP8266WebServer server(80);
const int output1 = 14;
const int output2 = 12;
const int output3 = 13;
const int output4 = 15;
boolean device1 = false;
boolean device2 = false;
boolean device3 = false;
boolean device4 = false;
void handleRoot() {
//digitalWrite(led, 1);
//server.send(200, "text/plain", "hello from esp8266!");
//digitalWrite(led, 0);
String cmd;
cmd += "<!DOCTYPE HTML>\r\n";
cmd += "<html>\r\n";
//cmd += "<header><title>ESP8266 Webserver</title><h1>\"ESP8266 Web Server Control\"</h1></header>";
cmd += "<head>";
cmd += "<meta http-equiv='refresh' content='5'/>";
cmd += "</head>";
if(device1){
cmd +=("<br/>Device1  : ON");
}
else{
cmd +=("<br/>Device1  : OFF");
}
if(device2){
cmd +=("<br/>Device2  : ON");
}
else{
cmd +=("<br/>Device2  : OFF");
}
if(device3){
cmd +=("<br/>Device3  : ON");
}
else{
cmd +=("<br/>Device3  : OFF");
}
```

```
if(device4){
cmd +=("<br/>Device4  : ON");
}
else{
cmd +=("<br/>Device4  : OFF");
}
cmd += "<html>\r\n";
server.send(200, "text/html", cmd);
}
//============================================================error================================
===
void handleNotFound(){
//digitalWrite(led, 1);
String message = "File Not Found\n\n";
message += "URI: ";
message += server.uri();
message += "\nMethod: ";
message += (server.method() == HTTP_GET)?"GET":"POST";
message += "\nArguments: ";
message += server.args();
message += "\n";
for (uint8_t i=0; i<server.args(); i++){
message += " " + server.argName(i) + ": " + server.arg(i) + "\n";
}
server.send(404, "text/plain", message);
//digitalWrite(led, 0);
}
//============================================setup============================================
void setup(void){
pinMode(output1, OUTPUT);
pinMode(output2, OUTPUT);
pinMode(output3, OUTPUT);
pinMode(output4, OUTPUT);
digitalWrite(output1, LOW);
digitalWrite(output2, LOW);
digitalWrite(output3, LOW);
digitalWrite(output4, LOW);
Serial.begin(115200);
WiFi.begin(ssid, password);  //from #include <ESP8266mDNS.h>
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")) {
Serial.println("MDNS responder started");
}
server.on("/", handleRoot);
server.on("/status1=1", [](){
```

```
server.send(200, "text/plain", "device1 = ON");
digitalWrite(output1, HIGH);
device1 = true;
});
server.on("/status1=0", [](){
server.send(200, "text/plain", "device1 = OFF");
digitalWrite(output1, LOW);
device1 = false;
});
server.on("/status2=1", [](){
server.send(200, "text/plain", "device2 = ON");
digitalWrite(output2, HIGH);
device2 = true;
});
server.on("/status2=0", [](){
server.send(200, "text/plain", "device2 = OFF");
digitalWrite(output2, LOW);
device2 = false;
});
server.on("/status3=1", [](){
server.send(200, "text/plain", "device3 = ON");
digitalWrite(output3, HIGH);
device3 = true;
});
server.on("/status3=0", [](){
server.send(200, "text/plain", "device3 = OFF");
digitalWrite(output3, LOW);
device3 = false;
});
server.on("/status4=1", [](){
server.send(200, "text/plain", "device4 = ON");
digitalWrite(output4, HIGH);
device4 = true;
});
server.on("/status4=0", [](){
server.send(200, "text/plain", "device4 = OFF");
digitalWrite(output4, LOW);
device4 = false;
});
server.onNotFound(handleNotFound);
server.begin();
Serial.println("HTTP server started");
}
void loop(void){
server.handleClient();
}
```

Analyzing the code

Okay lets check out what we have here and help you learn how to analyze a piece of code and understand better whats going on.

Just a reminder that I have all of my ESP full dev board GPIO pins connected to leds so that I can see when a pin is activated. I recommend you do the same.

//All the libraries we will need.

#include <WiFiClient.h>

#include <ESP8266WebServer.h>

#include <ESP8266mDNS.h>

//your password for your ssid (wifi network name)

const char* ssid = "Holdsds";

const char* password = "45_4fgdfg";

//This command picks the port (endpoint of communication, not to be confused with com port) of our wifi to be connected to. In theory, you can use any port you want from 0 to 1023, but some ports are usually/already in use from your computer by default and are in use from your internet or modem. Here you can check some of them: https://en.wikipedia.org/wiki/Port_(computer_networking)#Common_port_numbers .

ESP8266WebServer server(80); //// Create a webserver object that listens for HTTP request on port 80. We //pick the name of the ESPWebServer. in this case, he chose the name 'server'. you can pick any name you want, like //"**my_server**" or "**mitsos**".

//These are the four pins we will control

const int output1 = 14;

const int output2 = 12;

const int output3 = 13;

const int output4 = 15;

//obviously here he is making a bollean for each of the 'devices', assigning it to false (obviously because its off by default).

boolean device1 = false;

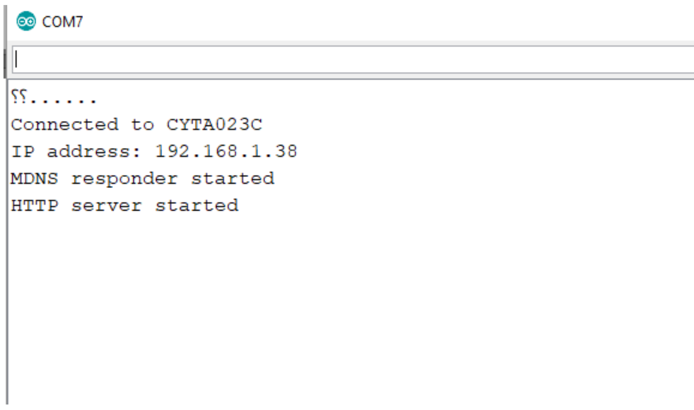
boolean device2 = false;

boolean device3 = false;

boolean device4 = false;

//From my little experience in website making, every time he calls the cmd command, he adds another piece of code in order to make a website! so the purple part is the title and the head of the website. And the rest of the code (blue part) is not pre-defined, but it checks out which led is on/off, and depending on that, it changes its output.

THIS MEANS THERE IS ALSO A WEBSITE RUNNING SOMEWHERE. Cool, so we need to find its ip, to see how it looks and how it acts. Open up and Check your serial monitor before you upload your sketch to the ESP full dev board. you will see it gives you an address. copy paste this address to google or any other search engine and you will see the state of these 4 pins (need to refresh the page to refresh the pin-states).

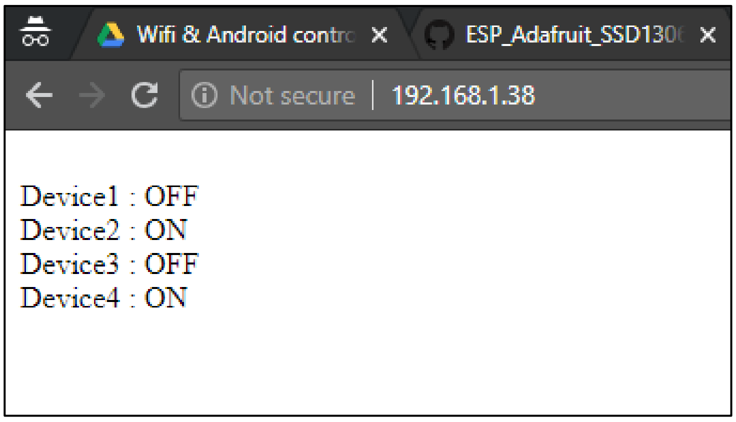


```
ʕʕ......
Connected to CYTA023C
IP address: 192.168.1.38
MDNS responder started
HTTP server started
```

Now, the red part is the "title" of the website. If you uncomment that, it will add a title to the website.

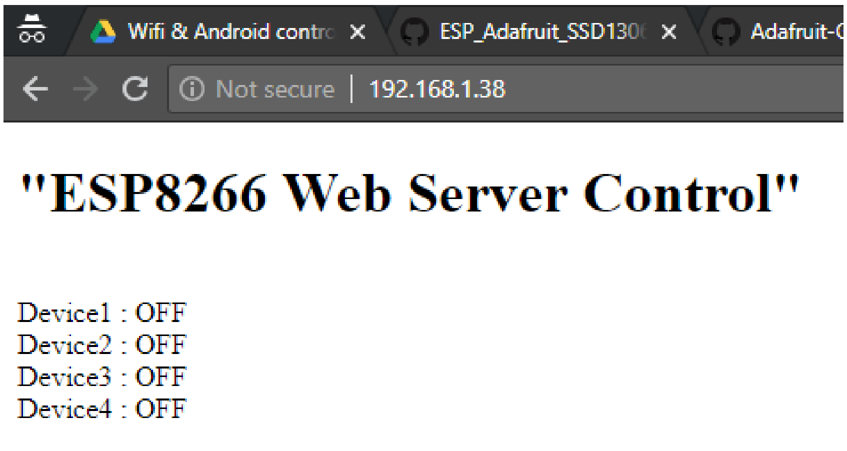
The Green part of the code tells the search engine every when (in seconds) the website will be refreshed. We need this so we wont resresh using 'our hands', since the pin state is not in 'live feedback', This means that we wont see any pin change if we somehow don't refresh the page. In this case, it refreshes every 5 sec.

```
void handleRoot() {
 String cmd;    //this creates a string named cmd, where we will write the html code and send it to the
server.
   cmd += "<!DOCTYPE HTML>\r\n";
   cmd += "<html>\r\n";
   //cmd += "<header><title>ESP8266 Webserver</title><h1>\"ESP8266 Web Server
Control\"</h1></header>";
   cmd += "<head>"
cmd += "<meta http-equiv='refresh' content='5'/>";
   cmd += "</head>";

   if(device1){
    cmd +=("<br/>Device1  : ON");
   }
   else{
    cmd +=("<br/>Device1  : OFF");
   }
   if(device2){
    cmd +=("<br/>Device2  : ON");
   }
   else{
    cmd +=("<br/>Device2  : OFF");
   }
    if(device3){
    cmd +=("<br/>Device3  : ON");
   }
   else{
    cmd +=("<br/>Device3  : OFF");
   }
   if(device4){
    cmd +=("<br/>Device4  : ON");
   }
   else{
    cmd +=("<br/>Device4  : OFF");
   }
   cmd += "<html>\r\n";
   server.send(200, "text/html", cmd); //we send the cmd string with all the code in it, to the server.
}
```



Device1 : OFF
Device2 : ON
Device3 : OFF
Device4 : ON

This is how it looks like from google, with the red code part commented



"ESP8266 Web Server Control"

Device1 : OFF
Device2 : OFF
Device3 : OFF
Device4 : OFF

This is how it looks like from google, with the red code part uncommented

Note that:The **server.send(a,b,c)** function is the reply of the server. It takes as first parameter '**a**' the HTTP status code. This HTTP status code (200 in this case) is the "Ok" response that the server should reply with.
The second parameter '**b**' is the kind of text that the server is sending back, aka the type of **cmd** text (third variable '**c**').
The third variable **c** is the html text the server is sending.

So with the help of **void handleRoot()** function, we make sure the server replies with the **server.send()** function.
You can see a list with http status code here https://en.wikipedia.org/wiki/List_of_HTTP_status_codes
The next part of this code that we will check, is the function **void handleNotFound()** which, as its name suggests, must be a error-checker function. meaning that, on a controlled environment aka (stable wifi signal, many successful

connection attempts), this function is optional and we can trim it or even not use it at all, in order to make our code smaller. so, I commented out the function:

```
        //void handleNotFound(){
        // //digitalWrite(led, 1);
        // String message = "File Not Found\n\n";
        // message += "URI: ";
        // message += server.uri();
        // message += "\nMethod: ";
        // message += (server.method() == HTTP_GET)?"GET":"POST";
        // message += "\nArguments: ";
        // message += server.args();
        // message += "\n";
        // for (uint8_t i=0; i<server.args(); i++){
        //   message += " " + server.argName(i) + ": " + server.arg(i) + "\n";
        // }
        // server.send(404, "text/plain", message);
        // //digitalWrite(led, 0);
//}
```

And I also commented out the part of the code where the function is used to check the error:

```
// server.onNotFound(handleNotFound);
```

Its time to check the **void setup()** function.

```
//we set the pins to output since we want to control the leds
        pinMode(output1, OUTPUT);
        pinMode(output2, OUTPUT);
        pinMode(output3, OUTPUT);
        pinMode(output4, OUTPUT);
//we initialize them to low (off)
         digitalWrite(output1, LOW);
         digitalWrite(output2, LOW);
         digitalWrite(output3, LOW);
         digitalWrite(output4, LOW);

         Serial.begin(115200); //in order to send data from/to the serial monitor.
         WiFi.begin(ssid, password); //from #include <ESP8266mDNS.h>
         Serial.println("");

         // Wait for connection
         while (WiFi.status() != WL_CONNECTED) { //while connection hasn't established,
           delay(500);
           Serial.print("."); //print dots
         }
         Serial.println(""); //used to change line without printing anything
        //Output the name of the connected network and the ip address
        Serial.print("Connected to ");
         Serial.println(ssid);
         Serial.print("IP address: ");
         Serial.println(WiFi.localIP());
```

//The purple code below is responsible for a feedback –kind of error-control- like what we had before, like the void handleNotFound() function we commented out above. so we can safely remove this one as well. But note that, in this case, by using MDNS.begin() inside the ‘if’ statement, not only checks if the condition is true or false, but also runs the command MDNS.begin at the moment, enabling the mDNS for our esp. If you don’t already know, the DNS is a name system, “Doman Name System”. It actually translates the ip of a website, to a domain name. In this case, we give the name ‘esp8266’ to out local ip address. Now you may be wondering how can this be usefull for you. Simply open the cmd of windows and ping to *esp8266.local* you will see that it will respond (there are many reports that it doesn’t respond due to the fact that windows cannot resolve mDNS names, so you may need a special third party program to see the respond. Don’t worry its not so necessary as long as you can see the respond from its IP and everything else works great). So, instead of trying to remember its ip, we just give it a name and use that instead.

```
 if (MDNS.begin("esp8266")) // Start the mDNS responder for esp8266.local. (use any mDns name you
//want)
 {
   Serial.println("MDNS responder started");
} //Here, I would like to add another two lines of code (in case you don’t remove it or commented out as I
//did):
 else
{Serial.println("Error setting up MDNS responder!"); }
}
server.on("/", handleRoot); //Tells the server to call handleRoot function when a client requests URI "/".
```

//Side note: The server listens for HTTP requests from clients (aka your android or a browser) using the
//**server.handleClient();** function inside the void loop(){}. In our case, it listens these requests from our android.

//the lines below in Blue are the part of code that is running when and whenever **server.handleClient()** (that is
//located in **loop()**) detects new requests.
//During the loop, we constantly check if a new HTTP request is received by running server.handleClient . If
//handleClient detects new requests, it will automatically execute the right functions that we specified in the setup.

```
 server.on("/status1=1", [](){
   server.send(200, "text/plain", "device1 = ON");
   digitalWrite(output1, HIGH);
   device1 = true;
 });
 server.on("/status1=0", [](){
   server.send(200, "text/plain", "device1 = OFF");
   digitalWrite(output1, LOW);
   device1 = false;
 });
 server.on("/status2=1", [](){
   server.send(200, "text/plain", "device2 = ON");
   digitalWrite(output2, HIGH);
   device2 = true;
 });
 server.on("/status2=0", [](){
   server.send(200, "text/plain", "device2 = OFF");
   digitalWrite(output2, LOW);
```

```
            device2 = false;
          });
          server.on("/status3=1", [](){
            server.send(200, "text/plain", "device3 = ON");
            digitalWrite(output3, HIGH);
            device3 = true;
          });
          server.on("/status3=0", [](){
            server.send(200, "text/plain", "device3 = OFF");
            digitalWrite(output3, LOW);
            device3 = false;
          });

          server.on("/status4=1", [](){
            server.send(200, "text/plain", "device4 = ON");
            digitalWrite(output4, HIGH);
            device4 = true;
          });
          server.on("/status4=0", [](){
            server.send(200, "text/plain", "device4 = OFF");
            digitalWrite(output4, LOW);
            device4 = false;
  });
```

//Then we start listening for HTTP requests by using server.begin .

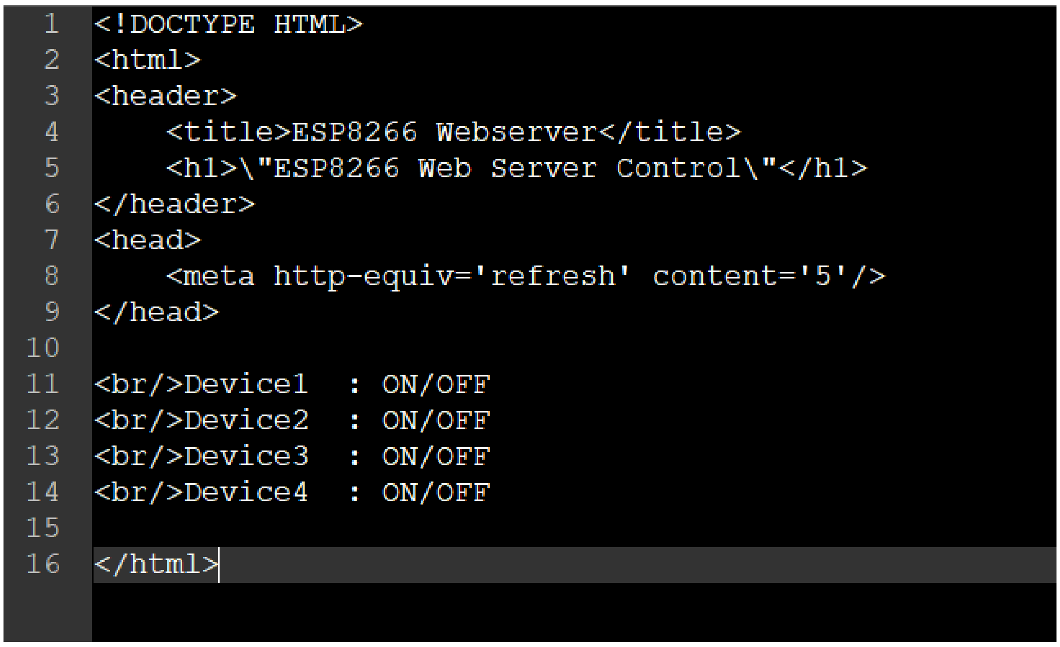
```
        server.begin(); // Actually start the server
        Serial.println("HTTP server started");

        void loop(void)
        {
          server.handleClient(); // Listen for HTTP requests from clients
}
```

//if we didn't specify that we wanted to run the handleRoot(), the server would just run the right code-the blue lines above- (every command that the server.on(){} instance gives), which in our case would be the commands of the android. so if we remove the handleRoot(), we will still be able to use our android to control the leds, but we wont be seing any website at our local ip.

Other sources: https://tttapa.github.io/ESP8266/Chap01%20-%20ESP8266.html
Something that will be needed for the oncoming project: If a client (browser or android) requests from the server (esp) the "/", this means by default (based on the ESP library that we added) that the client is attempting to access the root of the server.
On the example code above, the client (android) only send /status*X* = *Boolean* and the server responded to that, according to what server.on(){} was telling it (to the server) what to do.

Next, we will try to make google (the browser) to make requests from the server, so that we know exactly what we are requesting, since we cannot analyze the android application.

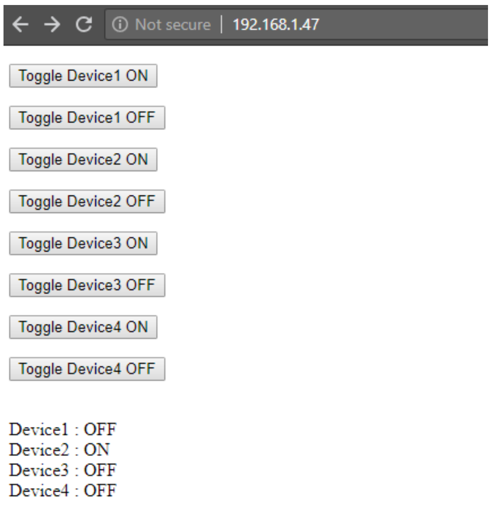
If you check the cmd string that is created, you will notice that this is the html code on our website (the \r\n on the code tells the html to change line):



```
<!DOCTYPE HTML>
<html>
<header>
    <title>ESP8266 Webserver</title>
    <h1>\"ESP8266 Web Server Control\"</h1>
</header>
<head>
    <meta http-equiv='refresh' content='5'/>
</head>

<br/>Device1  : ON/OFF
<br/>Device2  : ON/OFF
<br/>Device3  : ON/OFF
<br/>Device4  : ON/OFF

</html>
```

I recommend you check out some basics of the html markup language before continuing, I wont be explaining much on it.

Now, I will make the server look like this:



```
Device1 : OFF
Device2 : ON
Device3 : OFF
Device4 : OFF
```

So we wont need the android application anymore, and we will be able to control the leds from the computer as well!
note that I am not an expert on html, I nave only once make a small website, just to know how html and css works. We wont use any css here.
Reminder: If you make your website on the ESP too big, it will have problem on displaying it on the client (chrome). There are some ways to bypass that, we will see how later on.

The code for the above picture:

```
<!DOCTYPE HTML>
<html>
<header>
        <title>ESP8266 Webserver</title>
        <h1>\"ESP8266 Web Server Control\"</h1>
</header>
<head>
        <meta http-equiv='refresh' content='5'/>
</head>
```

```
<!--Toggle Device1 ON-->
<p>
<form action=\"/status1=1\" method=\"POST\">
<input type=\"submit\" value=\"Toggle Device1 ON\">
</form>
</p>

<!--Toggle Device1 OFF-->
<p>
<form action=\"/status1=0\" method=\"POST\">
<input type=\"submit\" value=\"Toggle Device1 OFF\">
</form>
</p>

<!--Toggle Device2 ON-->
<p>
<form action=\"/status2=1\" method=\"POST\">
<input type=\"submit\" value=\"Toggle Device2 ON\">
</form>
</p>

<!--Toggle Device2 OFF-->
<p>
<form action=\"/status2=0\" method=\"POST\">
<input type=\"submit\" value=\"Toggle Device2 OFF\">
</form>
</p>

<!--Toggle Device3 ON-->
<p>
<form action=\"/status3=1\" method=\"POST\">
<input type=\"submit\" value=\"Toggle Device3 ON\">
</form>
</p>

<!--Toggle Device3 OFF-->
<p>
<form action=\"/status3=0\" method=\"POST\">
<input type=\"submit\" value=\"Toggle Device3 OFF\">
</form>
</p>

<!--Toggle Device4 ON-->
<p>
<form action=\"/status4=1\" method=\"POST\">
<input type=\"submit\" value=\"Toggle Device4 ON\">
</form>
</p>
```

```
        <!--Toggle Device4 OFF-->
        <p>
        <form action=\"/status4=0\" method=\"POST\">
        <input type=\"submit\" value=\"Toggle Device4 OFF\">
        </form>
        </p>
</html>
```

As you may notice, in order to create these “Toggle Device buttons”, you need to make something called a “form” on html. so a **<form action=\”/foo\”** will send to the server a /foo request. we then need to define the method of the form. We will either use the GET (retrieve from sever) or POST (Send to the server) method. Since we want to send data to the server, we use the POST method. We also need to declare the input type, so html will know this is a button, and last, the **value**, which will be the name of the button.

We need to implement this code on the handleRoot() function. I did it for you. check it out

```
        void handleRoot() {
          String cmd;
            cmd += "<!DOCTYPE HTML>\r\n";
            cmd += "<html>\r\n";
            //cmd += "<header><title>ESP8266 Webserver</title><h1>\"ESP8266 Web Server
        Control\"</h1></header>";
            cmd += "<head>";
            cmd += "<meta http-equiv='refresh' content='100'/>";
            cmd += "</head>";

          //<!--Toggle Device1 ON-->
            cmd+="<p>\r\n <form action=\"/status1=1\" method=\"POST\">\r\n <input type=\"submit\"
        value=\"Toggle Device1 ON\">\r\n </form>\r\n </p>\r\n";

          //<!--Toggle Device1 OFF-->
            cmd+="<p>\r\n <form action=\"/status1=0\" method=\"POST\">\r\n <input type=\"submit\"
        value=\"Toggle Device1 OFF\">\r\n </form>\r\n </p>\r\n";

          //<!--Toggle Device2 ON-->
            cmd+="<p>\r\n <form action=\"/status2=1\" method=\"POST\">\r\n <input type=\"submit\"
        value=\"Toggle Device2 ON\">\r\n </form>\r\n </p>\r\n";

          //<!--Toggle Device2 OFF-->
            cmd+="<p>\r\n <form action=\"/status2=0\" method=\"POST\">\r\n <input type=\"submit\"
        value=\"Toggle Device2 OFF\">\r\n </form>\r\n </p>\r\n";

          //<!--Toggle Device3 ON-->
            cmd+="<p>\r\n <form action=\"/status3=1\" method=\"POST\">\r\n <input type=\"submit\"
        value=\"Toggle Device3 ON\">\r\n </form>\r\n </p>\r\n";

          //<!--Toggle Device3 OFF-->
```

```
    cmd+="<p>\r\n <form action=\"/status3=0\" method=\"POST\">\r\n <input type=\"submit\"
value=\"Toggle Device3 OFF\">\r\n </form>\r\n </p>\r\n";

   //<!--Toggle Device4 ON-->
    cmd+="<p>\r\n <form action=\"/status4=1\" method=\"POST\">\r\n <input type=\"submit\"
value=\"Toggle Device4 ON\">\r\n </form>\r\n </p>\r\n";

   //<!--Toggle Device4 OFF-->
    cmd+="<p>\r\n <form action=\"/status4=0\" method=\"POST\">\r\n <input type=\"submit\"
value=\"Toggle Device4 OFF\">\r\n </form>\r\n </p>\r\n";

    if(device1){
      cmd +=("<br/>Device1  : ON");
    }
    else{
      cmd +=("<br/>Device1  : OFF");
    }

    if(device2){
      cmd +=("<br/>Device2  : ON");
    }
    else{
      cmd +=("<br/>Device2  : OFF");
    }

     if(device3){
      cmd +=("<br/>Device3  : ON");
    }
    else{
      cmd +=("<br/>Device3  : OFF");
    }

    if(device4){
      cmd +=("<br/>Device4  : ON");
    }
    else{
      cmd +=("<br/>Device4  : OFF");
    }

    cmd += "</html>\r\n";
    server.send(200, "text/html", cmd);
}
```

You don't need to change anything else from the code 1.0. I am also pasting the whole code below, so you can copy-paste and run it.

```
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include <Adafruit_GFX.h>
#include <ESP_Adafruit_SSD1306.h>
const char* ssid = "CYTA023C";
const char* password = "026813_virus";
ESP8266WebServer server(80);
const int output1 = 4;
const int output2 = 5;
const int output3 = 12;
const int output4 = 14;
boolean device1 = false;
boolean device2 = false;
boolean device3 = false;
boolean device4 = false;
//==================================FUNCTIONS=========================================
void handleRoot() {
 String cmd;
    cmd += "<!DOCTYPE HTML>\r\n";
    cmd += "<html>\r\n";
    //cmd += "<header><title>ESP8266 Webserver</title><h1>\"ESP8266 Web Server
Control\"</h1></header>";
    cmd += "<head>";
    cmd += "<meta http-equiv='refresh' content='100'/>";
    cmd += "</head>";

  //<!--Toggle Device1 ON-->
    cmd+="<p>\r\n <form action=\"/status1=1\" method=\"POST\">\r\n <input type=\"submit\"
value=\"Toggle Device1 ON\">\r\n </form>\r\n </p>\r\n";

  //<!--Toggle Device1 OFF-->
    cmd+="<p>\r\n <form action=\"/status1=0\" method=\"POST\">\r\n <input type=\"submit\"
value=\"Toggle Device1 OFF\">\r\n </form>\r\n </p>\r\n";

  //<!--Toggle Device2 ON-->
    cmd+="<p>\r\n <form action=\"/status2=1\" method=\"POST\">\r\n <input type=\"submit\"
value=\"Toggle Device2 ON\">\r\n </form>\r\n </p>\r\n";

  //<!--Toggle Device2 OFF-->
    cmd+="<p>\r\n <form action=\"/status2=0\" method=\"POST\">\r\n <input type=\"submit\"
value=\"Toggle Device2 OFF\">\r\n </form>\r\n </p>\r\n";

  //<!--Toggle Device3 ON-->
    cmd+="<p>\r\n <form action=\"/status3=1\" method=\"POST\">\r\n <input type=\"submit\"
value=\"Toggle Device3 ON\">\r\n </form>\r\n </p>\r\n";
```

```
   //<!--Toggle Device3 OFF-->
     cmd+="<p>\r\n <form action=\"/status3=0\" method=\"POST\">\r\n <input type=\"submit\"
value=\"Toggle Device3 OFF\">\r\n </form>\r\n </p>\r\n";

   //<!--Toggle Device4 ON-->
     cmd+="<p>\r\n <form action=\"/status4=1\" method=\"POST\">\r\n <input type=\"submit\"
value=\"Toggle Device4 ON\">\r\n </form>\r\n </p>\r\n";

   //<!--Toggle Device4 OFF-->
     cmd+="<p>\r\n <form action=\"/status4=0\" method=\"POST\">\r\n <input type=\"submit\"
value=\"Toggle Device4 OFF\">\r\n </form>\r\n </p>\r\n";

     if(device1){
       cmd +=("<br/>Device1  : ON");
     }
     else{
       cmd +=("<br/>Device1  : OFF");
     }

     if(device2){
       cmd +=("<br/>Device2  : ON");
     }
     else{
       cmd +=("<br/>Device2  : OFF");
     }

      if(device3){
       cmd +=("<br/>Device3  : ON");
     }
     else{
       cmd +=("<br/>Device3  : OFF");
     }

     if(device4){
       cmd +=("<br/>Device4  : ON");
     }
     else{
       cmd +=("<br/>Device4  : OFF");
     }

     cmd += "</html>\r\n";
     server.send(200, "text/html", cmd);
}

//====================================SETUP=========================================
void setup(void){
  pinMode(output1, OUTPUT);
```

```
 pinMode(output2, OUTPUT);
 pinMode(output3, OUTPUT);
 pinMode(output4, OUTPUT);

 digitalWrite(output1, LOW);
 digitalWrite(output2, LOW);
 digitalWrite(output3, LOW);
 digitalWrite(output4, LOW);

 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")) {
  Serial.println("MDNS responder started");
 }
//====================================RIGHT FUNCTIONS====================
 server.on("/", handleRoot); //wont use HTTP_GET for now

 server.on("/status1=1", [](){
  server.send(200, "text/plain", "device1 = ON");
  digitalWrite(output1, HIGH);
  device1 = true;
 });
 server.on("/status1=0", [](){
  server.send(200, "text/plain", "device1 = OFF");
  digitalWrite(output1, LOW);
  device1 = false;
 });
 server.on("/status2=1", [](){
  server.send(200, "text/plain", "device2 = ON");
  digitalWrite(output2, HIGH);
  device2 = true;
 });

 server.on("/status2=0", [](){
  server.send(200, "text/plain", "device2 = OFF");
  digitalWrite(output2, LOW);
```

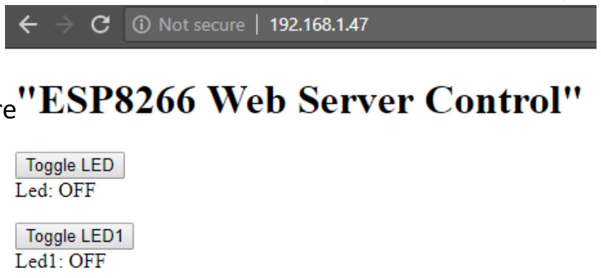
```
    device2 = false;
  });
  server.on("/status3=1", [](){
    server.send(200, "text/plain", "device3 = ON");
    digitalWrite(output3, HIGH);
    device3 = true;
  });
  server.on("/status3=0", [](){
    server.send(200, "text/plain", "device3 = OFF");
    digitalWrite(output3, LOW);
    device3 = false;
  });
  server.on("/status4=1", [](){
    server.send(200, "text/plain", "device4 = ON");
    digitalWrite(output4, HIGH);
    device4 = true;
  });
  server.on("/status4=0", [](){
    server.send(200, "text/plain", "device4 = OFF");
    digitalWrite(output4, LOW);
    device4 = false;
  });
  server.begin();
  Serial.println("HTTP server started");
}
//===================================================LOOP==========================
void loop(void){
  server.handleClient();
}
```

The \r\n in our html code, help us “see” the code in a better/organized form if we look at the page source, once we open up the IP on google.

Example 2.0 controlling two leds from a browser

As we saw from the previous example, we can use google or any other browser to control the leds and out esp full dev board in general. We don't need a special application like the one we used in example 1.0. Of course, we can only do these things locally, and for now we will have to connect our search engine or android to the same wifi hotspot as our esp.

I will make a final example of controlling the esp full dev board locally. We will now control two leds (pins 4 and 5), but instead of using two buttons for each state of each pin, we will use one button for each pin, which will change the state of the led every time we press it.


Not secure | 192.168.1.47

"ESP8266 Web Server Control"

Led: OFF

Led1: OFF

I also use The wifi multi library, where you can give your esp more than one options on which wifi to be connected ot, and it will automatically connect to the one that has the strongest connection signal.

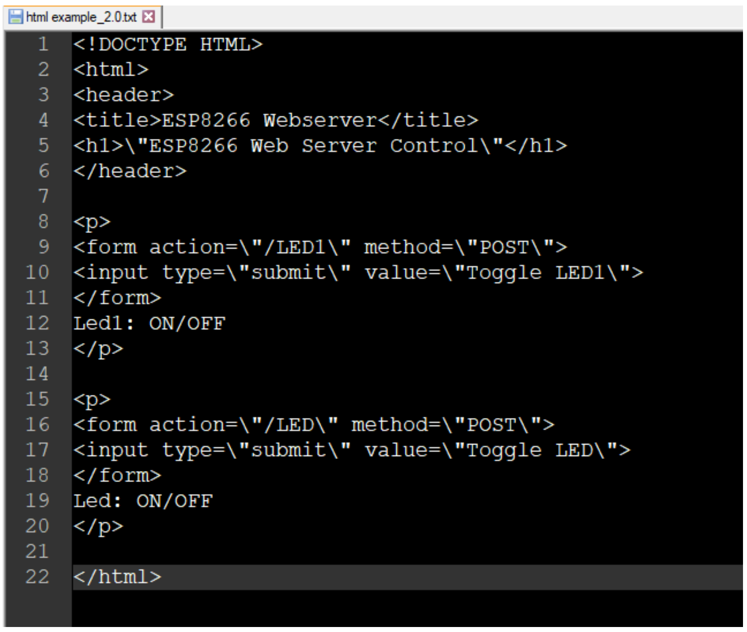
```
#include <WiFiClient.h>
#include <ESP8266WiFiMulti.h>
#include <ESP8266mDNS.h>
#include <ESP8266WebServer.h>
ESP8266WiFiMulti wifiMulti;    // Create an instance of the ESP8266WiFiMulti class, called 'wifiMulti'
ESP8266WebServer server(80);   // Create a webserver object that listens for HTTP request on port 80
const int led = 4;
const int led1= 5;
bool led_=false;
bool led1_=false;
void handleRoot();              // function prototypes for HTTP handlers
void handleLED();
void handleLED1();
void handleNotFound();

String cmd; //to store the html code before we send it
//======================================SETUP======================================
void setup(void){
 Serial.begin(115200);      // Start the Serial communication to send messages to the computer
 delay(10);
 Serial.println('\n');

 pinMode(led, OUTPUT);
  pinMode(led1, OUTPUT);

 wifiMulti.addAP("ssid name", "change pass");   // add Wi-Fi networks you want to connect to
 wifiMulti.addAP("ssid_from_AP_2", "your_password_for_AP_2");
 wifiMulti.addAP("ssid_from_AP_3", "your_password_for_AP_3");

 Serial.println("Connecting ...");
 int i = 0;
```

```
  while (wifiMulti.run() != WL_CONNECTED) { // Wait for the Wi-Fi to connect: scan for Wi-Fi networks, and
connect to the strongest of the networks above
    delay(250);
    Serial.print('.');
  }
  Serial.println('\n');
  Serial.print("Connected to ");
  Serial.println(WiFi.SSID());              // Tell us what network we're connected to
  Serial.print("IP address:\t");
  Serial.println(WiFi.localIP());           // Send the IP address of the ESP8266 to the computer

  if (MDNS.begin("esp8266")) {              // Start the mDNS responder for esp8266.local
    Serial.println("mDNS responder started");
  } else {
    Serial.println("Error setting up MDNS responder!");
  }
//=====================================RIGHT FUNCTIONS=========================
  server.on("/", HTTP_GET, handleRoot);   // Call the 'handleRoot' function when a client requests URI "/"
  server.on("/LED", HTTP_POST, handleLED);// Call the 'handleLED' function when a POST request is made to URI
"/LED"
  server.on("/LED1", HTTP_POST, handleLED1);// Call the 'handleLED' function when a POST request is made to
URI "/LED"
  server.onNotFound(handleNotFound);      // When a client requests an unknown URI (i.e. something other than
"/"), call function "handleNotFound"

  server.begin();                         // Actually start the server
  Serial.println("HTTP server started");
}
//=========================================LOOP==================================
void loop(void){
  server.handleClient();                  // Listen for HTTP requests from clients
}
//==================================FUNCTIONS===================================
void handleRoot() { // When URI / is requested, send a web page with a button to toggle the LED
  cmd="";

 cmd+= "<!DOCTYPE HTML>\r\n <html>\r\n <header>\r\n <title>ESP8266 Webserver</title>\r\n <h1>\"ESP8266
Web Server Control\"</h1>\r\n </header>\r\n <p>\r\n <form action=\"/LED\" method=\"POST\">\r\n <input
type=\"submit\" value=\"Toggle LED\">\r\n </form>\r\n";
  if(led_){
    cmd += ("Led: ON \r\n </p>");
  }
  else{
    cmd +=("Led: OFF\r\n </p>");
  }

   cmd+="<p>\r\n <form action=\"/LED1\" method=\"POST\">\r\n <input type=\"submit\" value=\"Toggle
LED1\">\r\n </form>\r\n ";
```

```
    if(led1_){
    cmd += ("Led1: ON \r\n </p>");
  }
  else{
    cmd +=("Led1: OFF\r\n </p>");
  }
  cmd+= "</html>\r\n";

  server.send(200, "text/html", cmd );
}
void handleLED() {                        // If a POST request is made to URI /LED
  digitalWrite(led,!digitalRead(led));      // Change the state of the LED
  led_=!led_;
  server.sendHeader("Location","/");       // Add a header to respond with a new location for the browser to go to
the home page again
  server.send(303);                        // Send it back to the browser with an HTTP status 303 (See Other) to redirect
}
void handleLED1() {                        // If a POST request is made to URI /LED
  digitalWrite(led1,!digitalRead(led1));      // Change the state of the LED
    led1_=!led1_;
  server.sendHeader("Location","/");       // Add a header to respond with a new location for the browser to go to
the home page again
  server.send(303);                        // Send it back to the browser with an HTTP status 303 (See Other) to redirect
}
void handleNotFound(){
  server.send(404, "text/plain", "404: Not found"); // Send HTTP status 404 (Not Found) when there's no handler
for the URI in the request
}
```

I give you the html code here on the right.

Its not so hard to understand how it works if you try it on your own and experimend a little bit.
Good luck!



```
<!DOCTYPE HTML>
<html>
<header>
<title>ESP8266 Webserver</title>
<h1>\"ESP8266 Web Server Control\"</h1>
</header>

<p>
<form action=\"/LED1\" method=\"POST\">
<input type=\"submit\" value=\"Toggle LED1\">
</form>
Led1: ON/OFF
</p>

<p>
<form action=\"/LED\" method=\"POST\">
<input type=\"submit\" value=\"Toggle LED\">
</form>
Led: ON/OFF
</p>

</html>
```

Tutorial by Christianidis Vasileios
email: baslisvirus@hotmail.com