**EMBEDDED SYSTEMS**

**(EL-419)**

**LABORATORY MANUAL**

**Spring 2019**

****

**Engr. Aneela Sabir**

**Introduction to ESP8266 using Arduino IDE**

**(LAB # 05)**

Student Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Roll No: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Section: \_\_\_\_

Date performed: \_\_\_\_\_\_\_\_\_\_\_\_\_, 2018

**MARKS AWARDED: \_\_\_\_\_\_\_\_/ 10**

**Lab # 05: Introduction to ESP8266 using Arduino IDE**

**Learning Objectives:**

1. Introduction to IoT
2. Wifi Modules
3. ESP8266 NodeMCU
4. Getting Started with NodeMcu using Arduion IDE
5. Running the Existing Projects
6. Controlling LED using Web server

#### **Equipment Required:**

1. NodeMCU ESP8266 Breakout Board
2. USB-A to micro-USB Cable
3. Jumper wires
4. LED
5. Bread board
6. Arduino IDE

**ESP8266 Introduction:**

**What is ESP8266?**

ESP8266 is a wifi SOC (system on a chip) produced by Espressif Systems. It is an highly integrated chip designed to provide full internet connectivity in a small package

**What is good for it?**

ESP8266 can be used as an external Wifi module, using the standard AT Command set Firmware by connecting it to any microcontroller using the serial UART, or directly serve as a Wifi-enabled micro controller, by programming a new firmware using the provided SDK.The GPIO pins allow Analog and Digital IO, plus PWM, SPI, I2C, etc.This board has been around for almost a year now, and has been used mostly in IoT contexts, where we want to add connectivity for example to an Arduino project. A wide adoption has been facilitated by the very modest price, ranging from 2.50 to 10 USD depending on the features offered by the manufacturers.

**Technical Features:**

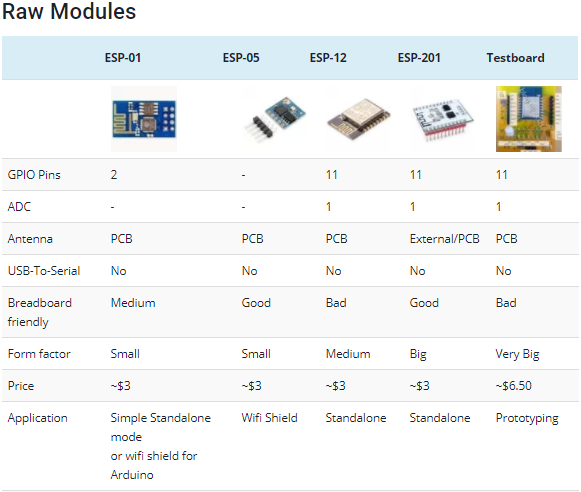
* 802.11 b / g / n
* Wi-Fi Direct (P2P), soft-AP
* Built-in TCP / IP protocol stack
* Built-in TR switch, balun, LNA, power amplifier and matching network
* Built-in PLL, voltage regulator and power management components
* 802.11b mode + 19.5dBm output power
* Built-in temperature sensor
* Support antenna diversity
* off leakage current is less than 10uA
* Built-in low-power 32-bit CPU: can double as an application processor
* SDIO 2.0, SPI, UART
* STBC, 1×1 MIMO, 2×1 MIMO
* A-MPDU, A-MSDU aggregation and the 0.4 Within wake
* 2ms, connect and transfer data packets
* standby power consumption of less than 1.0mW (DTIM3)

<http://fabacademy.org/archives/2015/doc/networking-esp8266.html>

**Varients:**

ESP8266-based boards are available from several vendors and with different breakout boards. Most of the boards differ in the number of pins made available, the amount of Flash memory for storing program and data, and the shielding on the SOC for certified boards.Some boards also support an external uFL antenna connector as well as the buil-in on-chip antenna.The most common variants are:

1. ESP-01
2. ESP-03
3. ESP-12
4. ESP-201



**Development Boards:**



<https://blog.squix.org/2015/03/esp8266-module-comparison-esp-01-esp-05.html>

**Software Development Kits:**

In late October 2014, Espressif Systems released a [software development kit](https://en.wikipedia.org/wiki/Software_development_kit) (SDK) that allowed the chip to be programmed, removing the need for a separate microcontroller Since then, there have been many official SDK releases from Espressif; Espressif maintains two versions of the SDK – one that is based on [FreeRTOS](https://en.wikipedia.org/wiki/FreeRTOS" \o "FreeRTOS) and the other based on callbacks.

An alternative to Espressif's official SDK is the open source ESP-Open-SD that is based on the [GCC](https://en.wikipedia.org/wiki/GNU_Compiler_Collection) toolchain.

Another alternative is the "Unofficial Development Kit" by Mikhail Grigorev.

Other SDKs (mostly open source) include:

* [NodeMCU](https://en.wikipedia.org/wiki/NodeMCU) – A Lua-based firmware.
* [Arduino](https://en.wikipedia.org/wiki/Arduino) – A C++ based firmware. This core enables the ESP8266 CPU and its Wi-Fi components to be programmed like any other Arduino device. The ESP8266 Arduino Core is available through GitHub.
* MicroPython – A port of [MicroPython](https://en.wikipedia.org/wiki/MicroPython) (an implementation of Python for embedded devices) to the ESP8266 platform.
* ESP8266 BASIC – An open source basic interpreter specifically tailored for the internet of things. Self hosting browser based development environment.
* Zbasic for ESP8266 – A subset of Microsoft's widely used Visual Basic 6 which has been adapted as a control language for the ZX microcontroller family and the ESP8266.
* Espruino – An actively maintained JavaScript SDK and firmware, closely emulating Node.js. Supports a few MCUs, including the ESP8266.
* [Mongoose OS](https://en.wikipedia.org/wiki/Mongoose_OS) – An open source Operating System for connected products. Supports ESP82666 and ESP32. Develop in C or JavaScript.
* ESP-Open-SDK – Free and open (as much as possible) integrated SDK for ESP8266/ESP8285 chips.
* ESP-Open-RTOS – Open source FreeRTOS-based ESP8266 software framework.
* Zerynth – [IoT](https://en.wikipedia.org/wiki/Internet_of_things) framework that allows programming ESP8266 and other microcontrollers using [Python](https://en.wikipedia.org/wiki/Python_(programming_language)).

<https://en.wikipedia.org/wiki/ESP8266>

**Introduction to NodeMCU ESP8266 module**



The Node MCU is an open source firmware and development kit that helps you to prototype your IoT product with ArduinoIDE. It includes firmware which runs on the ESP8266 Wi-Fi SoC. And hardware which is based on the ESP-12 module.

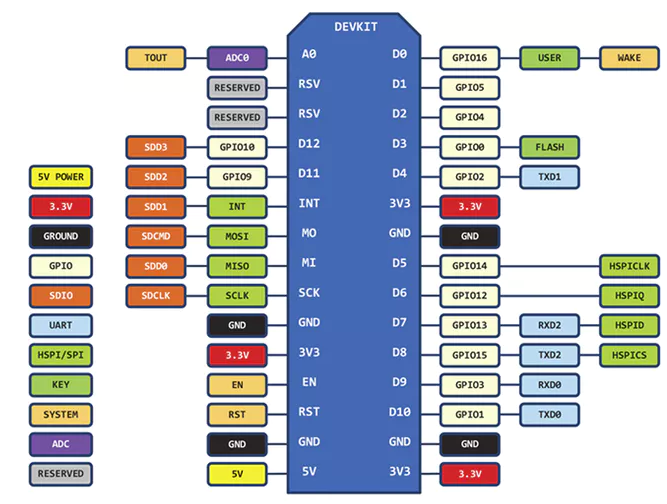
With just a few lines of code you can establish a WiFi connection and define input/output pins according to your needs exactly like arduino, turning your ESP8266 into a web server and a lot more. It is the WiFi equivalent of ethernet module. Now you have internet of things (iot) real tool.

With its USB-TTL , the NodeMCU Dev board supports directly flashing from USB port. It combines features of WIFI access point and station + microcontroller. These features   make the NodeMCU extremly powerful tool for Wifi networking. It can be used as access point and/or station, host a webserver or connect to internet to fetch or upload data.

**Features**

* Programmable WiFi module.
* Arduino-like (software defined) hardware IO.
* Can be programmed with the simple and powerful LUA programming language or Arduino IDE.
* USB-TTL included, plug & play.
* 10 GPIOs D0-D10
* PWM functionality
* I2C and SPI communication
* 1-Wire and ADC A0
* Wifi networking (can be used as access point and/or station, host a web server), connect to internet to fetch or upload data.
* Event-driven API for network applications.
* PCB antenna.

**NodeMCU ESP8266 Module Pins details:**



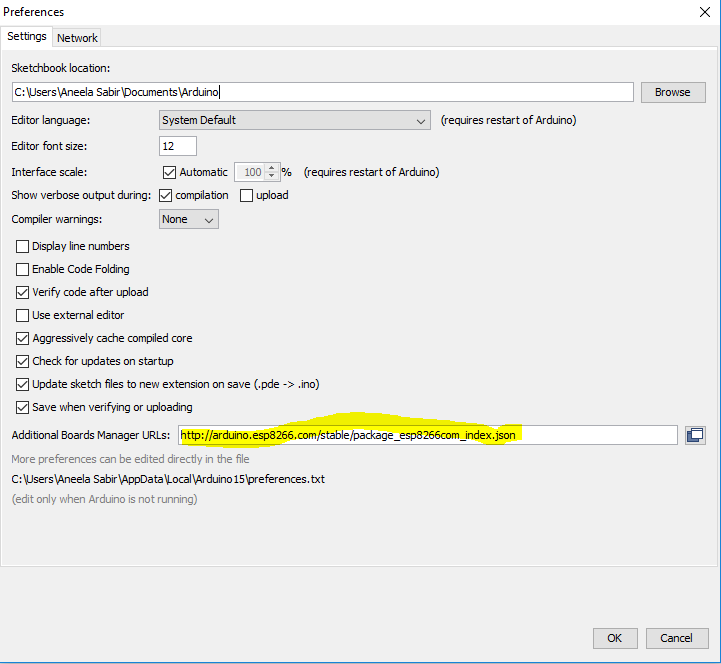
**Getting started with NodeMCU ESP8266 module using Arduino IDE**

First of all connect the NodeMCU ESP8266 module to your PC or laptop using the USB cable provided with it.

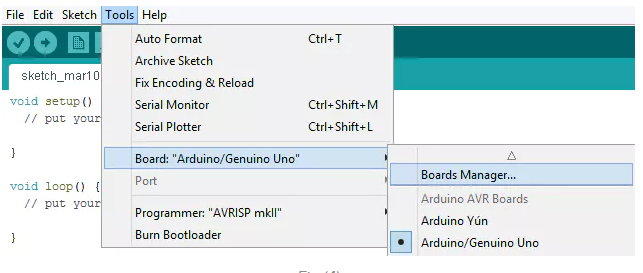
Your ArduinoIDE needs to know that you want to use this NodeMcu module. That is, you need to select the right board. For this, please follow the steps given below:

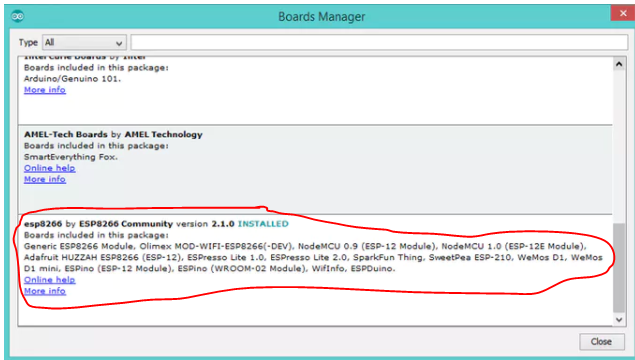
**Step 1:**Copy the given link into your preferences as shown in figure below:

<http://arduino.esp8266.com/stable/package_esp8266com_index.json>



**Step 2:** Install ESP8266 board in Arduino IDE.

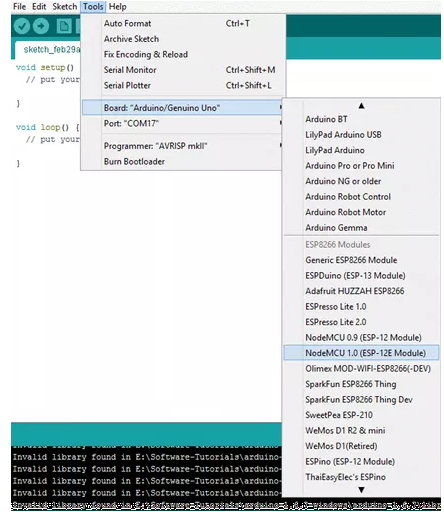




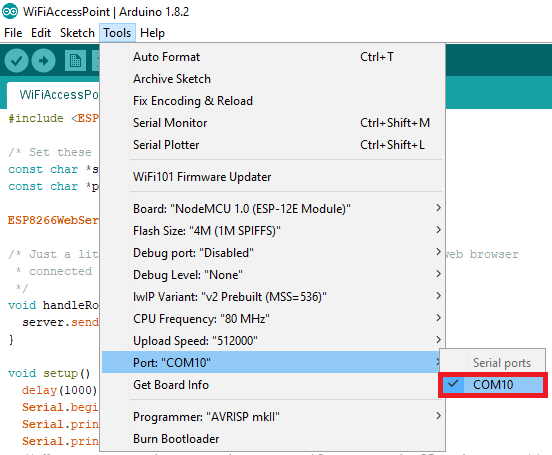
Note: If you are unable to install these libraries online, then copy the libraries provided you on SLATE and paste the folder into your Arduino packages at the given link:

**C:\Users\AppData\Local\Arduino15\**

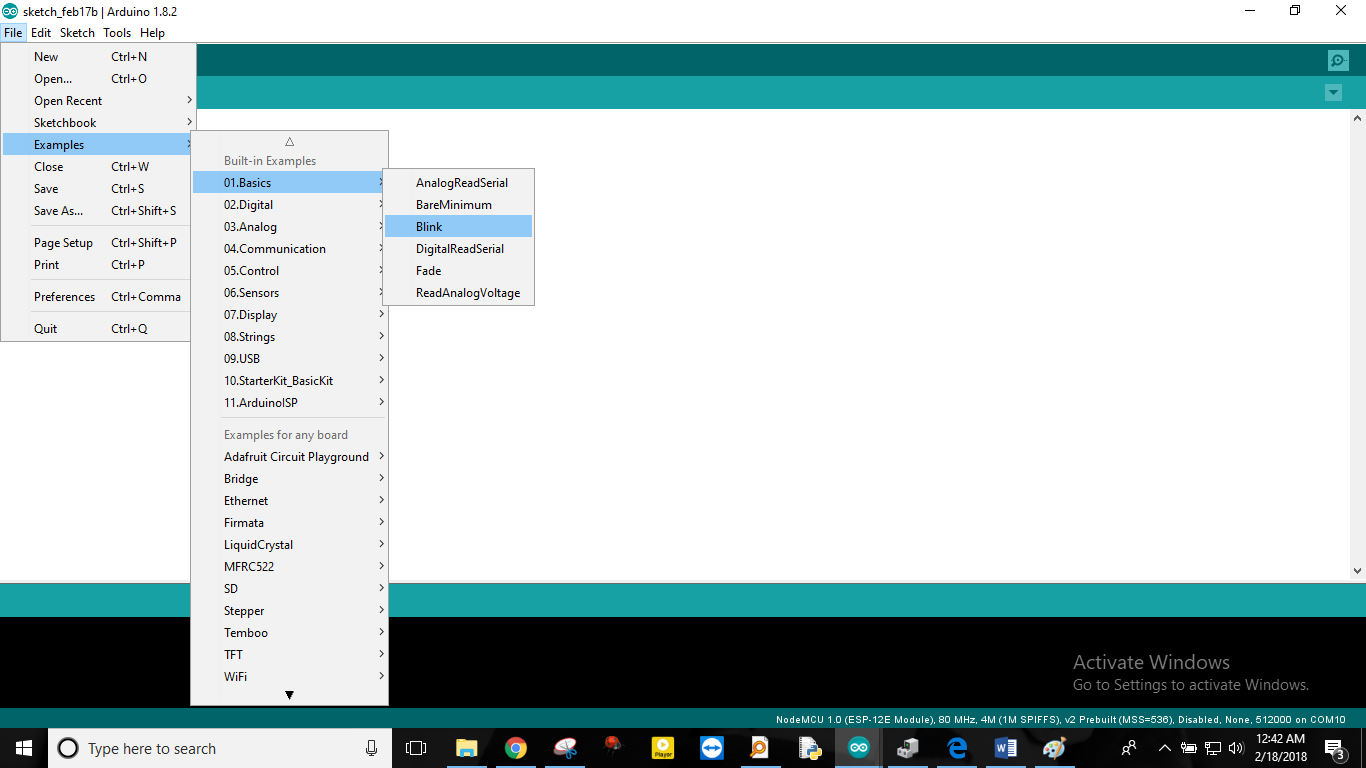
**Step 3:**After successfully installing ESP8266 library, select NodeMCU 1.0 board.



**Step 4:** Select port to upload program using Arduino IDE.



**Step 5: Running Example code**



**Example 1: LED Blinking**

// the setup function runs once when you press reset or power the board

void setup() {

// initialize digital pin LED\_BUILTIN as an output.

pinMode(LED\_BUILTIN, OUTPUT);

}

// the loop function runs over and over again forever

void loop() {

digitalWrite(LED\_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)

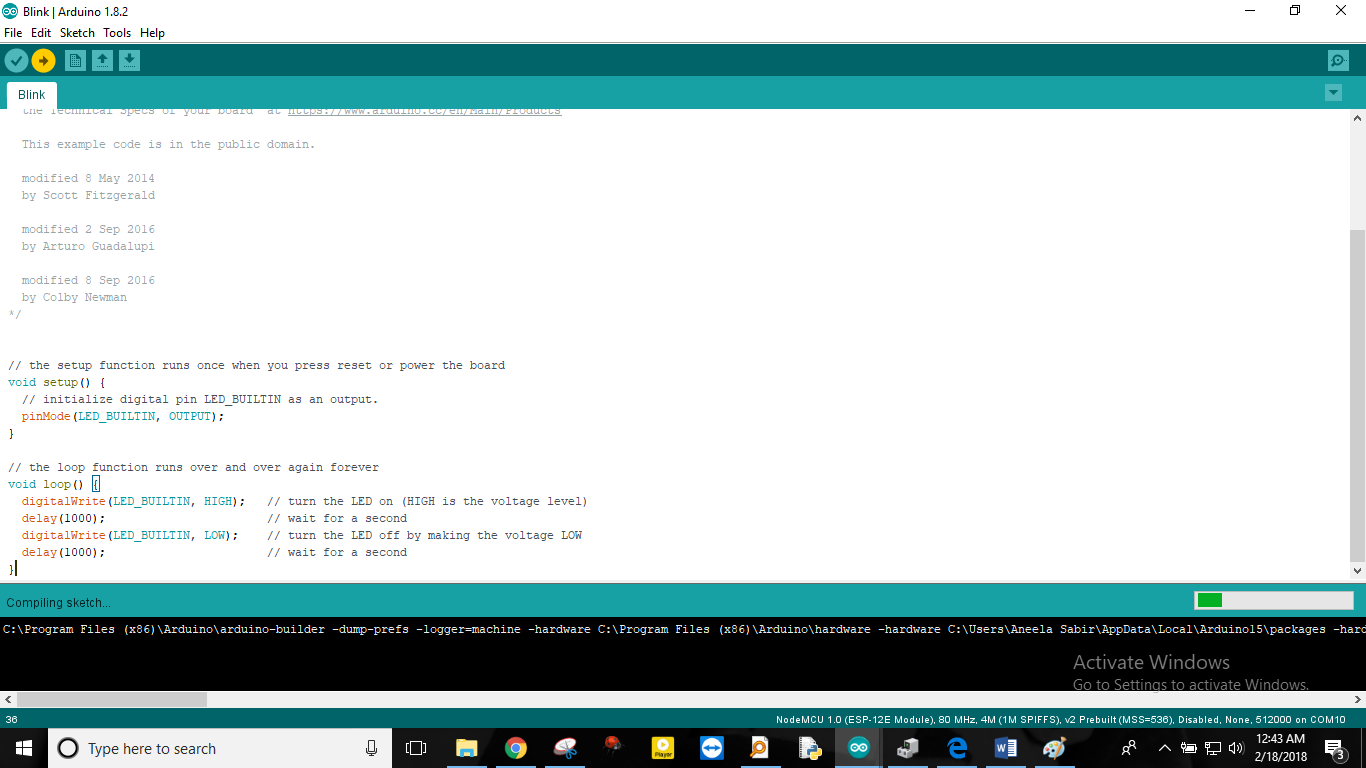
delay(1000); // wait for a second

digitalWrite(LED\_BUILTIN, LOW); // turn the LED off by making the voltage LOW

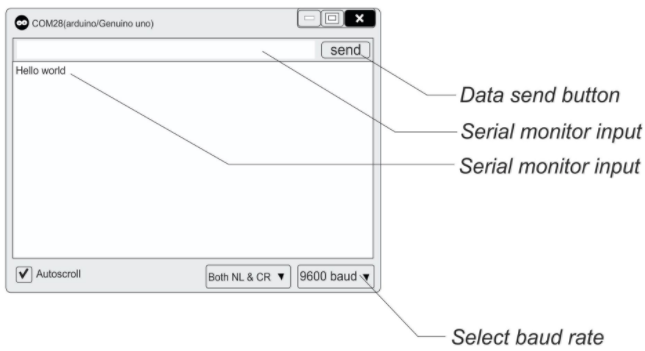
delay(1000); // wait for a second

}

**Step 6:** Upload program to NodeMCU development board.



## Example 2:**NodeMCU Serial Monitor on Arduino IDE**

void setup() {

**Serial**.begin(9600);

**Serial**.print("Hello World");

}

void loop() {

}

**Output:**Upload the code open the serial monitor and select 9600 baud rate. As the serial monitor opens “Hello World” will be seen on output panel.

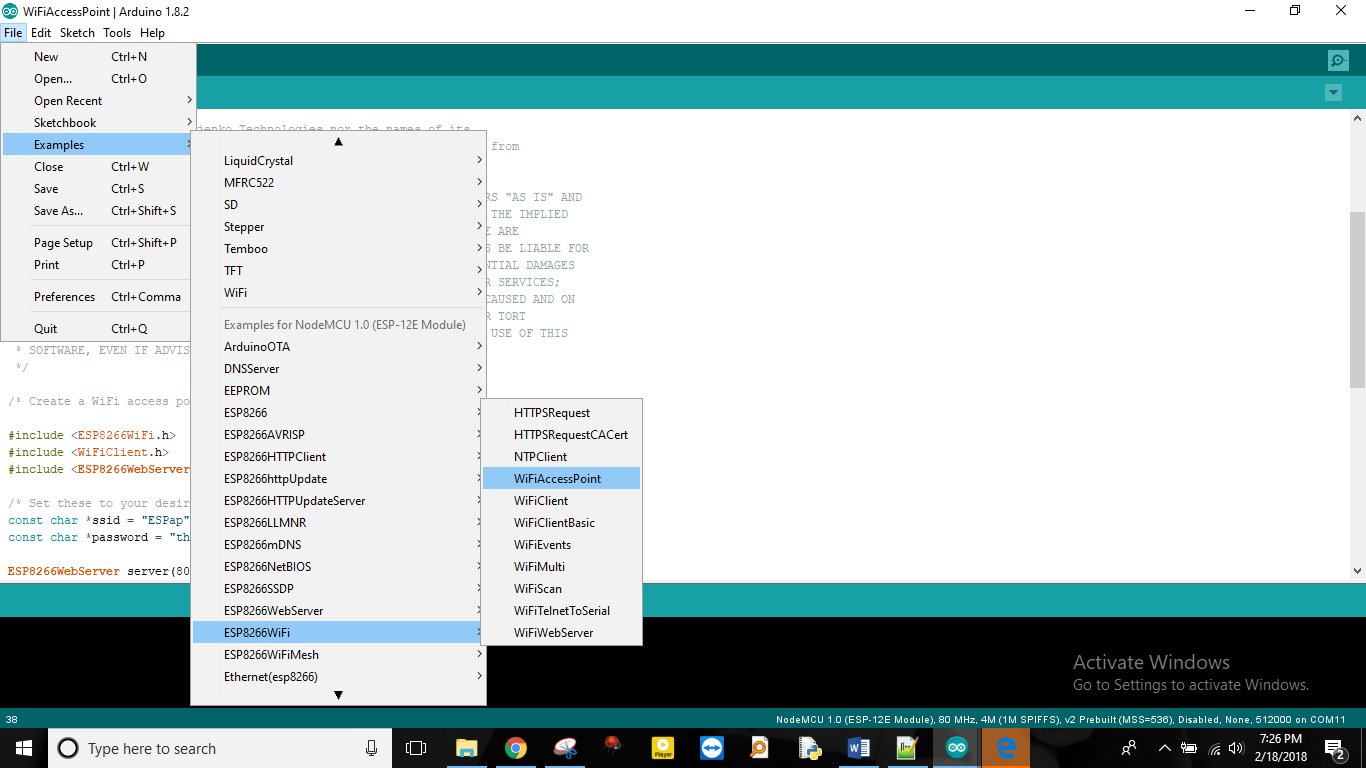
**Task 01: Run Example 1 and 2 on ESP8266.**

**Access Point (AP) and Station (STA)**

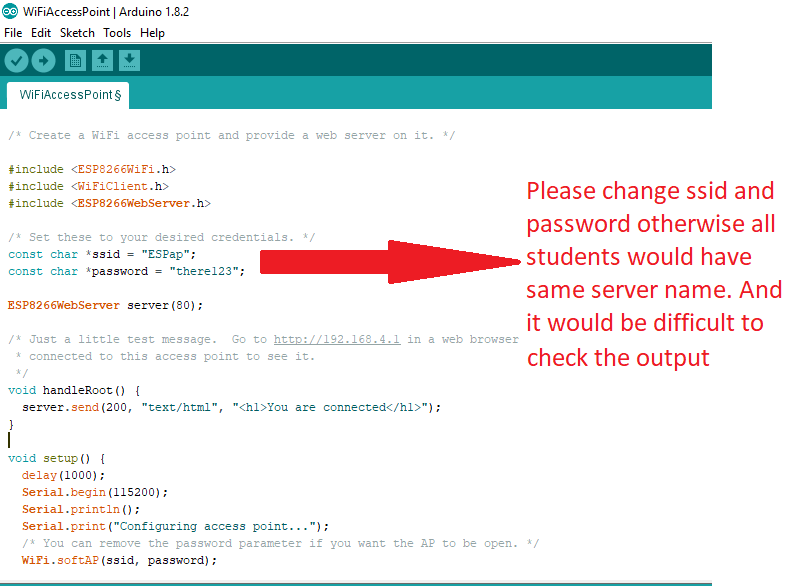
Once you start working with ESP module, you would come across these two terms frequently. Let us say you and your friend would like to surf the internet on your smart phones but since he does not have an active internet connection you decide to turn on your hotspot and your friend connects to it. Here your phone which is sourcing the internet connection is the Access Point (AP) and your friend’s phone which is using the internet is called the Station (STA).

ESP8266 module can be used in three modes, AP mode, STA mode or in both STA and AP mode (combined).

## ****ESP8266 as an Access Point:****



**Example 4. Wifi Access Point:**

****

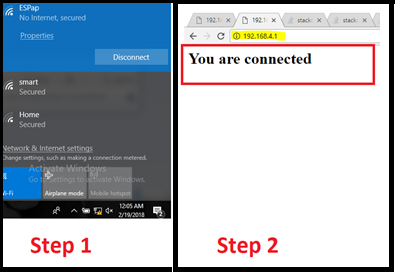
**Service Set Identifier (SSID):**

This is fairly a simple term. Almost all of us have used WIFI. The name of the Wi-Fi Network is called its SSID. When we have multiple access points for a station to connect to, the station should know which access point it should get connected, hence each Access Point (AP) is given an identity which is called the SSID.

**Verifying that ESP8266 module has been established as AP:**

Step 1- Connect to ESPap using password provided in the code.

Step 2- Go to Web browser and Enter 192.168.4.1. You will see a status “You are connected”.

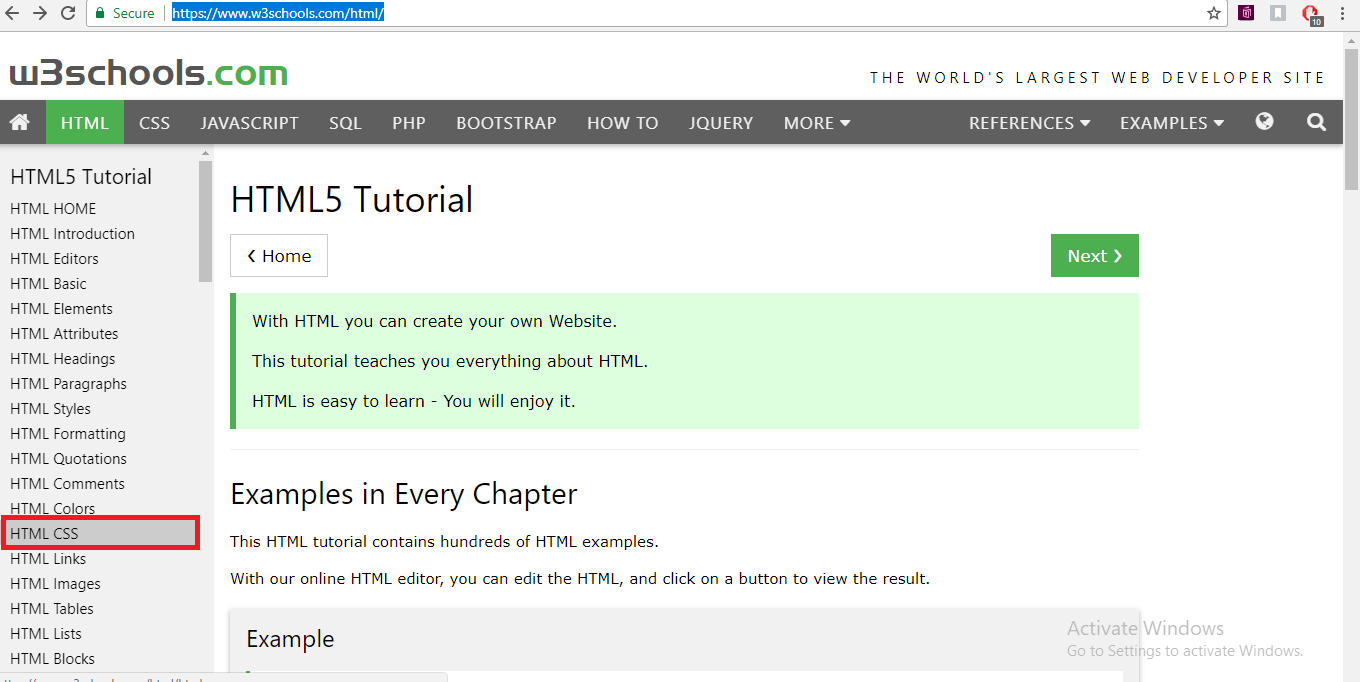
****

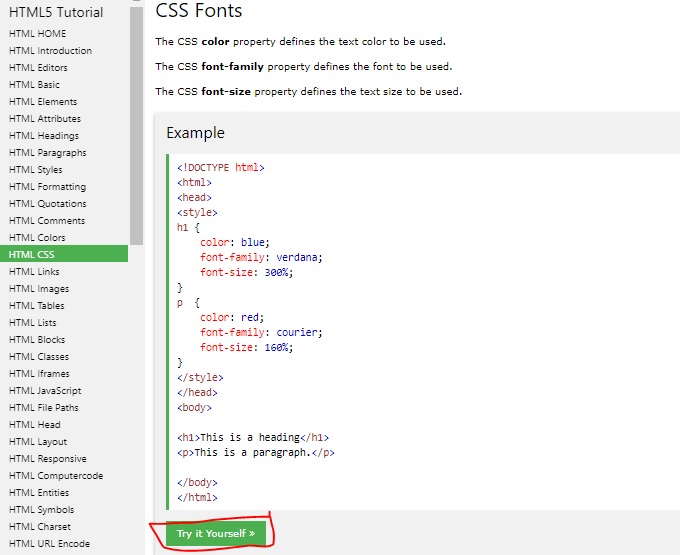
**Task 02: Run Wife Access Point example code. Upload it to your NodeMCU, and verify the results.**

**Task 03: Search example code to make ESP8266 module as a station. Observe the output and paste serial monitor screenshots here.**

Hint: you need to change ssid and password in example code. See ESP8266 Web Server based example.

**Learn to Add Simple HTML in Arduino Sketch**

1. Goto <https://www.w3schools.com/html/>
2. Select HTML CSS
3. Try an example code as shown in figure below:



1. Modify the highlighted code:



**Modified Code:**

<!DOCTYPE html>

<html>

<head>

<style>

h1 {

color: blue;

font-family: verdana;

font-size: 300%;

}

p {

color: red;

font-family: courier;

font-size: 160%;

}

</style>

</head>

<body>

<h1>Hello from ESP</h1>

<p>This is my first lab based on ESP8266.</p>

<p>Created by Aneela Sabir</p>

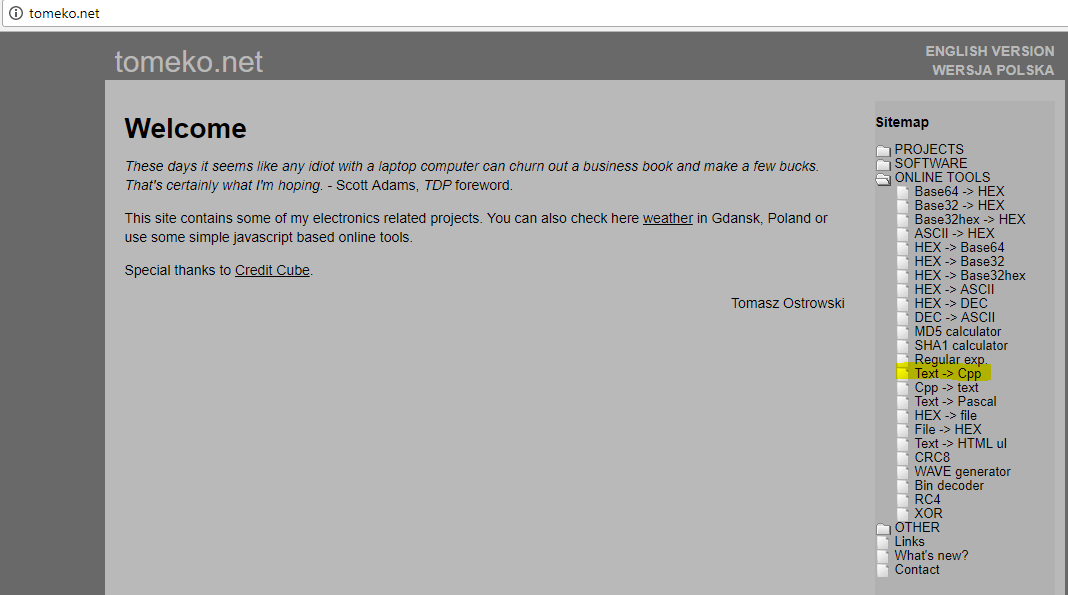
</body>

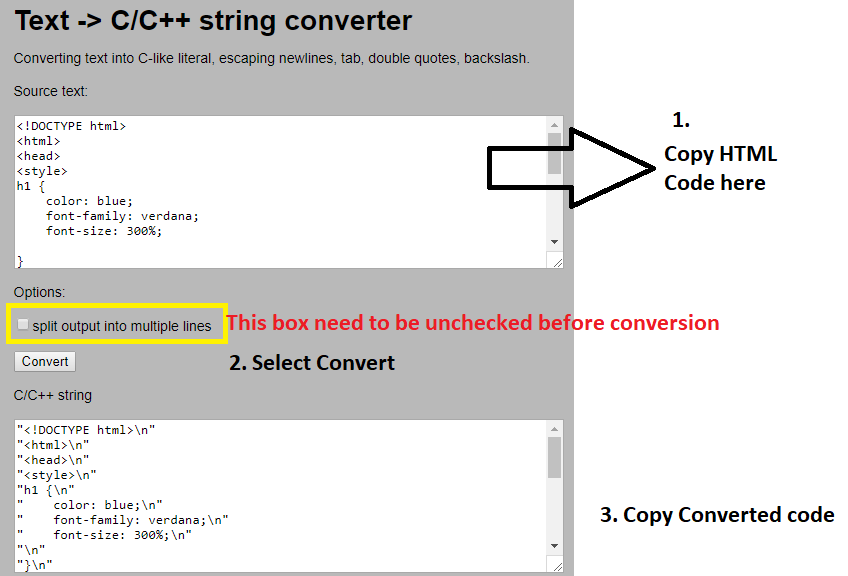
</html>



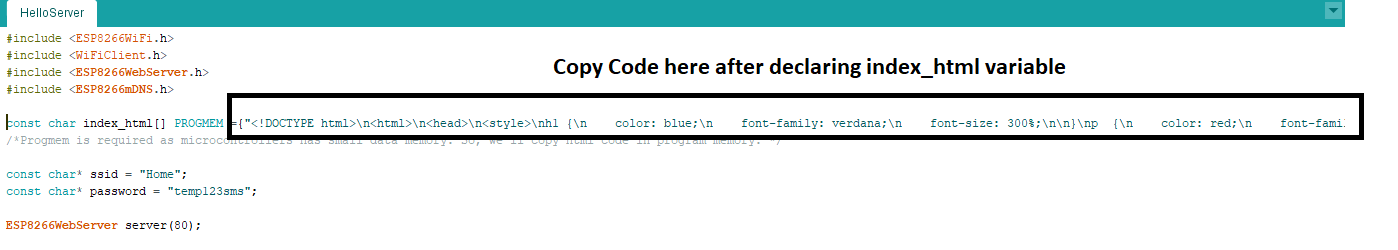
1. **Convert HTML to C-String Format:**

To convert HTML to C-string you can use online tool/software for this conversion. One such example is <http://tomeko.net/>.



****

1. Open HeloWerver Example in Arduino IDE. You need to paste HTML to C converted code in Arduino IDE.

****

1. After this you need to modify handleRoot Function:

void handleRoot() {

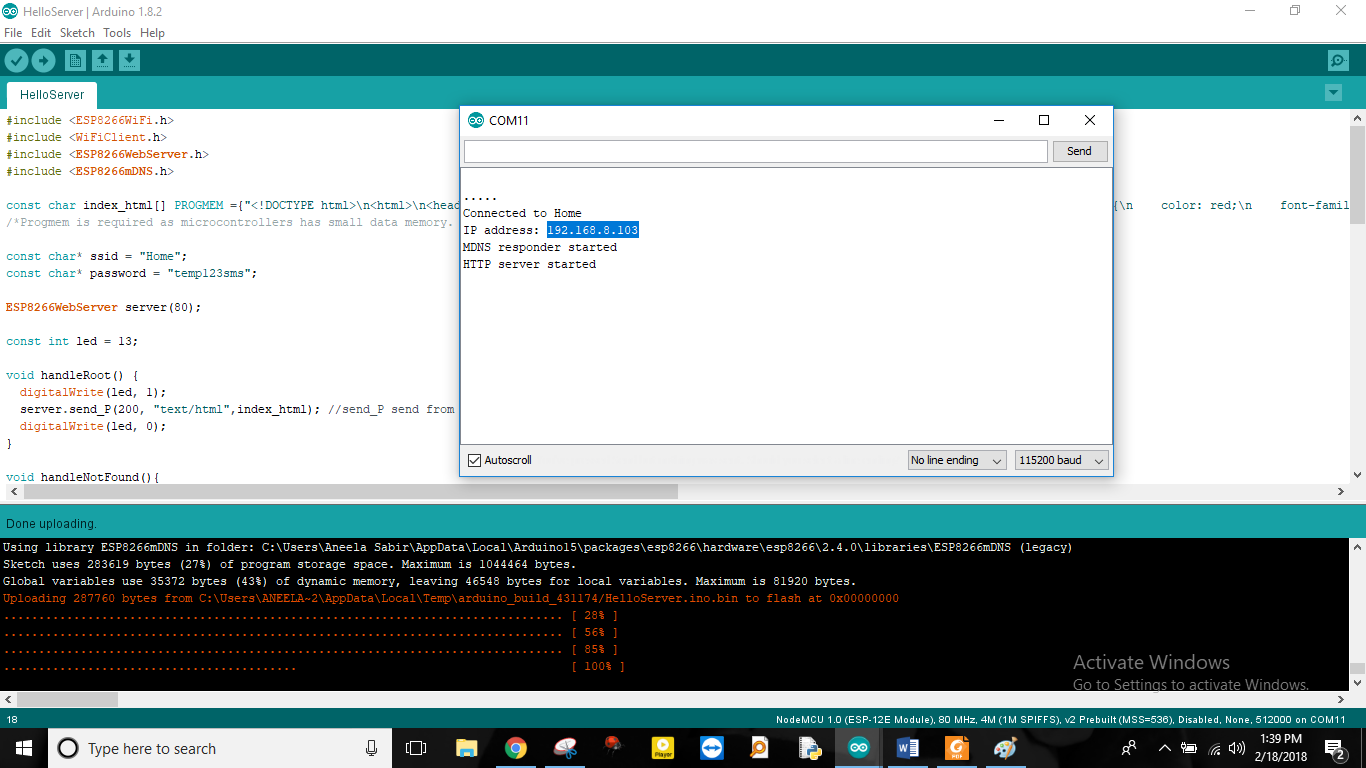
digitalWrite(led, 1);

**server.send(200, "text/html",index\_html);**

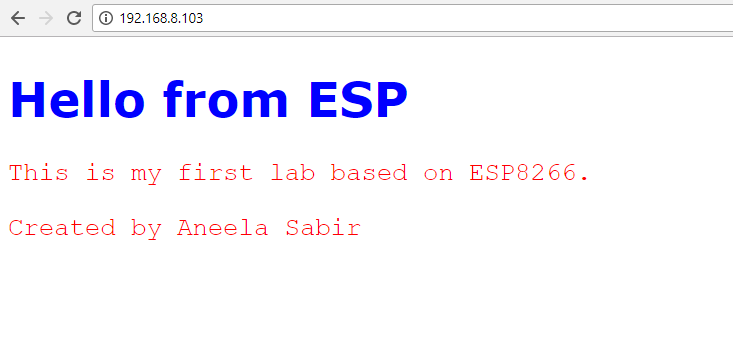
digitalWrite(led, 0);

}

1. Compile and Upload updated Code:



**Output:** Open IP Address in Web browser.



**Task 04: Search 2 more HTML code examples, convert to C string format and upload it to ESP8266 module.**

[**ESP8266 Webserver: Controlling an LED through WiFi**](https://techtutorialsx.com/2016/11/19/esp8266-webserver-controlling-a-led-through-wifi/)

The objective is to control an LED through WiFi, using the ESP8266. You need to download Code (named Control LED through Wifi) from SLATE. Details of the code are given in next section.

**Setup code:**

We start by including the libraries needed to connect the ESP8266 to a WiFi network and to set the HTTP server.

#include <ESP8266WiFi.h>  
#include <ESP8266WebServer.h>

Next, we declare a global object variable from the ESP8266WebServer class. This class will have the methods needed to set the HTTP server.

ESP8266WebServer server(80);

We passed 80 for the constructor, which indicates that the server will be listening on port 80. We will also declare a global variable to specify the LED pin and another for its state (on or off).

#define ledPin 2  
bool ledState = LOW;

There is no problem in initializing a Boolean variable with LOW because it will correspond to “0”.

We need to specify that the pin will be an output pin. We do this in the setup function.

pinMode(ledPin, OUTPUT);

Also in the setup function, we will specify the URLs where our server will be listening to incoming HTTP requests. We will define 3 URLs, one for each operation on the LED: turn on, turn off and toggle.

server.on(“/on”, turnOn);          //Associate the handler function to the path  
server.on(“/off”, turnOff);          //Associate the handler function to the path  
server.on(“/toggle”, toggle);     //Associate the handler function to the path

The first argument of the **on** method specifies the path where the server should listen, and the second argument specifies the name of the handling function that will be executed by the ESP8266 when a request to the corresponding path is received. We will specify those functions latter.

Bellow is the complete setup function. We start by connecting to the WiFi network, doing some prints to the serial port indicating if the connection was successful and we declare the paths of our HTTP server. We finalize the setup of our server by calling the **begin** method on the server object.

void setup() {

pinMode(ledPin, OUTPUT);

Serial.begin(115200);  
WiFi.begin(“YourNetworkName”, “YourPassword”); //Connect to the WiFi network

while (WiFi.status() != WL\_CONNECTED) { //Wait for connection

delay(500);  
Serial.println(“Waiting to connect…”);

}

Serial.print(“IP address: “);  
Serial.println(WiFi.localIP()); //Print the local IP

server.on(“/on”, turnOn);         //Associate the handler function to the path  
server.on(“/off”, turnOff);        //Associate the handler function to the path  
server.on(“/toggle”, toggle);   //Associate the handler function to the path

server.begin(); //Start the server  
Serial.println(“Server listening”);

}

To handle the actual incoming of HTTP requests, we need to call the **handleClient** method on the server object, on the main loop function.

void loop() {

server.handleClient();

}

## ****The handling functions****

For the turnOn handler function, we will start by setting the ledState variable to HIGH, since we want to turn on the LED.

ledState = HIGH;

We then write the state of the LED with the [digitalWrite](https://www.arduino.cc/en/Reference/DigitalWrite) function.

digitalWrite(ledPin, ledState);

In the last line of code of the handler function, we will just send the HTTP response, so the client knows the action was performed. The first argument of the function is the HTTP response code, the second the type of content of the response, and the third the actual response.

server.send(200, “text/plain”, “LED on”);

You can check the full handling function bellow.

void turnOn(){

ledState = HIGH;  
digitalWrite(ledPin, ledState);  
server.send(200, “text/plain”, “LED on”);

}

The handler for the turnOff URL is exactly the same, except for the fact that we will set the ledState variable to LOW.

void turnOff(){

ledState = LOW;  
digitalWrite(ledPin, ledState);  
server.send(200, “text/plain”, “LED off”);

}

The toggling handling function is also similar, except that we apply the [NOT operator](https://www.arduino.cc/en/Reference/Boolean) to the ledState variable and assign it to itself.

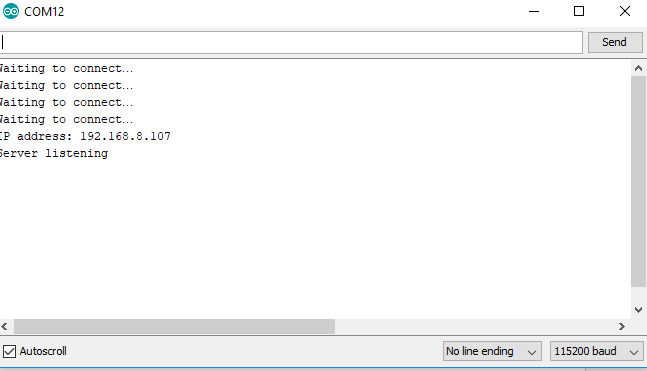
void toggle(){

ledState = !ledState;  
digitalWrite(ledPin, ledState);  
server.send(200, “text/plain”, “LED toggled”);

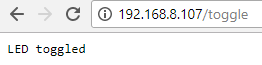
}

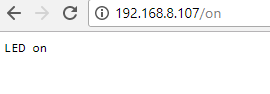
## ****Testing the code****

Serial Monitor:



Webpage:



**Task 05: Write code to control the status LED through Wifi using ESP8266.**