Getting Started with the ESP8266 and the Arduino IDE

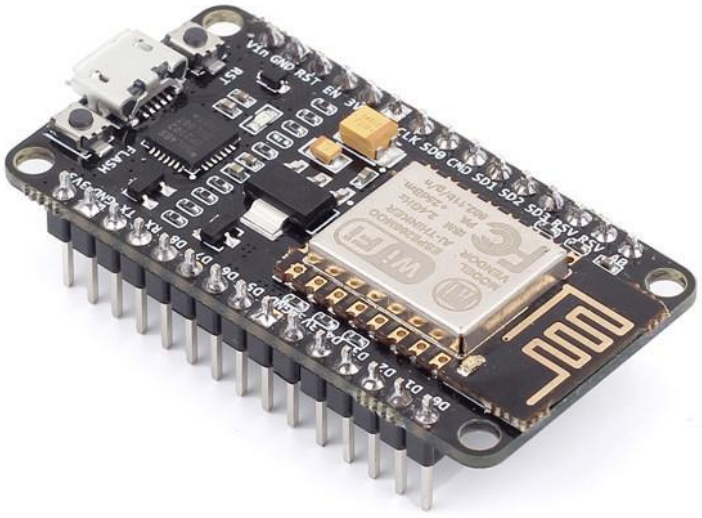
Welcome to the wonderful world of the Internet-of-Everything. And you are not going to just stand on the sidelines, but you're going to be right in the middle of it! The heart of any IOT project is three key items; first, some small hardware that can take external inputs and control external hardware, second, a connection to the internet, normally through a wireless link so that this hardware can be in a remote location, and third, a control program that can run on a host processor that can both monitor and also control the your remote hardware.

So let's go through the steps to show you how to use one flavor of remote hardware, in this case the ESP8266 NodeMCU variant, to create a simple system that can be controlled remotely by an external control processor.

Selecting the Hardware

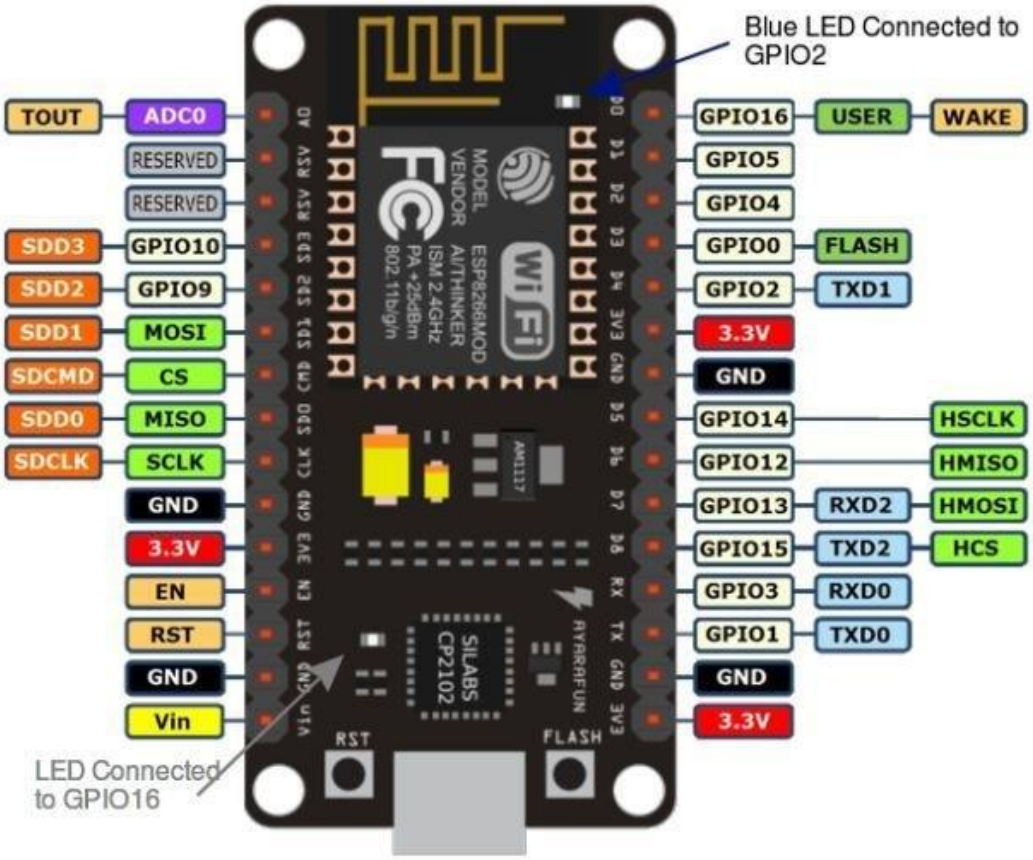
Since the introduction of the Arduino small processor ecosystem in 2003 many thousands and perhaps millions of developers have been able to easily and inexpensively learn and develop small microprocessor controlled projects. Arduino comes in many different flavors, including many that have different form factors and different processing capabilities. For more information go to www.arduino.cc and you can find out more about the diverse set of capabilities.

In this particular case you will be focusing on using the variant that normally goes by the name ESP8266 NodeMCU. The ESP8266 refers to a SOC (System On a Chip) that leverages the ease-of-use of the Arduino environment with a powerful system of both processing and wireless-lan capability. Here is a picture of one variant of the product.



You'll notice the micro-usb connector on the ESP8266 NodeMCU. This is used to communicate with the product to develop your application code. You'll also notice the antenna on the product, this is used to connect to the wireless-lan.

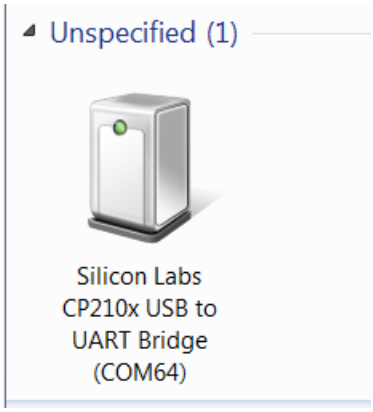
You'll also notice a number of labeled pins on the product. These are used to connect the product with the outside world. Since you'll almost certainly want to connect to the outside world, here is a detailed pin out of those connections:



You'll find out more detail later about the specifics on these pins throughout this guide. To get started, however, you'll want to plug your device into your host computer. This is as easy as plugging a USB cable into the ESP8266 NodeMCU and then into your host development machine.

If you are using a newer version of Microsoft Windows when you plug the ESP8266 NodeMCU into the system, it will try to install the drivers automatically. If the device fails to install, you may have to tell it where the drivers are. You will know if this

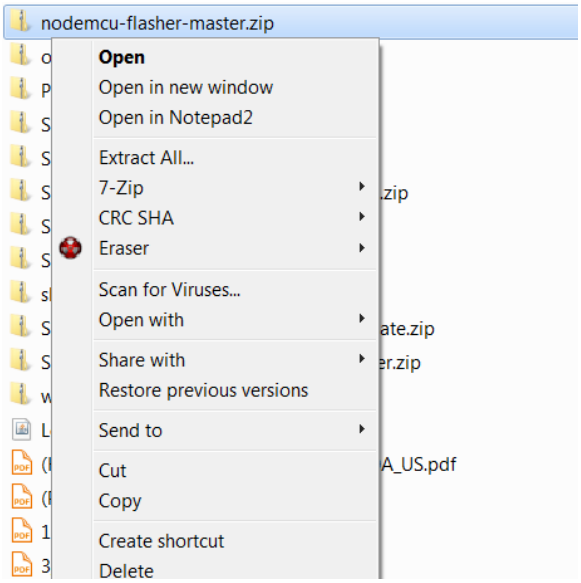
happens as you will get an error message saying **Device driver software was not successfully installed**. When your drivers are installed, you should see the following device when you navigate to **Start Menu** | **Devices and Printers**:



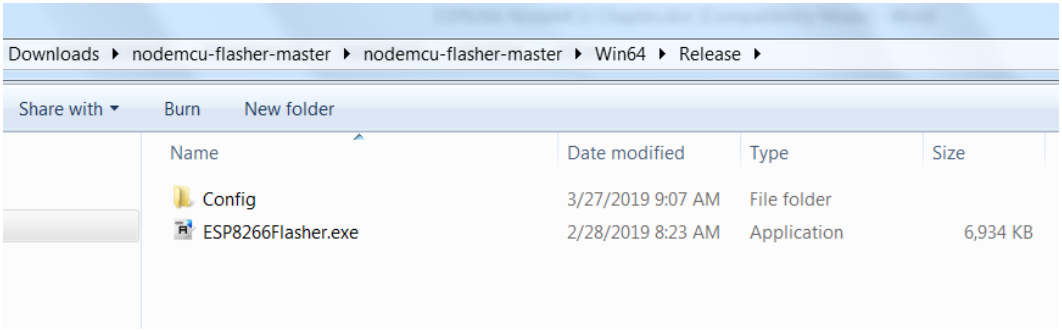
In this case, the device is connected to COM port 64. Note the COM port the ESP8266 is connected to as you'll need that in a minute. I f you are using an Apple Mac or Linux machine, follow the instructions at `arduino.cc/en/Guide/MacOSX` for the Mac and `playground.arduino.cc/Learning/Linux` for Linux on how to determine your USB port connection.

Flashing the ESP8266 NodeMCU with the Latest Firmware

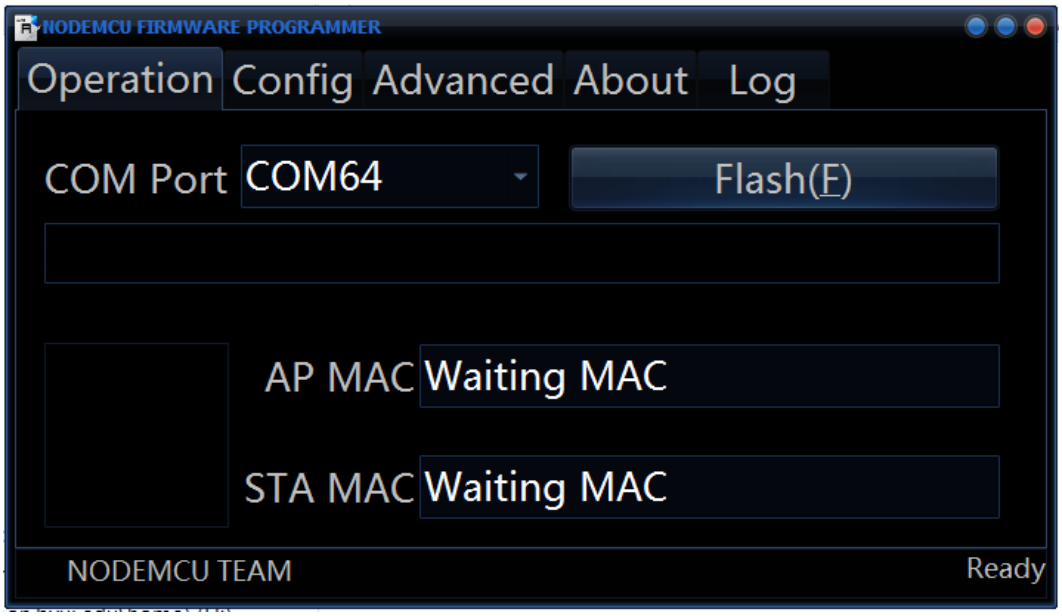
Before you can developing the code for your ESP8266 NodeMCU you'll want to flash the latest firmware. To do this first download the zip files from https://github.com/nodemcu/nodemcu-flasher and unzip the files. Then extract these files using Extract All…



This will create a set of directories. Go to the nodemcu-flasher-master->nodemcu-flasher-master-> Win64-> Release directory



And run the ESP8266Flasher.exe program. It should look like this:



Make sure the COM Port is set to the port you noted above. Then simply press the Flash(F) button, and your device will be flashed. You'll only need to do this once.

If you wish to use an Apple Mac or Linux machine to develop, you can use a windows machine to flash your device as you'll only need to do this once.

Introducing the Arduino IDE

Now that you have physically connected the ESP8266 NodeMCU to your host machine and flashed the device, you are ready to start the **Integrated Development Environment** (**IDE**). In this chapter, I'll start by covering how to use the IDE in Windows. Then, I'll cover any specific changes you might need to make if you are using a Mac machine.

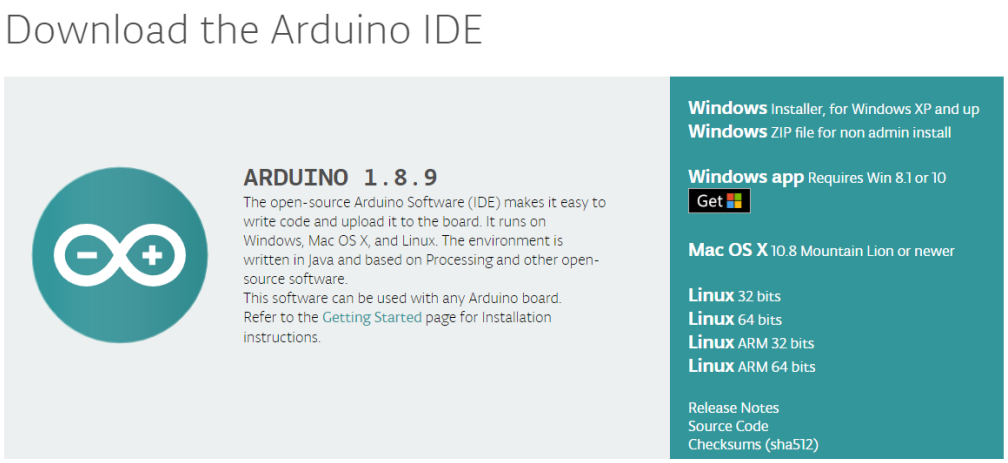
For this chapter, the objectives are as follows:

- Load and configure the Arduino IDE
- Download and run a simple example program

Using a Windows machine to develop with the ESP8266 NodeMCU

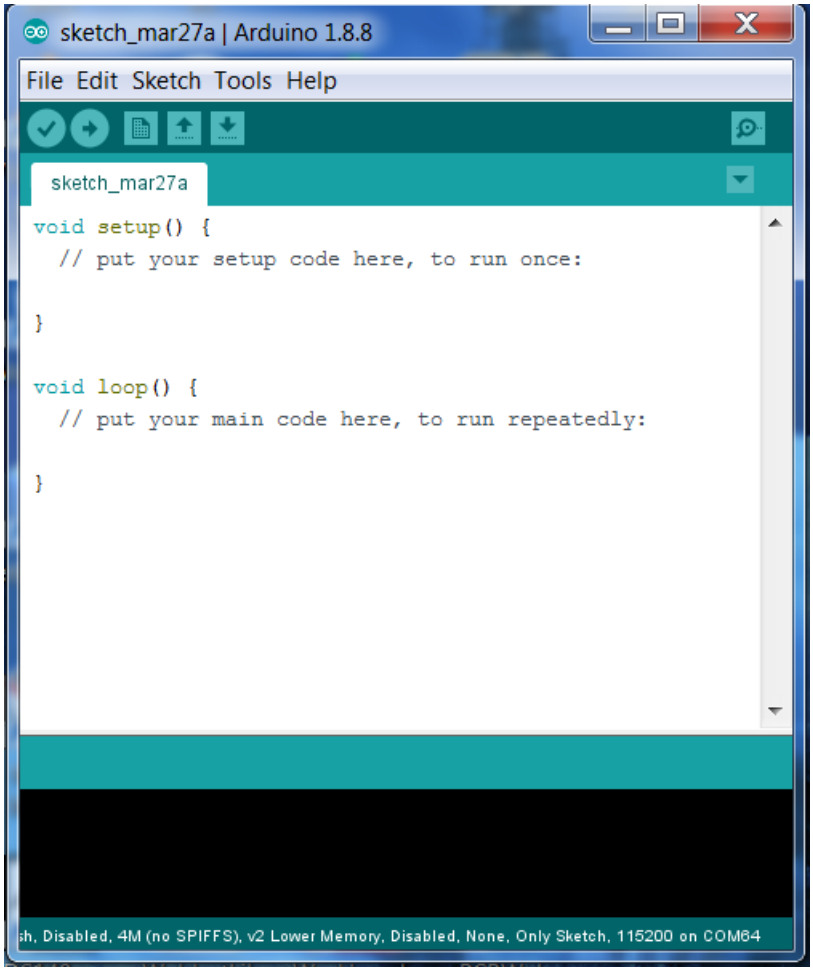
Running the IDE for the ESP8266 NodeMCU

Now that the device is installed you can run the IDE. Go to the www.arduino.cc website. Select the Software tab at the top of the web page. Select the Downloads selection. You should see a page that will allow you to either run the code from a web browser or to download the IDE. In this case you'll want to download the IDE. To do this go the section of the page that allows Downloads, and select the correct Download for your machine:



You can either just download the installer, or you can contribute and download the installer. Then run the installer from the Downloads directory. Once you have run the installer you should be able to run the IDE from the desktop or start menu.

When the IDE starts you should see something like the following screenshot:



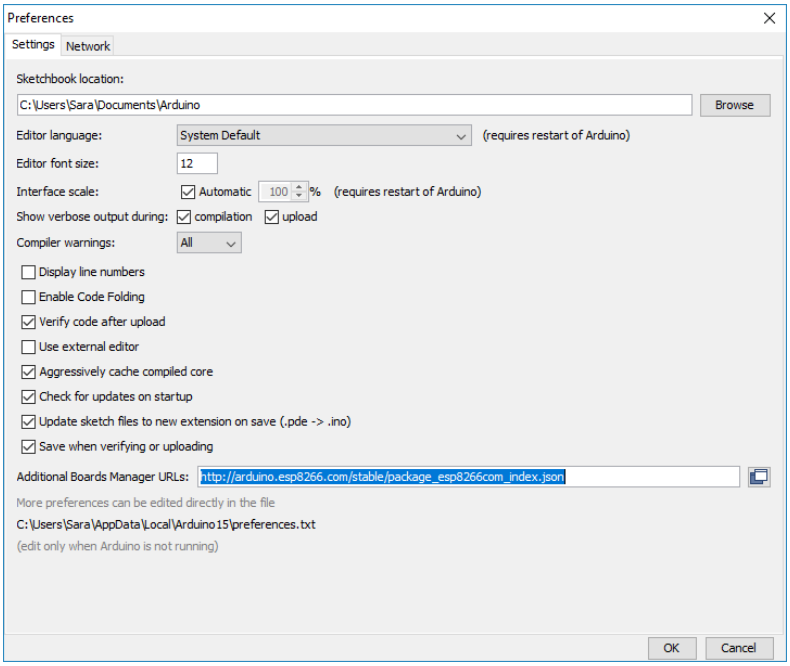
This is the environment you will use to develop your applications. The IDE will make it easy to compile the code, upload it to the device, and then run it.

Setting the IDE to your board

First, you'll need to add support for the ESP8266 NodeMCU on the Arduino IDE. To install the ESP8266 board in your Arduino IDE, follow these next instructions:

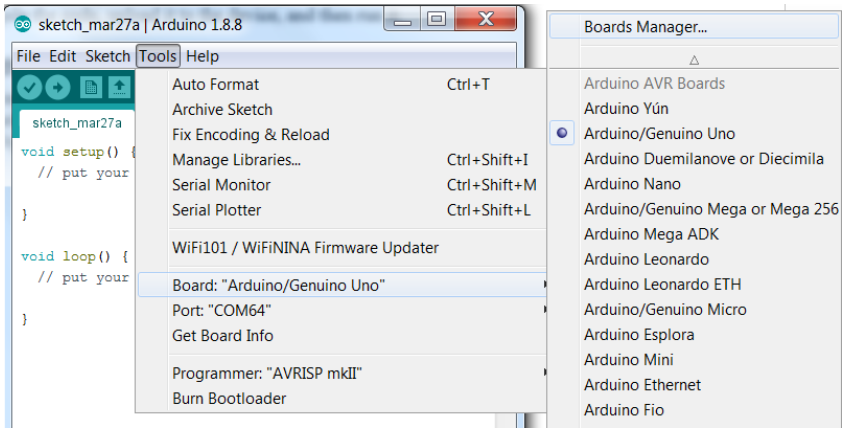
1) Open the preferences window from the Arduino IDE. Go to File > Preferences

2) Enter http://arduino.esp8266.com/stable/package_esp8266com_index.json into the "Additional Board Manager URLs" field as shown in the figure below. Then, click the "OK" button.

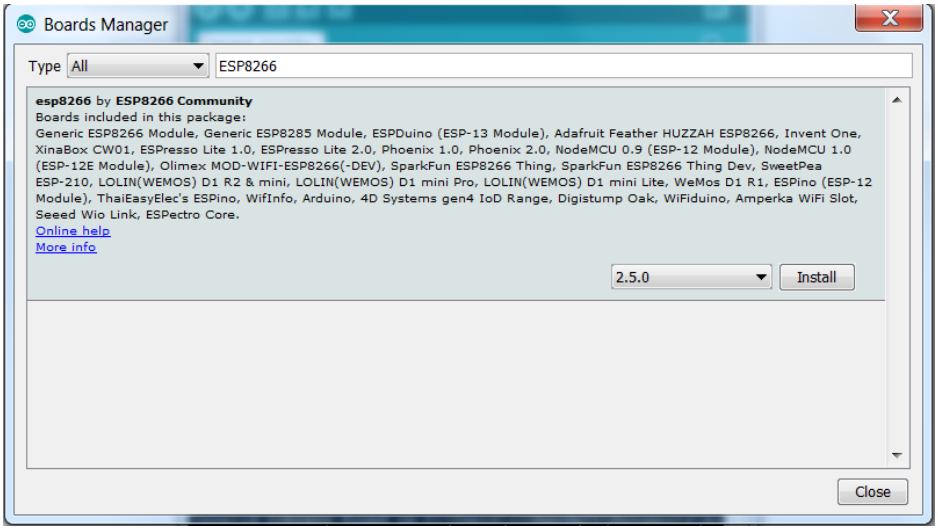


You'll then need to set the IDE to create code for the proper processor as different Arduino boards have slightly different hardware configurations. Fortunately, the IDE lets you set that by choosing the correct board.

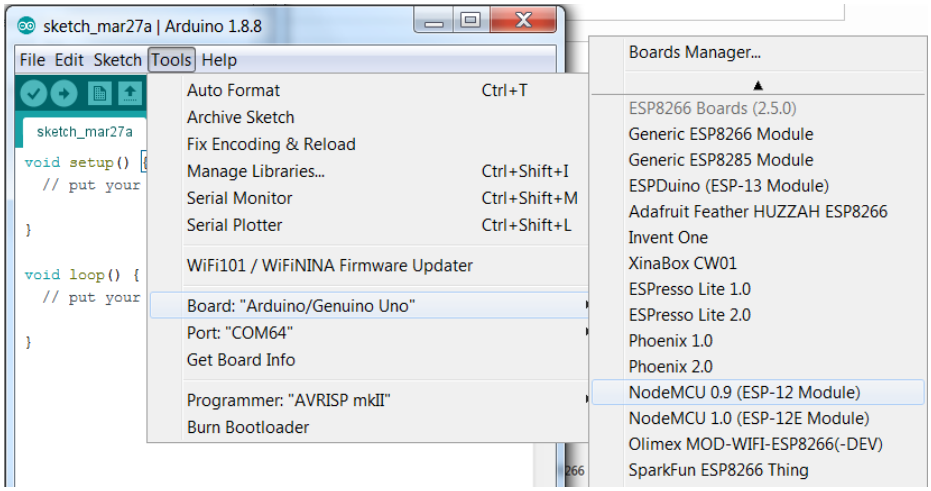
Before you can select your specific board you'll need to add it to the hardware support list. To do this select Tools | Board "Arduino/Genuino Uno" | Boards Manager…



This will bring up the Boards Manager. Now enter ESP8266 in the search selection, and you should see esp8266 by ESP8266 Community. Click the Install button.

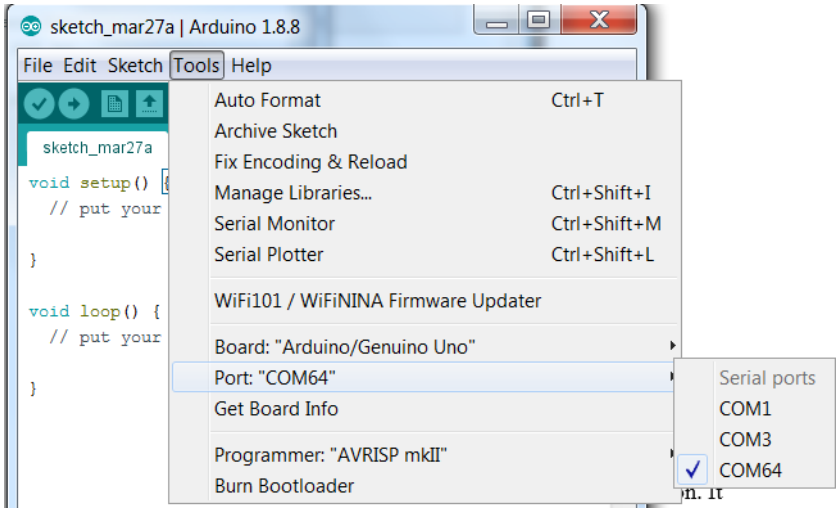


Now you can select the board. To do this, navigate to **Tools** | **Board** | NodeMCU 0.9 (ESP-12 Module), as shown in the following screenshot:



Selecting the proper COM port

The next step is to select the proper COM port. To do this, navigate to **Tools** | **Serial Port** | **COM64**, (the port you noted earlier) as shown in the following screenshot:



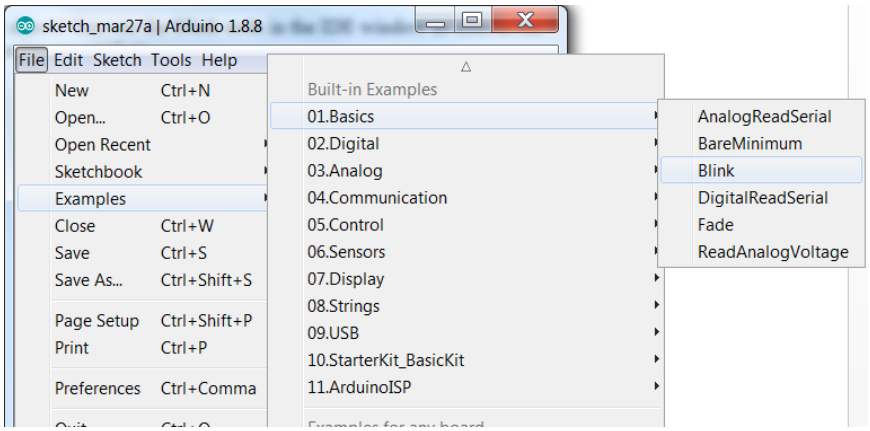
The IDE should now indicate that you are using the ESP8266 NodeMCU on COM23 in the lower-right corner of the IDE:

Opening and uploading a file to the ESP8266 NodeMCU

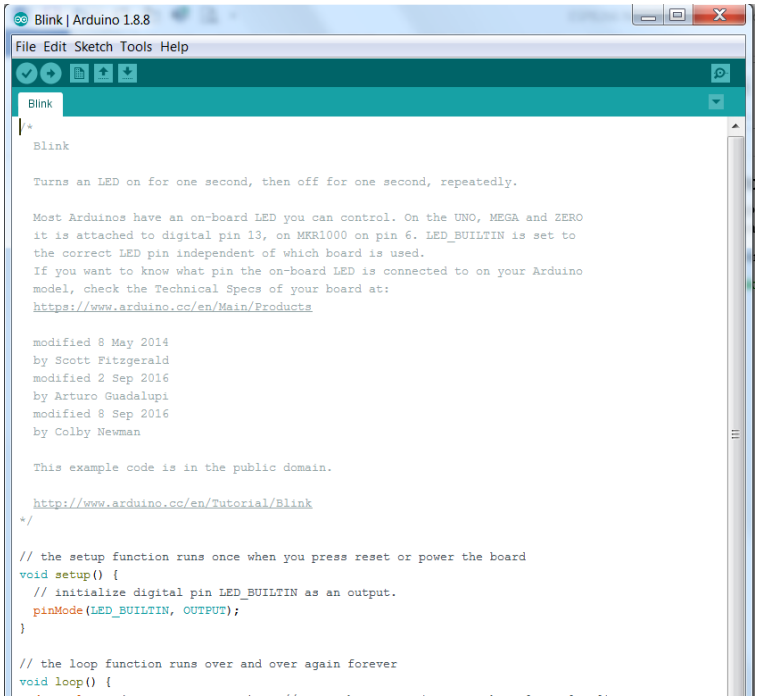
Now you can open and upload a simple example file. It is called the Blink application. It has already been written for you, so you'll won't need to do any coding.

To get a blink application, perform the following steps:

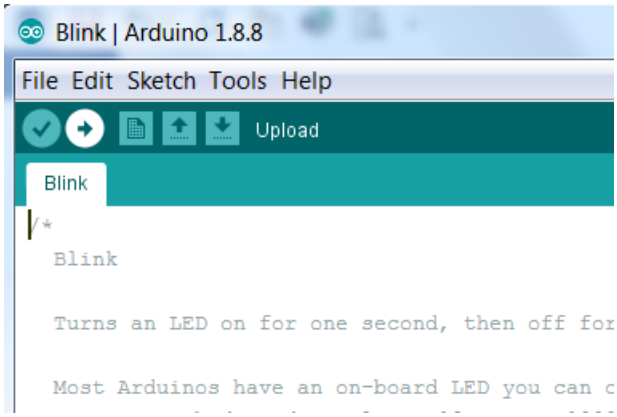
1. Navigate to **File** | **Examples** | **01.Basics** | **Blink**, as shown in the following screenshot:



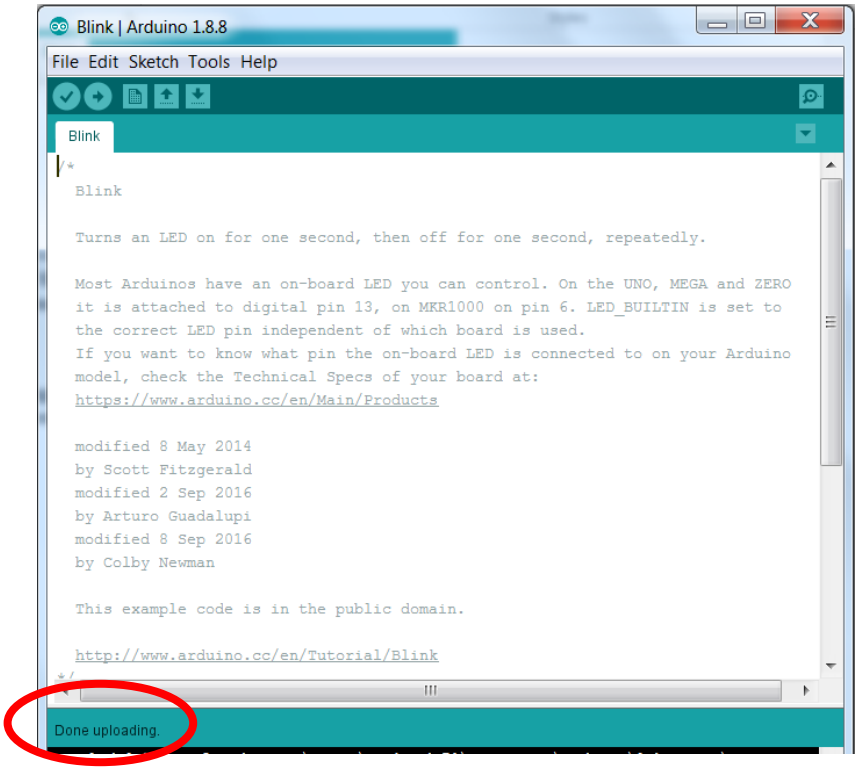
2. You should then see the Blink code in the IDE window as seen in the following screenshot:



3. You will select the **Upload** button, as shown in the following screenshot:



4. Once you have uploaded the file, it will give you an indication in the lower-left corner of the IDE display that the file is uploaded as shown in the following screenshot:

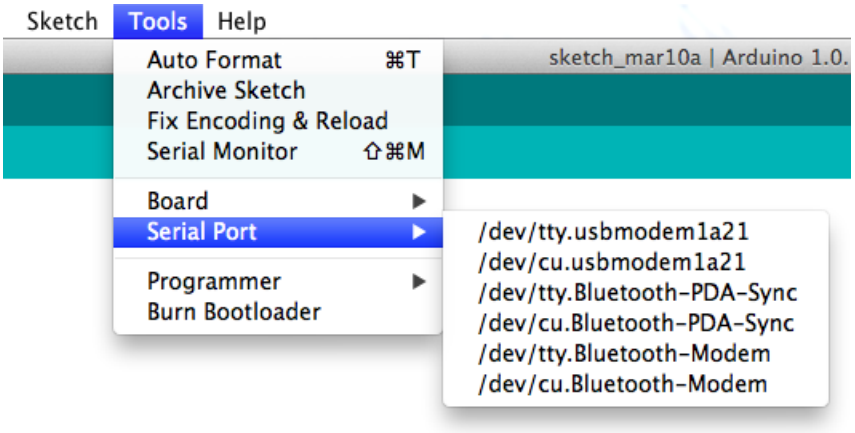


5. When the program is uploaded, it will automatically start running and the Blue LED on the ESP8266 NodeMCU will be blinking.

Using a Mac machine to develop with the Arduino IDE

Using a Mac machine instead of a Windows one is absolutely fine; however, you'll need to change just a couple of the steps. First, as noted earlier, download and install the Mac software from `http://Arduino.cc/en/guide/macOSX#.UxpobfldVHI`. When you plug in your ESP8266 NodeMCU, the system will recognize it and establish a connection. Now, open the Arduino IDE and select the proper board as shown above.

You will also need to select the serial port. When you navigate to the **Tools** | **Serial Port** selection, you should see the following screenshot:

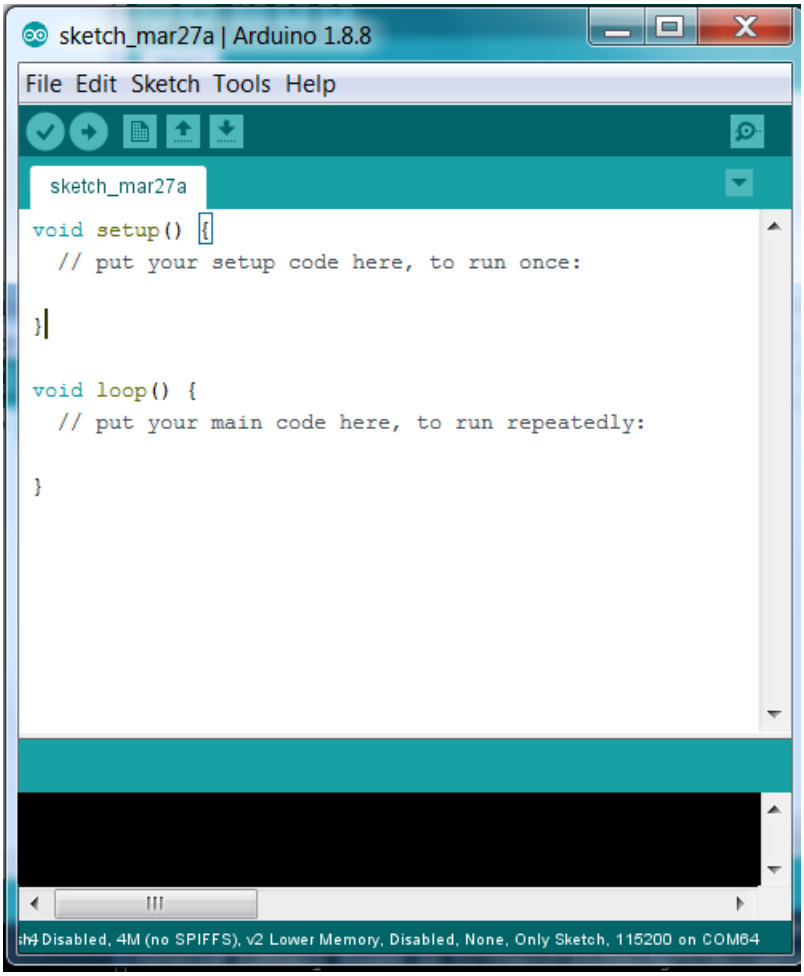


Select the choice that begins with **tty.usbmodem**. If you have other devices connected to the port, you may need to remove the ESP8266 NodeMCU and to see how this selection changes to identify which port is connected to the ESP8266 NodeMCU. You'll then be connected to the device. You should now be able to open the Blink example and run the code and see the Blue LED to flash.

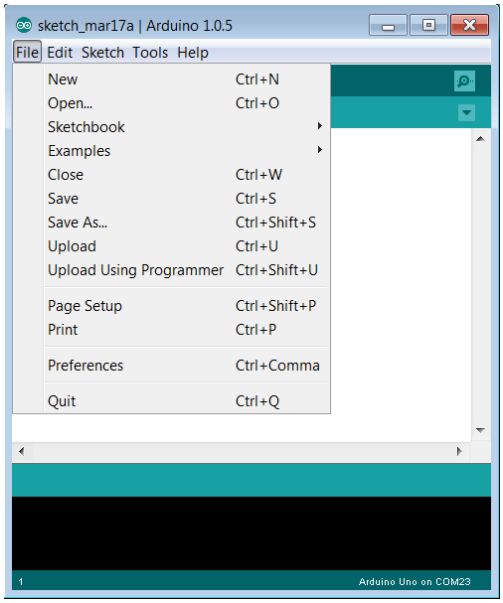
You've completed the first stage. You have your ESP8266 NodeMCU up and talking with your external computer, and know how to connect to the IDE to develop code. Your next step will be learning some programming basics so you can start doing all sorts of amazing things with your ESP8266 NodeMCU.

Creating, editing, and saving files on the ESP8266 NodeMCU

You know how to start the IDE, but you've not been introduced to all the functionality that is available, so let's take a quick tour of the IDE. Again, here is what the IDE should look like when you first start it up:



Notice at the top the five topic menus: **File**, **Edit**, **Sketch**, **Tools**, and **Help**. Each of these tabs holds a set of functionality that you'll need down the road. If you select the **File** tab, you should see the following:

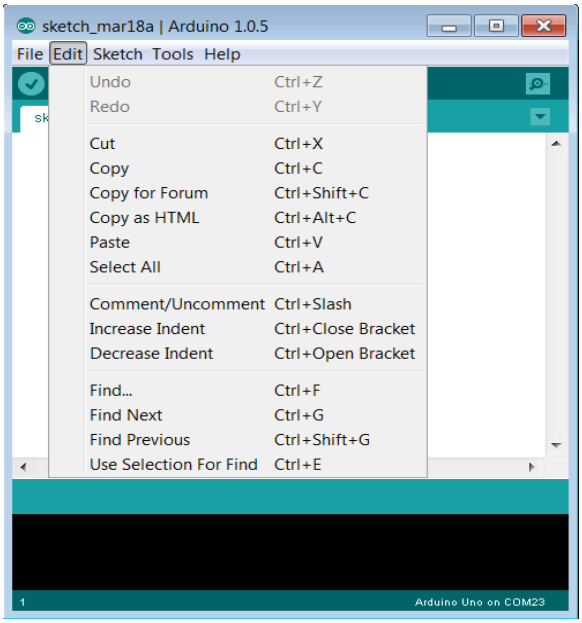


These selections let you create, open, upload and print files in the Arduino IDE.

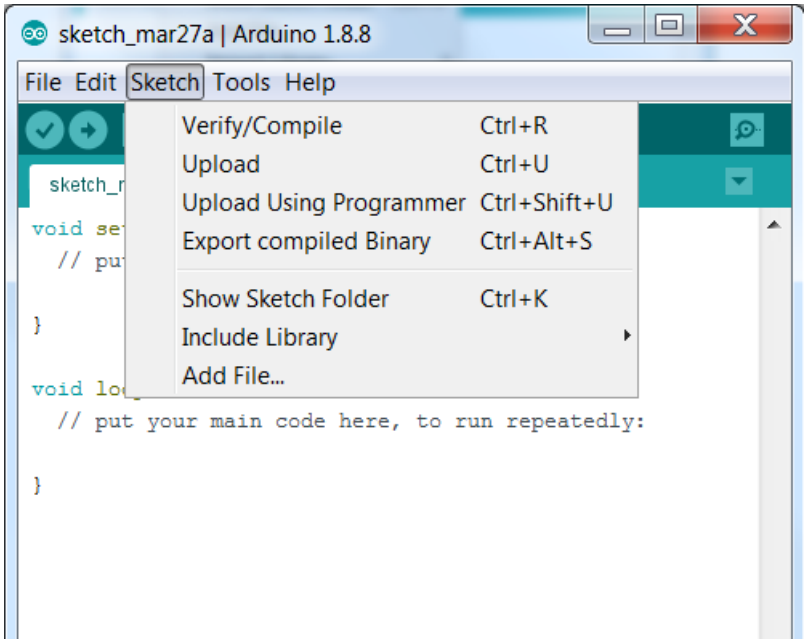
- **New** is very straightforward, you use this if you want to create a new file.
- **Open** is equally clear, you'll select this if you want to open a file that you created earlier.
- **Sketchbook** is probably a new term, one with which you are not familiar. Arduino programs are called sketches, and the Sketchbook keeps track of sketches that you have created. You'll also find sketches placed there if you have downloaded libraries associated with additional HW shields that you may have purchased.
- The **Examples** selection also holds a treasure trove of example programs created by the Arduino community. The IDE comes with a number of basic examples, but others can place examples there as well, normally as part of adding additional functionality associated with HW. *Chapter 4*, Accessing the GPIO Pins will cover that in more detail.
- **Close** is self-explanatory, this will close the open **Sketch**. **Save** and **Save As** allow you to save a sketch to a file. These will normally be appended with the `.ino` extension to differentiate them from other files.

- You've already used the **Upload** selection; it has the same operation as pressing the upload button directly on the IDE. This will compile and send your code to your ESP8266 NodeMCU. The **Upload Using Programmer** is used only when you want to use an external programmer to download code to the ESP8266 NodeMCU. This is sometimes used when you have access to an external programmer, and don't want to use the Arduino Bootloader. We'll not cover that case, if you want to know more, go to http://arduino.cc/en/Hacking/Programmer#.Uyhr_IUXdNp.

Page Setup and Print allow you to setup and print your sketch. Preferences brings up a set of selections to set defaults, including where to store your sketches, the default language, and other settings that you normally don't need to change.If you select the **Edit** tab you will get the following choices:



I'll not cover each of these choices in detail; they are standard edit type commands. If you chose the **Sketch** menu you should see this:

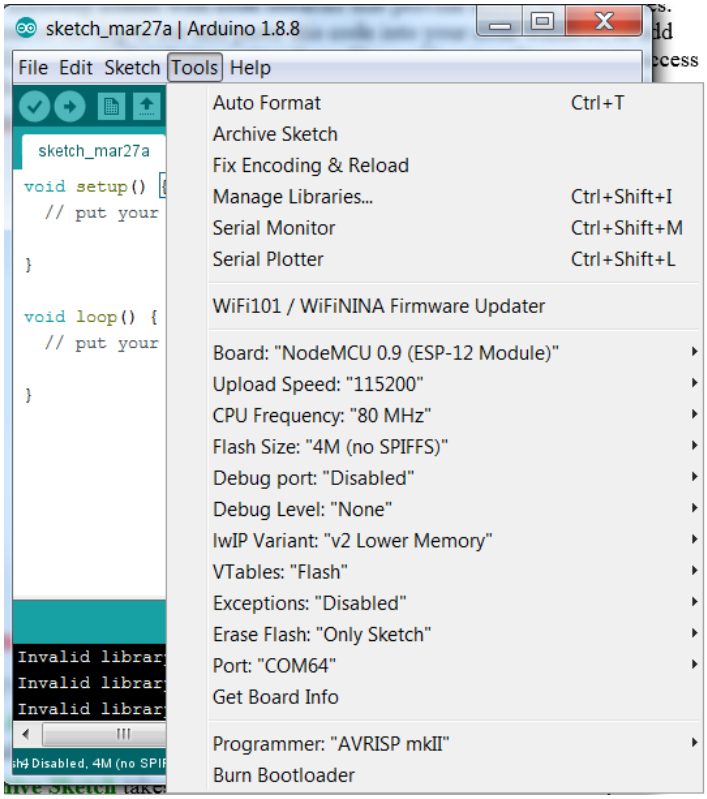


- The **Verify/Compile** selection on this tab allows you to compile and verify your code. This checks that your code can actually can be turned into a program that can be run on the ESP8266 NodeMCU before you try and actually upload it to the hardware.
- The **Upload** selection on this tab uploads the code to your device.
- The **Upload Using Programmer** selection on this tab uploads the code to a device that requires programing hardware.
- The **Export compile Binary** selection on this tab exports the code as a Binary, this can be used to create a file to program many devices.
- The **Show Sketch Folder** selection will open a window that will show all of the sketches in the default directory.
- The **Include Library** selection allows you to add additional libraries that support additional hardware functionalty.
- The **Add File**… choice allows you to add a file to your sketch. Now that may seem odd, as you may have assumed that a sketch is a file. But a sketch can

contain not only the code in one file, but code that is in several files. We cover this capability later in the chapter.

- The **Import Library**… selection is an important one as it allows you to bring in code capability from others; often this is associated with additional hardware that you want to access. The ESP8266 NodeMCU already comes with a large number of functions that you have access to; you don't have to write these sets of code, they are available to you to simply call. You can also add to this set of functionality by importing additional libraries. A1dditional HW normally comes with code libraries that provide access to its features. Instead of having to cut and paste this code into your code window, or add the files to your sketch, importing these files as libraries allows you to access the functionality as if it were in files already associated with your sketch.

The next menu is the **Tools** tab. When you select it you should see this:



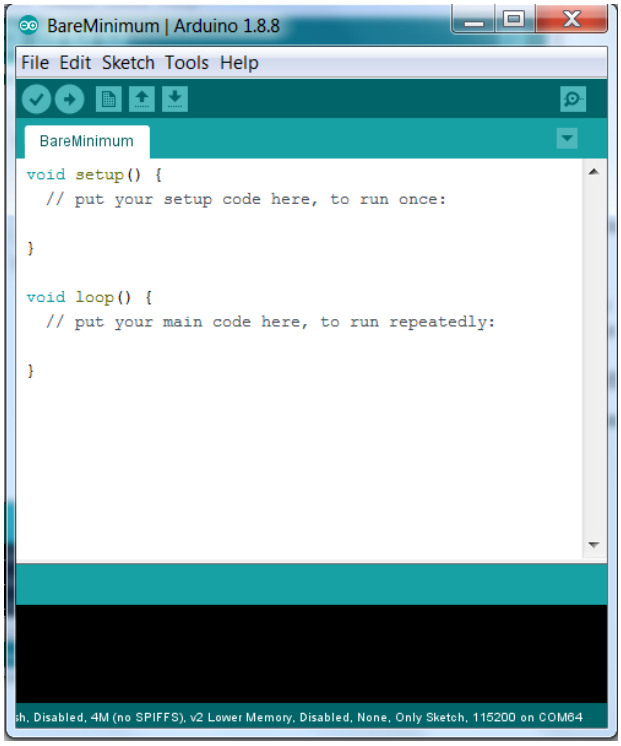
- You've already used the **Board** and **Serial Port** selections.
- The **Auto Format** selection automatically formats the code in the current sketch, placing indents where it thinks they should be.
- **Archive Sketch** takes the sketch and the file associated with it and places them in a *.zip file in the same directory.
- **Fix Encoding & Reload** will sometimes clean up files that are encoded with characters that can't be displayed correctly.
- The **Serial Monitor** selection opens a serial connection between you and the ESP8266 NodeMCU, you can use this to communicate with the board via a USB connection that acts like a serial port. You'll need to have the serial communication commands in your code.
- **Programmer** and **Burn Bootloader** allow you to access an Arduino system without the bootloader, and since you won't be needing those, we'll not cover them here.

The last menu provides help selections; you can explore that on your own. Now that you know your way around the IDE you can start actually programming your ESP8266 NodeMCU.

Basic C Programming on the ESP8266 NodeMCU

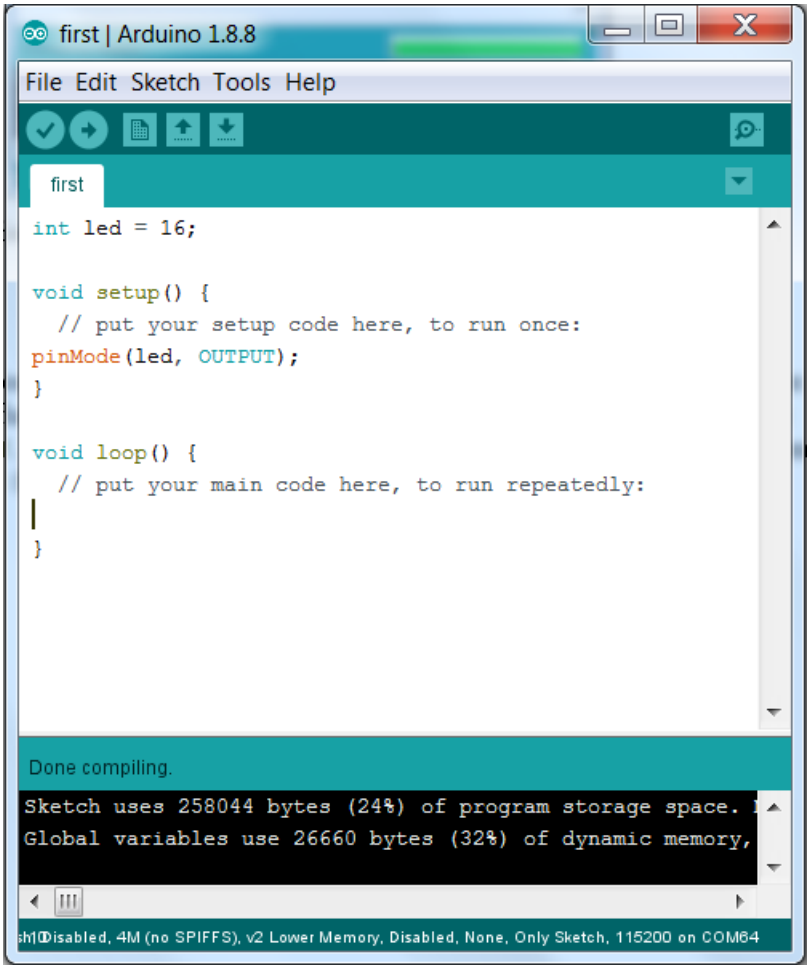
In this section, you'll be learning the C programming language, the language supported by the Arduino IDE. We are going to just cover some of the very basic concepts here. If you are new to programming, there are a number of different websites that provide tutorials. If you'd like to practice some of the basic programming concepts in C, try `www.cprogramming.com/tutorial.html` or `http://www.learn-c.org/` ./ ..

In this section we'll cover how to create a basic sketch. Enter some C code, compile the code, and then upload the code to your ESP8266 NodeMCU. To open a new sketch from the IDE that contains the minimum basic code, select **File ->Examples -> 01.BasicsBareMinimum..**You should now see this in your sketch:



This basic sketch provides two functions; a function is simply an organized set of instructions that the ESP8266 NodeMCU will execute. When the NodeMCU is powered on it begins to execute a list of instructions, one by one. These start in the bootloader, and configure everything and get the NodeMCU to a state that you can use it. Once it has completed executing the statements there, it looks for the `setup()` function. In this function, you will specify any additional setup activity that you need the NodeMCU to do. The NodeMCU will then move to the `loop()` function and begin executing the statements there. This loop function will be run not once, but over and over again until the power is turned off.

Now that you have a context, you can actually start writing some code into the setup and loop functions. Start by putting something in the `setup()` function. Change the setup function to look like this:



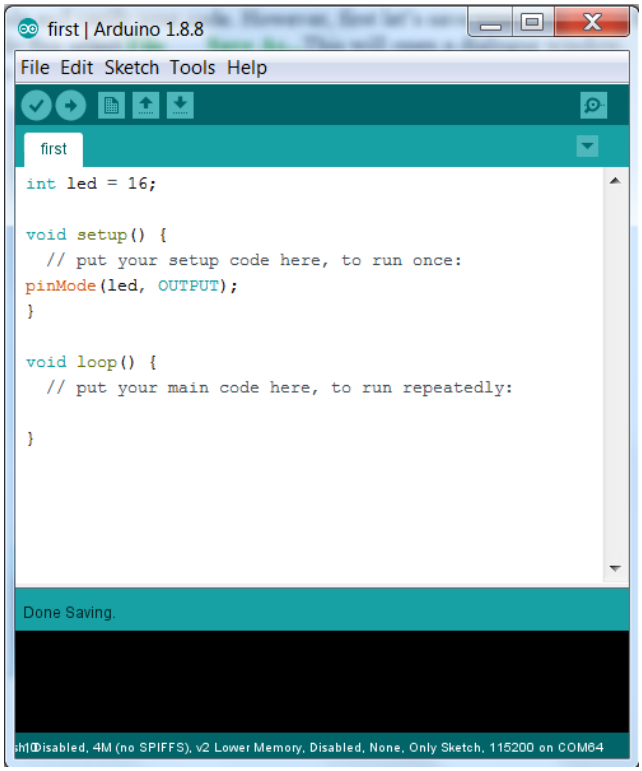
There are two changes that you should have been made.

- int led = 16 - This sets up a storage location in memory named led, and puts the value 16 into this variable. Since this variable is declared outside of any function, it is a global variable. Global variables are available to all functions. This particular variable will hold the value of the output pin that will light your LED. Notice that you need to declare the type of variable, which tells the NodeMCU how big a storage location to set aside for the variable. There are many types available, but you'll use just a few; int for

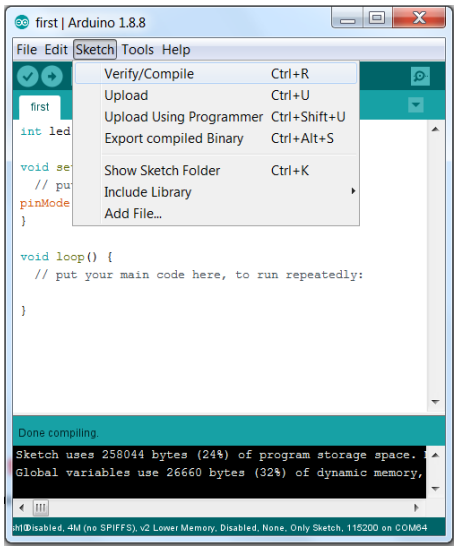
values that have no decimal, float for values that have a decimal, char for character variables and bool for values that are just true or false.

1. pinMode(led, OUTPUT); - This statement calls the function pinMode and passes two values in the argument of the function, led and OUTPUT. Now you might be a bit confused because the pinMode(led, OUTPUT) function is nowhere to be found in your code. This is a library function provided by the NodeMCU system. This particular function takes two arguments; the value of the pin to be set and either the state INPUT or OUTPUT. In this case we want pin 16 (the value stored in LED) to be an OUTPUT.The GPIO pins on the ESP8266 can be configured to either be inputs, or outputs, that is they can either accept a signal or send one.

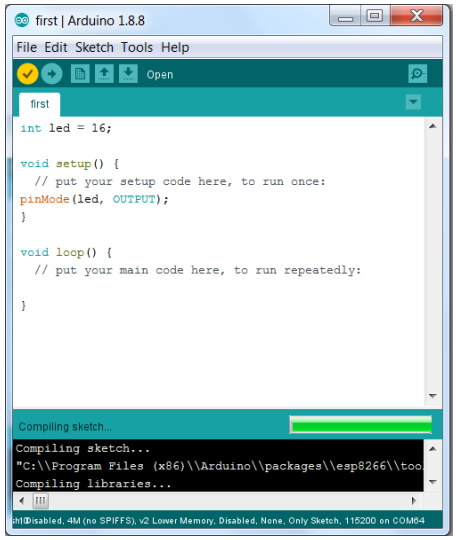
Now you can compile and verify your code. However, first let's save your code under the file name first. To do this select **File** **Save As...**This will open a dialogue window, enter **first**, and then click **Save**. Your IDE should now look something like this:



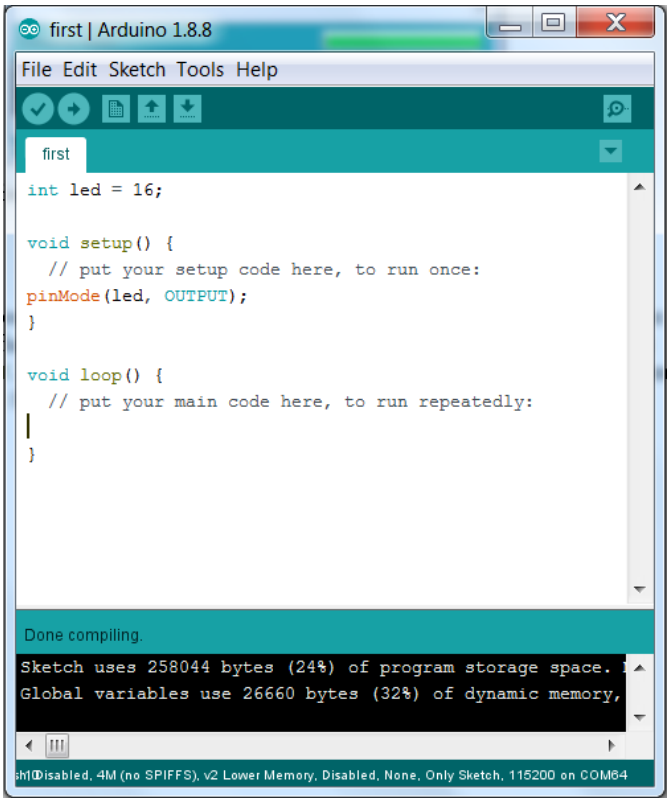
Now you should compile and verify your code, just to make sure you've typed everything in correctly. To do this select **Sketch | Verify/Compile**, like this:



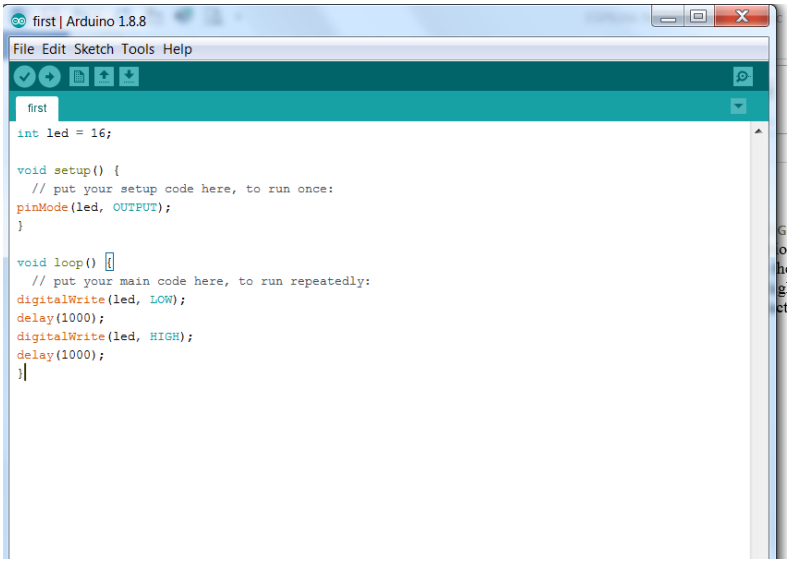
You should see something like this:



When the compile is complete you should see something like this:



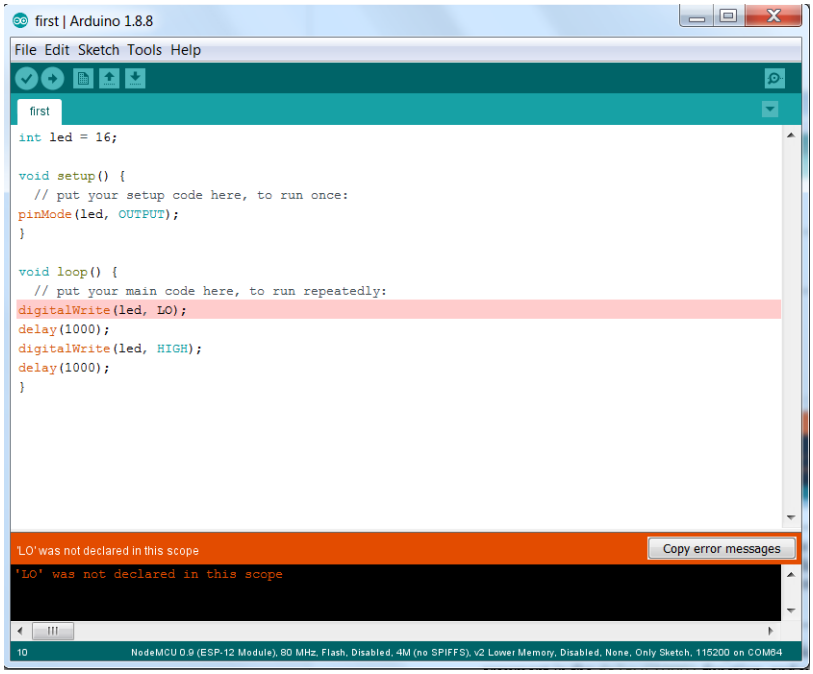
You could also upload your code to the ESP8266 NodeMCU and run your program, but it won't do anything yet, as you have not added enough functionality. You've only defined pin 16 (the one connected to the LED) as an `OUTPUT`. To actually see something on your ESP8266 NodeMCU, you'll need to add some code to the `loop()` function. Now add the following code to the loop function:



You have added the `digitalWrite(led, HIGH);` statement to your loop. The `digitalWrite(led, HIGH)` is another function that is available from the standard Arduino library. This will, each time through the loop, tell the pin 13 (defined with the `pinMode(led, OUTPUT);` statement) to go high, or light. The delay(1000) function simply stops program execution for the number of milliseconds specified in the parenthesis. Make these changes, Save your file, and this time you can compile and upload your code by selecting **File->Upload**. You can also use the upload button.

This selection will compile your code and upload it to the NodeMCU. If everything went as it should, your NodeMCU's blue LED should be flashing.

It might be helpful to show you what happens when you make a mistake. If I type `LO` instead of `LOW` in the first `digitalWrite(led, LO);` function, and then tried the upload, I would see something like this:



Notice that the yellow band shows me the line I mistyped, and tells me the '**LO**' **was not declared in this scope**. Misspelling are one of the biggest reasons why your code might not compile, so check your spelling when you see something like this.

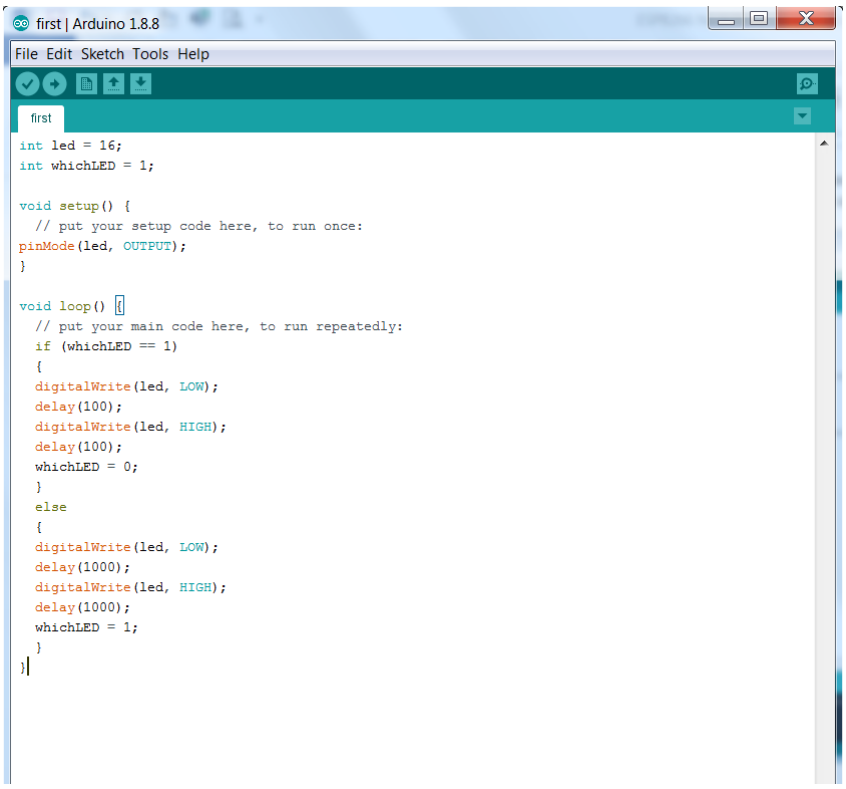
You now know the details behind your first sketch! Play with different values of the argument in the `delay(1000)` function, and your LED should flash at different rates.

Basic Programming Constructs on the NodeMCU

Now that you know how to enter and run a simple C program on the NodeMCU, let's look at some additional programming constructs. Specifically, you'll see what to do when you want to decide between two instructions to execute and how to execute a set of instructions a number of times.

The IF Statement

As you have seen, your programs normally start with the first line of code and then continue, executing the next line, until your program runs out of code. This is fine, but what if you want to decide between two different courses of action? We can do this in C using an **if statement**. Here is some example code:



You'll need to make several changes this time. The first is to add another global variable, `int whichLED = 1;` at the top of your program. Then you'll need to add several statements to your `loop()` function. Here is the detail, line by line:

1. `if (whichLED == 1)` – This is the if statement. The `if` statement evaluates the expression inside the parenthesis. This is the check statement. If it is true it does the next statement or set of statements enclosed by the {}. Notice the `==` instead of a single `=`. A single `=` in C is the assignment operator, which

means the storage location on the right is assigned the value on the left. The `==` operator is a comparison operator, and returns a true if the two values are equal, and false if they are not.

1. `{` - This begins the set of statements the program will execute if the comparisonstatement is true.
2. `digitalWrite(led, HIGH);` - These next four statements turn the LED on and off at a 100 msec rate.
3. `delay(100);`
4. `digitalWrite(led, LOW);`
5. `delay(100);`
6. `whichLED = 0;` - The variable whichLED is assigned a value of 0; This will make sure the next time through the loop it will execute the else statement.
7. `}` – This ends the set of statements that will execute if the comparison statement is true.
8. `else` – The else statement, which is optional, defines a statement or set of statements that should be executed if the comparison statement is false.
9. `{` - This begins the set of statements the program will execute if the comparison statement is false.
10. `digitalWrite(led, HIGH);` - These next four statements turn the LED on and off at a 1000 msec rate.
11. `delay(1000);`
12. `digitalWrite(led, LOW);`
13. `delay(1000);`
14. `whichLED = 1;` This assigns the variable whichLED to a value of 1; This will make sure the next time through the loop it will execute the if statement.
15. `}`

When you have this code typed in you can upload it. When it is uploaded you should see a short flash of the LED followed by a longer flash, much like a heartbeat.

The For Statement

Another useful construct is the **for** construct; it will allow us to execute a set of statements over and over for a specific number of times. Here is some code using this construct:

The code looks very similar to the code you've used before, but we've added two examples of the `for` construct. Here are some details of the `loop()`:

1. `for (int i = 0; i < 5; i++)` – Here is our first for loop. The `for` loop consists of three elements. The `int i = 0;` is the initializer statement. It is only done once, when you first execute the loop. In this case the initializer statement creates a storage location named `i`, and puts the value of 0 in it. The second part of the loop statement is the check statement. In this case the check statement is `i < 5`. If the statement is true, the loop executes. If it is false the loop stops and the program goes to the next statement after the `for` loop. The final part of the `for` loop is statement that is done at the end of each loop. In this case the statement, `i++`, simply means the processor will

add one to the value of i at the end of each loop. This loop, then will be done 5 times, for i = 0, 1, 2, 3, and 4. The check statement will fail when i = 5, and the loop will stop.

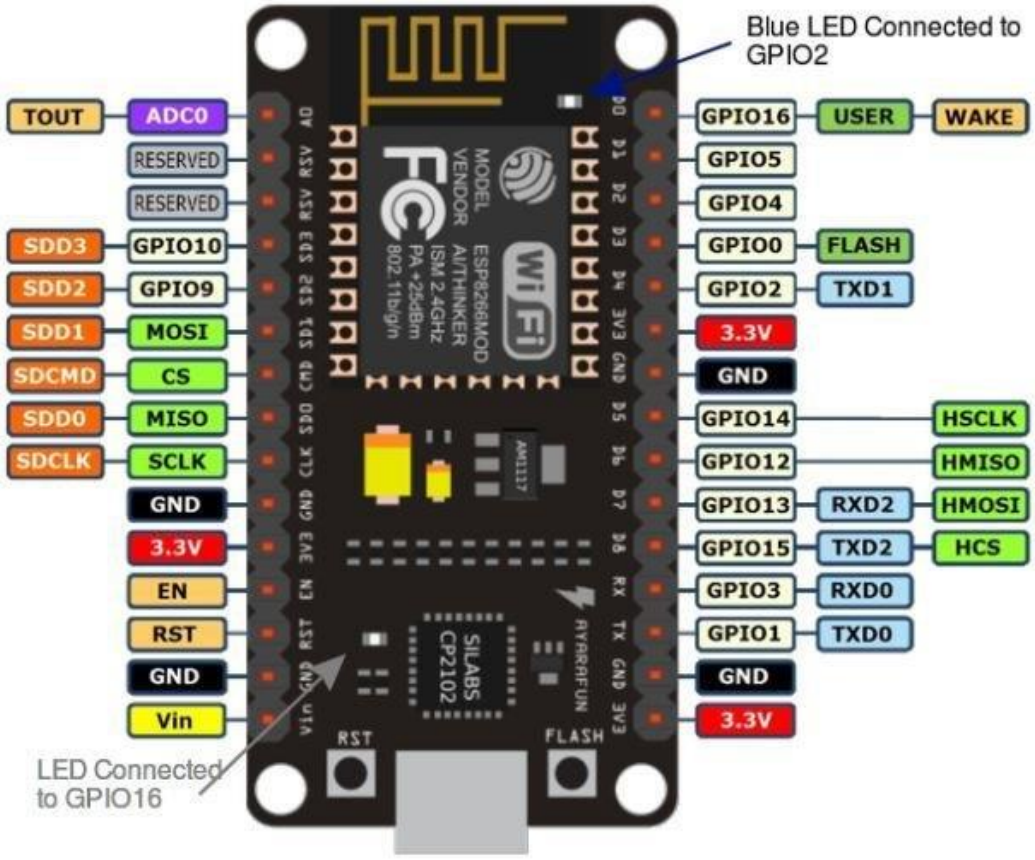
2. `{` - This bracket defines the start of the statements that may be looped.
3. `digitalWrite(led, HIGH);` - These next four statements will be executedeach time through the loop, and will flash the LED light quickly.
4. `delay(100);`
5. `digitalWrite(led, LOW);`
6. `delay(100);`
7. `}` – This ends the loop. Each time through the loop when this statement is reached, the loop statement is executed and the execution goes back to the top of the loop where the check statement is evaluated. If it is true, the statement is executed again. If it is false, the loop stops and execution goes to the statement just following the loop.
8. `for (int i = 0; i < 5; i++)` – This is another loop, just like the one above, except that it flashes the LED for a long time. Just like the first loop, it is executed 5 times.
9. `{`
10. `digitalWrite(led, HIGH);`
11. `delay(1000);`
12. `digitalWrite(led, LOW);`
13. `delay(1000);`
14. `}`

Now you can upload the program, and notice that there are five long flashes of the blue LED, followed by five short flashes.

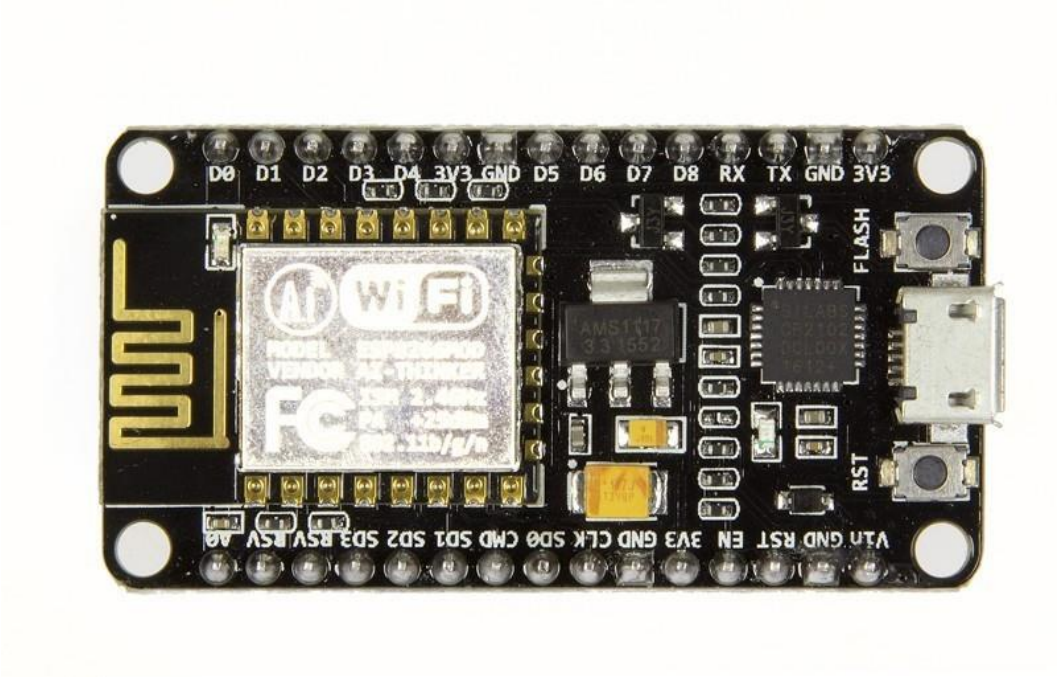
Now hopefully you've learned how to interact with the Arduino IDE and create, edit, upload and run programs on the NodeMCU You have also been exposed to the C programming language. If this is your first experience with programming, don't be surprised if you are still very uneasy with programing in general and if and for statements in particular. You've probably felt just as uncomfortable with your first introduction to the English language, you may not remember it.

The GPIO capability of the ESP8266 NodeMCU

The NodeMCU was built to access the outside world. Much of that access should be through the GPIO pins. Let's look closely at the ESP8266 NodeMCU. As noted earlier, here is a look at the pins available on the NodeMCU:



The ESP8266 NodeMCU comes with a set of 16 Digital pins and 1 Analog IO pin, along with some additional pins to provide power and serial IO. Fortunately the pins are well labeled on the board itself. Here is a close up:



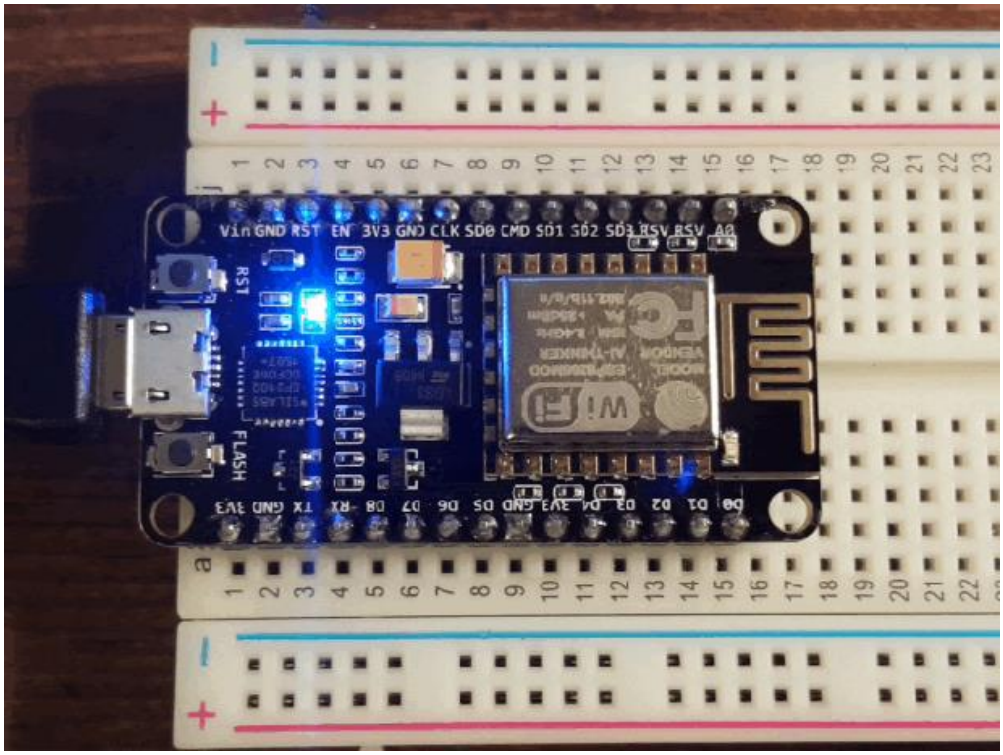
Here is a list of pins that are available, and a brief description of what each pin can do. A more in depth description of these pins will come later as you actually use them in some example projects:

ESP8266 NodeMCU Pin	**Description**
3v3	This pin provides 3.3 Volts. You can use it to power other devices.
GND	This pin provides a ground reference for the AREF pin.
Digital Pins	These 16 pins can be used to either read or write digital values. If input, the value will be read as either a 0 or 1 based on the voltage level at the input. If output, the value will be set to either a 0 or 1 logic voltage level (the logic level on the ESP8266 NodeMCU is a 3.3 V logic level.)
TX	This pin, and the RX pin next to it, provide a

	serial interface that can be used to communicate with other devices.
RX	This pin, and the TX pin next to it, provide a serial interface that can be used to communicate with other devices.
Analog A0	This pins is used as an A/D input to the ESP8266 NodeMCU read continuous Voltage values and turn them into integer values.
Vin	You can power the ESP8266 NodeMCU from this pin. This can be especially useful after you have uploaded your program, you can then disconnect the USB port and when you apply voltage to this pin your ESP8266 NodeMCU will boot and run the uploaded program.
3.3V	This is a voltage output set to 3.3 Volts.
RST	This pin will reset the processor, which will cause the program to be run from the beginning.
EN	This pin can be used to enable or disable the ESP8266 NodeMCU.
CLK, SD0, SD1, SD2, SD3	These pins are used to connect to an I2C or SPI interface.
IOREF	This provides either a 3.3V or 5 V reference, indicating the logic level of the board.

Now that you are aware of all of the GPIO capability you can start putting them to work. In order to do this, it is best to purchase a small breadboard and some jumper wires, this will make connecting to the outside world easier.

They are easy to find, you can purchase one at any electronics store, or on any electronic on-line sites. One of the more useful ways to use the ESP8266 NodeMCU is to plug it into the breadboard, this makes is easy to connect to the pins using the jumper wires. Here is a picture of the ESP8266 NodeMCU plugged into the board:



The jumper wires you want are the Male-to-Male solderless jumper wires. These jumper cables plug easily into the breadboard.

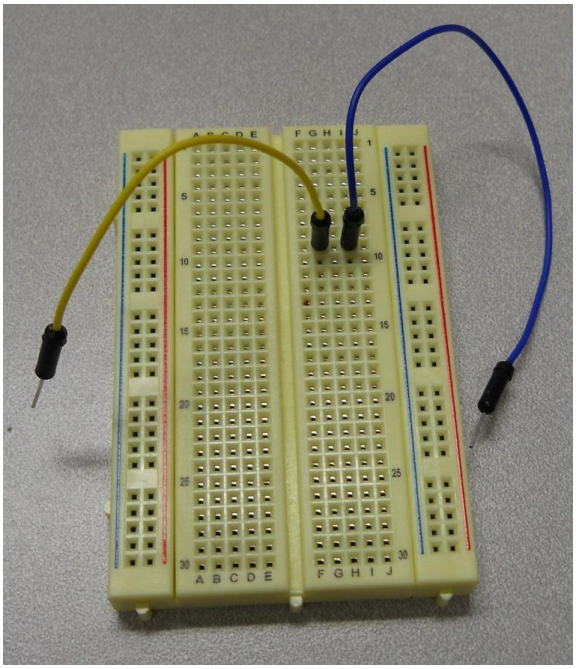
Making your first external hardware connection

Your first project will use the Digital IO pins to light up an LED. To do this you'll need to gather two more hardware pieces. The first is an LED (Light Emitting Diode). This is a small part with two leads that lights up when voltage is applied. They come in a wide variety of colors. If you want to buy them on-line search for a 3 mm LED. You can also get them at most electronics shops. You'll also need a resistor to limit the current to the LED, a 220 ohm resister would be the right size. Again, you can get them online or at most electronics shops.

If you get three of each LEDs and resistors you can exercise several of the Digital IO pins.

Now that you have all the bits and bobs, let's build your first hardware project. Before you plug anything in, let's look at the breadboard for a moment so that you can understand how you are going to use it to make connections. You'll be plugging your wires into the holes on the breadboard. The holes on the breadboard are connected in a unique way to make the connections you desire.

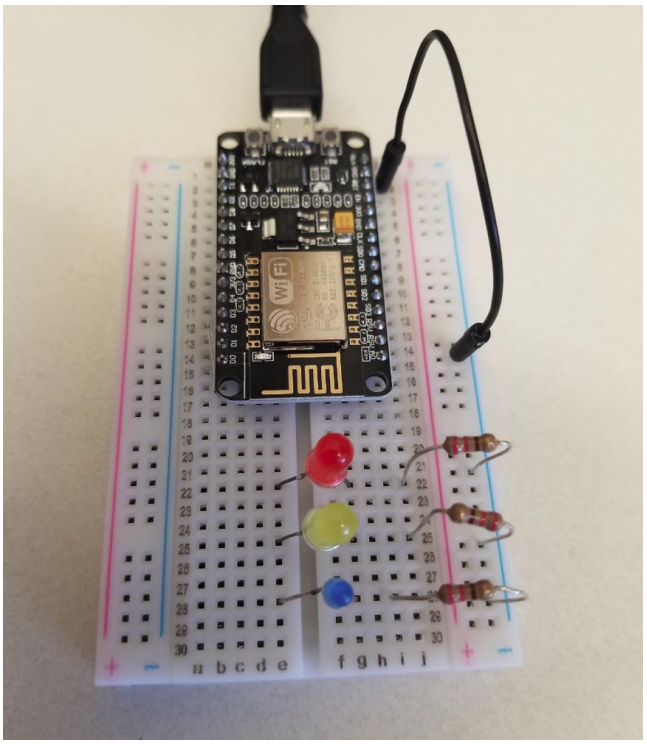
In the middle of the board the holes are connected across the board. So if you plug in wire, and another wire in the hole right next to it, these two wires will be connected, like this:



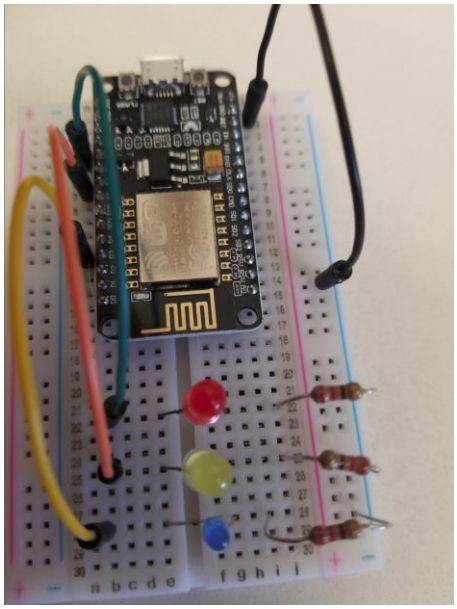
The two rows on each side of the board are generally designed to provide power, so they are connected up and down. So let's place the electronics parts on the Breadboard.

1. Place the LEDs so that one lead is on one side of the middle split of the Breadboard. The direction on the LED is important, make sure the longer of the two lead is on the left side of the hole. Now place the resisters on the holes on one side. The direction of the resistor does not make any difference, but make sure the second wire lead is placed in the row of holes at the end of the board.board. These will all be connected together, and will be connected

to the GND of the ESP8266 NodeMCU using one of the jumper cables like this:



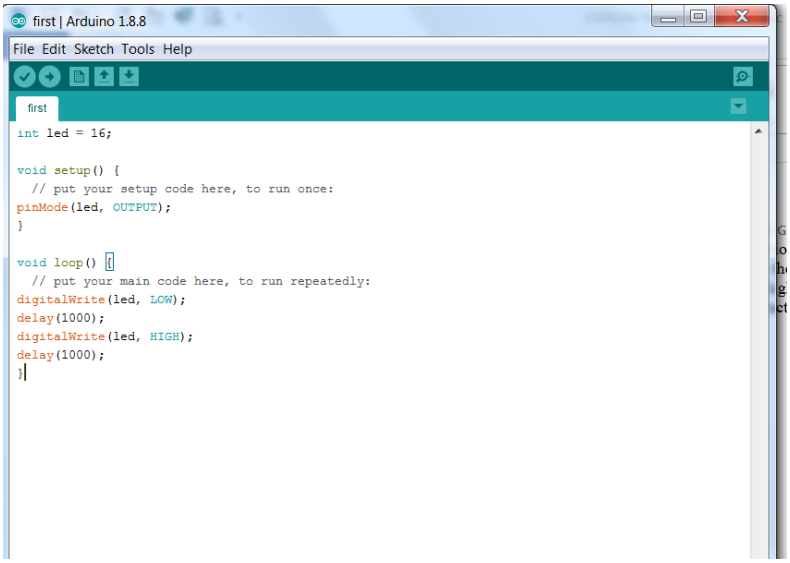
2. Finally, use jumper wires to connect the Digital IO pins D8, D6, and D3 to the holes on the Breadboard, like this:



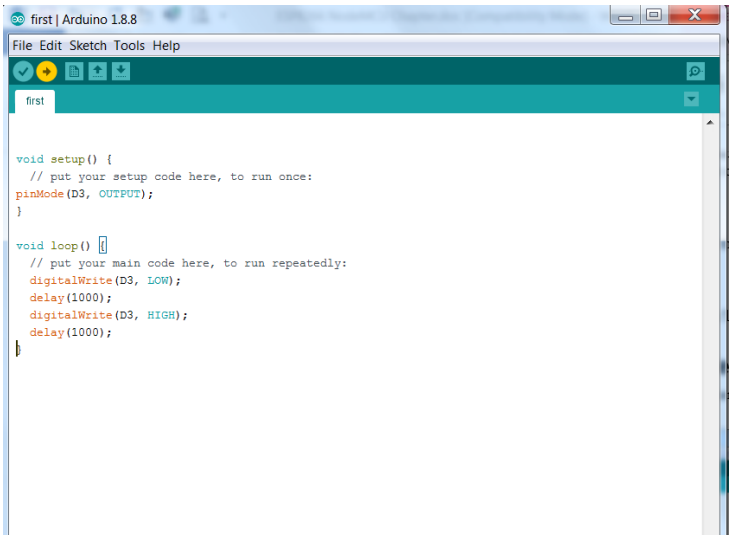
Now that the HW is configured correctly you'll need to add code to activate the LEDs.

Arduino IDE and LED Code

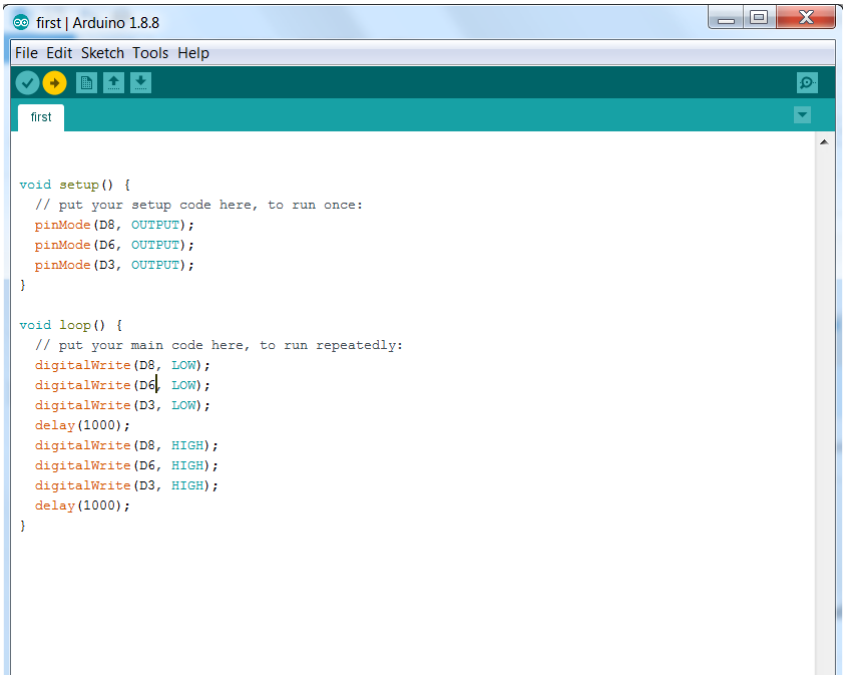
To create the code, start the Arduino IDE. Then recall the code you wrote earlier. The IDE should look like this:



If you remember this code, led 16 lit the blue LED on the board. Now let's change the code so instead of accessing led 16, let's have the signal route to the D8 pin. Here is the code for that:



If you upload and run this program the LED connected to D8 should flash. You'll need to add a similar bit of code to get the LEDs connected to pin D6 and D3. Add the following to the sketch on the Arduino IDE:



Here you are replicating the code for led connected to pin D8 to the second led pin D6 and the third led pin Dd3. You program them all to be output pins, and then in the main loop toggle between high and low. Notice I have three toggling LEDs.

If one or more of the LEDs don't light, check to make sure they are pushed firmly down into the board. You can also change the direction of the LED, perhaps you have the leads in the wrong direction on the board.

You can play with different patterns of LED sequences by using for loops and different wait states. Now that you have created your very first HW project, in the next section we'll cover how to connect to the ESP8266 over the wireless LAN and control HW using the Blynk mobile app.

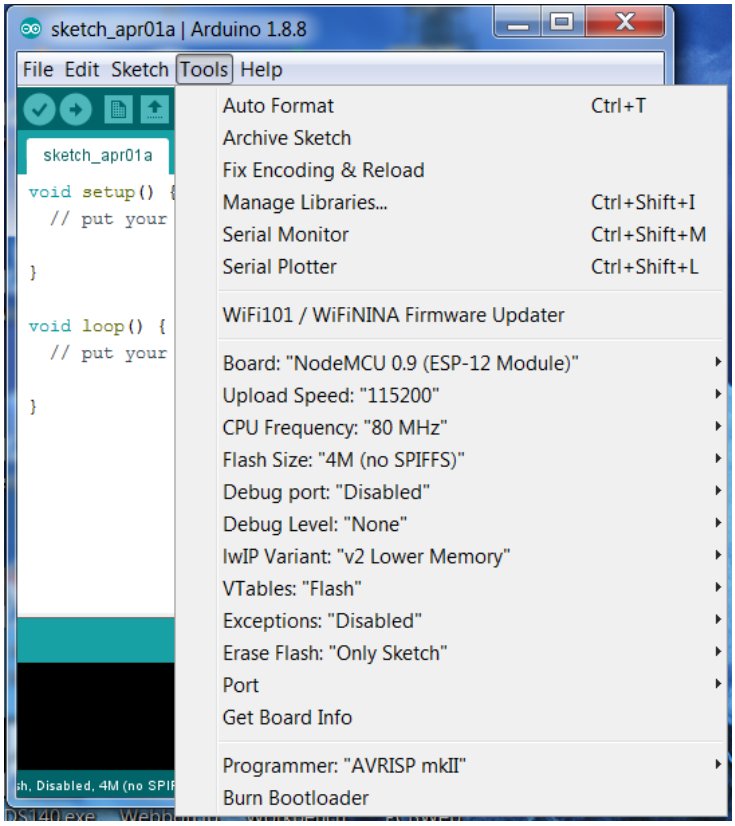
Accessing the Input/Output (GPIO) Pins remotely

Now that you have are familiar with the Arduino IDE and how to create, edit, and upload a program, and how to access the HW from local control, this chapter will now turn your focus to accessing these capabilities remotely. You'll get the chance to learn how to connect to and access, from an IOS device or Android, the capabilities of the GPIO pins.

Connecting the ESP8266 NodeMCU to the wireless LAN

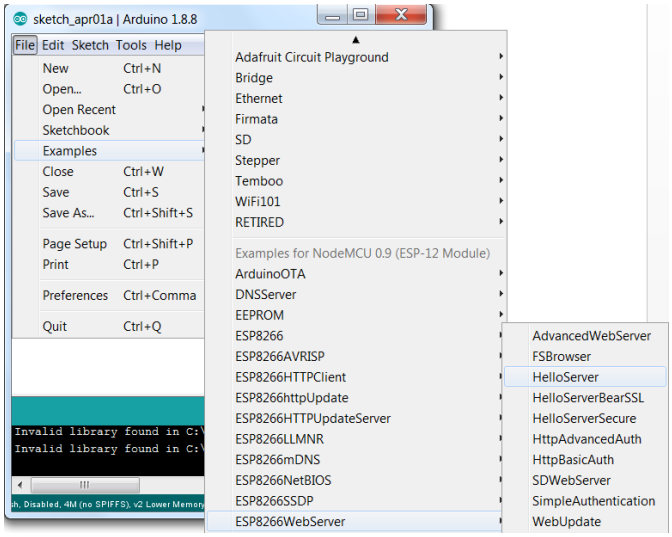
The ESP8266 NodeMCU was built to access the outside world, but it was also built to connect to a wireless network. This is facilitated by several example sets of code that can provide an easy how-to on this type of connection. Let's first connect to the internet and then use a browser window to share information with the ESP8266 NodeMCU.

The first step in this process is to bring up the Arduino IDE. You'll begin with the configuration you set up earlier. Just to review, if you select the Tools tab you should see this displayed:



Now you'll load an example set of code that sets up a simple web based server. Before you do this, however, you'll need to make sure you have a wireless router configuration set up, and that both your host computer as well as your ESP8266 NodeMCU can connect to it.

Once you have a viable wireless network, select **File->Examples->ESP8266WebServer->HelloServer**, like this:

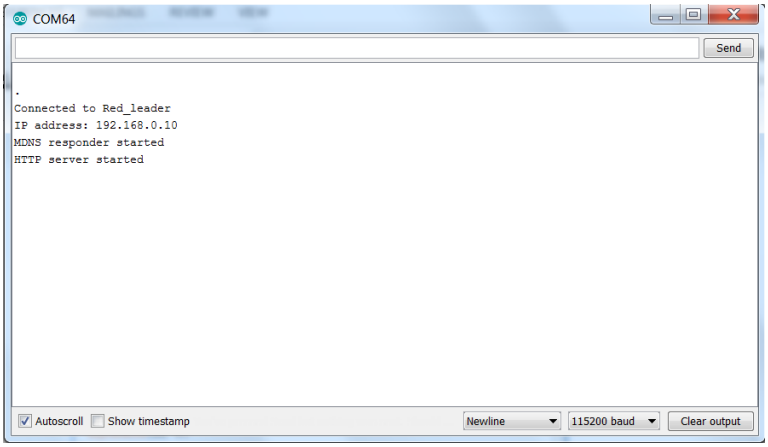


You should now see this code:

There are three changes you'll need to make to the code. The first two are to fill in the SID and Password on your network. Place these in the STASSID and STAPSK #define locations. The last change is to change the led value to 16 (the on-board led for the ESP8266 NodeMCU). Now upload the code to your board.

You can monitor the progress on the Serial Monitor. Bring up the Serial Monitor from the Tools -> Serial Monitor Selection. You will also want to make sure the Baud Rate of the connection is set to 115200 baud. That selection is in the lower right hand corner of the Serial Monitor.

If you are successful you should see something like this:



Now the server is running go to the host computer and open a web browser window. In the address space type 192.168.0.10 (the static IP address of your device.) You should see a text in the web browser windows that says "hello from the esp8266!." You are connected!

There are many other examples of how to set up the ESP8266 NodeMCU to connect to the outside world via its wireless lan capability. You can explore them later. For now you'll be wanting to open an example that will let you work with the Blynk app. But first a few words about the Blynk app.

Connecting and controlling your ESP8266 NodeMCU with the Blynk app

The Blynk is a Platform with iOS and Android apps to control Arduino, Raspberry Pi and the likes over the Internet.

Blynk was designed for the Internet of Things. It can control hardware remotely, it can display sensor data, it can store data, vizualize it and do many other cool things.

It's a digital dashboard where you can build a graphic interface for your project by simply dragging and dropping widgets. It's really simple to set everything up and you'll start tinkering in less than 5 mins. Blynk is not tied to some specific board or shield. Instead, it's supporting hardware of your choice, in this case the ESP8266 NodeMCU.

How does it work?

There are three major components in the platform:

Blynk App - allows to you create amazing interfaces for your projects using various widgets we provide.

Blynk Server - responsible for all the communications between the smartphone and hardware. You can use our Blynk Cloud or run your private Blynk server locally. It’s open-source, could easily handle thousands of devices and can even be launched on a Raspberry Pi.

Blynk Libraries - for all the popular hardware platforms - enable communication with the server and process all the incoming and outcoming commands.

Its features:

* Easy to use

* Awesome widgets like LCD, push buttons, labelled value, graphs

* Not restricted to local Wifi network

*Direct pin manipulation with no code writing

*Easy to integrate and add new functionality using virtual pins

Configuring the Arduino IDE to work with Blynk

The Blynk app has a set of library files which have to be included in the Arduino IDE environment before the project is executed.

1. Follow the link to install libraries

http://www.blynk.cc/getting-started/

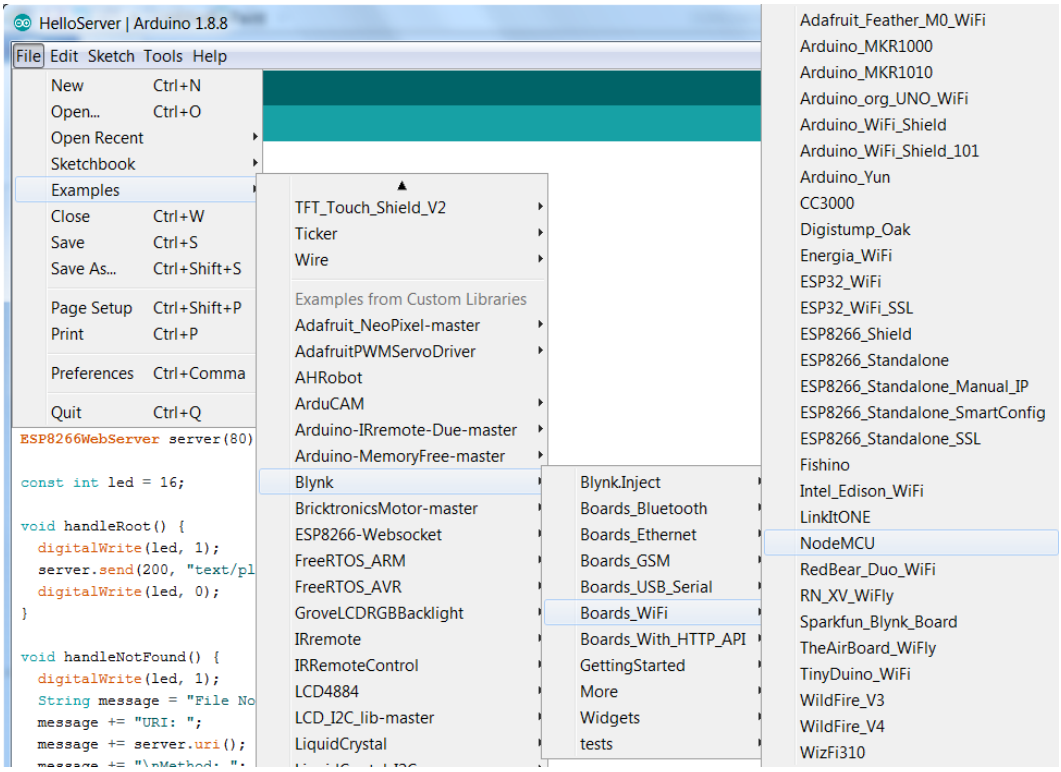
Scroll down the page and select the C++ libraries.

2. Once the Zip file is downloaded, extract it and individually copy all the folders to your libraries folder of your Arduino installation (often this is found in your Documents/Arduino directory).

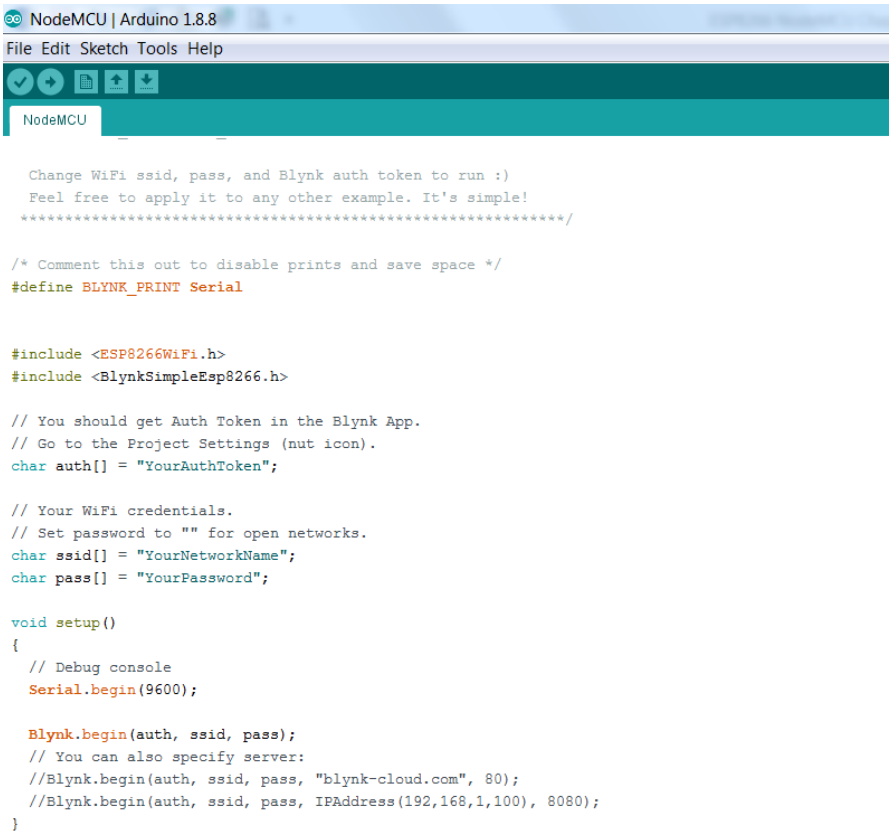
3. Once done just open Arduino IDE and go to **Sketch-> Include libraries** and you would see blynk in the menu under the contributed library heading.

4. If you see that then libraries have been included successfully.

Now let's look at an example. To do this you'll select Files->Examples. You should see a Blynk choice in the Examples from Custom Libraries.



As shown you'll want to load the Examples->Blynk->Boards_WiFi->NodeMCU selection, and then you should see this code:



Again you'll need to fill in the ssid and password for your network. But for this application you'll also need to fill in the authorization token you get from you Blynk app.

To get this you'll need to install and authorize your Blynk app on your device. Follow the directions at https://blynk.io/en/getting-started after installing the Blynk app. They are summarized here:

1) Create New Account in Blynk app

An Account is needed to save your projects and provide access from any smartphone you have. Use a valid email address as it will be later used often.

2) Create New Project

• Create New Project and choose the hardware you wish to use, in this case the ESP8266.
• Choose Dark or Light UI interface

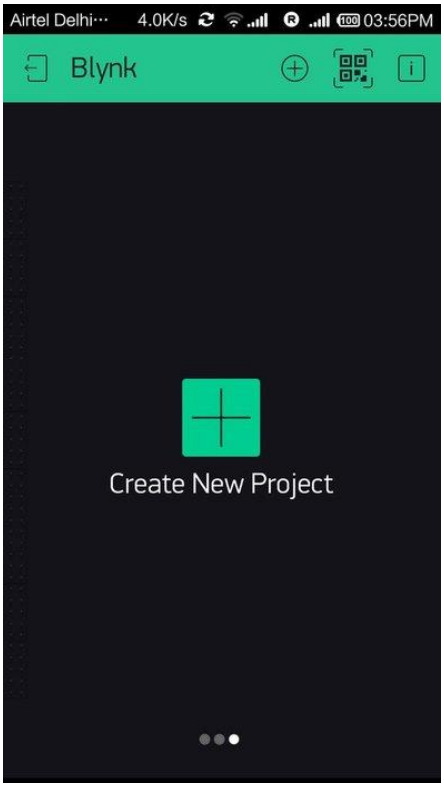
3) Get Auth Token

Check your inbox to see if you get an email from Blynk with the Auth Token. You will need it later.

Once you get the authorization token, then fill it in on the IDE.

Setting Up Blynk

Once you've downloaded the App, run it on your smart device. You should see this:



Select the Create New Project. Fill in the name of your project, and the hardware model. Make sure the Auth Token is correctly entered. Now select Create.

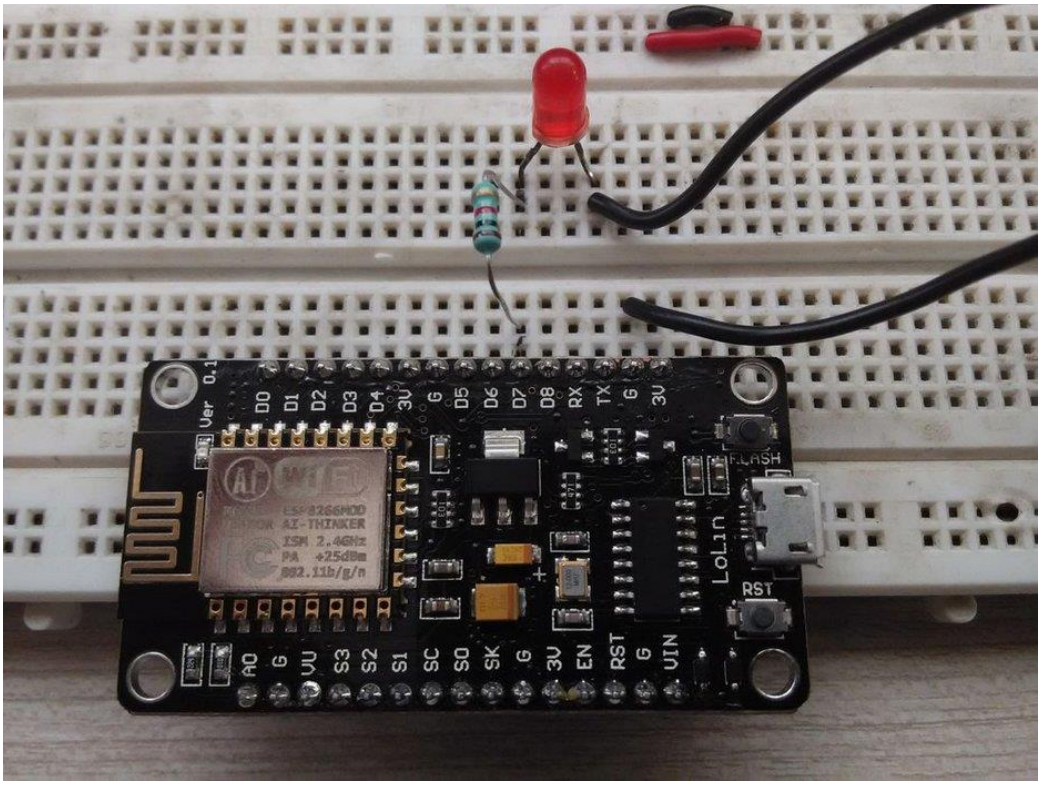
Now you will get your dashboard screen. Just click on the the top most button "+" on the right corner to add widgets to your project.

1. In this project you add a simple button and then configure its settings as Digital GP13 pin.
2. Tap the button.
3. Its your choice you can either have the button set as push type or as a switch
4. Then label the Button as ON and OFF in the settings

Note that since Blynk is free only to an extent, you have to choose your widgets wisely

Building the Circuit

Here is the circuit diagram:



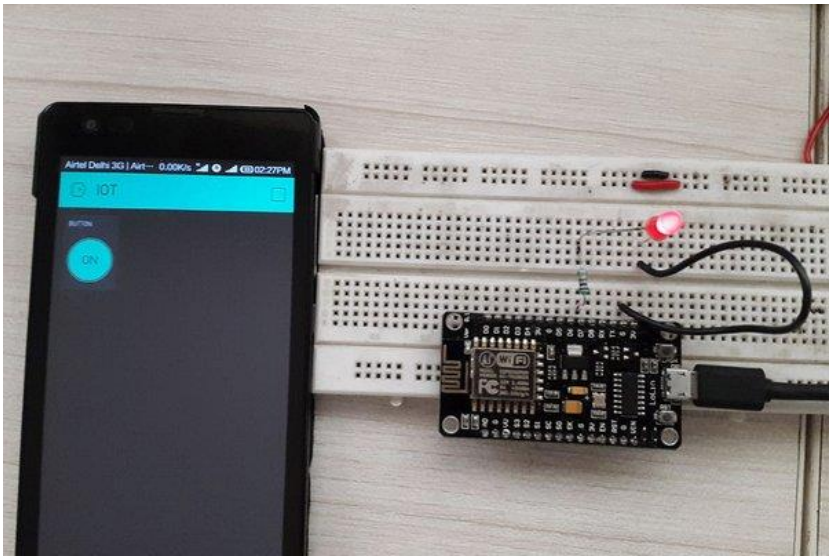
The connection is pretty simple just connect the Led to D7 pin via 330/220Ohm resistor (refer the diagram).

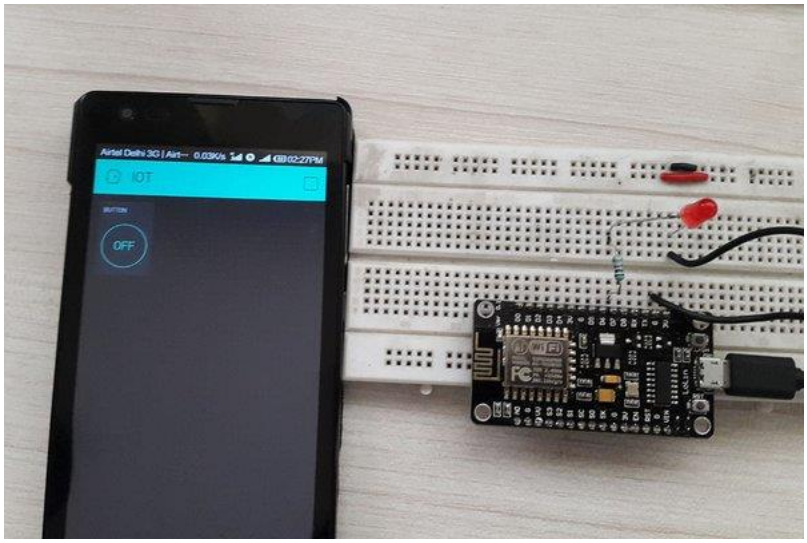
Uploading the Code

Now upload the code from the Blynk example using the Arduino IDE.

Now Execute the code in Blynk

The final step is to execute the Blynk example project.





Now open the Blynk app in the Phone. Let it connect to the internet, you will see your dashboard with a button. Press Play button on the top most right corner of the app. Now Press the Button and you would see the LED Turn ON and OFF.

There are lots of additional examples of how to use the Blynk app to control remote hardware and display remote measurements. The tutorial at https://www.hackster.io/helloworld1997/ultrasonic-sensor-with-blynk-and-nodemcu-50c074 shows how to connect to a remote ESP8266 NodeMCU with Blynk and display at distance measurement using the HC-SR04 sonar sensor.

This brief tutorial hopefully helped introduce you to the ESP8266 NodeMCU. But it is just the beginning. So feel free to explore.