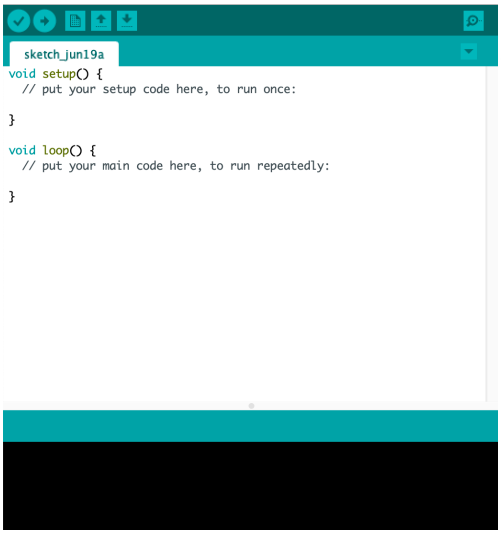
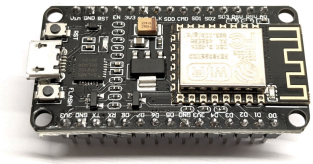
IDE Configuration

The blinking sketch

Project description:

In this project, you will setup the development environment to program a generic ESP8266 board. The **Arduino Software (IDE)** is an open-source software which makes it easy to write code and upload it to a variety of compatible microcontrollers. You will first download and install the Arduino programming environment and successively test and upload your first program to make the built-in LED on the ESP8266 board blink.





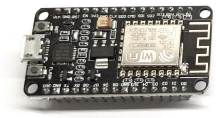
Project objectives:

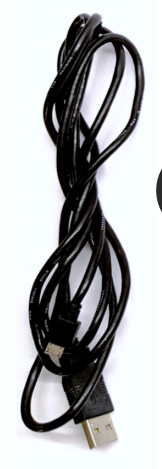
- Install, configure, and use the Arduino Software (IDE)
- Install the ESP8266 add-on to program ESP boards
- Connect the ESP8266 board to your computer via USB
- Load and run your first program

Project components:

Component Reference	Component Quantity	Component Name	Component Description
A	1	ESP8266 WiFi Module	A low-cost Wi-Fi microchip with full TCP/IP stack and microcontroller capability
B	1	Micro USB Cable	A USB cable to power and upload instructions to a microcontroller

A





B

Step One
Downloading and Installing the Arduino IDE

The **Arduino Software (IDE)** is an open-source software that makes it easy to program your microcontrollers. It is the main text editing program used for Arduino programming and can be also used to program other boards. It is available for Windows, macOS, and Linux.

First thing first, head to the Arduino software website and download your operating system compatible version. You can find the software download page at this link: https://www.arduino.cc/en/software.

Once you have finished with the download process, locate the downloaded zip file and extract its content. This will begin the software installation; please follow the promoted instructions to install the Arduino IDE.

Finally, launch the Arduino IDE like any other program. If the installation was successful, you should see the Arduino software appear on your screen as shown below:

Step Two
The Arduino IDE

As you can see, the Arduino IDE has a minimal design. Its job is to translate and compile your sketches (code files) into code that Arduino or other compatible boards can understand. Once the code is compiled it is then uploaded to the board's memory. The IDE programming language is based on C/C++ but with some restrictions due to the little available RAM found in most microcontrollers.

There are five headings located on the top right hand side of the Arduino IDE:

1. **Verify:** This button compiles your code and checks for potential errors. No code is uploaded to your board with this button.

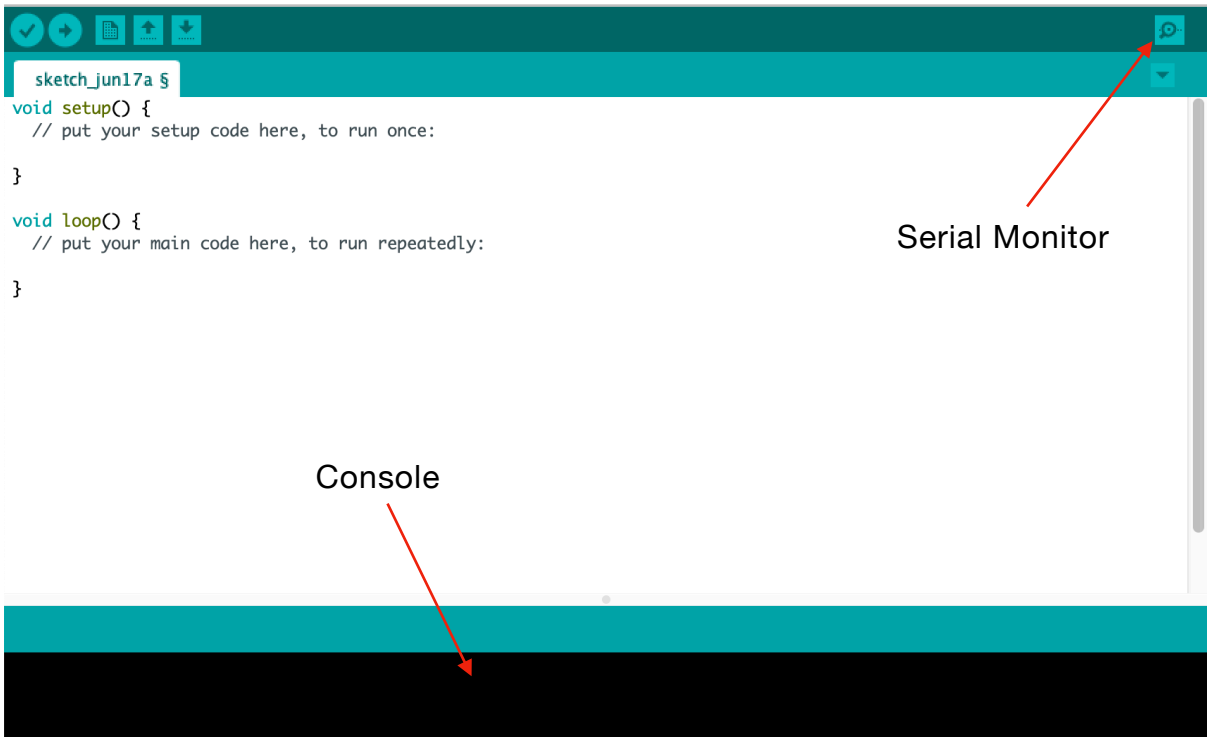
2. **Upload:** This button compiles your code and checks for potential errors. The code is also uploaded to your board with this button.

3. **New:** This button creates a brand new sketch. You can work on multiple sketches at the same time.

4. **Open:** This button opens an existing sketch.

5. **Save:** This button saves your current sketch.

Other two important sections of the IDE are the **console**, and the **serial monitor**:



Console: The console outputs important information about your code. It shows potential errors during compilation and upload.

Serial Monitor: The serial monitor is used mainly for interacting with the boards using the computer, and is a great tool for real-time monitoring and debugging. Used in combination with the Serial class, it allows to debug your microcontroller in real-time. You will see an example of the serial monitor in later projects.

On last useful section is the “down arrow” button located just below the serial monitor button. This is the tab manager and allows you to organise your code into multiple tabs.

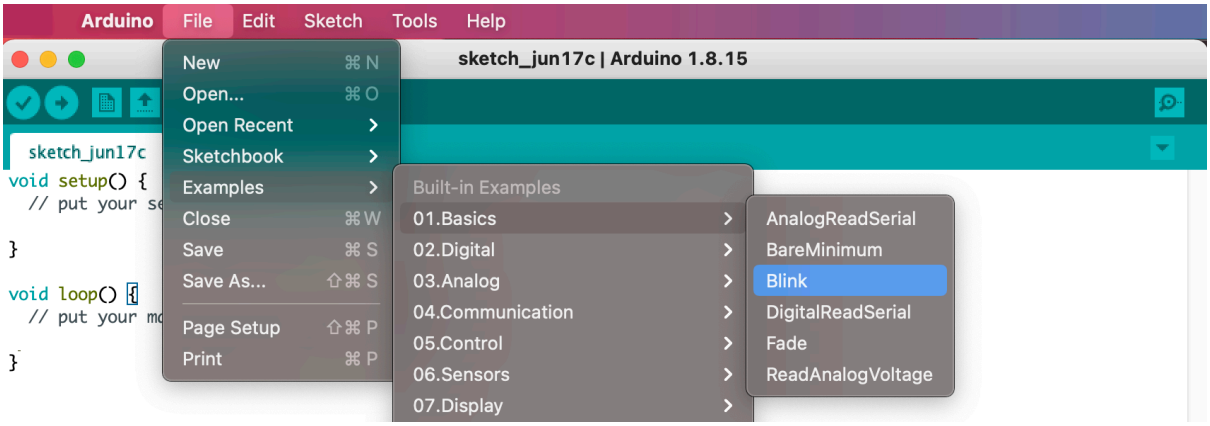
Now that you are familiar with the main sections of the Arduino IDE, it is time to load and explore a sketch example.

Step Three

The Built-in Blink Sketch Example

The Arduino IDE comes with some built-in examples. Let's explore the blinking sketch, a program that interacts with the built-in LED found in most boards.

In order to load the example sketch, use the main navigation bar and click on **File > Examples > 01.Basics > Blink**



The blinking sketch should now appear on your desktop:

The blinking sketch example uses the built-in LED that most Arduino boards have. This LED is connected to a digital pin and its number may vary from board type to board type. Let's explore the blinking code example.

The sketch has two main functions:

setup(): This function is required on every Arduino sketch. It defines the initial state of the Arduino upon boot and runs only once. Here, you initialise variables, define the pins functionality and mode, and define the initial state of the pins.

loop(): Similarly to the setup() function, the loop() function is also a must for every Arduino sketch and executes once setup() is complete. It is the main function and it runs in a loop over and over again. Here, you write the logic of your circuit that you want to execute multiple times.

The sketch begins with a **description** of the built-in example:

```
/*
  Blink

  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink
*/
```

The description is enclosed within the **/* */** characters. This is how you can create block comments in your sketches. Block comments are comments that can be formatted in multiple lines.

You can also create inline comment by preceding some text with the **//** symbol:

// This is an inline comment

Comments are ignored by the compiler.

Next the **setup()** function:

```
// 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);
}
```

You can see examples of inline commenting on the setup() function. Here, the function is used to initialise a digital pin as an output, meaning that we want to send out a signal to the pin and not receive (INPUT) a signal from the pin.

The **pinMode(LED_BUILTIN, OUPTUT)** line of code sets the specified pin number as output. The **LED_BUILTIN** variable, as instructed in the code description, automatically set the correct pin independent of which board is used.

In the case of an Arduino board, we are telling the program to initialise pin13 (the pin connected to the built-in LED on Arduino boards) as an output pin.

Next the **loop()** function:

```
// 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
}
```

The loop() function activates and deactivates the pin with a delay of 1000 milliseconds. The outcome of such code is a blinking behaviour for the initialised pin:

- Activate the pin -> **digitalWrite(LED_BUILTIN, HIGH)**
- Wait for 1 second (1000 milliseconds) -> **delay(1000)**
- Deactivate the pin -> **digitalWrite(LED_BUILTIN, LOW)**
- Wait for 1 second (1000 milliseconds) -> **delay(1000)**
- Repeat the process infinitely -> **loop()**

Step Four
ESP8266 Add-on and code upload

Now that you are familiar with the blinking example sketch, it is time to upload the code to your ESP8266 board.

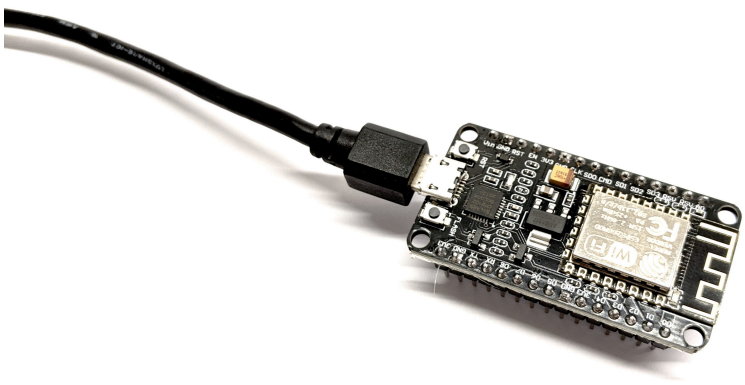
First thing, make sure that the code compiles successfully. Click on the **Verify** button to compile your code. If the code compiles successfully, you will get a confirmation from the console:

```
Done compiling.

Sketch uses 924 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables. Maximum is 2048 bytes.
```

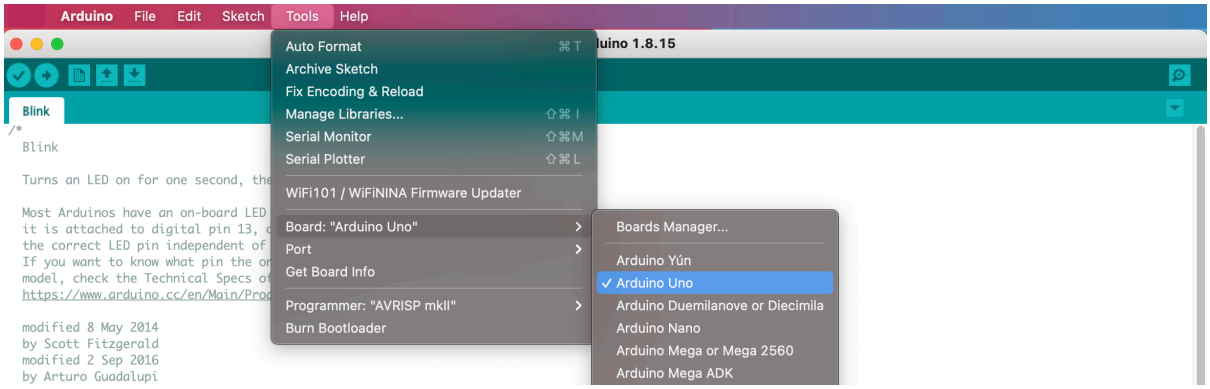
The next step is to connect the board to your computer. In order to connect the ESP8266 board to your computer, you need a Micro USB cable. The cable is use to both power and transfer data to your board.

Go ahead and connect your ESP8266 board to your computer via a Micro USB cable. You should see a temporary LED flash, suggesting that the board is connected.



Before you can upload the code, there are some settings that you need to configure to select the correct board.

The last step before you can upload the code to your board is to select the board type from the Arduino IDE drop down list. Use the main navigation bar and click on **Tools > Board** as shown on the image below:

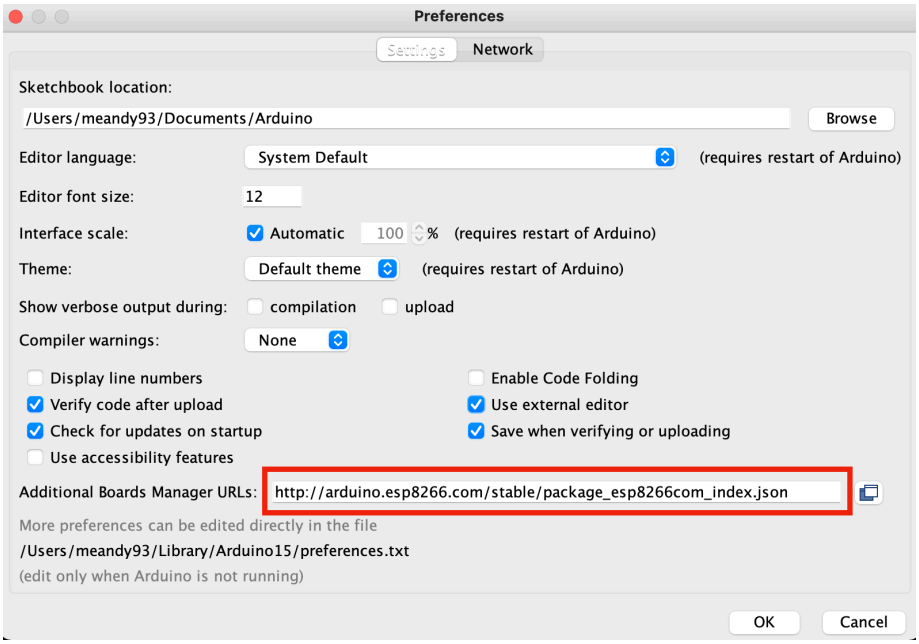


You may notice that there is no option for the ESP8266 board. This is because some board models are not pre-installed in the Arduino IDE, therefore you will need to install them before you can upload your code.

Use the main navigation bar and navigate to the **preferences** section. Note that the preferences tab location might differ across operative systems.

Once opened the preferences section, navigate to the **Settings** tab and paste the following link as shown below:

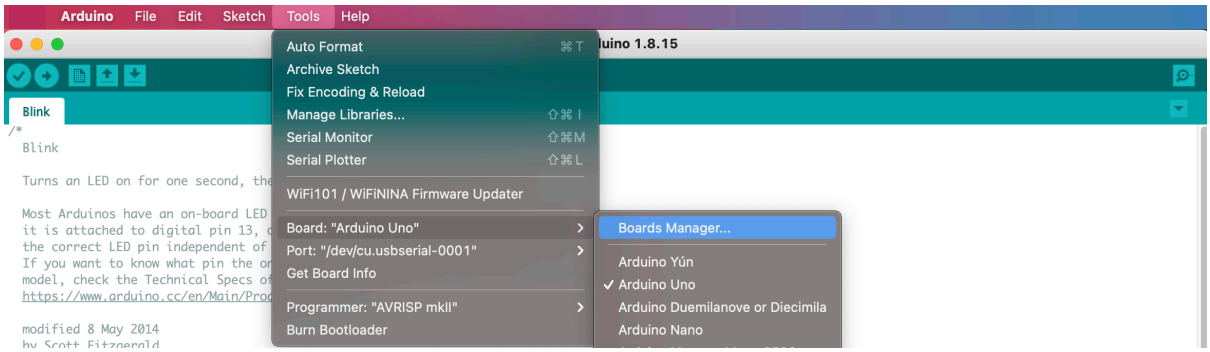
http://arduino.esp8266.com/stable/package_esp8266com_index.json



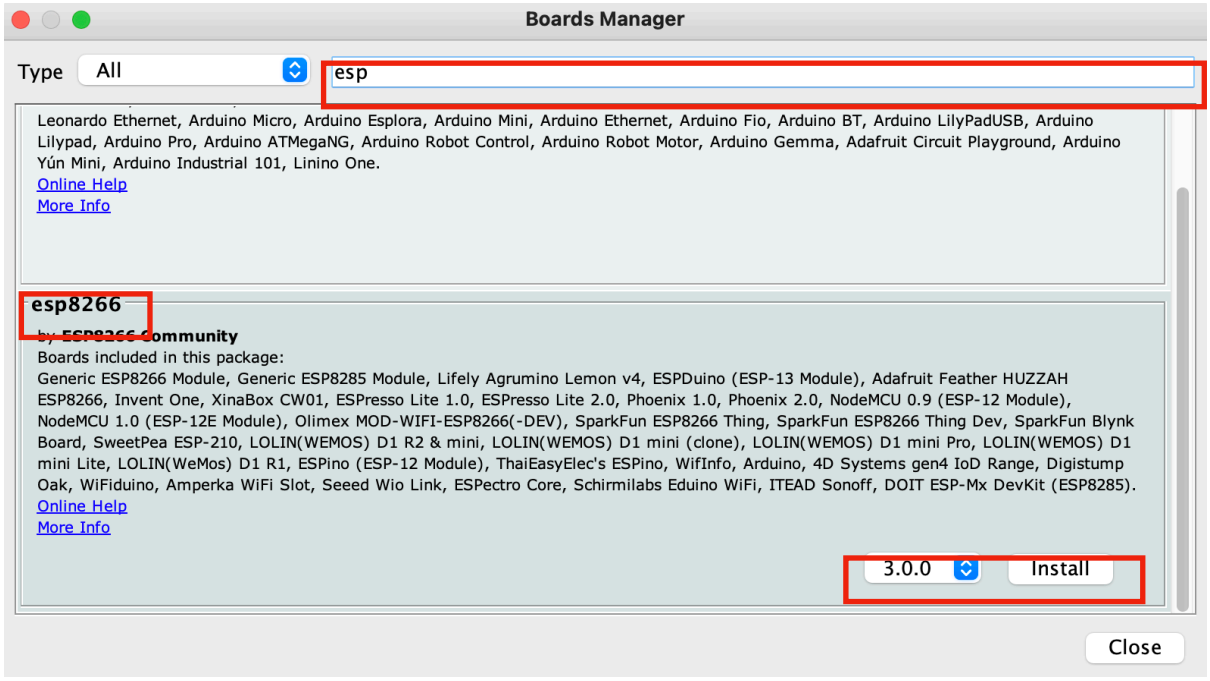
Click on the **OK** button to confirm the changes. This operation will install the drivers to also include ESP boards inside the Arduino IDE.

Note: Windows and Linux users might also need to install the following drivers: https://github.com/nodemcu/nodemcu-devkit/tree/master/Drivers

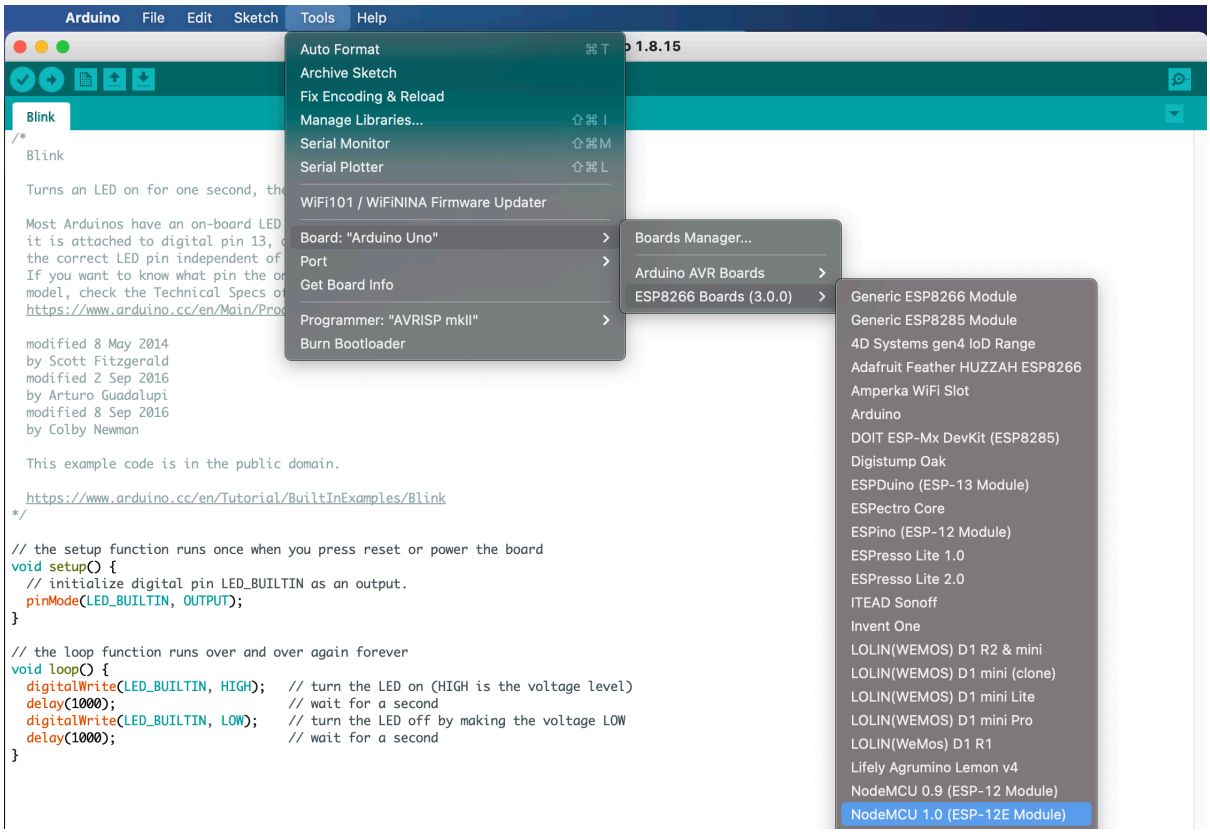
At this point use the main navigation bar and click on
Tools > Board > Boards Manager as shown on the image below:



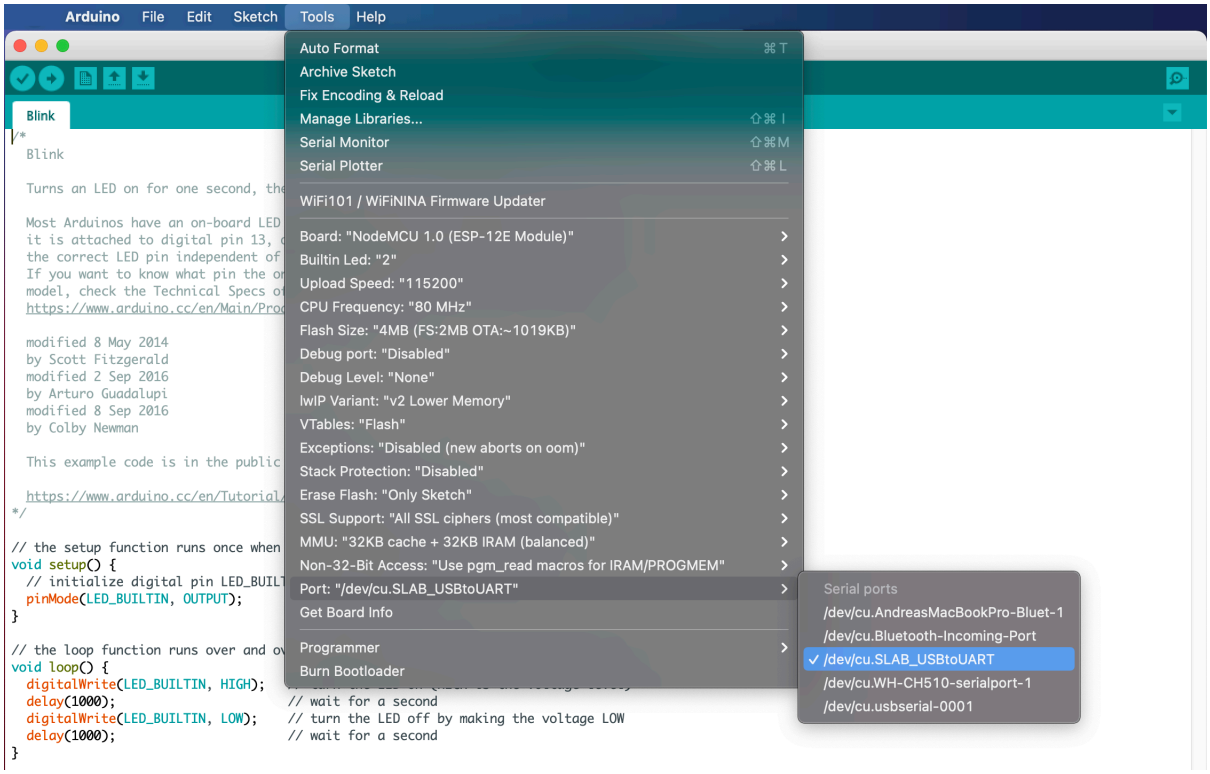
Type **esp** on the board manager search bar and install the latest version of the **esp8266** package:



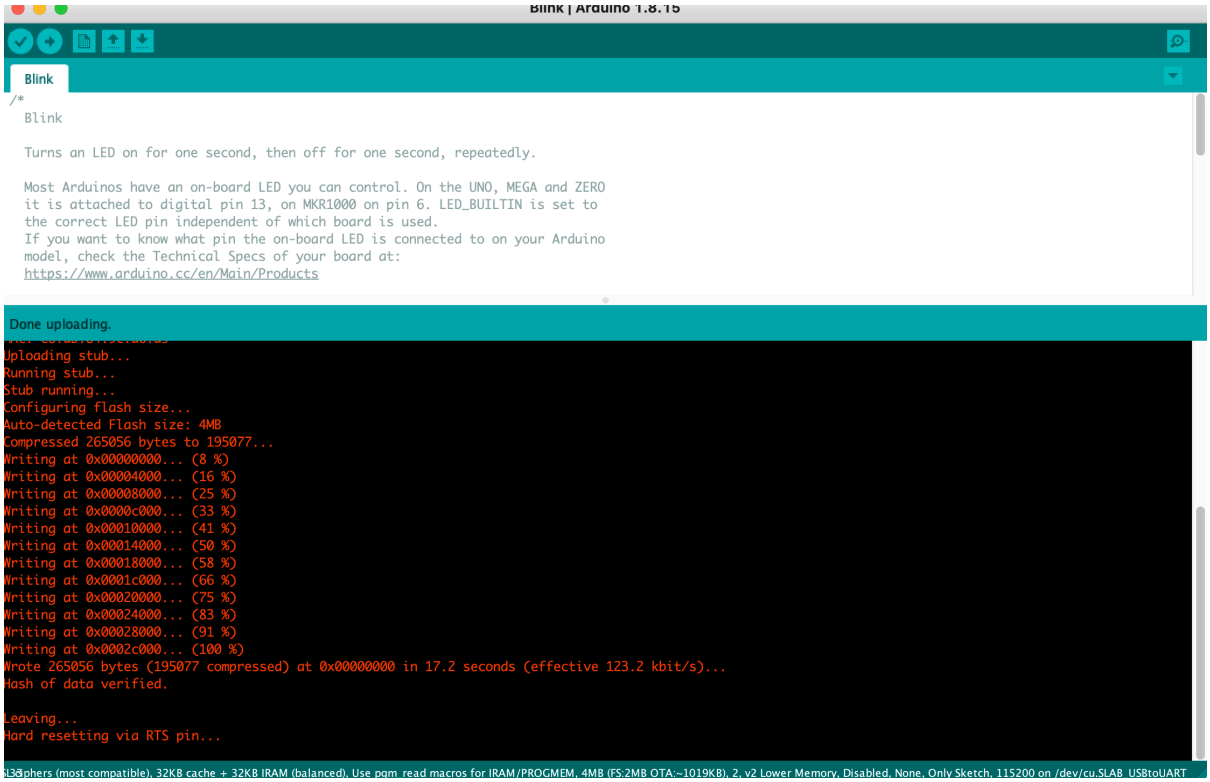
At this point you should be able to select ESP boards from **Tools > Board**. Go ahead and select the **NodeMCU 1.0 (ESP-12E Module)** board from the available list:



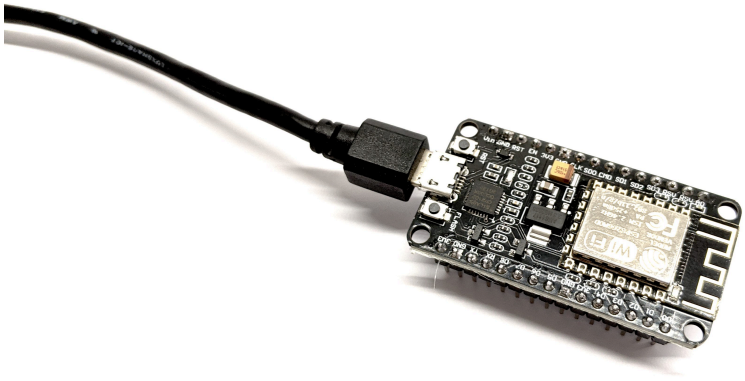
Finally, before you can upload the blinking sketch to your ESP board, you will need to select the correct USB port. You can do this from **Tools > Port**. Go ahead and select the USB port connected to your ESP board:



It is now time to upload the blinking sketch to your ESP board. Click on the **Upload** button and keep an eye on the console while your sketch gets uploaded:



Congratulations, you have uploaded your first sketch to the ESP board. You should see the built-in LED blink after every second.



Step Five
Additional Tasks

Now that you have successfully uploaded the blinking sketch to your ESP board, try to alter the code to modify the blinking behaviour.

The ESP board has 2 built-in LEDs connected to pin 2 and pin 16.
Try the following:

- Replace the **LED_BUILTIN** variable with the number 2. Upload the sketch. Does anything change? What about if you replace the variable with the number 16?
- Change the blink timing. Can you make the blink slower or faster?
  What about having the LED always on?
- Later in the course we will explore how you can upload a sketch to your board using the OTA (Over the Air) protocol. You might want to start doing some research on it.