# Background

This background sheet is for people who are not familiar with microprocessors or the Arduino environment.

## Digital inputs and outputs

The board has pins that can be set and read in your program. Each pin has an ID through which it can be accessed. What is a pins ID? Well - everything can be found from the NodeMCU 1.0 pinout seen in Figure 1!

Digital pins can be in two states, voltage or no voltage (0 or 1, thus digital). The ESP8266, also has one analog input/output, but it is not used in this workshop. In addition to digital and analog pins there are other pins as well. The two most important of these are the ground pins (GND) and voltage pins (3.3V).

Note that some pins are reserved. For example D0 is reserved for the "wake" function. Using reserved pins might cause extra trouble (mainly D0 and D3), so prefer unreserved pins.

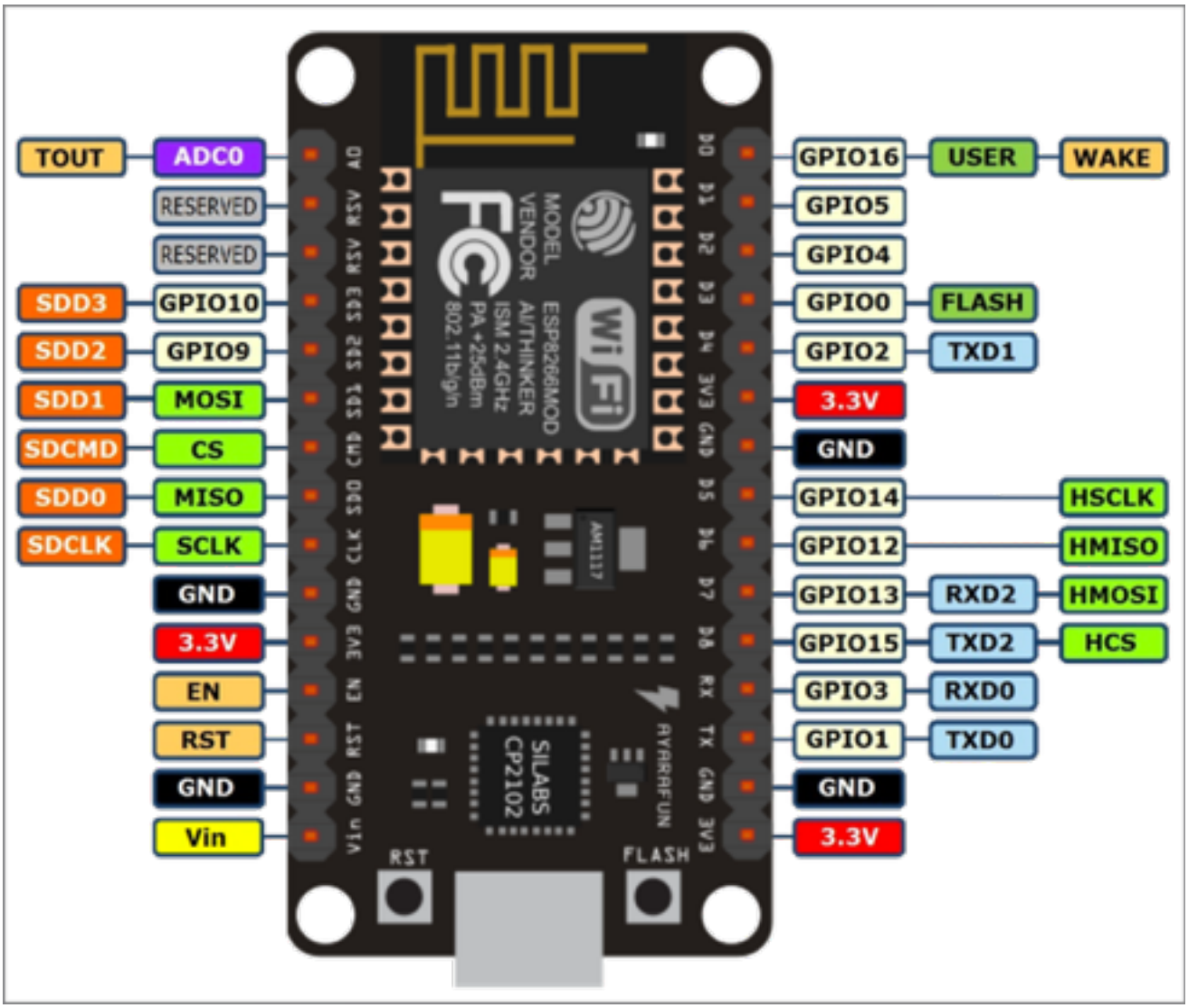



Figure 1. The NodeMCU 1.0 Pinout chart
https://bennthomsen.files.wordpress.com/2015/12/nodemcu_pinout_700-2.png?w=584

## So how does this work in your code?

First you tell your program that you are going to use a pin as input our output using the easy-to-memorize keywords INPUT and OUTPUT.
For example

```
pinMode(2, INPUT)
```

Not surprisingly, if you set something as INPUT, you can read from it. If you set it as OUTPUT, you can write to it. Writing is done with a function called digitalWrite and reading with digitalRead. Pins can be in two states, HIGH or LOW which correspond to either on or off. digitalRead returns an integer 1 or 0, which again corresponds HIGH and LOW.

An useful summary of these functions can be found in the Arduino documentation.
https://www.arduino.cc/en/Reference/pinMode

## A little more about coding

You have two default functions in the Arduino environment, setup and loop. Setup is called once, when the systems starts. After that, loop is called indefinitely.

Usually before the setup you do definitions, for example, you might want to name your pin numbers to be more memorable. Note that we use “#define”, which is used for static definitions. Defines are not actual variables.

```
#define redLed 2;
#define blueLed 4;
```

In the setup you set pinModes or do other one time only operations.

```
pinMode(redLed, OUTPUT);
pinMode(blueLed, OUTPUT);
```

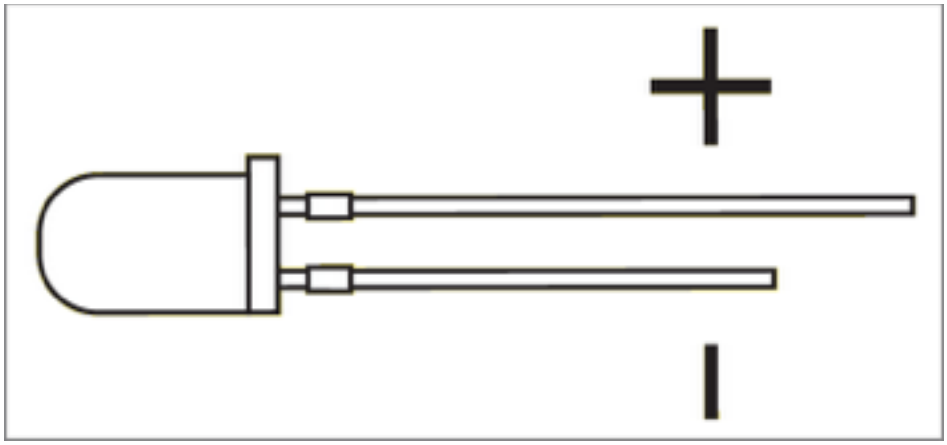
In the loop you have your main code

```
digitalWrite(redLed, HIGH);
delay(100);
digitalWrite(redLed, LOW);
delay(100);
```

This would result in a blinking led. Notice the new function - delay. Since the loop function is looped through indefinitely, it is useful to add small delays. In addition, the chip can do other tasks while your code is delayed, such as manage wireless connections.

## How to not destroy components

We'll be using components, for example leds, which actually care in which direction and how much current goes through them. This means if you put them the wrong way in they break. Connect the positive end to the digitalOutput and negative end to the ground.



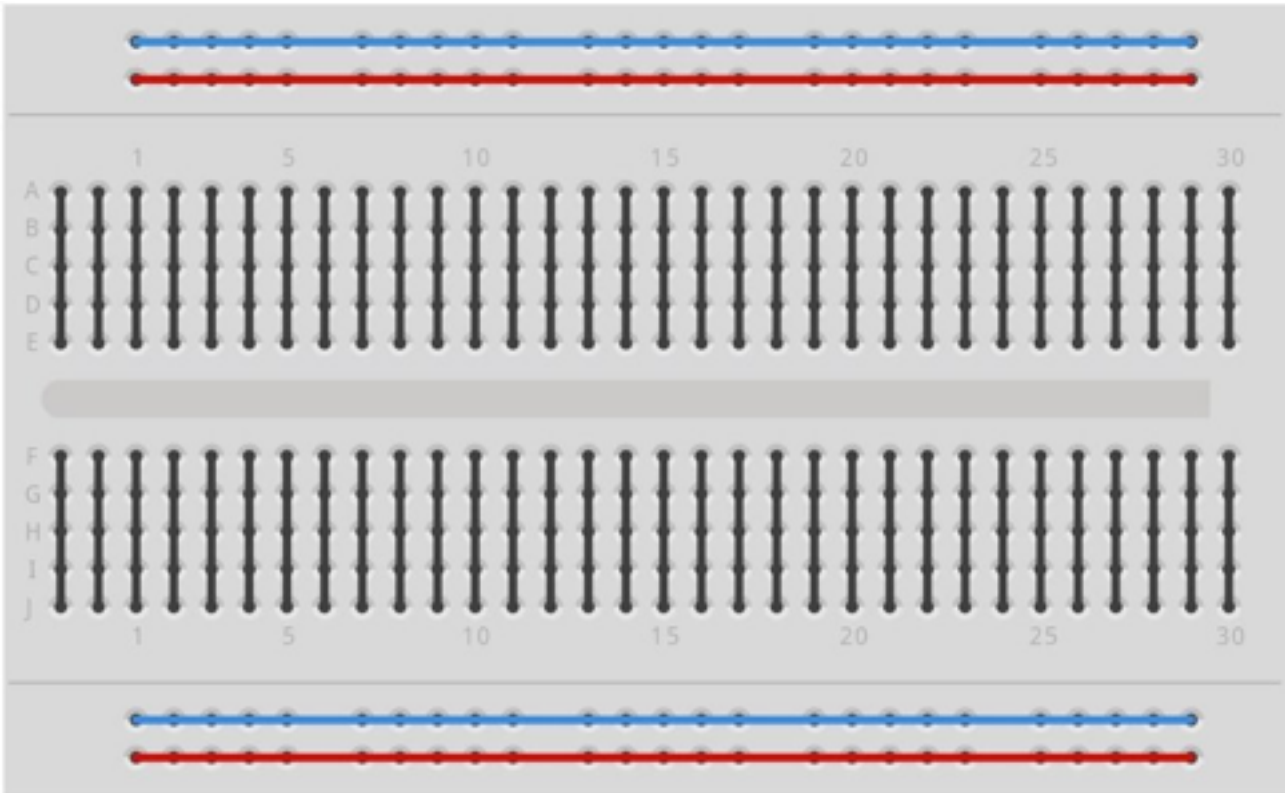
https://cdn.sparkfun.com/assets/0/c/5/d/a/518d2d78ce395f2675000000.png

To limit the current add a resistor to the circuit. So the simplest led test would look something like this
https://evothings.com/doc/examples/images/arduino-led-tcp-sketch.png

With this setup you can toggle your led on and off by changing the pin voltage (with digitalWrite). Alternatively, you could put a button in the circuit - put the digital output on, and switch the led on and off with the button.

## How to use the breadboard

The breadboard is used simply for convenience to replace wires. The breadboards helps you connect components to each other. To see how pins in the breadboard are connected refer to this image



**http://energia.nu/wordpress/wp-content/uploads/2015/07/Sidekick_Breadboard_Internal_Connections.jpg**