# Task 1

Connect the NodeMCU with a USB-cable to your PC. Open Arduino IDE and select the right usb port from Tools -> Port.

To confirm your setup is ok open a blink example from File -> Examples -> ESP8266 -> Blink. Then press the upload button (arrow in the left upper corner). If a blue led starts blinking slowly your setup is ready!

## Part A

In the background material it was described how to toggle a led in a circuit with a button. In task 1 however, we'll want to do this digitally. Read a button is and then toggle a led accordingly.

In the Arduino documentation you can find an extra pinMode called INPUT_PULLUP. That's very useful in this task. Just wire your button from a digital input to ground and use INPUT_PULLUP. This setting pulls the digital pin up to a HIGH voltage, but you can still read it. Additionally, if the pin is connected to the ground (such as when pressing a button) the voltage will drop to LOW.

## Part B

What if instead of a button you had a switch? You should be able to use the exact same code.

## Part C

What is really the difference between a button and a switch? Not much. Code your own switch from a button by having a toggle variable that is changed whenever the button is pressed.

Your button switch is probably a lot more buggy than the hardware switch, especially if you have low delay. To counter this you can implement debouncing

**Materials**
https://www.arduino.cc/en/Tutorial/Debounce
https://www.arduino.cc/en/Reference/pinMode

# Task 2

The first task was quite simple input and output. Next we look at sensors and more complicated output.

## Part A

We'll be using a DHT22 Temperature/Humidity sensor. Google DHT22 pinout or pins and connect your sensor accordingly. The data line can be connected to any digital pin.

Next you'll want to install a DHT library. The secret rule of coding is to never code anything if it already exists.

Go to Sketch -> Include Library -> Manage Libraries. Search for "DHT" click the first suggestion and select version 1.2.3. For whatever reason the Publishers have decided that 1.3 should be a version that doesn't work at all. So install the second to newest version, and open an example from

File -> Examples -> DHT Sensor Library -> DHTtester

Change the pin definition to your the digital pin you connected it to. Save the file with a different name since you can't overwrite examples.

Now upload your program and open the Serial Monitor from the Tools menu. You have a sensor!

## Part B

Connect your sensor to a more cool output.

Connect your sensor to different colored leds. You can use normal leds, or In your hardware stack you have a WS2812. Try to look for a library that could help you use it.

Connect the led to your Temperature / Humidity data.

**Materials**
https://learn.sparkfun.com/tutorials/ws2812-breakout-hookup-guide#adding-firmware

# Task 3

It's time to drop the wires. In this task we'll look into using Wifi and batteries.

## Part A

A good starting point is the esp8266 WifiClient example.

If your unsure how HTTP requests work you might wan't to look some background info. In simple cases you really don't need to know much though. A web address consists of two parts the host and the url. For example if we would do a request on the site http://google.com/search?q=http+requests google.com would be the host and /search?q=http+requests would be the URL. To send a get request in the simplect case we would send the following three lines to the host (just like in the Wifi example)

GET /search?q=http+request HTTP/1.1
Host: google.com
Connection: close

Configure the client to send you temperature measurements to a visualisation site we made at http://temperatureviz.azurewebsites.net

To submit data send a get request to the data site with id, value and label parameters.

For example you could submit the following url to your browser for testing.

temperatureviz.azurewebsites.net/data?id=TestID&value=22&label=1

id is the display name you choose. Choose an unique ID.
Value is the value you wan't to submit.
With label you can toggle you id label on and off by sending 0 or 1.

## Part B

In part we will not only make our Esp wireless but we'll also look into how make it reasonably energy efficient.

We'll do this by deepsleeping the node between measurements. You can use ESP.deepSleep() to achieve this. Remember to connect GPIO16 to the Reset pin so the ESP wakes back from the sleep.

Since WIFi concumes quite a lot of energy the ESP isn't an ideal wireless IoT sensor. However, you can achieve a reasonable demo efficiency and there are some hacks, such as removing leds, to make the battery last longer.

**Materials**
https://github.com/esp8266/Arduino/blob/master/doc/libraries.md#esp-specific-apis
http://blog.falafel.com/esp8266nodemcu-deep-sleep/