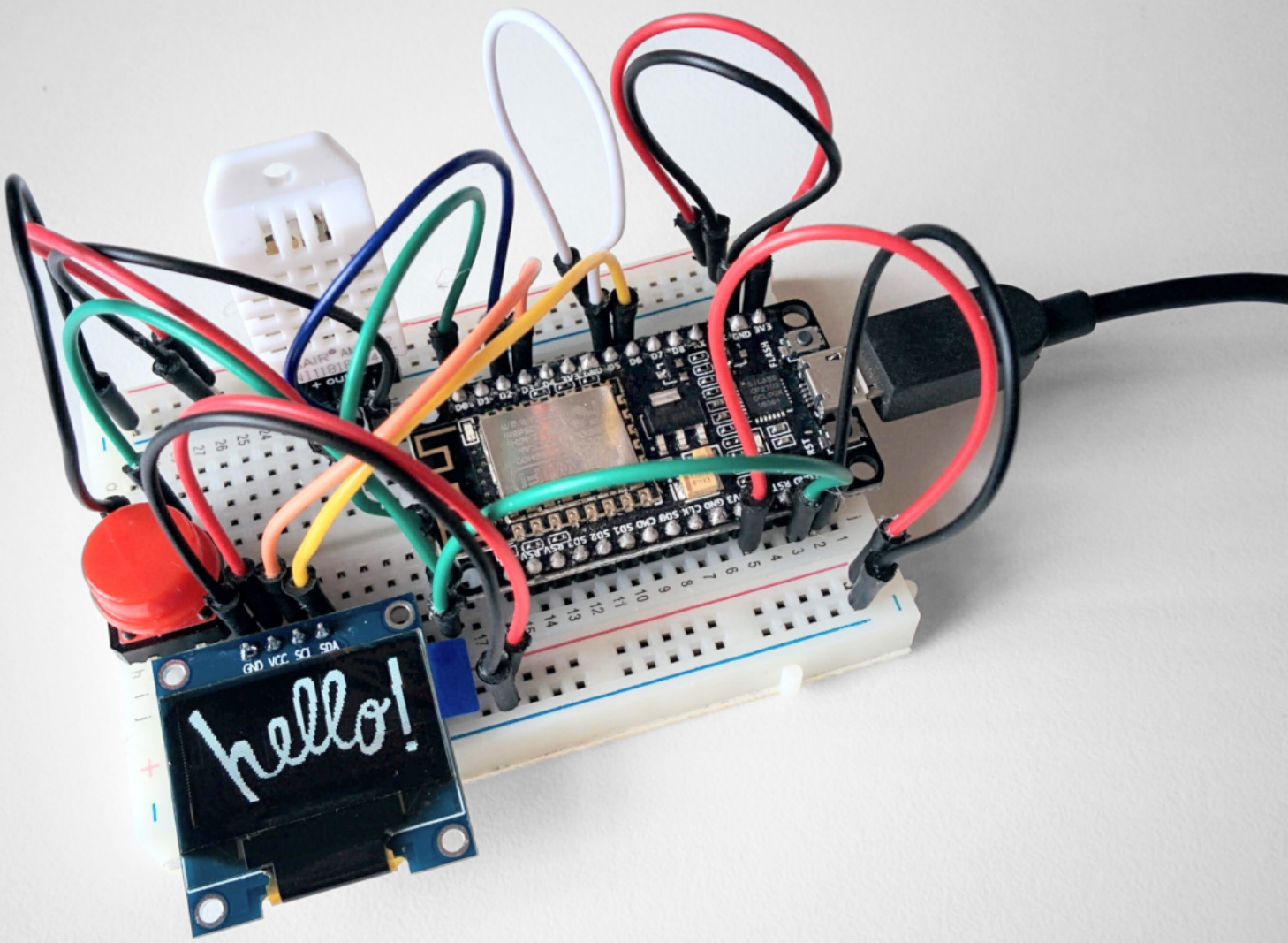
MicroPython

and the Internet of Things



Miguel Grinberg

MicroPython and the Internet of Things

Miguel Grinberg

2019-04-24

Contents

Preface

In 2018 I moved to a new house that was very cold. The heating controller in this house was one of the newest, as it gave me the option to see and change the heating schedules from my smartphone or computer. Yet, I found that I constantly needed to make manual adjustments to the heating schedule because I was cold. After an investigation I determined that the thermostats installed in the house were very inaccurate, often reporting a temperature that was a few degrees higher than what I measured with my own thermometer. Because I could easily start and stop the heating by sending HTTP requests to the heating controller, I decided to build my own thermostats and use them instead of the stock ones. This was the excuse that I made to learn MicroPython!

I built two Wi-Fi thermostats for less than $10 USD each and installed them in the two levels of the house, and that solved my heating problem without having to make any major modifications in a house I did not own. I hope after you learn how to work with MicroPython you will also come up with ideas for little devices that can improve your life in some way.

1 Who This Book Is For

This book is for curious developers interested in learning how to create small Internet enabled devices, in the so called "Internet of Things" category. The book is written in tutorial form, with each chapter introducing new concepts that build upong the previous material.

2 Requirements

The first chapter of the book includes a "shopping list" of hardware components that you need to build the examples featured in this book. In addition to that, you will need a computer which will be used to program the microcontroller. You can use any Windows, Mac OS or

Linux computer, as long as it has one available USB port where the microncontroller can be connected.

3 How To Work With The Example Code

I have released the complete source code for this book on a GitHub repository[1]. In general you will not need this repository as the entire source code for all the examples is shown in the videos and written articles, but having the complete code examples in a centralized location can help you find mistakes in your own code. Also, in the last chapter of the book I am using some images files and third-party code that you will be able to download from this repository.

4 Conventions Used In This Book

This book frequently includes commands that you need to type in a terminal session. For these commands, a **$** will be shown as a command prompt. This is a standard prompt for many Linux shells, but may look unfamiliar to Microsoft Windows users. For example:

```
$ python hello.py
hello
```

In a lot of the terminal examples, you are going to be required to have an activated *virtual environment* (do not worry if you don't know what this is yet, you will find out very soon!). For those examples, the prompt will appear as **(venv) $**:

```
(venv) $ python hello.py
hello
```

You will also need to interact with the MicroPython REPL, the interactive Python prompt. Examples that show statements that need to be entered in a Python interpreter session will use a **>>>** prompt, as in the following example:

```
>>> print('hello!')
hello
```

[1] https://github.com/miguelgrinberg/micropython-iot-tutorial

In all cases, lines that are not prefixed with a `$` or `>>>` prompt, are output printed by the command, and should not be typed.

Chapter 1

Welcome

1.1 About This Tutorial

This is a tutorial for Python beginners who want to learn to program devices to interact with the physical world and with the so called "Cloud". You are going to learn how to program with MicroPython[1], a version of the Python language designed to run on microcontrollers[2]. The applications that you are going to learn how to write are going to read data from sensors, display information on small screens, react to the push a button, and upload or download data from the Internet. Lots of cool stuff!

This tutorial is focused on the software side more than on the hardware side. I'm going to use a high-level "Lego" approach for building circuits, so instead of having to solder components into circuit boards, you'll be building your experiments on breadboards[3], and interconnecting components with jumper wires[4]. You will be able to assemble and disassemble your circuits with ease, and more importantly, reuse components as you move through the tutorial and later when you build your own projects.

[1] https://micropython.org/
[2] https://en.wikipedia.org/wiki/Microcontroller
[3] https://en.wikipedia.org/wiki/Breadboard
[4] https://en.wikipedia.org/wiki/Jump_wire

1.2 Requirements

To be able to follow this tutorial you just need to have basic coding experience with Python. You do not need to have any previous knowledge of microcontrollers, electronics, or even MicroPython.

You will also need a Windows, Mac or Linux computer with a free USB port, as you will connect a microcontroller to your computer to program it. If your computer is one of the newer ones that only come with USB-C ports, then you are going to need an adapter so that you can connect a regular USB cable.

1.3 The Shopping List

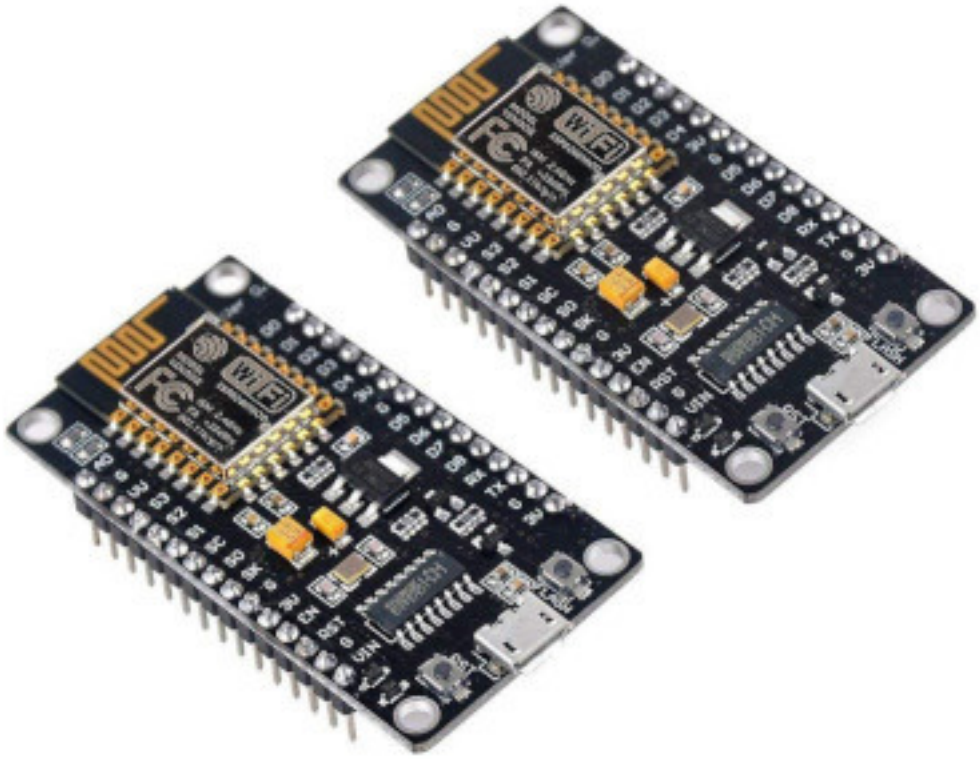
In this section I will give you a short list with the components that you are going to need to build all the examples featured in this tutorial. As a reference, I'm providing links to Amazon US and UK for all of them (disclaimer: my Amazon affiliate link is embedded in these links), but you do not need to buy the exact products I'm linking. There are many manufacturers for these components, so use the links I provide as a reference and then if you prefer, buy your components from your favorite retailer. If you want a recommendation other than Amazon, Ebay is also a great place to shop for electronic components, and you'll probably find better prices.

1.3.1 Microcontroller

MicroPython runs on a few different types of microcontrollers, but for this tutorial I'm going to work with just one model: the ESP8266[5]. Note that there are a few different boards that you can buy with the same chip. The model that you want to get is the one informally referred to as "development board", and more formally known with the ESP-12 model name. These boards come with the microcontroller mounted on it, a small printed Wi-Fi antenna, a micro-USB input for power and programming, and 30 pins that insert straight into a breadboard.

The cost of these devices is so low that you are going to find that most sellers offer them in pairs or in four-packs. I recommend that you do order at least two.

[5] https://en.wikipedia.org/wiki/ESP8266

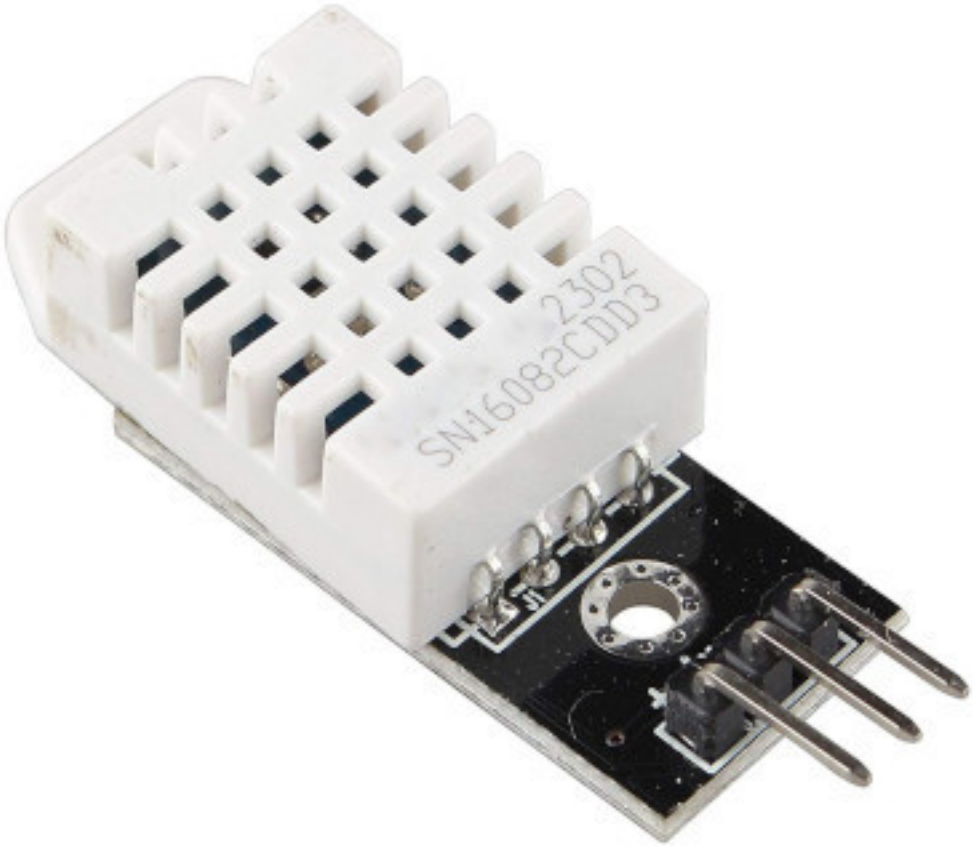


Buy from Amazon US[6] - Buy from Amazon UK[7]

1.3.2 Temperarure and Humidity Sensor

To demonstrate how to work with sensors, I'm going to use a DHT22 temperature and humidity sensor. This little device reads and reports the temperature and humidity levels in the environment. Depending on where you buy it, you may find models that come with three or four pins. Both are fine for this tutorial.

[6] https://amzn.to/2CKnDXN
[7] https://amzn.to/2U9A3SA

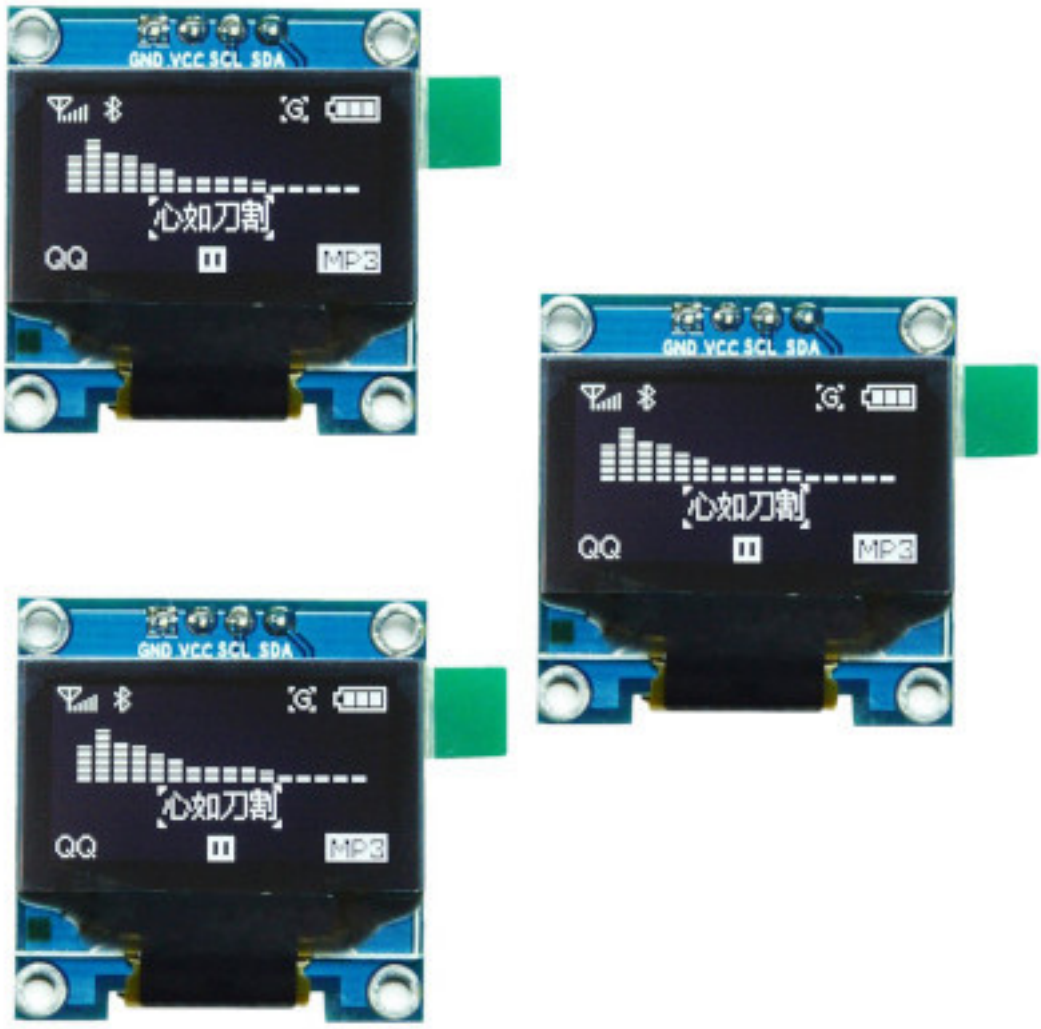


Buy from Amazon US[8] - Buy from Amazon UK[9]

1.3.3 Screen

Some of the examples will display information on a small 128x64 pixel OLED screen. Once again, these are so cheap that you may want to get a few.

[8] https://amzn.to/2Tq2CqO
[9] https://amzn.to/2OkjbU4



Buy from Amazon US[10] - Buy from Amazon UK[11]

1.3.4 Push Button

Some of the examples will be triggered by a button push. You are not going to need more than one button at a time for this tutorial, but as with the above you may want to also get a pack, since buttons are always handy to have.

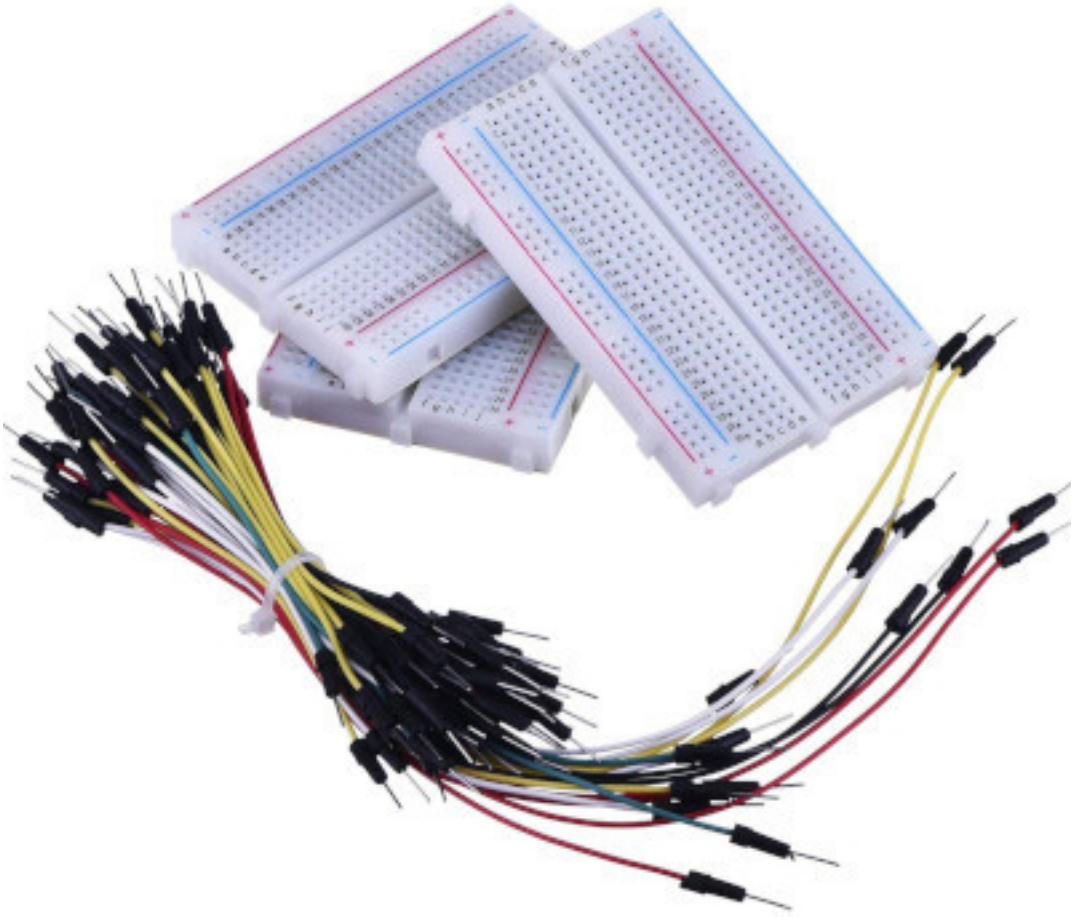
[10] https://amzn.to/2Ygzztq
[11] https://amzn.to/2TWUlQ3

Buy from Amazon US[12] - Buy from Amazon UK[13]

1.3.5 Breadboard and Jumper Wires

To be able to easily build and take apart your circuits without soldering, you are going to need a breadboard and some jumper wires. I recommend that you get more than one breadboard as well.

[12] https://amzn.to/2OiPKS8
[13] https://amzn.to/2U6zWrc



Buy from Amazon US[14] - Buy from Amazon UK[15]

1.3.6 USB to MicroUSB Cable

You will need a USB cable to connect your microcontroller to your computer. This is going to provide power, and also a serial connection through which you'll upload code into the microcontroller memory. This is the same cable that charge most Android phones, so you may already have one you can use.

[14] https://amzn.to/2CCcjww
[15] https://amzn.to/2OiTTFG

Buy from Amazon US[16] - Buy from Amazon UK[17]

1.3.7 Power Bank (Optional)

A power bank is not required for this tutorial, but if you have one, you will be able to use it to power your microcontroller independently of the computer after you have uploaded your code.

1.4 Let's Build Some Stuff!

I hope you find this a fun and interesting tutorial. Once you have the materials listed above you are ready to move on to the next chapter and start building stuff!

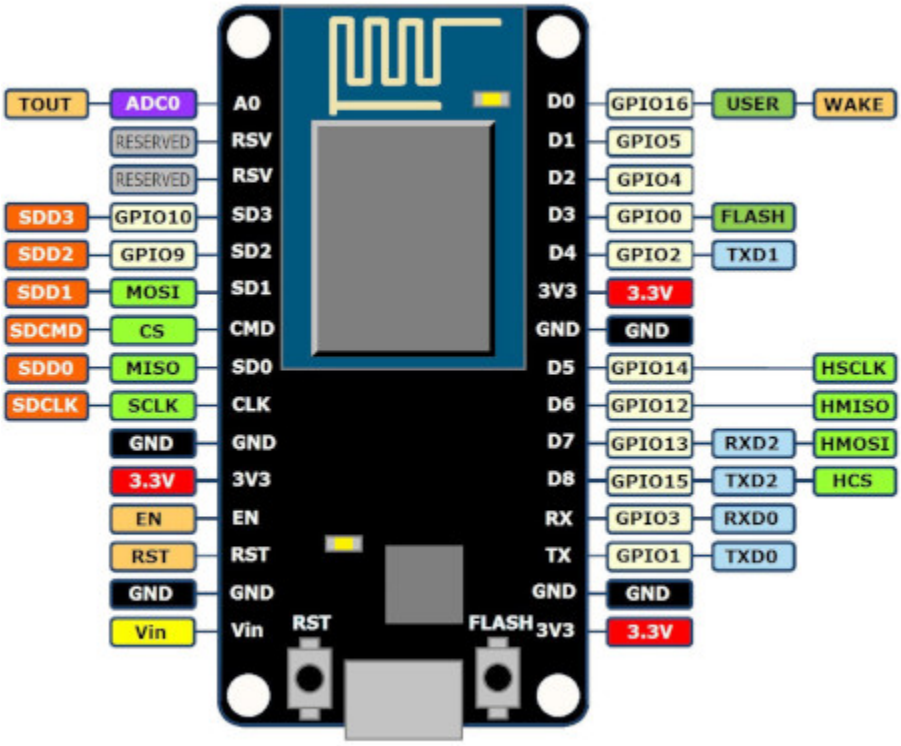
[16] https://amzn.to/2YgEpqt
[17] https://amzn.to/2JwgCQ1

Chapter 2

Hello, MicroPython!

2.1 The ESP8266 Microcontroller Board

To begin, I want to give you a very brief overview of the components in your ESP8266 microcontroller. Below you can see a diagram that should more or less match your board (though maybe not exactly, since this board is open source and there are a lot of different manufacturers):



The ESP8266 microcontroller chip is the bigger silver colored box. Above it you can see a zig-zag gold line. That is the Wi-Fi antenna. You can see 15 pins on each side, each with a label that should be printed on the board, with names such as `D0`, `D1`, `3V3`, `GND`, etc. To make matters confusing, the diagram shows that all the pins have other labels shown on the outside, usually associated with their function.

There are three pins labeled `3V3` or `3.3V`, which provide electric current. The name derives from the amount of current, which is 3.3 volts. These pins can be used to deliver input current to other components that are part of your circuit. There are also four `GND` or "ground" pins. These provide a ground connection, which is required to close the circuit when you connect a component. Electric current always flows towards the ground, so when you connect a device to the microcontroller you will be making two actual connections. A connection from a `3.3V` pin will deliver current to the component, and a connection from the component to a `GND` pin will close the circuit, effectively forcing the current to flow through the component. As you will learn later in this tutorial, all components have a clearly marked pin for input current typically called `VCC`, and one for connection to ground called `GND`.

The pins labeled with a **D<n>** are data pins. These are the pins that are used for communication between the microcontroller and other connected devices. This is going to be covered in later chapters of this tutorial.

The board has two small built-in LEDs (Light Emitting Diodes[1], or just tiny light bulbs if you prefer) that you can see in yellow in the diagram. The top LED is connected to the pin labeled **D4** or **GPIO2**, while the bottom LED is connected to **D0**, which is also **GPIO16**. By the end of this chapter you will know how to turn the LEDs on and off with Python code!

Out of the remaining pins, the **A0** is interesting to mention because it is the only "analog" pin in the board. All the other pins can have a binary value, either 0 or 1, but the analog pin can have a range of values, determined by the amount of electrical current passing through the pin. Note that this tutorial does not make use of the analog pin.

Another interesting pin is **RST**, which can be used to send a reset signal to the board. In future chapters you are going to learn how to use this pin to "wake" the microcontroller when it is in a deep sleep state.

The pin labeled **Vin** can be used to provide power to the board, as an alternative to the much more convenient micro-USB port. All the examples you are going to build in this tutorial are powered through the micro-USB port, so you are not going to use this pin.

The remaining pins are in most boards reserved for the microcontroller's own use, so there isn't a lot of interesting features to mention there. If you are curious about any use you can give them in your particular board, you should consult the documentation that is specific to it, as these pins are not always wired in the same way.

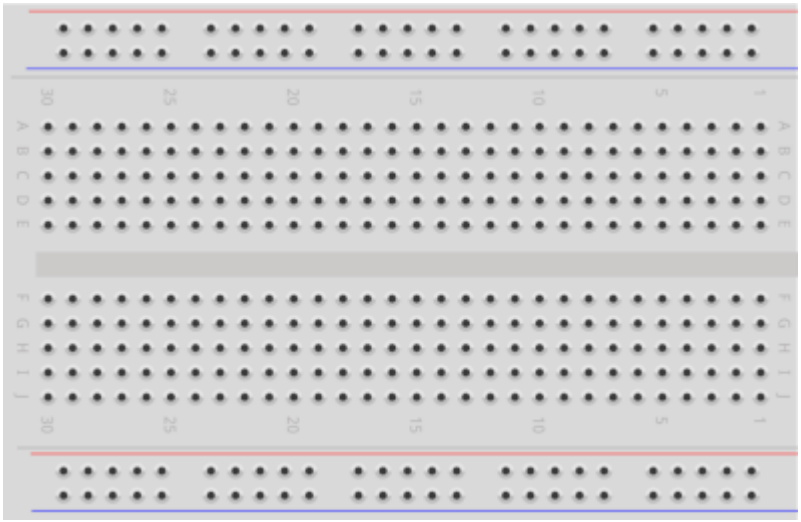
In the bottom center you have the micro-USB input connector. In this tutorial you will always power the board through this connector. If you connect the other end of the cable to your computer, then the board is going to appear as a serial device in your computer. This serial connection is going to be used to upload code into the microcontroller.

To complete this overview, note that the board has two small buttons labeled **RST** and **FLASH**, on the sides of the micro-USB port. The **FLASH** button prepares the board to be flashed with a new firmware, but this button is rarely needed, as current versions of the board can be put in flash mode through the serial connection. The **RST** button is, as I'm sure you can guess, a plain and simple reset button, and pressing it restarts the board.

[1] https://en.wikipedia.org/wiki/Light-emitting_diode

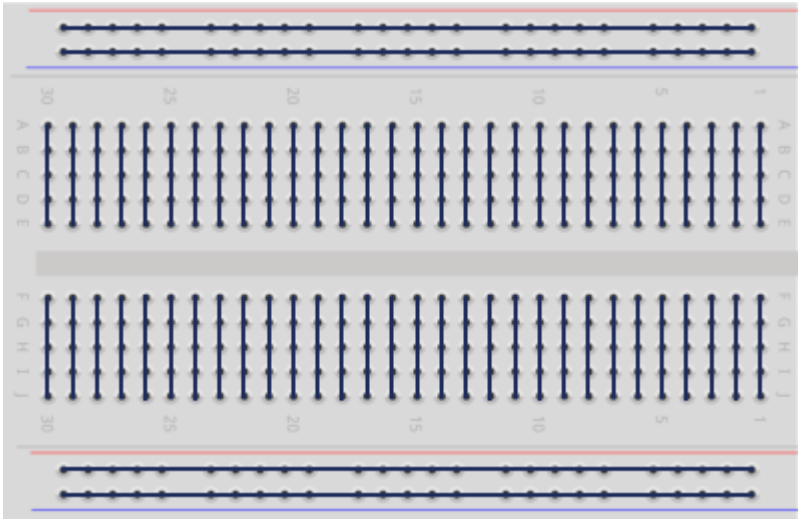
2.2 The Breadboard

The first task that I'm going to show you how to do is to set up the microcontroller board on a breadboard. So first, let me tell you how breadboards work. Here is a diagram that should match the breadboard that you have:



The key to understand how to work with a breadboard is to visualize the internal connection between the holes. Once you know which holes are internally connected, you know that you can create a connection between two pins by inserting them in a pair of connected holes. It's really that simple.

Below I created a diagram that shows these internal connections between holes:



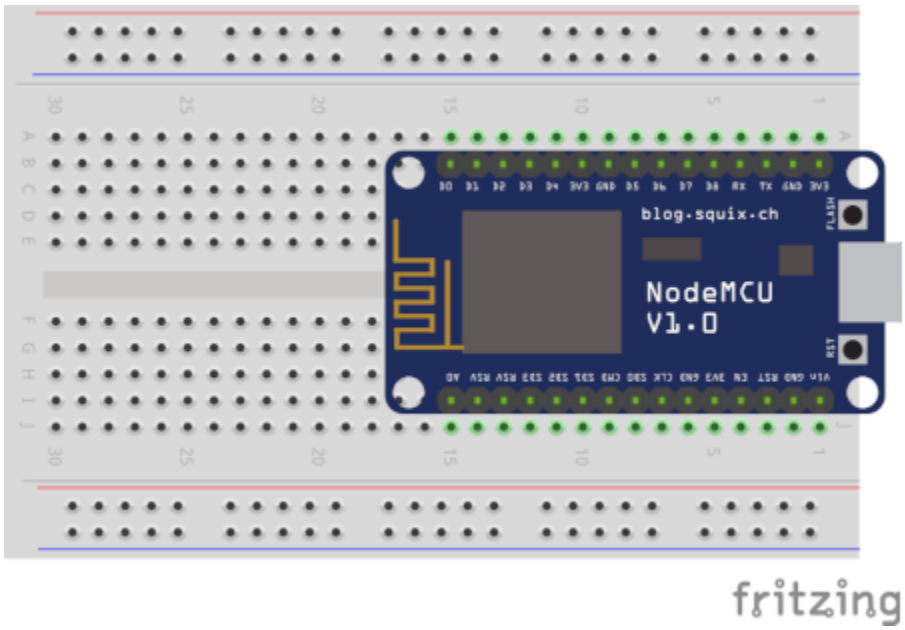
The top two and top bottom rows of holes are usually referred to as the "power strips", and they actually work in a similar way to a household power strip. For example, when you connect a pin that delivers current (such as any of the `3.3V` pins in the ESP8266 board) into any of the holes in the top row, then all the remaining holes in that row can be used by other devices to draw power for themselves, so effectively this is equivalent to having a direct connection between each device and the power source. This top line is marked with a red line and in some breadboards with a "+" label, which indicates that as a convention, it should be used for current, as in my example. The second row of holes is marked in blue and with a "-" label. The convention is to use this line as a ground connection. There are two more positive and negative power strips at the bottom of the breadboard.

The connections used by the wholes in the middle are more tricky to understand. Here the holes are grouped in fives. The ten rows of holes in the two middle sections are labeled with the letters `a` through `j`, but unlike the power strip rows, these rows are not connected. The connections for these pins are vertical, and cover just the group of five vertical holes in each section. The columns are labeled from `1` to `30`, but in many breadboards you can only see a label every five columns. Using these labels, you can see that the `a1`, `b1`, `c1`, `d1`, and `e1` holes are all connected with each other, so any pins that are plugged into these holes will be all connected. Similarly, pins `f1`, `g1`, `h1`, `i1` and `j1` are connected between them, but they do not share a connection with the top five holes.

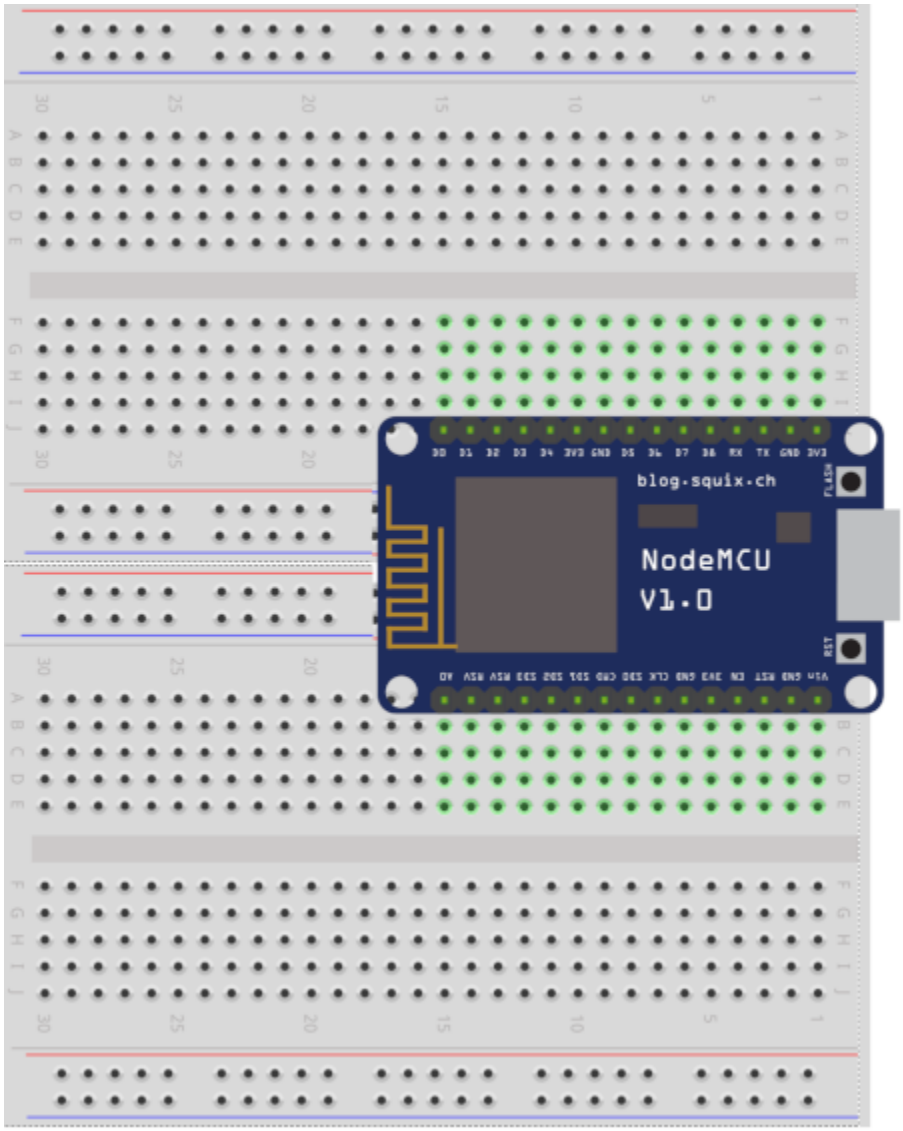
If you think this is starting to make sense, you are ready to install the microcontroller into the breadboard. Depending on the physical dimensions of your microcontroller, there are two possible ways to do this.

Most likely you have a newer microcontroller board, which is the smaller kind. To find out for sure, place the breadboard on your desk with a red power strip on top and column 1 to

the right, and then see if you can align your microcontroller in the middle section, with the micro-USB port facing right. Do not insert it into the breadboard yet. The top 15 pins from the microcontroller should be in the `b` row of your breadboard, columns 1 on the right to 15 on the left. The bottom strip of microcontroller pins should be aligned with the `i` row. The `a` and `j` rows should be free and visible from the top. Here is a diagram:



If you were able to align the pins as shown in the diagram, then gently press on the board until the pins are fully inserted into the holes. If your microcontroller is too large to fit in the way I described, then you have one of the bigger boards. These work in exactly the same way as the newer counterparts, but have slightly larger dimensions, which means that there is no way to fit them in the middle section of the breadboard while leaving at least a row on each side to make connections. The solution that I have used for these boards is to straddle them across two breadboards, as show in the diagram below:



With this set up, the top row of pins in the microcontroller board is in row `j` of the top breadboard, and each pin has four free holes to make connections. The bottom row of microcontroller pins fits in the `a` row of the bottom breadboard, also leaving four holes for connections to each pin.

In the diagrams that follow I'm going to assume you have the newer and smaller board installed on a single breadboard, since the older boards are mostly out of circulation by now.

2.3 Wiring the Breadboard Power Strips

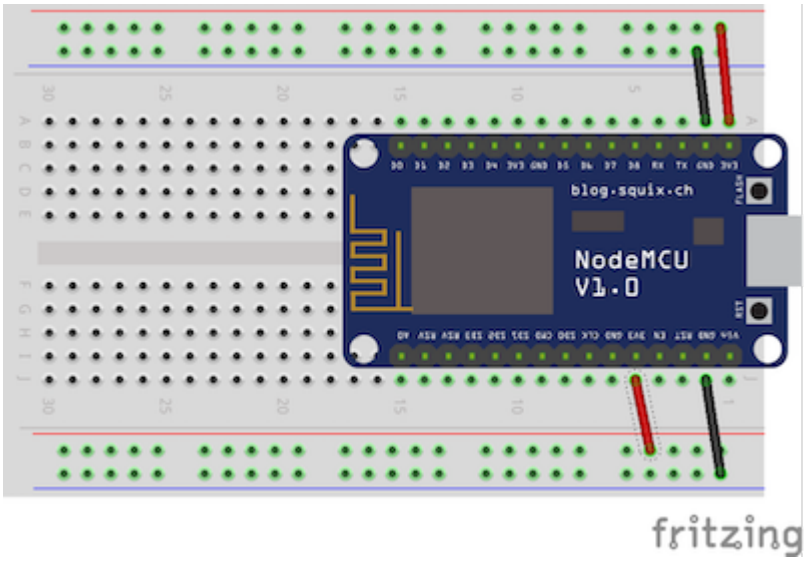
The next task is to set up the delivery of power to the breadboard. I explained above that the top and bottom two rows of holes are used to deliver current and ground to other devices. While in this chapter there are no additional devices that need to make use of power, it is always a good idea to have the power properly distributed on the breadboard for when it is needed.

To do this you are going to need four jumper wires. If you have different kinds of wires, make sure the ones you select have all male pins. If you have wires of several colors available, you may want to use two red wires for the positive connections, and two black or blue wires for the ground.

The connections that you need to make are four:

- From one of the top `3V3` pins (for example, the `a1` hole) to any hole in the breadboard's top row
- From one of the top `GND` pins (for example, the `a2` hole) to any hole in the breadboard's second row from the top
- From one of the bottom `3V3` pins (for example, the `j5` hole) to any hole in the breadboard's second row from the bottom
- From one of the bottom `GND` pins (for example, the `j2` hole) to any hole in the breadboard's bottom row

The end result should look more or less like the following diagram:



In case this isn't clear, no other component is drawing power from the microcontroller at this time, so these connections that supply power and ground to the breadboard aren't really doing anything yet, but it's nice to have the breadboard in a state that is ready to support additional components later on.

This is enough work in the physical world for this chapter. Now I'm going to take to the software side and show you how to prepare your computer so that you can install and run MicroPython on your board.

2.4 Setting Up Your Computer

If you have a Linux computer, then you do not need to install any device drivers for the microcontroller to be recognized. But if you have a Mac or a Windows machine, a driver is needed to allow the computer to recognize the microcontroller as a serial device. To download the driver installer package visit this page: https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers[2].

The tools that you are going to use to communicate with the ESP8266 are written in Python, so what you need to do now (if you haven't yet) is install Python on your computer. I have tested this tools with Python 3.7, which is the current release of Python as I'm writing this. If your operating system does not provide a pre-packaged Python, you can go to https://python.org[3] to

[2] https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

[3] https://python.org

download an official build for any of the supported operating systems.

Start by opening a terminal window. On Linux or Mac I will assume that you are running a **bash**, **zsh** or similar shell, while on Windows it will be the Command Prompt window. First create a new directory where you are going to store files associated with this tutorial:

```
$ mkdir micropython-tutorial
$ cd micropython-tutorial
```

In all the command that you are going to see in this tutorial I'm going to use the **$** as an indication that you are in the command prompt. The **$** is not part of the command.

The next step is to create a Python virtual environment. This is the recommended way to install packages, so that they are private to your project instead of installed system-wide. You can create a virtual environment with the following command:

```
$ python -m venv venv
```

This command is asking Python to run the **venv** package (the first **venv**) and create a virtual environment named **venv** (the second **venv**). If you want to use a different name for your virtual environment, replace the second **venv** with name you want to use. After the command completes, you should see a subdirectory with this name. Inside this subdirectory there is a private copy of the Python interpreter.

Note that if the command above does not work, it is possible that in your system the Python interpreter is called **python3** and not **python**. In that case, the command should be:

```
$ python3 -m venv venv
```

Now you are going to activate this new virtual environment. The virtual environment activation configures the Python interpreter installed inside the virtual environment as the currently active Python that is invoked when you type **python** in the command line. This activation is temporary, by the way, nothing in your system is modified. The command to do the activation is different depending on the operating system. If you are using Linux or Mac, the command is:

```
$ source venv/bin/activate
(venv) $ _
```