
Gizmo Documentation

Release 0.1

BSG et al.

Jul 05, 2019

Contents

1	Syllabus	3
2	Contents	5
2.1	Introduction	5
2.1.1	Preparing your Computer	5
2.1.2	Intro to Git & GitHub	5
2.1.3	Warmup for Python	8
2.2	Arduino	18
2.2.1	Introduction to Arduino	19
2.2.2	Basics	29
2.2.3	Sensors	36
2.2.4	Actuators	42
2.2.5	Combined Sense & Actuation	46
2.2.6	Why Arduino?	64
2.2.7	Alternative Microcontrollers	65
2.3	Raspberry Pi	65
2.3.1	Assembling Pi workstation	65
2.3.2	Setting up your SD Card	71
2.3.3	Configuring the Pi	72
2.3.4	Headless Setup	84
2.3.5	Accessing Networks	84
2.3.6	Software	87
2.3.7	Connecting Remotely	88
2.3.8	Backing up your SD card	104
2.3.9	GPIO	105
2.3.10	Using Peripherals	115
2.3.11	Why Raspberry Pi?	123
2.3.12	Alternatives to the Raspberry Pi	123
2.4	Communication	123
2.4.1	Protocol: Serial	123
2.5	Sensors & Actuators	127
2.6	Supplementary Material	127
2.6.1	Crontab - scheduling commands	128
2.6.2	Material to be added	129
2.6.3	Useful links	129
3	Missing Material	131

Welcome to the Gizmo documentation for 2018.

This new format is designed to be more generalised to support the electromechanical work which happens in many projects across the Design Engineering undergraduate degree.

Where possible, all contributions and acknowledgements will be put on pages as external links to the source. For more general acknowledgements, they will be put here rather than trying to find all the specific places in the documentation.

- [Syllabus](#)
- [Contents](#)
- [Missing Material](#)

CHAPTER 1

Syllabus

Class	Date	Contents
DE2 Gizmo Class 4	Wednesday 17th October 2018	<ul style="list-style-type: none">• <i>Introduction to Arduino</i>• <i>Basics</i>• <i>Sensors</i>• <i>Actuators</i>
DE2 Gizmo Class 6	Wednesday 24th October 2018	<ul style="list-style-type: none">• <i>Combined Sense & Actuation</i>
DE2 Gizmo Class 8	Wednesday 24th October 2018	<ul style="list-style-type: none">• <i>Raspberry Pi</i><ul style="list-style-type: none">– Chapter 1– Chapter 3– Chapter 5.2– Chapter 6– Chapter 7.1– Chapter 8
DE2 Gizmo Class 10	Wednesday 14th November 2018	<ul style="list-style-type: none">• <i>Raspberry Pi</i> Chapter 8

2.1 Introduction

This section of the documentation collates a number of useful topics to learn for the Gizmo project, as well as for other projects later in the degree. We recommend you take your time to read and follow the examples/tutorials for each of these as they become necessary in your projects.

2.1.1 Preparing your Computer

Todo: Guidance for managing installs and software on personal computers needs to be written here.

macOS

Brew

Windows

Linux

Text editor

Atom / VSCode / Sublime

2.1.2 Intro to Git & GitHub

What is a VCS?

A version control system (VCS) is one that allows the author to track changes to a file over a period of time. It timestamps information and allows multiple copies of the same work to be compared and merged together. Nowadays ‘git’ is the go-to VCS for software and many other types of media.

Understanding Git

To understand Git, there are a number of good tutorials out there to get you started. If you feel like you want to learn from the source material, you can [check that out here](#). However we recommend that you learn from the videos produced by The Coding Train.

Tip: If you need a complete explanation on all the different aspects of Git - he has also created a complete playlist of videos explaining it all.

Overview

For a quick overview, we recommend you check out **‘just a simple guide for getting started with git<<http://rogerdudler.github.io/git-guide/>>’**.

Cheatsheet

If you need a quick cheatsheet for Git commands, [Tower has a pretty good one put together](#)

The following graphic, also made by Tower, is a guide to how the Git workflow works:



Understanding the Workflow of Version Control

presented by TOWER > Version control with Git - made easy



The Basics

1

Start a New Project

```
$ git init
```

Executing the "git init" command in the root folder of your new project creates a new and empty Git repository. You're ready to start getting your files under version control!

Work on an Existing Project

```
$ git clone <remote-url>
```

The "git clone" command is used to download a copy of an existing repository from a remote server. When this is done, you have a full-featured version of the project on your local computer – including its complete history of changes.

2

Work on Your Files

Modify, rename and delete files or add new ones. Do all of this in your favorite editor / IDE / file browser – there's nothing to watch out for in this step!



File Status



Files that aren't yet under version control are called "**untracked**"...
...while files that your version control system already knows about are "**tracked**" files.



A tracked file can either be "**unmodified**" (meaning it wasn't changed since the last commit)...
...or "**modified**" (meaning it has local changes since it was last committed).



Using a graphical interface for Git

There are a number of programs that you can use to visually interact with your Git repository. This makes working with Git *very easy*.

- [GitHub Desktop](#) (free)
- [Sourcetree](#) (free)
- [GitKraken](#) (free)
- [Tower](#) (discounted for students)

GitHub

GitHub and Bitbucket are the two biggest public services for storing VCS repositories. We recommend you use the official [GitHub guides](#). These will help you transistion your understanding of using Git locally, into using GitHub to host your files.

2.1.3 Warmup for Python

It is expected that you develop you Python skills within the Computing 1 module of your first year. You will also be completing your Computing 2 module alongside Gizmo and as such we don't go into too much detail on how the language works - you're expected to know that! What follows is a brief refresher, but in most cases we heavily recommend a quick Google search if you've forgotten certain Python syntax.

About Python Language

Remember that you are intelligent, and you can learn, but the computer is simple and very fast, but can not learn by itself. Therefore, for you to communicate instructions on the computer, it is easier for you to learn a computer Language (e.g. Python) than for the computer to learn English.

Python can be **easy to pick up and friendly to learn**. [Python](#) is a **general-purpose** interpreted , interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985-1990. There are two main python versions: 2.7 and 3. For this course, we will use 2.7 since it is the most common or popular used.

Todo: Review whether Python 2.7 is the correct option nowadays - or whether Python 3.x is more appropriate.

Basic Practise

Let's get familiar with Python by playing in the terminal in the interactive mode (you type a line at a time, and the interpreter responds). You invoke the interpreter and brings up the following prompt:

```
$ python
Python 2.7.9 (default, Sep 17 2016, 20:26:04)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Note: The \$ symbol when used in a code block like this represents typing on the command line. When you open the terminal, your username and current folder (directory) is usually shown on the left side of the \$ sign and then you can type your commands to the right side. In this case, we've typed the word `python` and then pressed *enter*.

Strings, integers, and floating points:

```
>>> print "Hello, Python!"
>>> x = 1          # Integer assignment
>>> y = 1005.00    # Floating points
>>> name = "John"  # A string
>>> print x
1
>>> print y
1005.0
>>> print name
John
```

In Python, the **standard order of operations** are evaluated from left to right following order (memorised by many as PEMDAS):

- **P**arentheses
- **E**xponents
- **M**ultiplication and **D**ivision
- **A**ddition and **S**ubtraction

Note: `//` is the floor division in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero.

```
>>> 3/4 * 5  # First division and then Multiplication
>>> 3.0 / 4 * 5
>>> (3.0 / 4) * 4
>>> 2**8
>>> -11.0//3
>>> 11.0//3 # Result floored (rounded away from zero)
>>> -11.0/3
>>> z = float(5)
>>> z
>>> z = int(5.25)
>>> z
>>> 10%7 # Remainder of a division
>>> 'abc' + 'fgb' # strings
```

Comparison operators:

Name	Syntax
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

```
>>> 2 == 3
False
# We got a boolean
>>> 3 == 3
True
>>> 2 < 3
True
>>> "a" < "aa"
True
```

Data Types

The data stored in memory can be of different types; Python has five: **Numbers, Strings, List, Tuple, and Dictionary**.

```
>>> type(x) # numbers
>>> type(y)
>>> type(name) # String
```

Strings in Python are a set of characters represented by the quotation marks. Python allows for either pair of single or double quotes.

Subsets of strings can be taken using the slice operator (`[]` and `[:]`) with indexes starting at 0 at the beginning of the string and working their way from -1 to the end.

The plus (+) sign is the string concatenation operator, and the asterisk (*) is the repetition operator. For example:

```
>>> string = 'Hello World!'
>>> print string           # Prints complete string
>>> print string[0]        # Prints first character of the string
>>> print string[2:5]      # Prints characters starting from 3rd to 5th
>>> print string[2:]       # Prints string starting from 3rd character
>>> print string * 2       # Prints string two times
>>> print string + "TEST"  # Prints concatenated string
```

Lists are the most versatile data types in Python. A list contains items separated by commas and enclosed in square brackets (`[]`) — similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator (`[]` and `[:]`) with indexes starting at 0 at the beginning of the list and working their way to ending -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

```
>>> list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
>>> tinylist = [123, 'john']

>>> print list             # Prints complete list
>>> print list[0]          # Prints first element of the list
>>> print list[1:3]        # Prints elements starting from 2nd till 3rd
>>> print list[2:]         # Prints elements starting from 3rd element
>>> print tinylist * 2     # Prints list two times
>>> print list + tinylist  # Prints concatenated lists
```

A **tuple** is another sequence data type that is similar to the list. It consists of some values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets (`[]`), and their elements and size can be changed, while tuples are enclosed in parentheses (`()`) and cannot be updated - **immutable**. Tuples can be

thought of as read-only lists.

```
>>> tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')

>>> print tuple           # Prints complete list
>>> print tuple[0]        # Prints first element of the list
>>> print tuple[1:3]      # Prints elements starting from 2nd till 3rd
>>> print tuple[2:]       # Prints elements starting from 3rd element
>>> print tinytuple * 2   # Prints list two times
>>> print tuple + tinytuple # Prints concatenated lists
```

Invalid operations on a tuple but valid on a list:

```
>>> tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
>>> list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
>>> tuple[2] = 1000      # Invalid syntax with tuple
>>> list[2] = 1000       # Valid syntax with list
```

Python's **dictionaries** are hash table type. They work like associative arrays and consist of key-value pairs. A dictionary key can be almost any Python type but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object. Dictionaries are enclosed by curly braces (`{}`), and values can be assigned and accessed using square braces (`[]`).

```
>>> dict = {}
>>> dict['one'] = "This is one"
>>> dict[2] = "This is two"
# keys are: name, code and dept; values are: john, 6734 and sales
>>> tinydict = {'name': 'john', 'code':6734, 'dept': 'sales'}

>>> print dict['one']     # Prints value for 'one' key
>>> print dict[2]         # Prints value for 2 key
>>> print tinydict        # Prints complete dictionary
>>> print tinydict.keys() # Prints all the keys
>>> print tinydict.values() # Prints all the values
```

To quit the Python interpreter:

```
>>> quit()
```

Scripts

A Script is a sequence of statements (lines) into a file using a text editor and tells Python interpreter to execute the statements in the file.

- We can write a program in our script like a recipe or installation of software. At the end of the day, a program is a **sequence** of steps to be done in order.
- Some of the steps can be **conditional**, that means that sometimes they can be skipped.
- Sometimes a step or group of steps are to be **repeated**.
- Sometimes we store a set of steps that will be used over and over again in several parts of the program (**functions**).

Note: Have a look on the code [style guide](#) for a good coding practise. As a first good practise, do not name files or folders with space in between:

- Bad -> example 1.py
- Good -> example_one.py

Further explanation as to why we use underscores, and not `exampleOne.py` for example, can be found in documents such as the PEP8 Guide to Python.

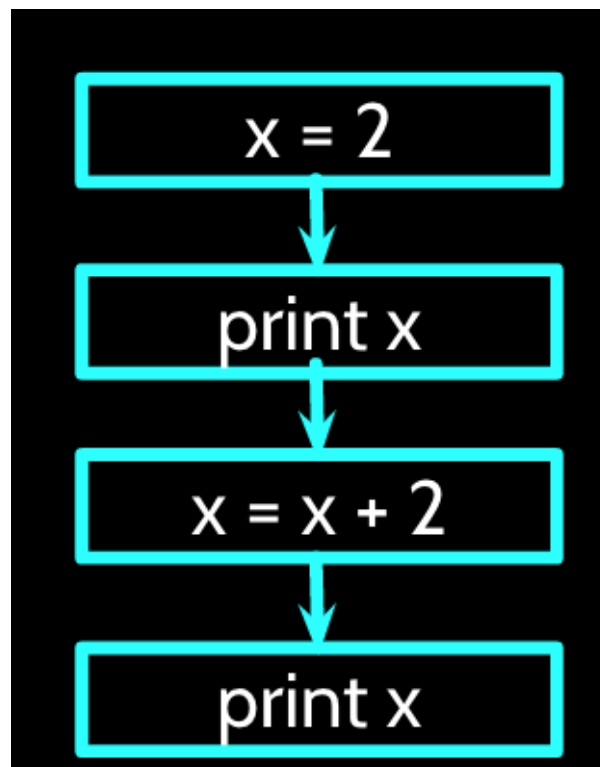
We will make a simple script:

```
$ pwd
$ /home/pi
$ mkdir codes/python_examples
$ cd codes/python_examples
$ nano example_flow.py
```

Then you can type in the editor:

```
#!/usr/bin/env python
x = 2
print x
x = x + 2
print x
```

When a program is running, it flows from one step to the next. As programmers, we set up “paths” for the program to follow.



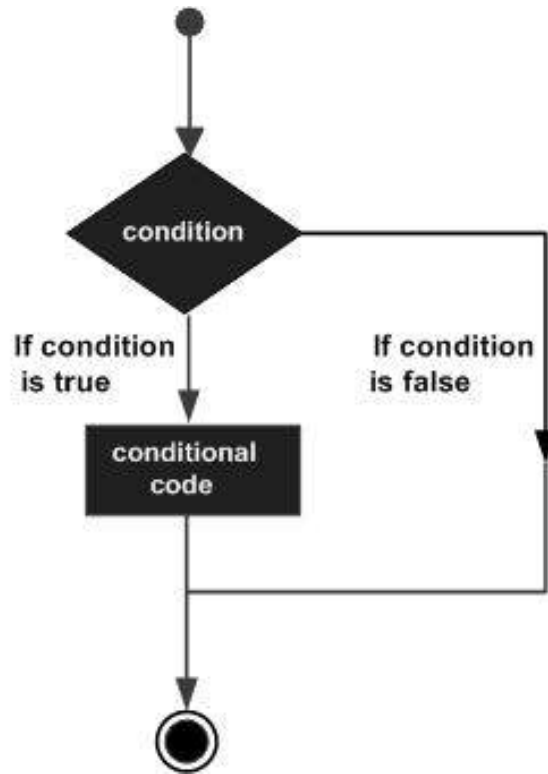
Close the text editor, and then you can execute it in two ways:

```
$ python example_flow.py
```

The other is to give the script the access permissions to be an executable file through the `chmod` Linux command:

```
$ chmod u+x example_flow.py
$ ./example_flow.py
```

Now let's do an example where we have a **conditional** that implies a decision-making about a situation. Decision making is the anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions. The following diagram illustrates the conditional:



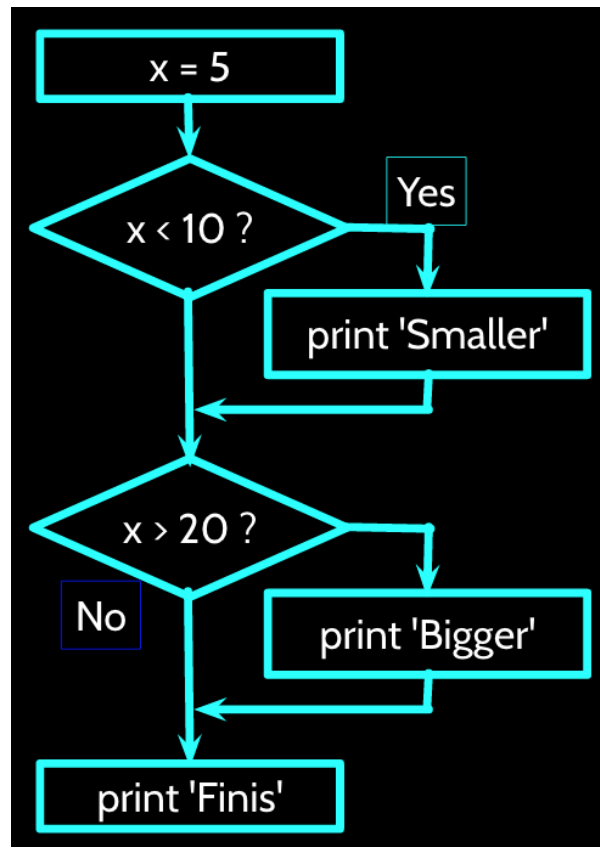
```
$ nano example_conditional.py
```

Now let's add the code:

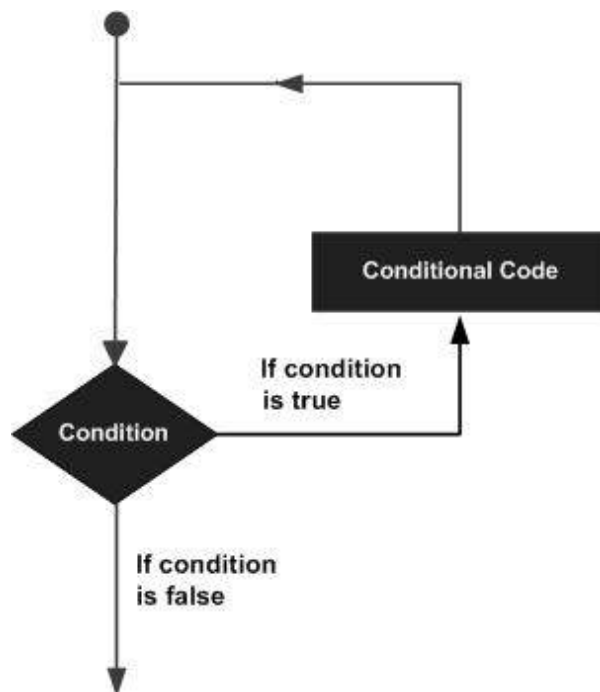
```
#!/usr/bin/env python
x = 5
if x < 10:
    print 'Smaller'
elif x > 20:
    print 'Bigger'
print 'Finis' #outside conditional
```

```
$ chmod u+x example_conditional.py
$ ./example_conditional.py
```

Flow of the code:



A *loop statement* allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement:



While loops repeats a statement or group of statements while a given condition is `True`. It tests the condition before executing the loop body.

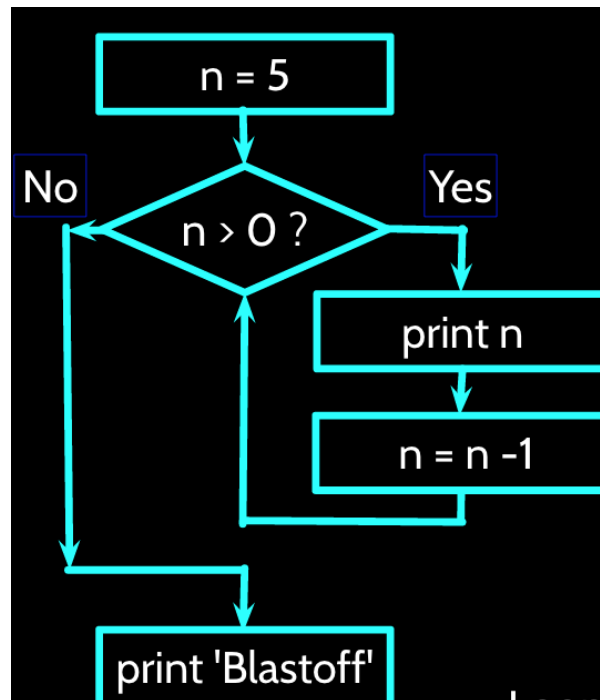
Now let's add the code to our script called *example_while_loop.py*:

```
#!/usr/bin/env python
n = 5
while n > 0:
    print n
    n = n - 1
print 'Blastoff!' #outside loop
```

Before running, remember to give the permissions:

```
$ chmod u+x example_while_loop.py
$ ./example_while_loop.py
```

Flow of the code:



Loops (repeated steps) have *iteration variables* that change each time through a loop (like *n*). Often these *iteration variables* go through a sequence of numbers.

For loop executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

Now let's add the code to our script called *_example_for_loop.py*:

```
#!/usr/bin/env python

# Area of a circle = pi * r**2

# Library
import numpy as np

# List are called iterables
list = [1, 2, 3, 4, 5, 6]

for radius in list:
```

(continues on next page)

(continued from previous page)

```
area = np.pi * radius ** 2
print "The area of a circle of radius ", radius
print "cm is", area, "cm^2"
print "Finished to calculate the areas of circles"
```

```
$ chmod u+x example_for_loop.py
$ ./example_for_loop.py
```

Here we are importing the [Numpy library](#). That is the fundamental package for scientific computing with Python. We are adding a short alias to the library to call its methods, in this case, the value of Pi.

Functions

A function is a block of organised, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

Now, let's make a function that can be used in the for loop example.

```
$ nano example_function_circle_area.py
```

```
#!/usr/bin/env python

# Area of a circle = pi * r**2

# Library Numpy
import numpy as np

def area_circle(radius):
    'Function that calculates the area of a circle'
    area = np.pi * radius ** 2
    return area

# List are called interables
list = [1, 2, 3, 4, 5, 6]

for radius in list:
    area = area_circle(radius)
    print "The area of a circle of radius ", radius
    print "cm is", area, "cm^2"
print "Finished to calculate the areas of circles"
```

```
$ chmod u+x example_function_circle_area.py
$ ./example_function_circle_area.py
```

We can see that we get the same result but it is more organised and we can use the function in other sections of our code.

Now let's ask the user to provide a list:

```
$ nano example_function_circle_area_user_1.py
```

```
# Area of a circle = pi * r**2

# Library Numpy
```

(continues on next page)

(continued from previous page)

```

import numpy as np
# Library to Safely evaluate an expression node
# or a string containing a Python expression
import ast

# List are called interables
list_raw = raw_input('Provide a list of radius in cm like \
[3, 2, 12, 6]: \n')
list = ast.literal_eval(list_raw)

def area_circle(radius):
    'Function that calculates the area of a circle'
    area = np.pi * radius ** 2
    return area

for radius in list:
    area = area_circle(radius)
    print "The area of a circle of radius ", radius
    print "cm is", area, "cm^2"
print "Finished to calculate the areas of circles"

```

```

$ chmod u+x example_function_circle_area_user_1.py
$ ./example_function_circle_area_user_1.py

```

If we do not use the `ast` library to evaluate a string containing a Python expression (in this case a list), we will get an error since Python will interpret as a string type and not a list type.

A second way to do it is by using the `sys` module which provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter.

Now let's ask the user to provide a list by passing the strings directly:

```
$ nano example_function_circle_area_user_2.py
```

```

#!/usr/bin/env python

# Usage instructions:
# ./example_function_circle_area_user_2.py "[1, 2, 3]"

# Area of a circle = pi * r**2

# Library Numpy
import numpy as np
# Library to Safely evaluate an expression node
# or a string containing a Python expression
import ast
# Module provides access to some variables
# used or maintained by the interpreter
import sys

list_raw = sys.argv[1]
list = ast.literal_eval(list_raw)

```

(continues on next page)

(continued from previous page)

```
def area_circle(radius):  
    'Function that calculates the area of a circle'  
    area = np.pi * radius ** 2  
    return area  
  
for radius in list:  
    area = area_circle(radius)  
    print "The area of a circle of radius ", radius  
    print "cm is", area, "cm^2"  
print "Finished to calculate the areas of circles"
```

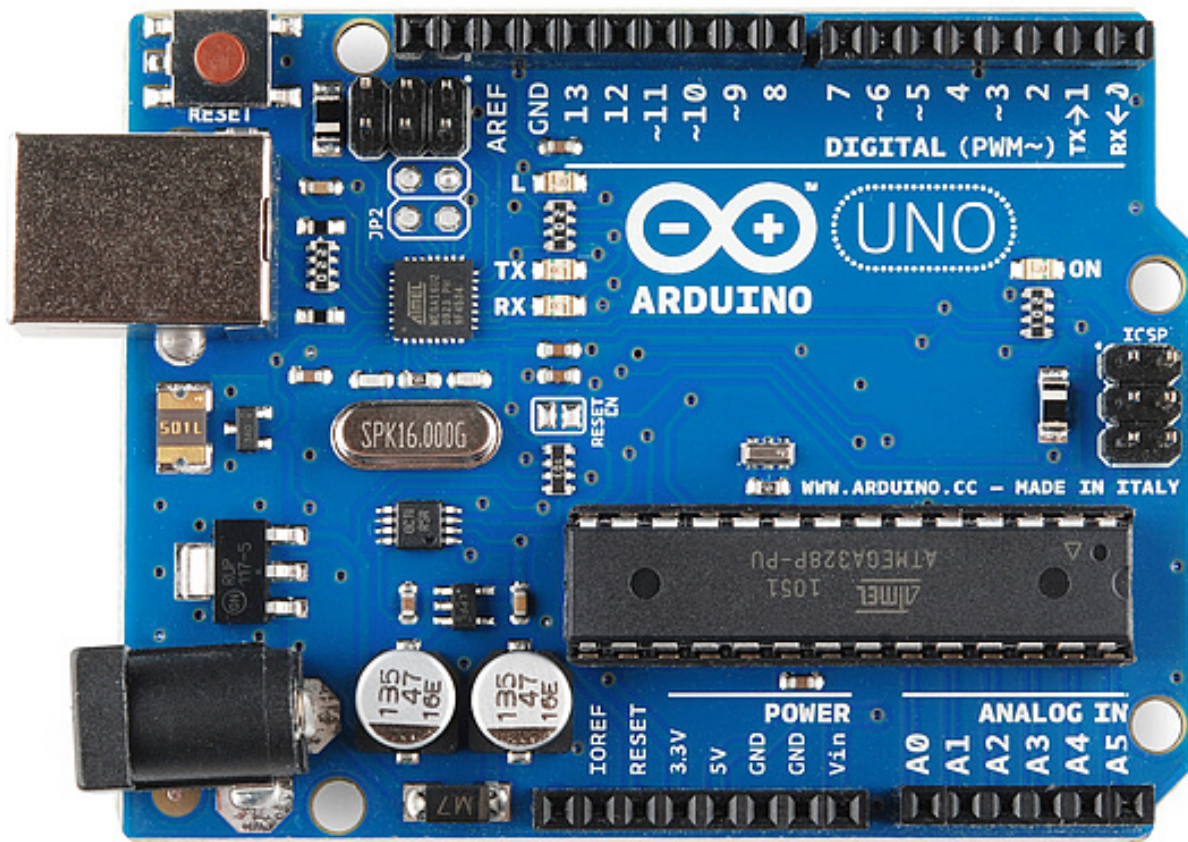
```
$ chmod u+x example_function_circle_area_user_2.py  
$ ./example_function_circle_area_user_2.py "[1, 2, 3]"
```

References

1. Charles Severance course: Python for everybody
-

2.2 Arduino

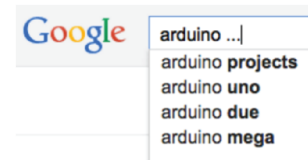
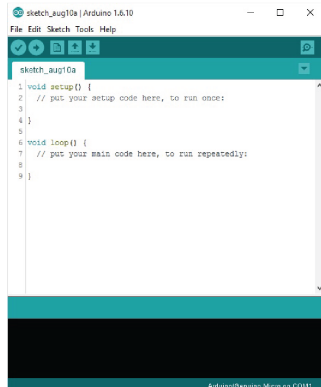
Welcome to the guide on Arduino. See below for *Why Arduino?* and possible *Alternative Microcontrollers*.



2.2.1 Introduction to Arduino

- *What is Arduino?*
- *What is Arduino: Hardware*
 - *Arduino UNO Board Structure*
 - *Digital pins*
 - *Analog pins*
 - *PWM*
- *What is Arduino: Software*
 - *Quick tour of the Arduino IDE*
 - *Anatomy of an Arduino Sketch*
 - *Arduino Language*
 - * *Variables*
 - * *Further Arduino Syntax*

What is Arduino?



Software
IDE

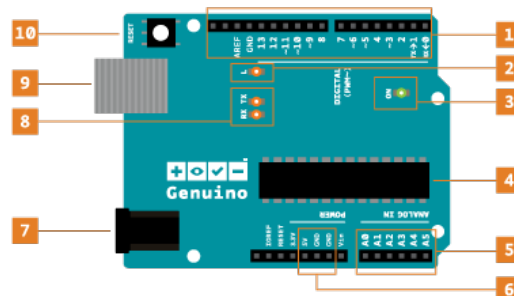
Hardware
Board

Community
Web

What is Arduino: Hardware

An Arduino is essentially a microcontroller. That is a small “computer” (SoC - System on a Chip) on a single integrated circuit containing a processor core, memory and programmable input/output peripherals (a.k.a. sensors and actuators).

Arduino UNO Board Structure



1. Digital pins Use these pins with `digitalRead()`, `digitalWrite()`, and `analogWrite()`. `analogWrite()` works only on the pins with the PWM symbol.
2. Pin 13 LED The only actuator built-in to your board. Besides being a handy target for your first blink sketch, this LED is very useful for debugging.
3. Power LED Indicates that your Arduino is receiving power. Useful for debugging.
4. ATmega microcontroller The heart of your board.
5. Analog in Use these pins with `analogRead()`.

- Here's a scheme of it's pin's functionalities:



They can be controlled with two functions:

- We will see in detail how to control them later on.

Note: When a digital pin is **high** (at 5V) it can be represented as `HIGH` or `1`. Conversely when it is **low** (at 0V) it can be represented as `LOW` or `0`.

Pins Configured as INPUT

Arduino (Atmega) pins default to inputs, so they don't need to be explicitly declared as inputs with `pinMode()` when you're using them as inputs. Pins configured this way are said to be in a high-impedance state. Input pins make extremely small demands on the circuit that they are sampling. This means that it takes very little current to move the input pin from one state to another, and can make the pins useful for such tasks as implementing a capacitive touch sensor, reading an LED as a photodiode, or reading an analog sensor with a scheme such as `RCTime`.

This also means however, that pins configured as `pinMode(pin, INPUT)` with nothing connected to them, or with wires connected to them that are not connected to other circuits, will report seemingly random changes in pin state, picking up electrical noise from the environment, or capacitively coupling the state of a nearby pin.

Pins Configured as INPUT with Pullup/Pulldown Resistors

Often it is useful to steer an input pin to a known state if no input is present. In doing so you cancel the random values of the pin when nothing is connected to it. This can be done by adding a pull-up resistor (to +5V), or a pulldown resistor (resistor to ground) on the input. A 10K resistor is a good value for a pull-up or pull-down resistor.

Note: On the Arduino there are built-in pullup resistors that can be activate via code setting `pinMode()` as `INPUT_PULLUP`. This effectively inverts the behaviour of the `INPUT` mode, where `HIGH` means the sensor is off, and `LOW` means the sensor is on.

When connecting a sensor to a pin configured with `INPUT_PULLUP`, the other end should be connected to ground. In the case of a simple switch, this causes the pin to read `HIGH` when the switch is open, and `LOW` when the switch is pressed.

You can find more information about pull-up/pull-down resistors [in this guide](#).

Pins configured as OUTPUT

Pins configured as `OUTPUT` with `pinMode()` are said to be in a low-impedance state. This means that they can provide a substantial amount of current to other circuits. Atmega pins can source (provide positive current) or sink (provide negative current) up to 40 mA (milliamps) of current to other devices/circuits. This is enough current to brightly light up an LED (don't forget the series resistor), or run many sensors, for example, but not enough current to run most relays, solenoids, or motors.

Short circuits on Arduino pins, or attempting to run high current devices from them, can damage or destroy the output transistors in the pin, or damage the entire Atmega chip. Often this will result in a "dead" pin in the microcontroller but the remaining chip will still function adequately. For this reason it is a good idea to connect `OUTPUT` pins to other devices with 470Ω or 1k resistors, unless maximum current draw from the pins is required for a particular application.

Analog pins

The Atmega controllers used for the Arduino contain an onboard 6 channel analog-to-digital (A/D) converter, that correspond to the analog pins (A0-A5). The converter has 10 bit resolution, returning integers from 0 to 1023. This means that when we want to read the pin's value it maps input voltages between 0 and 5 volts into integer values between 0 and 1023. The analog pins can be controlled with two functions:

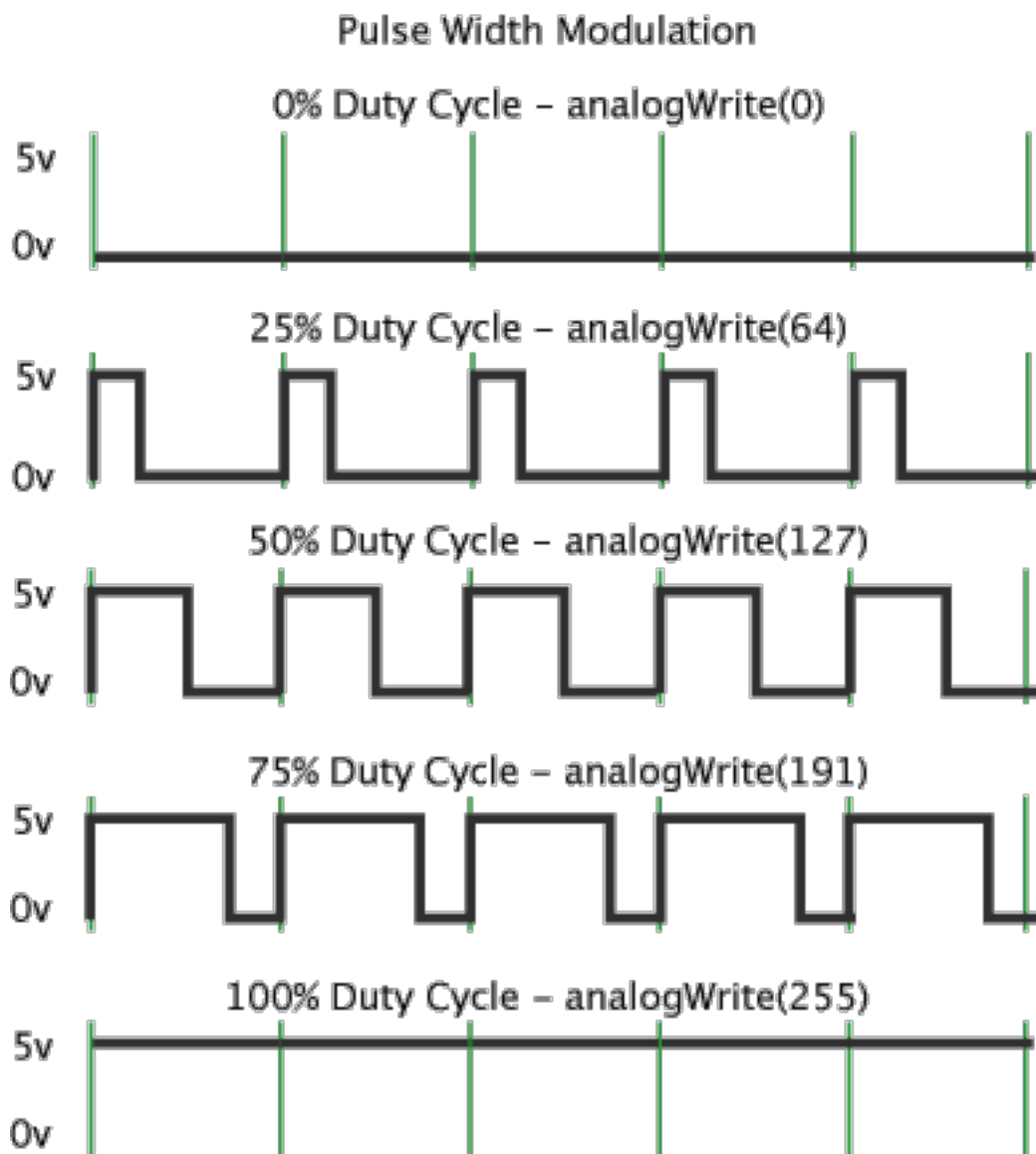
- `analogRead()`: reads the value of the pin
- `analogWrite()`: sets the output value of the pin

While the main function of the analog pins for most Arduino users is to read analog sensors, the analog pins also have all the functionality of general purpose input/output (GPIO) pins (the same as digital pins 0 - 13). Therefore they can be used as digital pins with the digital functions.

The Atmega datasheet also cautions against switching analog pins in close temporal proximity to making A/D readings (`analogRead`) on other analog pins. This can cause electrical noise and introduce jitter in the analog system. It may be desirable, after manipulating analog pins (in digital mode), to add a short delay before using `analogRead()` to read other analog pins.

PWM

In the graphic below, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to `analogWrite()` is on a scale of 0 - 255, such that `analogWrite(255)` requests a 100% duty cycle (always on), and `analogWrite(127)` is a 50% duty cycle (on half the time) for example.



Note: For more informations you can check the [Arduino Microcontroller](#) section of their Foundations.

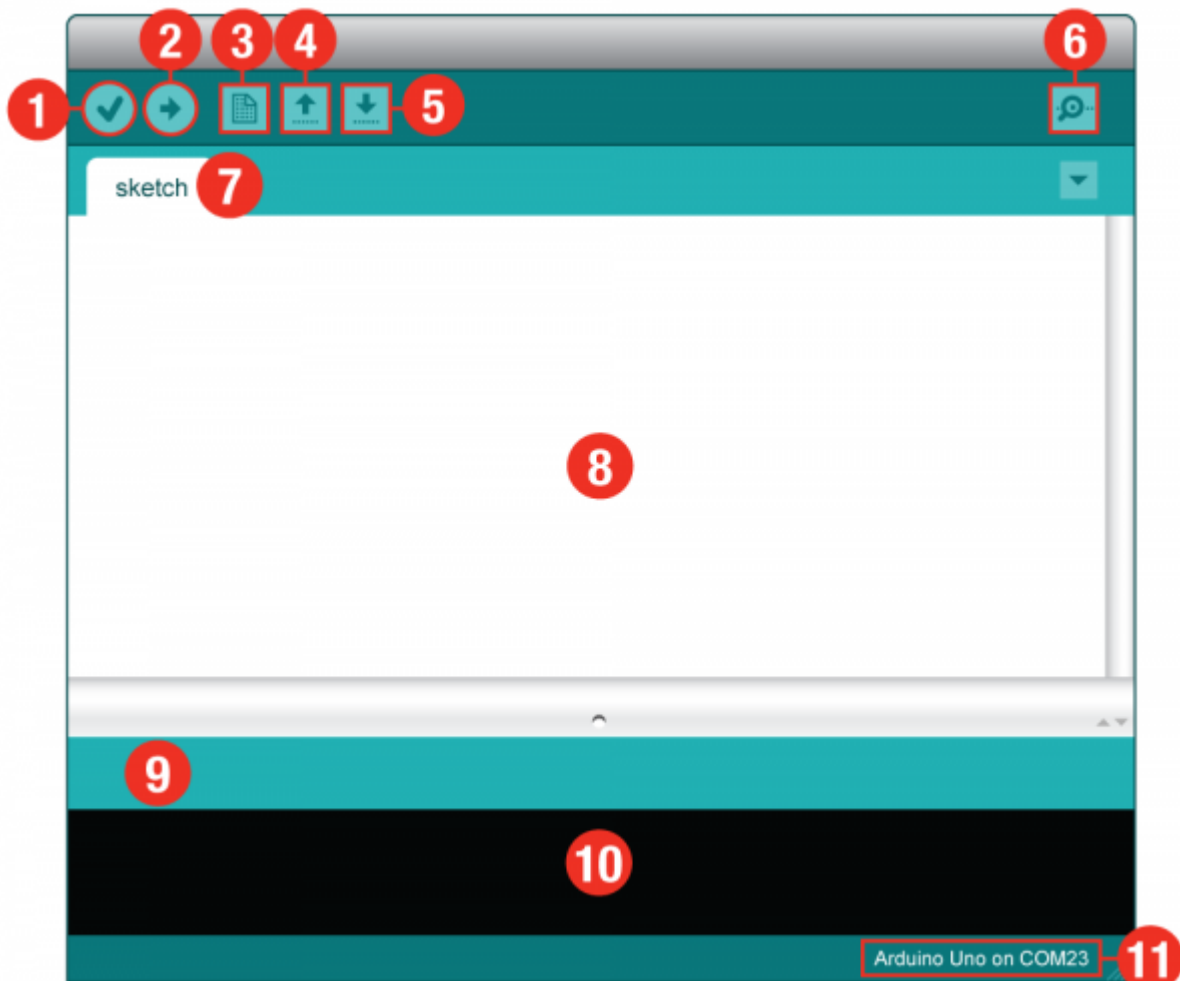
What is Arduino: Software

Please download the Arduino IDE from [Arduino's website](#) if you are having any trouble there are setup guides for every operating system on [this page](#).

Quick tour of the Arduino IDE

Arduino has it's own integrated development environment that simplifies some of the operations. The IDE manages library, offers a built-in compiler and has a lot of examples and references.

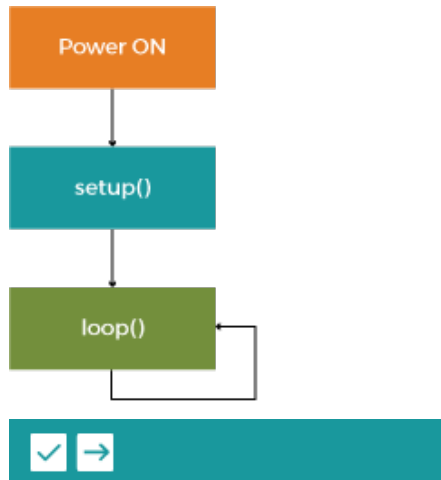
Here is a screenshot of how the IDE looks like and all its functionalities:



1. **Verify:** Compiles and approves your code. It will catch errors in syntax (like missing semi-colons or parenthesis).
2. **Upload:** Sends your code to the Uno. When you click it, you should see the lights on your board blink rapidly.
3. **New:** This buttons opens up a new code window tab.
4. **Open:** This button will let you open up an existing sketch.
5. **Save:** This saves the currently active sketch.
6. **Serial Monitor:** This will open a window that displays any serial information your Uno Board is transmitting. It is very useful for debugging.
7. **Sketch Name:** This shows the name

of the sketch you are currently working on. 8. **Code Area:** This is the area where you compose the code for your sketch. 9. **Message Area:** This is where the IDE tells you if there were any errors in your code. 10. **Text Console:** The text console shows complete error messages. When debugging, the text console is very useful. 11. **Board and Serial Port:** Shows you what board and the serial port selections

Anatomy of an Arduino Sketch



```

void setup() {
}

void loop() {
}

```

Each Arduino program is called a “sketch”. Each Sketch has two main function `setup()` and `loop()`:

- `setup()` is called when a sketch starts. Use it to initialise variables, pin modes, start using libraries, etcetera. The setup function will only run once, after each powerup or reset of the Arduino board.
- `loop()` is called after `setup()`, and it loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

Here’s an example of what an Arduino sketch looks like:

```

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

  This example code is in the public domain.
  */

// Pin 13 has an LED connected.
// give it a name:
int pin = 13;

// the setup routine runs once when you press reset:

```

(continues on next page)

(continued from previous page)

```
void setup() {  
    // initialize the digital pin as an output.  
    pinMode(pin, OUTPUT);  
}  
  
// the loop routine runs over and over again forever:  
void loop() {  
    digitalWrite(pin, HIGH);    // turn the LED on (HIGH is the voltage level)  
    delay(1000);                // wait for a second  
    digitalWrite(pin, LOW);     // turn the LED off by making the voltage LOW  
    delay(1000);                // wait for a second  
}
```

Arduino Language

The programming language used in Arduino comes from C++ and it is quite different from Python. Although the names of the functions are self explanatory as they are written in plain English.

The most visible difference is that you have to terminate each line with a semicolon (;) to end a statement, if you forget one the compiler will warn you about it.

Variables

The fundamental difference from Python is that Arduino doesn't use dynamic typing. This means that when we create a variable we have to tell the compiler what *type* of variable it is:

```
int pin = 13;
```

We have just created a variable whose type is **int**, whose name is **pin**, whose **value** is 13. Later on in the program, you can refer to this variable by its name, at which point its value will be looked up and used. For example, in this statement:

```
pinMode(pin, OUTPUT);
```

We are setting the pin 13 as an OUTPUT pin, what we have written would be equivalent to `pinMode(13, OUTPUT);`. You can change the value of a variable using an assignment (indicated by an equals sign). For example:

```
pin = 12;
```

will change the value of the variable to 12. Notice that we don't specify the type of the variable: it's not changed by the assignment. That is, **the name of the variable is permanently associated with a type; only its value changes**.

When you assign one variable to another, you're making a copy of its value and storing that copy in the location in memory associated with the other variable. Changing one has no effect on the other. For example, after:

```
int pin = 13;  
int pin2 = pin;  
pin = 12;
```

only `pin` has the value 12; `pin2` is still 13.

Arduino has many types. To name a few: `int`, `float`, `string`, `boolean`, `char`. To learn more about the specificities about each type we invite you to look at the [Arduino Reference Page](#).

Further Arduino Syntax

We have collected useful examples of Arduino syntax for some major functions you are familiar with in Python.

If statement

```
if (pinFiveInput < 500) {
    // action A
} else {
    // action B
}
```

While loop

```
while (expression) {
    // statement(s)
}
```

For loop

```
for (int i=0; i <= 255; i++) {
    analogWrite(pwmPin, i);
    delay(10);
}
```

That can be translated into pseudocode as:

```
for (initialisation; condition; increment) {
    //statement(s);
}
```

Comments

For inline comments you can use `//`. For multiple-lines comments instead use `/*` at the beginning and `*/` at the end.

```
// this is an inline comment

/* this is a comment
on multiple lines */
```

Comparison Operators

- `x == y` (x is equal to y)
- `x != y` (x is not equal to y)
- `x < y` (x is less than y)
- `x > y` (x is greater than y)
- `x <= y` (x is less than or equal to y)
- `x >= y` (x is greater than or equal to y)

Arithmetic Operations

- `=` (assignment operator)
- `+` (addition)
- `-` (subtraction)
- `*` (multiplication)

- / (division)
- % (modulo)

Boolean Operations

- && (and)
- || (or)
- ! (not)

Functions

Segmenting code into functions allows a programmer to create modular pieces of code that perform a defined task and then return to the area of code from which the function was “called”. The typical case for creating a function is when one needs to perform the same action multiple times in a program. Standardising code fragments into functions has several advantages:

- Functions help the programmer stay organised. Often this helps to conceptualise the program.
- Functions codify one action in one place so that the function only has to be thought out and debugged once.
- This also reduces chances for errors in modification, if the code needs to be changed.
- Functions make the whole sketch smaller and more compact because sections of code are reused many times.
- They make it easier to reuse code in other programs by making it more modular, and as a nice side effect, using functions also often makes the code more readable.
- There are two required functions in an Arduino sketch, `setup()` and `loop()`. Other functions must be created outside the brackets of those two functions. As an example, we will create a simple function to multiply two numbers.

Anatomy of a C function

Datatype of data returned,
any C datatype.

Parameters passed to
function, any C datatype.

"void" if nothing is returned.

Function name

```
int myMultiplyFunction(int x, int y){  
    int result;  
    result = x * y;  
    return result;  
}
```

Return statement,
datatype matches
declaration.

Curly braces required.

Our function needs to be declared outside any other function, so `myMultiplyFunction()` can go either above or below the `loop()` function.

```

void setup() {
  Serial.begin(9600);
}

void loop() {
  int i = 2;
  int j = 3;
  int k;

  k = myMultiplyFunction(i, j); // k now contains 6
  Serial.println(k);
  delay(500);
}

int myMultiplyFunction(int x, int y) {
  int result;
  result = x * y;
  return result;
}

```

Another Function Example: This function will read a sensor five times with `analogRead()` and calculate the average of five readings. It then scales the data to 8 bits (0-255), and inverts it, returning the inverted result.

```

int readAndCompute() {
  int i;
  int sval = 0;

  for (i = 0; i < 5; i++) {
    sval = sval + analogRead(0); // sensor on analog pin 0
  }

  sval = sval / 5; // average
  sval = sval / 4; // scale to 8 bits (0 - 255)
  sval = 255 - sval; // invert output
  return sval;
}

```

To call our function we just assign it to a variable, the following code should be placed inside `loop()`.

```

int sens;
sens = readAndCompute();

```

For further reference about the syntax and language you can check these notes:

- <https://www.arduino.cc/en/Tutorial/Variables>
- <https://www.arduino.cc/en/Reference/HomePage>

2.2.2 Basics

In this section we are going to see the Arduino equivalent of the scripts we ran from the Raspberry Pi.

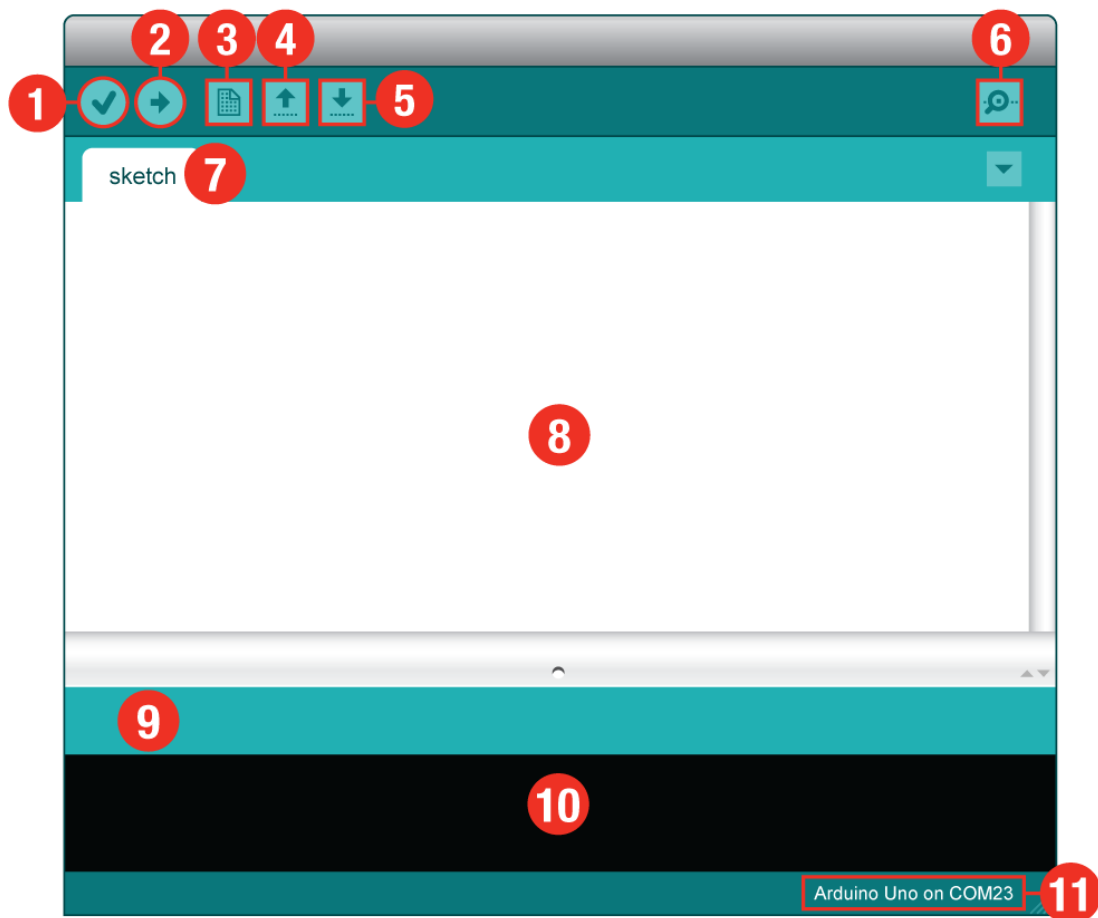
- *Example: Blink*
 - *Alternative Hardware*

- *Understanding the “Blink” code*
- *Example: Led PWM*
 - *Hardware*
 - *Code*
 - *Understanding the “Fade” code*
- *Example: Button*
 - *Hardware*
 - *Code*
- *Challenge*

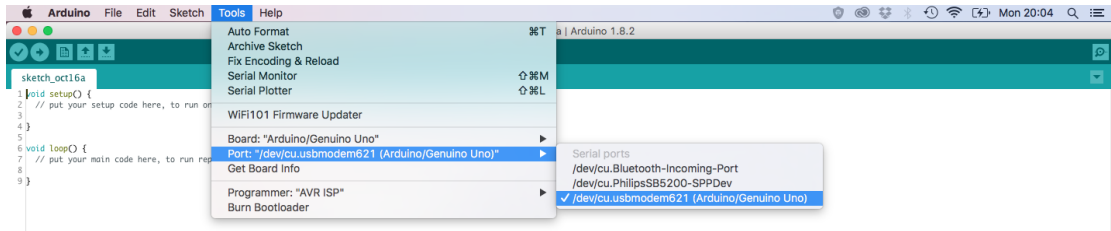
Example: Blink

We are going to run the “blink” sketch we have seen early on in this tutorial. It is the most basic sketch a sort of “Hello World!” for Arduino. It makes the built-in LED on pin 13 blink in intervals of 1 second.

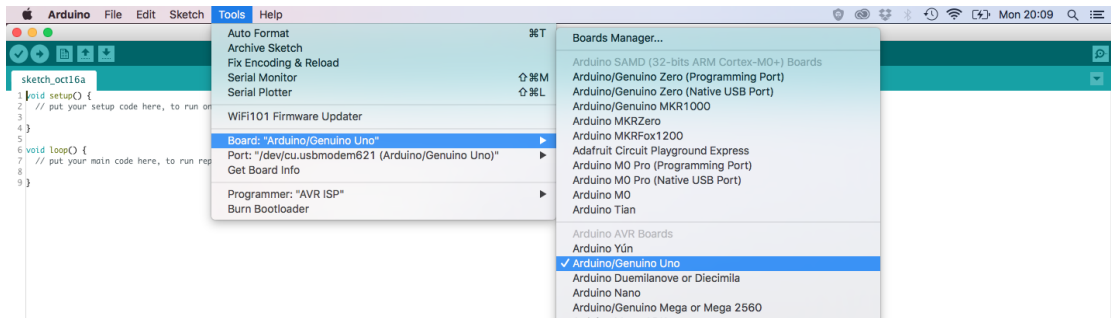
1. Connect the Arduino to your laptop with the USB cable
2. Open the IDE



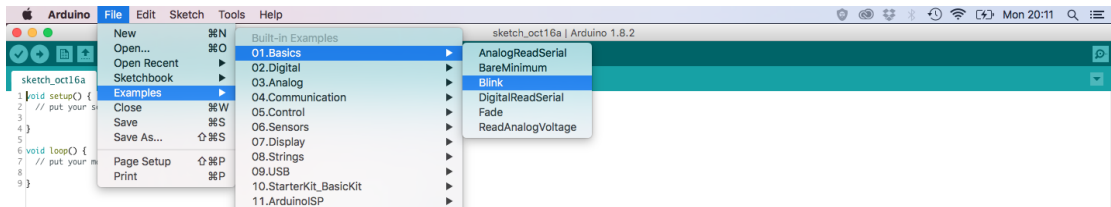
- Click **Tools** → **Serial Port** and select the USB serial port to which your Arduino is connected to (the path changes with operating system and USB port you are using, so the name might be different for you).



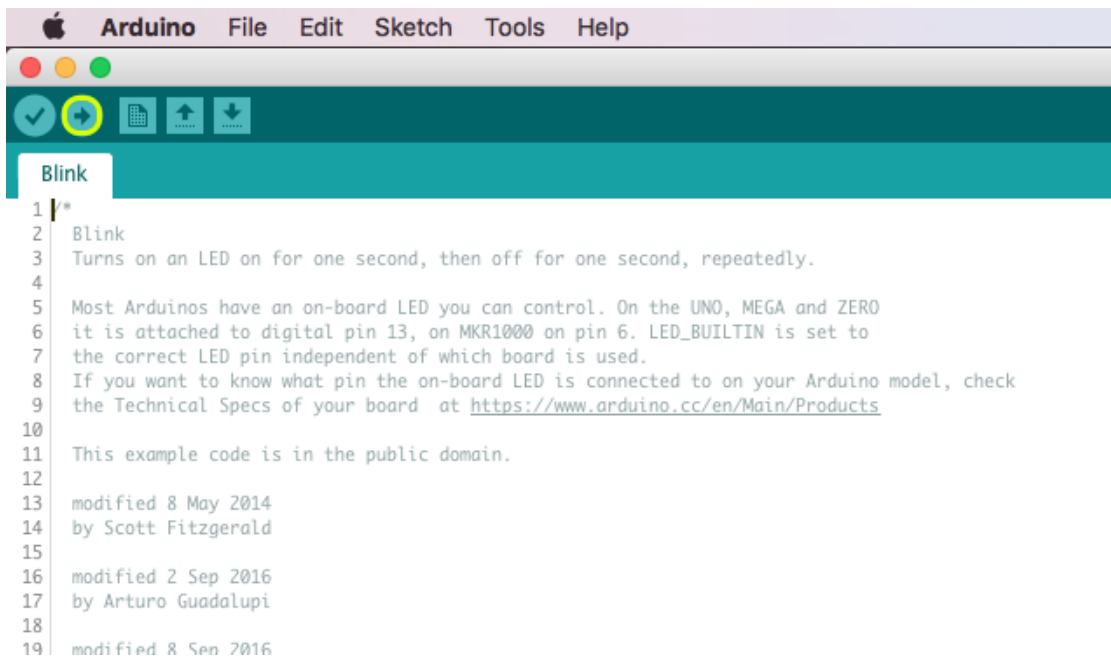
- Then, select the right board: click **Tools** → **Board** → **Arduino Uno**.



- Then you can open the basic sketch “Blink” by clicking on **File** → **Example** → **01. Basics** → **Blink**.



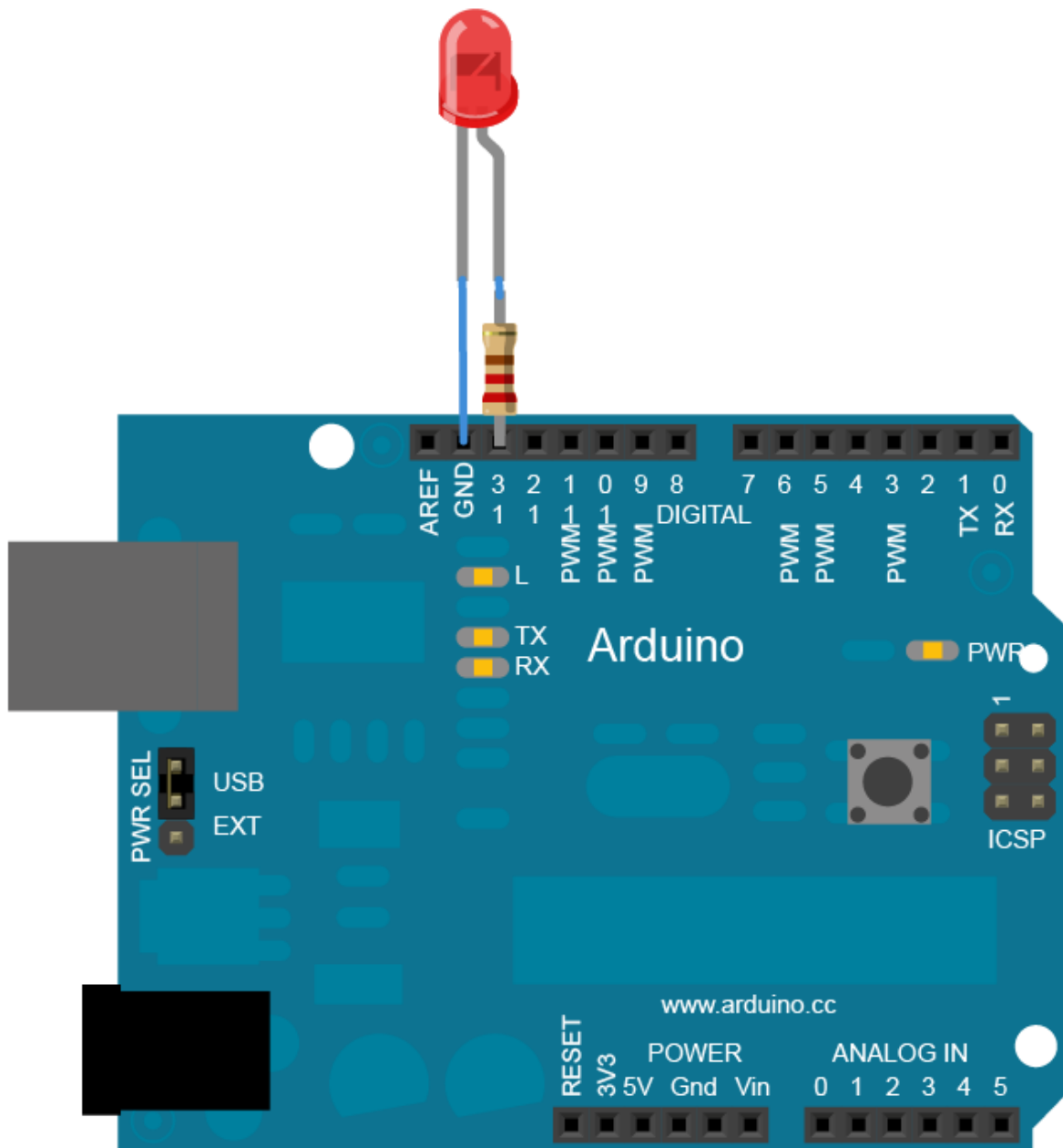
- You can then upload the sketch on the Arduino by clicking the “Upload” button (the one with a right arrow).



7. Once uploaded you will see the LED on pin 13 blink.

Alternative Hardware

You might want to try to use an external LED. Here's the wiring diagram:



Understanding the “Blink” code

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

- `pinMode(LED_BUILTIN, OUTPUT);` here we are initialising pin 13 as a output pin, note that instead of using a pin number we are using the `LED_BUILTIN` constant that stands for “pin 13”
- `digitalWrite(LED_BUILTIN, HIGH);` here we are turning the LED on by setting the digital output as `HIGH` (`HIGH` is maximum voltage level, 5V)
- `delay(1000);` here we are waiting for a second, note that the `delay()` function takes as a parameter milliseconds
- `digitalWrite(LED_BUILTIN, LOW);` here we are turning the LED off by making the voltage `LOW` (0V)
- `delay(1000);` here we are waiting for a second again

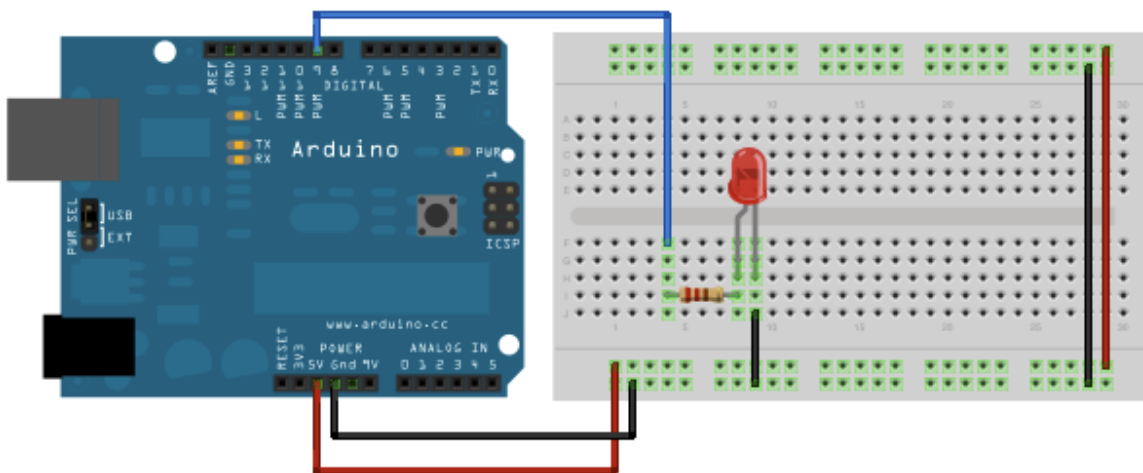
We have setup general instructions in our setup, those instructions won’t change when our sketch is running. We have inserted all the functions in the main loop so that they can be repeated infinitely.

Try to tweak the delays to see how the timing differs.

Example: Led PWM

Here we will see how to do pulse-width modulation with the Arduino using a LED.

Hardware



Code

For the code you can upload the built-in example “Fade” from **File** → **Example** → **01. Basics** → **Fade**.

Understanding the “Fade” code

```
int led = 9;
int brightness = 0;
int fadeAmount = 5;

void setup() {
  pinMode(led, OUTPUT);
}

void loop() {

  analogWrite(led, brightness);
  brightness = brightness + fadeAmount;

  if (brightness <= 0 || brightness >= 255) {
    fadeAmount = -fadeAmount;
  }

  delay(30);
}
```

- `int led = 9;` here we are creating a variable of type `int` with name `led` and storing in it the pin number that our LED is connected to, note that we are using pin number 9 which is one of the PWM-capable pins (marked by the ~ sign).
- `int brightness = 0;` here we are creating a variable of type `int` with name `brightness` and assigning the initial value of 0
- `int fadeAmount = 5;` here we are storing the amount we want the LED to fade for each interval in the `fadeAmount` variable
- `pinMode(led, OUTPUT);` here we are declaring the led pin as an output note that this would be equivalent to this `pinMode(9, OUTPUT);`
- `analogWrite(led, brightness);` here we are writing on pin 9 (`led`) the brightness values
- `brightness = brightness + fadeAmount;` here we are adding a `fadeAmount` to the brightness level
- Then

```
if (brightness <= 0 || brightness >= 255) {
  fadeAmount = -fadeAmount;
}
```

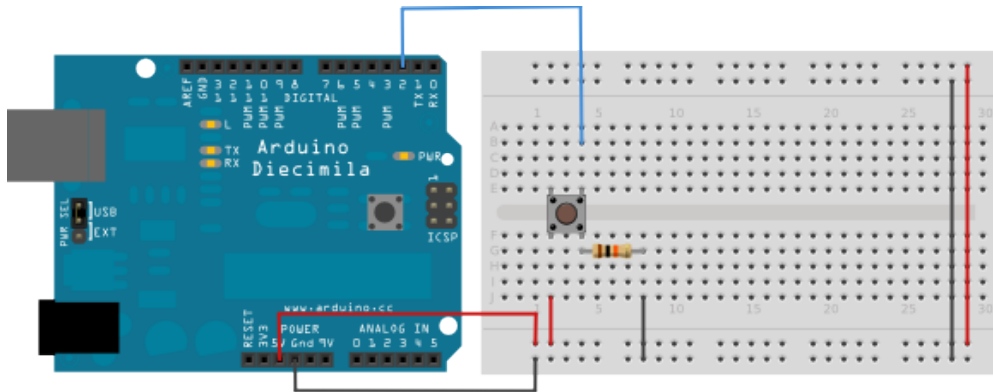
checks that the brightness level never takes invalid values (below 0 or above 255)

- `delay(30);` a short delay to make the dimming effect more visible

Example: Button

Here we are going to see how Arduino receives signals from input devices using a button.

Hardware



Code

For the code you can copy and paste the following code:

```
int pushButton = 2;

void setup() {
  Serial.begin(9600);
  pinMode(pushButton, INPUT);
}

void loop() {
  int buttonState = digitalRead(pushButton);
  Serial.println(buttonState);
  delay(1);
}
```

- `Serial.begin(9600);` here we are opening the [serial communication](#) with a baud rate of 9600 Bauds
- `pinMode(pushButton, INPUT);` here we are setting the button's pin as input
- `int buttonState = digitalRead(pushButton);` here we are reading the voltage of the button and memorising it in the variable `buttonState`
- `Serial.println(buttonState);` here we are printing the values in the Serial Monitor
- `delay(1);` a short delay to stabilise the readings

To monitor what your Arduino is printing open the serial monitor by clicking on the serial monitor button:



Challenge

Important: You must demonstrate your build & code to the tutor team

We challenge you to combine the previous three sketches (Blink, Fade, Button) to create one that, with the press of the button, controls 2 LEDs such that:

- when the button is pressed one of the two LEDs fades to 25% of its brightness and the other one blinks once
- when the button is released the faded LED returns to 100% brightness.

You can find further help [here](#).

Acknowledgements

Some material was taken from [the Arduino website](#).

2.2.3 Sensors

In this section we are going to see how to receive data from different sensors on the Arduino. The sensors we are going to use are:

1. Potentiometer
2. Light Sensor

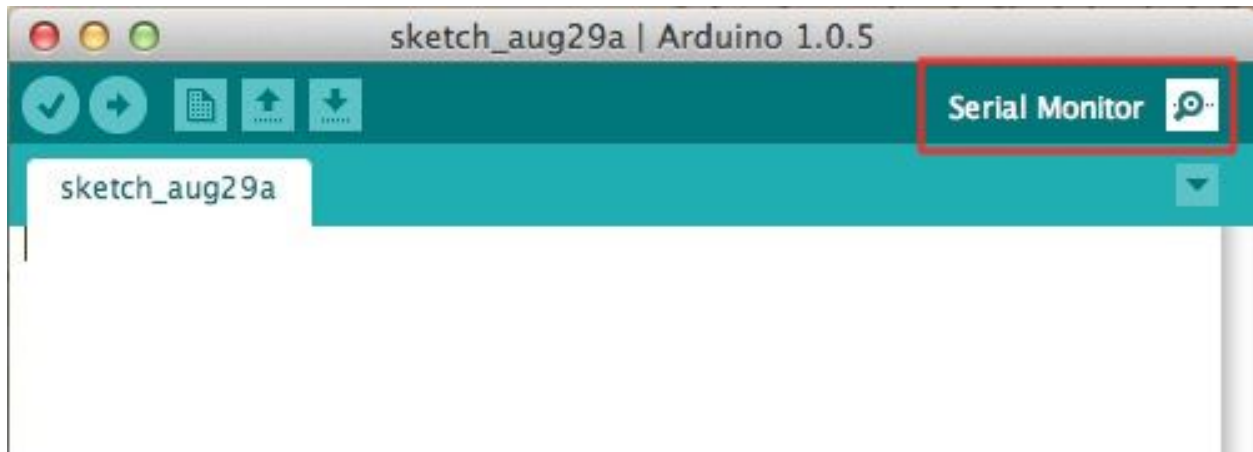
At the beginning of this session you should have collected a kit that is made of:

- Arduino
- 1 Potentiometer
- 1 Photo-resistor
- 1 LED
- 1 Resistor (you determine the value)

Reading the sensor information

For these exercises, there are two approaches to reading the sensor data.

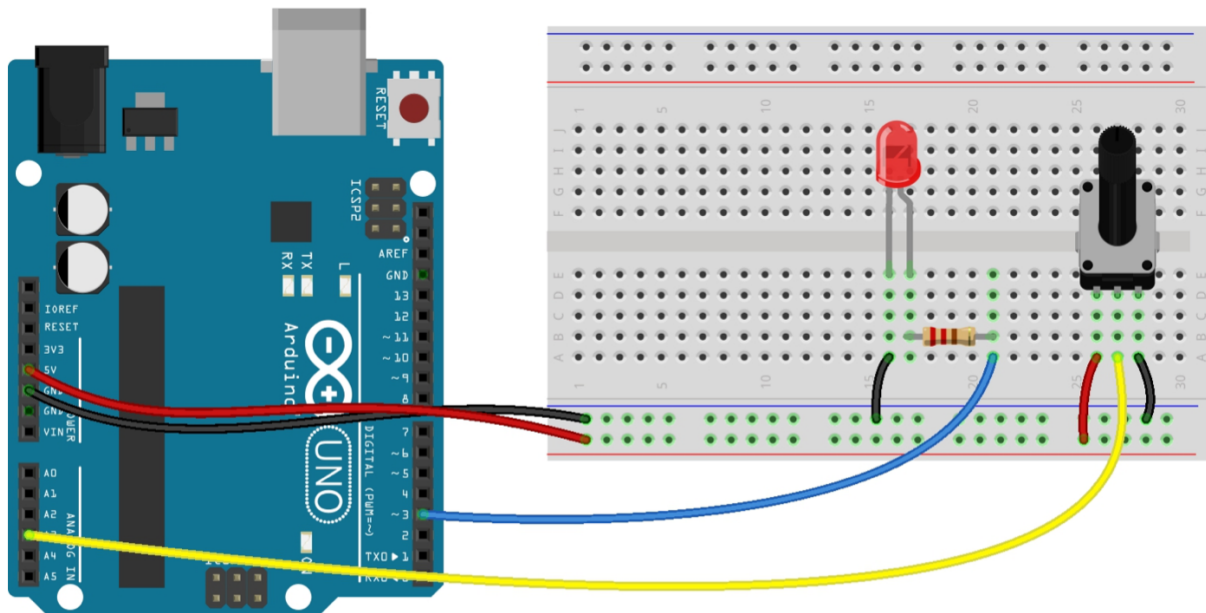
The easy approach: You could use your computer (Windows/Mac) to run the Arduino IDE. For this approach you need to view your output from the Arduino with the Serial Monitor. Your sensor information will be output here when the `Serial.println()` / `Serial.print()` / `Serial.write()` function is used. This can be found with the image below:



Potentiometer

A potentiometer is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. In this example, potentiometer values are read in through an 'Analog In' pin. The values are then used to control the brightness of an LED.

Example Circuit



Code

```
/* FSR simple testing sketch.

Connect one end of FSR to power, the other end to Analog 0.
Then connect one end of a 10K resistor from Analog 0 to ground

For more information see www.ladyada.net/learn/sensors/fsr.html */

int fsrPin = 0;      // the FSR and 10K pulldown are connected to a0
int fsrReading;      // the analog reading from the FSR resistor divider

void setup(void) {
  // We'll send debugging information via the Serial monitor
  Serial.begin(9600);
}

void loop(void) {
  fsrReading = analogRead(fsrPin);

  Serial.print("Analog reading = ");
  Serial.print(fsrReading);      // the raw analog reading

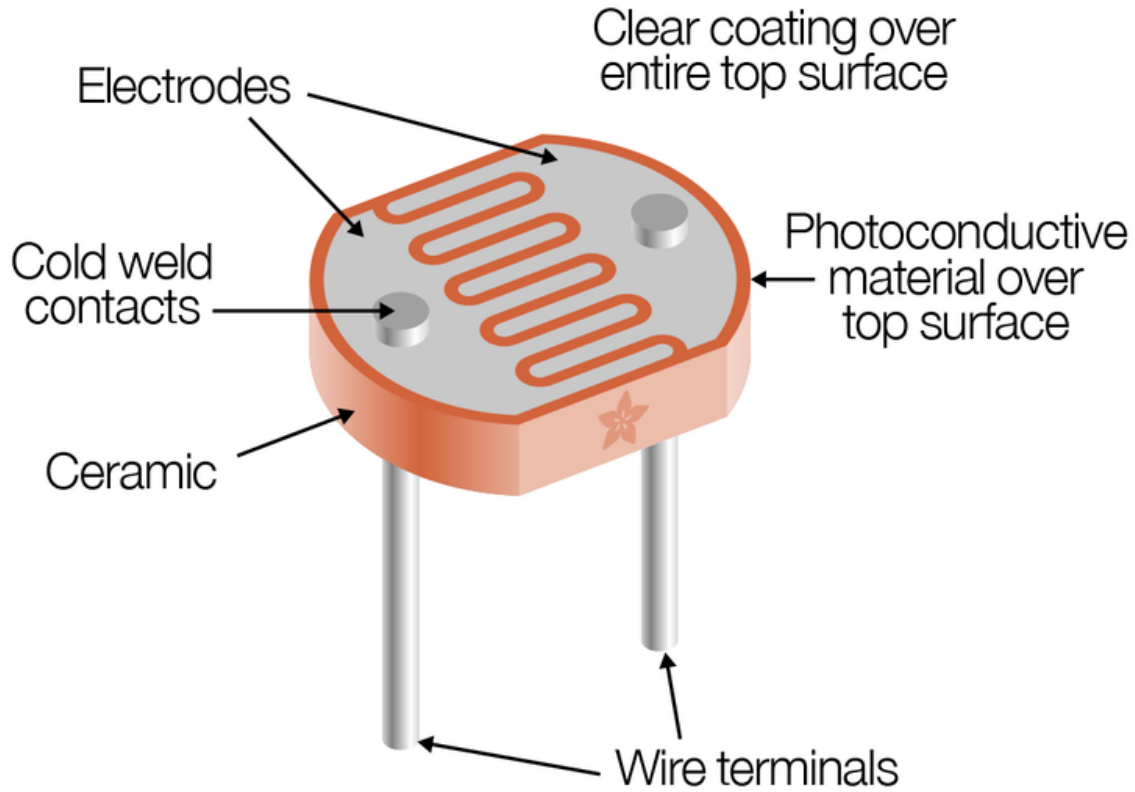
  // We'll have a few thresholds, qualitatively determined
  if (fsrReading < 10) {
    Serial.println(" - No pressure");
  } else if (fsrReading < 200) {
    Serial.println(" - Light touch");
  } else if (fsrReading < 500) {
    Serial.println(" - Light squeeze");
  } else if (fsrReading < 800) {
    Serial.println(" - Medium squeeze");
  } else {
    Serial.println(" - Big squeeze");
  }
  delay(1000);
}
```

Light Sensor

For the light sensing are going to use a photo-resistor or Cadmium-sulfide cell. CdS cells are little light sensors. As the squiggly face is exposed to more light, the resistance goes down. When its light, the resistance is about 5-10K Ω , when dark it goes up to 200K Ω .

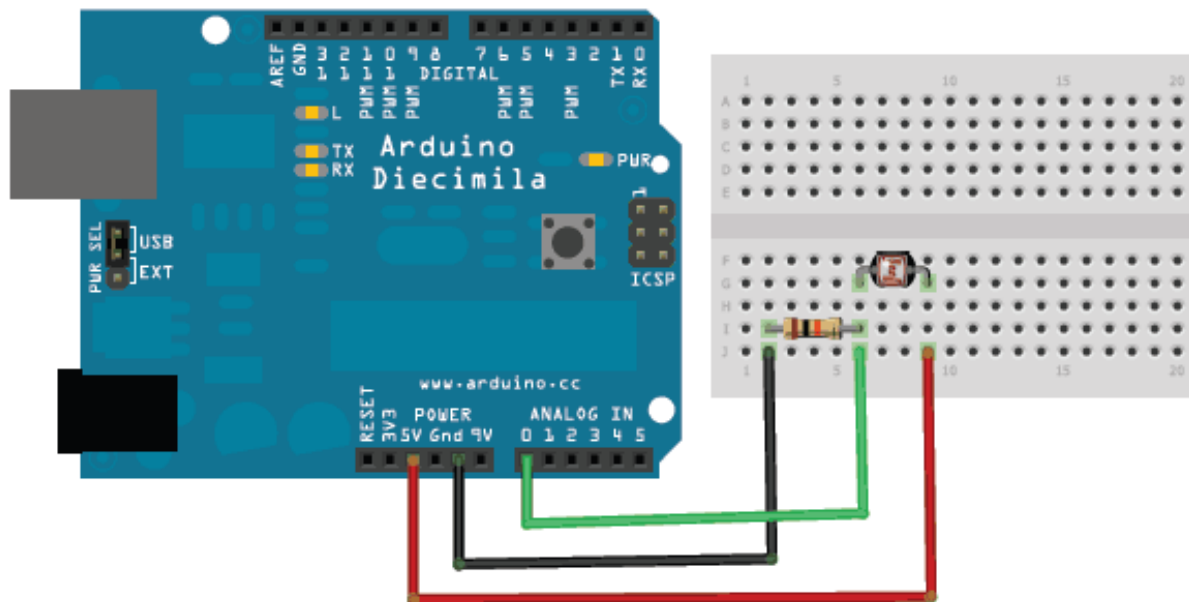
They are very low cost, easy to get in many sizes and specifications, but are very inaccurate. Each photocell sensor will act a little differently than the other, even if they are from the same batch. The variations can be really large, 50% or higher! For this reason, they shouldn't be used to try to determine precise light levels in lux or millicandela. Instead, you can expect to only be able to determine basic light changes.

For most light-sensitive applications like "is it light or dark out", "is there something in front of the sensor (that would block light)", "is there something interrupting a laser beam" (break-beam sensors), or "which of multiple sensors has the most light hitting it", photocells can be a good choice!



Example Circuit

To use, connect one side of the photo cell (either one, its symmetric) to power (for example 5V) and the other side to your microcontroller's analog input pin. Then connect a 10K pull-down resistor from that analog pin to ground. The voltage on the pin will be 2.5V or higher when its light out and near ground when its dark.



Code

In the Arduino IDE you will find under **File** → **Examples** → **10.StarterKit_BasicKit** → **p04ColorMixingLamp** an example sketch that uses three photo-resistors to control three LEDs to create a colour-changing lamp. We challenge you to tweak the code to fit your needs and to read the value from one photo-resistor.

Challenge

Important: You must demonstrate your build & code to the tutor team

Modify the code and circuit to read the value from one photo-resistor and operate 3 LEDs.

Colour Mixing Lamp Code:

```

/*
  Arduino Starter Kit example
  Project 4 - Color Mixing Lamp

  This sketch is written to accompany Project 3 in the
  Arduino Starter Kit

  Parts required:
  1 RGB LED
  three 10 kilohm resistors
  3 220 ohm resistors
  3 photoresistors
  red green and blue colored gels

```

(continues on next page)

(continued from previous page)

```

Created 13 September 2012
Modified 14 November 2012
by Scott Fitzgerald
Thanks to Federico Vanzati for improvements

http://www.arduino.cc/starterKit

This example code is part of the public domain
*/

const int greenLEDPin = 9;    // LED connected to digital pin 9
const int redLEDPin = 10;     // LED connected to digital pin 10
const int blueLEDPin = 11;    // LED connected to digital pin 11

const int redSensorPin = A0;  // pin with the photoresistor with the red gel
const int greenSensorPin = A1; // pin with the photoresistor with the green gel
const int blueSensorPin = A2; // pin with the photoresistor with the blue gel

int redValue = 0; // value to write to the red LED
int greenValue = 0; // value to write to the green LED
int blueValue = 0; // value to write to the blue LED

int redSensorValue = 0; // variable to hold the value from the red sensor
int greenSensorValue = 0; // variable to hold the value from the green sensor
int blueSensorValue = 0; // variable to hold the value from the blue sensor

void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);

  // set the digital pins as outputs
  pinMode(greenLEDPin, OUTPUT);
  pinMode(redLEDPin, OUTPUT);
  pinMode(blueLEDPin, OUTPUT);
}

void loop() {
  // Read the sensors first:

  // read the value from the red-filtered photoresistor:
  redSensorValue = analogRead(redSensorPin);
  // give the ADC a moment to settle
  delay(5);
  // read the value from the green-filtered photoresistor:
  greenSensorValue = analogRead(greenSensorPin);
  // give the ADC a moment to settle
  delay(5);
  // read the value from the blue-filtered photoresistor:
  blueSensorValue = analogRead(blueSensorPin);

  // print out the values to the serial monitor
  Serial.print("raw sensor Values \t red: ");
  Serial.print(redSensorValue);
  Serial.print("\t green: ");
  Serial.print(greenSensorValue);
  Serial.print("\t Blue: ");

```

(continues on next page)

(continued from previous page)

```
Serial.println(blueSensorValue);

/*
In order to use the values from the sensor for the LED,
you need to do some math. The ADC provides a 10-bit number,
but analogWrite() uses 8 bits. You'll want to divide your
sensor readings by 4 to keep them in range of the output.
*/

redValue = redSensorValue / 4;
greenValue = greenSensorValue / 4;
blueValue = blueSensorValue / 4;

// print out the mapped values
Serial.print("Mapped sensor Values \t red: ");
Serial.print(redValue);
Serial.print("\t green: ");
Serial.print(greenValue);
Serial.print("\t Blue: ");
Serial.println(blueValue);

/*
Now that you have a usable value, it's time to PWM the LED.
*/
analogWrite(redLEDPin, redValue);
analogWrite(greenLEDPin, greenValue);
analogWrite(blueLEDPin, blueValue);
}
```

Acknowledgements

Based on [this Adafruit guide](#) and [Adafruit's Photocell's page](#).

2.2.4 Actuators

In this section we are going to learn about actuators and how to control them.

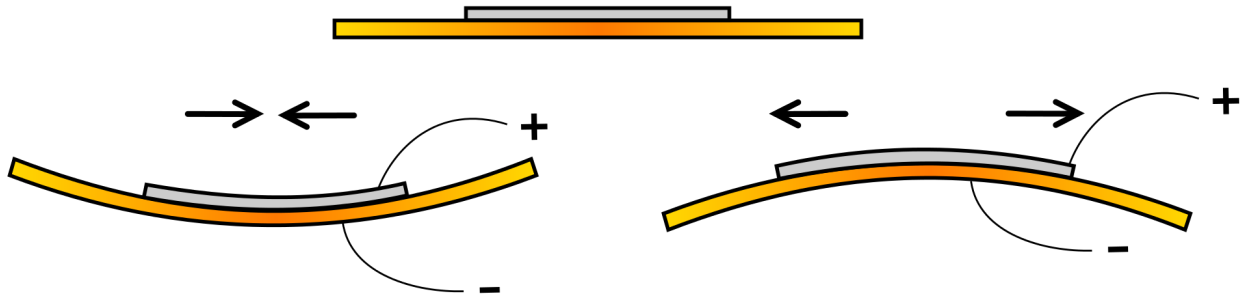
1. Piezo Buzzer
2. Servo Motor

At the beginning of this session you should have collected a kit that is made of:

- [Buzzer](#)
- [Servo Motor](#)

Piezo Buzzer

A [piezoelectric speaker](#) (sometimes colloquially called a “piezo”) or buzzer is a loudspeaker that uses the piezoelectric effect for generating sound. The initial mechanical motion is created by applying a voltage to a piezoelectric material, and this motion is typically converted into audible sound using diaphragms and resonators.

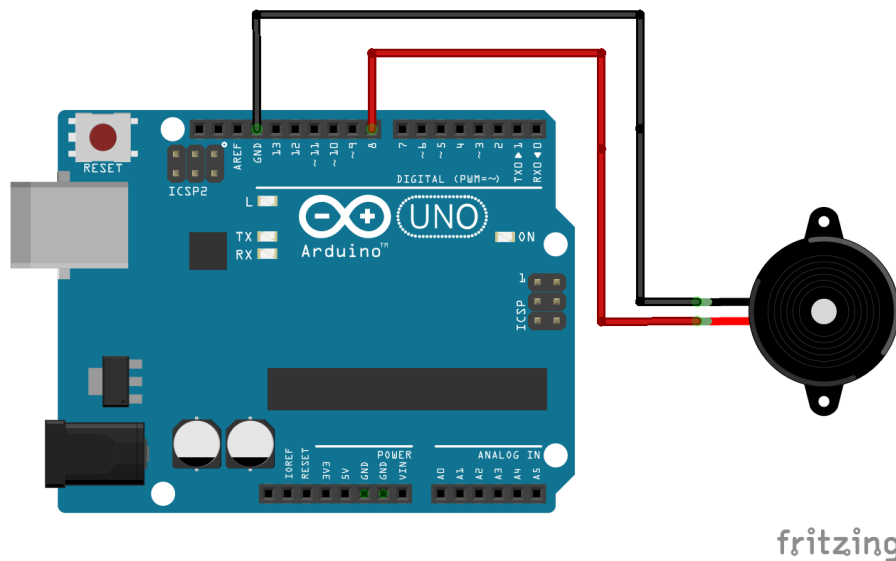


When fixed to a metallic diaphragm and excited with an alternating voltage, the diameter of the disc varies by a small amount, this causes dishing of the diaphragm which gives a much louder sound.

Example Circuit

From the kit you are going to need:

- Buzzer
- Jumper Wires
- Arduino



Code

For the code you can use the **Example** → **2.Digital** → **toneMelody**. Play around with the sketch and `tone()` command. You may find it useful for whenever you want to make musical notes. More information on the pitches Arduino library and tone command [here](#). Try also **Example** → **10.StarterKit_BasicKit** → **p06 LightTheremin**.

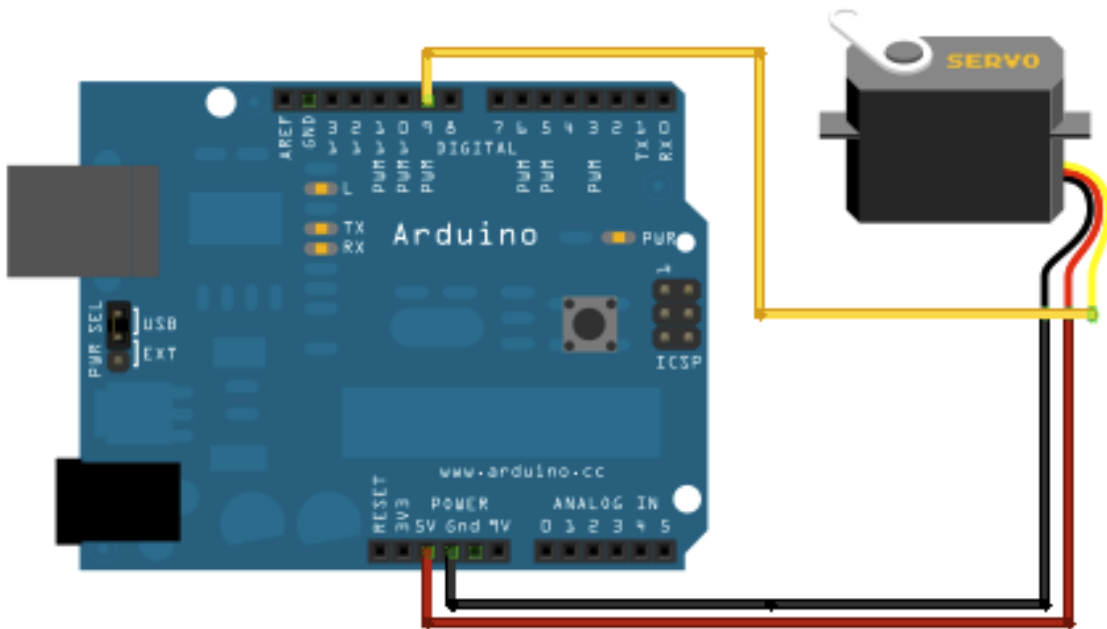
Servo Motor

A [servo motor](#) is a rotary actuator or linear actuator that allows for precise control of angular or linear position, velocity and acceleration. It consists of a suitable motor coupled to a sensor for position feedback. It also requires a relatively sophisticated controller, often a dedicated module designed specifically for use with servomotors.

Example Circuit

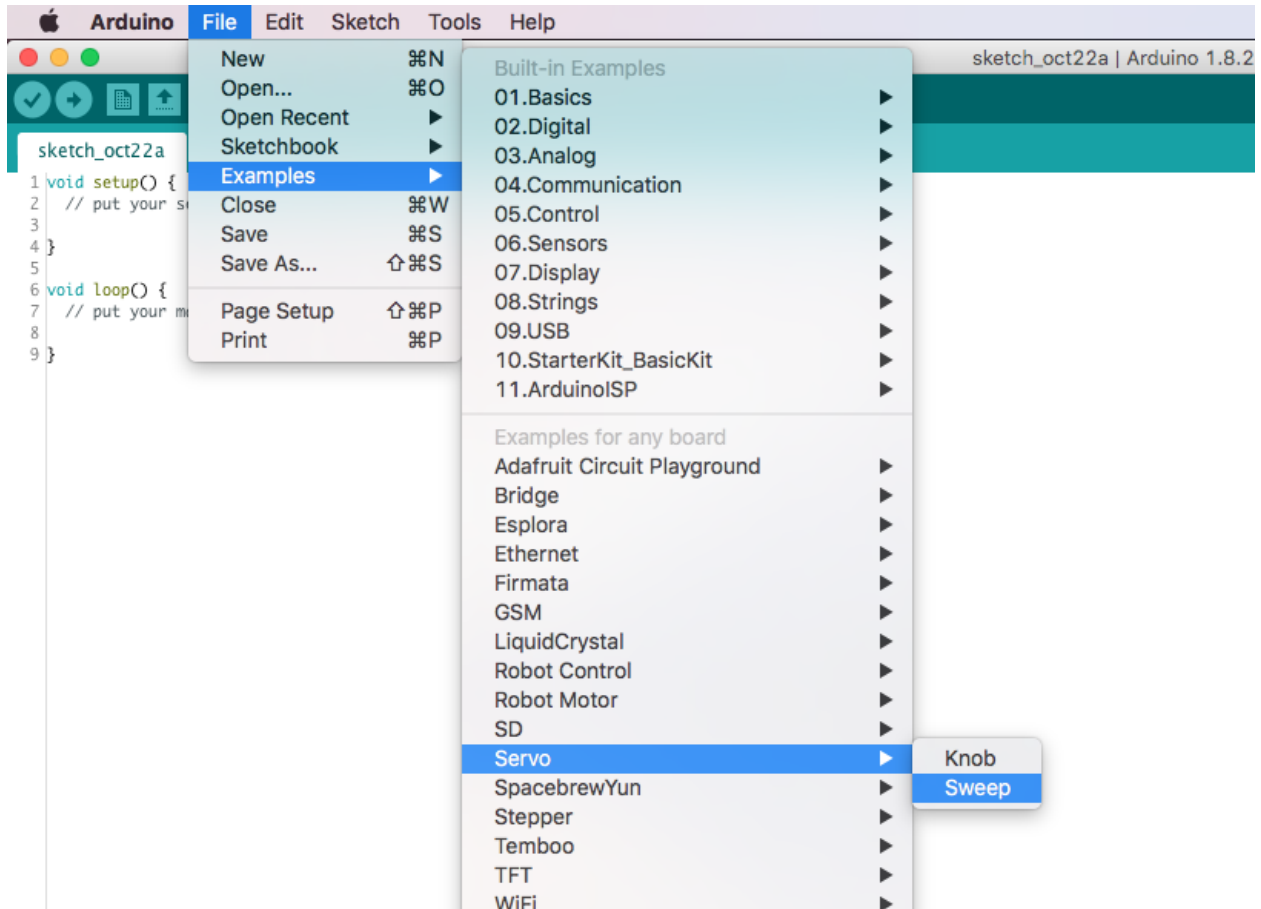
From the kit you are going to need:

- Servo Motor
- Jumper Wires
- Arduino



Code

For this example you are going to use the built-in [servo library](#) by Arduino and we are going to use the built-in sketch **Example** → **Servo** → **Sweep**.



```

/* Sweep
  by BARRAGAN <http://barraganstudio.com>
  This example code is in the public domain.

  modified 8 Nov 2013
  by Scott Fitzgerald
  http://www.arduino.cc/en/Tutorial/Sweep
  */

#include <Servo.h>

Servo myservo;  // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0;    // variable to store the servo position

void setup() {
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}

void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
}

```

(continues on next page)

(continued from previous page)

```
for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
  myservo.write(pos);                // tell servo to go to position in variable 'pos'
  delay(15);                          // waits 15ms for the servo to reach the position
}
```

Challenge

Important: You must demonstrate your build & code to the tutor team

We challenge you to combine previous sketches (Button, Sweep, tone) to create one that:

- With the press of a button, sweeps a servo in one direction
- With the press of a second button, sweeps the same servo in the opposite direction
- When the servo has swept its maximum travel, the buzzer should sound (beep).

Acknowledgements

- [Adafruit Learn](#)
 - [Wikiwand Piezoelectric Speaker](#)
-

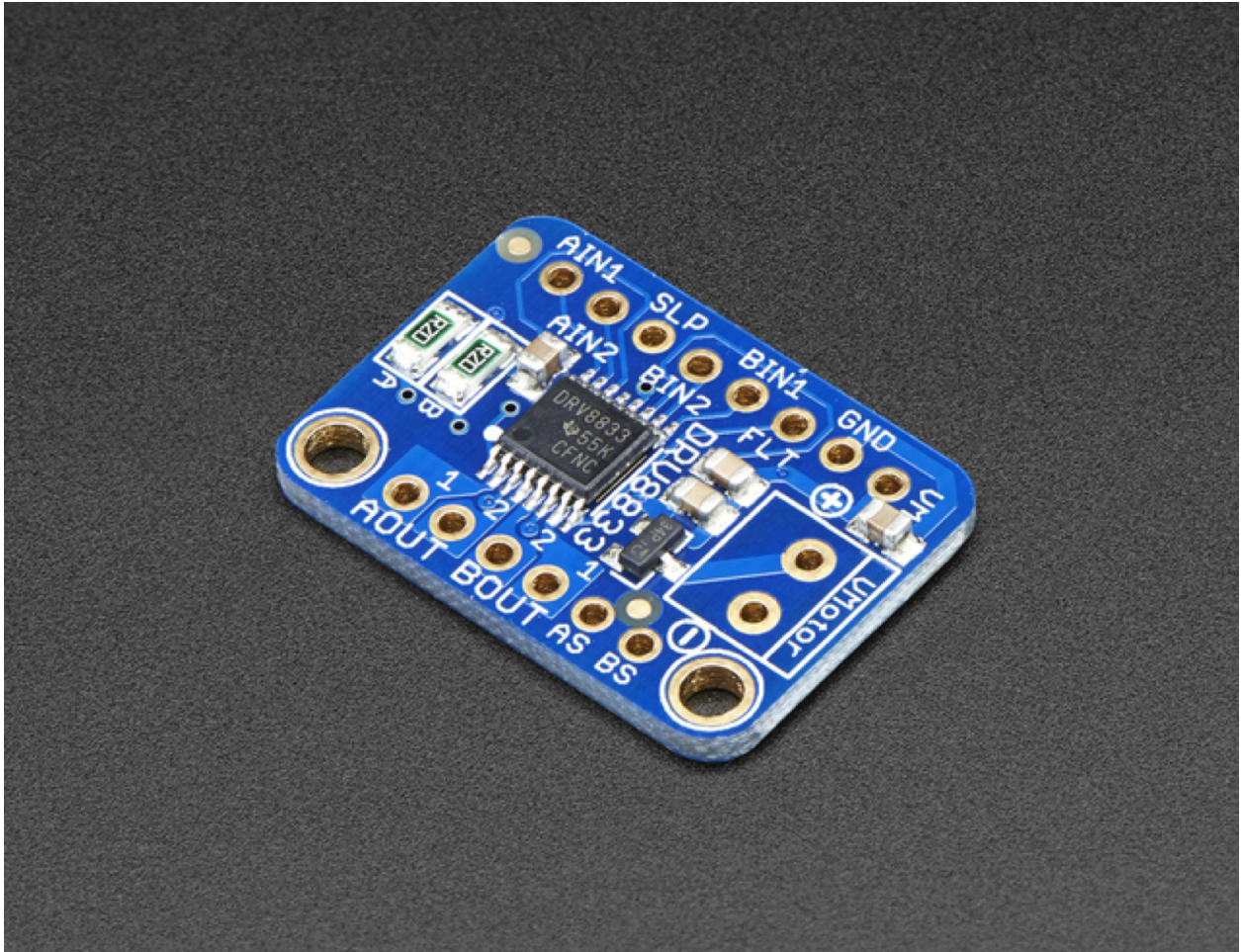
2.2.5 Combined Sense & Actuation

Material Covered

- DC motor control (speed and direction)
- Capacitive touch

Motor

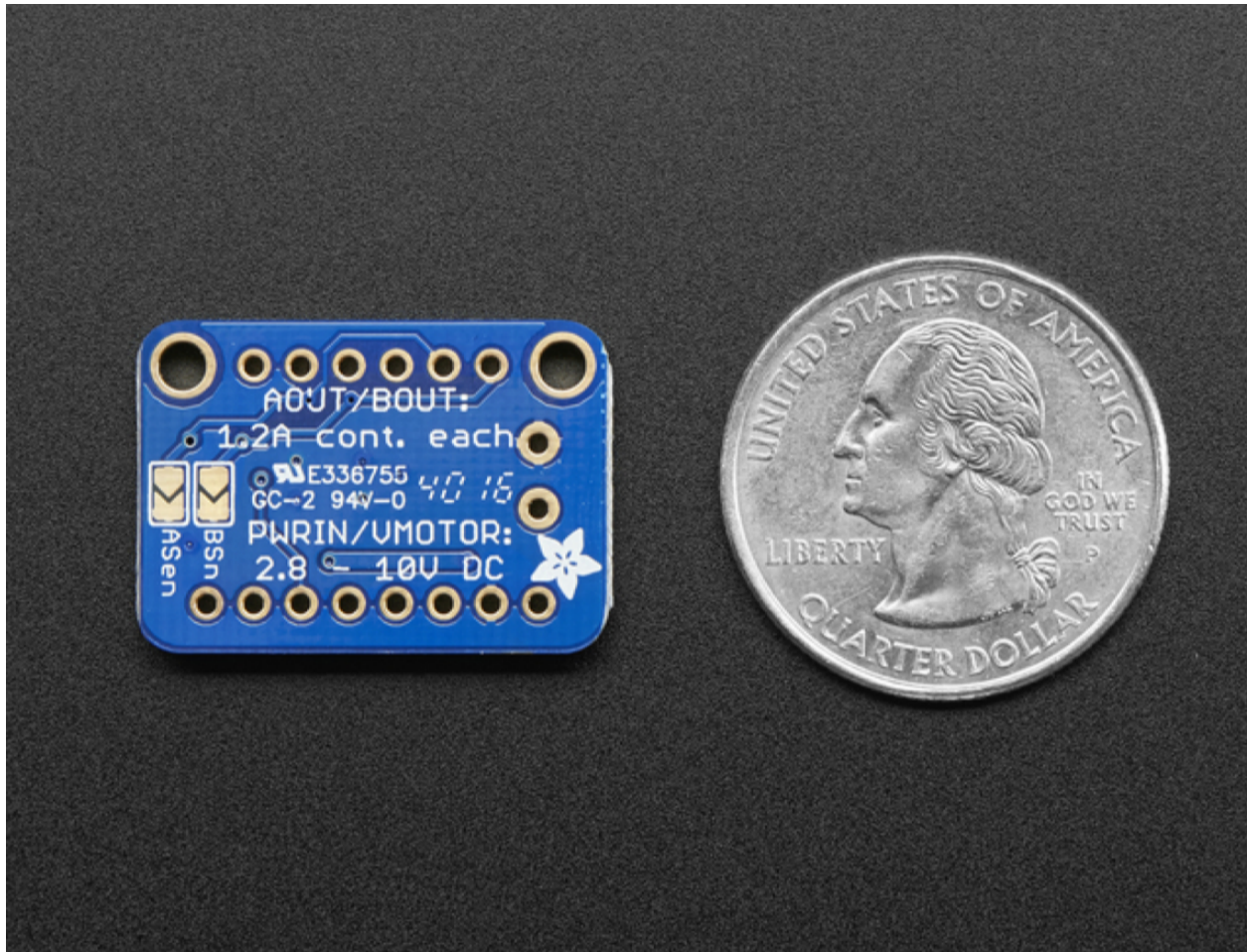
Intro to Motor Driver



A motor driver is a current amplifier; the function of motor drivers is to take a low-current control signal and transform it into a higher-current signal that can drive a motor.

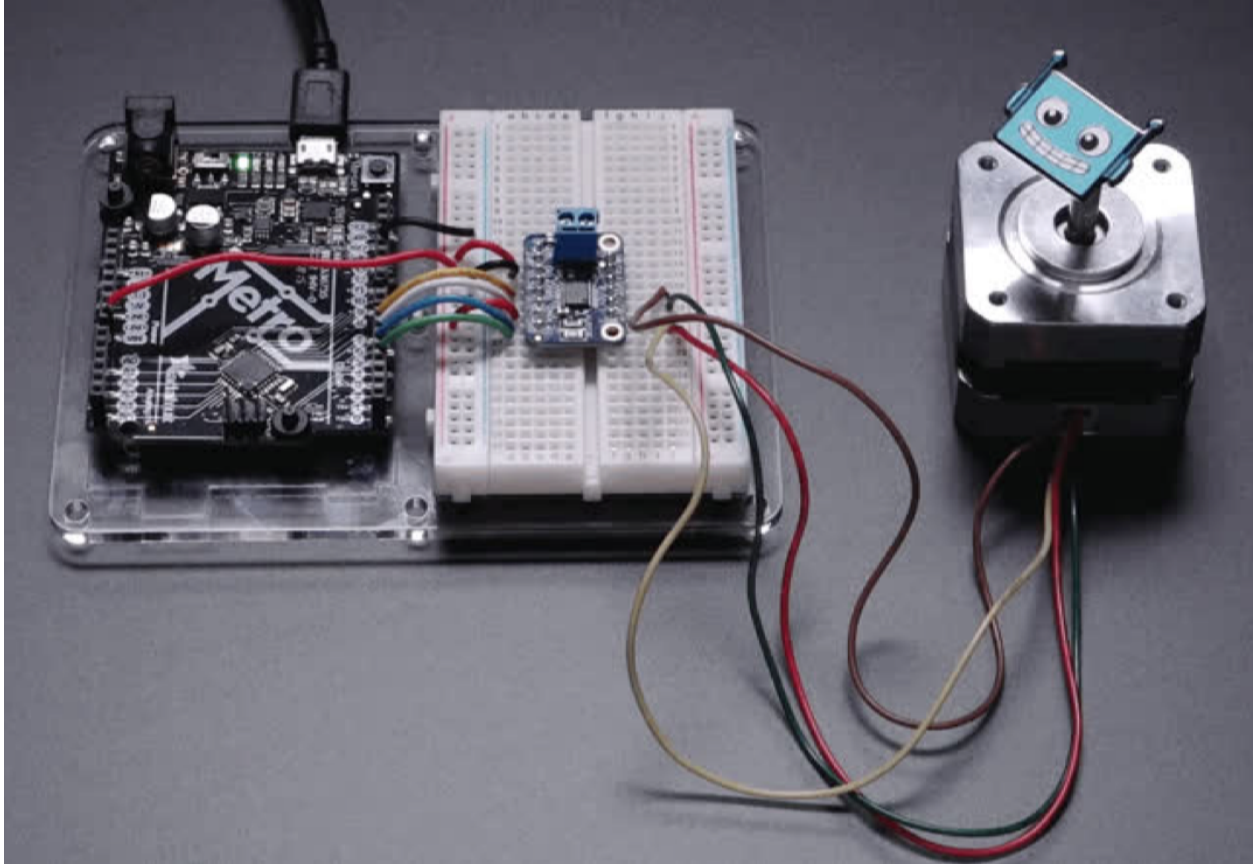
The Adafruit DRV8833 Motor Driver contains two full H-bridges (four half H-bridges). That means you can spin two DC motors bi-directionally or step one bi-polar or uni-polar stepper with up to 1.2A per channel. This motor driver chip is a nice alternative to the TB6612 driver.

The DRV8833 chip is better for low voltage uses (it can run from 2.7V up to 10.8V motor power) and has built in current limiting capability. The driver is set it up for 1A current limiting so you don't get more than 2A per chip, but you can also disable the current limiting, or change it to a different limit!



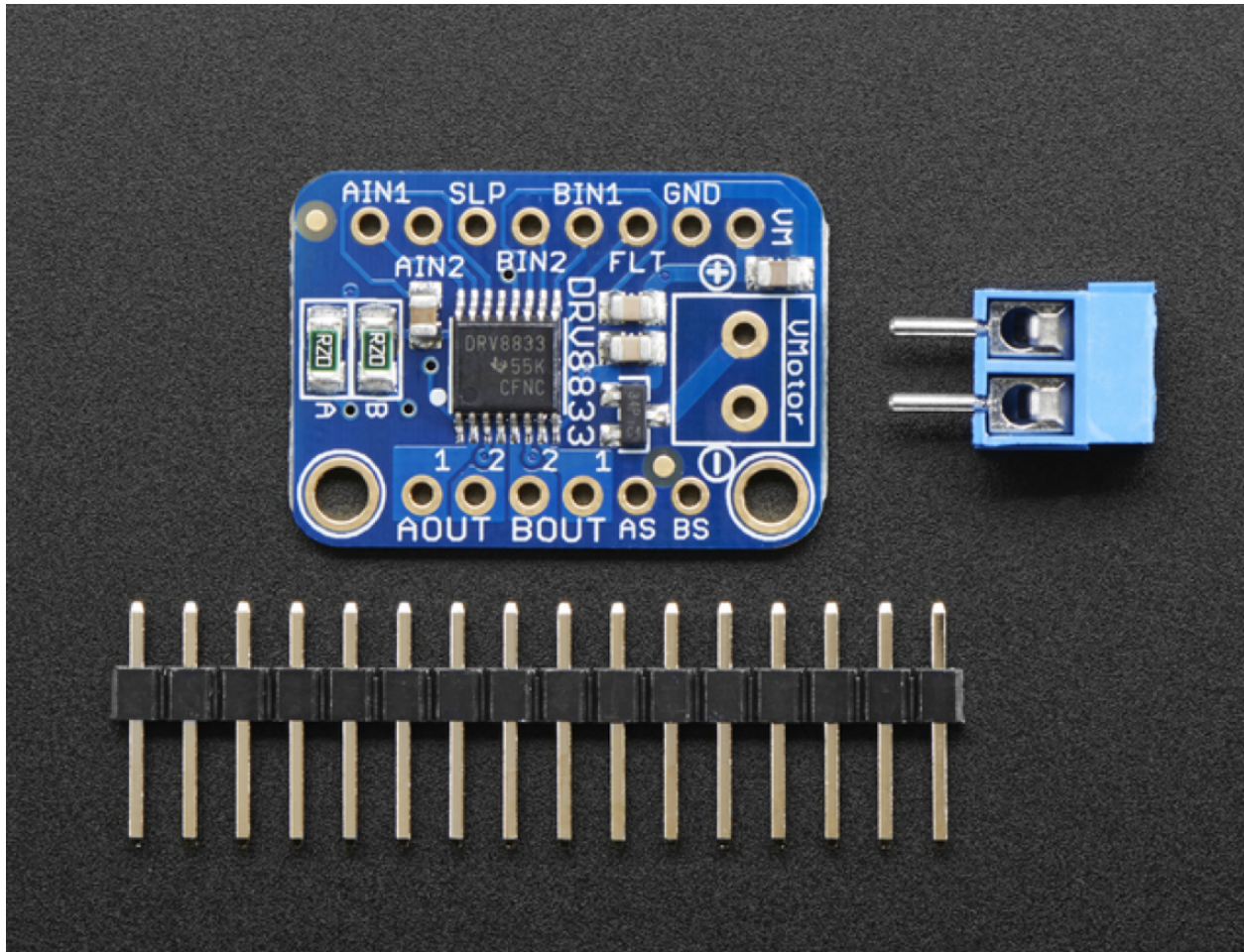
The DRV8833 chip is soldered to a breakout board, with a polarity protection FET on the motor voltage input. Only use with motors that draw 1.2 Amp or less. This is the limit of the chip. They can handle a peak of 2A but only for a short amount of time.

The Driver comes with built in kick-back diodes internally so you do not have to worry about the inductive kick damaging your project or driver! You also don't have to worry as much about burning out the chip with overdriving since there is current limiting.



There's two digital inputs per H-bridge (one for each half of the bridge), you can PWM one of the inputs to control motor speed. Runs at 2.7V-10.8V logic/motor power. The motor voltage is the same as the logic voltage, but logic voltage from 2.7V or greater will work so no need to worry if you are powering the motors from 9V and using 3.3V logic. [For higher voltages, check out the TB6612.](#) [For much higher voltages and currents check out the DRV8871!](#)

The Motor Driver Comes as one assembled and tested breakout board plus a small strip of header. We've done the soldering to attach the header onto the breakout PCB.



Pinouts

Power Pins

- **Vmotor** - This is the voltage for the motors, not for the logic level. Keep this voltage between 2.7V and 10.8V. This power supply will get noisy so if you have a system with analog readings or RF other noise-sensitive parts, you may need to keep the power supplies separate (or filtered!). The terminal block has a simple polarity protection on the + pin that feeds into VM. The VM pin is not protected, but VMotor is!
- **GND** - This is the shared logic and motor ground. All grounds are connected.

Signal in Pins

These are all “2.7V or higher logic level” inputs:

- **AIN1, AIN2** - these are the two inputs to the Motor A H-bridges. If you want to use speed control, PWM the pin that is normally high. If you don't need PWM control, connect them to logic high/low.
- **BIN1, BIN2** - these are the two inputs to the Motor B H-bridges. If you want to use speed control, PWM the pin that is normally high. If you don't need PWM control, connect them to logic high/low.
- **FLT** - This is the **Fault** output, which will drive low if there's a thermal shutdown or overcurrent. Note it is open drain so connect a pull-up resistor to your desired logic voltage!

- **SLP** - this is the sleep pin for quickly disabling the driver. **By default it is pulled low with an internal 500K resistor, so the chip is not active!** Connect to a logic high pin (or 5V supply) either directly or via a pull-up resistor to enable the motor control!

Current Limit Pins

The DRV8833 can perform current limiting for each motor H-bridge. Basically a resistor is connected between Asen and ground to set the Motor A limit (ditto for Bsen and Motor B)

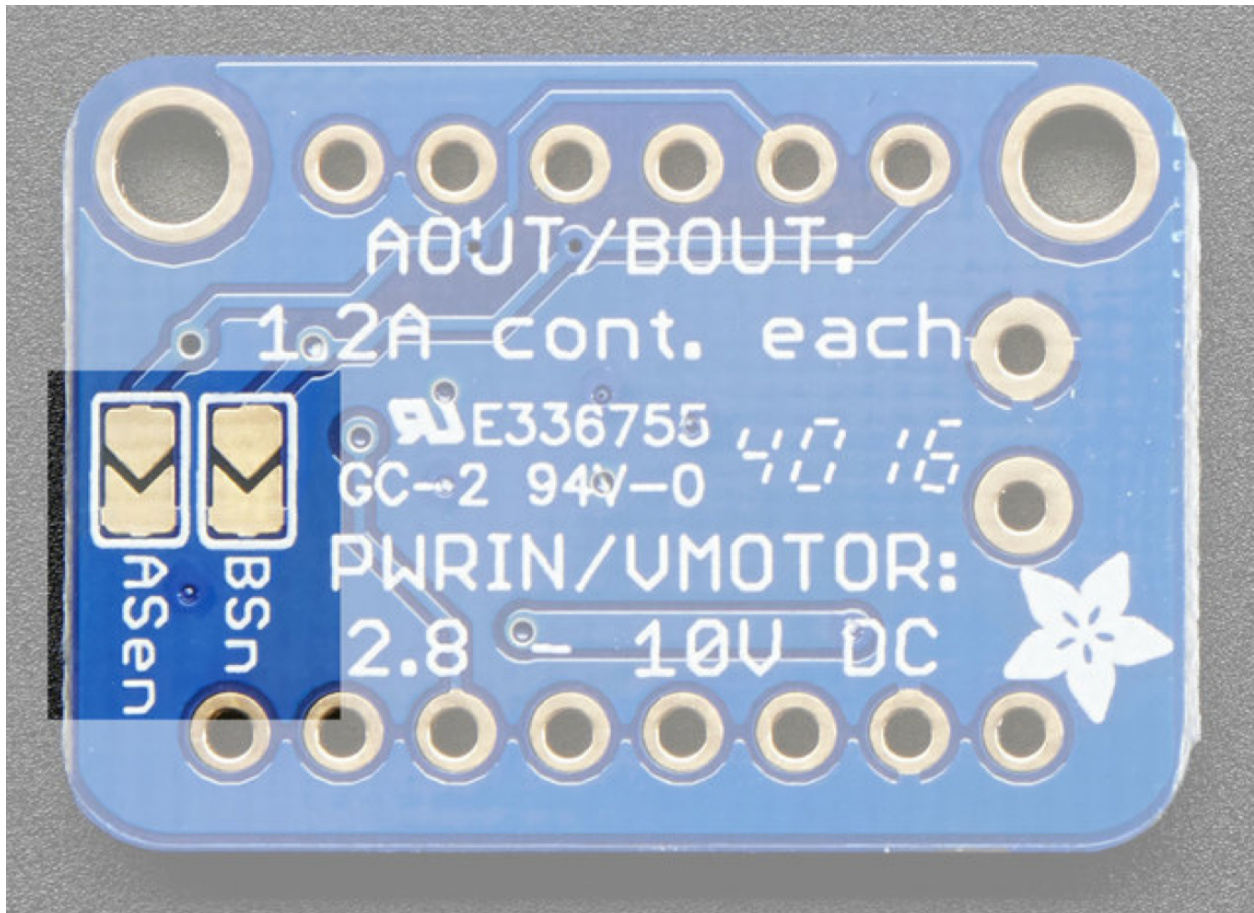
The current limiting rule is:

$$\text{LimitCurrent(amps)} = \frac{0.2V}{R_{SENSE}}$$

By default, there are two 1206-sized $0.2\ \Omega$ resistors on the board for both motors.

If you'd like to raise the limit, you can put a $0.2\ \Omega$ from Asen to ground, which will then make the RSENSE equal to $0.1\ \Omega$ (2 parallel $0.2\ \Omega$ resistors) for a limit of 2A.

You can also *disable* current limiting by soldering *closed* the two jumpers on the back.



If you want a lower current limit, remove/destroy the $0.2\ \Omega$ resistor on the board and add your own resistor value between Asen or Bsen and ground.

Motor Out Pins

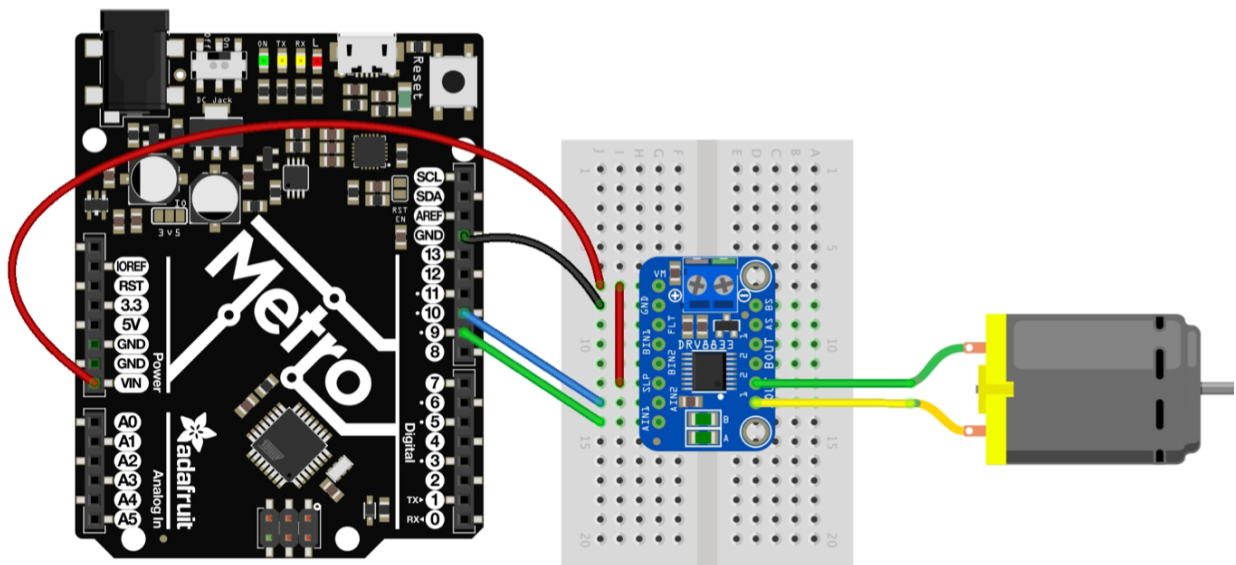
These are motor power outputs

- **Motor A** - these are the two outputs for motor A, controlled by AIN1 and AIN2
- **Motor B** - these are the two outputs for motor B, controlled by BIN1 and BIN2

DC Motor

A **DC motor** is any of a class of rotary electrical machines that converts direct current electrical energy into mechanical energy. The most common types rely on the forces produced by magnetic fields. Nearly all types of DC motors have some internal mechanism, either electromechanical or electronic, to periodically change the direction of current flow in part of the motor.

Assembly



- AIN1 to pin 9
- AIN2 to pin 10
- SLP to 5V
- GND to Arduino GND
- Vm to 5V
- motorA to DC motor

Code

For this sketch copy and paste the following code. You should see the motor speed up in one direction, slow down and then rotate in the opposite direction:

```
#define MOTOR_AIN1 9
#define MOTOR_AIN2 10

int MAX_PWM = 255;
int MIN_PWM = 50;

void setup() {
  Serial.begin(9600);
  pinMode(MOTOR_AIN1, OUTPUT);
  pinMode(MOTOR_AIN2, OUTPUT);
}

void loop() {
  // ramp up forward
  digitalWrite(MOTOR_AIN1, LOW);

  for (int i=MIN_PWM; i<MAX_PWM; i++) {
    analogWrite(MOTOR_AIN2, i);
    delay(10);
  }

  // forward full speed for one second
  delay(1000);

  // ramp down forward
  for (int i=MAX_PWM; i>=MIN_PWM; i--) {
    analogWrite(MOTOR_AIN2, i);
    delay(10);
  }

  // ramp up backward

  digitalWrite(MOTOR_AIN2, LOW);

  for (int i=MIN_PWM; i<MAX_PWM; i++) {
    analogWrite(MOTOR_AIN1, i);
    delay(10);
  }

  // backward full speed for one second
  delay(1000);

  // ramp down backward
  for (int i=MAX_PWM; i>=MIN_PWM; i--) {
    analogWrite(MOTOR_AIN1, i);
    delay(10);
  }
}
```

Challenge 1

Important: You must demonstrate your build & code to the tutor team

We challenge you to combine the previous sketch with the potentiometer (Pot) sketch from the previous Chapters such that the Pot controls the speed of the motor. No need to reverse it!

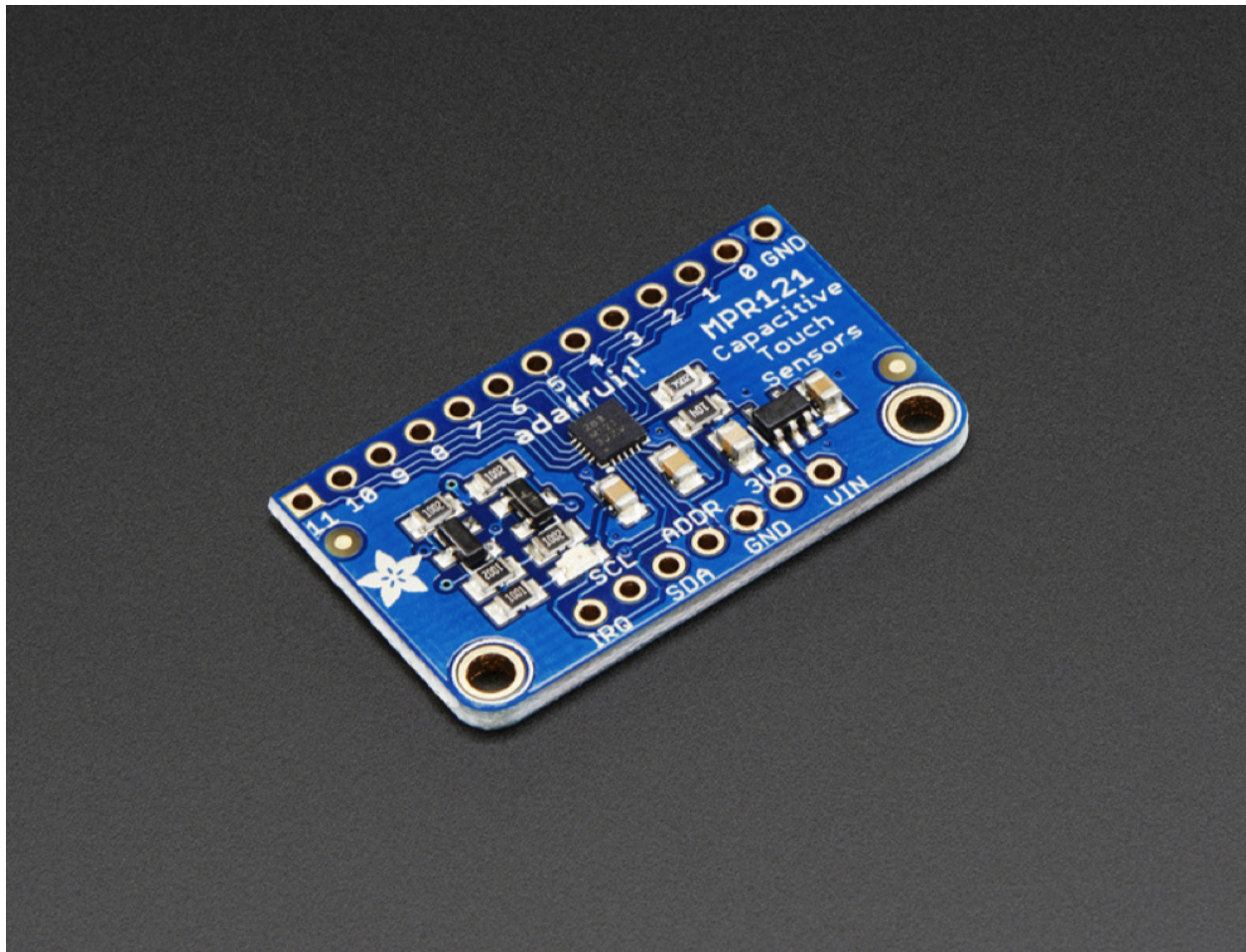
Challenge 2

Important: You must demonstrate your build & code to the tutor team

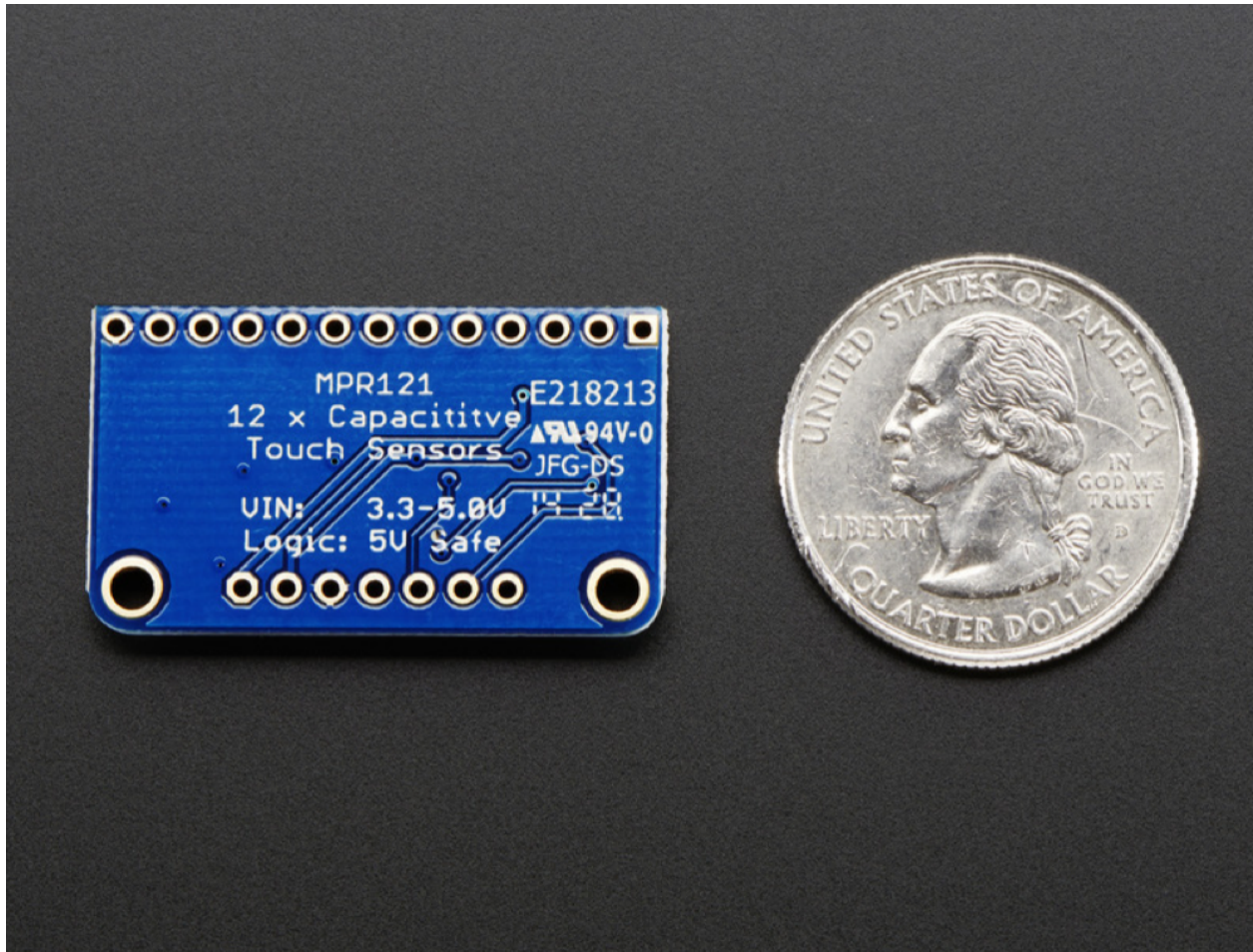
We challenge you to combine the first motor sketch, with your sketch from Challenge 1 and the photocell sketches from previous Chapters:

- Use two photocells to control the direction and speed of two brushed DC motors
- Assume that you want to use this circuit and software, in order to cause a robot to move toward a light source

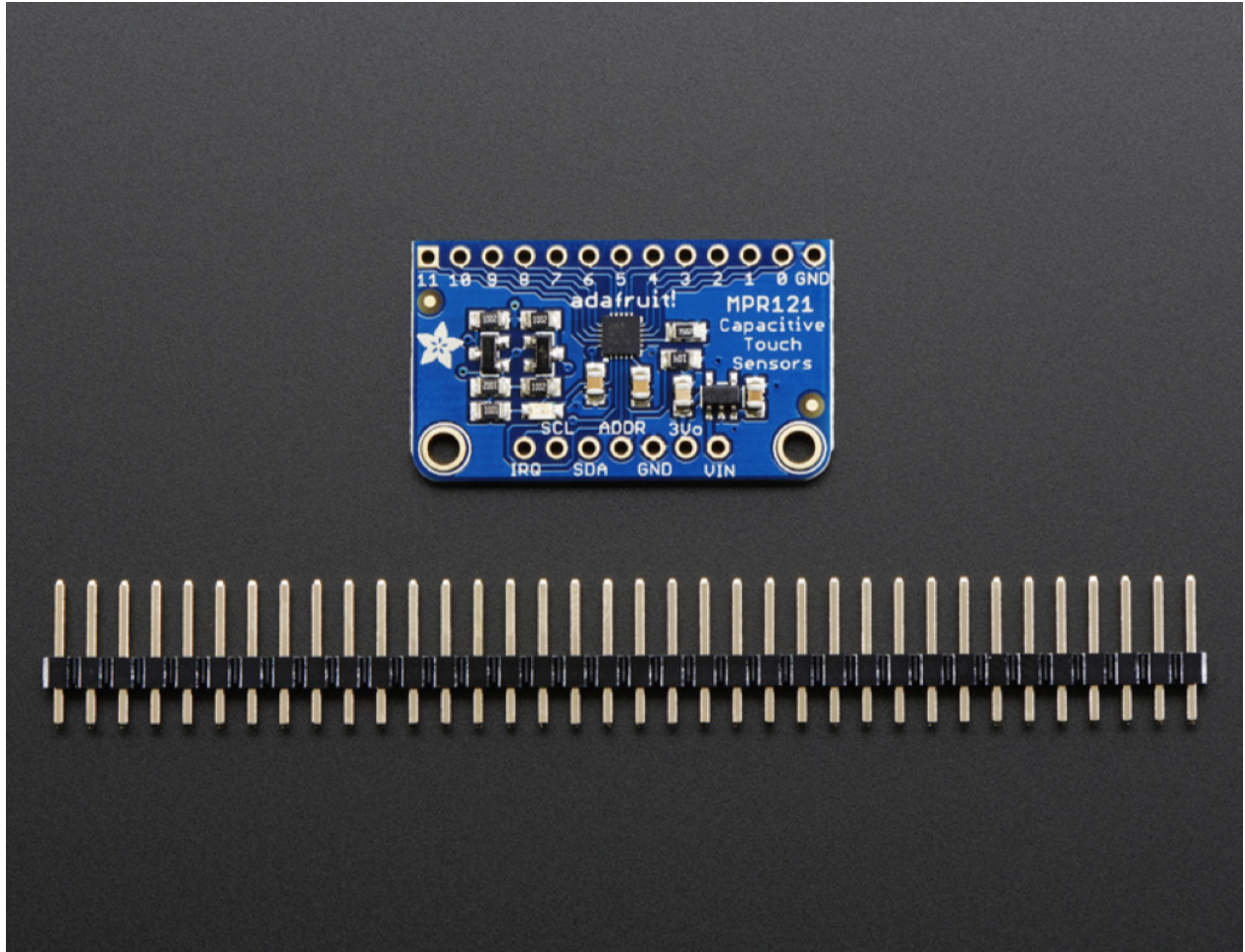
Capacitive Touch Sensor



Add lots of touch sensors to your next microcontroller project with this easy-to-use 12-channel capacitive touch sensor breakout board, starring the MPR121. This chip can handle up to 12 individual touch pads.

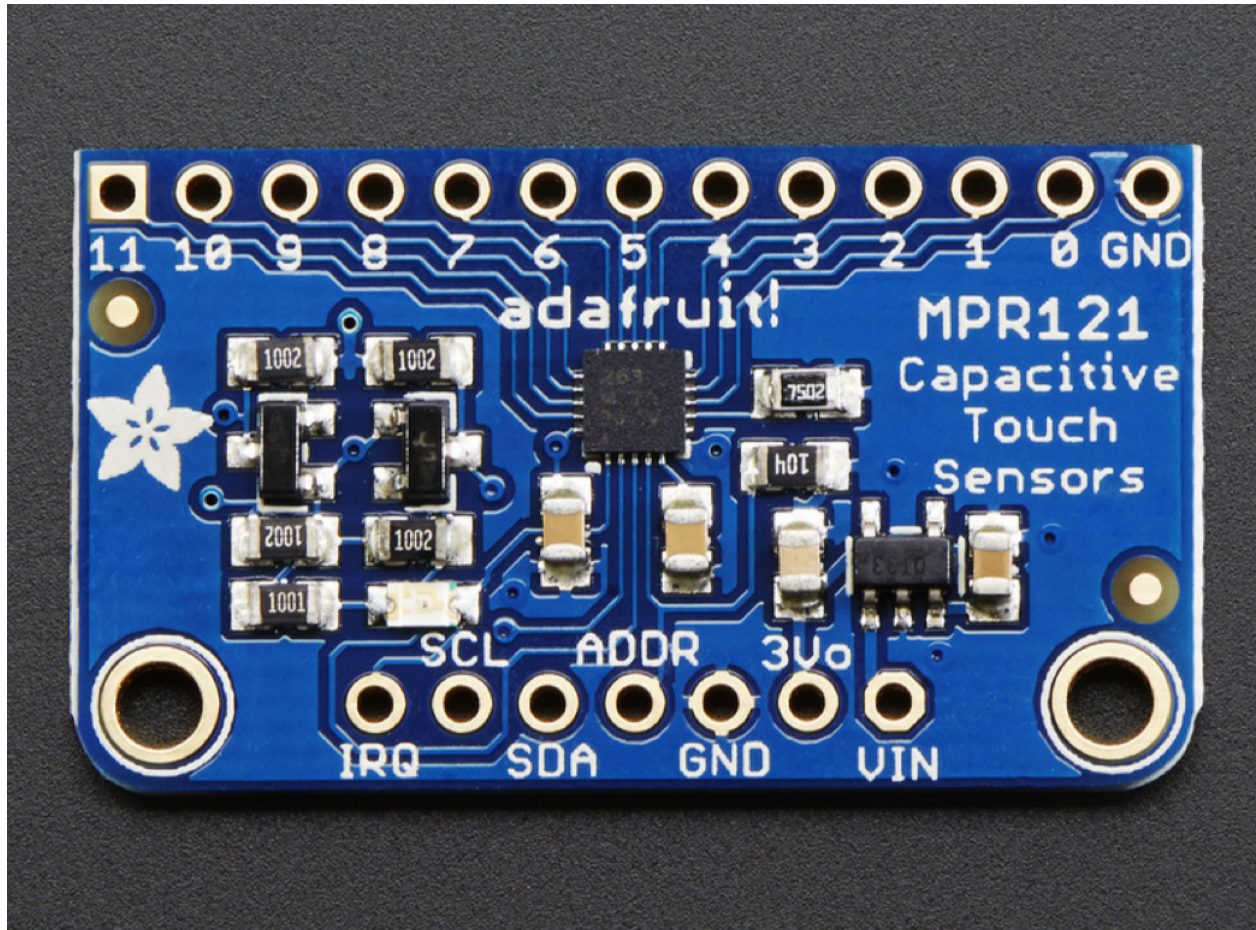


The MPR121 has support for only I2C, which can be implemented with nearly any microcontroller. You can select one of 4 addresses with the ADDR pin, for a total of 48 capacitive touch pads on one I2C 2-wire bus. Using this chip is a lot easier than doing the capacitive sensing with analog inputs: it handles all the filtering for you and can be configured for more or less sensitivity.



The breakout board provides a 3V regulator and I2C level shifting so its safe to use with any 3V or 5V microcontroller/ processor like Arduino. There is an LED onto the IRQ line so it will blink when touches are detected, making debugging by sight a bit easier on you. For contacts, we suggest using copper foil or pyralux, then solder a wire that connects from the foil pad to the breakout.

Pinouts



The little chip in the middle of the PCB is the actual MPR121 sensor that does all the capacitive sensing and filtering. The breakout board comes with all the extra components you need to get started, and ‘break out’ all the other pins you may want to connect to onto the PCB.

Power Pins

The sensor on the breakout requires 3V power. Since many customers have 5V microcontrollers like Arduino, we tossed a 3.3V regulator on the board. Its ultra-low dropout so you can power it from 3.3V-5V

- **Vin** - this is the power pin. Since the chip uses 3 VDC, we have included a voltage regulator on board that will take 3-5VDC and safely convert it down. To power the board, give it the same power as the logic level of your microcontroller - e.g. for a 5V micro like Arduino, use 5V
- **3Vo** - this is the 3.3V output from the voltage regulator, you can grab up to 100mA from this if you like
- **GND** - common ground for power and logic

I2C Pins

Don't worry too much about how these work - it will be covered in a later Chapter!

- **SCL** - I2C clock pin, connect to your microcontrollers I2C clock line.

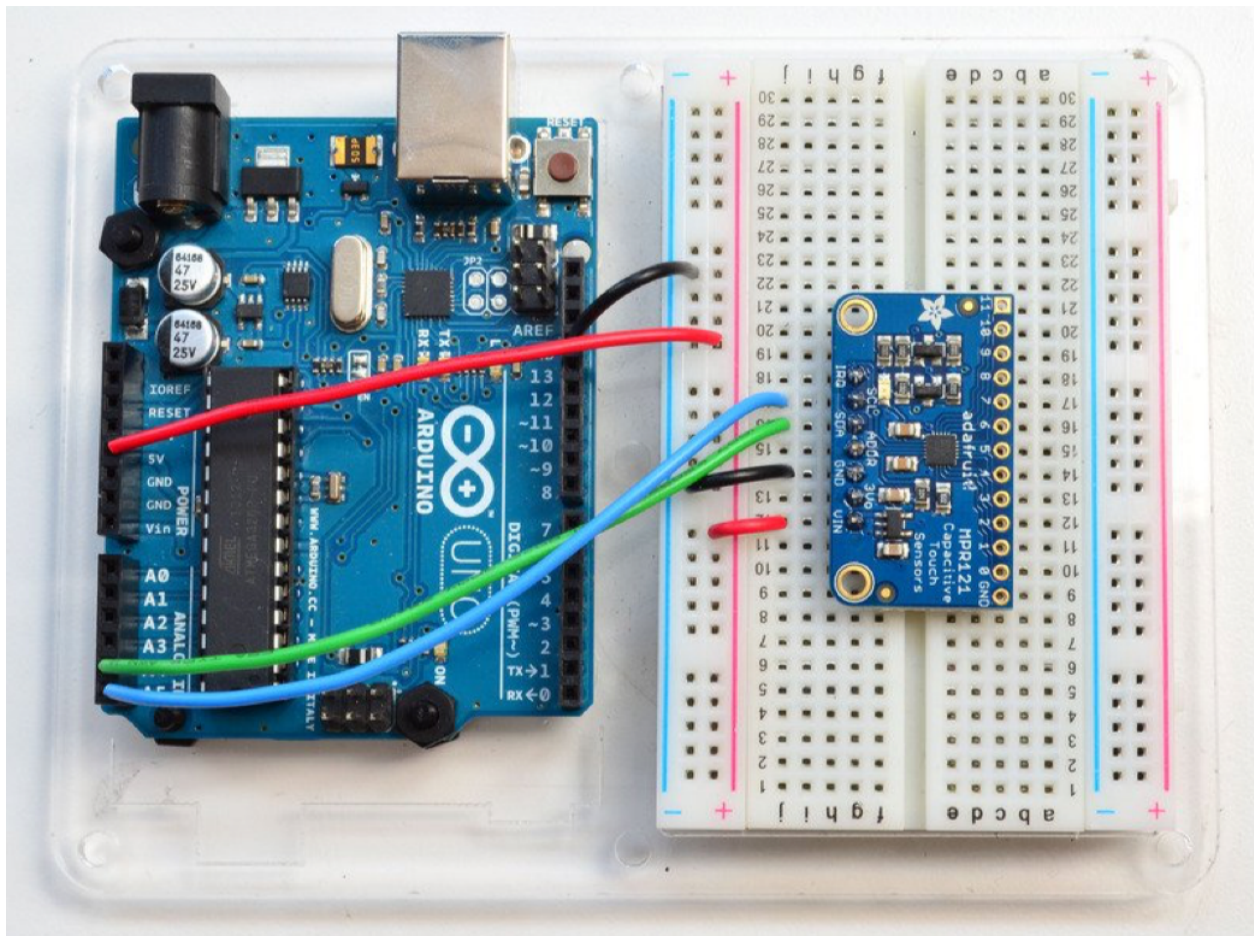
- **SDA** - I2C data pin, connect to your microcontrollers I2C data line.

IRQ and ADDR Pins

- **ADDR** is the I2C address select pin. By default this is pulled down to ground with a 100K resistor, for an I2C address of 0x5A. You can also connect it to the 3Vo pin for an address of 0x5B, the SDA pin for 0x5C or SCL for address 0x5D
- **IRQ** is the Interrupt Request signal pin. It is pulled up to 3.3V on the breakout and when the sensor chip detects a change in the touch sense switches, the pin goes to 0V until the data is read over i2c

Wiring

You can easily wire this breakout to any microcontroller, we'll be using an Arduino. For another kind of microcontroller, just make sure it has I2C, then port the code - its pretty simple stuff!



- Connect **Vin** to the power supply, 3-5V is fine. Use the same voltage that the microcontroller logic is based off of. For most Arduinos, that is 5V
- Connect **GND** to common power/data ground
- Connect the **SCL** pin to the I2C clock **SCL** pin on your Arduino. On an UNO & '328 based Arduino, this is also known as **A5**, on a Mega it is also known as **digital 21** and on a Leonardo/Micro, **digital 3**

- Connect the **SDA** pin to the I2C data **SDA** pin on your Arduino. On an UNO & '328 based Arduino, this is also known as **A4**, on a Mega it is also known as **digital 20** and on a Leonardo/Micro, **digital 2**

The MPR121 **ADDR** pin is pulled to ground and has a default I2C address of 0x5A You can adjust the I2C address by connecting **ADDR** to other pins:

- ADDR not connected: 0x5A
- ADDR tied to 3V: 0x5B
- ADDR tied to SDA: 0x5C
- ADDR tied to SCL: 0x5D

We suggest sticking with the default for the test demo, you can always change it later.

Download Adafruit_MPR121

To begin reading sensor data, you will need to [download Adafruit_MPR121_Library from our github repository](#). You can do that by visiting the GitHub repo and manually downloading or, easier, just click this button to download the zip:

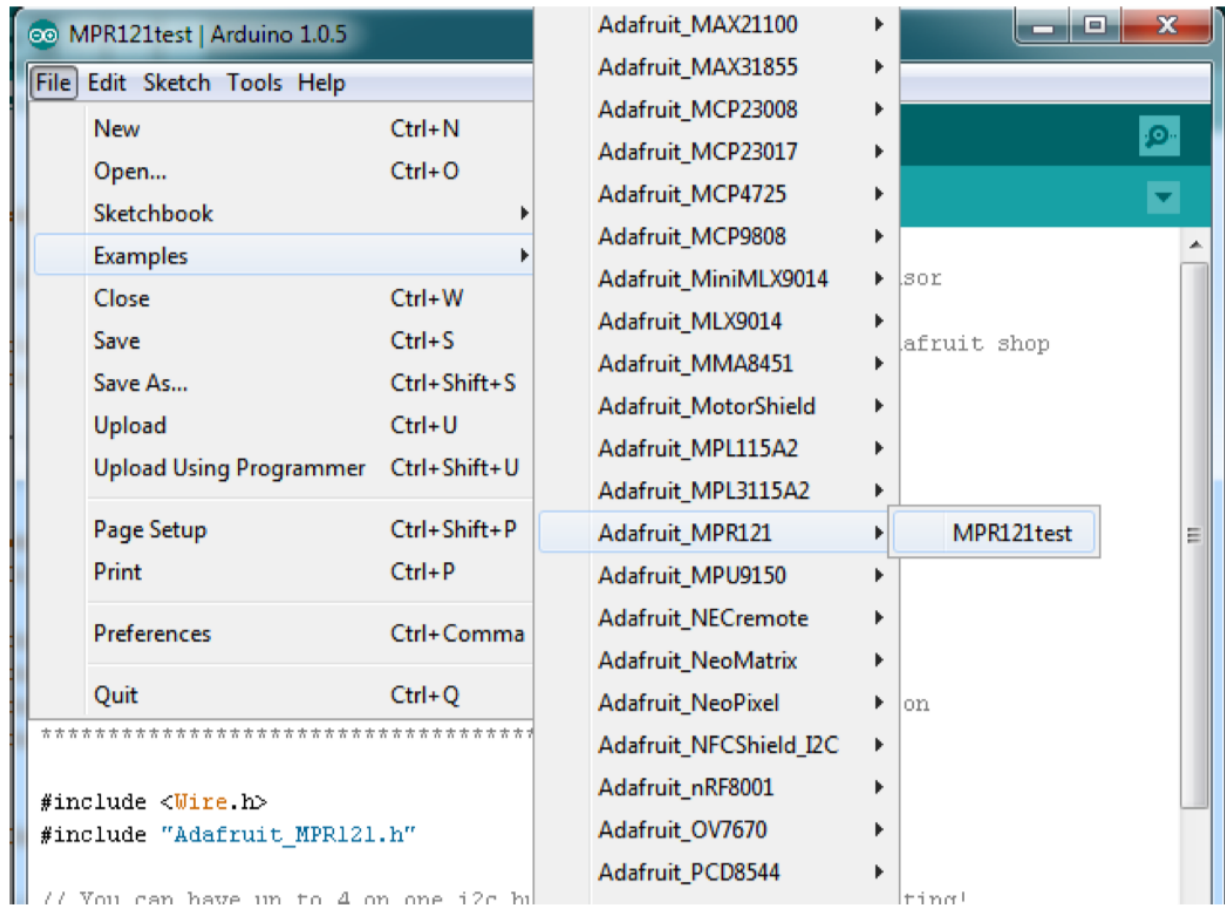
Rename the uncompressed folder **Adafruit_MPR121** and check that the **Adafruit_MPR121** folder contains **Adafruit_MPR121.cpp** and **Adafruit_MPR121.h**

Place the **Adafruit_MPR121** library folder your `arduinofolder/libraries/` folder. You may need to create the `libraries/` subfolder if its your first library. Restart the IDE.

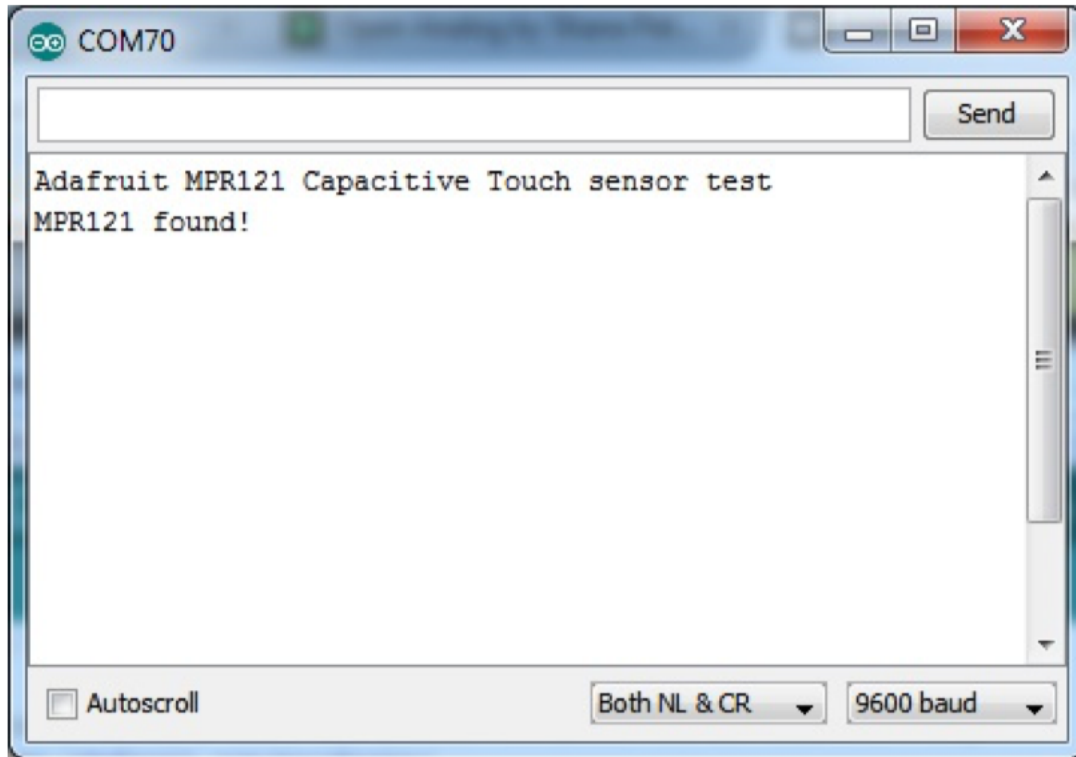
[There is a great tutorial on Arduino library installations.](#)

Load Demo

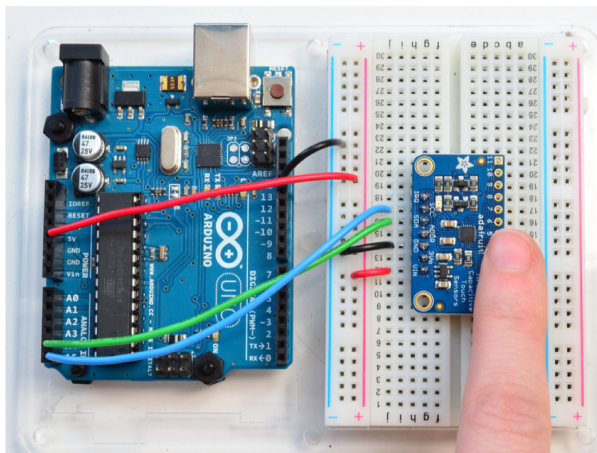
Open up **File** → **Examples** → **Adafruit_MPR121** → **MPR121test** and upload to your Arduino wired up to the sensor.

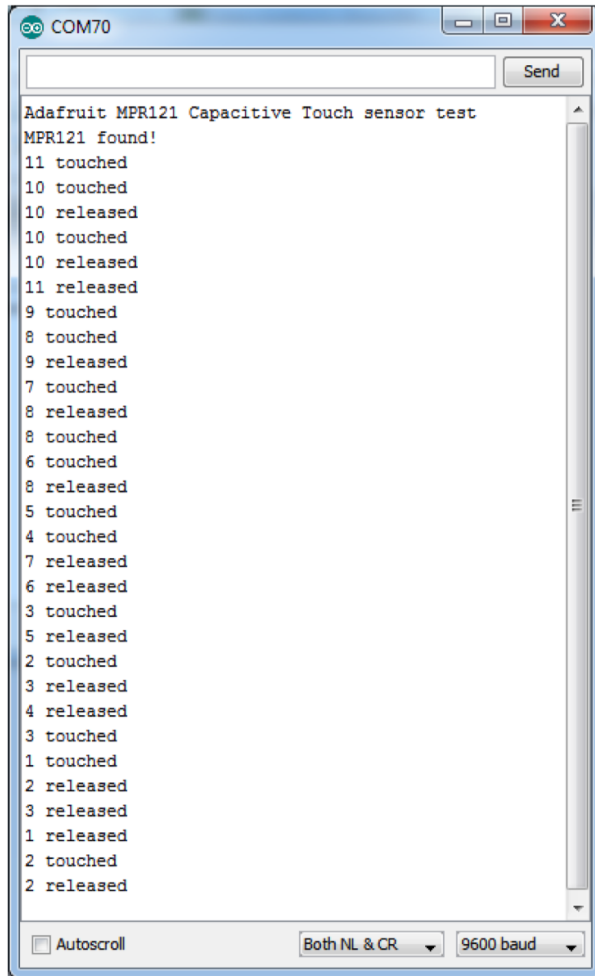


Thats it! Now open up the serial terminal window at 9600 speed to begin the test.



Make sure you see the “MPR121 found!” text which lets you know that the sensor is wired correctly. Now touch the 12 pads with your fingertip to activate the touch-detection:





For most people, that's all you'll need! Our code keeps track of the 12 'bits' for each touch and has logic to let you know when a context is touched or released.

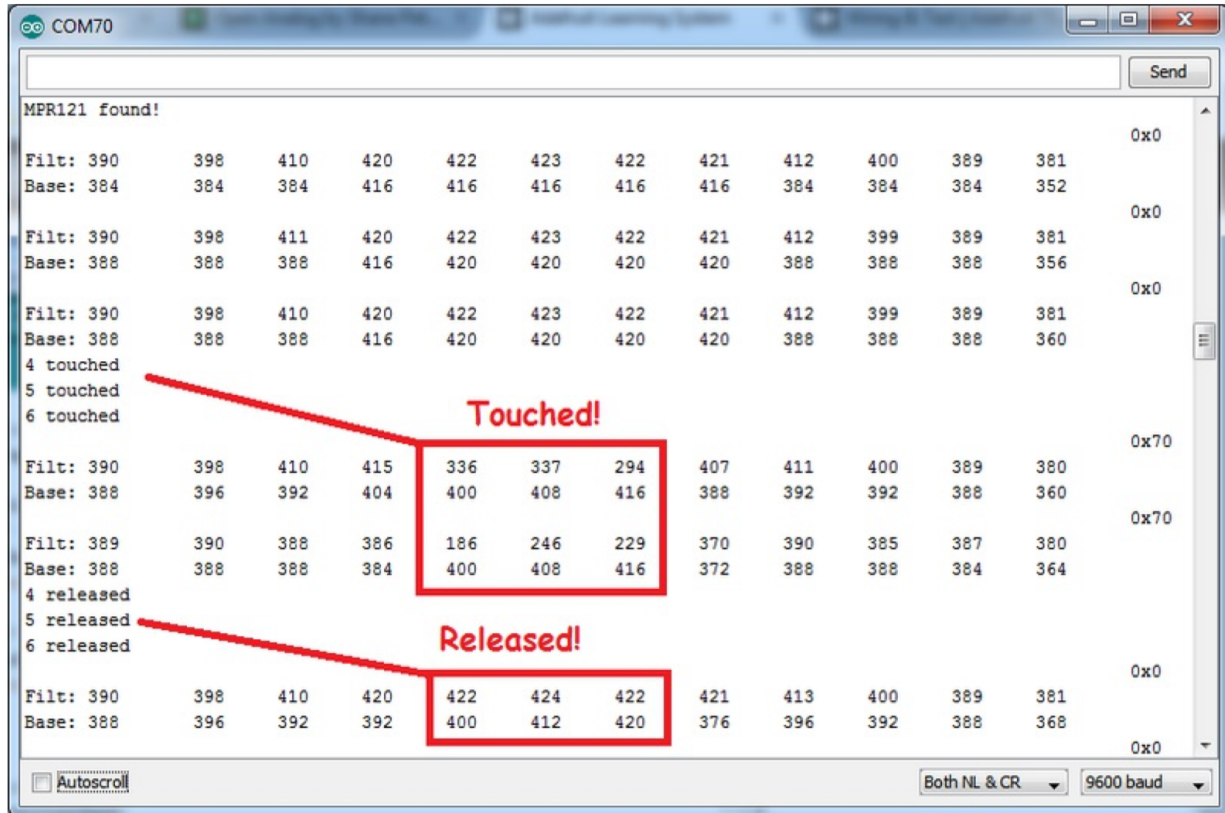
If you're feeling more advanced, you can see the 'raw' data from the chip. Basically, what it does it keep track of the capacitance it sees with "counts". There's some baseline count number that depends on the temperature, humidity, PCB, wire length etc. Where's a dramatic change in number, its considered that a person touched or released the wire.

Comment this "return" line to activate that mode:

```
// comment out this line for detailed data from the sensor! return;
```

Then reupload. Open up the serial console again - you'll see way more text.

Each reading has 12 columns. One for each sensor, #0 to #11. There's two rows, one for the 'baseline' and one for the current filtered data reading. When the current reading is within about 12 counts of the baseline, that's considered untouched. When the reading is more than 12 counts smaller than the baseline, the chip reports a touch.



Most people don't need raw data too much, but it can be handy if doing intense debugging. It can be helpful if you are tweaking your sensors to get good responsivity.

Library Reference

Since the sensors use I2C, there's no pins to be defined during instantiation. You can just use:

```
Adafruit_MPR121 cap = Adafruit_MPR121();
```

When you initialise the sensor, pass in the I2C address. It can range from 0x5A (default) to 0x5D

```
cap.begin(0x5A)
```

`begin()` returns true if the sensor was found on the I2C bus, and false if not.

Touch detection

99% of users will be perfectly happy just querying what sensors are currently touched. You can read all at once with `cap.touched()` which returns a 16 bit value. Each of the bottom 12 bits refers to one sensor. So if you want to test if the #4 is touched, you can use

```
if (cap.touched() && (1 << 4)) {
  // do something
}
```

You can check its not touched with:

```
if ( !(cap.touched() && (1 < 4)) ) {  
    // do something  
}
```

Raw Data

You can grab the current baseline and filtered data for each sensor with:

```
filteredData(sensorNumber);  
baselineData(sensorNumber);
```

It returns a 16-bit number which is the number of counts, there's no unit like "mg" or "capacitance". The baseline is initialized to the current ambient readings when the sensor `begin()` is called - you can always reinitialize by re-calling `begin()`! The baseline will drift a bit, that's normal! It is trying to compensate for humidity and other environmental changes.

If you need to change the thresholds for touch detection, you can do that with:

```
setThresholds(uint8_t touch, uint8_t release)
```

By default, the touch threshold is 12 counts, and the release is 6 counts. It's reset to these values whenever you call `begin()` by the way.

Challenge 3

Important: You must demonstrate your build & code to the tutor team

We challenge you to combine the touch sensor sketch with a simple analog output. Either a sound output using the `tone()` command to create a touch sensitive electric organ... or a RGB LED that changes colour when a touch pad is activated.

Acknowledgements

Adapted from these guides [1, 2]

2.2.6 Why Arduino?

The Raspberry Pi and Arduino are complementary platforms and one doesn't exclude the other. If you combine their capabilities you can achieve amazing results. But why would you use Arduino?

- **The community!** Arduino has a lot of materials readily available online, from libraries, to examples. If you have something in mind probably someone has already done it and shared the documentation.
- **Uses very little power** and boots very quickly.
- **Runs at 5V logic level** whereas the Pi digital pins operate at 3.3V.
- **Incredibly cheap hardware** which is useful for powering/controlling prototype electronics which might end up damaging your controller. You have to be more careful with a Pi as they are more easily damaged and more expensive to replace!

- **Real-time capabilities** are more readily accessible whereas there are limitations and constraints to getting the same performance from the Linux-based kernel in Raspbian.

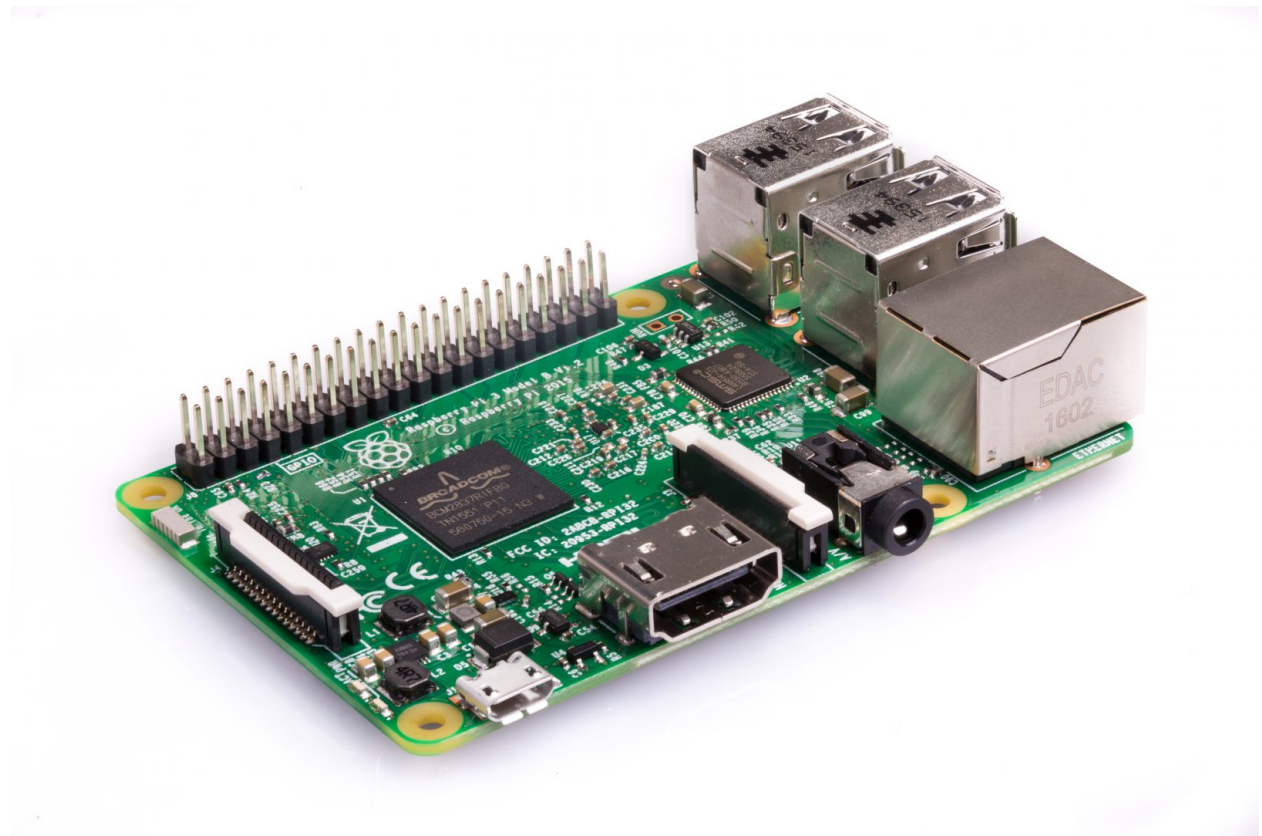
2.2.7 Alternative Microcontrollers

- ARM mBed
- STM32
- Teensy
- pyBoard runs microPython

Note that microPython is an incredibly useful alternative to the Arduino community and there are a number of new and highly functional boards built specially for microPython.

2.3 Raspberry Pi

Welcome to the Raspberry Pi guide. In this you will find information and tutorials on all aspects of setting up and using your Pi. See below for *Why Raspberry Pi?* and possible *Alternatives to the Raspberry Pi*.



2.3.1 Assembling Pi workstation

The goal of this section is to set up the physical components to the Raspberry Pi. Later on we will look at how we might run a Pi ‘headless’. This means we can control the Pi remotely without the need for a screen, keyboard, mouse, or any other peripherals directly connected to the Pi itself.

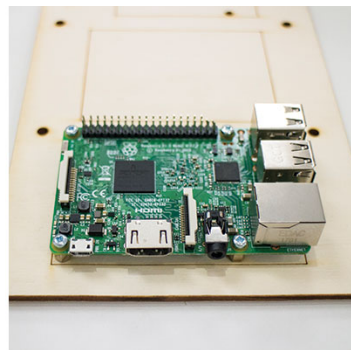
To do this, we will be using [SSH](#) protocol. SSH will remotely connect you to a Pi over a local or global network. For this, you need to complete the Networks section before you can move onto the Connecting Remotely section.

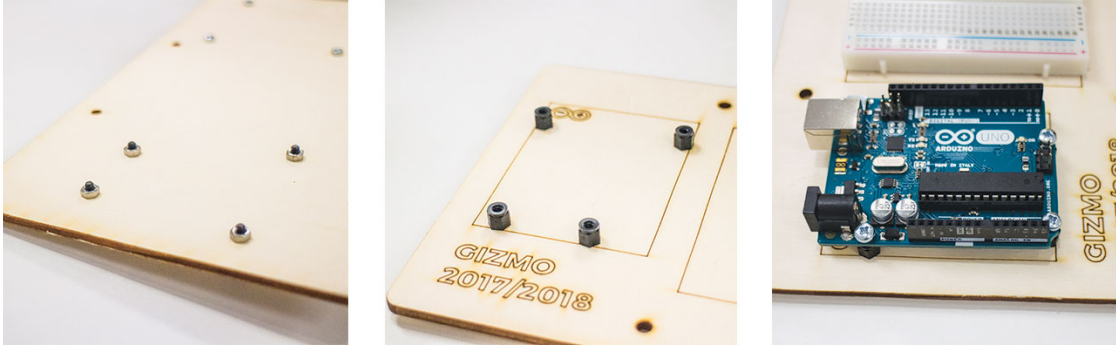
When using the Pi in this advanced format, we will be using the ‘Terminal’ a lot. If you are new to the terminal can be a bit overwhelming at first, don’t panic and just follow the steps carefully! There are many great resources on the internet to help you understand how to use a terminal, including this website.

Getting going

At first we will setup the Pi using peripherals. Each team should get the following equipment:

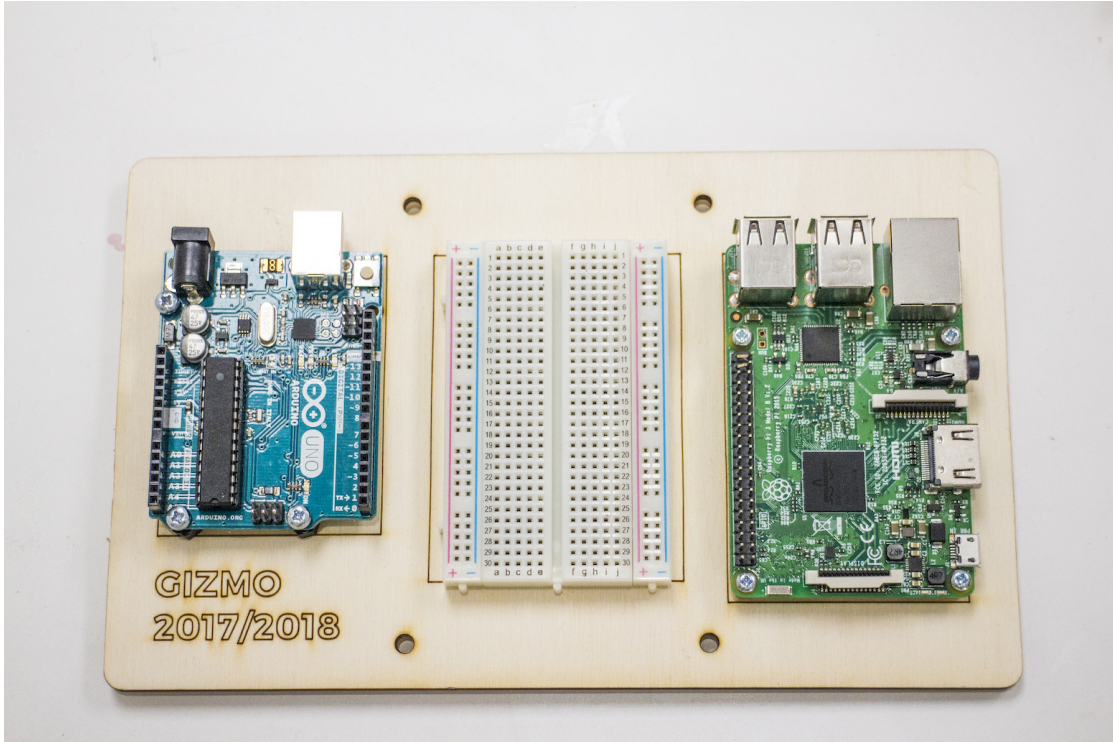
- 1 Touchscreen
- 1 Raspberry to touchscreen HDMI-HDMI plug
- 1 Touchscreen pen
- 1 Raspberry Pi Power Supply
- 1 Raspberry Pi
- 1 Keyboard
- 1 SD Card
- 1 Wooden plate
- 8 M2.5 Bolts
- 4 M2.5 Standoffs
- 3 M3 Bolts
- 4 M3 Spacers
- 4 M3 Nuts
- 1 Breadboard
- 1 Arduino
- 1 USB A to USB B Cable
- 1 Screwdriver
- 1 Pair of Pliers



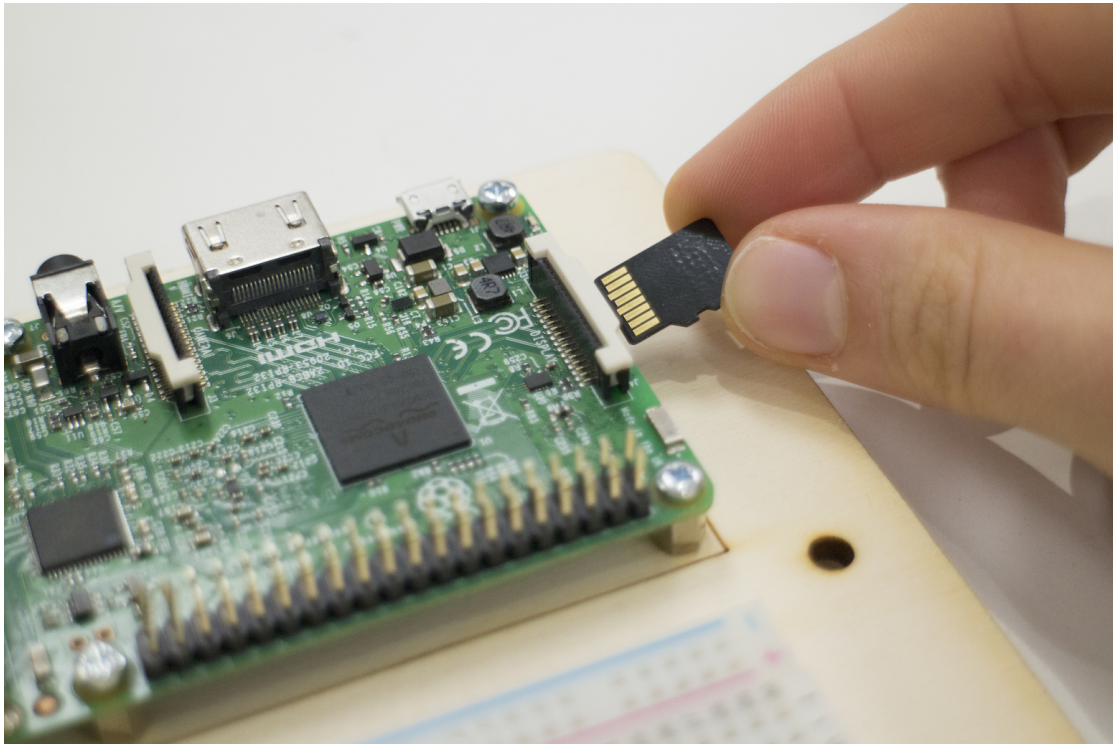


3. To attach the breadboard to the wooden plate, peel off the back of it to expose the adhesive strip and glue it to the wooden plate:

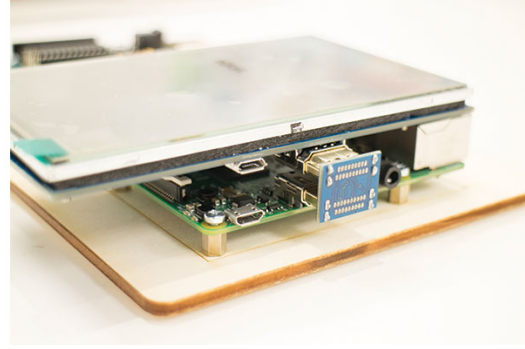
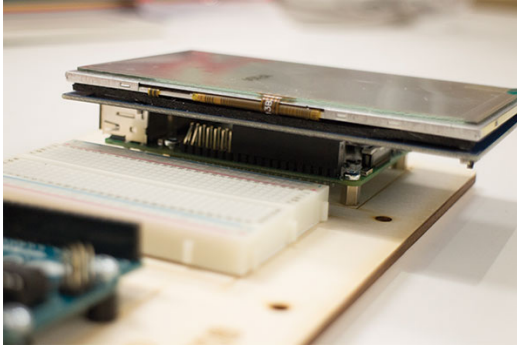




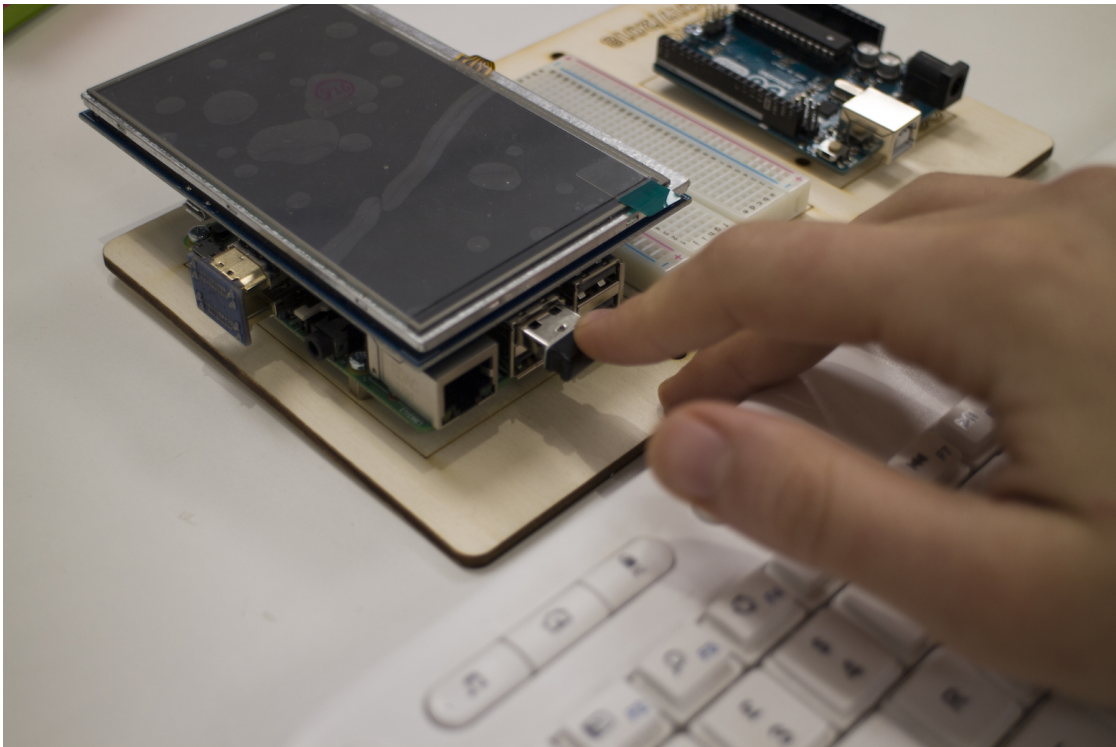
4. Insert the micro-SD card in the back of the Pi, like so:



5. Connect the touchscreen to the Pi, connecting it to the pins and with the HDMI plug, like so:



6. Connect the keyboard with the USB.



7. Using the power cable, power up the Pi and the screen:



8. The Pi will start the setup, if the screen doesn't illuminate check that it is on.

Note: Go to the next section to find out more about the SD card we have provided.

2.3.2 Setting up your SD Card

When setting up a new Raspberry Pi, you will need to install an operating system (OS) onto your SD card. There are many operating system options, however the official and most commonly used is [Raspbian OS](#). For Gizmo you will set up a modified version of Raspbian.

When setting up a new SD card yourself, we recommend using NOOBS (New Out Of Box Software). It is an easy installer to get a Pi up and running quickly (including Raspbian OS). For information on how to set up with NOOBS, use [this guide](#).

Downloading your Disk Image

Usually, you would download a zipped image file directly from the [Raspberry Pi website](#).

However, for Gizmo we have already added the drivers required to use the touchscreen. We followed [this guide](#) from the manufacturer. You can use it to find additional steps to modify the configuration of the touchscreen if you need to.

Flashing your Disk Image

1. First you must format (wipe) the SD Card. [Download SDFormatter here](#) .
2. Use SDFormatter to format the SD card. Please be careful and make sure you select the correct drive letter.
3. [Download Etcher](#) if not installed.

4. Use Etcher to flash the image to the SD card.
 - Please be careful that the correct drive letter is selected.
 - On Mac: if you backed up the image yourself (see Chapter 8: *Backing up your SD card*) you may have to change the file extension from “.cdr” to “.iso”
5. You're done! If all has gone well the Raspberry Pi should now boot when started with the new SD Card.

Tip: Your Gizmo Raspian OS has been optimised for the 5 inch touchscreen - so will always display the screen in low resolution (800x480). If you move to a full size screen you should reverse the steps taken [here](#) to avoid potato resolution.

2.3.3 Configuring the Pi

raspi-config

- *Keyboard*
- *Timezone*
- *User password*
- *Enable SSH*
- *Expand root partition*
- *Reboot*

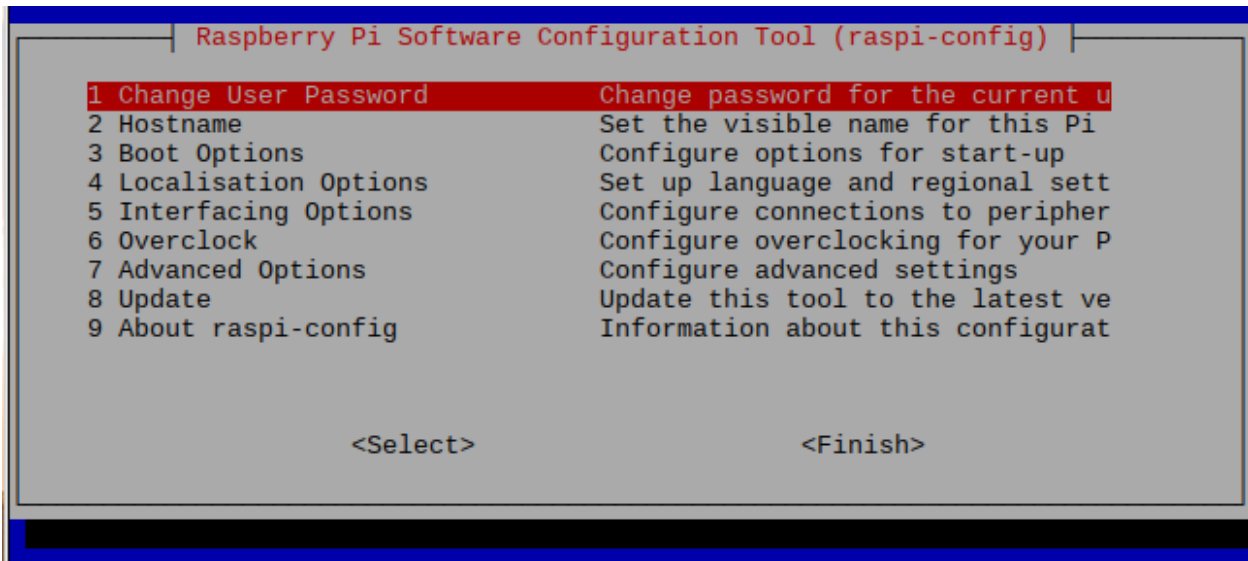
A lot of the Pi's system settings are configured in `raspi-config`, a [terminal/shell](#) based tool. When we run this tool, we will run them as a **root user**, the root has the permission to modify files or default settings as an administrator. By default on Raspbian (the operating system of our Pi) the **root user** is `pi` and the **root password** associated to the root user is `raspberrypi`. To operate as a root user in the terminal every command is preceded by the `sudo` (super user do) command.

Type the following command and press 'Enter' to open the configuration menu of the Pi:

```
$ sudo raspi-config
```

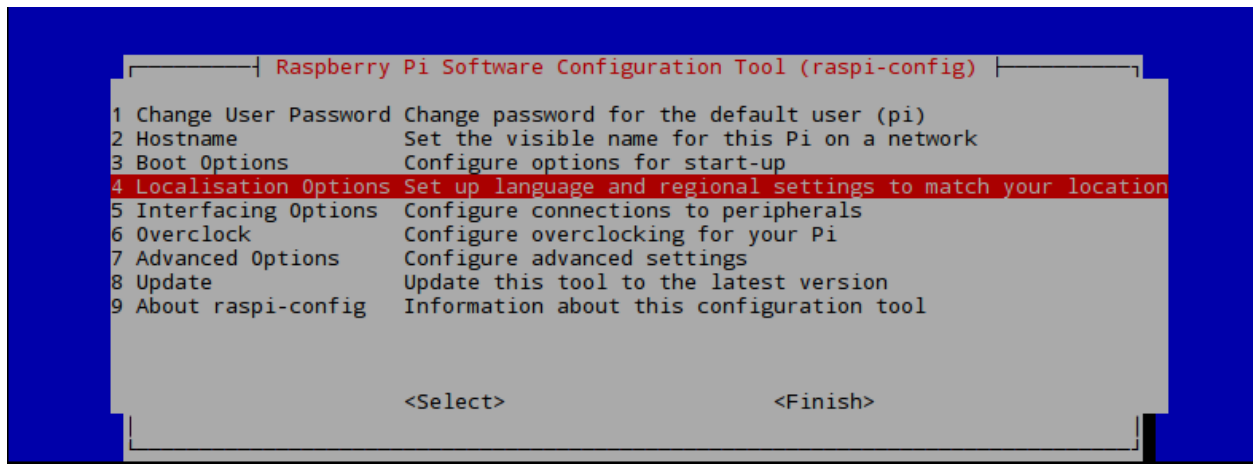
Note: The \$ symbol in front of the command signifies when this command should be written in the terminal window. When writing the command you should not write the dollar sign, only the command: `sudo raspi-config`.

The terminal will show a menu. The options can be navigated with the vertical keys of your keyboard, to accept the options press 'Enter', to finish press the lateral keys of the keyboard.

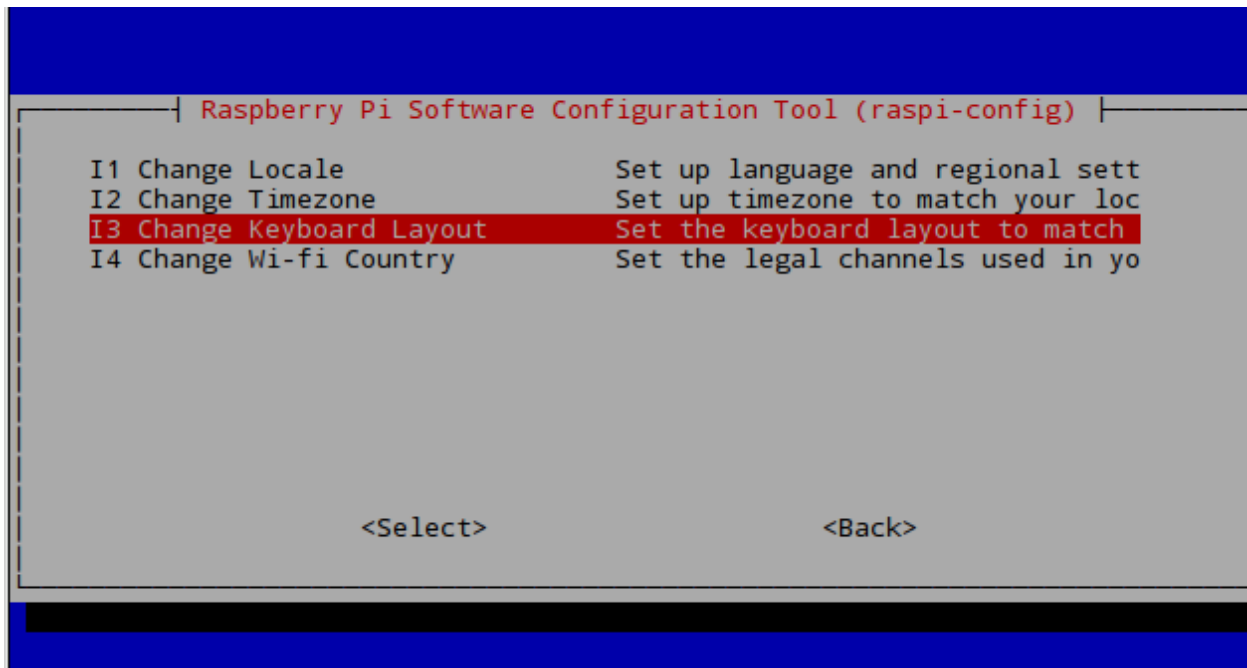


Keyboard

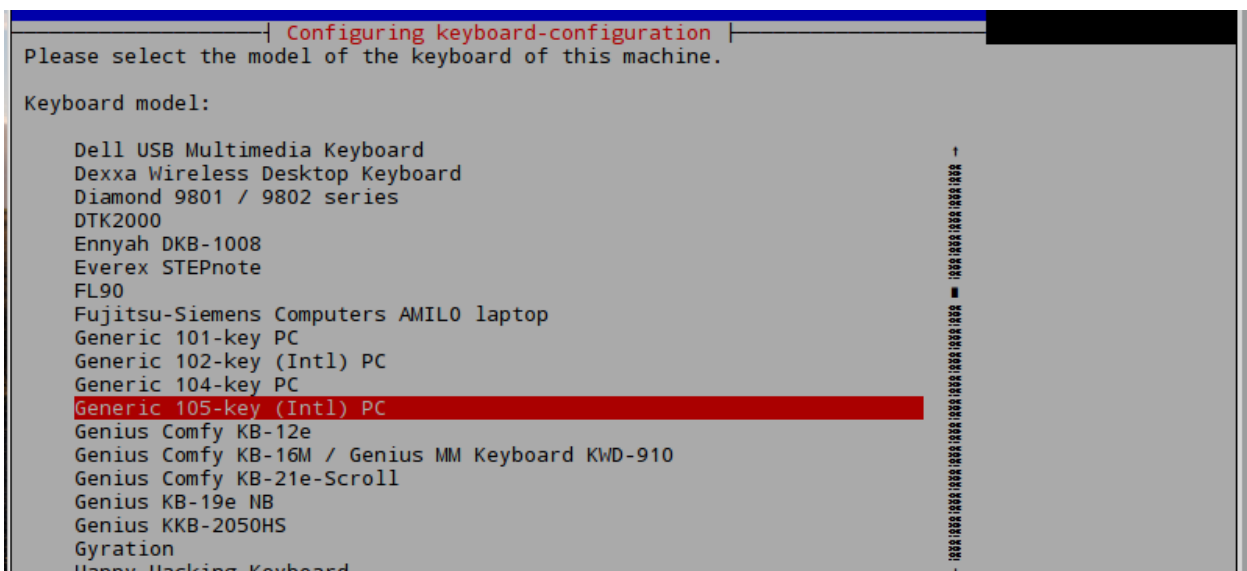
First we set up the keyboard to prevent any problem when we will change the root password. We access the option: 4 *Localisation Options*:



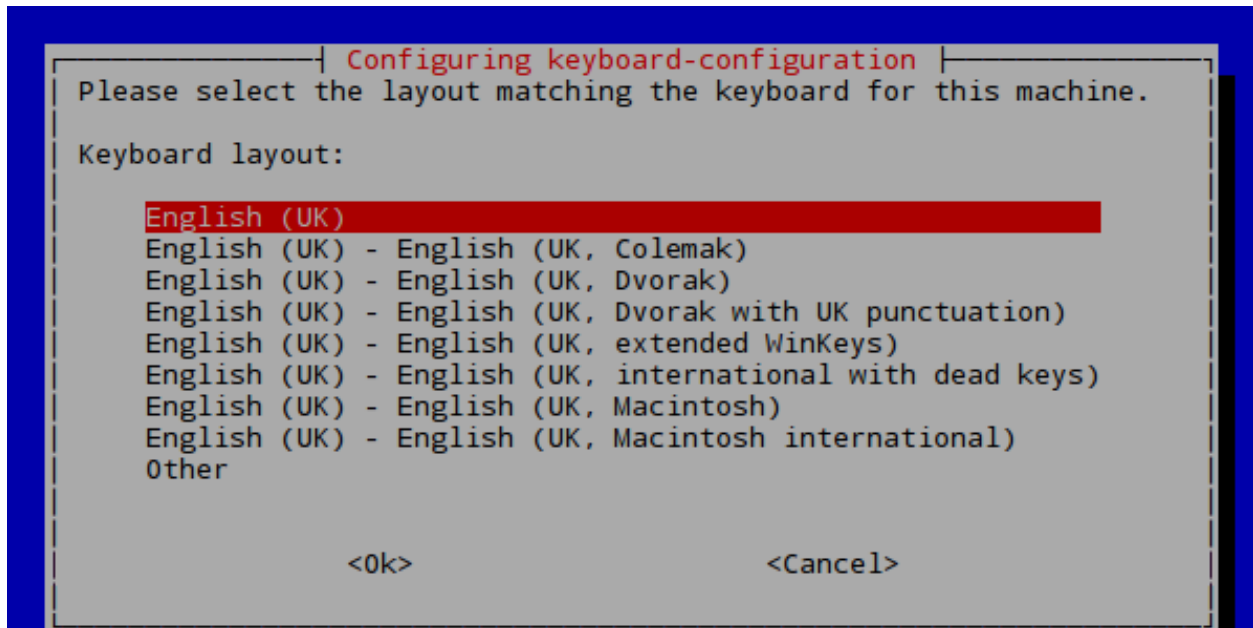
Then *Change Keyboard Layout*:



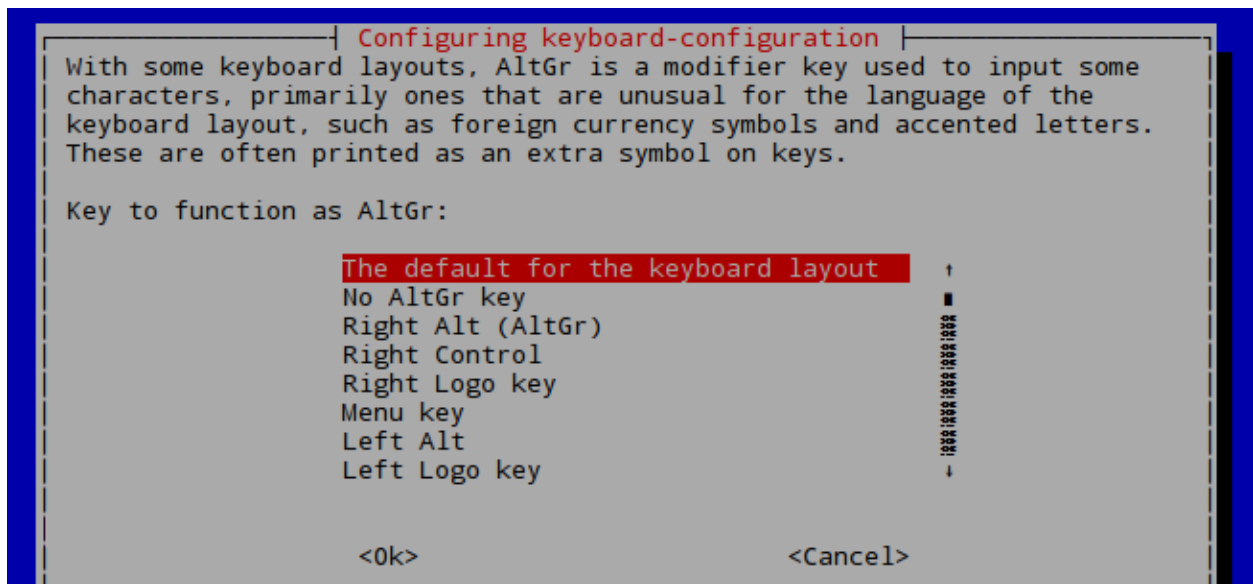
Then we choose *Generic 105 key*:

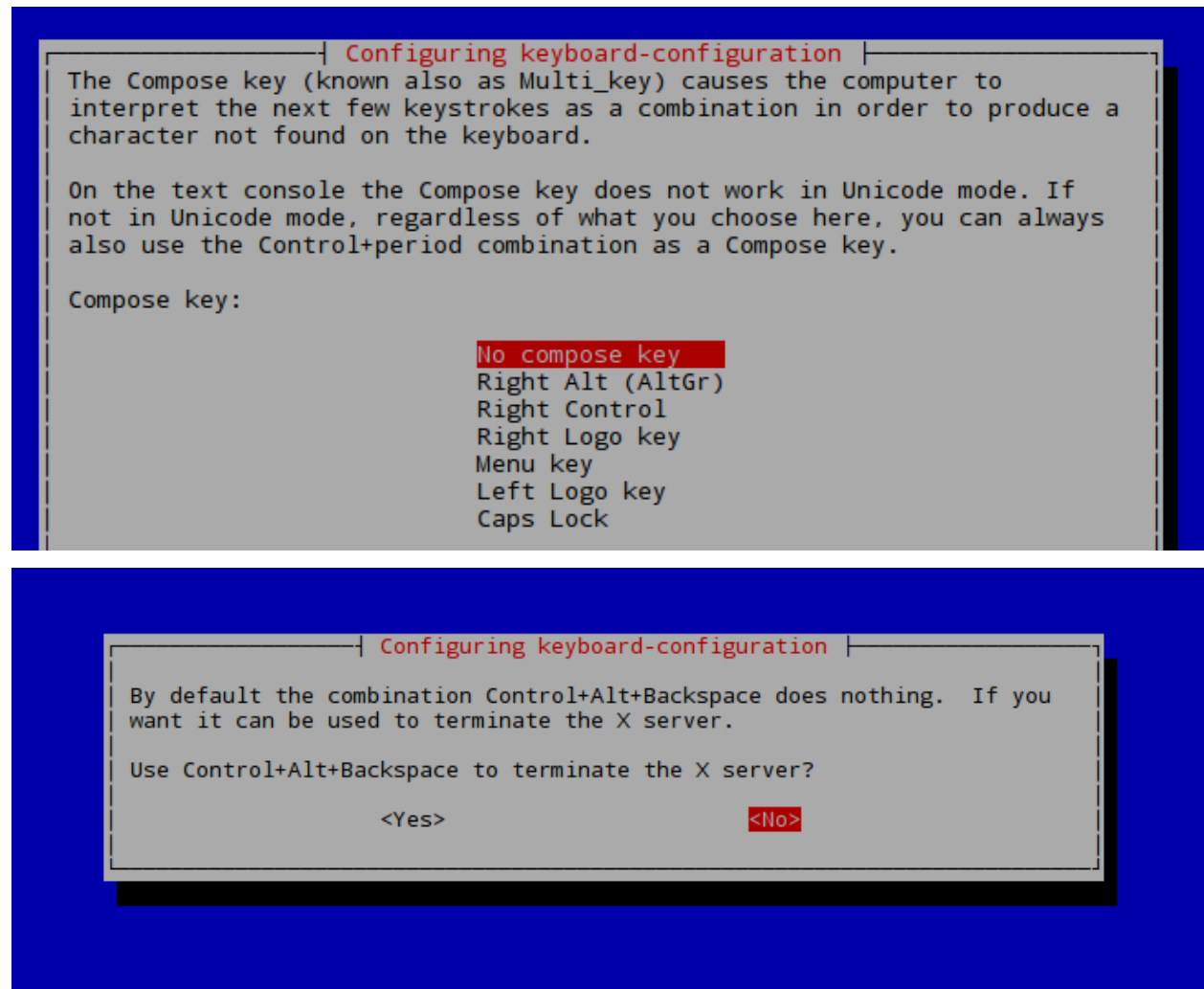


And then *English (UK)*:



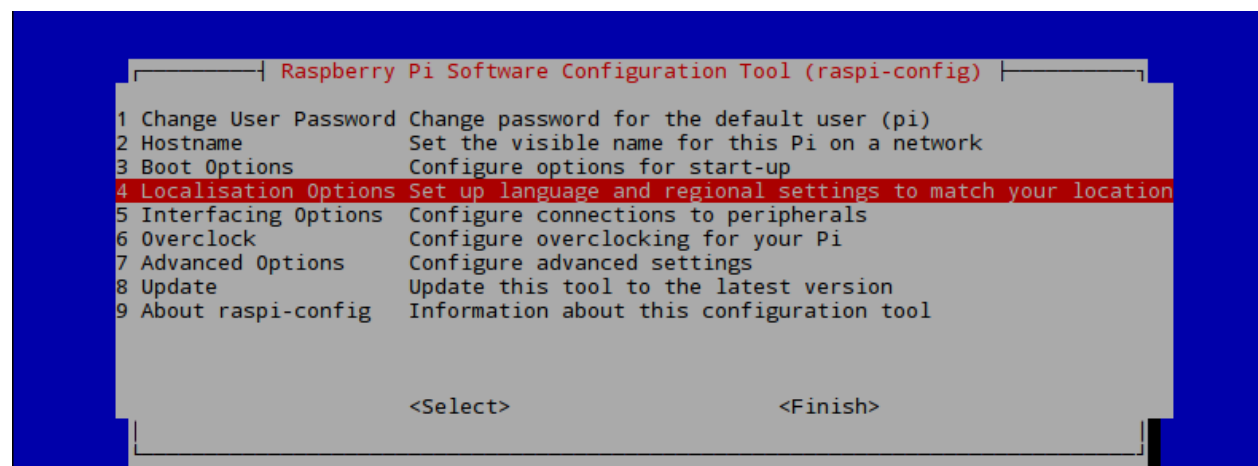
Then we can choose the default options that the menu is prompting by pressing enter:



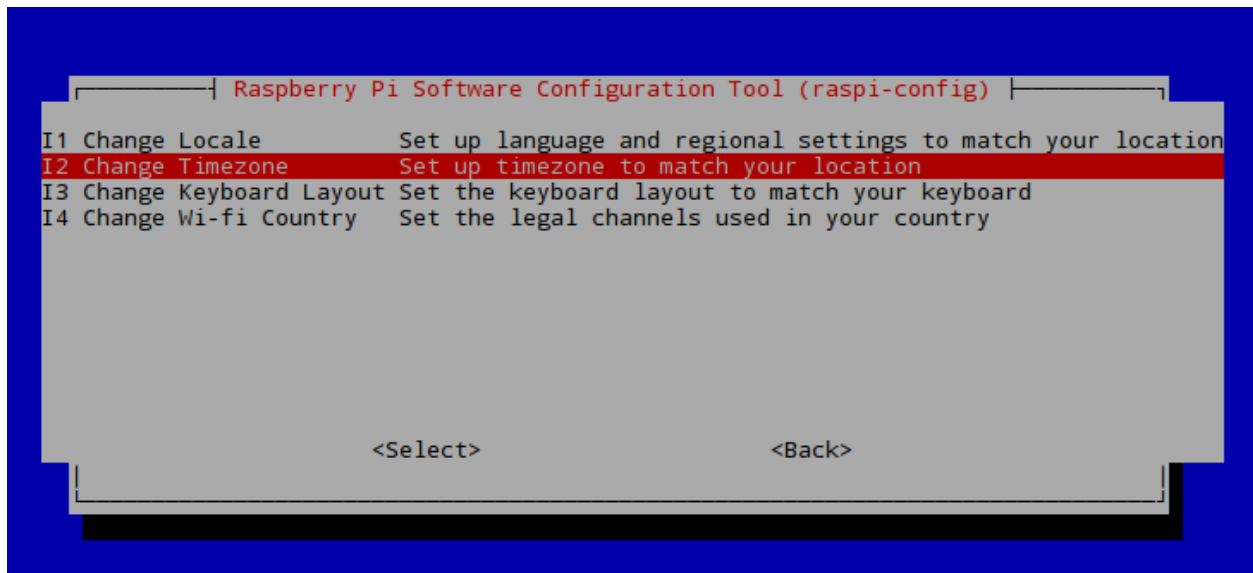


Timezone

Then we are re-directed to the main menu, now we change the timezone from the *4 Localisation Options* menu.



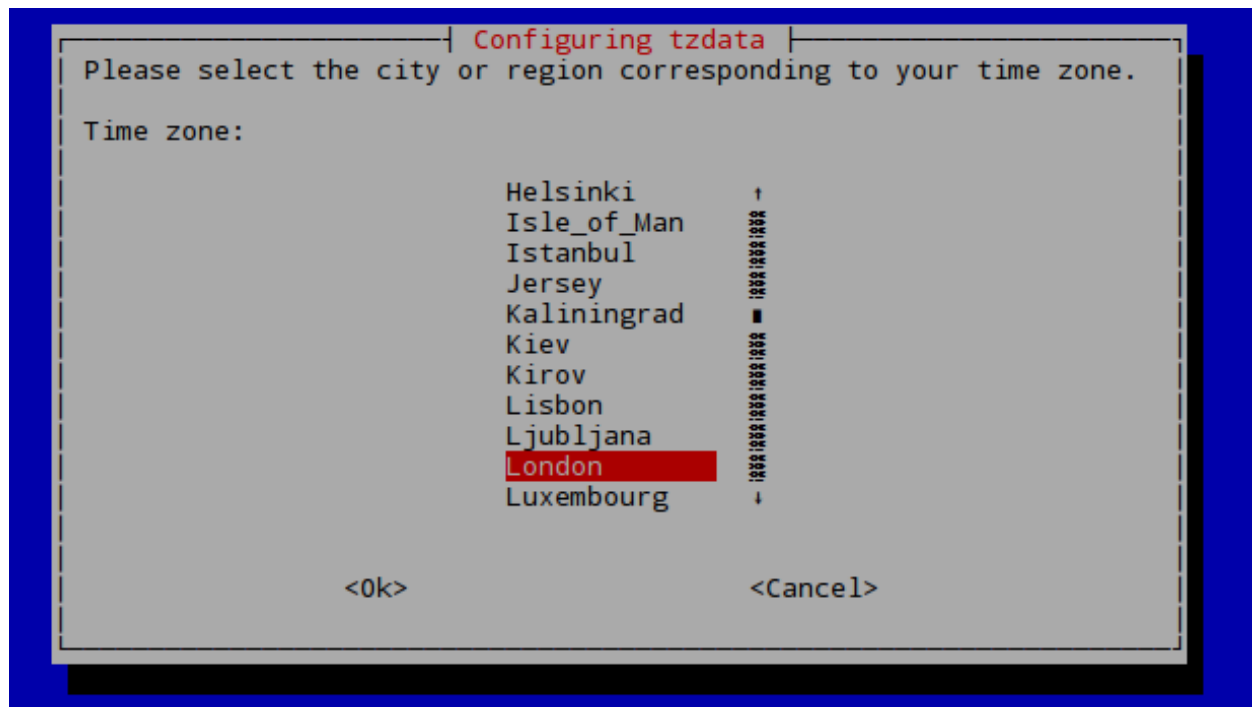
Then we choose *Change Timezone*:



Then *Europe*:

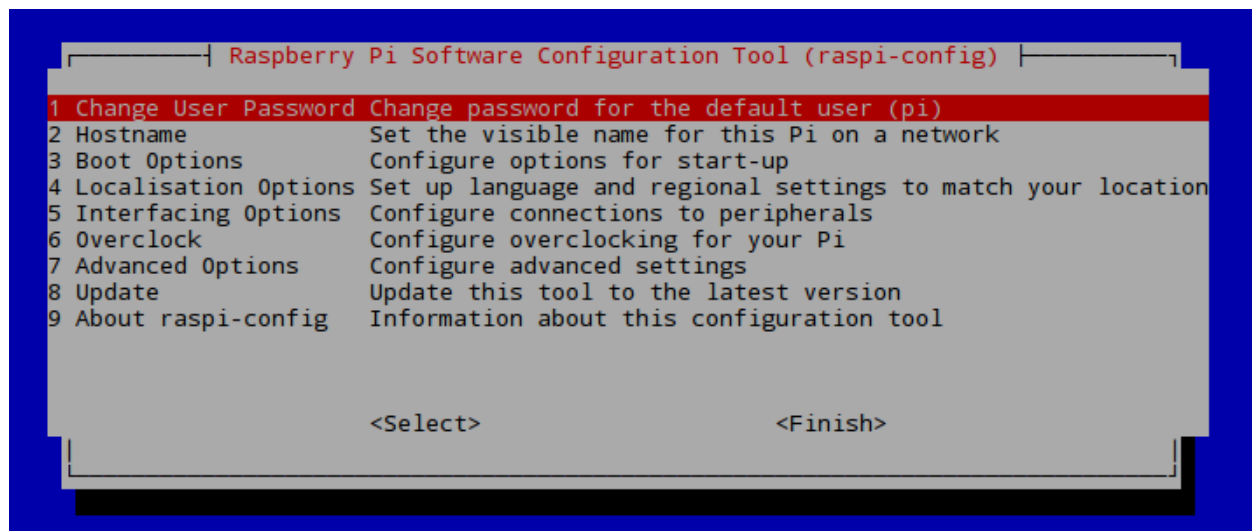


Then *London*:

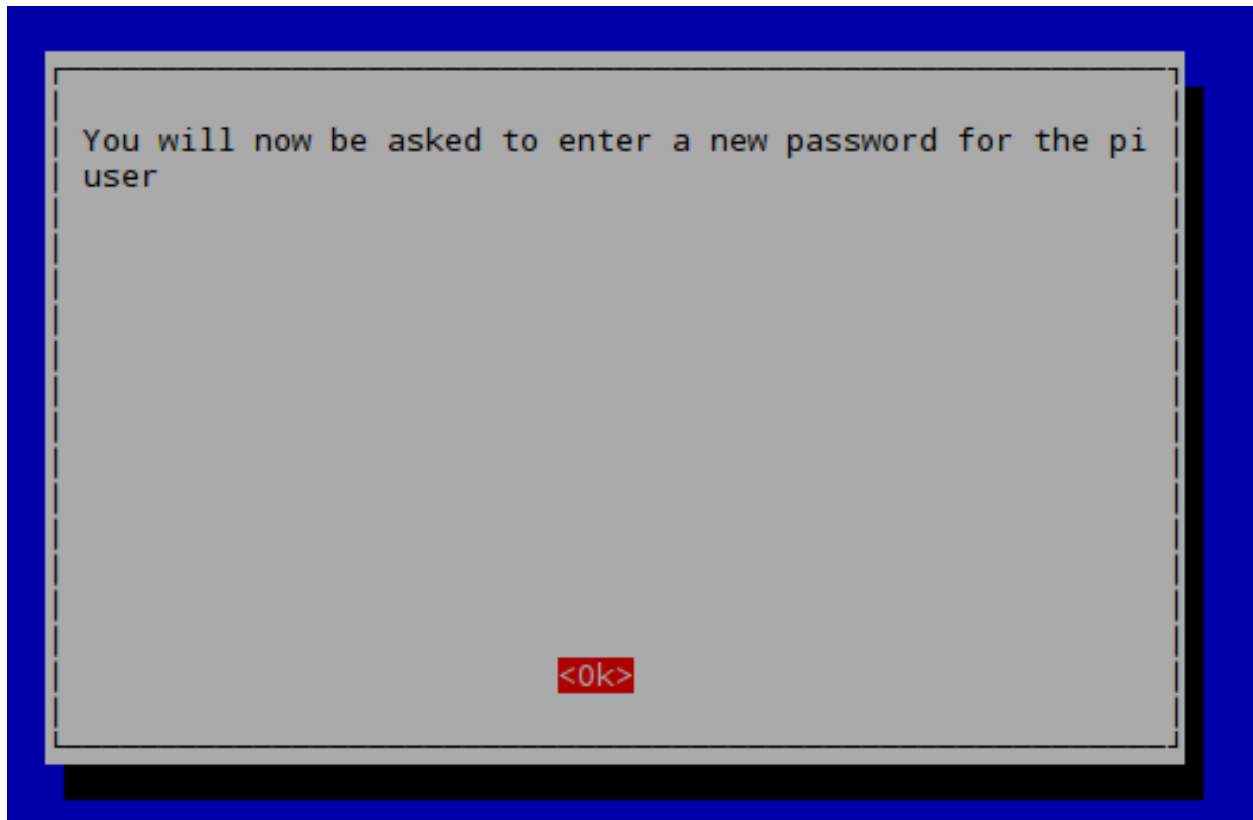


User password

Now we will change the root user password. This increases the security of the connection we will establish from our laptop to the Pi. Since you are sharing this Pi with your colleagues choose a password together. To change the password we are re-directed to the main menu and here we choose the first option: *1 Change User Password*:



Then we agree to change the password:



Type the new password twice:

```
pi@raspberrypi:~ $ sudo raspi-config
^[[Aupdate-rc.d: warning: start and stop actions are no longer supported; falling back to defaults
update-rc.d: warning: start and stop actions are no longer supported; falling back to defaults
Reloading keymap. This may take a short while

Current default time zone: 'Europe/London'
Local time is now:      Fri Oct  6 14:13:45 BST 2017.
Universal Time is now:  Fri Oct  6 13:13:45 UTC 2017.

Enter new UNIX password:
Retype new UNIX password: █
```

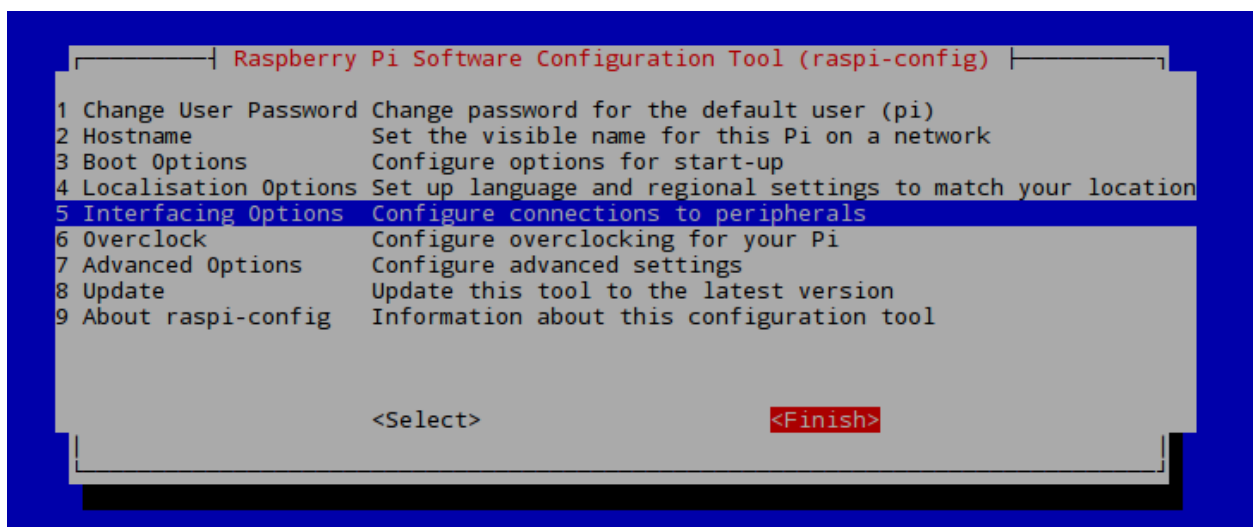
Note: Nothing will appear on screen when you are typing the password. This is normal - it's still working! If you need to cancel, press `Ctrl+C` on the keyboard.



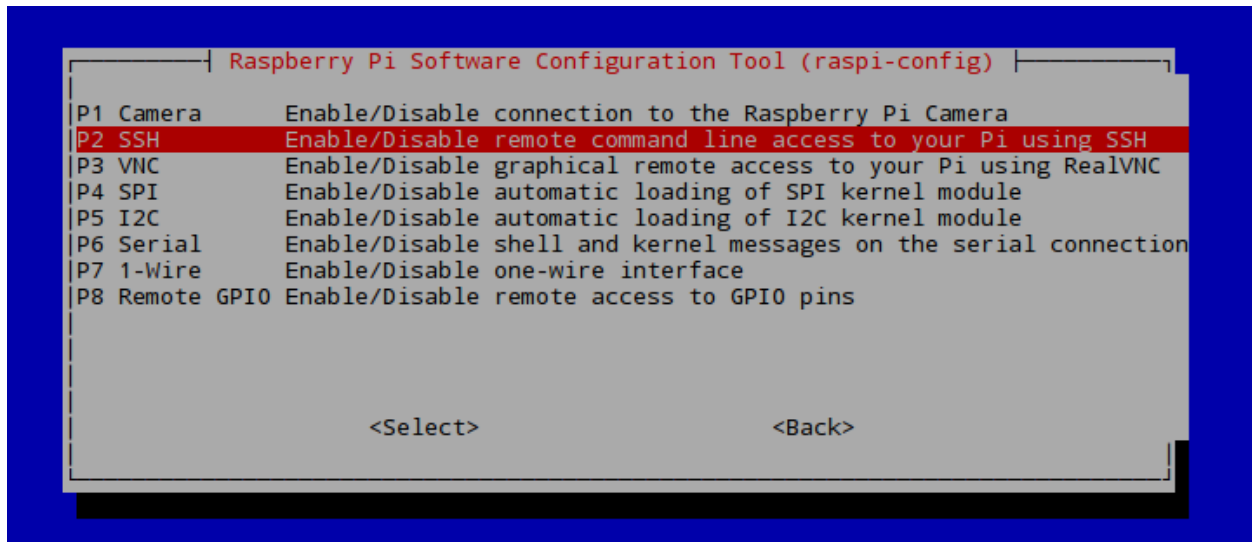
We have set the new password. Do not reboot the Pi yet.

Enable SSH

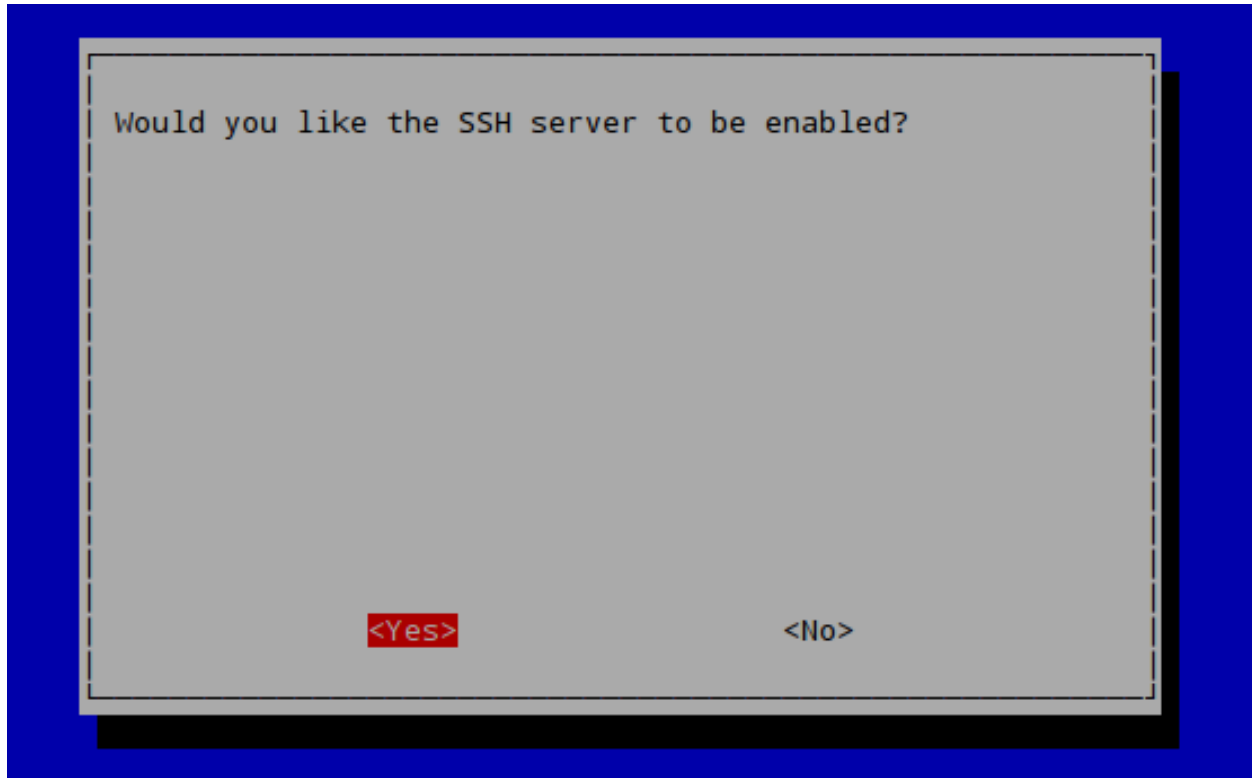
We will now check that the SSH is enabled. We need to enable it to connect with the Pi remotely. From the main menu we access: *5 Interfacing Options*:



Then we select *SSH*:



Then we confirm that we want to enable the SSH server:



We confirm again:

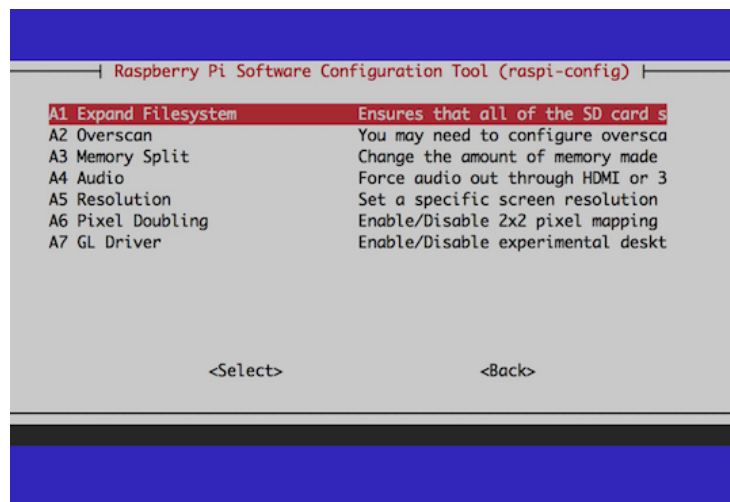


Exit the menu by pressing the right arrow twice to select *Finish* and press the Enter key. You will re-enter the terminal.

Expand root partition

Lastly we will expand the root partition to fill the SD card. From the main menu we access: *7 Advanced Options*:

We select *1 Expand Filesystem*.



We confirm the changes. You will need to reboot the Pi for changes to be implemented (see next step).



Note: SD Cards can be split into partitions to separate storage of data. The partition has been shrunk to make the download size smaller, but we now want to expand the root partition to fill the whole SD card (allowing more space for file storage). If you do not do this you will find you are unable to create new files.

Reboot

Now reboot the Pi to ensure all your changes are made:

```
$ sudo reboot now
```

Tip: Following the new kernel update “Raspbian Stretch” released in September 2017, some users have found that their settings are reset after reboot. If so, please perform Pi Configuration within the X-Environment:

1. Click on Raspbian Icon
 2. Preference
 3. Raspberry Pi Settings
-

Adding users

A guide on adding new users to the Pi can be found [here](#). Generally this is not necessary, and you can continue to use the `pi` account. Just remember to change the user password for `pi` from `raspberry` to something new!

You can create additional users on your Raspbian installation with the `adduser` command. Enter `sudo adduser bob` and you will be prompted for a password for the new user `bob`. Leave this blank if you do not want a password. However, we recommend that each user get a password to access remotely in the future, for example:

```
ssh bob@123.343.1.105
```

You can delete a user on your system with the command `userdel`. Apply the `-r` flag to remove their home folder too:

```
sudo userdel -r bob
```

The default `pi` user on Raspbian is a sudoer. This gives the ability to run commands as root when preceded by `sudo`, and to switch to the root user with `sudo su`.

2.3.4 Headless Setup

Important: You **do not** need to do this setup if you are using a screen and keyboard. If you have a screen and keyboard available, move onto the next chapter.

It can be useful to set up a new Pi even when you *don't have a monitor and keyboard* spare to do a normal set up. Fortunately, there is a way to set up a new Pi without using these peripherals. This is called 'setting the Pi up headless'. The following guide is derived from the [official Pi documentation](#).

Note: This method is slightly more advanced, so if you are a little unsure of how to do it, ask someone to help you with it.

1. Download the latest Raspbian disk image [from the Pi website](#).
2. Format the SD card using [SDFormatter](#).
3. Use [Etcher](#) to flash the Raspbian image (.img) to the SD card.
4. Open the `boot` folder on the SD card.
5. Copy the example `wpa_supplicant.conf` file (see [Accessing Networks](#)) into the `boot` folder.
6. Copy the `ssh` file into the `boot` folder.
7. Insert the SD into the Pi and power on.

2.3.5 Accessing Networks

WiFi via GUI

When running the Pi in desktop mode, you can join new Wifi Networks in a similar way to how you would do it on a Macintosh. In the menu bar at the top, on the right-hand side click on the wireless icon. Then from there you can select from the list of discovered networks to join them.

Note: If they have a padlock next to them then they require a password to join.

WPA Supplicant

- *Backup*
- *Edit*
- *Encrypting Your Password*

Since the College has a more complex form of authentication (username *and* password required). We will setup the Pi to connect to the IC network a slightly different way. We are going to modify a configuration file called `wpa_supplicant.conf`.

Backup

First we back up the configuration file `wpa_supplicant.conf`. We create the backup file `wpa_supplicant.conf_backup` in case we need to restore it later. *It's important that you don't edit this backup after creating it.* To do so we enter the command:

```
$ sudo cp /etc/wpa_supplicant/wpa_supplicant.conf /etc/wpa_supplicant/wpa_supplicant.
↪conf_backup
```

Edit

Then we edit the `wpa_supplicant.conf`. The default text editor installed in the Pi is *nano*. To edit a file with the nano editor is sufficient to enter the command `nano /path/to/file`. Therefore to edit `wpa_supplicant.conf` we enter the following command with admin user permission:

```
$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

We edit the file so that the all the content appears like this:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
    ssid="Imperial-WPA"
    proto=RSN
    key_mgmt=WPA-EAP
    pairwise=CCMP
    auth_alg=OPEN
    eap=PEAP
    identity="ic\COLLEGE_USERNAME"
    password="YOUR_PASSWORD"
    priority=7
}
```

Where `COLLEGE_USERNAME` has to be replaced with your college username and `YOUR_PASSWORD` with the password associated to it.

Important: The configuration is case sensitive, so make sure you do not have typos. **Even the slightest error in this file can cause the networking to fail** so make sure it exactly like this.

Note: If you want to connect your Pi to the **eduroam** network, then set `identity="COLLEGE_USERNAME@ic.ac.uk"`. Apply the same procedure for setting the password as seen below.

In the nano editor, to exit, press `Ctrl + x`. The editor will then present you with different options such as save the file or exit without modifying the file: `y/n`. We press `y` and then press `enter`. The editor now asks us for the name of the file we are saving, but as it already fills out the previous name for us, we press `enter` again.

Now we can check if the connection works by rebooting your RPi. Reboot it by entering:

```
$ sudo reboot now
```

Once the system starts again the Pi should connect automatically to the WiFi.

Encrypting Your Password

1. In order not to store the password in a plain text we substitute our password with an **encrypted** one using a **MD4 hash generator**. You can generate the hash with the following Linux command:

```
$ echo -n 'YOUR_PASSWORD' | iconv -t utf16le | openssl md4
```

You will have to substitute `YOUR_PASSWORD` with the password related to the account in the `wpa_supplicant.conf`. This will be the only time you'll have to type it in plain text. Ask your colleagues to look away from the screen if you are not comfortable in them seeing your password.

2. The previous command will display the encrypted password on your terminal like this:

```
$ (stdin)= a6c71eedc2eacbca84003336a4a62a1c
```

We **copy the string** that was generated in your terminal screen (i.e. `'a6c71eedc2eacbca84003336a4a62a1c'`).

Tip: You can save the hash from your password in a file and then read its content:

```
$ echo -n 'YOUR_PASSWORD' | iconv -t utf16le | openssl md4 > hash.txt
$ cat hash.txt
```

The first command creates the encrypted password and stores it in the `__hash.txt__` file. The second command reads the content of the `__hash.txt__` file. In general we use the `cat` command to read and concatenate files.

3. Then we open the `wpa_supplicant.conf` file again:

```
$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

4. In the password field replace `"YOUR_PASSWORD"` with the string you generated as hexadecimal characters, adding the `'hash:'` prefix as shown in the example bellow:

```
network={
    ssid="Imperial-WPA"
    proto=RSN
    key_mgmt=WPA-EAP
    pairwise=CCMP
    auth_alg=OPEN
```

(continues on next page)

(continued from previous page)

```
eap=PEAP
identity="ic\COLLEGE_USERNAME"
password=hash:a6c71eedc2eacbca84003336a4a62a1c
}
```

5. The last security step to perform is to remove the bash history (the one that stores all the commands we had typed on the terminal). Therefore, we enter the following commands:

```
$ history -w
$ history -c
```

6. Then we reboot the Pi to check that the password was properly set up:

```
$ sudo reboot now
```

7. And you are done!

Pi as a hotspot

The Raspberry Pi can act as a standalone network. This can be useful in some situations where you do not want to rely on a separate wireless network, or when you might be going to a new location that cannot provide you with a network to use. Remember though that a standalone network made by the Pi will not be connected to the internet. You can find the [guide to set up a standalone network here](#).

2.3.6 Software

Now that you have your Pi connected to the internet, you should make sure it is completely up-to-date.

Removing packages

When you install a standard build of Raspbian, a lot of packages get installed for you. To make sure some of the larger packages aren't wasting space on your SD card, you can remove them. We recommend you remove some of these. This can be done by running each of the following lines one-by-one on the command line.

```
sudo apt-get purge libreoffice wolfram-engine sonic-pi scratch
sudo apt-get clean
sudo apt-get autoremove
```

You can reinstall any of these again later if you need them.

Operating System

Run the following line in the terminal to update your Pi. Note: this might take some time.

```
sudo apt-get update && sudo apt-get upgrade
```

Installing packages

Now the OS is updated, we need to install Python. To install Linux packages onto our Pi we use the command: `sudo apt-get install <name_of_package>` in the terminal. Each installation could take a few minutes.

1. Run each of the following two lines to install C lib, needed by Python

```
sudo apt-get -y install libffi-dev
sudo apt-get -y install libssl-dev
```

2. Installing Python, run each line one-by-one ensuring they complete

```
sudo apt-get -y install build-essential python-dev python-openssl
sudo apt-get -y install python-setuptools
sudo apt-get -y remove --purge python-pip
sudo apt-get -y install python-pip
sudo pip install --upgrade pip
```

Next, we're going to take you through how to connect remotely to your Pi over a network (without needing to use a monitor keyboard and mouse with the Pi). However we strongly recommend you remember to take a backup of your SD card later. You can find a guide on how to do this later on: [Backing up your SD card](#).

2.3.7 Connecting Remotely

SSH via remot3.it

remot3.it services allow you to connect easily and securely to your Pi from a mobile app, browser window and a terminal. It allows you to control remote computers (such as the Pi) using TCP hosts such as SSH. You will be able to connect to your Pi from laptop or desktop at home. The free remot3.it account allows for multiple registered services and 8 hours connections on up to 1 concurrent service(s).

1. To configure weaved in our Pi, first we need to open an account on the [remot3.it website](#). You can register from your laptop or desktop.
2. Once you have an account: from your Pi terminal, we need to install weaved (which is the precursor on which remot3.it is based) to be able to connect our Pi. To install it:

```
sudo apt-get -y install weavedconnectd
```

3. Then we will open the weaved installer to link your Pi to your remot3.it account:

```
sudo weavedinstaller
```

4. Enter your remot3.it account username and password. Next, you will see this menu:

```

=====
Protocol          Port      Service      Weaved Name
=====

***** Main Menu *****
*
*      1) Attach/reinstall Weaved to a Service      *
*      2) Remove Weaved attachment from a Service  *
*      3) Exit                                       *
*
*****

Please select from the above options (1-3):
█

```

5. Then enter a name for your Pi (e.g. “pi01”). You can make it up, but remember to make a name easy for you to identify a specific Pi in case you have more than one attached to the weaved service:

```

Enter a name for your device (e.g. my_Pi_001).

The Device Name identifies your device in the remot3.it portal.
Your services will be grouped under the Device Name.

Only letters, numbers, underscore, space and dash are allowed.

myPi
.

```

6. Initially you won’t have any Weaved services installed, so the upper part is empty. Enter **1** to attach Weaved to an existing TCP service (host) on your Raspberry Pi. You should now see the following screen:

```
=====
Protocol          Port      Service      Weaved Name
=====

***** Protocol Selection Menu *****
*
* 1) SSH on default port 22              *
* 2) Web (HTTP) on default port 80       *
* 3) VNC on default port 5901            *
* 4) Custom (TCP)                        *
* 5) Return to previous menu             *
*
*****

Please select from the above options (1-5):
█
```

7. Enter **1** for SSH.
8. Next, we accept the default port (**y**).

```
The default port for SSH is 22.
Would you like to continue with the default port assignment? [y/n] █
```

9. The installer confirms your choice and asks you to give this connection a name:

```
We will attach a Weaved connection to the following service:

Protocol: ssh Port #: 22

Enter a name for this Service Attachment (e.g. SSH-Pi).
This name will be shown in your Weaved Device List.

Raspberry-Pi-SSH █
```

10. You will now return to the main menu, where you can see your Weaved Service Connection installed, then enter **3** to exit.

```

=====
Protocol      Port      Service      Weaved Name
=====
SSH           22        sshd          Raspberry-Pi-SSH

***** Main Menu *****
*
*      1) Attach/reinstall Weaved to a Service      *
*      2) Remove Weaved attachment from a Service  *
*      3) Exit                                       *
*
*****

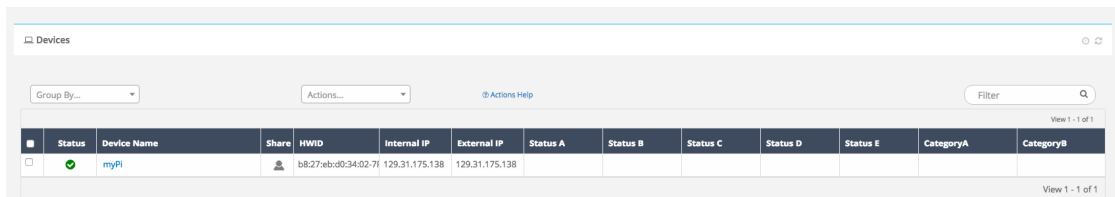
Please select from the above options (1-3):

```

Your Pi is now ready to run headless (without a direct connection to a screen), we just have to connect with it over SSH on our laptop to control it from the terminal. We have created two access guides, one for Linux and Mac Users and the other for Windows ([Accessing from Windows](#)).

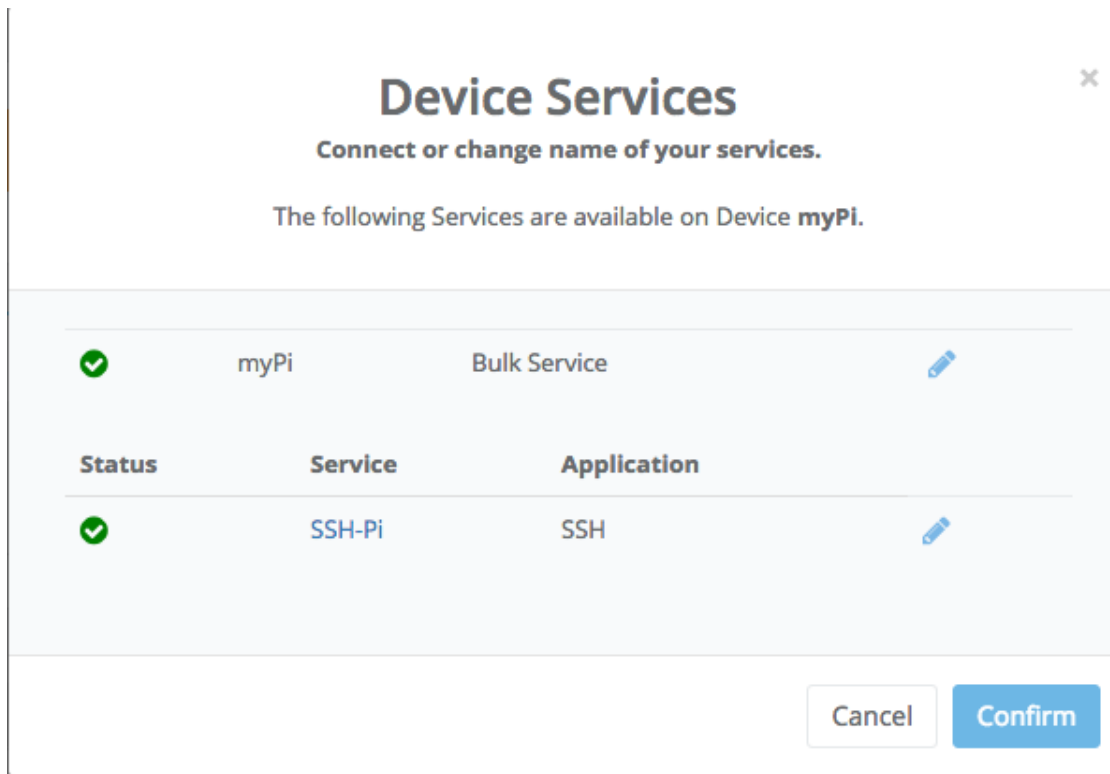
Accessing from Linux or macOS

1. We will now see how to access using your laptop to your Pi from the terminal. First, if you login to your remot3.it account, you will get a list of your devices:



Status	Device Name	Share	HWID	Internal IP	External IP	Status A	Status B	Status C	Status D	Status E	Category A	Category B
	myPi		b8:27:eb:00:34:02:7f	129.31.175.138	129.31.175.138							

2. In your case you will have just one item. When you click on the name of you device, a pop-up will open:



3. Click on the name of your ssh service and then “Confirm”.
4. A second pop-up will appear:

✕

SSH Connection

Your SSH connection to SSH-Pi is ready.

Use the following options in your SSH client application.

Copy and paste these values to your SSH application:

proxy54.yoics.net

30015

Or, copy and paste one of these command lines into your terminal window, based on your SSH username:

For pi username	ssh -l pi proxy54.yoics.net -p 30015
For root username	ssh -l root proxy54.yoics.net -p 30015
All others *	ssh -l LOGIN proxy54.yoics.net -p 30015

* Replace LOGIN with your device login name.

[Click here](#) for additional help

Back

Close

We copy the command after *For pi username*, in this example it is: `ssh -l pi proxy54.yoics.net -p 30015`. For you it will be different.

5. Then, paste the command in your laptop or desktop terminal app. (Optional alternative app for Mac)
6. The terminal is going to show you this message:

```
The authenticity of host '[proxy54.yoics.net]:32455 ([188.40.38.135]:32455)' can't be established.  
ECDSA key fingerprint is SHA256:lLaYgF4o08Uo0Ax5ZqHBZA7gBAkzz9AbdwyItUY4U8U.  
Are you sure you want to continue connecting (yes/no)?
```

Type yes.

7. Then, you will be prompted to enter a password, you should enter the password of the `pi` user of your Pi. By default it is `raspberrypi` but you should have changed it in an earlier chapter (*User password*).

You will see on your laptop's terminal that now you are user `pi`. You are connected from your laptop to your Pi. As long as your Pi is connected to the internet, you can remotely log into it and control it, so you don't need to use the display and mouse anymore.

For some more details on remote connections see [Alternative ways to connect via SSH](#).

Tip: To manage remote terminal sessions we suggest you use *Screen*, check out the guide later in the section *SSH*

using *Screen*.

Accessing from Windows

If your computer operative system is Windows, to access remotely you will need to install PuTTY, which is a free implementation of SSH and Telnet for Windows and Unix platforms.

1. To download it click [here](#).

There are cryptographic signatures available for all the files we offer below. We also supply cryptographically signed lists of checksums. Our signature policy, visit the [Keys page](#). If you need a Windows program to compute MD5 checksums, you could try this one at [pc-tool](#) by its author.)

Binaries

The latest release version (beta 0.67)

This will generally be a version we think is reasonably likely to work well. If you have a problem with the release version, it might be worth seeing if we've already fixed the bug, before reporting it.

For Windows on Intel x86

PuTTY:	putty.exe	(or by FTP)	(signature)
PuTTYtel:	puttytel.exe	(or by FTP)	(signature)
PSCP:	pscp.exe	(or by FTP)	(signature)
PSFTP:	psftp.exe	(or by FTP)	(signature)
Plink:	plink.exe	(or by FTP)	(signature)
Pageant:	pageant.exe	(or by FTP)	(signature)
PuTTYgen:	puttygen.exe	(or by FTP)	(signature)

A .ZIP file containing all the binaries (except PuTTYtel), and also the help files

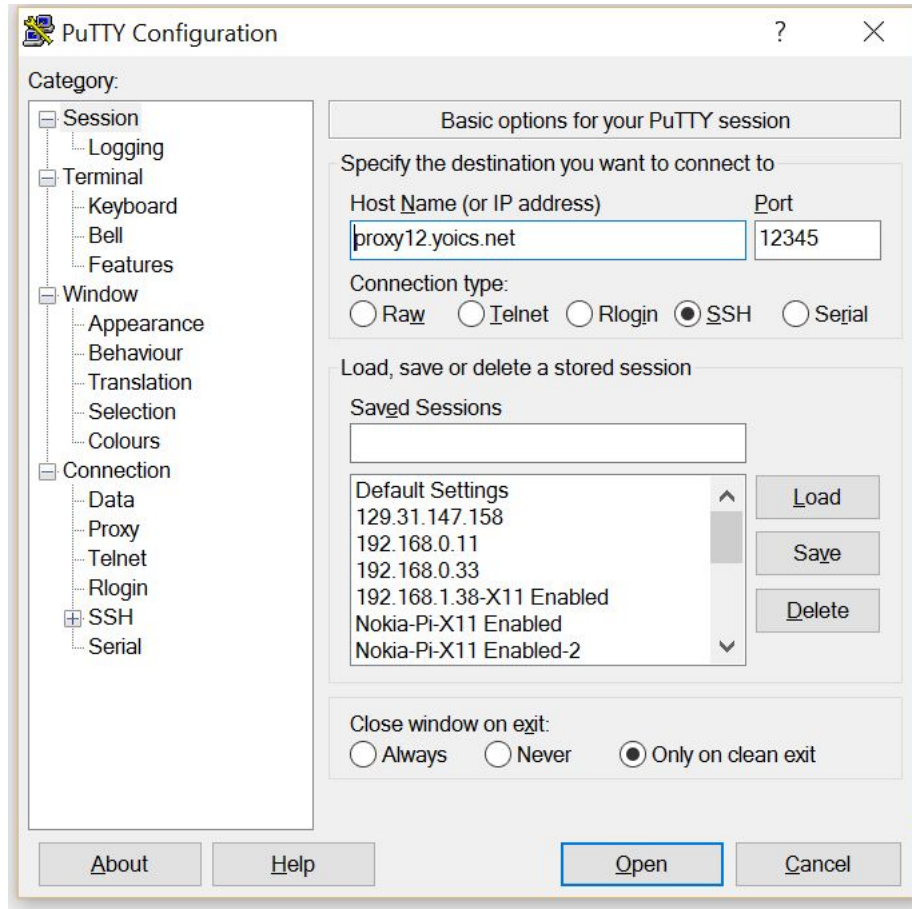
Zip file:	putty.zip	(or by FTP)	(signature)
-----------	---------------------------	-----------------------------	-----------------------------

A Windows MSI installer package for everything except PuTTYtel

Installer:	putty-0.67-installer.msi	(or by FTP)	(signature)
------------	--	-----------------------------	-----------------------------

Legacy Inno Setup installer. [Reportedly insecure!](#) Use with caution, if the MSI fails.

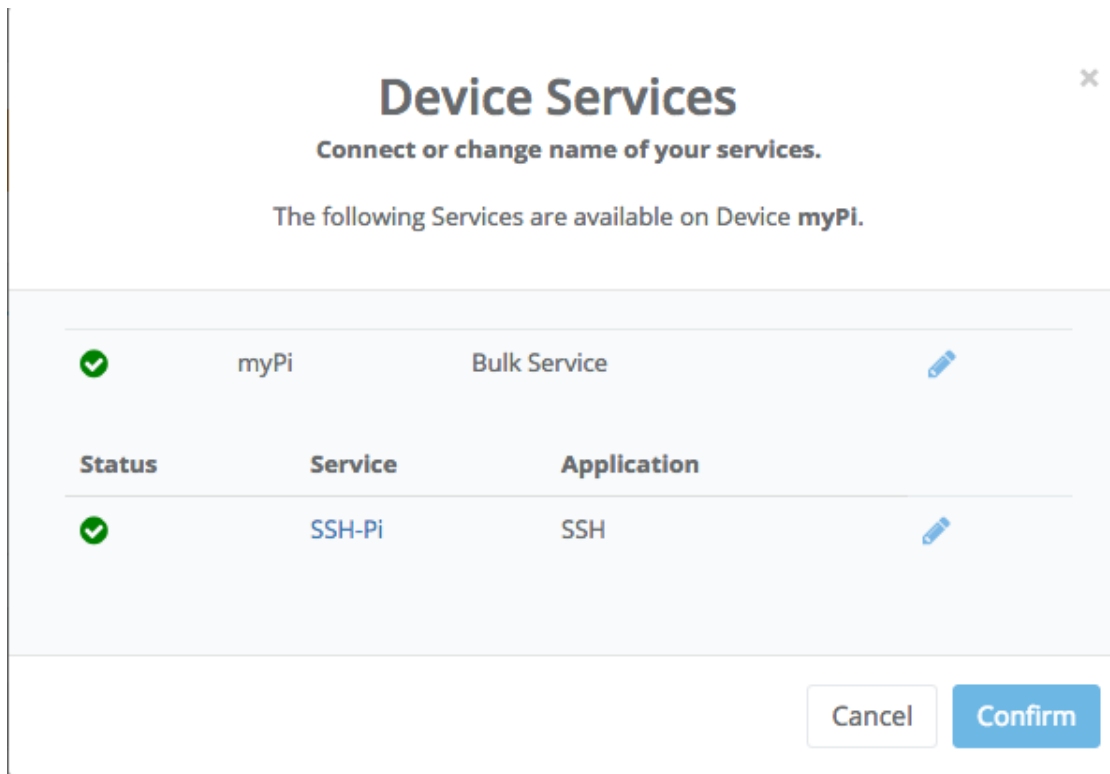
2. Once downloaded, proceed with the standard installation.
3. Once installed double click on the **putty.exe** and you will see a window that looks like the one below:



4. Then, if you login to your remot3.it account, you will get a list of the services linked to your devices:

Status	Device Name	Share	HWID	Internal IP	External IP	Status A	Status B	Status C	Status D	Status E	CategoryA	CategoryB
	myPi		b8:27:eb:00:34:02-7f	129.31.175.138	129.31.175.138							

5. In your case you will have just one item. When you click on the name of you device, a pop-up will open:



6. Click on the name of your ssh service and then “Confirm”.
7. A second pop-up will appear:

×

SSH Connection

Your SSH connection to SSH-Pi is ready.

Use the following options in your SSH client application.

Copy and paste these values to your SSH application:

proxy54.yoics.net

30015

Or, copy and paste one of these command lines into your terminal window, based on your SSH username:

For pi username	ssh -l pi proxy54.yoics.net -p 30015
For root username	ssh -l root proxy54.yoics.net -p 30015
All others *	ssh -l LOGIN proxy54.yoics.net -p 30015

* Replace LOGIN with your device login name.

[Click here](#) for additional help

Back

Close

5. Insert the server address and port obtained from remot3.it into Putty and connect!

Note: Rather than typing in `ssh -l pi <server> -p <port>`, you just need to insert the server url and port from remot3.it.

6. When asked for username and password, use your Pi username and password to log-in.

Note: This is not weaved username and password. The default password is `raspberry` but you should have changed it in an earlier chapter (*User password*).

```
login as: pi
pi@proxy50.rt3.io's password:
```

To exit your putty session, type “exit” and enter.

Tip: To manage remote terminal sessions we suggest you use *Screen*, check out the guide later in the section *SSH using Screen*.

For some more details on remote connections see [Alternative ways to connect via SSH](#).

SSH using Screen

Remember you can be connected to your Pi for up to 8 hours using **remot3.it**, after that time you have to connect again to your account and do the same access procedure we explained in the previous sections. Therefore we will show you how a *virtual terminal* can help you when you are working remotely on your Pi.

Screen is a full-screen software program allows you to use multiple windows (virtual VT100 terminals) in Unix. It offers a user to open several separate terminal instances inside a one single terminal window manager.

The screen application is very useful, if you are dealing with multiple programs from a command line interface and for separating programs from the terminal shell. It also allows you to share your sessions with others users and detach/attach terminal sessions.

When to use Screen?

One of the advantages of *Screen*, is that you can detach it. Then, you can restore it without losing anything you have done on the *Screen*. One of the typical scenario where *Screen* is of great help is when you are in the middle of SSH session and you want to download a file, update the operative, or transfer a big file to your RPi. The process could be 2 hours long. If you disconnect the SSH session, or suddenly the connection lost by accident, then the download process will stop. You have to start from the beginning again. To avoid that, we can use screen and detach it.

Installing Screen

Screen allows you to use multiple windows (virtual VT100 terminals) in Unix. If your local computer crashes, or you are connected remotely and lose the connection, the processes or login sessions you establish through screen don't get lost. To install Screen you can enter the following command on the Pi terminal:

```
sudo apt-get -y install screen
```

How to use Screen

- When you are in your terminal, you can create a *screen* or virtual terminal e.g. we will name the screen `mysession`:

```
pi@RPi-Dendrite:~ $ screen -S mysession
```

- Then you will be automatically attached to the `mysession` screen, that from now on we will call just "*screen*". You can now execute commands and work in the terminal without worrying to loose your work:

```
root@RPi-Dendrite:/home/pi# echo "hello world"
hello world
root@RPi-Dendrite:/home/pi#
```

- You can detach from the "*screen*" by pressing `Ctrl-A` and then `d`. Once detached we will be returned to our Pi terminal outside any *screen* session. To check the list of *active screens*: `screen -ls`

```
pi@RPi-Dendrite:~ $ screen -ls
There is a screen on:
      29097.mysession (14/10/16 12:23:55)      (Detached)
1 Socket in /var/run/screen/S-pi.

pi@RPi-Dendrite:~ $
```

- We get a list with all the screen IDs. If we want to attach to a particular *screen* we can enter `screen -r name_of_terminal` like in the example below:

```
pi@RPi-Dendrite:~ $ screen -r 29097
```

Basic commands to work with Screen

Screen command	Description
<code>screen -S name_of_terminal</code>	Assigning name to the virtual terminal or screen session
<code>screen -ls</code>	List all the virtual sessions or screens opened
<code>screen -X -S name_of_terminal quit</code>	Kill an specific virtual terminal.
<code>screen -r name_of_terminal</code>	Attach to the virtual terminal or screen
Press Ctrl-A and d	Detach from virtual terminal or screen
Press Ctrl-A and K	This command will leave and kill the virtual terminal or screen
Press Ctrl-A and n	Switching to the next virtual terminal or screen
Press Ctrl-A and p	Switching to the previous virtual terminal or screen

For additional commands check out the [Screen Cheatsheet](#)

Alternative ways to connect via SSH

We already know how to connect through remot3.it service, but we know that the connection lasts 8 hours and it allows us to work on one terminal session at a time. Therefore, with the help of remot3.it and another commands we can connect to our Pi for longer and using multiple terminals. In this section we are going to connect to our Pi using its IP address.

If you do not know what is an IP address, please go to the [this video](#) for a quick explanation. The IPs can be dynamic or static, but what is the difference? When a device is assigned a static IP address, the address does not change. Most devices use dynamic IP addresses, which are assigned by the network when they connect and change over time (which is the case for our Pi on the Imperial-WPA).

Get IP address from remot3.it

remot3.it displays the external IP of the devices you have registered. You can get your Pi's one in the *External IP* Tab:

<input type="checkbox"/>	Status	Device Name	Share	HWID	Internal IP	External IP	S
<input type="checkbox"/>	✓	myPi		b8:27:eb:d0:34:02-7f	129.31.175.191	129.31.175.191	
<input type="checkbox"/>	✗	myPi2 Last Online 2017-09-05T11:34:24.560Z		-smFcF04VvrE2qT2R	129.31.168.31	129.31.168.31	

Note: If you are connected with your laptop to the same network of your RPi the internal and external IP addresses will be the same like in the example above.

Get IP address from the terminal

We can use a command to check the different internet connections available on our system: `ifconfig` or `ifconfig -a`.

```
$ ifconfig
```

This command allows to know the IP addresses assigned to our Pi. The `wlan0`, indicates the status of the WiFi, and `eth0` shows the status of the Ethernet (wired) connection. In the next screen shoot shows an example of a Pi connected to the internet using the ethernet port. The red oval shows where to find the IP address assigned to the Pi for the Ethernet connection.

```
$ ifconfig
eth0      Link encap:Ethernet  HWaddr b8:27:eb:8a:16:3c
          inet addr:155.191.155.198  Bcast:155.198.155.255  Mask:255.255.255.0
          inet6 addr: 2001:630:12:1027:c5c0:89b:49f9:3256/64 Scope:Global
          inet6 addr: fe80::113e:d0e3:ce3c:eb0c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8908137 errors:0 dropped:222 overruns:0 frame:0
          TX packets:3734940 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1349950831 (1.2 GiB)  TX bytes:2241920928 (2.0 GiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:208 errors:0 dropped:0 overruns:0 frame:0
          TX packets:208 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:17240 (16.8 KiB)  TX bytes:17240 (16.8 KiB)

wlan0     Link encap:Ethernet  HWaddr b8:27:eb:df:43:69
          inet6 addr: fe80::ccdf:6559:a515:3c0f/64 Scope:Link
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:1786884 errors:0 dropped:4 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:605441611 (577.3 MiB)  TX bytes:0 (0.0 B)
```

You can find your IP address for the WiFi connection in the corresponding `wlan0 inet addr` field.

Connect knowing your IP

Once you know the IP (e.g. your IP is 192.31.123.122), you can access using your laptop terminal to the Pi like this:

```
$ ssh pi@192.31.123.122
```

Tip: The syntax for this command is `ssh username@ipaddress`. You may want to log in using a different username to the default `pi` if you created one.

Note: The IP addresses at Imperial are dynamic (most IPs are dynamic), meaning they are constantly changing and being reallocated as needed. It could be that your IP changes in a couple of hours, a day, or a bit longer, so be prepared to have to repeat the steps above to rediscover what your new IP address is.

VNC GUI control

Todo: Using VNC for remote GUI control will be added later.

Transferring files

Using terminal

If are programming on your laptop and you want to transfer your code to test it in your Pi, you can use a number of different methods:

- Secure Copy (`scp`)
- SSH File Transfer Protocol (`sftp`).
- Secure File Transfer Protocol (`ftps`)

Secure Copy (`scp`)

`scp` - Securely copy a file from one location to another.

The general syntax is as follows: `scp copy_from copy_to`

The locations are written relatively. So if you were to copy a file from one place on your local computer to another place, you would simply provide the path:

```
scp /home/pi/Desktop/myprogram.py /home/pi/Desktop/myfolder/
```

Which would copy the file `myprogram.py` to a folder within the same location.

Note: When you use SSH to remotely 'log in' to a computer such as a Pi, then all the commands you then type into your terminal are considered 'local'.

If you wanted to copy something from your computer to the Pi then that would be considered a local-to-remote copy.

For that you would need to :

1. Get the path to the file locally on your computer.
2. Get the path to the location on the Pi you would like to save it.

```
scp /Users/username/Desktop/program.py pi@192.168.1.10:/home/pi/Desktop
```

As you can see we copied the file `/Users/username/Desktop/program.py` **from** the local computer **to** a remote computer (which is why we need to prefix it with the username and IP address `pi@192.168.1.10`) in the location specified `/home/pi/Desktop`.

We can even copy a file back by reversing the order of the commands:

```
scp pi@192.168.1.10:/home/pi/Desktop/program.py /Users/username/Desktop/
```

Hint: If you want to copy a folder (not an individual file) then you need to add the *recursive* flag to the command. This tells the terminal that you want to copy the folder and all its sub-contents to the new location. i.e.

```
scp /Users/username/Desktop/Gizmo_Folder/ pi@192.168.1.10:/home/pi/Desktop/
```

SFTP

1. First we log into a session with the Pi using the correct username and IP address

```
sftp pi@192.168.1.1
```

2. Once establish the connection through SFTP, we can navigate around using `cd` (change directory) and `pwd` (print working directory) and `ls` (to list contents of current directory).
3. Once we have the a file to download from the remote computer (Pi):

```
get /path/to/file
```

Or for a folder:

```
get -r /path/to/directory/
```

4. To transfer files on our (local) computers to the remote (Pi) we can put:

```
put /path/of/local/file
```

The same flags that work with `get` apply to `put`. So to copy an entire local directory:

```
put -r /path/of/local/directory/
```

Note: More details and examples of SFTP can be [found here](#).


Using Software

Instead a terminal, we can use to transfer files using a software that mounts any remote server storage as a local disk in the Finder.app on Mac and the File Explorer on Windows. We suggest:

- [Cyberduck](#)

Cyberduck Quick Reference Guide

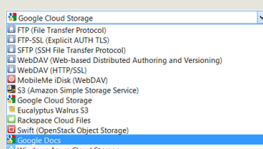
By Andrew Fogo & Becca Schmidt













Open a Connection

1. Open the **Open Connection** dialog box by either
 - (a) Clicking the Open Connection icon (🌐) on the toolbar
 - (b) Selecting *File > Open Connection* from the menu bar
 - (c) Pressing **ctrl + O** (PC) or **⌘ + O** (Mac) on your keyboard
2. From the dropdown box, select your desired connection type
3. Enter your necessary credentials
4. Press the **Connect** button

Your directory and list of files will appear.



Helpful Shortcuts

Open Connection  ctrl + O  ⌘ + O	Disconnect  ctrl + Y  ⌘ + Y
Upload File/Folder  alt + ↑	Download File/Folder  alt + ↓
Create New Bookmark  ctrl + O + B  ⌘ + O + B	Open Transfers Window  ctrl + T  ⌘ + T

Transfer Files

First make sure you are in your desired connection location.

Upload

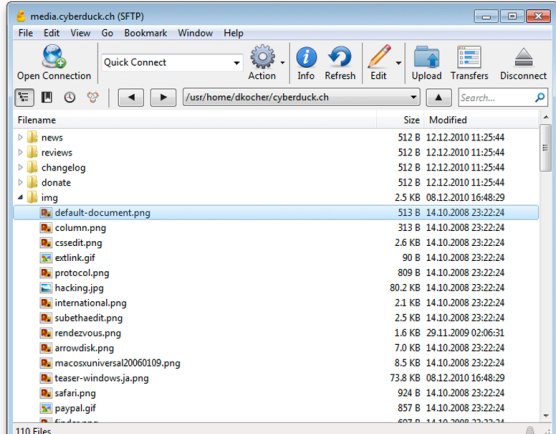
1. Upload file(s) or folder(s) by either
 - (a) Clicking the Upload icon (📁) on the toolbar
 - (b) Selecting *File > Upload* from the menu bar
 - (c) Pressing **alt + ↑** on your keyboard
 - (d) Clicking the Action icon (⚙️) and selecting Upload
2. Browse for the file(s) or folder(s) you wish to upload
3. Press the **Choose** button

The Transfer dialog box will appear and indicate the status of the transfer.

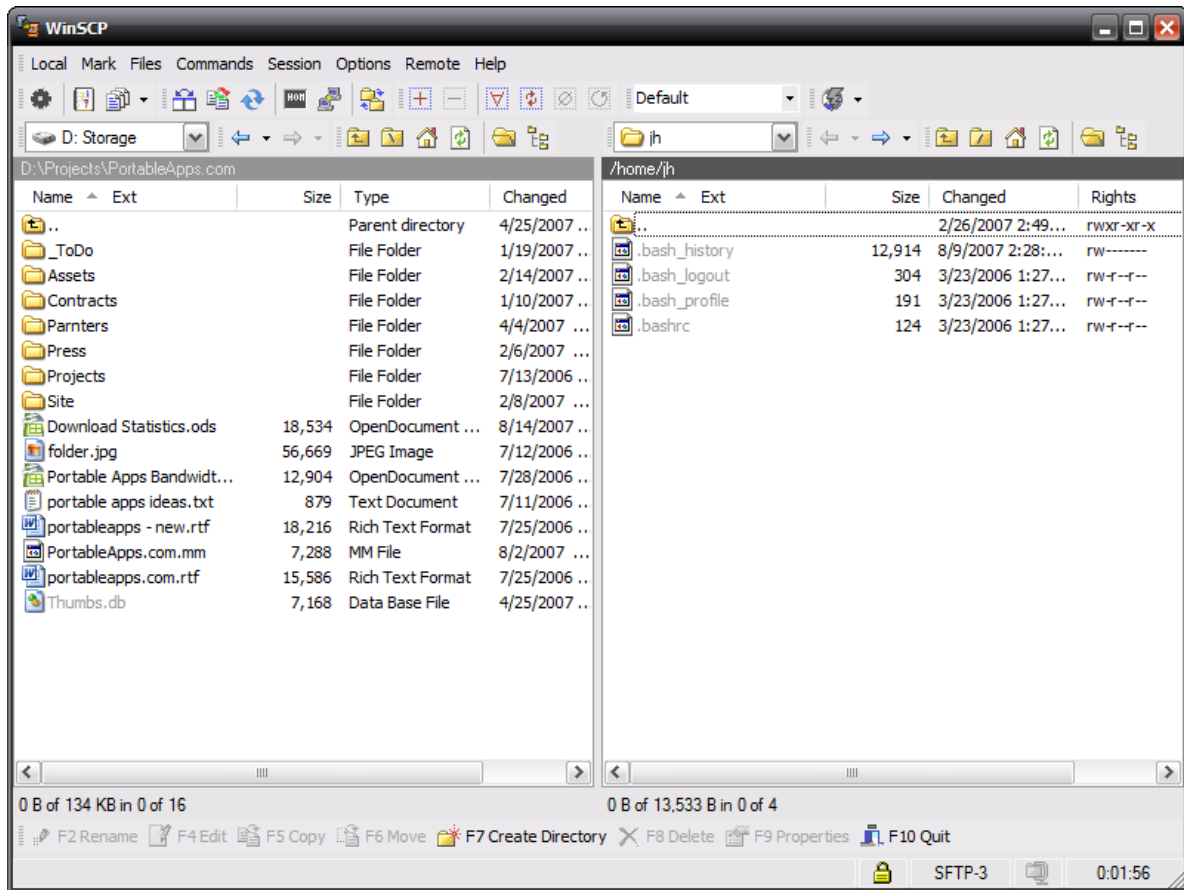
Download

1. Select the file(s) or folder(s) you wish to download
2. Download file(s) or folder(s) by either
 - (a) Right click your mouse and press download
 - (b) Selecting *File > Download* from the menu bar
 - (c) Pressing **alt + ↓** on your keyboard
 - (d) Clicking the Action icon (⚙️) and selecting Download

Tip: You may also drag & drop files or folders to upload and download them.



- For just Windows you can use: [WinSCP](#)



2.3.8 Backing up your SD card

It is useful and **always** advisable to backup a working copy of your Pi image. For example, make a backup copy after setting up WiFi. The next time the WiFi is not working, you can reformat the SD card and reinsert this backup copy to revert back to previous version. After this, your Pi will connect to the WiFi right away like before.

Important: Remember that if you SD card becomes corrupted you will likely have to wipe it and lose all your files to reformat it. Despite this being unlikely, always remember to take backups at key milestones throughout a project so that you can restore to a recent state and not lose too much work.

We also strongly recommend using Git to manage project files. It allows for collaborative working and means you can easily restore your project files onto any new device you might need.

Backup using Windows

1. [Download Win 32 Disk Imager](#) if none installed on your computer.
2. Insert the SDCard into your computer (e.g. via card reader or SD card slot if your computer has one).
3. Open Win 32 Disk Imager. Select a location and give a file name for the backup image.
4. Select the right drive.
5. Click Read.

6. Once done, keep this backup copy safe. Please note that the size of the backup is the same size of your SD Card. So please be mindful that it will take a considerable amount of disk space.

Backup using macOS

1. Open DiskUtility
2. Select “APPLE SD Card Reader Media”
3. Click on *File* → *New File* → *Image from “Untitled”*
4. Leave the selections “CD/DVD” and “no encryption”
5. Insert your password when asked
6. You’re done!

Restoring and image to the SD Card

If you have an image saved somewhere, you can restore it to your SD card at any time to revert back to that version. To do this, check the section on *Flashing your Disk Image*.

If you wish to install a fresh Raspbian OS, you should look at the chapter on *Setting up your SD Card*.

2.3.9 GPIO

The Raspberry Pi has General Purpose Input Output pins. This guide will give an short insight into controlling the GPIO pins on the Raspberry Pi using a Python library called **GPIOzero**.

1. *Materials needed*
2. *GPIO pinout*
3. *Analog vs. Digital*
4. *Blink*
5. *LED PWM*
6. *Button*
7. *Combining everything*

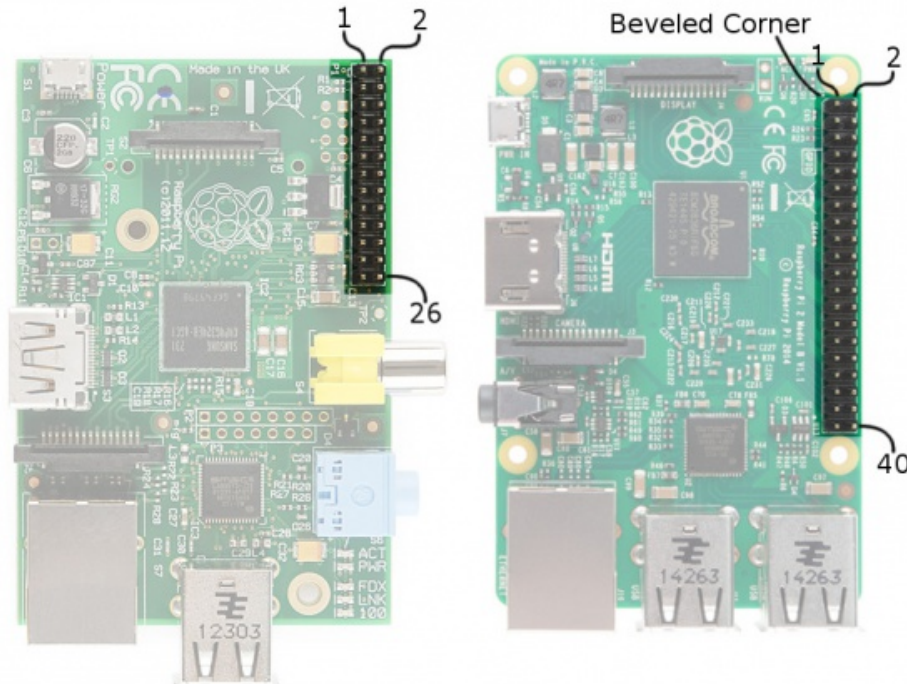
Your Raspberry Pi is more than just a small computer, it is a hardware prototyping tool! The RPi has **bi-directional I/O pins**, which you can use to drive LEDs, spin motors, or read button presses. To drive the RPi’s I/O lines requires a bit of programming. You can use a *variety of programing languages*, but we decided to use a really solid, easy language for driving I/O: **Python**.

Materials needed

- Raspberry Pi 3 B
- Breadboard
- Jumper Wires(M/F)
- Momentary Pushbutton Switch
- Resistors
- 2 LEDs

GPIO Pinout

Raspberry has its GPIO over a standard male header on the board. From the first models to the latest, the header has expanded from 26 pins to 40 pins while maintaining the original pinout.























There are (at least) two, different numbering schemes you may encounter when referencing **Pi pin numbers**:

1. **Broadcom (SoC) chip-specific** pin numbers.
2. **P1 physical** pin numbers.

You can use either number-system, but when you are programming how to use the pins, it requires that you declare which scheme you are using at the very beginning of your program. We will see this later.

The next table shows all 40 pins on the P1 header, including any particular function they may have, and their dual numbers:

Raspberry Pi 3 GPIO Header				
Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)		DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)		(I ² C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Rev. 2
29/02/2016

www.element14.com/RaspberryPi

In the next table, we show another numbering system along with the ones we showed above: **Pi pin header numbers and element14 given names, wiringPi numbers, Python numbers, and related silkscreen on the wedge.** The Broadcom pin numbers in the table are related to RPi Model 2 and later only.

Wedge Silk	Python (BCM)	WiringPi GPIO	Name	P1 Pin Number		Name	WiringPi GPIO	Python (BCM)	Wedge Silk
			3.3v DC Power	1	2	5v DC Power			
SDA		8	GPIO02 (SDA1, I2C)	3	4	5v DC Power			
SCL		9	GPIO03 (SCL1, I2C)	5	6	Ground			
G4	4	7	GPIO04 (GPIO_GCLK)	7	8	GPIO14 (TXD0)	15		TXO
			Ground	9	10	GPIO15 (RXD0)	16		RXI
G17	17	0	GPIO17 (GPIO_GEN0)	11	12	GPIO18 (GPIO_GEN1)	1	18	G18
G27	27	2	GPIO27 (GPIO_GEN2)	13	14	Ground			
G22	22	3	GPIO22 (GPIO_GEN3)	15	16	GPIO23 (GPIO_GEN4)	4	23	G23
			3.3v DC Power	17	18	GPIO24 (GPIO_GEN5)	5	24	G24
MOSI		12	GPIO10 (SPI_MOSI)	19	20	Ground			
MISO		13	GPIO09 (SPI_MISO)	21	22	GPIO25 (GPIO_GEN6)	6	25	G25
		(no worky 14)	GPIO11 (SPI_CLK)	23	24	GPIO08 (SPI_CE0_N)	10		CD0
			Ground	25	26	GPIO07 (SPI_CE1_N)	11		CE1
IDSD		30	ID_SD (I2C ID EEPROM)	27	28	ID_SC (I2C ID EEPROM)	31		IDSC
G05	5	21	GPIO05	29	30	Ground			
G6	6	22	GPIO06	31	32	GPIO12	26	12	G12
G13	13	23	GPIO13	33	34	Ground			
G19	19	24	GPIO19	35	36	GPIO16	27	16	G16
G26	26	25	GPIO26	37	38	GPIO20	28	20	G20
			Ground	39	40	GPIO21	29	21	G21

This table shows that the RPi not only gives you access to the bi-directional I/O pins, but also

- Serial (UART),
- I2C,
- SPI,
- and even some Pulse width modulation (PWM — a.k.a. “analog output”).

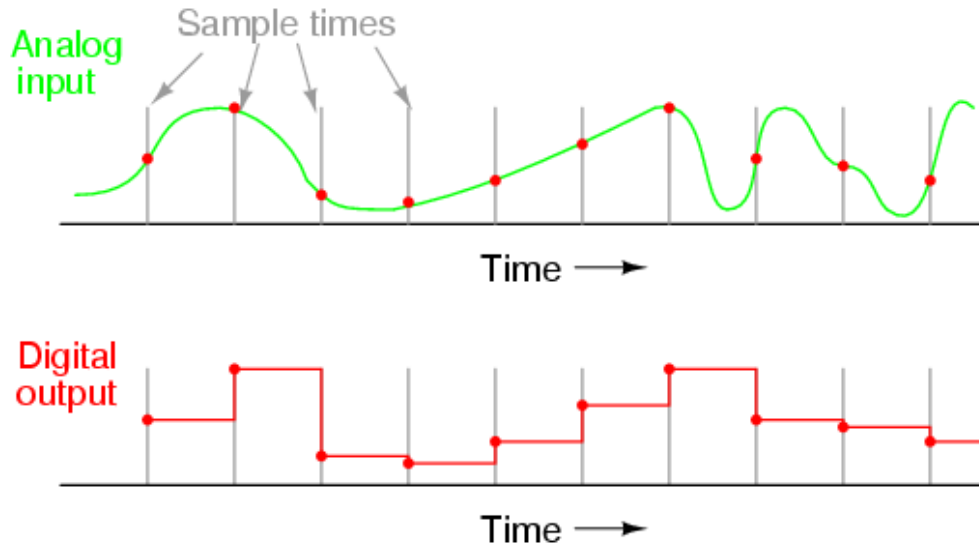
Tip: There is a useful online guide for finding the assignment and numbering of each pin, along with other guides, that you may find useful at: pinout.xyz

Analog vs. Digital

Before starting with our practise, we will revise the difference between **analog** and **digital** signals. Both are used to transmit information, usually through **electric signals**. In both these technologies, the information, such as any audio

or video, is transformed into electric signals. The **difference between analog and digital**:

- In **analog technology**, information is translated into electric pulses of varying amplitude.
- In **digital technology**, translation of information is into binary format (zero or one) where each bit is representative of two distinct amplitudes.



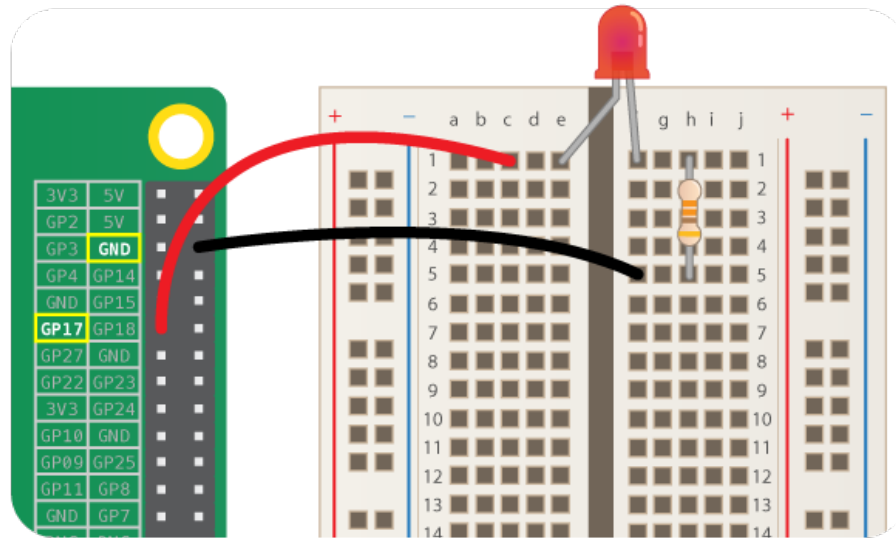
Comparison chart

Blink

We will start with a very easy example, the classic “Blink” example, later we will do the same with our Arduino and see the differences.

Hardware Setup

We start assembling the circuit as shown in the diagram below.



The Code

For the code we are going to use the [GPIOzero](#) library which is based on the [standard GPIO](#) library.

1. From your laptop's terminal connect to the RPi
2. Create a folder called "code" and inside it a file called "blinker.py":

```
$ mkdir code
$ cd code
$ nano blinker.py
```

Note: You may not need to create a new folder every time for the script. i.e. once you have created the folder code, you can create the scripts within the folder with `nano <script name>`. The command `nano` in this case is to open the nano editor.

3. Copy and paste this code:

```
#!/usr/bin/env python

from gpiozero import LED
from time import sleep

led = LED(17)

while True:
```

(continues on next page)

(continued from previous page)

```
led.on()
sleep(1)
led.off()
sleep(1)
```

4. Save and exit

5. Run this script with the command:

```
sudo python ./blinker.py
```

6. To stop the script from running press CTRL+C

7. To make the script an executable run:

```
$ sudo chmod u+x blinker.py
```

Now you can execute it with just this command:

```
$ ./blinker.py
```

8. Yay! The LED is blinking!

Understanding the “Blink” example

```
#!/usr/bin/env python
```

This line is used to tell which interpreter (in our case Python) to use when the file is made into an executable.

When we use Python to control our GPIO pins, we always need to import the corresponding Python module, which goes at the top of the script:

```
import gpiozero as gpio
```

Here, we are giving a shorter name to the module “GPIOzero”, in order to call the module through our script. This line is fundamental for every script requiring GPIO functions. If you want to import only certain classes from “GPIOzero” you could also specify the components. As an example, let’s say if you are interested in only the LED:

```
from gpiozero import LED
```

Or if want to use the Button and LED class.

```
from gpiozero import LED, Button
```

And if we are just importing the function sleep from the [time library](#), we will later use it to make the LED blink.

```
from time import sleep
```

In the next line:

```
led = LED(17)
```

Here we are creating a variable called `led` and we are initialising it with an object of the class `LED`. An object of the class `LED` to be initialised takes as a parameter the pin number to which the LED is connected to, in our case the pin number is 17 (BCM 17, not physical pin number 17).

Note: GPIOzero uses **ONLY** Broadcom (BCM) pin numbering, instead of physical pin numbering and it is not configurable, so when referring to pins in one of your scripts always use this numbering:

```
while True:
```

Here we are basically asking to Python to loop forever. In fact the `while` statements loops through its code until the initial condition becomes false, in our case never.

```
led.on()
sleep(1)
led.off()
sleep(1)
```

Here we are using two methods of the class `LED` of GPIOzero. `on()` switches the device on and `off()` turns it off. We are calling the two functions with a 1 second interval, in fact the function `sleep()` suspends execution for the given number of seconds.

LED PWM

Use the same layout for the electronics as before.

What is PWM?

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (3.3 Volts for RPi and 5 Volts for Arduino) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of “on time” is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

Hint: For more information check out [this link](#).

The Code

Repeat the same steps of “Blink” to upload the code below, this time call the file `led-pwm.py` and save it in the `code` folder that we have previously created. It’s up to you to make the code executable or not.

```
#!/usr/bin/env python

from gpiozero import PWMLED
from time import sleep

led = PWMLED(17)

while True:
    led.value = 0 # off
    sleep(1)
```

(continues on next page)

(continued from previous page)

```

led.value = 0.5 # half brightness
sleep(1)
led.value = 1 # full brightness
sleep(1)

```

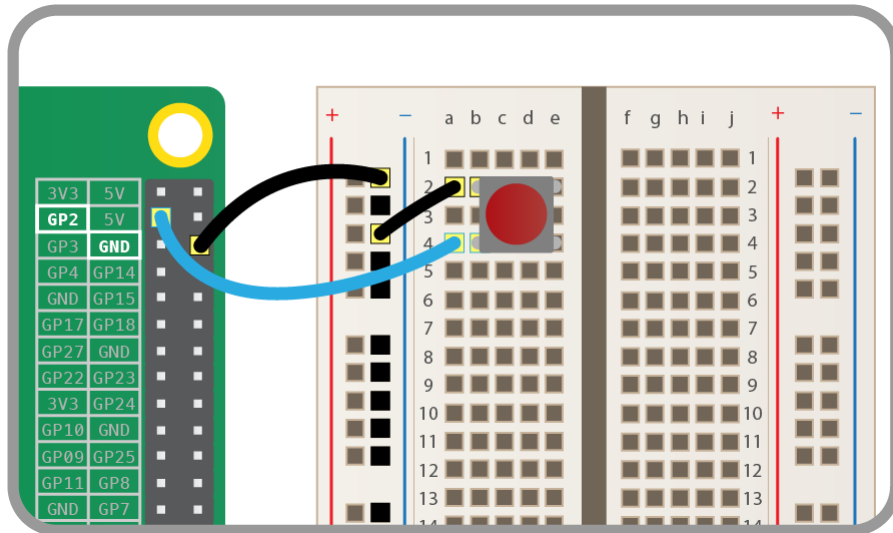
Understanding “LED PWM” code

The main difference here is that we are using the `class PWMLED` instead of the class `LED`. The `PWMLED` class has an extra parameter that we can tweak which is `value`. `value` indicates the duty cycle of this PWM device. `0.0` is off, `1.0` is fully on. Values in between may be specified for varying levels of power in the device.

Button

Hardware Setup

We start assembling the circuit as shown in the diagram below.



The Code

Repeat the same steps of “Blink” to upload the code below, this time call the file `button.py` and save it in the code folder that we have previously created. It’s up to you to make the code executable or not.

```

#!/usr/bin/env python

from gpiozero import Button

button = Button(2) # we first create an instance of the Button class
buttonWasPressed = False # 1st flag will help us track if the button was pressed in
↳ the last loop
buttonWasReleased = False # 2nd flag will help us track if the button was released in
↳ the last loop

```

(continues on next page)

(continued from previous page)

```
while True:

    if button.is_pressed:
        buttonWasReleased = False # reset back to false since the button is now_
→being pressed

        # we only want the print() code to be run once,
        # so if it was pressed the last time the code looped, don't print it this_
→time!
        if not buttonWasPressed:
            print("Button is pressed")

            # since we have now run this code, we don't want
            # it to run the next time the code loops, so
            buttonWasPressed = True

    else:
        # this code is run when the button is not being pressed
        buttonWasPressed = False
        if not buttonWasReleased:
            print("Button is released")
            buttonWasReleased = True
```

Understanding “Button” code

Here we are using the `class Button` from `GPIOzero`. This class has many functions and parameter, so make sure you check out the reference. Here we are using the `is_pressed` property of the class. `is_pressed` returns `True` if the device is currently active and `False` otherwise.

In this example, we also introduce the concept of flags. Flags are a way to help us keep track of the binary state of a particular thing by storing them as boolean variables (`1/0` or `True/False`). In this case, we need to keep track of the binary state of whether the button is was pressed, and the binary state of whether the button was released.

Note: We need to keep track of the *actions* that occurred, not the state of the button itself as this is already monitored `is_pressed` property of the `Button()` class.

In the code above, we use these flags to prevent the repetition of a print statement. i.e. If we did not have them, then the “*Button is released*” statement would print repetitively until the button was pressed, and vice versa.

Flags allow us to ensure the print statement is only *printed* on the first iteration of the loop. Every iteration thereafter will skip the print statement. This occurs until there is a change in the `is_pressed` property, at which point the respective flag is reset to `False`.

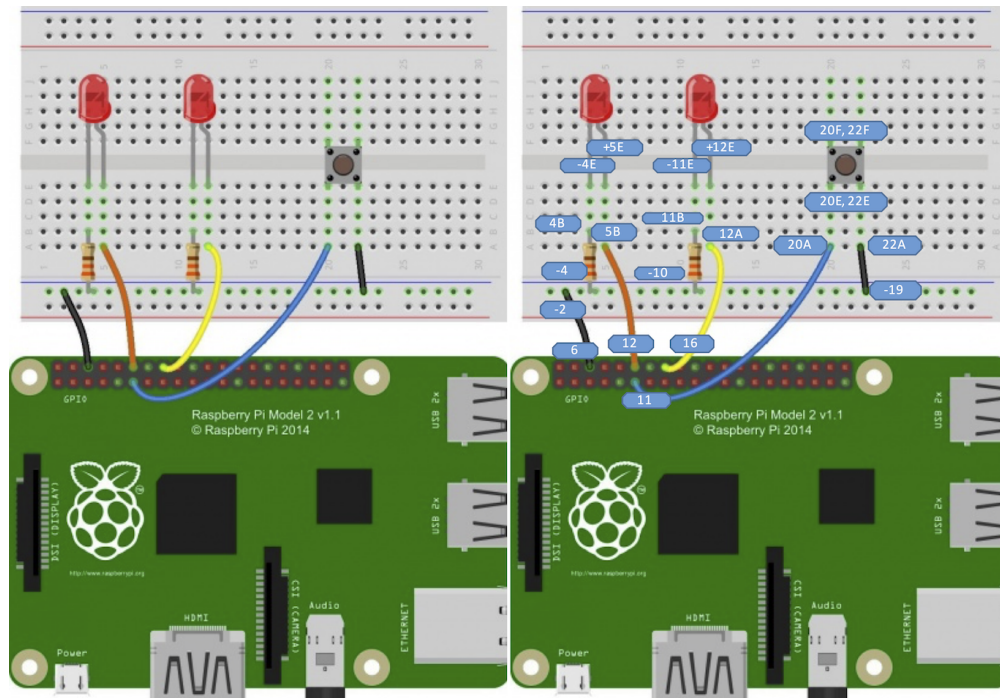
Combining Everything

Now we challenge you to combine all the previous three scripts to create one. Make the script in order that:

- when the button is pressed one of the two LEDs fades to 25% of its brightness and the other one blinks once
- when the button is released the PWM LED goes back to 100% brightness.

Hardware Setup

We start assembling the circuit as shown in the diagram below.



Code Tips

Use the `when_pressed` and `when_released` properties of the `Button` class. You can find the code to control one LED with the button, [here](#).

Acknowledgements

- Based on the GPIOzero library [notes](#),
- [this reference](#),
- and [this Sparkfun intro](#).

2.3.10 Using Peripherals

Motor HAT

Overview

The **DC+Stepper Motor HAT from Adafruit** is a perfect add-on for any motor project as it can drive up to 4 DC or 2 Stepper motors with full PWM speed control. However, the Raspberry Pi does not have a lot of PWM pins, we use a fully-dedicated PWM driver chip onboard to both control motor direction and speed. This chip handles all the motor and speed controls over I2C. Only two GPIO pins (SDA & SCL) are required to drive the multiple motors, and since it is I2C you can also connect any other I2C devices or HATs to the same pins.

Note: I2C is a very commonly used standard designed to allow one chip to talk to another. So, since the Raspberry Pi can talk I2C we can connect it to a variety of I2C capable chips and modules.

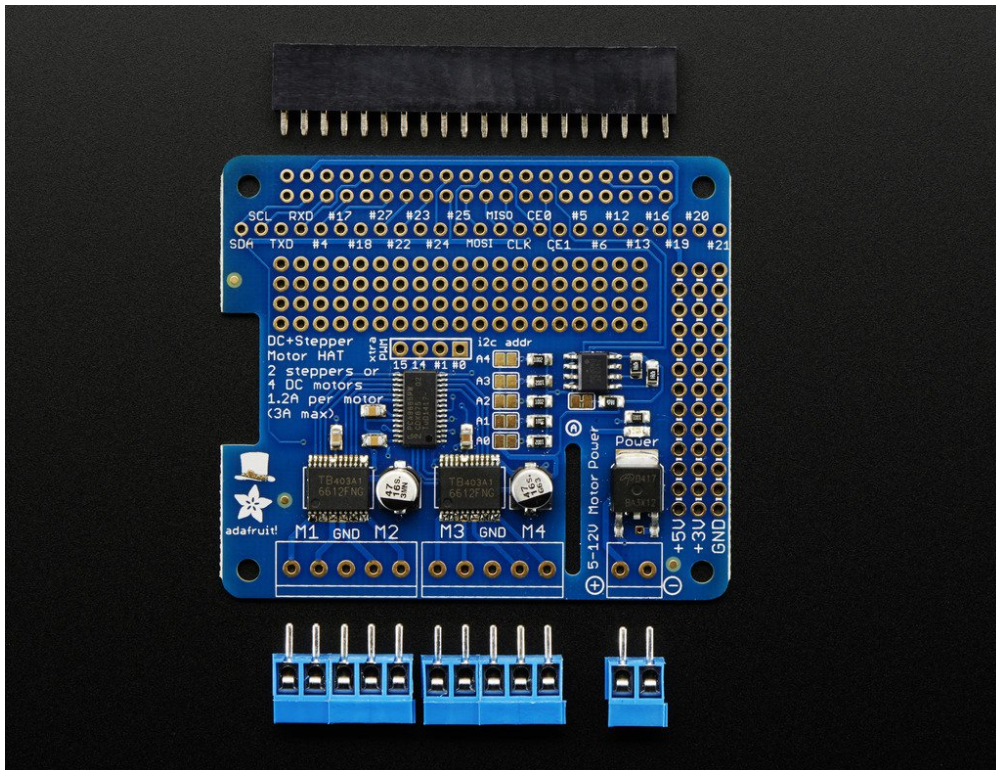
You can find out more information on I2C in the [Communication](#) section.

Features

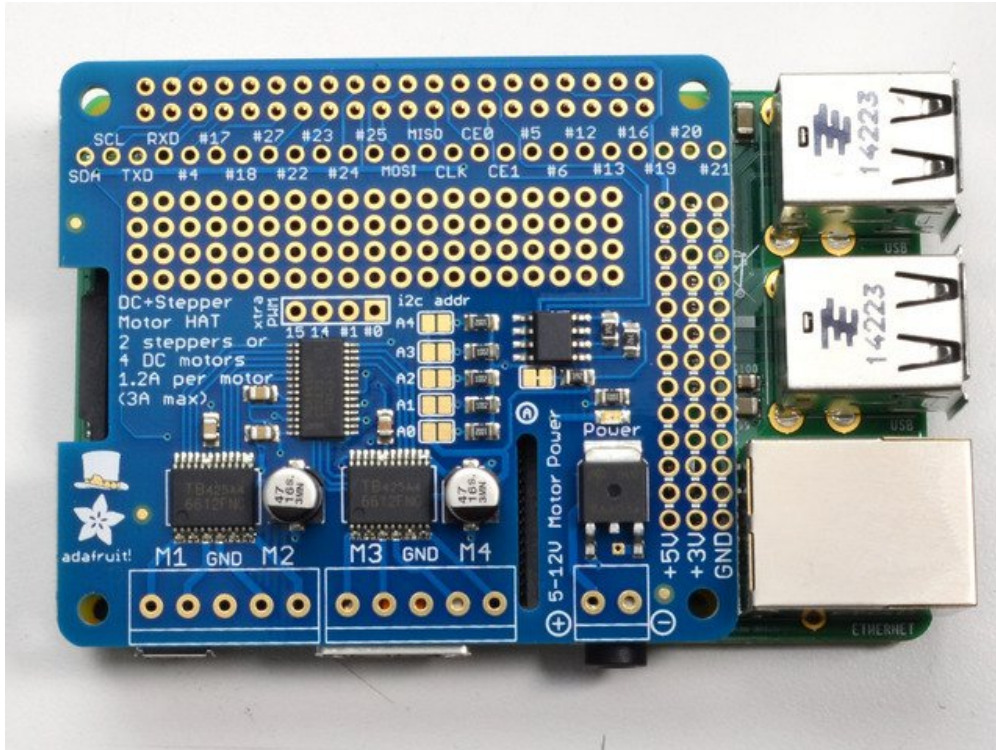
- 4 H-Bridges: TB6612 MOSFET chipset provides 1.2A per bridge (3A brief peak) with thermal shutdown protection, internal kickback protection diodes. Can run motors on 4.5VDC to 13.5VDC.
- **Up to 4 bi-directional DC motors** with individual 8-bit speed selection (so, about 0.5% resolution).
- **Up to 2 stepper motors** (unipolar or bipolar) with single coil, double coil, interleaved or micro-stepping.
- Big terminal block connectors to easily hook up wires (18-26AWG) and power.
- Polarity protected 2-pin terminal block and jumper to connect external 5-12VDC power.
- Install the easy-to-use Python library.

Assembly

The Motor HAT comes with an assembled and tested HAT, terminal blocks, and 2x20 plain header. Some soldering is required to assemble the headers on. Here we leave a link with a [step-by-step](#) guide of how to solder the headers and a video to show you [tips on soldering](#).



Once the motor HAT is assembled, we place it on top so that the short pins of the 2x20 header line up with the pads on the HAT.



Powering Motors

Note the HAT does not power the Raspberry Pi, and we strongly recommend having two separate power supplies - one for the Pi and one for the motors, as motors can put a lot of noise onto a power supply and it could cause stability problems.

Voltage requirements

The motor controllers on this HAT are designed to run from **5V to 12V**. Therefore, the first important thing is to verify the voltage specifications for the motor. Some small hobby motors are only intended to run at 1.5V, but its just as common to have 6-12V motors.

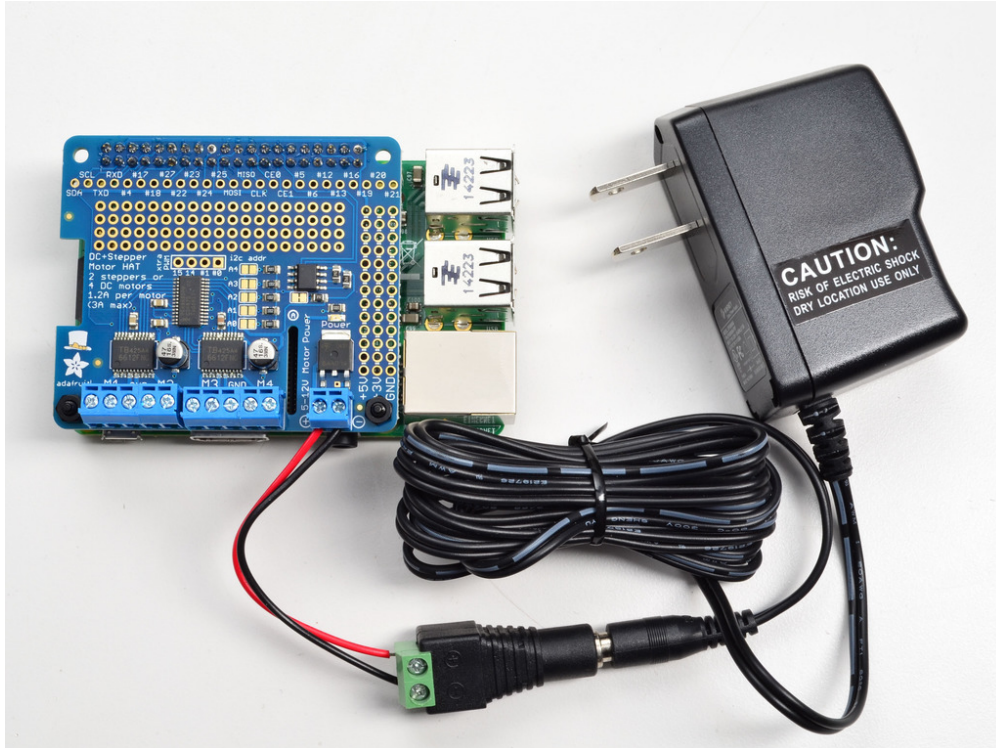
Warning: Most 1.5-3V motors WILL NOT WORK or will be damaged by 5V power.

Current requirements

The motor driver chips that come with the kit are designed to provide up to **1.2 A per motor**, with 3A peak current. Note that once you head towards 2A you will probably want to put a heat-sink on the motor driver, otherwise you will get thermal failure, possibly burning out the chip.

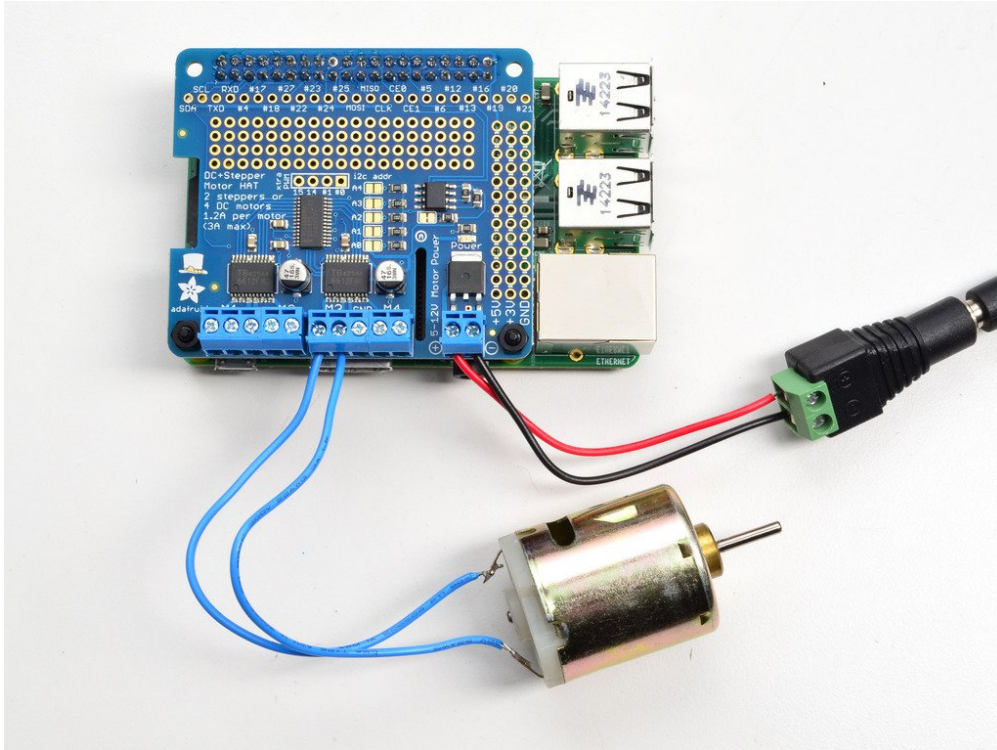
Important: You can not run motors off of a 9V battery, so don't waste your time/batteries!

Therefore, you can use a 9V 1A, 12V 1A, or 12V 5A DC regulated switching power adapter. In case you want to make it portable, you can use a big Lead Acid or multiple-AA NiMH battery pack of 4 to 8 batteries to vary the voltage from about 6V to 12V as your motors require.



Connecting DC Motors

To connect a motor, simply solder two wires to the terminals and then connect them to either the **M1**, **M2**, **M3**, or **M4**. If your motor is running ‘backwards’ from the way you like, just swap the wires in the terminal block. For this demo, please connect it to **M3**.



Installing Software

1. We can download the Python library to control DC and stepper motors. Before you start, we need to install the python `smbus` library. For the latter, execute the following command:

```
$ sudo apt-get install python-smbus
```

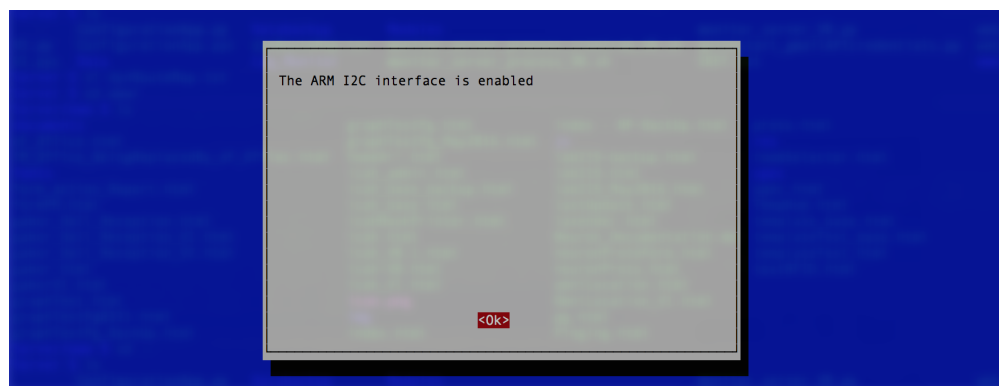
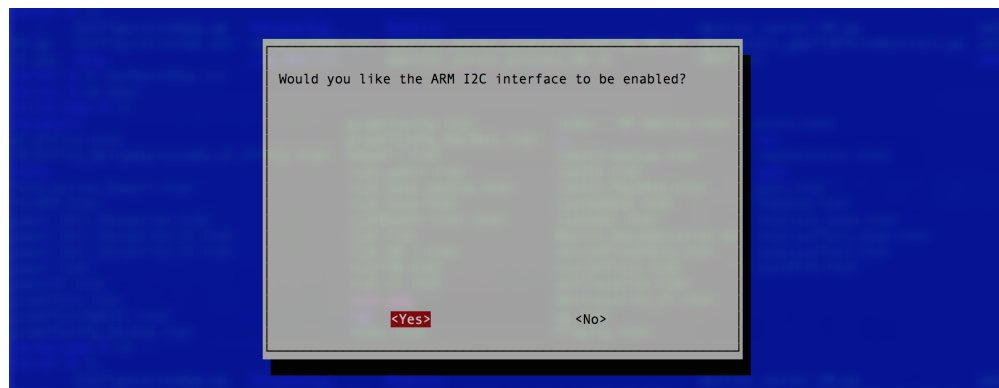
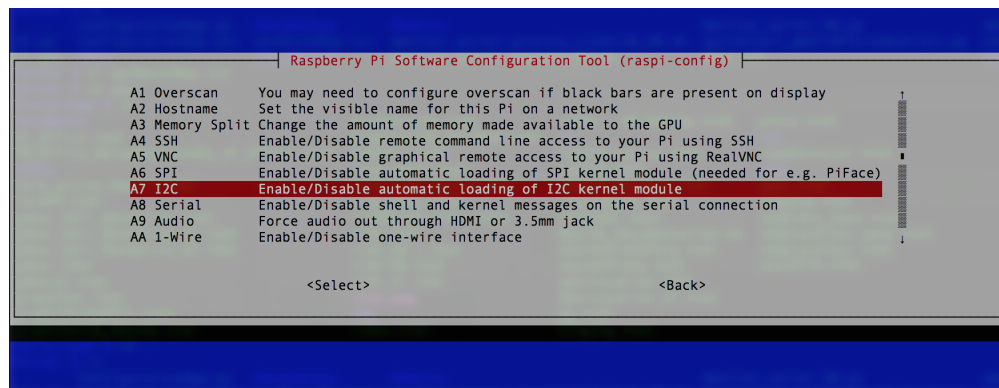
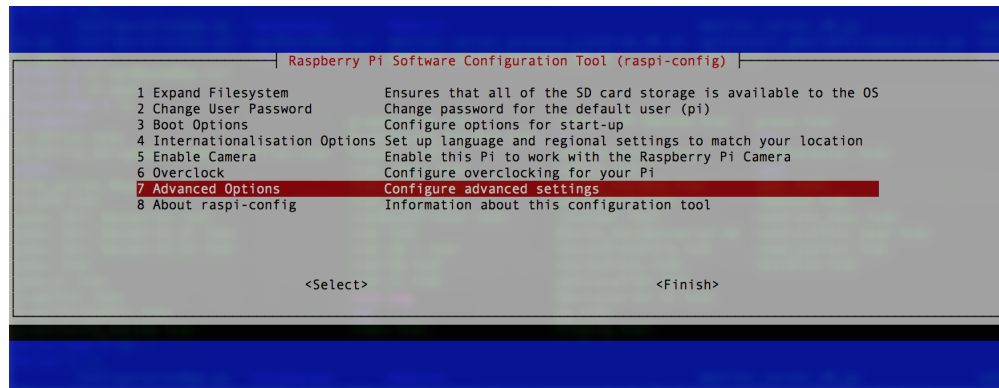
2. Now, we download the code as:

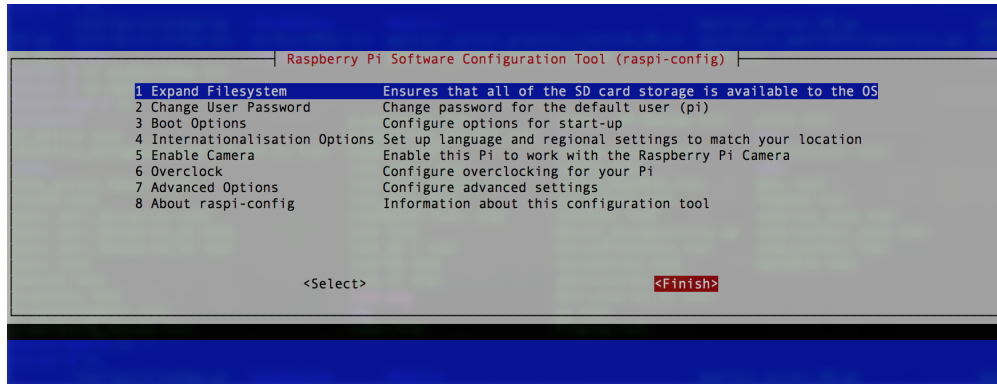
```
$ cd code
$ git clone https://github.com/adafruit/Adafruit-Motor-HAT-Python-Library.git
$ cd Adafruit-Motor-HAT-Python-Library
$ sudo python setup.py install
```

3. Before going further to the next step, we need to configuring the I2C if has not been done yet. Run:

```
$ sudo raspi-config
```

and follow the prompts to install I2C support for the ARM core and linux kernel:





Now reboot!

- Now you can get started with testing to watch your motor spin back and forth. First access to:

```
$ cd Adafruit-Motor-HAT-Python/examples
$ nano DCTest.py
```

Here you will see the code which shows you everything the MotorHAT library can do and how to do it.

DC motor control

- Start with importing at least these libraries:

```
#!/usr/bin/python
from Adafruit_MotorHAT import Adafruit_MotorHAT, Adafruit_DCMotor

import time
import atexit
```

- The MotorHAT library contains a few different classes, one is the **MotorHAT class** itself which is the main PWM controller. You always need to create an object, and set the address (or frequency). By default the address is 0x60. We can change this address, but for now we are not going to do it.

```
# create a default object, no changes to I2C address or frequency
mh = Adafruit_MotorHAT(addr=0x60)
```

- The PWM driver is 'free running' - that means that even if the python code or Pi linux kernel crashes, the PWM driver will still continue to work. But it means that the motors **DO NOT STOP** when the python code quits.

For that reason, we strongly recommend this 'at exit' code when using DC motors, it will do its best to shut down all the motors.

```
# recommended for auto-disabling motors on shutdown!
def turnOffMotors():
    mh.getMotor(1).run(Adafruit_MotorHAT.RELEASE)
    mh.getMotor(2).run(Adafruit_MotorHAT.RELEASE)
    mh.getMotor(3).run(Adafruit_MotorHAT.RELEASE)
    mh.getMotor(4).run(Adafruit_MotorHAT.RELEASE)

atexit.register(turnOffMotors)
```

- Now that you have the motor HAT object, note that each HAT can control up to 4 motors. That means you can have multiple HATs running.

To create the actual DC motor object, you can request it from the MotorHAT object you created above with `getMotor(num)` with a value between 1 and 4, for the terminal number that the motor is attached to

```
# In this case is M3
myMotor = mh.getMotor(3)
```

DC motors are simple beasts, you can basically only set the speed and direction.

5. **To set the speed**, call `setSpeed(speed)` where `speed` varies from 0 (off) to 255 (Maximum). This is the PWM duty cycle of the motor.

```
# set the speed to start, from 0 (off) to 255 (max speed)
myMotor.setSpeed(150)
```

6. **To set the direction**, we use the function `run(direction)` where `direction` is a constant from one of the following:

7. Remember that

- `Adafruit_MotorHAT.FORWARD` - DC motor spins forward.
- `Adafruit_MotorHAT.BACKWARD` - DC motor spins backward.
- `Adafruit_MotorHAT.RELEASE` - DC motor is 'off', not spinning but will also not hold its place.

```
while True:
    print("Forward! ")
    myMotor.run(Adafruit_MotorHAT.FORWARD)

    print("\tSpeed up...") # This will loop from 0-254
    for i in range(255):
        myMotor.setSpeed(i)
        time.sleep(0.01) # It will stop 10 ms

    print("\tSlow down...") # This will loop from 244-0
    for i in reversed(range(255)):
        myMotor.setSpeed(i)
        time.sleep(0.01)

    print("Backward! ")
    myMotor.run(Adafruit_MotorHAT.BACKWARD)

    print("\tSpeed up...")
    for i in range(255):
        myMotor.setSpeed(i)
        time.sleep(0.01)

    print("\tSlow down...")
    for i in reversed(range(255)):
        myMotor.setSpeed(i)
        time.sleep(0.01)

    print("Release")
    myMotor.run(Adafruit_MotorHAT.RELEASE)
    time.sleep(1.0)
```

Acknowledgements

Reference [1] Reference [2]

Available I/O e.g. cameras/screens/touch screens/drivers

Todo: More information will be added here soon.

2.3.11 Why Raspberry Pi?

The Raspberry Pi definitely performs best when there are heavy calculations into play but also is great for:

- More processing power, so the Raspberry can come to the rescue.
- Graphical applications
- Big Data projects
- Internet or network connectivity projects (IoT)
- The need for USB peripherals such as a web cam
- ... and many more applications!

To choose between the two there's is this [Make:zine article](#) that can help you out.

2.3.12 Alternatives to the Raspberry Pi

Here are some alternatives to the Raspberry Pi, although it's important to remember that the Pi really is the best in its field with many *many* examples available around the internet.

- [Arduino Yun](#) it is an Arduino with WiFi connection capabilities ideal for IoT
- [Intel IoT Developer Kit](#)
- [Photon](#) very tiny Wifi enabled board
- [Beaglebone](#) it is a Raspberry Pi competitor with some analog pins functionality
- [ESP8266](#)

2.4 Communication

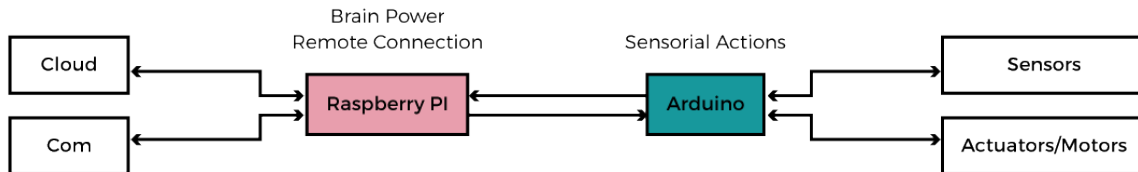
2.4.1 Protocol: Serial

Serial is an important protocol for transferring data between two computational devices. In telecommunication and data transmission, serial communication is the process of sending data one bit at a time, sequentially, over a communication channel or computer bus. This is in contrast to parallel communication, where several bits are sent as a whole, on a link with several parallel channels. Serial communication is used for all long-haul communication and most computer networks. [1]

Combining the Arduino for the Raspberry Pi

In this tutorial we will see how to connect your Raspberry Pi to your Arduino. We will start by installing the Arduino IDE that you have already seen and then move to some command line tools. These tools don't need a graphical interface and you can use them without a monitor. The last section will give you some advices on how to install these tools on your laptop.

By combining the strengths of both the Raspberry Pi and an Arduino we get a very useful control system - you can think of the Raspberry Pi as a computer that offers computational power and network capabilities, whereas Arduino communicates with sensors and actuators:



Talking Over Serial

To communicate between the Raspberry Pi and the Arduino over a serial connection, we'll use the built-in Serial library on the Arduino side, and the Python serial module on the Pi side.

1. To install the serial module, run the following commands on your Pi terminal:

```
$ sudo apt-get install python-serial python3-serial
```

2. Now we want to **upload a new sketch on the Arduino**. Plug into your personal computer and upload the code as follows:

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  for(byte n=0; n<255; n++){
    Serial.write(n);
    delay(50);
  }
}
```

This code counts upward and sends each number over the serial connection. Note that in Arduino, `Serial.write()` sends the actual number in the byte type, the actual 8-bit representation of the number.

3. Now plug the Arduino to the USB of the Raspberry. Then in the Pi terminal let's open the Python shell by typing:

```
$ python
```

This will launch the Python interpreter and the `>>>` prompt should appear.

4. Now we type:

```
from serial import Serial
```

If successful, when you press enter you should see no errors, and the cursor will return to the >>> prompt. Now we can use the `Serial` class to connect to our Arduino.

5. We create a variable of type `Serial` that represents the serial connection with our Arduino:

```
serial_from_arduino = Serial('/dev/ttyACM0')
```

6. We can read one bit at a time what the Arduino has written over serial like this:

```
input = serial_from_arduino.read(1)
```

7. Then we print it in the console like this:

```
print(ord(input))
```

The function `ord()`, given a string of length one, returns the value of the byte when the argument is an 8-bit string. You should see a 0 being printed.

Note: If you have problems with this step, make sure you have properly done step 2 (uploading the new sketch to Arduino).

Tip: For the next step, you need to use **Ctrl+D** to exit the Python prompt.

8. Now that we have tested that the connection works we want to write a Python script that reads the messages from serial, so we type:

```
$ cd home/pi/
$ nano serialEcho.py
```

And we paste this code:

```
import serial
port = "/dev/ttyACM0"
serial_from_arduino = serial.Serial(port, 9600)
serial_from_arduino.flushInput()
while True:
    if (serial_from_arduino.inWaiting() > 0):
        input = serial_from_arduino.read(1)
        print(ord(input))
```

The meaning of each line is as follows:

- `import serial`: just like before we import the serial library
- `port = "/dev/ttyACM0"`: this time we save the port path in a variable
- `serial_from_arduino = serial.Serial(port, 9600)`: we create an object of the class `Serial`, this time we specify the baud rate of our serial connection
- `serial_from_arduino.flushInput()`: we clear out the input buffer
- `while True:`: we put the reading functions in a loop so we keep on reading the values written by the Arduino constantly

- `if (serial_from_arduino.in_Waiting() > 0) ::` we check that we are receiving bytes (i.e. that the input buffer is not empty)
- `input = serial_from_arduino.read(1):` we read the content of the input buffer one byte at a time
- `print(ord(input)):` we interpret the incoming byte and we print it in the console

The Arduino is sending a number to the Python script, which interprets that number as a string. The input variable will contain whatever character maps to that number in the ASCII table. To get a better idea, try replacing the last line of the Python script with this:

```
print(str(ord(input)) + " = the ASCII character " + input + ".")
```

In Python to check if what you are getting is a string you can use [the method explained here](#).

Raspberry Pi to Arduino

To have the Raspberry Pi write and Arduino read (and turn on the built-in LED) you can use this code:

1. On the Arduino side upload this code:

```
const int ledPin = 13;

void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  if (Serial.available()) {
    light(Serial.read() - '0');
  }
  delay(500);
}

void light(int n) {
  for (int i = 0; i < n; i++) {
    digitalWrite(ledPin, HIGH);
    delay(100);
    digitalWrite(ledPin, LOW);
    delay(100);
  }
}
```

2. On the Raspberry Pi run this code:

```
import serial
serialToArduino = serial.Serial('/dev/ttyACM0', 9600)
serialToArduino.write('3')
```

Other methods

Sometimes the communication over Serial is not the best option for your project or you might want to make your Pi and Arduino communicate in another way, or maybe communicate to other boards, so see the other Chapters for possible alternatives, but don't limit yourself to the ones listed. They are just brief introductions with plenty of links for a more in-depth knowledge. We leave this exploration to your curiosity!

[Serial over GPIO](#) with this method you can use the same code we have used before, the only difference is the physical connection. **Remember** you need to use the same voltage level between the two digital pins. Since the Pi and Arduino operate at different voltage levels you will need a voltage converter.

Tip: Since you are connecting the Arduino to the Pi using USB, and the Pi is a computer, there's no reason why you couldn't run the Python scripts above using your own computer instead of the Pi.

This is very easy to set up on a Mac or Linux.

If you have Mac special attention to the path of the USB port. It is going to look like this `/dev/tty.usbmodem411` to find your port name you can enter the command `ls /dev/tty.usb*`.

Also on Mac there is no `apt-get` command. You have to install another package manager, the most common one which we also recommend using [Homebrew](#) and to install any package use `brew install PACKAGE_NAME`.

- [I2C](#) is protocol that allows two devices to talk to each other using only two buses: a clock one (SCL bus) and a data one (SDA bus). It can allow up to 127 slaves connected to one master to exchange information. It is a very common protocol for Arduino as it is used to communicate with various sensors. There is a [dedicated library called Wire](#) in Arduino that you can readily use.
- [SPI](#) is a synchronous serial communication interface specification used for short distance communication, primarily in embedded systems. It uses four buses: clock (SCK), two data lines (MISO: Master Output Slave Input, MOSI: Master Input Slave Output) and a select line(SS) to choose among the multiple slave devices.
- [Noduino](#) is a JavaScript and Node.js framework for accessing basic Arduino controls from web applications using HTML5, Socket.IO and Node.js.
- [UDP](#)
- [MQTT](#)

2.5 Sensors & Actuators

Todo: Sensors & Actuators index page

- Basic Electronics
- Sensors
- Actuators

2.6 Supplementary Material

Please feel free to make additional suggestions!

This section documents common tools, commands, and functionality of Unix-based operating systems such as Raspbian, Linux, and macOS.

Todo: More supplementary material will be added here later.

2.6.1 Crontab - scheduling commands

Crontab (chron tab → time table) is a scheduling assistant built into Unix systems. It is very useful for running commands in a terminal environment at specific times.

Say you have a python3 file called `script.py` that posts to the internet, and you need to run it at a regular interval. One option would be to keep the script running and sleep it using `time.sleep(s)` to pause it for `s` seconds. However if this is running over a long period, this is not ideal. What happens if the script crashes for some reason?

First thing is to make the python3 script an executable. To do this you must first ensure you have included a [shebang line](#). In the python file, it must be the first line of the file:

```
#!/usr/bin/python3
import a
import b
import c

# ... rest of file
```

Then you can change the permissions of the file to make it executable:

```
$ chmod u+x /home/pi/script.py
```

Note: This assumes that `script.py` is saved in the `/home/pi` directory.

To check that this worked, you can use the `ls` command when in the home directory.

```
$ cd /home/pi
$ ls
```

The contents of the directory will be printed, and the `script.py` file will be coloured differently (either green or red) instead of white. This means the script can now be executed directly (meaning you don't need to write `python3` beforehand):

```
$ /home/pi/script.py
```

This is important as you can now add it to the crontab configuration file.

```
$ crontab -e
```

This will open the crontab configuration looks like this:

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
```

(continues on next page)

(continued from previous page)

```
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow    command

@reboot /home/pi/script.py
*/10 * * * * /home/pi/script.py
```

As you can see, two lines have been added to the bottom. The first one set runtime to `@reboot` which means the `/home/pi/script.py` command gets run right away once the Pi is rebooted.

The second line is configured to run the command on *every 10th minute*. You can use sites such as crontab.guru to help you configure the exact timings you wish.

2.6.2 Material to be added

- Supplementary Material
 - Screen cheatsheet (remote connection to Pi)
 - Github cheatsheet
 - Git workflow
 - Git cheatsheet
 - Markdown cheatsheet
 - Pi vs Arduino
 - Other useful links
- Helpful Resources (port from Robotics 1)
 - Github and Git
 - Python
 - * Structuring large Python projects
 - * Writing code: conventions and documentation
 - * Differences between Python 2 and 3
 - * Python tricks (pyclean)
 - * RST (restructuredtext) and sphinx markup
 - * sphinx and read the docs
 - Ground rules for programming

2.6.3 Useful links

Text Editors

- [Pycharm \(for Python\)](#)

- Atom
- Sublime
- Visual Studio Code

File Transfer Software

- WinSCP
- Cyberduck
- MobaXTerm

SSH Applications

- Putty for Windows

Git

- Setting up a Github Account at Imperial
- GitHub Basic Course
- Try Git
- Collaboration Workflow (Short)
- Collaboration Workflow (Comprehensive)
- Github for Desktop
- Github Student Developer Pack
- Sourcetree

Raspberry Pi

- RaspberryPi
- Adafruit Learn
- Make

3D Objects Library

- Thingiverse
- MyMiniFactory

Specific for Imperial College Students

- Getting GitHub Enterprise

CHAPTER 3

Missing Material

Todo: Guidance for managing installs and software on personal computers needs to be written here.

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/de-gizmo/checkouts/latest/docs/source/introduction/computers.rst`, line 5.)

Todo: Review whether Python 2.7 is the correct option nowadays - or whether Python 3.x is more appropriate.

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/de-gizmo/checkouts/latest/docs/source/introduction/python.rst`, line 14.)

Todo: More information will be added here soon.

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/de-gizmo/checkouts/latest/docs/source/raspberrypi/peripherals.rst`, line 238.)

Todo: Using VNC for remote GUI control will be added later.

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/de-gizmo/checkouts/latest/docs/source/raspberrypi/remote-connection.rst`, line 284.)

Todo: Sensors & Actuators index page

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/de-gizmo/checkouts/latest/docs/source/sense-actuate/index.rst`, line 4.)

Todo: More supplementary material will be added here later.

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/de-gizmo/checkouts/latest/docs/source/supplementary/index.rst`, line 9.)