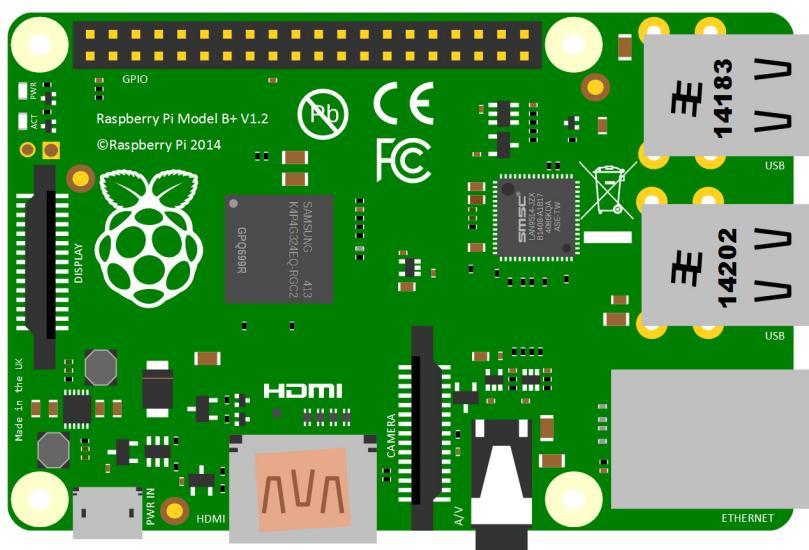


# Raspberry Pi



## Measure Record Explore

Measure the world, record the data  
and display it graphically.

03:00

06:00

09:00

12:00

15:00

18:00

21:00

# Raspberry Pi: Measure, Record, Explore.

Measure the world, record the data and display it graphically.

Malcolm Maclean

This book is for sale at <http://leanpub.com/RPiMRE>

This version was published on 2016-01-22



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2014 - 2016 Malcolm Maclean

## Tweet This Book!

Please help Malcolm Maclean by spreading the word about this book on [Twitter](#)!

The suggested tweet for this book is:

[I just downloaded Raspberry Pi: Measure, Record, Explore.](#)

The suggested hashtag for this book is [#RPiMRE](#).

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

<https://twitter.com/search?q=%23RPiMRE>

## **Also By Malcolm Maclean**

[D3 Tips and Tricks](#)

[Leaflet Tips and Tricks](#)

[Just Enough Linux](#)

[Just Enough Co-Authoring in Leanpub](#)

[Just Enough ownCloud on a Raspberry Pi](#)

[Just Enough Raspberry Pi](#)

[Just Enough Ghost on a Raspberry Pi](#)

*For my Father and my Son  
Merry Christmas*

# Contents

<b>Introduction</b> . . . . .	<b>1</b>
Welcome! . . . . .	1
What are we trying to do? . . . . .	2
Measure . . . . .	2
Record . . . . .	2
Explore . . . . .	2
Who is this book for? . . . . .	3
What tools / equipment will we use? . . . . .	3
Raspberry Pi . . . . .	3
MySQL . . . . .	5
Apache Web Server . . . . .	5
PHP . . . . .	6
Python . . . . .	6
JavaScript . . . . .	7
d3.js . . . . .	7
Sensors . . . . .	7
Where can I get more information? . . . . .	8
raspberrypi.org . . . . .	8
Google+ . . . . .	8
reddit . . . . .	8
Google Groups . . . . .	8
Raspberry Pi Stack Exchange . . . . .	8
adafruit . . . . .	8
<b>Setting up the Raspberry Pi</b> . . . . .	<b>9</b>
Hardware . . . . .	10
The Raspberry Pi . . . . .	10
Case . . . . .	10
SD Card . . . . .	11
Keyboard / Mouse . . . . .	12
Video . . . . .	13
Network . . . . .	14
Power supply . . . . .	15
Operating System . . . . .	16
Welcome to Raspbian . . . . .	16
Sourcing and Setting Up . . . . .	16
Downloading . . . . .	17

## CONTENTS

Writing Raspbian to the SD Card . . . . .	17
Installing Raspbian . . . . .	19
Software Updates . . . . .	19
GUI Desktop . . . . .	21
Static IP Address . . . . .	23
The Netmask . . . . .	23
Distinguish Dynamic from Static . . . . .	24
Setting a Static IP Address on the Raspberry Pi. . . . .	24
Default Gateway . . . . .	24
Edit the <code>interfaces</code> file . . . . .	25
Remote access . . . . .	27
Remote access via TightVNC . . . . .	27
Setting up the Client (Windows) . . . . .	27
Setting up the Server (Raspberry Pi) . . . . .	30
Copying and Pasting between Windows and the Raspberry Pi . . . . .	33
Starting TightVNC at boot on the Pi. . . . .	34
Remote access via SSH . . . . .	39
Setting up the Server (Raspberry Pi) . . . . .	39
Installing SSH on the Raspberry Pi. . . . .	39
Setting up the Client (Windows) . . . . .	39
Setting up a WiFi Network Connection . . . . .	43
Web Server and PHP . . . . .	46
Tweak permissions for easier web file editing . . . . .	47
Database . . . . .	49
MySQL . . . . .	49
phpMyAdmin . . . . .	50
Allow access to the database remotely . . . . .	54
Create users for the database . . . . .	55
Create a database . . . . .	57
Backup the Configured SD Card . . . . .	58
Exploring data with a simple line graph . . . . .	59
The full code . . . . .	59
PHP . . . . .	63
The code . . . . .	63
HTML . . . . .	66
CSS . . . . .	67
JavaScript and d3.js . . . . .	68
Setting up the margins and the graph area. . . . .	70
Getting the Data . . . . .	71
Formatting the Date / Time. . . . .	73
Setting Scales Domains and Ranges . . . . .	74
Setting up the Axes . . . . .	77
Adding data to the line function . . . . .	78
Adding the SVG area. . . . .	78
Actually Drawing Something! . . . . .	80
Wrap Up . . . . .	81

## CONTENTS

<b>Single Temperature Measurement . . . . .</b>	<b>82</b>
Measure . . . . .	82
Hardware required . . . . .	82
Connect . . . . .	82
Test . . . . .	84
Record . . . . .	86
Database preparation . . . . .	86
Record the temperature values . . . . .	88
Code Explanation . . . . .	90
Explore . . . . .	92
The Code . . . . .	92
Different MySQL Selection Options . . . . .	96
<b>Multiple Temperature Measurements . . . . .</b>	<b>99</b>
Measure . . . . .	99
Hardware required . . . . .	99
Connect . . . . .	99
Test . . . . .	100
Record . . . . .	103
Database preparation . . . . .	104
Record the temperature values . . . . .	105
Code Explanation . . . . .	107
Recording data on a regular basis with cron . . . . .	110
Explore . . . . .	111
The Code . . . . .	112
PHP . . . . .	117
JavaScript . . . . .	117
<b>System Information Measurement . . . . .</b>	<b>125</b>
Measure . . . . .	125
Hardware required . . . . .	125
Measured Parameters . . . . .	125
System Load . . . . .	125
Memory Used . . . . .	126
Disk Used . . . . .	127
Raspberry Pi Temperature . . . . .	128
Record . . . . .	129
Database preparation . . . . .	129
Record the system information values . . . . .	131
Code Explanation . . . . .	133
load . . . . .	134
ram . . . . .	134
disk . . . . .	135
temperature . . . . .	136
Main program . . . . .	136
Recording data on a regular basis with cron . . . . .	137

## CONTENTS

Explore . . . . .	137
The Bullet Graph . . . . .	138
The Code . . . . .	140
HTML / JavaScript . . . . .	140
PHP . . . . .	146
<b>Basic GPIO Input Sensors . . . . .</b>	<b>150</b>
Measure . . . . .	152
Hardware required . . . . .	152
The KY003 Hall Effect Sensor . . . . .	152
Connect . . . . .	154
Test . . . . .	155
Record . . . . .	155
Database preparation . . . . .	155
Record the events . . . . .	157
Code Explanation . . . . .	159
Start the code automatically at boot . . . . .	162
Explore . . . . .	164
The Code . . . . .	165
PHP . . . . .	169
CSS (Styles) . . . . .	169
JavaScript . . . . .	170
<b>Pressure and Temperature measurement with the BMP180 . . . . .</b>	<b>174</b>
Measure . . . . .	175
Hardware required . . . . .	175
The BMP180 Sensor . . . . .	175
Connect . . . . .	177
Test . . . . .	178
Record . . . . .	187
Database preparation . . . . .	187
Record the readings . . . . .	188
Code Explanation . . . . .	190
Recording data on a regular basis with cron . . . . .	191
Explore . . . . .	193
The Code . . . . .	193
PHP . . . . .	197
CSS (Styles) . . . . .	198
JavaScript . . . . .	198
Bibliography . . . . .	201
<b>Connecting Analog Sensors to the Raspberry Pi . . . . .</b>	<b>202</b>
Analog and Digital . . . . .	202
Analog . . . . .	202
Digital . . . . .	203
Analog to Digital Conversion (ADC) . . . . .	203
The Sensor . . . . .	204

## CONTENTS

Data Visualization . . . . .	205
Measure . . . . .	206
Hardware required . . . . .	206
The ADS1015 Analog to Digital Converter . . . . .	206
The Light Dependant Resistor (LDR or Photoresistor) Sensor . . . . .	207
Connect . . . . .	209
Test . . . . .	210
Record . . . . .	218
Database preparation . . . . .	218
Record the readings . . . . .	219
Code Explanation . . . . .	221
Recording data on a regular basis with cron . . . . .	223
Explore . . . . .	224
The Code . . . . .	224
PHP . . . . .	228
CSS (Styles) . . . . .	229
JavaScript . . . . .	229
Bibliography . . . . .	232
 Web Scraping . . . . .	233
OK, so what <i>is</i> web scraping? . . . . .	233
Measure . . . . .	235
Hardware required . . . . .	235
Software required . . . . .	235
Let the scraping begin . . . . .	236
Record . . . . .	241
Database preparation . . . . .	241
Record the reader numbers . . . . .	243
Code Explanation . . . . .	245
Recording data on a regular basis with cron . . . . .	246
Explore . . . . .	246
The Code . . . . .	248
Description . . . . .	252
Nesting the data . . . . .	253
Wrangle the data . . . . .	254
Cheating with the domain . . . . .	254
data vs datum . . . . .	256
Setting up the clipPaths . . . . .	256
Clipping and adding the areas . . . . .	258
Draw the lines and the axes . . . . .	261
Adding a bit more to our difference chart . . . . .	262
Add a Y axis label . . . . .	262
Add a title . . . . .	262
Adding the legend . . . . .	263
Link the areas . . . . .	264
The final result . . . . .	265

## CONTENTS

<b>Raspberry Pi Tips and Tricks . . . . .</b>	<b>266</b>
Changing the default keyboard layout . . . . .	266
Changing the default local time . . . . .	267
Access the Pi with a ‘name’ or an IP address . . . . .	268
Transfer files easily to / from the Pi . . . . .	271
Bonus: Edit Files on the Pi from your Desktop Editor . . . . .	274
<b>Hardware . . . . .</b>	<b>277</b>
Raspberry Pi A+ . . . . .	277
USB Port . . . . .	278
Video Out . . . . .	278
USB Power Input Jack . . . . .	279
MicroSD Flash Memory Card Slot . . . . .	279
Stereo and Composite Video Output . . . . .	280
40 Pin Header . . . . .	281
Raspberry Pi B+ . . . . .	282
USB Ports . . . . .	282
Video Out . . . . .	282
Ethernet Network Connection . . . . .	283
USB Power Input Jack . . . . .	283
MicroSD Flash Memory Card Slot . . . . .	284
Stereo and Composite Video Output . . . . .	284
40 Pin Header . . . . .	285
Raspberry Pi B . . . . .	286
USB Ports . . . . .	286
HDMI Video Out . . . . .	287
Composite Video Out . . . . .	287
Ethernet Network Connection . . . . .	288
USB Power Input Jack . . . . .	288
SD Flash Memory Card Slot . . . . .	289
Audio Output . . . . .	290
26 Pin Header . . . . .	290
Cases . . . . .	292
Multicomp MC-RP002-CLR . . . . .	292
Side views . . . . .	293
Fitting the Raspberry Pi . . . . .	294
DIY Open Multi-stack Pi . . . . .	295
Sensors . . . . .	298
DS18B20 Programmable Resolution 1-Wire Digital Thermometer . . . . .	298
Accessories . . . . .	299
VGA to HDMI Adapter . . . . .	299
In-line switch for USB power supply . . . . .	299
Multiple Outlet USB Power Supply . . . . .	300
<b>Linux Command Glossary . . . . .</b>	<b>302</b>
apt-get . . . . .	302

## CONTENTS

apt-get update . . . . .	302
apt-get upgrade . . . . .	302
apt-get install . . . . .	303
cat . . . . .	303
cd . . . . .	304
chmod . . . . .	305
chown . . . . .	306
crontab . . . . .	307
ifconfig . . . . .	308
ls . . . . .	310
modprobe . . . . .	311
sudo . . . . .	312
usermod . . . . .	312
<b>Appendices . . . . .</b>	<b>313</b>
Raspberry Pi Quick Set-up . . . . .	313
Download Raspbian Image . . . . .	313
Writing Raspbian to the SD Card . . . . .	313
Installing Raspbian . . . . .	313
Software Updates . . . . .	313
Static IP Address . . . . .	314
Remote access via TightVNC . . . . .	314
On Windows . . . . .	314
On the Raspberry Pi . . . . .	315
Starting TightVNC at boot . . . . .	316
Copying and Pasting between Windows and the Raspberry Pi via TightVNC	317
Remote access via SSH . . . . .	318
Setting up the Server (Raspberry Pi) . . . . .	318
Setting up the Client (Windows) . . . . .	318
Setting up a WiFi Network Connection . . . . .	319
Web Server and PHP . . . . .	320
Adjust permissions for web files . . . . .	320
Database . . . . .	321
phpMyAdmin . . . . .	321
Allow remote database access . . . . .	322
Add users to the database . . . . .	322
Create a database . . . . .	323
Understanding JavaScript Object Notation (JSON) . . . . .	324

# Introduction

## Welcome!

Hi there. Congratulations on being interested enough in the process of measuring and learning about the world around you to have gotten your hands on this book.

If you haven't guessed already, this will be a journey of discovery for both of us. I have grand plans to 'play' with computers and use them to know a bit more about what is happening in the physical environment. I know that this sort of effort has been done already by others, but I want to go a little farther and provide the ability to capture data, to store it in a flexible way and to interact with it as well.

Ambitious? Perhaps :-). But I'd like to think that if you're reading this, perhaps I managed to make some headway. I dare say that like other books I have written (or are in the process of writing) they will remain a work in progress. They are living documents, open to feedback, comment, expansion, change and improvement. Please feel free to provide your thoughts on ways that I can improve things. Your input would be much appreciated.

You will find that I have typically eschewed a simple "Do this approach" for more of a story telling exercise. This means that some explanations are longer and more flowery than might be to everyone's liking, but there you go, try to be brave :-)

I'm sure most authors try to be as accessible as possible. I'd like to do the same, but be warned... There's a good chance that if you ask me a technical question I may not know the answer. So please be gentle with your emails :-).

Email: d3noobmail+rpi@gmail.com

## What are we trying to do?

Put simply, we are going to measure some aspect of the physical world, store the measured values in a database and then make them available to explore. We will refer to the steps as ‘Measure’, ‘Record’ and ‘Explore’ or M.R.E.



Some of you will be aware of the American military M.R.E. which is a packaged ‘Meal Ready to Eat’. So it seems only appropriate that our Raspberry Pi will be packaged as a M.R.E. system.

### Measure

The measurement process will involve using a means of sensing some aspect of the physical world (such as temperature, pressure, movement, levels) and working out how we can use a minimalist computing element to carry out this task. For the most part, we’ll use a [Raspberry Pi](#)<sup>1</sup> as the measurement, recording and presentation device (although we will look at other options). The Raspberry Pi is an exceptionally small single board computer that has become almost the defacto standard for smaller computing projects and learning.

### Record

Once we have measured something it seems a shame to lose that information. So instead we will store it in a database. We’ll use a [MySQL](#)<sup>2</sup> database and configure it so that the measurement process can store the information directly to it.

### Explore

Once we have all this data it would be a shame not to be able to do something with it, so we will build a simple system for recalling and visualising the data via a web page. With this in place you will be able to browse to your data on your home network.



### Security

The internet is equal parts magical resource for learning and minefield of potential disasters. The intent for this book is to build a system to connect in a home network to explore the world a little and to learn. Our focus will not be to plug every potential vulnerability that the World Wide Web has spawned. Don’t have any illusions that what we will do will be secure in any way. In other words don’t store your financial history or medical records on it and don’t use it to control the electrical connection to your refrigerator. Having said that, don’t live in fear. Paranoia about evil hackers pwn-ing your ‘stuff’ is only going to slow down your learning. Take sensible precautions and focus on the fun!

---

<sup>1</sup><http://www.raspberrypi.org/>

<sup>2</sup><http://www.mysql.com/>

## Who is this book for?

You!

Just by virtue of taking an interest and getting hold of a copy of this book you have demonstrated a desire to learn, to explore and to challenge yourself. That's the most important criteria you will want to have when trying something new. Your experience level will come second place to a desire to learn.

Having said that, it may be useful to be comfortable using the Windows operating system (I'll be using Windows 7 for the set-up of the devices since that would probably classify as (currently) the world's most ubiquitous operating system), you should be aware of Linux as an alternative operating system, but you needn't have tried it before. If you know anything about electronics it will help, but we'll break anything tricky down into bite sized chunks. If you've done any programming before that will be useful, but again, we'll make it easy and spell out what's going on as it comes up. The best thing to remember is that before you learn anything new, it pretty much always appears indistinguishable from magic, but once you start having a play, the mystery quickly falls away.

## What tools / equipment will we use?

To accomplish our goals we are going to use a range of tools and pieces of equipment. They will all be low cost or free. The idea is that the price of the equipment should not be a major impediment to learning how all this goes together. Hopefully this book should also shorten the selection process for working out which parts you will need.

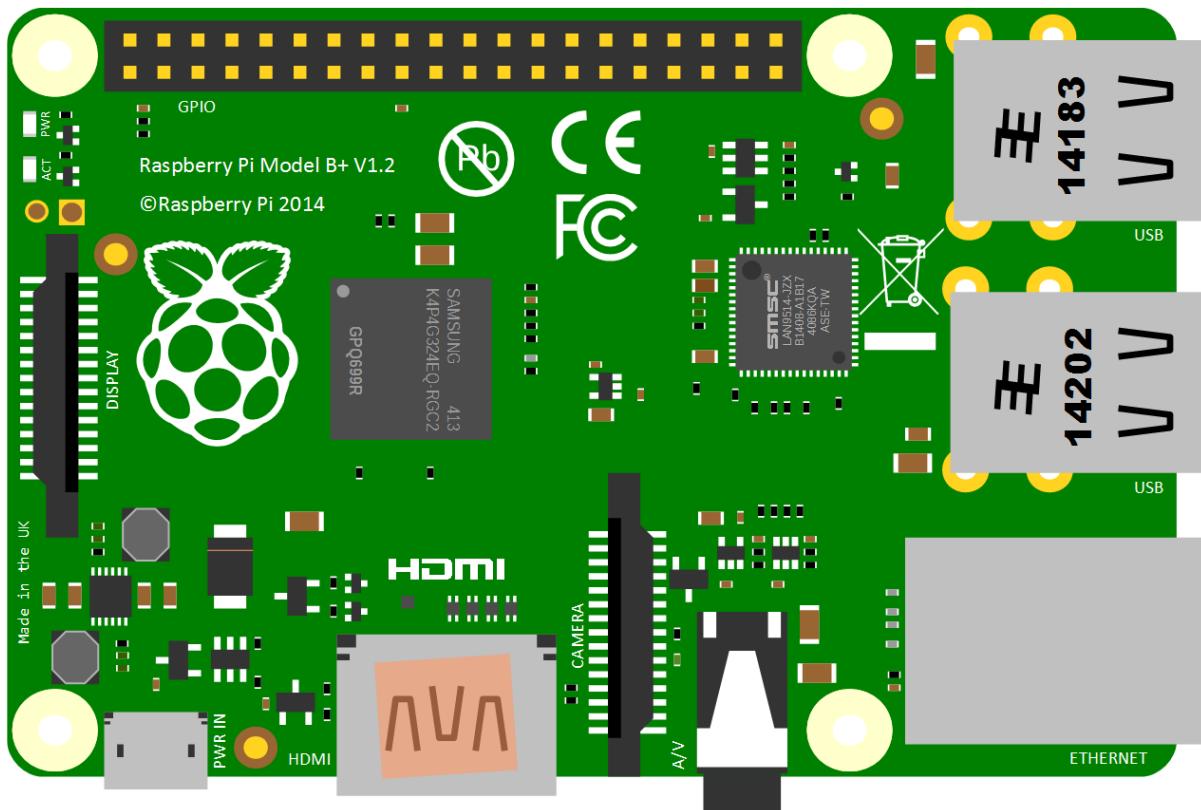
### Raspberry Pi

In the words of the totally awesome [Raspberry Pi<sup>3</sup>](#) foundation;

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It's capable of doing everything you'd expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.

---

<sup>3</sup><http://www.raspberrypi.org/help/what-is-a-raspberry-pi/>



The Raspberry Pi B+ Board

It really is an extraordinary device that is all the more extraordinary for the altruistic effort that brought it into being.

There are (at time of writing) five different models on the market. The A, B, A+, B+ and the ‘2 model B’ (which I’m just going to call the B2). The projects that we’ll follow will typically use either the the B+ or the B2 for no reason other than they offer a good range of USB ports (4), 512 or 1024 MB of RAM, an HDMI video connection, an Ethernet connection and 17 General Purpose Input / Output (GPIO) pins. For all intents and purposes either the B+ or B2 can be used interchangeably for the projects so long as the latest version of the Raspbian operating system is used (or at least one released on or after the 31st of January 2015).

There is a more detailed description in a later chapter where we can examine the specifications more closely, but for the purposes of each project we will describe what is required to know on a case by case basis.

## MySQL



MySQL Logo

MySQL<sup>4</sup> is an Open Source database supported by Oracle.

It would arguably be the world's most popular database and while it has gone through some 'interesting' recent times following its acquisition by Oracle, it remains the standard by which other databases are measured.

## Apache Web Server



Apache Logo

The Apache HTTP Server<sup>5</sup> project has been responsible for the development and maintenance of the most widely used web server on the internet. It is the standard for serving web content.

---

<sup>4</sup><http://www.mysql.com/>

<sup>5</sup><http://httpd.apache.org/>

## PHP



PHP Logo

PHP<sup>6</sup> is a scripting language for the web. That is to say that it is a programming language which is executed when you load web pages and it helps web pages do dynamic things.

PHP is executed remotely on the server that supplies the web page. This might sound a bit redundant, but it's a big deal. This means that the PHP which is executed doesn't form part of the web page, but it can form the web page. The implication here is that the web page you are viewing can be altered by the PHP code that runs on a remote server. This is the dynamic aspect of it.

## Python



Python Logo

Python is a widely used general-purpose programming language. Its design philosophy supports code readability, and its syntax aims to allow programmers to express concepts in fewer lines of code than would be possible in many other languages. The language provides constructs intended to enable clear programs on both a small and large scale.

This has been the most popular language for users of the Raspberry Pi to interface with sensors, so we will use it as a matter of course.

---

<sup>6</sup><http://php.net/>

## JavaScript



JavaScript Logo

JavaScript<sup>7</sup> is what's called a 'scripting language'. It is the code that will be contained inside our web page that will allow us to do some neat graphical representations with our measured data.

## d3.js



d3.js Logo

d3.js<sup>8</sup> (hereafter abridged as D3) is "*a JavaScript library for manipulating documents based on data*".

D3 stands for Data Driven Documents, which seems appropriate for the projects that we will undertake.

It's an Open Source JavaScript library that is very popular as the framework for displaying information in a web browser. I might be slightly biased in using it here as I am also the author of the book [D3 Tips and Tricks<sup>9</sup>](#), which you can download for free from [Leanpub<sup>10</sup>](#).

## Sensors

The act of measuring some physical aspect of the world around us with a computer requires us to find a way of converting a physical change in the environment into an electrical signal of some

---

<sup>7</sup><http://en.wikipedia.org/wiki/JavaScript>

<sup>8</sup><http://d3js.org/>

<sup>9</sup><https://leanpub.com/D3-Tips-and-Tricks>

<sup>10</sup><https://leanpub.com/>

kind. These signals can take many forms and are supplied by sensors which act as the interface between the thing being measured and the computer. As we work through different things we want to measure we will describe each sensor and how it works.

## **Where can I get more information?**

The Raspberry Pi as a concept has provided an extensible and practical framework for introducing people to the wonders of computing in the real world. At the same time there has been a boom of information available for people to use them. The following is a far from exhaustive list of sources, but from my own experience it represents a useful subset of knowledge.

**raspberrypi.org**

**Google+**

**reddit**

**Google Groups**

**Raspberry Pi Stack Exchange**

**adafruit**

# Setting up the Raspberry Pi

While the Raspberry Pi is a capable computer, we still need to install software on it to allow us to gather our data, store it and display it.

The software we will be using is based on the Linux Operating System. If you don't have any familiarity with Linux, then you are about to have an exposure to it. Rest assured that we will take our time and explain things as we go. If you are a bit more confident and know your sudo from your network mask, there is a [Raspberry Pi Quick Set-up](#) section for the software loading process in the appendices.

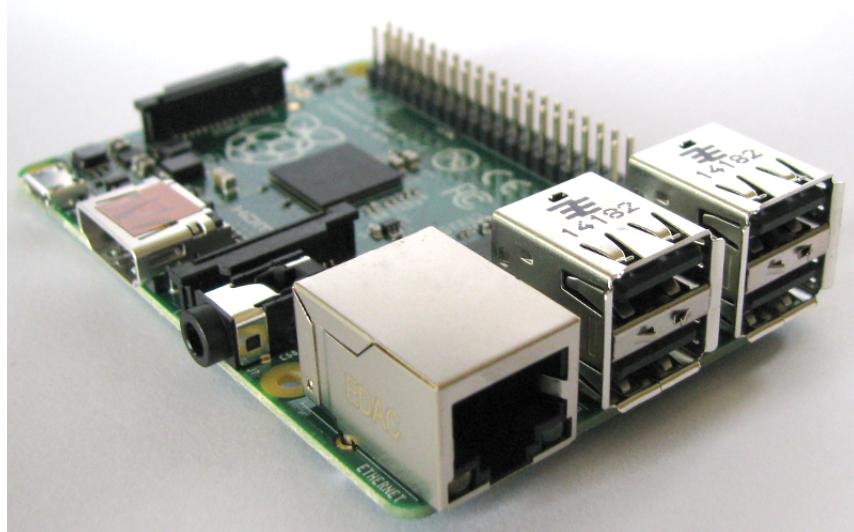
One of the important aspects of the different projects that we are going to work on is that they have a common base. The process in the following sections will form that base, so that with each new measurement technique we take with a different sensor, we will assume that our Raspberry Pi environment has been set up with the following software. This will allow each project / chapter to be approached separately if desired.

## Hardware

To make a start you will require some basic hardware to get you up and running and talking to the Raspberry Pi.

### The Raspberry Pi

I know this is kind of obvious, but you will need a Raspberry Pi :-). As mentioned earlier, we will focus on using either the B+ or B2 model (they are interchangeable for the projects), but since this book is very much a living document I am hopeful that in the future we should be able to make the comparison with other models.



Raspberry Pi B+

The Raspberry Pi has a great range of connection points and we will begin our set-up of the device by connecting quite a number of them to appropriate peripherals.

### Case

Get yourself a simple case to sit the Pi out of the dust and detritus that's floating about. For the purposes of ongoing development I have found that having one that leaves the top side of the Pi exposed (and the connections there accessible) is useful.

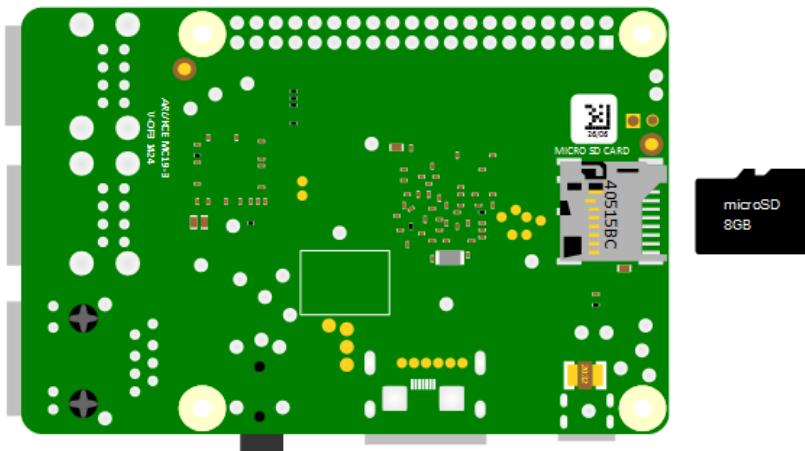
## SD Card

The Raspberry Pi needs to store the Operating System and working files on a micro SD card (actually a micro SD card for the B+ model, but a full size SD card if you're using a B model).



MicroSD Card

The microSD card receptacle is on the rear of the board and is of a 'push-push' type which means that you push the card in to insert it and then to remove it, give it a small push and it will spring out.

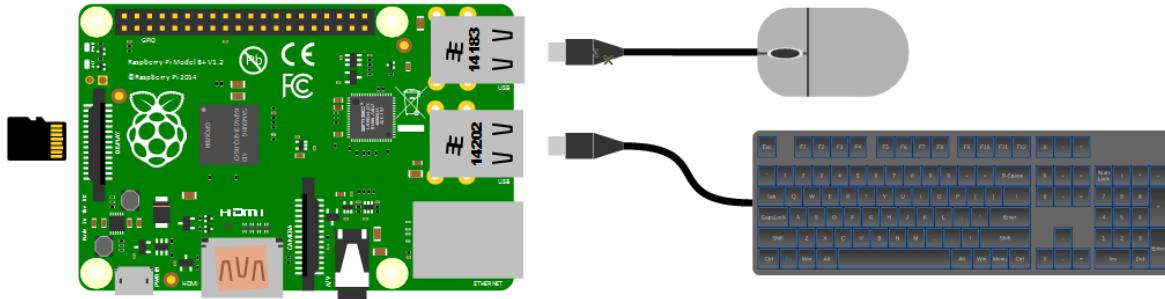


MicroSD Card Positioning

This is the equivalent of a hard drive for a regular computer, but we're going for a minimal effect. We will want to use a minimum of an 8GB card (smaller is possible, but 8 is recommended). Also try to select a higher speed card if possible (class 10 or similar) as it is anticipated that this should speed things up a bit.

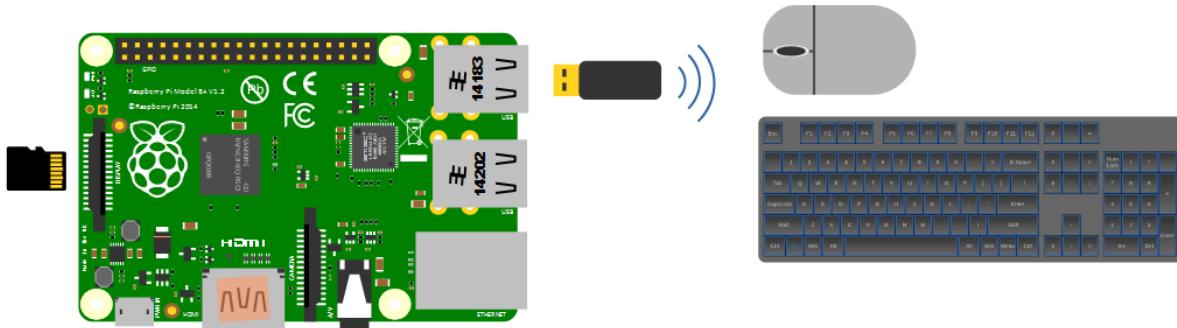
## Keyboard / Mouse

While we will be making the effort to access our system via a remote computer, you will need a keyboard and a mouse for the initial set-up. Because the B+ and B2 models of the Pi have 4 x USB ports, there is plenty of space for you to connect wired USB devices.



Wired Keyboard and Mouse

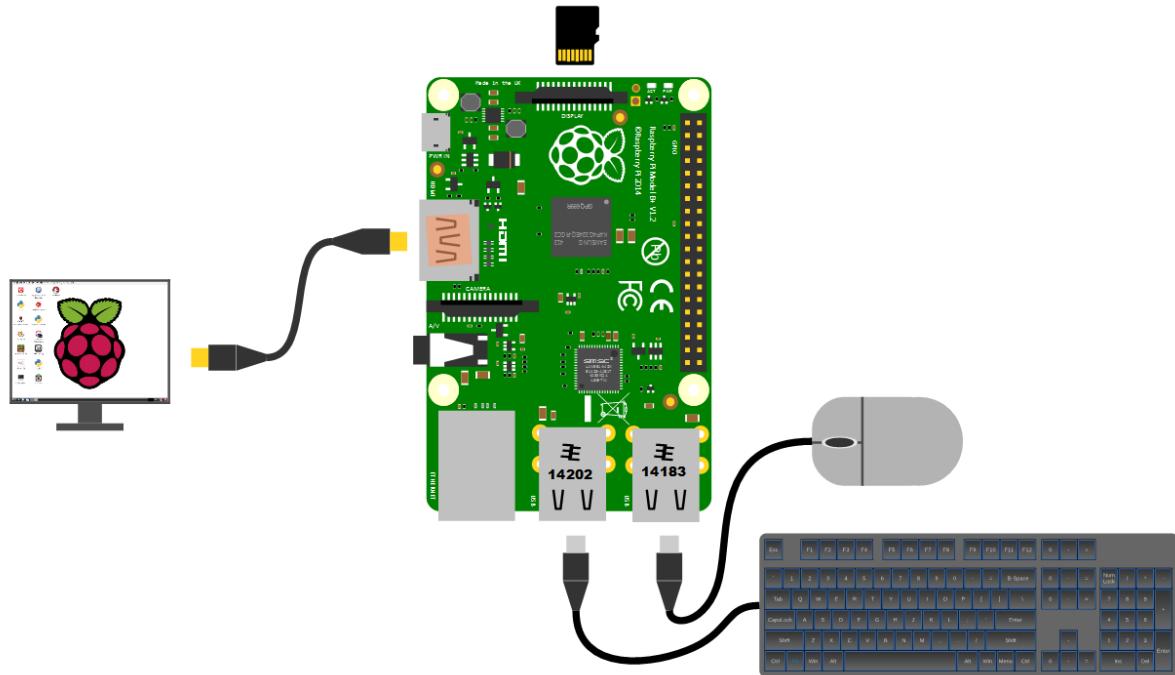
A wireless combination would most likely be recognised without any problem and would only take up a single USB port, but as we will build towards a remote capacity for using the Pi, the nicety of a wireless connection is not strictly required.



Wireless Keyboard and Mouse

## Video

The Raspberry Pi comes with an HDMI port ready to go which means that any monitor or TV with an HDMI connection should be able to connect easily.



HDMI Connected Monitor

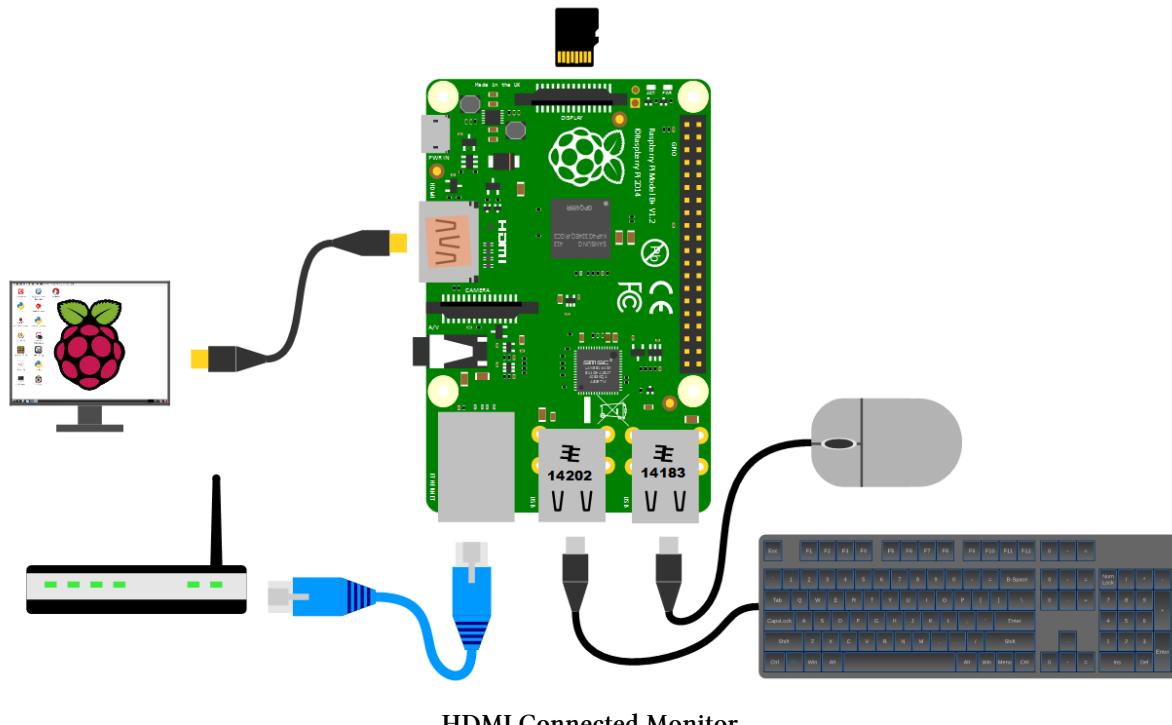
Because this is kind of a hobby thing you might want to consider utilising an older computer monitor with a DVI or 15 pin D connector. If you want to go this way you will need an adapter to convert the connection.



VGA to HDMI Adapter

## Network

The B+ and B2 models of the Raspberry Pi have a standard RJ45 network connector on the board ready to go. In a domestic installation this is most likely easiest to connect into a home ADSL modem or router.

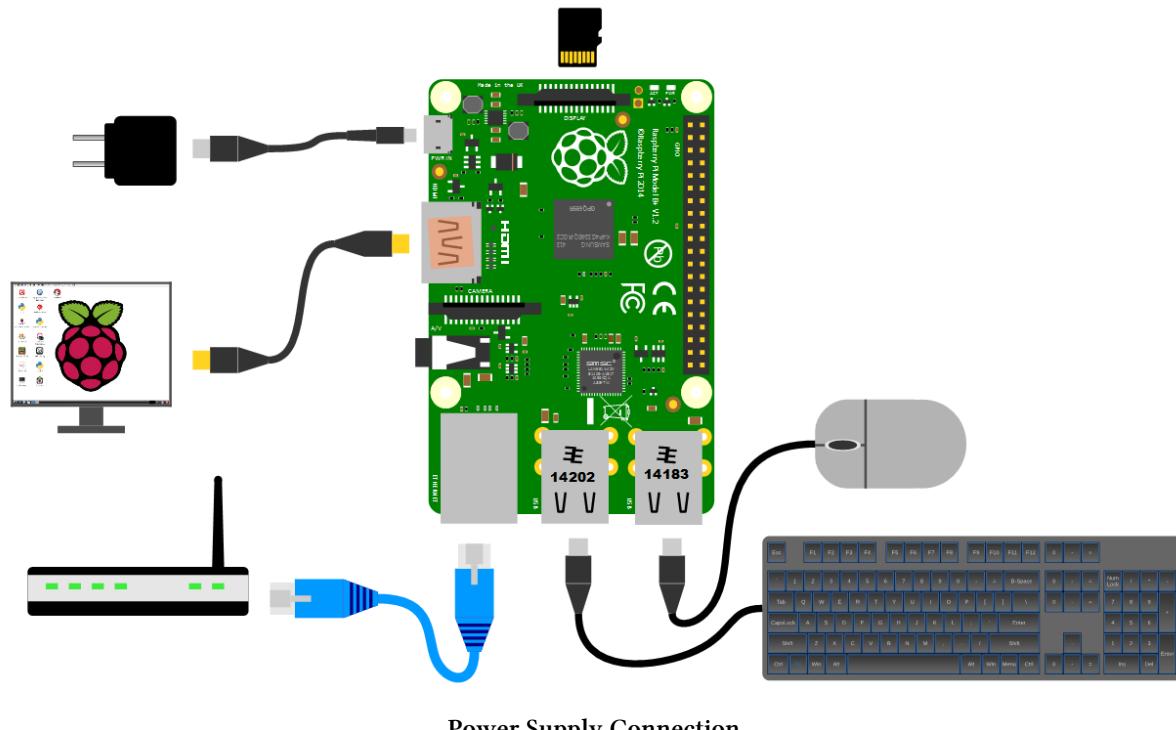


HDMI Connected Monitor

This ‘hard-wired’ connection is great for a simple start, but we will ultimately work towards a wireless solution.

## Power supply

The pi can be powered up in a few ways. The simplest is to use the microUSB port to connect from a standard USB charging cable. You probably have a few around the house already for phones or tablets.



Power Supply Connection

It is worth knowing that depending on what use we wish to put our Raspberry Pi to we might want to pay a certain amount of attention to the amount of current that our power supply can supply. The B+ model will function adequately with a 700mA supply, but if we want to look towards using multiple wireless devices or supplying sensors that demand power from the Pi, we should consider a supply that is capable of an output up to 2A.

## Operating System

As mentioned we will be using the Linux Operating system on our Raspberry Pi. More specifically we will be installing a ‘distribution’ (version) of Linux called Raspbian<sup>11</sup>.

Linux<sup>12</sup> is a computer operating system that is can be distributed as free and open-source software<sup>13</sup>. The defining component of Linux is the Linux kernel, an operating system kernel first released on 5 October 1991 by Linus Torvalds.

Linux was originally developed as a free operating system for Intel x86-based personal computers. It has since been made available to a huge range of computer hardware platforms and is a leading operating system on servers, mainframe computers and supercomputers. Linux also runs on embedded systems, which are devices whose operating system is typically built into the firmware and is highly tailored to the system; this includes mobile phones, tablet computers, network routers, facility automation controls, televisions and video game consoles. Android, the most widely used operating system for tablets and smart-phones, is built on top of the Linux kernel. In our case we will be using a version of Linux that is assembled to run on the ARMv6 CPU<sup>14</sup> used in the Raspberry Pi.

The development of Linux is one of the most prominent examples of free and open-source software collaboration. Typically, Linux is packaged in a form known as a Linux distribution, for both desktop and server use. Popular mainstream Linux distributions include Debian, Ubuntu and the commercial Red Hat Enterprise Linux. Linux distributions include the Linux kernel, supporting utilities and libraries and usually a large amount of application software to carry out the distribution’s intended use.

A distribution intended to run as a server may omit all graphical desktop environments from the standard install, and instead include other software to set up and operate a solution stack such as LAMP (Linux, Apache, MySQL and PHP). Because Linux is freely re-distributable, anyone may create a distribution for any intended use.

## Welcome to Raspbian

The Raspbian Linux distribution is based on Debian Linux. You might well be asking if that matters a great deal. Well, it kind of does since Debian is such a widely used distribution that it allows Raspbian users to leverage a huge quantity of community based experience in using and configuring the software.

## Sourcing and Setting Up

On your desktop machine you are going to download the Raspbian software and write it onto the SD card. This will then be installed into the Raspberry Pi.

---

<sup>11</sup><http://www.raspbian.org/>

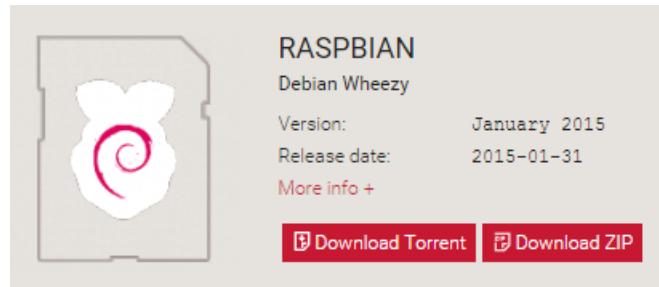
<sup>12</sup><http://en.wikipedia.org/wiki/Linux>

<sup>13</sup>[http://en.wikipedia.org/wiki/Free\\_and\\_open-source\\_software](http://en.wikipedia.org/wiki/Free_and_open-source_software)

<sup>14</sup>[http://en.wikipedia.org/wiki/ARM\\_architecture](http://en.wikipedia.org/wiki/ARM_architecture)

## Downloading

The best place to source the latest version of the Raspbian Operating System is to go to the raspberrypi.org page; <http://www.raspberrypi.org/downloads/>.



Raspbian Download

You can download via bit torrent or directly as a zip file, but whatever the method you should eventually be left with an 'img' file for Raspbian.

To ensure that the projects we work on can be used with either the B+ or B2 models we need to make sure that the version of Raspbian we download is from 2015-01-13 or later. Earlier downloads will not support the more modern CPU of the B2.



2015-01-31-raspbian

Image File

## Writing Raspbian to the SD Card

Once we have the image file we need to get it onto our SD card.

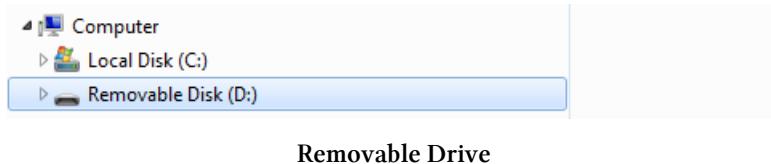
We will work through an example using Windows 7, but for guidance on other options (Linux or Mac OS) raspberrypi.org has some great descriptions of the processes [here<sup>15</sup>](#).

We will use the Open Source utility Win32DiskImager which is available from [sourceforge<sup>16</sup>](#). This program allows us to install our Raspbian disk image onto our SD card. Download and install Win32DiskImager.

<sup>15</sup><http://www.raspberrypi.org/documentation/installation/installing-images/README.md>

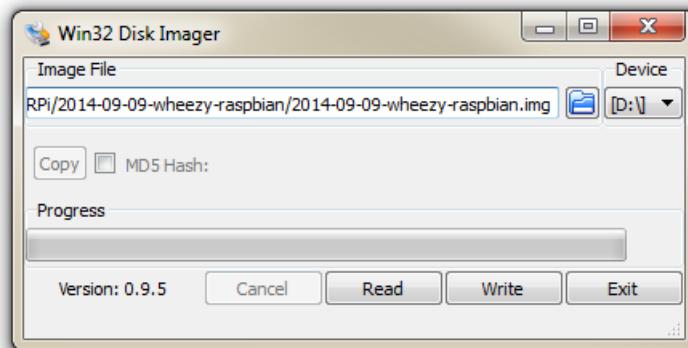
<sup>16</sup><http://sourceforge.net/projects/win32diskimager/>

You will need an SD card reader capable of accepting your micro SD card (you may require an adapter or have a reader built into your desktop or laptop). Place the card in the reader and you should see a drive letter appear in Windows Explorer that corresponds with the SD card.



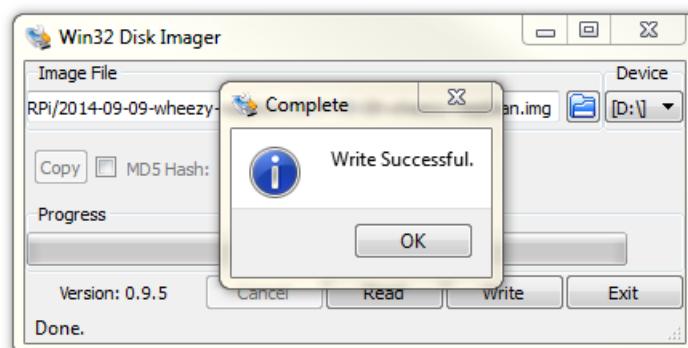
In the screenshot above the removable drive has the letter 'D'. The letter that appears on your system may be different. It is important that you use the correct drive letter for YOUR system.

Start the Win32 Disk Imager program.



Win32 Disk Imager

Select the correct drive letter for your SD card (make sure it's the right one) and the Raspbian disk image that you downloaded. Then select 'Write' and the disk imager will write the image to the SD card. It should only take about 3-4 minutes with a class 10 SD card.



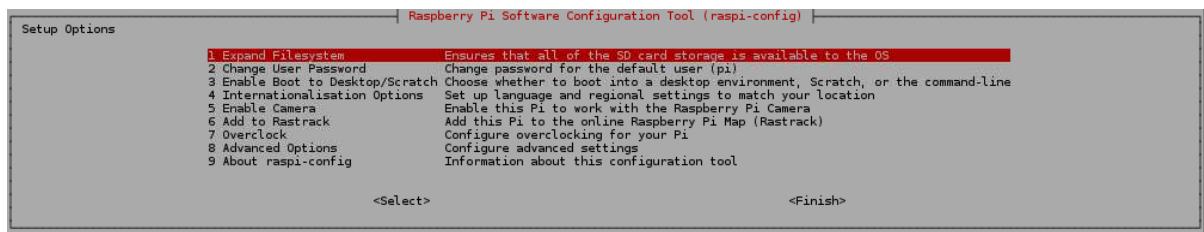
Win32 Disk Imager

Once the process is finished exit the disk imager and eject the card from the computer and you're done.

## Installing Raspbian

Make sure that you've completed the previous section and have a Raspbian disk image written to a micro SD card. Insert the SD card into the slot on the Raspberry Pi and turn on the power.

You will see a range of information scrolling up the screen before eventually being presented with the Raspberry Pi Software Configuration Tool.



Raspberry Pi Software Configuration Tool

Using this tool you can first ensure that all of the SD card storage is available to the Operating System. Once this has been completed lets leave the other settings where they are for the moment and select finish. This will allow you reboot the Pi and take advantage of the full capacity of the SD card.

Once the reboot is complete you will be presented with the console prompt to log on;

```
Raspbian GNU/Linux 7 raspberrypi tty1
```

```
raspberrypi login:
```

The default username and password is:

Username: pi

Password: raspberry

Enter the username and password.

Congratulations, you have a working Raspberry Pi and are ready to start getting into the thick of things!

Firstly we'll do a bit of house keeping.

## Software Updates

The first thing we'll do is make sure that we have the latest software for our system. This is a useful thing to do as it allows any additional improvements to the software you will be using to be enhanced or security of the operating system to be improved. This is probably a good time to mention that you will need to have an Internet connection available.

Type in the following line which will find the latest lists of available software;

```
sudo apt-get update
```

You should see a list of text scroll up while the Pi is downloading the latest information.

Then we want to upgrade our software to latest versions from those lists using;

```
sudo apt-get upgrade
```

The Pi should tell you the lists of packages that it has identified as suitable for an upgrade and along with the amount of data that will be downloaded and the space that will be used on the system. It will then ask you to confirm that you want to go ahead. Tell it 'Y' and we will see another list of details as it heads off downloading software and installing it.

(The sudo portion of the command makes sure that you will have the permission required to run the apt-get process. For more information on the sudo command check out the Glossary [here](#). For more on the apt-get update/upgrade commands see [here](#))

## GUI Desktop

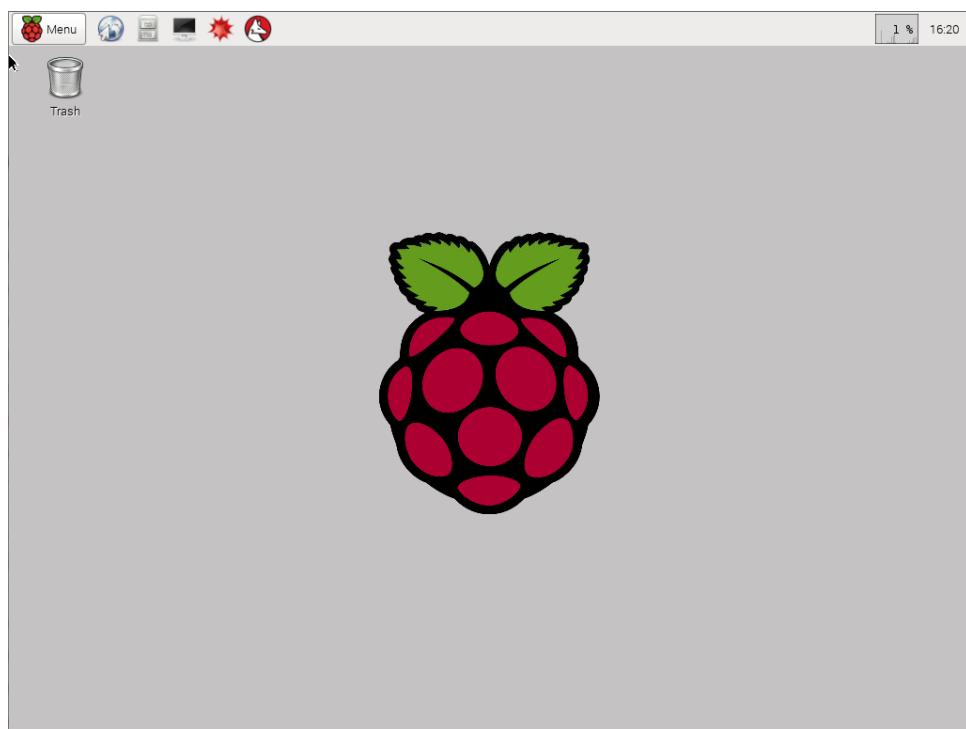
At this point you should have found yourself staring at a screen full of text and successfully logged on to your Raspberry Pi.

You are currently working on the ‘Command Line’ or the ‘CLI’ (Command Line Interface). This is an environment that a great number of Linux users feel comfortable in and from here they are able to operate the computer in ways that can sometimes look like magic. Brace yourself... We are going to work on the command line for quite a bit while working on the Raspberry Pi. This may well be unfamiliar territory for a lot of people and to soften the blow we will also carry out a lot of our work in a Graphical User Interface (GUI).

We can see what we will be using by running the command (just type it in and press return);

```
startx
```

The command `startx` launches the ‘X’ session, which is to say the basic framework for a GUI environment: drawing and moving windows on the display device and interacting with a mouse and keyboard. The Raspbian distribution we are using has a desktop already set up with a range of programs ready to go that can be accessed from the menu button.



Raspbian Desktop

Running a GUI environment is a burden to the computer. It takes a certain degree of computing effort to maintain the graphical interface, so as a matter of course we will only use the `startx` command when we are wanting to interact with the computer via the desktop.

The work that we'll be doing with the computer can be carried out in this environment without problem, but it should be envisaged that the Raspberry Pi will be tucked away somewhere out of sight and in the ideal world we wouldn't need to be directly connected to it to interact with it. The following section describes how to configure our Pi and our desktop Windows computer so that we can remotely access the Raspberry Pi and it won't require having a keyboard, mouse and video screen connected. Be aware however that we are going to raise the bar slightly higher in terms of computing knowledge.

## Static IP Address

As noted in the previous section, enabling remote access requires that we begin to stretch ourselves slightly. In particular we will want to assign our Raspberry Pi a static IP address.

An Internet Protocol address (IP address) is a numerical label assigned to each device (e.g., computer, printer) participating in a computer network that uses the Internet Protocol for communication.

There is a strong likelihood that our Raspberry Pi already has an IP address and it should appear a few lines above the ‘login’ prompt when you first boot up;

```
My IP address is 10.1.1.25
```

```
Raspbian GNU/Linux 7 raspberrypi tty1
```

```
raspberrypi login:
```

In this example the IP address 10.1.1.25 belongs to the Raspberry Pi.

This address will probably be a ‘dynamic’ IP address and could change each time the Pi is booted. For the purposes of using the Raspberry Pi as a web platform, database and with remote access we need to set a fixed IP address.

This description of setting up a static IP address makes the assumption that we have a device running on the network that is assigning IP addresses as required. This sounds like kind of a big deal, but in fact it is a very common service to be running on even a small home network and it will be running on the ADSL modem or similar. This function is run as a service called **DHCP<sup>17</sup>** (Dynamic Host Configuration Protocol). You will need to have access to this device for the purposes of knowing what the allowable ranges are for a static IP address. The most likely place to find a DHCP service running in a normal domestic situation would be an an ADSL modem or router.

## The Netmask

A common feature for home modems and routers that run DHCP devices is to allow the user to set up the range of allowable network addresses that can exist on the network. At a higher level you should be able to set a ‘netmask’ which will do the job for you. A netmask looks similar to an IP address, but it allows you to specify the range of addresses for ‘hosts’ (in our case computers) that can be connected to the network.

A very common netmask is 255.255.255.0 which means that the network in question can have any one of the combinations where the final number in the IP address varies. In other words with a netmask of 255.255.255.0 the IP addresses available for devices on the network 10.1.1.x range from 10.1.1.0 to 10.1.1.255 or in other words any one of 256 unique addresses.

---

<sup>17</sup>[http://en.wikipedia.org/wiki/Dynamic\\_Host\\_Configuration\\_Protocol](http://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol)

## Distinguish Dynamic from Static

The other service that our DHCP server will allow is the setting of a range of addresses that can be assigned dynamically. In other words we will be able to declare that the range from 10.1.1.20 to 10.1.1.255 can be dynamically assigned which leaves 10.1.1.0 to 10.1.1.19 which can be set as static addresses.

You might also be able to reserve an IP address on your modem / router. To do this you will need to know what the MAC (or hardware address) of the Raspberry Pi is. To find the hardware address on the Raspberry Pi type;

```
ifconfig -a
```

(For more information on the `ifconfig` command check out the [Glossary](#))

This will produce an output which will look a little like the following;

```
eth0 Link encap:Ethernet HWaddr 00:08:C7:1B:8C:02
      inet addr:10.1.1.26 Bcast:10.1.1.255 Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
            RX packets:53 errors:0 dropped:0 overruns:0 frame:0
            TX packets:44 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:4911 (4.7 KiB) TX bytes:4792 (4.6 KiB)
```

The figures `00:08:C7:1B:8C:02` are the Hardware or MAC address.

Because there are a huge range of different DHCP servers being run on different home networks, I will have to leave you with those descriptions and the advice to consult your devices manual to help you find an IP address that can be assigned as a static address. Make sure that the assigned number has not already been taken by another device. Hopefully we would hold a list of any devices which have static addresses so that our Pi's address does not clash with any other device.



Be aware that if you don't have a section of your IP address range set aside for static addresses you run the risk of having the DHCP service unwittingly assign a device that wants a dynamic address with the same value that you have already assigned for your Raspberry Pi. Such a conflict is not a good thing.

For the sake of the upcoming projects we will assume that the address 10.1.1.8 is available.

## Setting a Static IP Address on the Raspberry Pi.

### Default Gateway

Before we start configuring we will need to find out what the default gateway is for our network. A default gateway is an IP address that a device will use when it is asked to go to an address

that it doesn't immediately recognise. This would most commonly occur when a computer on a home network wants to contact a computer on the Internet. The default gateway is therefore typically the address of the modem on your home network.

We can check to find out what our default gateway is from Windows by going to the command prompt (Start > Accessories > Command Prompt) and typing;

```
ipconfig
```

This should present a range of information including a section that looks a little like the following;

Ethernet adapter Local Area Connection:

```
IPv4 Address . . . . . : 10.1.1.15
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 10.1.1.1
```

The default gateway is therefore '10.1.1.1'.

### Edit the `interfaces` file

On the Raspberry Pi at the command line we are going to start up a text editor and edit the file that holds the configuration details for the network connections.

The file is `/etc/network/interfaces`. That is to say it's the `interfaces` file which is in the `network` directory which is in the `etc` directory which is in the root (`/`) directory.

To edit this file we are going to type in the following command;

```
sudo nano /etc/network/interfaces
```



The `sudo` portion of the command makes sure that you will have the permission required to edit the `interfaces` file, `nano` is the name of the text editor and `/etc/network/interfaces` is telling the computer which file to edit.

The `nano`<sup>18</sup> file editor will start and show the contents of the `interfaces` file which should look a little like the following;

---

<sup>18</sup><http://www.nano-editor.org/>

```
auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

We are going to change the line that tells the network interface to use DHCP (`iface eth0 inet dhcp`) to use our static address that we decided on earlier (`10.1.1.8`) along with information on the netmask to use and the default gateway. So replace the line...

```
iface eth0 inet dhcp
```

... with the following lines (and don't forget to put YOUR address, netmask and gateway in the file, not necessarily the ones below);

```
iface eth0 inet static
address 10.1.1.8
netmask 255.255.255.0
gateway 10.1.1.1
```

Once you have finished press `ctrl-x` to tell nano you're finished and it will prompt you to confirm saving the file. Check your changes over and then press '`y`' to save the file (if it's correct). It will then prompt you for the file-name to save the file as. Press `return` to accept the default of the current name and you're done!

To allow the changes to become operative we can type in;

```
sudo reboot
```

This will reboot the Raspberry Pi and we should see the (by now familiar) scroll of text and when it finishes rebooting you should see;

```
My IP address is 10.1.1.8
```

```
Raspbian GNU/Linux 7 raspberrypi tty1
```

```
raspberrypi login:
```

Which tells us that the changes have been successful (bearing in mind that the IP address above should be the one you have chosen, not necessarily the one we have been using as an example).

## Remote access

To allow us to work on our Raspberry Pi from our normal desktop we will give ourselves the ability to connect to the Pi from another computer. This will mean that we don't need to have the keyboard / mouse or video connected to the Raspberry Pi and we can physically place it somewhere else and still work on it without problem. This process is called 'remotely accessing' our computer .

To do this we need to install an application on our windows desktop which will act as a 'client' in the process and software on our Raspberry Pi to act as the 'server'. There is a couple of different ways that we can accomplish this task. One way is to give us access to the Pi GUI from a remote computer (so you can use the Raspberry Pi desktop in the same way that we did with the `startx` command earlier) using a program called TightVNC and the other way is to get access to the command line (where all we do is type in our commands (like when we first log into the Pi)) via what's called SSH access.



You don't need to install both of these methods of remote access (or either if you want to keep using the Pi from its own keyboard, mouse and screen) but using one or the other would be a neat thing to allow you to put the Raspberry Pi in a location completely separate from your immediate location.

Which you choose to use depends on how you feel about using the device. If you're more comfortable with a GUI environment, then TightVNC will be the solution. This has the disadvantage of using more computing resources on the Raspberry Pi so if you are considering working it fairly hard, then SSH access may be a better option.

## Remote access via TightVNC

The software we will install is called [TightVNC<sup>19</sup>](#). It is free for personal and commercial use and implements a service called [Virtual Network Computing<sup>20</sup>](#). We need to set up instances of it on the client (the Windows desktop machine) and the server (the Raspberry Pi).



### The Client-Server model

The 'client-server' model of computing is a very common term. It refers to a distribution of tasks or workload between computers to allow an application or service to operate. In this case the provider of the service (the server) is the Raspberry Pi and the user of the service (the client) is the Windows machine.

### Setting up the Client (Windows)

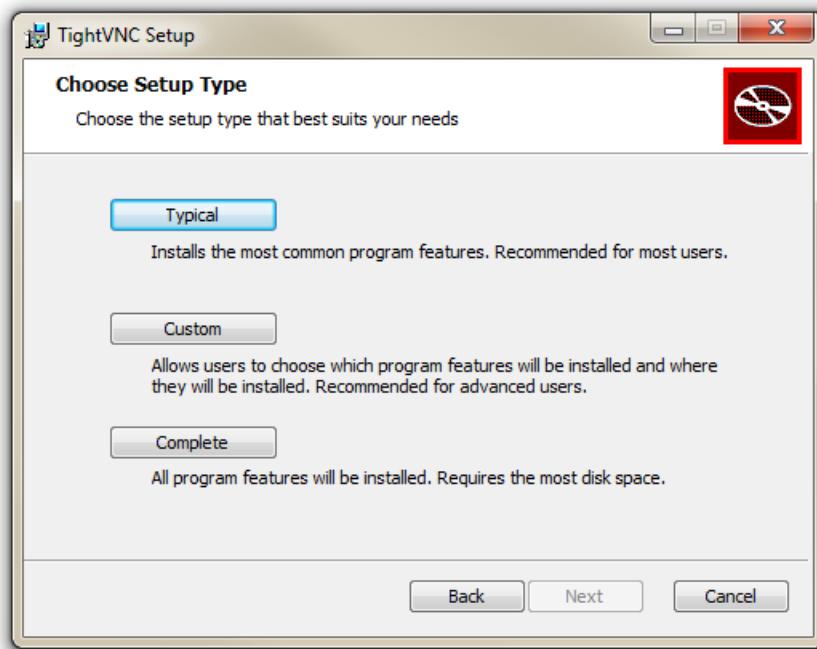
To install TightVNC for windows, go to the [downloads page<sup>21</sup>](#) and select the appropriate version for your operating system.

<sup>19</sup><http://www.tightvnc.com/>

<sup>20</sup>[http://en.wikipedia.org/wiki/Virtual\\_Network\\_Computing](http://en.wikipedia.org/wiki/Virtual_Network_Computing)

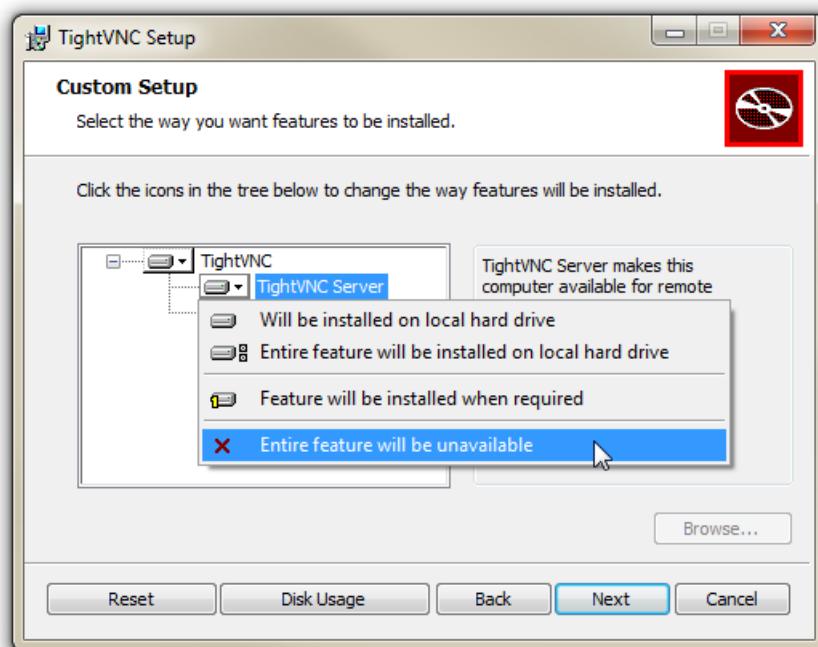
<sup>21</sup><http://www.tightvnc.com/download.php>

Work through the installation process answering all the questions until you get to the screen asking what set-up type to choose.



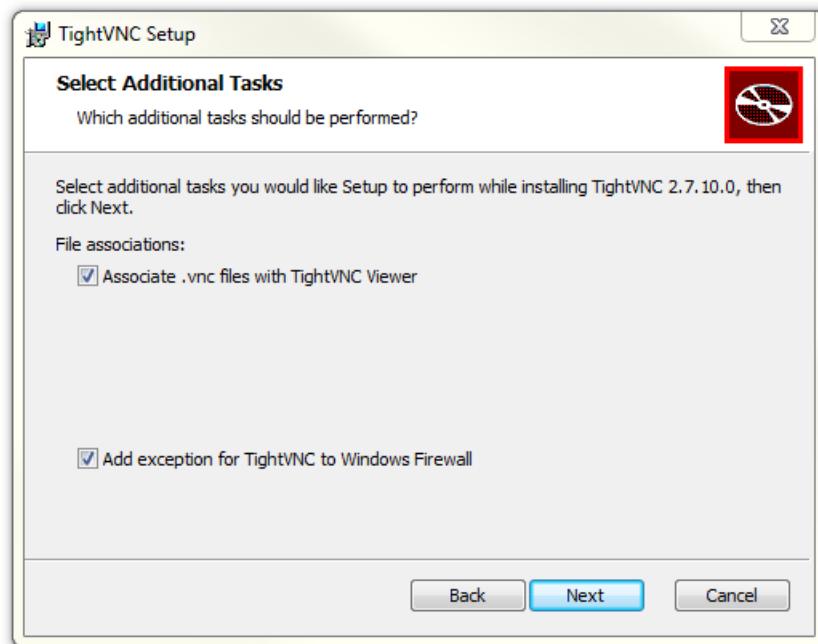
TightVNC Set-up

We only want to install the viewer for the software (since we don't want to make our Windows desktop available to other computers on the network). So click on 'Custom' and then click on the 'TightVNC Server' drop-down and select 'Entire feature will be unavailable'. This will prevent the server side of the software being installed on our machine (thanks to Graham Travener for the hint to disable this), then select 'Next'.



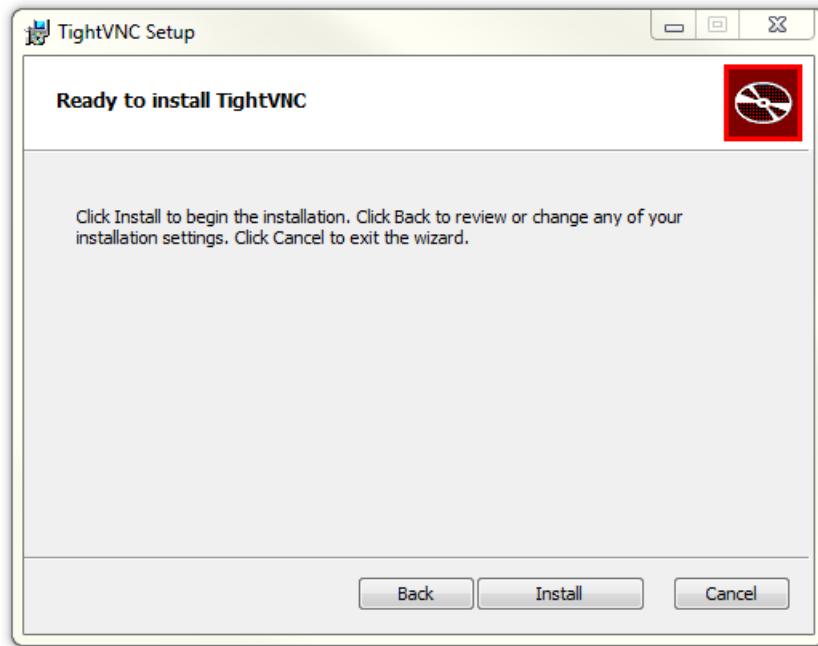
### TightVNC Viewer

The 'Select Additional Tasks' selections can be left at their defaults.



### TightVNC Additional Tasks

Then click on 'Install'.



### TightVNC Install

Click on ‘Finish’ after a short install period and you should be done. You can now find ‘TightVNC Viewer’ from the start menu (but don’t bother running it yet as we still need to set up the Raspberry Pi!).

## Setting up the Server (Raspberry Pi)

We’ll approach the process of installing the TightVNC server on the Raspberry Pi in two stages. In the first stage we’ll install the software, run it and test it. In the second stage we’ll configure it so that it starts automatically when the Raspberry Pi boots up which means that we can work remotely from that point.



Be Warned. This form of connection cannot be regarded as secure enough to connect via the Internet. We’re only using it for the convenience and because we should arguably be more interested in learning about using computers on a home network than being worried about whether or not we will be ‘hacked’. Make no expectations of security for this connection or the data on the Raspberry Pi, but don’t let that stop you using it.

Installing software on the Raspberry Pi is a pretty easy task. All you need to do is from the command line, type;

```
sudo apt-get install tightvncserver
```

You will recall that the `sudo` portion of the command makes sure that you will have the correct permissions (in this case to run a command). The command that is run is one of the family of `apt-get` commands, which deal with package management. The `install` part of the command tells

the apt-get program to not just get the software, but to install it as well. Lastly, `tightvncserver` is the application that will be installed. For more information on the `apt-get` command, see the [Glossary](#).

The Raspberry Pi may ask you if you want to continue once it's done some checks that it can get the software (and any other software that it might be dependant on). Just answer 'y' if prompted.

After a short scrolling of messages, `tightvncserver` should be installed!

Now we can run the program by typing in;

```
tightvncserver
```

You will be prompted to enter a password that we will use on our Windows client software to authenticate that we are the right people connecting. Feel free to enter an appropriate password (believe it or not in this crazy security conscious world, there is a maximum length of password of 8 characters).

You will be asked if you want to have a 'view-only' password that would allow a client to look at, but not interact with the remote desktop (in this case I wouldn't bother).



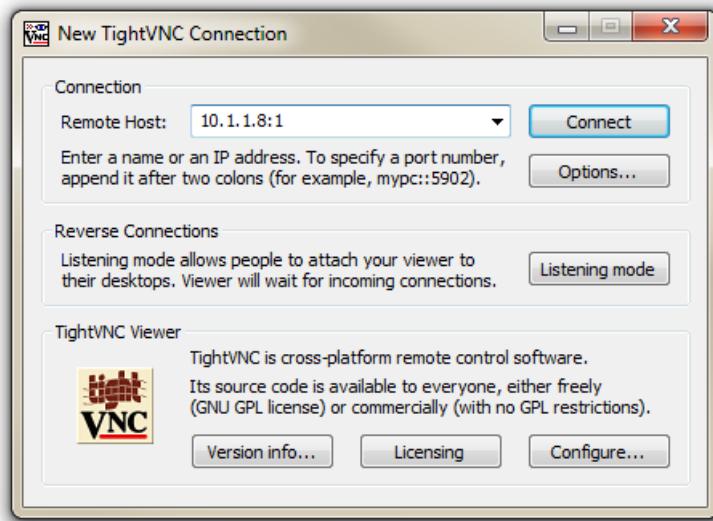
You will be asked for this information the first time you run `tightvncserver`, but from then on it will simply use your initial settings (so remember the password).

The software will then assign us a desktop and print out a couple of messages. One that we will need to note will say something like;

```
New 'X' desktop is raspberrypi:1
```

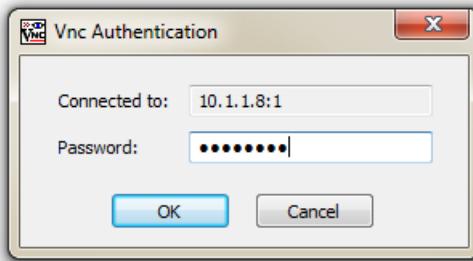
The :1 informs us of the number of the desktop session that we will be looking at (since you can have more than one).

Now on the Windows desktop, start the TightVNC Viewer program. We will see a dialogue box asking which remote host we want to connect to. In this box we will put the IP address of our Raspberry Pi followed by a colon (:) and the number of the desktop that was allocated in the `tightvncserver` program (`10.1.1.8:1`).



TightVNC Start

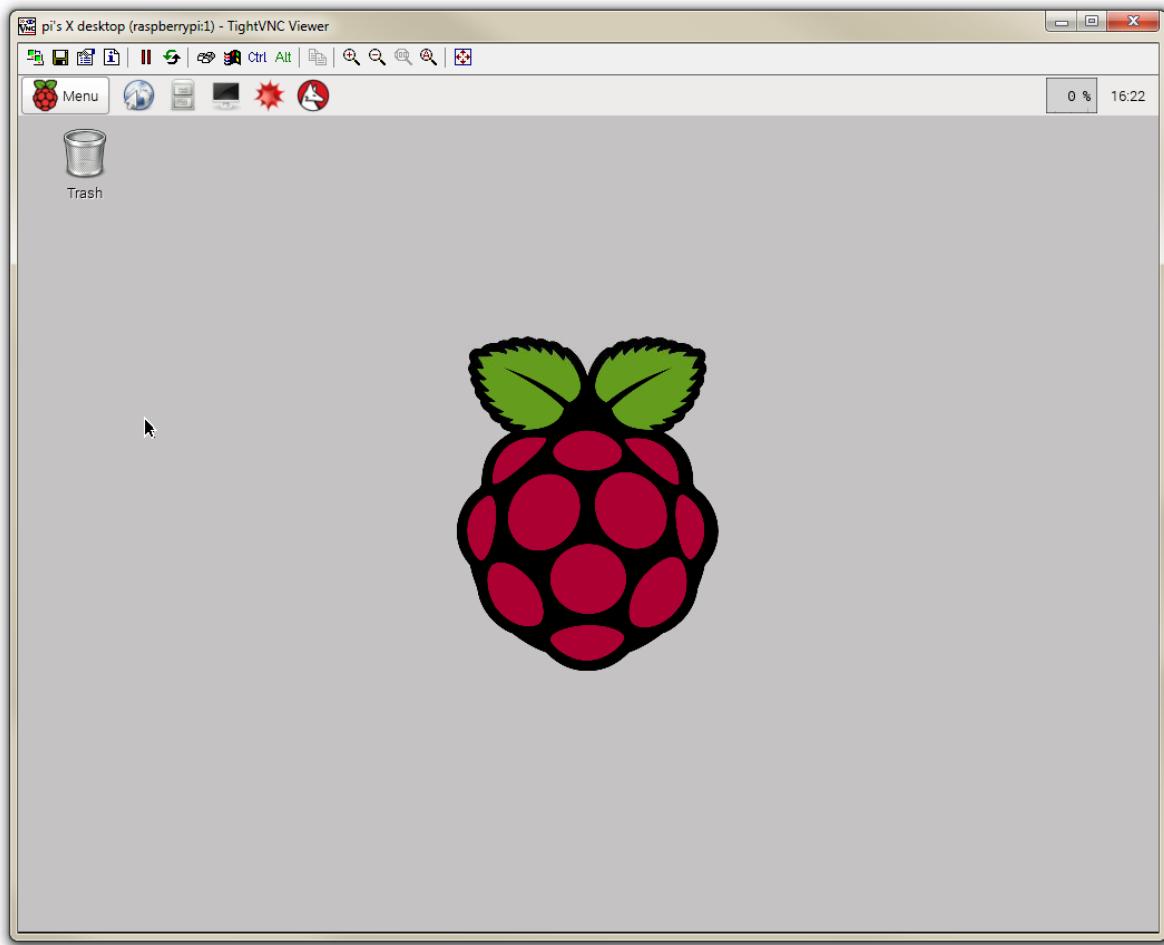
We will be prompted for the password that we set to access the remote desktop;



TightVNC Password

In theory we will then be connected to a remote desktop view of the Raspberry Pi from our Windows desktop.

Take a moment to interact with the connection and confirm that everything is working as anticipated.



TightVNC Desktop

## Copying and Pasting between Windows and the Raspberry Pi

Remotely accessing the Raspberry Pi is a great thing to be able to do, but to make the experience even more useful we need to have the ability to copy and paste between the Windows environment and the Raspberry Pi.

We can certainly survive without this feature, but being able to carry out research on a more powerful machine and then copy-paste code from one to the other is a real advantage.

On the raspberry Pi, first we have to install ‘[autocutsel](#)<sup>22</sup>’ as follows;

```
sudo apt-get install autocutsel
```

Then we need to edit the ‘xstartup’ file as follows;

---

<sup>22</sup><http://www.nongnu.org/autocutsel/>

```
nano /home/pi/.vnc/xstartup
```

... and add in the line autocutsel -fork to start it when the graphical display starts;

```
#!/bin/sh

xrdb $HOME/.Xresources
xsetroot -solid grey
autocutsel -fork
#x-terminal-emulator -geometry 80x24+10+10 -ls -title "$VNCDESKTOP Desktop" &
#x-window-manager &
# Fix to make GNOME work
export XKL_XMODMAP_DISABLE=1
/etc/X11/Xsession
```

Make sure that you place the autocutsel -fork in the position indicated in the example above as otherwise it will not work as desired.

All that remains is to reboot the Raspberry Pi for the changes to take effect.

```
sudo reboot
```

## Starting TightVNC at boot on the Pi.

Having a remote desktop is extremely useful, but if we need to run the tightvncserver program on the Raspberry Pi each time we want to use the remote desktop, it would be extremely inconvenient. What we will do instead is set up tightvncserver so that it starts automatically each time the Raspberry Pi boots up.

To do this we are going to use a little bit of Linux cleverness. We'll explain it as we go along, but be aware, some will find the explanations a little tiresome (if you're already familiar) but I'm sure that there will be some readers who will benefit.

Our first task will be to edit the file /etc/rc.local. This file can contain commands that get run on start-up. If we look at the file we can see that there is already few entries in there;

```

#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
    printf "My IP address is %s\n" "$_IP"
fi

exit 0

```

The first set of lines with a hash mark (#) in front of them are comments. These are just there to explain what is going on to someone reading the file.

The lines of code towards the bottom clearly have something to do with the IP address of the computer. In fact they are a short script that checks to see if the Raspberry Pi has an IP address and if it does, it prints it out. If you recall when we were setting out IP address earlier in the chapter, we could see the IP address printed out on the screen when the Pi booted up like so

```
My IP address is 10.1.1.8
```

```
Raspbian GNU/Linux 7 raspberrypi tty1
```

```
raspberrypi login:
```

This piece of script in `rc.local` is the code responsible for printing out the IP address!

We will add the following command into `rc.local`;

```
su - pi -c '/usr/bin/tightvncserver :1'
```

This command switches user to be the ‘pi’ user with `su - pi`. The `su` stands for ‘switch user’ the dash (-) makes sure that the user pi’s environment (like all their settings) are used correctly and `pi` is the user.

The `-c` option declares that the next piece of the line is going to be the command that will be run and the part inside the quote marks (`'/usr/bin/tightvncserver :1'`) is the command.

The command in this case executes the file `tightvncserver` which is in the `/usr/bin` directory and it specifies that we should start desktop session 1 (`:1`).

To do this we will edit the `rc.local` file with the following command;

```
sudo nano /etc/rc.local
```

Add in our lines so that the file looks like the following;

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
    printf "My IP address is %s\n" "$_IP"
fi

# Start tightvncserver
su - pi -c '/usr/bin/tightvncserver :1'

exit 0
```

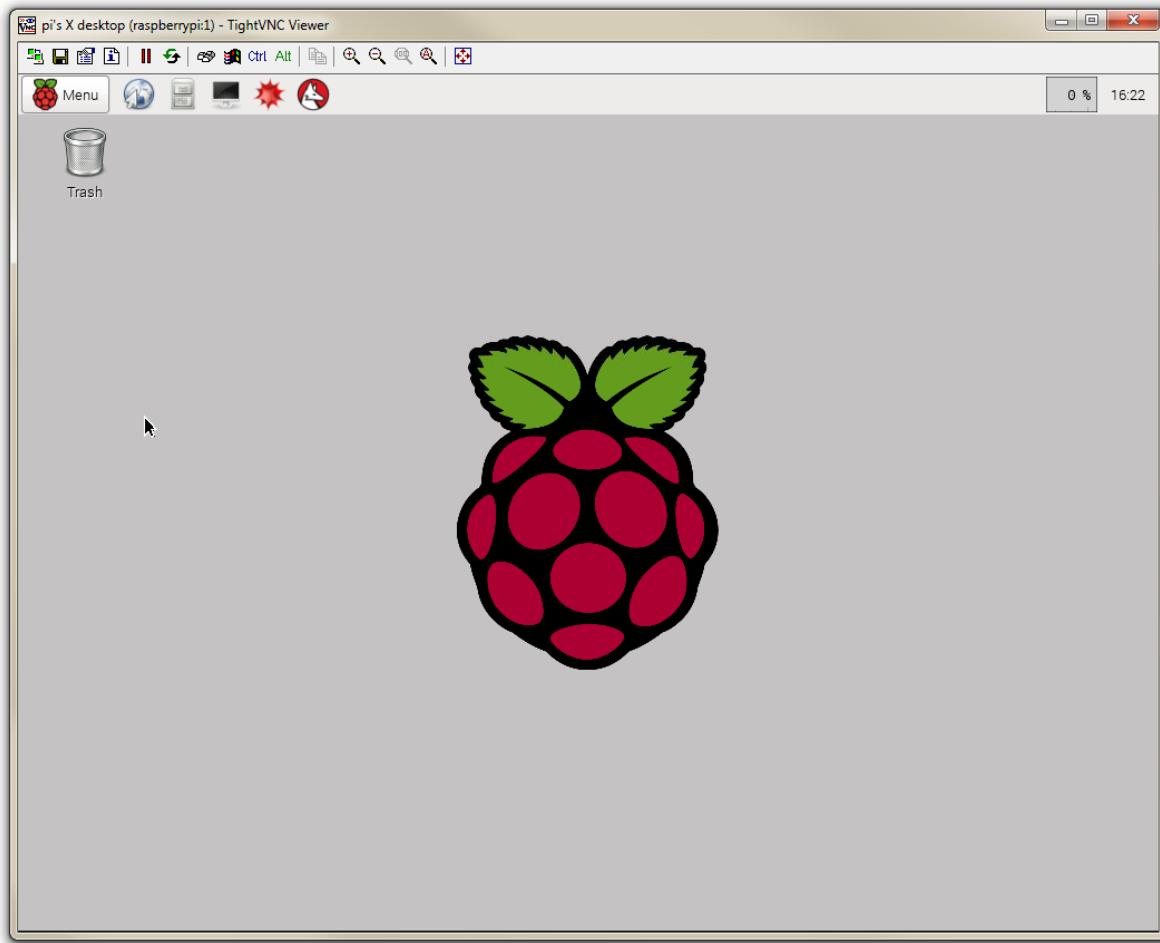
(We can also add our own comment into the file to let future readers know what's going on)

That should be it. We should now be able to test that the service starts when the Pi boots by typing in;

```
sudo reboot
```

When the Raspberry has finished starting up again, we should be able to see in the list of text that shows up while the boot sequence is starting the line `New 'X' desktop is raspberrypi:1`.

Assuming that this is the case, we can now start the TightVNC Viewer program on the Windows desktop and we will be able to see a remote view of the Raspberry Pi's desktop.



TightVNC Desktop



There are alternative ways to allow tightvncserver to start automatically, and I am sure that they are completely valid for different purposes. One of the most popular allows the remote desktop to start as the root user. The instructions above deliberately start the desktop as the pi user to attempt to mirror the user experience as if accessing the Raspberry Pi directly. It also preserves a degree of security in that the user can not automatically start messing up some of the more delicate functions of the computer (they just have to think a little more about it :-)).

Now for the big test.

Power off the Raspberry Pi;

```
sudo poweroff
```

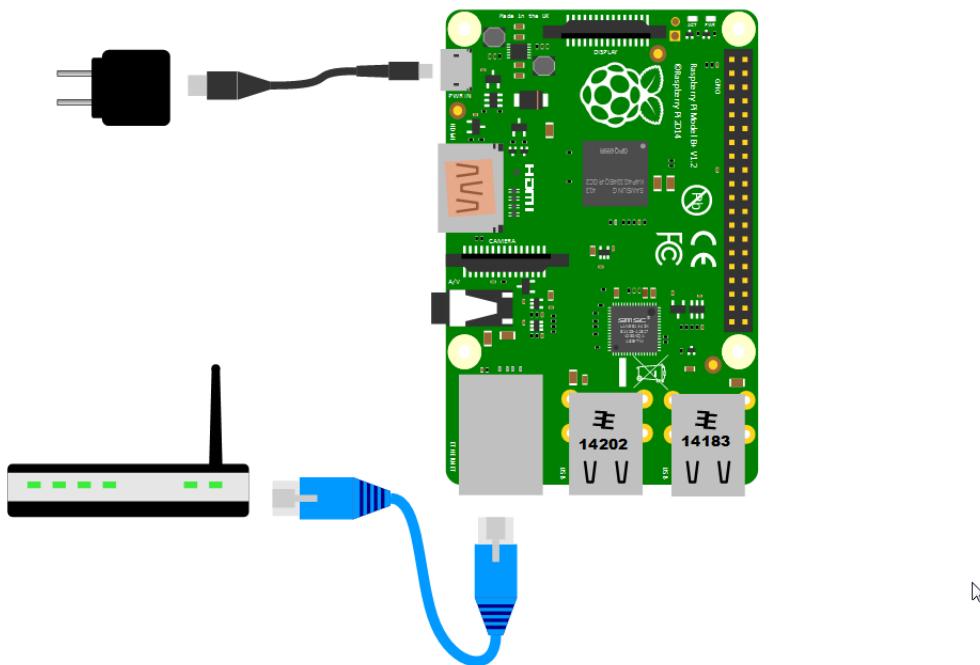
Now physically turn off the power to the Pi.

Unplug the keyboard / mouse and the video from the unit so that there is only the power connector and the network cable left plugged in.

Now turn the power back on.

(We will need to wait for 30 seconds or so while it boots up)

Start the TightVNC Viewer program on the Windows desktop and we will be able to see a remote view of the Raspberry Pi's desktop but this time the Pi doesn't have a keyboard / mouse or video screen attached.



Raspberry Pi Liberated by TightVNC

If this is the first time that you've done something like this it can be a very liberating feeling. Suddenly you can see possibilities for using the Raspberry Pi that do not involve having it physically tethered to a lot of extra peripherals. And if you're anyone like me, the next thing you do is ask yourself, "*How can I get rid of that network cable?*".

## Remote access via SSH

Secure Shell ([SSH<sup>23</sup>](#)) is a network protocol that allows secure data communication, remote command-line login, remote command execution, and other secure network services between two networked computers. It connects, via a secure channel over an insecure network, a server and a client running SSH server and SSH client programs, respectively (there's the client-server model again).

In our case the SSH program on the server is running `sshd` and on the Windows machine we will use a program called ‘PuTTY’.

### Setting up the Server (Raspberry Pi)

This is definitely one of the easiest set-up steps since SSH is already installed on Raspbian.

To check that it is there and working type the following from the command line;

```
/etc/init.d/ssh status
```

The Pi should respond with the message that the program `sshd` is running.

```
pi@raspberrypi ~ $ /etc/init.d/ssh status
[ ok ] sshd is running.
```

sshd Running

### Installing SSH on the Raspberry Pi.



You only need to carry out this step if SSH is **not** installed.

If for some reason SSH is *not* installed on your Pi, you can easily install with the command;

```
sudo apt-get install ssh
```

Once this has been done SSH will start automatically when the Raspberry Pi boots up.

### Setting up the Client (Windows)

The client software we will use is called ‘[Putty<sup>24</sup>](#)’. It is open source and available for download from [here<sup>25</sup>](#).

<sup>23</sup>[http://en.wikipedia.org/wiki/Secure\\_Shell](http://en.wikipedia.org/wiki/Secure_Shell)

<sup>24</sup><http://www.putty.org/>

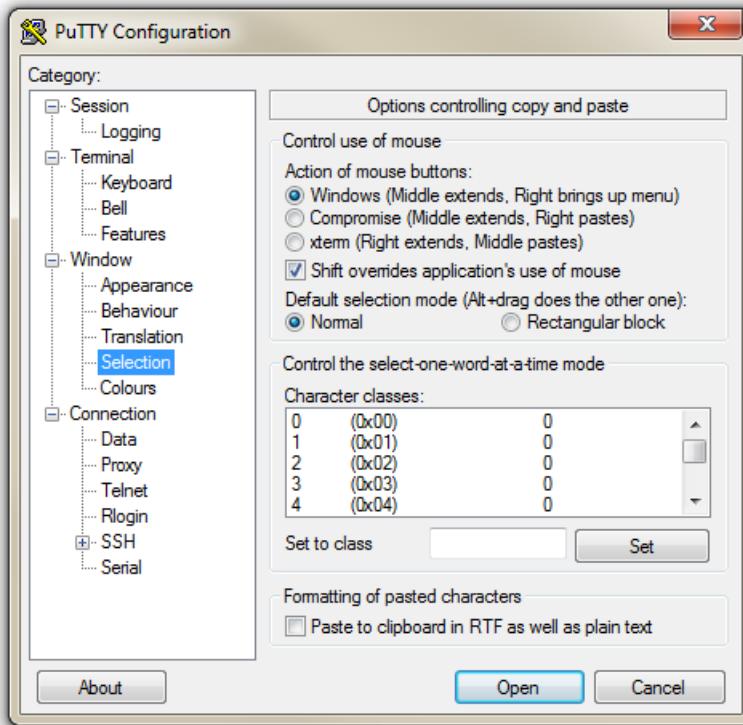
<sup>25</sup><http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>

On the download page there are a range of options available for use. The best option for us is most likely under the ‘For Windows on Intel x86’ heading and we should just download the ‘putty.exe’ program.

Save the file somewhere logical as it is a stand-alone program that will run when you double click on it (you can make life easier by placing a short-cut on the desktop).

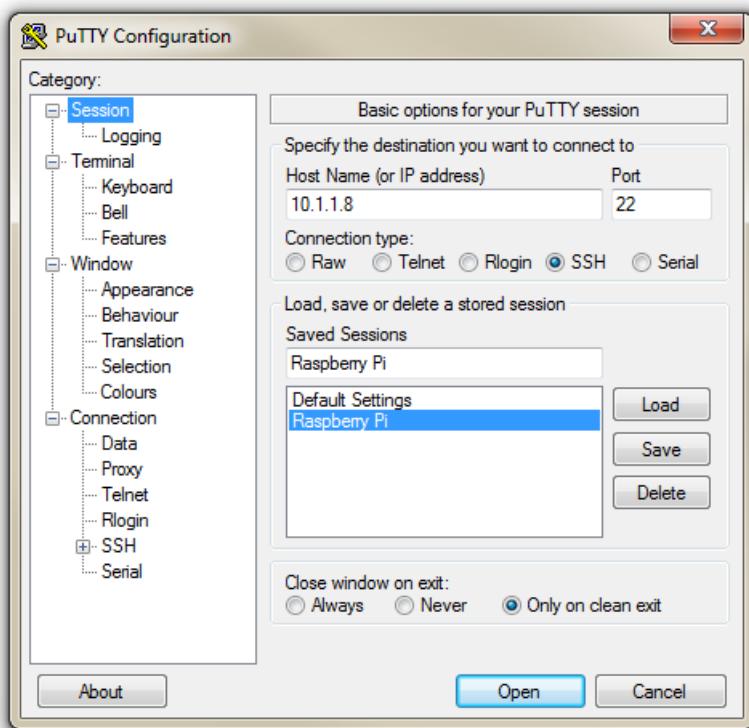
Once we have the file saved, run the program by double clicking on it and it will start without problem.

The first thing we will set-up for our connection is the way that the program recognises how the mouse works. In the ‘Window’ Category on the left of the PuTTY Configuration box, click on the ‘Selection’ option. On this page we want to change the ‘Action of mouse’ option from the default of ‘Compromise (Middle extends, Right paste)’ to ‘Windows (Middle extends, Right brings up menu)’. This keeps the standard Windows mouse actions the same when you use PuTTY.



PuTTY Selection Set-up

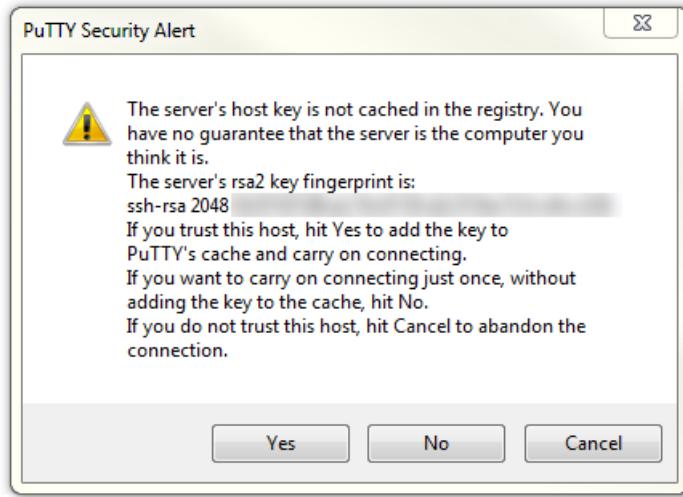
Now select the ‘Session’ Category on the left hand menu. Here we want to enter our static IP address that we set up earlier (10.1.1.8 in the example that we have been following, but use *your* one) and because we would like to access this connection on a frequent basis we can enter a name for it as a saved session (In the scree-shot below it is imaginatively called ‘Raspberry Pi’). Then click on ‘Save’.



### PuTTY Session Set-up

Now we can select our raspberry Pi Session (per the screen-shot above) and click on the 'Open' button.

The first thing you will be greeted with is a window asking if you trust the host that you're trying to connect to.

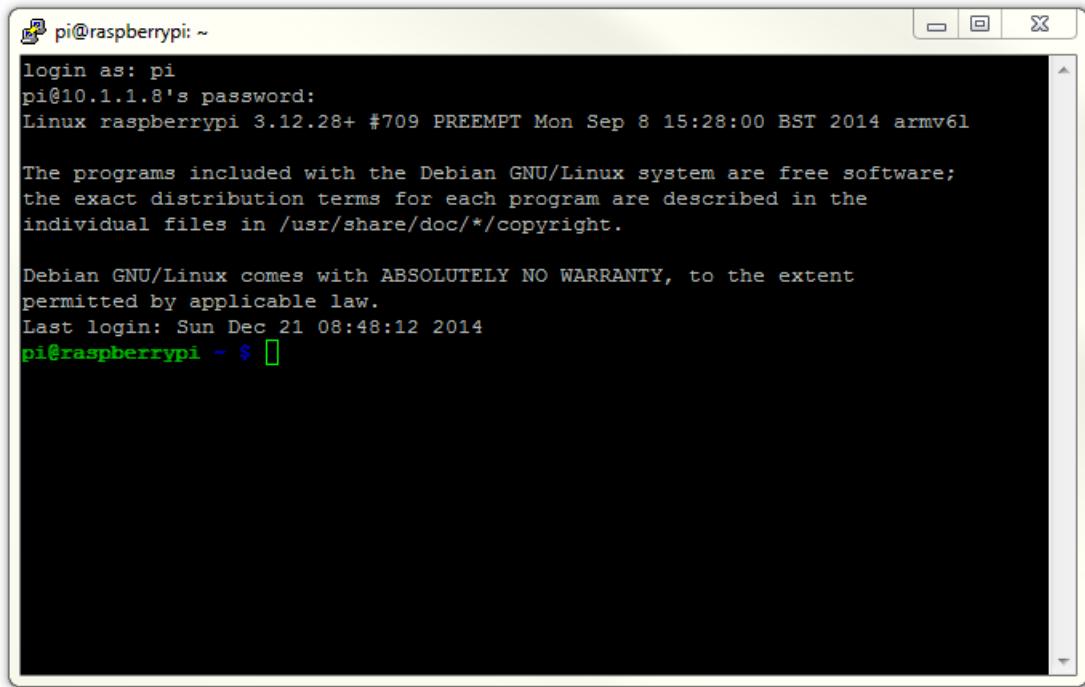


### PuTTY Session Connection

In this case it is a pretty safe bet to click on the 'Yes' button to confirm that we know and trust the connection.

Once this is done, a new terminal window will be shown with a prompt to login as: . Here we

can enter our user name ('pi') and then our password (if it's still the default it is 'raspberry').



```
pi@raspberrypi: ~
login as: pi
pi@10.1.1.8's password:
Linux raspberrypi 3.12.28+ #709 PREEMPT Mon Sep 8 15:28:00 BST 2014 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Dec 21 08:48:12 2014
pi@raspberrypi ~ $
```

#### PuTTY Session Connected

There you have it. A command line connection via SSH. Well done.

As I mentioned at the end of the section on remotely accessing the Raspberry Pi's GUI, if this is the first time that you've done something like this it can be a very liberating feeling. To complete the feeling of freedom let's set up a wireless network connection.

## Setting up a WiFi Network Connection

Our set-up of the Raspberry Pi has us at a point where we are able to carry out all the (computer interface) interactions we will require via a remote desktop. However, the Raspberry Pi is making that remote connection via a fixed network cable. It could be argued that to fulfil the ultimate aspirations of sensing different aspects of our world we will need to be able to place the platform that we want to do the measuring in a position where we have no physical network connection. The most obvious solution to this conundrum is to enable a wireless connection.

It should be noted that enabling a wireless network will not be a requirement for everyone and as such, I would only recommend it if you need to. It means that you will need to purchase a USB WiFi dongle and correctly configure it which as it turns out can be something of an exercise. In my own experience, I found that choosing the right wireless adapter was the key to making the job simple enough to be able to recommend it to new comers. Not all WiFi adapters are well supported and if you are unfamiliar with the process of installing drivers or compiling code, then I would recommend that you opt for an adapter that is supported and will work ‘out of the box’. There is an [excellent page on elinux.org<sup>26</sup>](http://elinux.org/RPi_USB_Wi-Fi_Adapters) which lists different adapters and their requirements. I eventually opted for the Edimax EW-7811Un which literally ‘just worked’ and I would recommend it to others for its ease of use and relatively low cost (approximately \$15 US).



Edimax WiFi USB Adapter



Bearing in mind that we are going to be adjusting our network connection, it is highly recommended that the following configuration changes take place with the keyboard / mouse and monitor connected to the Raspberry Pi (I.e. not via a remote desktop).

To install the wireless adapter we should start with the Pi powered off and install it into a convenient USB connection. When we turn the power on we will see the normal range of messages scroll by, but if we’re observant we will note that there are a few additional lines concerning a USB device. These lines will most likely scroll past, but once the device has finished powering up and we have logged in we can type in...

<sup>26</sup>[http://elinux.org/RPi\\_USB\\_Wi-Fi\\_Adapters](http://elinux.org/RPi_USB_Wi-Fi_Adapters)

```
dmesg
```

... which will show us a range of messages about drivers that are loaded to support discovered hardware.

Towards the end of the list (it shouldn't have scrolled off the window) will be a series of messages that describe the USB connectors and what is connected to them. In particular we could see a group that looks a little like the following;

```
[3.382731] usb 1-1.2: new high-speed USB device number 4 using dwc_otg
[3.494250] usb 1-1.2: New USB device found, idVendor=7392, idProduct=7811
[3.507749] usb 1-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[3.520230] usb 1-1.2: Product: 802.11n WLAN Adapter
[3.542690] usb 1-1.2: Manufacturer: Realtek
[3.560641] usb 1-1.2: SerialNumber: 00345767831a5e
```

That is our USB adapter which is plugged into USB slot 2 (which is the '2' in `usb 1-1.2:`). The manufacturer is listed as 'Realtek' as this is the manufacturer of the chip-set in the adapter that Edimax uses.

In the same way that we edited the `/etc/network/interfaces` file earlier to set up the static IP address we will now edit it again with the command...

```
sudo nano /etc/network/interfaces
```

This time we will edit the `interfaces` file so that it looks like the following;

```
auto lo

iface lo inet loopback

iface eth0 inet dhcp

allow-hotplug wlan0
auto wlan0

iface wlan0 inet static
    address 10.1.1.8
    netmask 255.255.255.0
    gateway 10.1.1.1
    wpa-ssid "homenetwork"
    wpa-psk "h0mepassw0rd"
```

Here we have reverted the `eth0` interface (the wired network connection) to have it's network connection assigned dynamically (`iface eth0 inet dhcp`).

Our wireless lan (`wlan0`) is now designated to be a static IP address (with the details that we had previously assigned to our wired connection) and we have added the ‘ssid’ (the network name) of the network that we are going to connect to and the password for the network. Save the file.



If you’re not sure about the name (ssid) of your network, a simple test would be to use a phone or tablet to see what WiFi connection it is using (assuming that you are using your own wifi connection).

To allow the changes to become operative we can type in;

```
sudo reboot
```

Once we have rebooted, we can check the status of our network interfaces by typing in;

```
ifconfig
```

This will display the configuration for our wired Ethernet port, our ‘Local Loopback’ (which is a fancy way of saying a network connection for the machine that you’re using, that doesn’t require an actual network (ignore it in the mean time)) and the `wlan0` connection which should look a little like this;

```
wlan0 Link encap:Ethernet HWaddr 80:1f:02:f4:21:85
      inet addr:10.1.1.8 Bcast:10.1.1.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:213 errors:0 dropped:90 overruns:0 frame:0
          TX packets:54 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:88729 (86.6 KiB) TX bytes:6467 (6.3 KiB)
```

This would indicate that our wireless connection has been assigned the static address that we were looking for (10.1.1.8).

We should be able to test our connection by connecting to the Pi via the TightVNC Viewer on the Windows desktop.

In theory you are now the proud owner of a computer that can be operated entirely separate from all connections except power.

## Web Server and PHP

Because we want to be able to explore the data we will be collecting, we need to set up a web server that will present web pages to other computers that will be browsing within the network (remembering that this is not intended to be connected to the internet, just inside your home network). At the same time as setting up a web server on the Pi we will install PHP.

At the command line run the following command;

```
sudo apt-get install apache2 php5 libapache2-mod-php5
```

We're familiar with apt-get already, but this time we're including more than one package in the installation process. Specifically we're including apache2, php5 and libapache2-mod-php5.

'apache2' is the name of the web server and php5, libapache2-mod-php5 are for PHP.

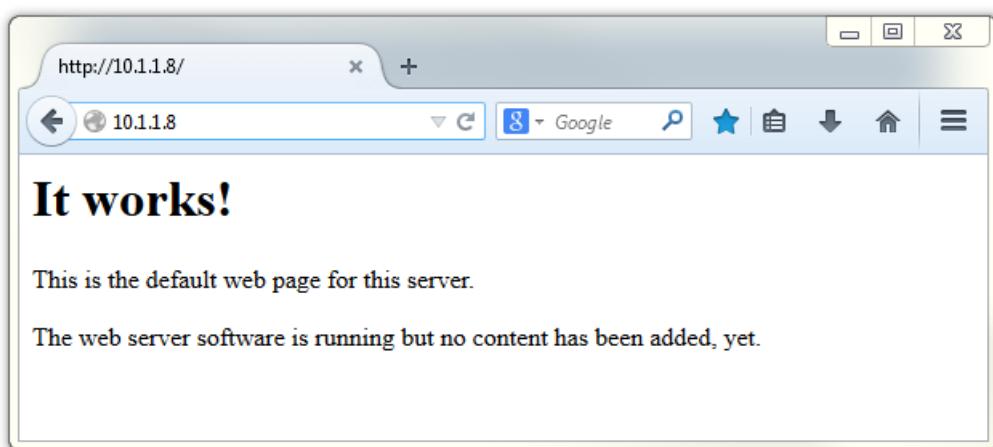
The Raspberry Pi will advise you of the range of additional packages that will be installed at the same time (to support those we're installing (these additional packages are dependencies)). Agree to continue and the installation will proceed. This should take a few minutes or more (depending on the speed of your Internet connection).

Once complete we need to restart our web server with the following command;

```
sudo service apache2 restart
```

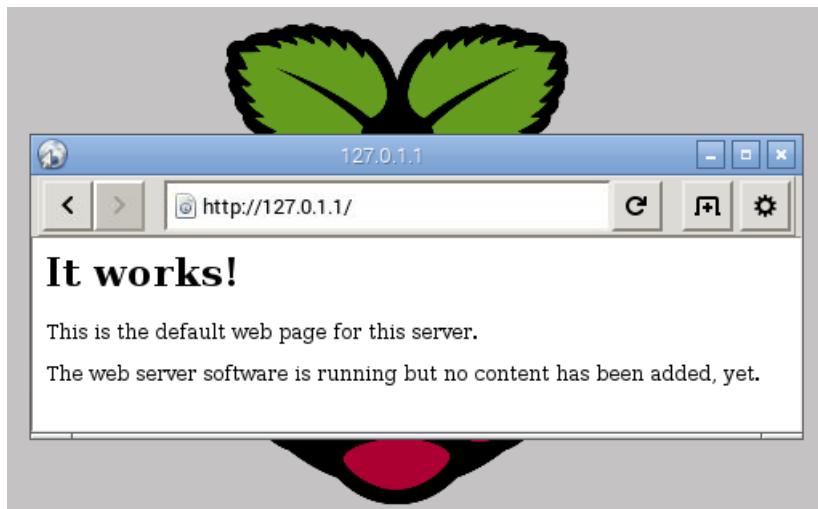
We can now test our web server from the windows desktop machine.

Open up a web browser and type in the IP address of the Raspberry Pi into the URL bar at the top. Hopefully we will see...



Testing the web server

We could also test the web server from the Pi itself. From the Raspberry Pi's desktop start the Epiphany Web Browser and enter either 10.1.1.8 (or 127.0.1.1 which is the address that the Pi can 'see' internally (called the '[localhost<sup>27</sup>](#)' address)) into the URL bar at the top. We should see the following...



Testing the web server

Congratulations! We have a web server.

## Tweak permissions for easier web file editing

The default installation of the Apache web server has the location of the files that make up our web server owned by the 'root' user. This means that if we want to edit them we need to do so with the permissions of the root user. This can be easily achieved by typing something like `sudo nano /var/www/index.html`, but if we want to be able to use an editor from our GUI desktop, we will need the permissions to be set to allow the 'pi' user to edit them without having to invoke `sudo`.

We're going to do this by making a separate group ('www-data' will be its name) the owners of the `/var/www` directory (where our web files will live) then we will add the 'pi' user to the 'www-data' group.

We start by making the 'www-data' group and user the owner of the `/var/www` directory with the following command;

```
sudo chown www-data:www-data /var/www
```

(For more information on the `chown` command check out the [Glossary](#))

Then we allow the 'www-data' group permission to write to the directory;

---

<sup>27</sup><http://en.wikipedia.org/wiki/Localhost>

```
sudo chmod 775 /var/www
```

(For more information on the `chmod` command check out the [Glossary](#))

Then we add the ‘pi’ user to the ‘www-data’ group;

```
sudo usermod -a -G www-data pi
```

(For more information on the `usermod` command check out the [Glossary](#))

This change in permissions are best enacted by rebooting the Raspberry Pi;

```
sudo reboot
```

Now the ‘pi’ user has the ability to edit files in the `/var/www` directory without problem.

## Database

As mentioned earlier in the book, we will use a MySQL database to store the information that we collect.

We will install MySQL in a couple of steps and then we will install the database administration tool phpMyAdmin to make our lives easier.

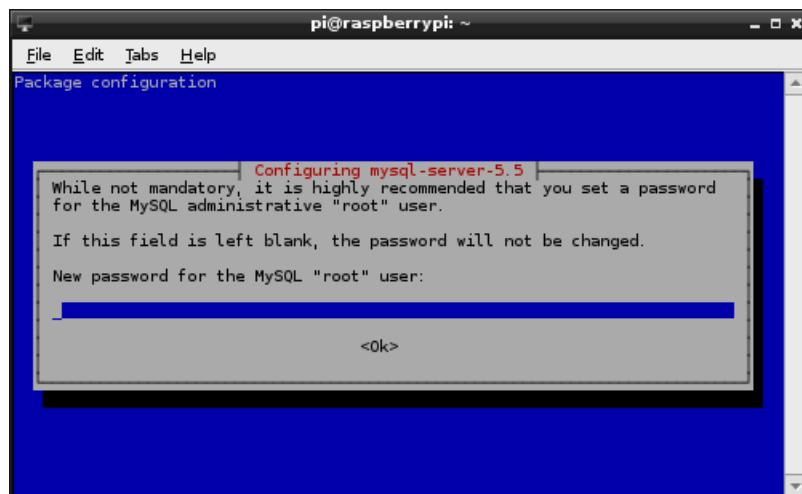
## MySQL

From the command line run the following command;

```
sudo apt-get install mysql-server
```

The Raspberry Pi will advise you of the range of additional packages that will be installed at the same time. Agree to continue and the installation will proceed. This should take a few minutes or more (depending on the speed of your internet connection).

You will be prompted (twice) to enter a root password for your database. Note it down somewhere safe;



MySQL Installation

Make this a reasonably good password. You won't need it too much, so it's reasonable to make it more secure.

Once this installation is complete, we will install a couple more packages that we will use in the future when we integrate PHP and Python with MySQL. To do this enter the following from the command line;

```
sudo apt-get install mysql-client php5-mysql python-mysqldb
```

Agree to the installed packages and the installation will proceed fairly quickly.

That's it! MySQL server installed. However, it's not configured for use, so we will install phpMyAdmin to help us out.

## phpMyAdmin

phpMyAdmin<sup>28</sup> is free software written in PHP to carry out administration of a MySQL database installation.

To begin installation run the following from the command line;

```
sudo apt-get install phpmyadmin
```

Agree to the installed packages and the installation will proceed.

You will receive a prompt to ask what sort of web server we are using;

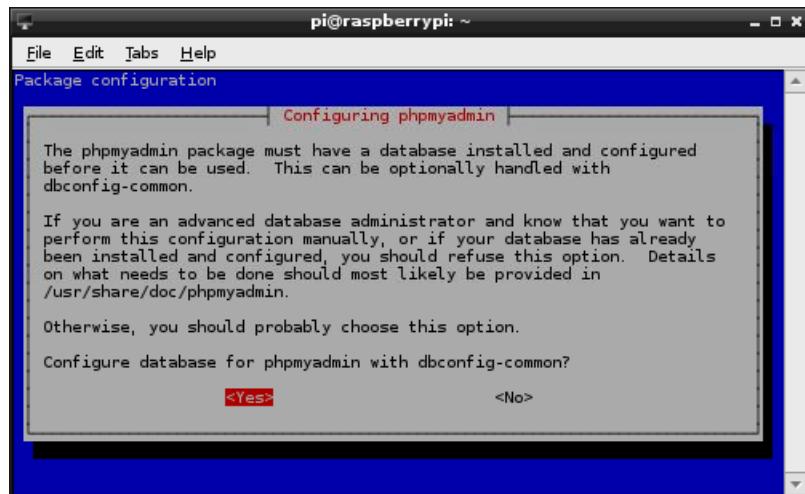


phpMyAdmin Installation

Select 'apache2' and tab to 'Ok' to continue.

We will then be prompted to configure the database for use with phpMyAdmin;

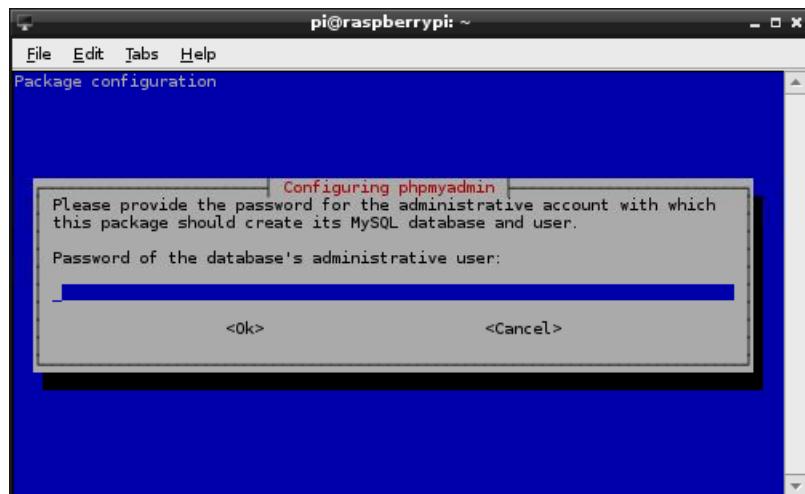
<sup>28</sup>[http://www.phpmyadmin.net/home\\_page/index.php](http://www.phpmyadmin.net/home_page/index.php)



phpMyAdmin Configuration

We want the program to look after it for us, so select ‘Yes’ and continue.

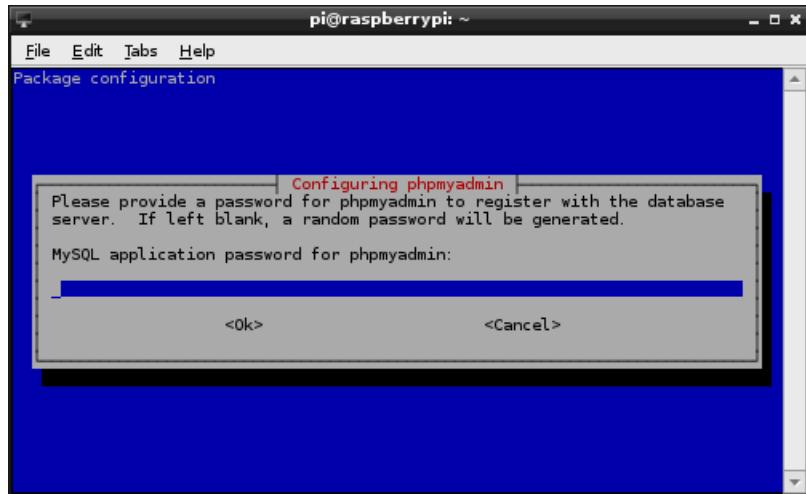
We will then be prompted for the password for the administrative account for the MySQL database.



MySQL Administrative Password

This is the root password for MySQL that we set up earlier. Enter it and tab to ‘Ok’ to continue.

We will then be prompted for a password for phpMyAdmin to access MySQL.



#### phpMyAdmin Administrative Password

I have used the same password as the MySQL root password in the past to save confusion, but over to you. Just make sure you note it down :-). Then tab to 'Ok' to continue (and confirm).

The installation should conclude soon.

Once finished, we need to edit the Apache web server configuration to access phpMyAdmin. To do this execute the following command from the command line;

```
sudo nano /etc/apache2/apache2.conf
```

Get to the bottom of the file by pressing ctrl-v a few times and there we want to add the line;

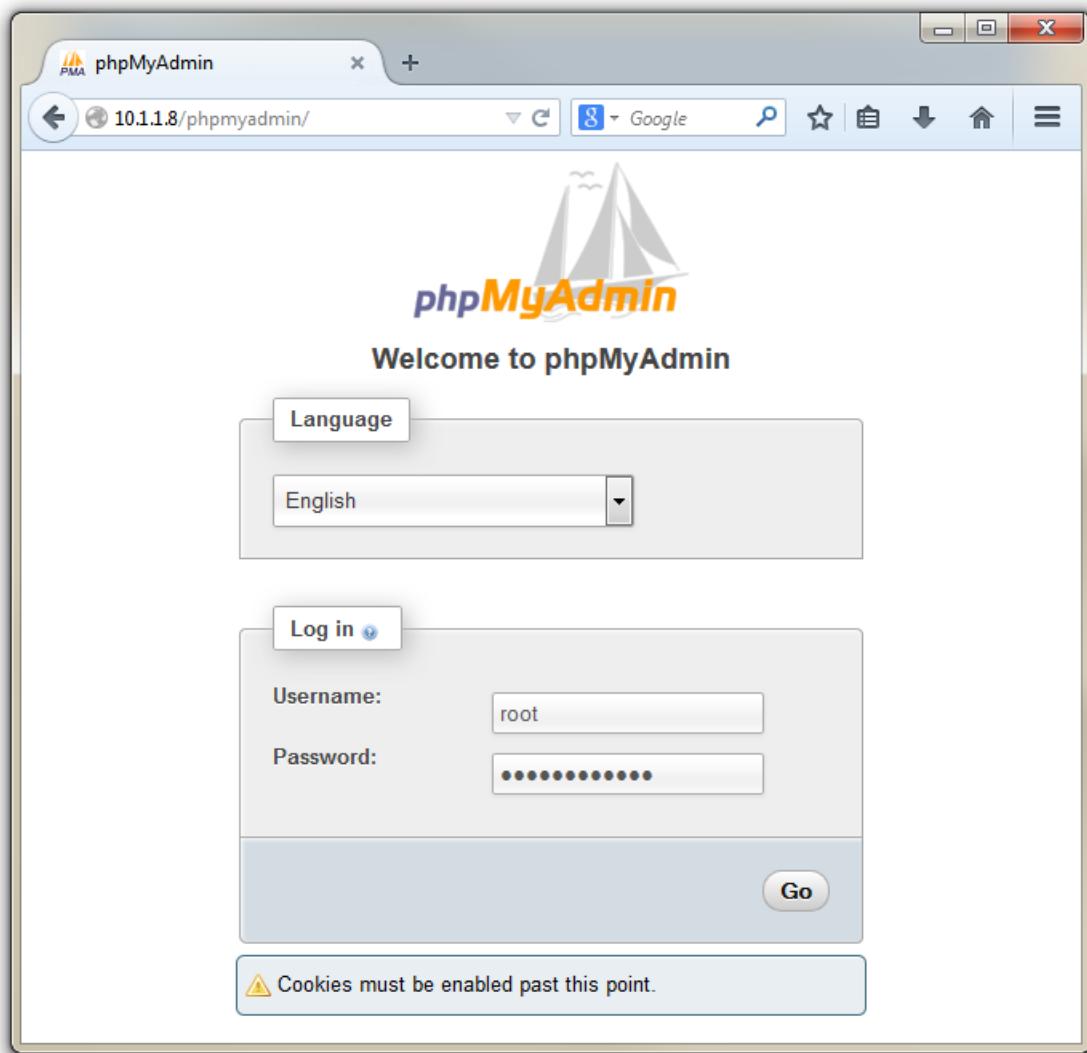
```
Include /etc/phpmyadmin/apache.conf
```

Save the file and then restart Apache2;

```
sudo service apache2 restart
```

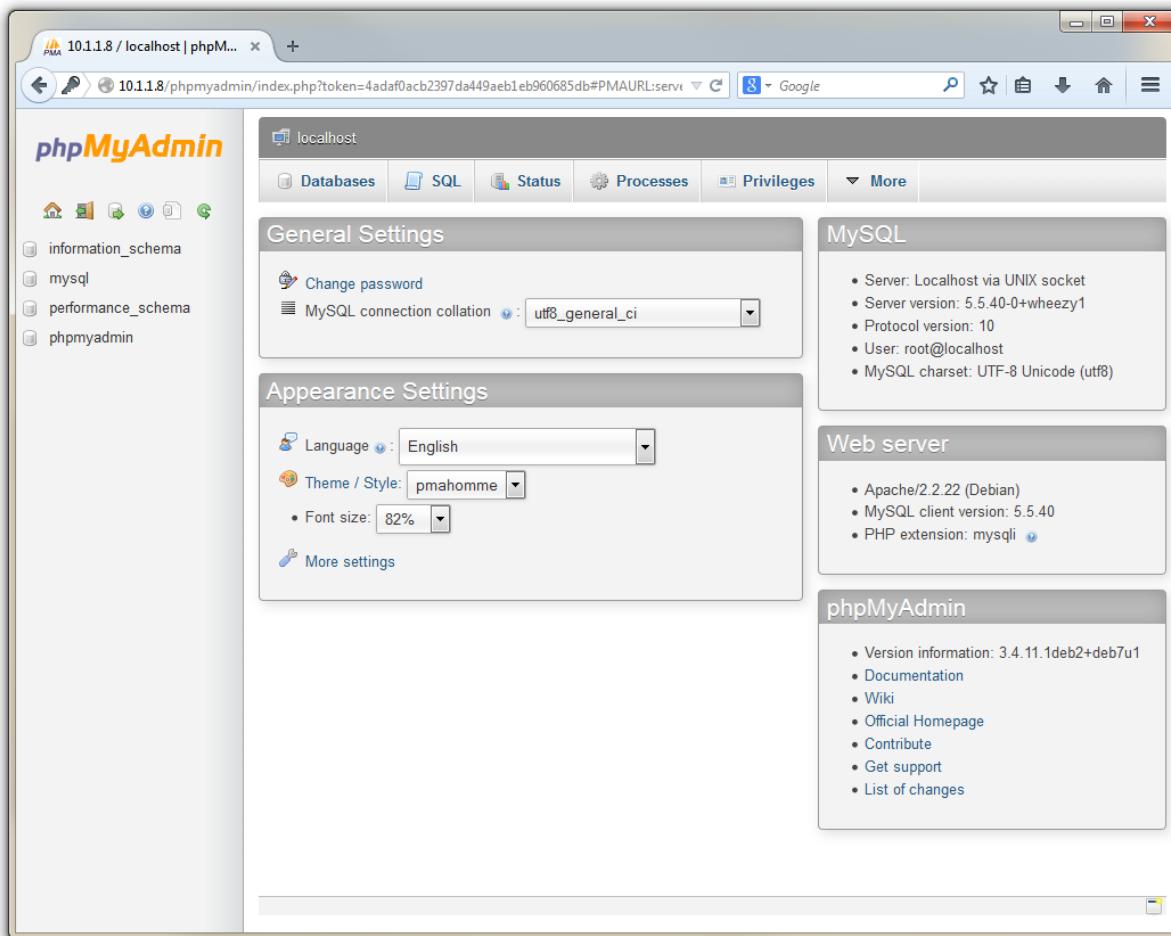
This should have everything running.

Now if we go to our browser on the Windows (or on the Raspberry Pi) desktop and enter the IP address followed by /phpmyadmin (in the case of our example 10.1.1.8/phpmyadmin) it should start up phpMyAdmin in the browser.



### phpMyAdmin Web

If you enter the username as 'root' and the MySQL root password that we set earlier, it will open up the phpMyAdmin interface.



phpMyAdmin Web Interface

## Allow access to the database remotely

It might seem a little strange to say that we want to allow access to the database remotely since this would appear to be part of the grand plan all along. But there are different types of access and in particular there may be a need for a remote computer to access the database directly.

This direct access occurs when (for example) a web server on a different computer to the Raspberry Pi wants to use the data. In this situation it would need to request access to the database over the network by referencing the host computer that the database was on (in this case we have specified that it is on the computer at the IP address 10.1.1.8).

By default the Raspberry Pi's Operating System is set up to deny that access and if this is something that you want to allow this is what you will need to do.

On the Raspberry Pi we need to edit the configuration file 'my.cnf' in the directory /etc/mysql/. We do this with the following command;

```
sudo nano /etc/mysql/my.cnf
```

Scroll down the file a short way till we find the section [`mysqld`]. Here we need to edit the line that reads something similar to;

```
bind-address      = 127.0.0.1
```

This line is telling MySQL to only listen to commands that come from the ‘localhost’ network (127.0.0.1). Essentially only the Raspberry Pi itself. We can change this by inserting a ‘#’ in front of the line which will turn the line into a comment instead of a configuration option. So change it to look like this;

```
# bind-address      = 127.0.0.1
```

Once we have saved the file, we need to restart the MySQL service with the following command;

```
sudo service mysql restart
```

## Create users for the database

Our database has an administrative (root) user, but for our future purposes, we will want to add a couple more users to manage the security of the database in a responsible way.

If we click on the ‘Privileges’ tab in phpMyAdmin we can see the range of users that are already set up.

User	Host	Password	Global privileges	Grant	Action
debian-sys-maint	localhost	Yes	ALL PRIVILEGES	Yes	Edit Privileges  Export
phpmyadmin	localhost	Yes	USAGE	No	Edit Privileges  Export
root	127.0.0.1	Yes	ALL PRIVILEGES	Yes	Edit Privileges  Export
root	::1	Yes	ALL PRIVILEGES	Yes	Edit Privileges  Export
root	localhost	Yes	ALL PRIVILEGES	Yes	Edit Privileges  Export
root	raspberrypi	Yes	ALL PRIVILEGES	Yes	Edit Privileges  Export

phpMyAdmin Users

We will create an additional two users. One that can only read (select) data from our database and another that can put data into (insert) the database. Keeping these functions separate gives us some flexibility in how we share the data in the future. For example we could be reasonably comfortable providing the username and password to allow reading the data to a wide range of people, but we should really *only* allow our future scripts the ability to insert data into the database.

From the ‘Privileges’ tab, select ‘Add a new user’;

**Login Information**

User name:	Use text field: pi_select
Host:	Any host 1
Password:	Use text field: *****
Re-type:	*****
Generate password:	<b>Generate</b>

**phpMyAdmin Login Information**

Enter a user name and password.

Then we scroll down a little and select the type of access we want the pi\_select user to have in the ‘Global privileges’ section. For the pi\_select user under ‘Data’ we want to tick ‘SELECT’.

**Global privileges (Check All / Uncheck All)**

*Note: MySQL privilege names are expressed in English*

<b>Data</b>	<b>Structure</b>	<b>Administration</b>
<input checked="" type="checkbox"/> SELECT <input type="checkbox"/> INSERT <input type="checkbox"/> UPDATE <input type="checkbox"/> DELETE <input type="checkbox"/> FILE	<input type="checkbox"/> CREATE <input type="checkbox"/> ALTER <input type="checkbox"/> INDEX <input type="checkbox"/> DROP <input type="checkbox"/> CREATE TEMPORARY TABLES <input type="checkbox"/> SHOW VIEW <input type="checkbox"/> CREATE ROUTINE <input type="checkbox"/> ALTER ROUTINE <input type="checkbox"/> EXECUTE <input type="checkbox"/> CREATE VIEW <input type="checkbox"/> EVENT <input type="checkbox"/> TRIGGER	<input type="checkbox"/> GRANT <input type="checkbox"/> SUPER <input type="checkbox"/> PROCESS <input type="checkbox"/> RELOAD <input type="checkbox"/> SHUTDOWN <input type="checkbox"/> SHOW DATABASES <input type="checkbox"/> LOCK TABLES <input type="checkbox"/> REFERENCES <input type="checkbox"/> REPLICATION CLIENT <input type="checkbox"/> REPLICATION SLAVE <input type="checkbox"/> CREATE USER

**phpMyAdmin SELECT user**

Then press the ‘Create User’ button and we’ve created a user.

For the second user with the ability to insert data (let’s call the user ‘pi\_insert’), go through the same process, but tick the ‘SELECT’ *and* the ‘INSERT’ options for the data.

When complete we should be able to see that both of our users look similar to the following;

<input type="checkbox"/> pi_insert	%	Yes	SELECT, INSERT	No	<a href="#">Edit Privileges</a>	<a href="#">Export</a>
<input type="checkbox"/> pi_select	%	Yes	SELECT	No	<a href="#">Edit Privileges</a>	<a href="#">Export</a>
phpMyAdmin INSERT user						

## Create a database

When we read data from our sensors, we will record them in a database. MySQL is a database program, but we still need to set up a database inside that program. In fact while we will set up a database in this step, when we come to record and explore our data we will be dealing with a ‘table’ of data that will exist inside a database.

For the purposes of getting ourselves set up we need to create a database. If we go to the ‘Databases’ tab we can enter a name for a new database (here I’ve called one ‘measurements’) and click on ‘Create’.

The screenshot shows the phpMyAdmin interface with the following details:

- Top Navigation Bar:** Shows 'localhost' and tabs for Databases, SQL, Status, Processes, Privileges, Export, Import, and More.
- Databases Tab:** Shows a sub-menu with 'Create new database' and a text input field containing 'measurements'.
- Database List:** Shows a list of existing databases: 'information\_schema', 'mysql', 'performance\_schema', and 'phpmyadmin'. The 'mysql' database is currently selected.
- Total Count:** Shows 'Total: 4' at the bottom of the database list.
- Bottom Footer:** Shows 'phpMyAdmin New Database'.

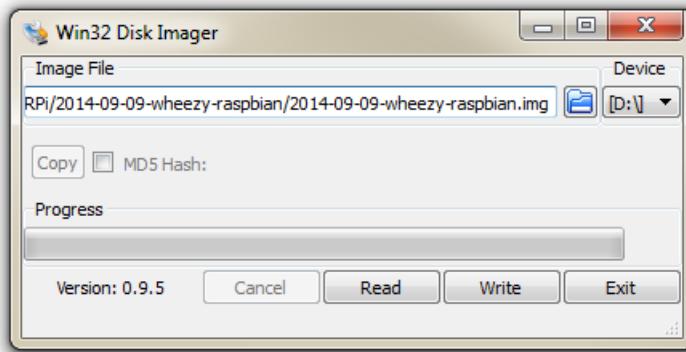
Congratulations! We’re set up and ready to go.

## Backup the Configured SD Card

Once we have installed our base of software it is a good idea to copy an image of the SD card onto the Windows machine so that we can recreate (clone) the installation back to the current state.

We will use the Open Source utility Win32DiskImager again. Use the SD card reader capable of accepting your micro SD card and note the drive letter of the SD card.

Start the Win32 Disk Imager program.



Win32 Disk Imager

Select the correct drive letter for your SD card (make sure it's the right one) and enter a new name and location for our new Raspbian disk image. Then select 'Read' and the disk imager will read the image from the SD card and create a new disk image on the windows machine at the location you have specified.

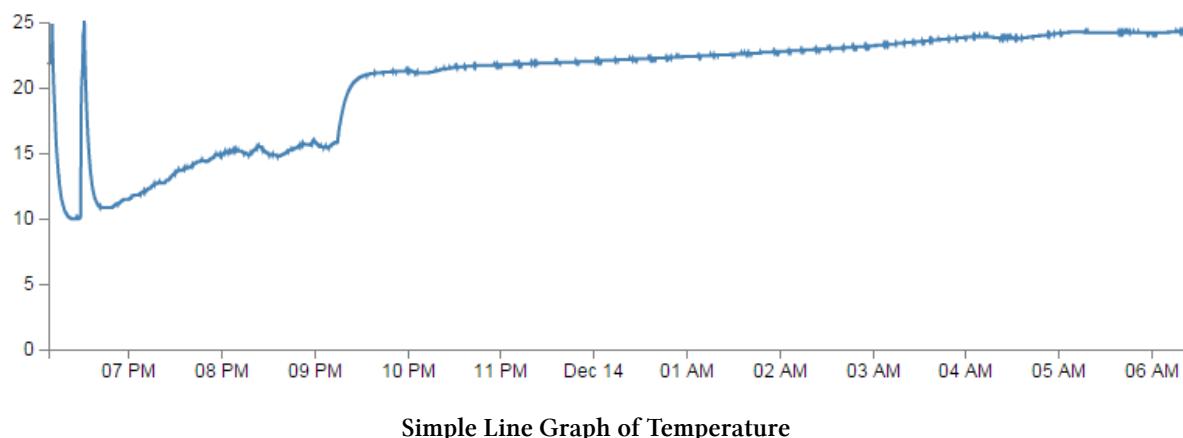
Once the process is finished, exit the disk imager and eject the card from the computer and you're done.

## Exploring data with a simple line graph

As explained earlier in the chapter, our aim is to present the data we are recording in a form that will make it easy to interpret and digest. What is presented in this portion of the set-up of the Raspberry Pi is not required to get up and going. Instead it is a description of a simple visualisation technique that will probably be applicable for just about any measured value over time. This is also the code for the first project that we will examine (measuring a single temperature).

To achieve this aim a simple mechanism to show data changing over time is a line graph. To do this we will build a web page that our web server will host that will look at the data in a table on our database and show it using a combination of PHP, HTML, JavaScript and d3.js.

Ultimately we're going to aim for a simple line graph that will look a little like this (except the data will be different of course);



You would be well within your rights to wonder if ‘That’s it?’. But think of this simple graph as the foot into the door of presenting data automatically and dynamically. We will explore different techniques for showing data as we measure different things and hopefully learn a little more about visualizing information as we go. In the mean time, the sum of all our efforts is a simple line graph :-).



For the curious, the line graph above is real data from my first ever attempt to measure temperature with the Raspberry Pi. The two spikes at the start are my son and I both holding the temperature probe in our hands to see the variation in the temperature and then the gradual increase is the temperature outside the window. The sudden jump upwards is when we brought the probe back inside.

## The full code

The code for this web page may look slightly intimidating at first glance, but we will step through it and explain what we have and it can form the basis for displaying a range of the data that we collect.



The explanation below is not as full as it could possibly be. It is intended to get you up and running and showing something on the screen, but for a fuller explanation check out ‘D3 Tips and Tricks<sup>29</sup>’ From Leanpub (Hey, it’s free!).

The full code will be named `s_temp.php` and will be saved in the `/var/www` directory (or possibly `var/www/html` depending on your version of Raspbian). The full script (which in this case is for the single temperature measurement) is as follows;

```
<?php
$hostname = 'localhost';
$username = 'pi_select';
$password = 'xxxxxxxxxx';

try {
    $dbh = new PDO("mysql:host=$hostname;dbname=measurements", $username, $pa\
ssword);

    /** The SQL SELECT statement */
    $sth = $dbh->prepare(
        "SELECT `dtg`, `temperature` FROM `temperature`"
    );
    $sth->execute();

    /* Fetch all of the remaining rows in the result set */
    $result = $sth->fetchAll(PDO::FETCH_ASSOC);

    /** close the database connection */
    $dbh = null;

}

catch(PDOException $e)
{
    echo $e->getMessage();
}

$json_data = json_encode($result);
?>
```

---

<sup>29</sup><https://leanpub.com/D3-Tips-and-Tricks>

```
<!DOCTYPE html>
<meta charset="utf-8">
<style> /* set the CSS */
```

```
body { font: 12px Arial;}
```

```
path {
  stroke: steelblue;
  stroke-width: 2;
  fill: none;
}
```

```
.axis path,
.axis line {
  fill: none;
  stroke: grey;
  stroke-width: 1;
  shape-rendering: crispEdges;
}
```

```
</style>
<body>
```

```
<!-- load the d3.js library -->
<script src="http://d3js.org/d3.v3.min.js"></script>
```

```
<script>
```

```
// Set the dimensions of the canvas / graph
var margin = {top: 30, right: 20, bottom: 30, left: 50},
    width = 800 - margin.left - margin.right,
    height = 270 - margin.top - margin.bottom;
```

```
// Parse the date / time
var parseDate = d3.time.format("%Y-%m-%d %H:%M:%S").parse;
```

```
// Set the ranges
var x = d3.time.scale().range([0, width]);
var y = d3.scale.linear().range([height, 0]);
```

```
// Define the axes
var xAxis = d3.svg.axis().scale(x)
  .orient("bottom");
```

```
var yAxis = d3.svg.axis().scale(y)
  .orient("left").ticks(5);
```

```

// Define the line
var valueline = d3.svg.line()
  .x(function(d) { return x(d.dtg); })
  .y(function(d) { return y(d.temperature); });

// Adds the svg canvas
var svg = d3.select("body")
  .append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
  .append("g")
    .attr("transform",
      "translate(" + margin.left + ", " + margin.top + ")");

// Get the data
<?php echo "data=".json_data.";" ?>
data.forEach(function(d) {
  d.dtg = parseDate(d.dtg);
  d.temperature = +d.temperature;
});

// Scale the range of the data
x.domain(d3.extent(data, function(d) { return d.dtg; }));
y.domain([0, d3.max(data, function(d) { return d.temperature; })]);

// Add the valueline path.
svg.append("path")
  .attr("d", valueline(data));

// Add the X Axis
svg.append("g")
  .attr("class", "x axis")
  .attr("transform", "translate(0, " + height + ")")
  .call(xAxis);

// Add the Y Axis
svg.append("g")
  .attr("class", "y axis")
  .call(yAxis);

</script>
</body>

```



The full code can be found in the code samples bundled with this book (s\_temp.php).

The code is roughly in three blocks. The first block is the PHP section at the start of the script. This is everything between the `<?php` and `?>` instances. The second is an HTML section between `<!DOCTYPE html>` and `</body>`. The third is actually contained within the second section and is the code between `<script>` and `</script>`.

## PHP

The job of our block of PHP at the start of the file is to select the data that we want from our database and to store it in a variable that we will then use later in the code.

The cool thing about PHP is that when it runs, it runs on the server where the file exists (in this case the Raspberry Pi). This means that that entire block is essentially invisible to a remote user just wanting to view the graph (if they chose to look at the code that drew the graph). That's a good thing, because you will notice that we have our password in the file and broadcasting passwords to the world (in spite of the fact that this is never intended to go outside a home network) is not really a good thing. More alert readers will pick up that we are using a technique for querying the database called PDO. This is a newer method for accessing a MySQL (and other types) database and is a more secure method than the traditional method (which early readers of this book would have seen).

The other thing that observant readers will notice is that we actually have two pieces of PHP code in the file. The large block at the start and a smaller snippet in the middle of the HTML and JavaScript (`<?php echo "data=".json_data.";" ?>`). We'll cover the second snippet when we talk about the JavaScript portion as by then it will be a lot more obvious what we're trying to do.

## The code

The section of code we will explain first is as follows;

```

<?php
$hostname = 'localhost';
$username = 'pi_select';
$password = 'xxxxxxxxxx';

try {
    $dbh = new PDO("mysql:host=$hostname;dbname=measurements",
                  $username, $password);

    /*** The SQL SELECT statement ***/
    $sth = $dbh->prepare(
        "SELECT `dtg`, `temperature` FROM `temperature`");
    $sth->execute();

    /* Fetch all of the remaining rows in the result set */
    $result = $sth->fetchAll(PDO::FETCH_ASSOC);
}

```

```

/** close the database connection **/
$dbo = null;

}

catch(PDOException $e)
{
    echo $e->getMessage();
}

$json_data = json_encode($result);
?>

```

The <?php line at the start and the ?> line at the end form the wrappers that allow the requesting page to recognise the contents as PHP and to execute the code rather than downloading it for display.

The following lines set up a range of important variables;

```

$hostname = 'localhost';
$username = 'pi_select';
$password = 'xxxxxxxxxx';

```

Hopefully you will recognise that these are the configuration details for the MySQL database that we set up. There's the user and the password (remember, the script isn't returned to the browser, the browser doesn't get to see the password and in this case our user has a very limited set of privileges). There's the host address that contains our database (in this case it's local (on the same server), so we use localhost, but if the database was on a remote server, we would just include its address (10.1.1.8 in our example)) and there's the database we're going to access.

We are using the try ... catch technique in the code to identify when an exception to our code has occurred (think of an exception like an error, but one you can continue on from). An exception can be thrown, and caught ("caught") within PHP. Code may be surrounded in a try block, to facilitate the catching of potential exceptions. Each try must have at least one corresponding catch block. Normal execution will continue after that last catch block defined in sequence.

When an exception is thrown, code following the statement will not be executed, and PHP will attempt to find the first matching catch block. If an exception is not caught, a PHP Fatal Error will be issued with an "Uncaught Exception ..." message, unless a handler has been defined with set\_exception\_handler().

Then we use those variables to define how we will connect to the server and the database...

```
$dbh = new PDO("mysql:host=$hostname;dbname=measurements",
    $username, $password);
```

... then we prepare our query that will request the data from the database table...

```
$sth = $dbh->prepare(
    "SELECT `dtg`, `temperature` FROM `temperature`"
);
```

(This particular query is telling the database to SELECT our date/time data (from the dtg column) and the temperature values (from the temperature column) FROM the table temperature.)

... and execute the query.

```
$sth->execute();
```

We fetch all the returned data and place it in an associative array.

```
$result = $sth->fetchAll(PDO::FETCH_ASSOC);
```

Then we close the connection to the database.

```
$dbh = null;
```

...and then we check to see if it was successful. If it wasn't, we output the exception message with our catch section;

```
catch(PDOException $e)
{
    echo $e->getMessage();
}
```

We then encode the contents of our array in a the JSON format (follow the link for a section in the appendices that explains how JSON works).

```
$json_data = json_encode($result);
```

Whew!

## HTML

This stands for HyperText Markup Language and is the stuff that web pages are made of. Check out the definition and other information on [Wikipedia<sup>30</sup>](#) for a great overview. Just remember that all we're going to use HTML for is to hold the code that we will use to present our information.

The HTML code we are going to examine also includes [Cascading Style Sheets<sup>31</sup>](#) (everyone appears to call them 'Style Sheets' or 'CSS') which are a language used to describe the formatting (or "look and feel") of a document written in a markup language (HTML in this case). The job of CSS is to make the presentation of the components we will draw with D3 simpler by assigning specific styles to specific objects. One of the cool things about CSS is that it is an enormously flexible and efficient method for making everything on the screen look more consistent and when you want to change the format of something you can just change the CSS component and the whole look and feel of your graphics will change.

Here's the HTML portions of the code;

```
<!DOCTYPE html>
<meta charset="utf-8">
<style>

    The CSS is in here

</style>
<body>
<script src="http://d3js.org/d3.v3.min.js"></script>

<script>

    The D3 JavaScript code is here

</script>
</body>
```

Compare it with the full code. It kind of looks like a wrapper for the CSS and JavaScript. You can see that it really doesn't boil down to much at all (that doesn't mean it's not important).

There are plenty of good options for adding additional HTML stuff into this very basic part for the file, but for what we're going to be doing, we really don't need to bother too much.

One thing probably worth mentioning is the line;

```
<script src="http://d3js.org/d3.v3.min.js"></script>
```

That's the line that identifies the file that needs to be loaded to get D3 up and running. In this case the file is sourced from the official d3.js repository on the internet (that way we are using

---

<sup>30</sup><http://en.wikipedia.org/wiki/HTML>

<sup>31</sup><http://en.wikipedia.org/wiki/Css>

the most up to date version). The D3 file is actually called `d3.v3.min.js` which may come as a bit of a surprise. That tells us that this is version 3 of the `d3.js` file (the `v3` part) which is an indication that it is separate from the `v2` release, which was superseded in late 2012. The other point to note is that this version of `d3.js` is the minimised version (hence `min`). This means that any extraneous information has been removed from the file to make it quicker to load.

The two parts that we left out are the CSS and the D3 JavaScript.

## CSS

The CSS is as follows;

```
body { font: 12px Arial; }

path {
    stroke: steelblue;
    stroke-width: 2;
    fill: none;
}

.axis path,
.axis line {
    fill: none;
    stroke: grey;
    stroke-width: 1;
    shape-rendering: crispEdges;
}
```

Cascading Style Sheets give you control over the look / feel / presentation of web content. The idea is to define a set of properties to objects in the web page.

They are made up of ‘rules’. Each rule has a ‘selector’ and a ‘declaration’ and each declaration has a property and a value (or a group of properties and values).

For instance in the example code for this web page we have the following rule;

```
body { font: 12px Arial;}
```

`body` is the selector. This tells you that on the web page, this rule will apply to the ‘body’ of the page. This actually applies to all the portions of the web page that are contained in the ‘body’ portion of the HTML code (everything between `<body>` and `</body>` in the HTML bit). `{ font: 12px Arial; }` is the declaration portion of the rule. It only has the one declaration which is the bit that is in between the curly braces. So `font: 12px Arial;` is the declaration. The property is `font:` and the value is `12px Arial;`. This tells the web page that the font that appears in the body of the web page will be in 12 px Arial.

What about the bit that’s like;

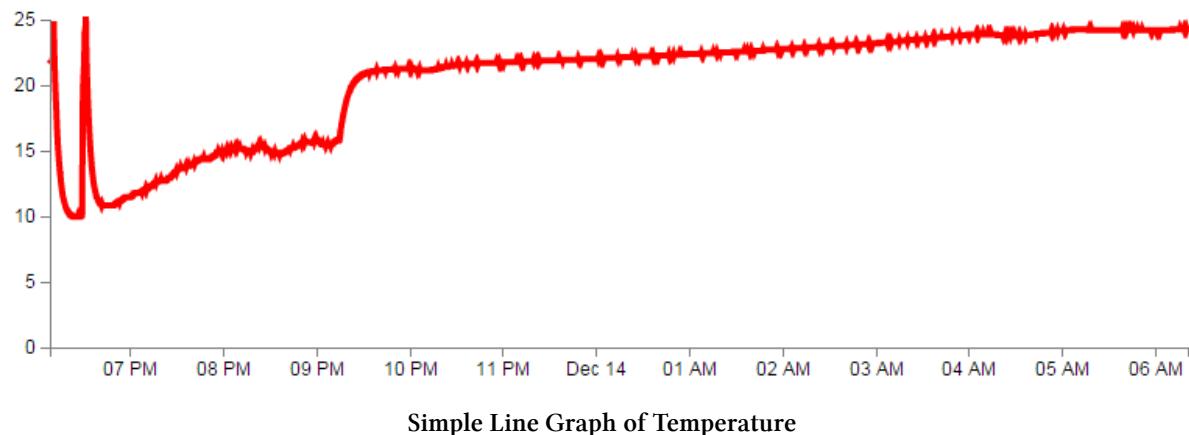
```
path {
  stroke: steelblue;
  stroke-width: 2;
  fill: none;
}
```

Well, the whole thing is one rule, ‘path’ is the selector. In this case, ‘path’ is referring to a line in the D3 drawing nomenclature.

For that selector there are three declarations. They give values for the properties of ‘stroke’ (in this case colour), ‘stroke-width’ (the width of the line) and ‘fill’ (we can fill a path with a block of colour).

We could test the changes by editing our file to show a thicker line in a different colour as so;

```
path {
  stroke: red;
  stroke-width: 4;
  fill: none;
}
```



By all means have a play with the settings and see what the end result is.

## JavaScript and d3.js

JavaScript<sup>32</sup> is what’s called a ‘scripting language’. It is the code that will be contained inside the HTML file that will make D3 do all its fanciness. In fact, D3 is a JavaScript Library. JavaScript is the language D3 is written in.

Knowing a little bit about this would be really good, but to be perfectly honest, I didn’t know anything about it before I started writing ‘D3 Tips and Tricks<sup>33</sup>’. I read a book along the way ([JavaScript: The Missing Manual<sup>34</sup>](#) from O’Reilly) and that helped with context, but the examples

<sup>32</sup><http://en.wikipedia.org/wiki/JavaScript>

<sup>33</sup><https://leanpub.com/D3-Tips-and-Tricks>

<sup>34</sup><http://shop.oreilly.com/product/9780596515898.do>

that are available for D3 graphics are understandable, and with a bit of trial and error, you can figure out what's going on.

The D3 JavaScript part of the code is as follows;

```
// Set the dimensions of the canvas / graph
var margin = {top: 30, right: 20, bottom: 30, left: 50},
    width = 800 - margin.left - margin.right,
    height = 270 - margin.top - margin.bottom;

// Parse the date / time
var parseDate = d3.time.format("%Y-%m-%d %H:%M:%S").parse;

// Set the ranges
var x = d3.time.scale().range([0, width]);
var y = d3.scale.linear().range([height, 0]);

// Define the axes
var xAxis = d3.svg.axis().scale(x)
    .orient("bottom");

var yAxis = d3.svg.axis().scale(y)
    .orient("left").ticks(5);

// Define the line
var valueline = d3.svg.line()
    .x(function(d) { return x(d.dtg); })
    .y(function(d) { return y(d.temperature); });

// Adds the svg canvas
var svg = d3.select("body")
    .append("svg")
        .attr("width", width + margin.left + margin.right)
        .attr("height", height + margin.top + margin.bottom)
    .append("g")
        .attr("transform",
            "translate(" + margin.left + "," + margin.top + ")");

// Get the data
<?php echo "data=".json_data.";" ?>
data.forEach(function(d) {
    d.dtg = parseDate(d.dtg);
    d.temperature = +d.temperature;
});

// Scale the range of the data
x.domain(d3.extent(data, function(d) { return d.dtg; }));
```

```

y.domain([0, d3.max(data, function(d) { return d.temperature; })]);

// Add the valueline path.
svg.append("path")
    .attr("d", valueline(data));

// Add the X Axis
svg.append("g")
    .attr("class", "x axis")
    .attr("transform", "translate(0," + height + ")")
    .call(xAxis);

// Add the Y Axis
svg.append("g")
    .attr("class", "y axis")
    .call(yAxis);

```

There's quite a bit of detail in the code, but it's not so long that we can't work out what's doing what.

Let's examine the blocks bit by bit to get a feel for it.

## **Setting up the margins and the graph area.**

The part of the code responsible for defining the image area (or the area where the graph and associated bits and pieces is placed ) is this part.

```

var margin = {top: 30, right: 20, bottom: 30, left: 50},
    width = 800 - margin.left - margin.right,
    height = 270 - margin.top - margin.bottom;

```

This is really (*really*) well explained on Mike Bostock's page on margin conventions here [http://bl.ocks.org/3019563<sup>35</sup>](http://bl.ocks.org/3019563), but at the risk of confusing you here's my crude take on it.

The first line defines the four margins which surround the block where the graph (as an object) is positioned.

```
var margin = {top: 30, right: 20, bottom: 30, left: 50},
```

So there will be a border of 30 pixels at the top, 20 at the right and 30 and 50 at the bottom and left respectively. Now the cool thing about how these are set up is that they use an array to define everything. That means if you want to do calculations in the JavaScript later, you don't need to put the numbers in, you just use the variable that has been set up. In this case margin.right = 20!

So when we go to the next line;

---

<sup>35</sup><http://bl.ocks.org/3019563>

```
width = 800 - margin.left - margin.right,
```

the width of the inner block of the canvas where the graph will be drawn is 600 pixels – margin.left – margin.right or 600-50-20 or 530 pixels wide. Of course now you have another variable ‘width’ that we can use later in the code.

Obviously the same treatment is given to height.

Another cool thing about all of this is that just because you appear to have defined separate areas for the graph and the margins, the whole area in there is available for use. It just makes it really useful to have areas designated for the axis labels and graph labels without having to juggle them and the graph proper at the same time.

That is the really cool part of this whole business. D3 is running in the background looking after the drawing of the objects, while you get to concentrate on how the data looks without too much maths!

## Getting the Data

We’re going to jump forward a little bit here to the bit of the JavaScript code that loads the data for the graph.

I’m going to go out of the sequence of the code, because if you know what the data is that you’re using, it will make explaining some of the other functions that are coming up much easier.

The section that grabs the data is this bit.

```
<?php echo "data=".json_data.";" ?>
data.forEach(function(d) {
    d.dtg = parseDate(d.dtg);
    d.temperature = +d.temperature;
});
```

There’s lots of different ways that we can get data into our web page to turn into graphics. And the method that you’ll want to use will probably depend more on the format that the data is in than the mechanism you want to use for importing.

In our case we actually use our old friend PHP to declare it. The line...

```
<?php echo "data=".json_data.";" ?>
```

...uses PHP to inject a line of code into our JavaScript that declares the data variable and its values. This is actually a fairly pivotal moment in the understanding of how PHP on a web page works. On the server (our Raspberry Pi) the script contains the line;

```
<?php echo "data=".json_data.";" ?>
```

But, when a client (the web browser) accesses the script, that line is executed as code and the information that is pushed to the client is

```
data=[{"dtg": "2014-12-13 18:08:08", "temperature": "22"}, ...];
```

The `data=` portion is ‘echo’ed to the script directly and the `dtg` and `temperature` information is from the variable that we declared earlier in our initial large PHP block when we queried our database.

After declaring what our data is, we need to do a little housekeeping to make sure that the values are suitable for the script.

```
data.forEach(function(d) {
    d.dtg = parseDate(d.dtg);
    d.temperature = +d.temperature;
});
```

This block of code simply ensures that all the numeric values that are pulled out of the data are set and formatted correctly. The first line sets the data variable that is being dealt with (called slightly confusingly ‘`data`’) and tells the block of code that, for each group within the ‘`data`’ array it should carry out a function on it. That function is designated ‘`d`’.

```
data.forEach(function(d) {
```

The information in the array can be considered as being stored in rows. Each row consists of two values: one value for ‘`dtg`’ and another value for ‘`temperature`’.

The function is pulling out values of ‘`dtg`’ and ‘`temperature`’ one row at a time.

Each time it gets a value of ‘`dtg`’ and ‘`temperature`’ it carries out the following operations;

```
    d.dtg = parseDate(d.dtg);
```

For this specific value of date/time being looked at (`d.dtg`), `d3.js` changes it into a date/time format that is processed via a separate function ‘`parseDate`’. (The ‘`parseDate`’ function is defined in a separate part of the script, and we will examine that later.) For the moment, be satisfied that it takes the raw date information from the data in a specific row and converts it into a format that `D3` can then process. That value is then re-saved in the same variable space.

The next line then sets the ‘`temperature`’ variable to a numeric value (if it isn’t already) using the ‘`+`’ operator.

```
    d.temperature = +d.temperature;
```



This appears to be good practice when the format of the number being pulled out of the data may not mean that it is automatically recognised as a number. This line will ensure that it is.

So, at the end of that section of code, we have gone out and picked up our data and ensured that it is formatted in a way that the rest of the script can use correctly.

But anyway, let’s get back to figuring what the code is doing by jumping back to the end of the margins block.

## Formatting the Date / Time.

One of the glorious things about the World is that we all do things a bit differently. One of those things is how we refer to [dates and time](#)<sup>36</sup>.

In my neck of the woods, it's customary to write the date as day - month – year. E.g 23-12-2012. But in the United States the more common format would be 12-23-2012. Likewise, the data may be in formats that name the months or weekdays (E.g. January, Tuesday) or combine dates and time together (E.g. 2012-12-23 15:45:32). So, if we were to attempt to try to load in some data and to try and get D3 to recognise it as date / time information, we really need to tell it what format the date / time is in.



### Does Time Matter?

You might be asking yourself “What’s the point?” All you want to do is give it a number and it can sort it out somehow. Well, that is true, but if you want to really bring out the best in your data and to keep maximum flexibility in representing it on the screen, you will want D3 to play to its strengths. And one of those is being able to adjust dynamically with variable time values.

The line in the JavaScript that parses the time is the following;

```
var parseDate = d3.time.format("%Y-%m-%d %H:%M:%S").parse;
```

This line is used when the `data.forEach(function(d)` portion of the code (that we looked at a couple of pages back) used `d.dtg = parseDate(d.dtg);` as a way to take a date/time in a specific format and to get it recognised by D3. In effect it said “*take this value that is supposedly a date and make it into a value I can work with*”.

The function used is the `d3.time.format(specifier)` function where the specifier in this case is the mysterious combination of characters `%Y-%m-%d %H:%M:%S`. The good news is that these are just a combination of directives specific for the type of date we are presenting.

The % signs are used as prefixes to each separate format type and the ‘-’ (minus) signs are literals for the actual ‘-’ (minus) signs that appear in the date to be parsed.

The Y refers to the year with century as a decimal number.

The m refers to the month as a decimal number [01,12].

The d refers to a zero-padded day of the month as a decimal number [01,31].

The H refers to the hour (24-hour clock) as a decimal number [00,23].

The M refers to the number of minutes as a decimal number [00,59].

The S refers to seconds as a decimal number [00,61].

And the y refers to the year (without the centuries) as a decimal number.

If we look at a subset of the data from our database we see that indeed, the dates therein are formatted in this way.

---

<sup>36</sup>[http://en.wikipedia.org/wiki/Date\\_format\\_by\\_country](http://en.wikipedia.org/wiki/Date_format_by_country)

That's all well and good, but what if your data isn't formatted exactly like that?

Good news. There are multiple different formatters for different ways of telling time and you get to pick and choose which one you want. Check out the Time Formatting page on the D3 Wiki for a the authoritative list and some great detail, but the following is the list of currently available formatters (from the d3 wiki);

- %a - abbreviated weekday name.
- %A - full weekday name.
- %b - abbreviated month name.
- %B - full month name.
- %c - date and time, as “%a %b %e %H:%M:%S %Y”.
- %d - zero-padded day of the month as a decimal number [01,31].
- %e - space-padded day of the month as a decimal number [ 1,31].
- %H - hour (24-hour clock) as a decimal number [00,23].
- %I - hour (12-hour clock) as a decimal number [01,12].
- %j - day of the year as a decimal number [001,366].
- %m - month as a decimal number [01,12].
- %M - minute as a decimal number [00,59].
- %p - either AM or PM.
- %S - second as a decimal number [00,61].
- %U - week number of the year (Sunday as the first day of the week) as a decimal number [00,53].
- %w - weekday as a decimal number [0(Sunday),6].
- %W - week number of the year (Monday as the first day of the week) as a decimal number [00,53].
- %x - date, as “%m/%d/%y”.
- %X - time, as “%H:%M:%S”.
- %y - year without century as a decimal number [00,99].
- %Y - year with century as a decimal number.
- %Z - time zone offset, such as “-0700”.
- There is also a a literal “%” character that can be presented by using double % signs.

## Setting Scales Domains and Ranges

This is another example where, if you set it up right, D3 will look after you forever.



### Scales, Ranges and the Ah Ha!" moment.

The “Ah Ha!” moment for me in understanding ranges and scales was after reading Jerome Cukier’s great page on ‘[d3:scales and color](#)<sup>37</sup>’. I thoroughly recommend you read it (and plenty more of the great work by Jerome) as he really does nail the description in my humble opinion. I will put my own description down here, but if it doesn’t seem clear, head on over to Jerome’s page.

From our basic web page we have now moved to the section that includes the following lines;

---

<sup>37</sup><http://www.jeromecukier.net/blog/2011/08/11/d3-scales-and-color/>

```
var x = d3.time.scale().range([0, width]);
var y = d3.scale.linear().range([height, 0]);
```

The purpose of these portions of the script is to ensure that the data we ingest fits onto our graph correctly. Since we have two different types of data (date/time and numeric values) they need to be treated separately (but they do essentially the same job). To examine this whole concept of scales, domains and ranges properly, we will also move slightly out of sequence and (in conjunction with the earlier scale statements) take a look at the lines of script that occur later and set the domain. They are as follows;

```
x.domain(d3.extent(data, function(d) { return d.dtg; }));
y.domain([0, d3.max(data, function(d) { return d.temperature; })]);
```

The idea of scaling is to take the values of data that we have and to fit them into the space we have available.

First we make sure that any quantity we specify on the x axis fits onto our graph.

```
var x = d3.time.scale().range([0, width]);
```

Here we set our variable that will tell D3 where to draw something on the x axis. By using the d3.time.scale() function we make sure that D3 knows to treat the values as date / time entities (with all their ingrained peculiarities). Then we specify the range that those values will cover (.range) and we specify the range as being from 0 to the width of our graphing area (See! Setting those variables for margins and widths are starting to pay off now!).

Then we do the same for the Y axis.

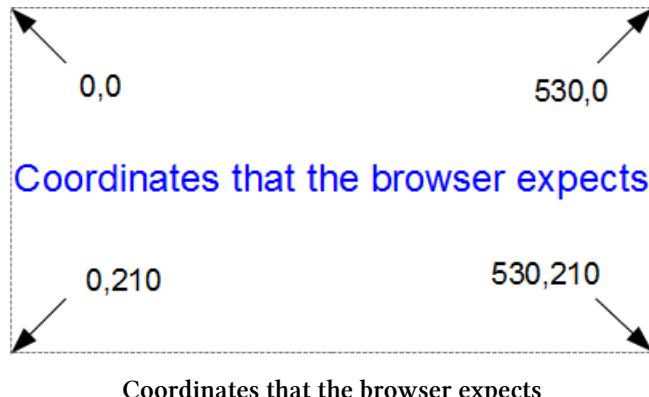
```
var y = d3.scale.linear().range([height, 0]);
```

There's a different function call (d3.scale.linear()) but the .range setting is still there.

Now hang on, what's going on with the [height, 0] part in y axis scale statement? The astute amongst you will note that for the time scale we set the range as [0, width] but for this one ([height, 0]) the values look backwards.

Well spotted.

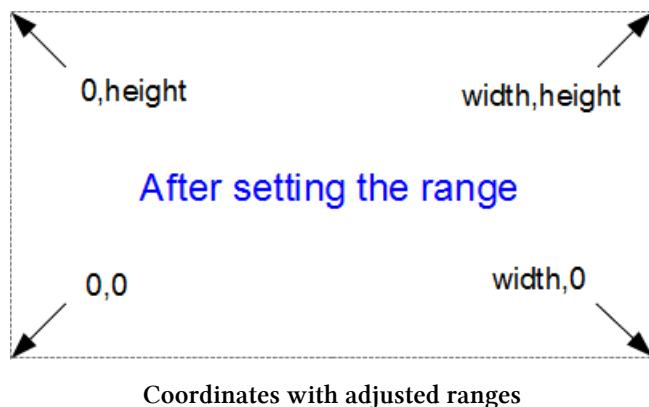
This is all to do with how the screen is laid out and referenced. Take a look at the following diagram showing how the coordinates for drawing on our screen work;



The top left hand of the screen is the origin or 0,0 point and as we go left or down the corresponding x and y values increase to the full values defined by height and width.

That's good enough for the time values on the x axis that will start at lower values and increase, but for the values on the y axis we're trying to go against the flow. We want the low values to be at the bottom and the high values to be at the top.

No problem. We just tell D3 via the statement `y = d3.scale.linear().range([height, 0]);` that the larger values (height) are at the low end of the screen (at the top) and the low values are at the bottom (as you most probably will have guessed by this stage, the `.range` statement uses the format `.range([closer_to_the_origin, further_from_the_origin])`). So when we put the height variable first, that is now associated at the top of the screen.



We've scaled our data to the graph size and ensured that the range of values is set appropriately. What's with the domain part that was in this section's title?

Come on, you remember this little piece of script don't you?

```
x.domain(d3.extent(data, function(d) { return d.dtg; }));
y.domain([0, d3.max(data, function(d) { return d.temperature; })]);
```

While it exists in a separate part of the file from the scale / range part, it is certainly linked.

That's because there's something missing from what we have been describing so far with the set up of the data ranges for the graphs. We haven't actually told D3 what the range of the data is.

So, the `.domain` function is designed to let D3 know what the scope of the data will be. This is what is then passed to the `scale` function.

Looking at the first part that is setting up the x axis values, it is saying that the domain for the x axis values will be determined by the `d3.extent` function which in turn is acting on a separate function which looks through all the ‘date’ values that occur in the ‘data’ array. In this case the `.extent` function returns the minimum and maximum value in the given array.

- `function(d) { return d.dtg; }` returns all the ‘dtg’ values in ‘data’. This is then passed to...
- The `.extent` function that finds the maximum and minimum values in the array and then...
- The `.domain` function which returns those maximum and minimum values to D3 as the range for the x axis.

Pretty neat really. At first you might think it was overly complex, but breaking the function down into these components allows additional functionality with differing scales, values and quantities. In short, don’t sweat it. It’s a good thing.

## Setting up the Axes

Now we come to our next piece of code;

```
var xAxis = d3.svg.axis().scale(x)
    .orient("bottom");

var yAxis = d3.svg.axis().scale(y)
    .orient("left").ticks(5);
```

I’ve included both the x and y axes because they carry out the formatting in very similar ways. It’s worth noting that this is not the point where the axes get drawn. That occurs later in the script.

D3 has its own axis component that aims to take the fuss out of setting up and displaying the axes. So it includes a number of configurable options.

Looking first at the x axis;

```
var xAxis = d3.svg.axis().scale(x)
    .orient("bottom");
```

The axis function is called with `d3.svg.axis()`. Then the scale is set using the x values that we set up in the scales, ranges and domains section using `.scale(x)`. Then we tell the graph to orientate itself to the bottom of the graph `.orient("bottom")`.

The code that formats the y axis is pretty similar;

```
var yAxis = d3.svg.axis().scale(y)
    .orient("left").ticks(5);
```

The only significant difference is that we define the number of ticks we would like to have on that axis (for the x axis we just let D3 pick an appropriate number)

## Adding data to the line function

We're getting towards the end of our journey through the script now. The next step is to get the information from the array 'data' and to place it in a new array that consists of a set of coordinates that we are going to plot.

```
var valueline = d3.svg.line()
    .x(function(d) { return x(d.dtg); })
    .y(function(d) { return y(d.temperature); });
```

I'm aware that the statement above may be somewhat ambiguous. You would be justified in thinking that we already had the data stored and ready to go. But that's not *strictly* correct.

What we have is data in a raw format, we have added pieces of code that will allow the data to be adjusted for scale and range to fit in the area that we want to draw, but we haven't actually taken our raw data and adjusted it for our desired coordinates. That's what the code above does.

The main function that gets used here is the `d3.svg.line()` function<sup>38</sup>. This function uses accessor functions to store the appropriate information in the right area and in the case above they use the `x` and `y` accessors (that would be the bits that are `.x` and `.y`). The `d3.svg.line()` function is called a 'path generator' and this is an indication that it can carry out some pretty clever things on its own accord. But in essence its job is to assign a set of coordinates in a form that can be used to draw a line.

Each time this line function is called on, it will go through the data and will assign coordinates to 'dtg' and 'temperature' pairs using the 'x' and 'y' functions that we set up earlier (which of course are responsible for scaling and setting the correct range / domain).

Of course, it doesn't get the data all by itself, we still need to actually call the `valueline` function with 'data' as the source to act on. But never fear, that's coming up soon.

## Adding the SVG area.

As the title states, the next piece of script forms and adds the space that D3 will then use to draw on.

---

<sup>38</sup><https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-line>

```
var svg = d3.select("body")
.append("svg")
.attr("width", width + margin.left + margin.right)
.attr("height", height + margin.top + margin.bottom)
.append("g")
.attr("transform",
"translate(" + margin.left + ", " + margin.top + ")");
```

So what exactly does that all mean?

Well D3 needs to be able to have a space defined for it to draw things. When you define the space it's going to use, you can also give it an identifying name and attributes.

In the example we're using here, we are ‘appending’ an SVG element (an element that we are going to draw things on) to the `<body>` element of the HTML page.



In human talk that means that on the web page and bounded by the `<body>` tag that we saw in the HTML part, we will have an area to draw on. That area will be ‘width’ plus the left and right margins wide and ‘height’ plus the top and bottom margins wide.

We also add an element ‘g’ that is referenced to the top left corner of the actual graph area on the canvas. ‘g’ is actually a grouping element in the sense that it is normally used for grouping together several related elements. So in this case those grouped elements will have a common reference.

Interesting things to note about the code. The `.attr("stuff in here")` parts are attributes of the appended elements they are part of.

For instance;

```
.append("svg")
.attr("width", width + margin.left + margin.right)
.attr("height", height + margin.top + margin.bottom)
```

tells us that the ‘svg’ element has a “width” of `width + margin.left + margin.right` and the “height” of `height + margin.top + margin.bottom`.

Likewise...

```
.append("g")
.attr("transform",
"translate(" + margin.left + ", " + margin.top + ")");
```

tells us that the element “g” has been transformed by moving(translating) to the point `margin.left`, `margin.top`. Or to the top left of the graph space proper. This way when we tell something to be drawn on our canvas, we can use the reference point “g” to make sure everything is in the right place.

## Actually Drawing Something!

Up until now we have spent a lot of time defining, loading and setting up. Good news! We're about to finally draw something!

We jump lightly over some of the code that we have already explained and land on the part that draws the line.

```
svg.append("path")
    .attr("d", valueline(data));
```

This area occurs in the part of the code that has the data loaded and ready for action.

The `svg.append("path")` portion adds a new path element. A path element represents a shape that can be manipulated in lots of different ways (see more here: [http://www.w3.org/TR/SVG/paths.html<sup>39</sup>](http://www.w3.org/TR/SVG/paths.html)). In this case it inherits the 'path' styles from the CSS section and on the following line (`.attr("d", valueline(data));`) we add the attribute "d".

This is an attributer that stands for 'path data' and sure enough the `valueline(data)` portion of the script passes the 'valueline' array (with its x and y coordinates) to the path element. This then creates a `svg` element which is a path going from one set of 'valueline' coordinates to another.

Then we get to draw in the axes;

```
svg.append("g")
    .attr("class", "x axis")
    .attr("transform", "translate(0, " + height + ")")
    .call(xAxis);

svg.append("g")
    .attr("class", "y axis")
    .call(yAxis);
```

We have covered the formatting of the axis components earlier. So this part is actually just about getting those components drawn onto our canvas.

Both axes start by being appended to the "g" group. Then each has its own classes applied for styling via CSS.

Feel free to mess about with these CSS styles to change the appearance of your axes.

On the x axis, we have a transform statement (`.attr("transform", "translate(0, " + height + ")")`). If you recall, our point of origin for drawing is in the top left hand corner. Therefore if we want our x axis to be on the bottom of the graph, we need to move (transform) it to the bottom by a set amount. The set amount in this case is the height of the graph proper (height).

The last part of the two sections of script (`.call(xAxis);` and `.call(yAxis);`) call the x and y axis functions and initiate the drawing action.

---

<sup>39</sup><http://www.w3.org/TR/SVG/paths.html>

## Wrap Up

Well that's it. In theory, you should now be a complete D3 ninja.

OK, perhaps a slight exaggeration. In fact there is a strong possibility that the information I have laid out here is at best borderline useful and at worst laden with evil practices and gross inaccuracies.

But look on the bright side. Irrespective of the nastiness of the way that any of it was accomplished or the inelegance of the code, if the picture drawn on the screen is relatively pretty, you can walk away with a smile. :-)

This section concludes a very basic description of one type of a graphic that can be built with D3. We will look at adding value to it in subsequent chapters.

Those with a smattering of knowledge of any of the topics I have butchered above (or below) are fully justified in feeling a large degree of righteous indignation. To those I say, please feel free to amend where practical and possible, but please bear in mind this was written from the point of view of someone with no experience in the topic and therefore try to keep any instructions at a level where a new entrant can step in.

# Single Temperature Measurement

This project will measure temperature using a single DS18B20 sensor. This project will use the waterproof version of the sensor since it has more potential practical applications.

## Measure

### Hardware required

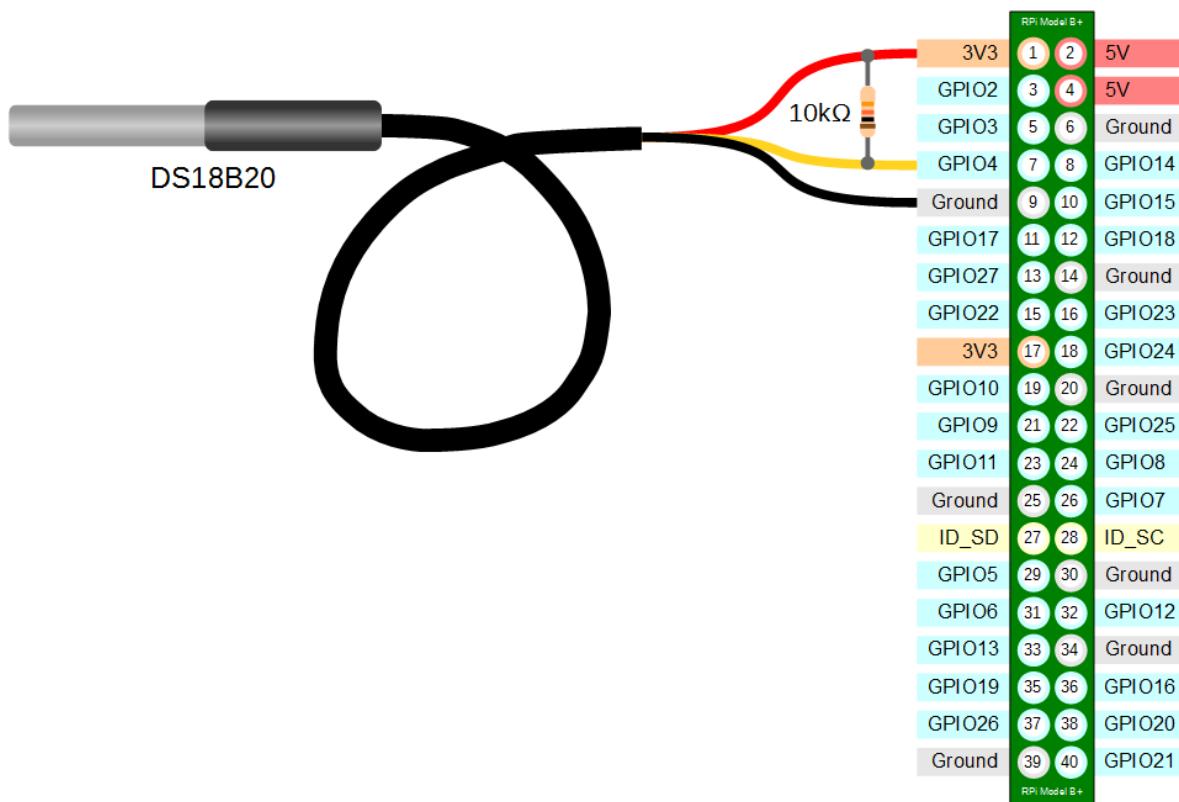
- DS18B20 sensor (the waterproof version)
- 10k Ohm resistor
- Jumper cables
- Cobbler Board
- Ribbon cable

## Connect

The DS18B20 sensor needs to be connected with the black wire to ground, the red wire to the 3V3 pin and the blue or yellow (some are blue and some are yellow) wire to the GPIO4 pin. A resistor between the value of 4.7k Ohms to 10k Ohms needs to be connected between the 3V3 and GPIO4 pins to act as a ‘pull-up’ resistor.

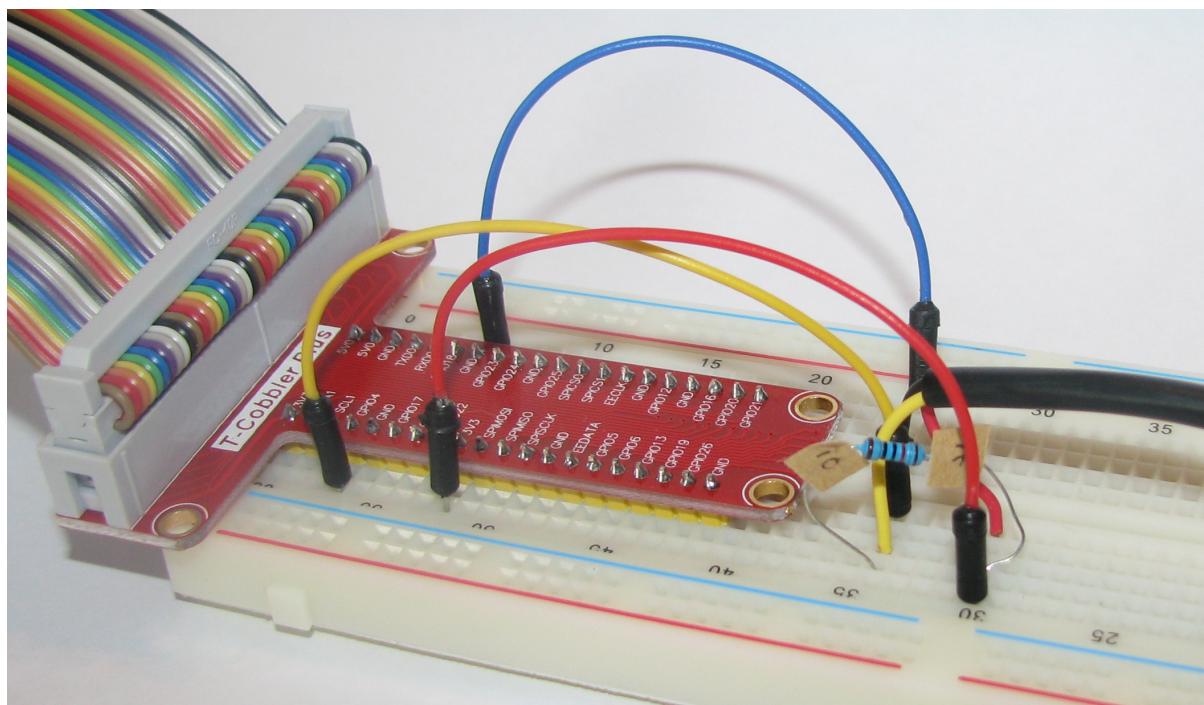
The Raspbian Operating System image that we are using only supports GPIO4 as a 1-Wire pin, so we need to ensure that this is the pin that we use for connecting our temperature sensor.

The following diagram is a simplified view of the connection.



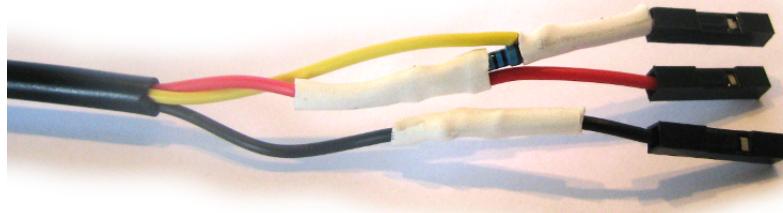
Single DS18B20 Connection

To connect the sensor practically can be achieved in a number of ways. You could use a Pi Cobbler break out connector mounted on a bread board connected to the GPIO pins.



Single DS18B20 Connection via Bread Board

Or we could build a minimal configuration that will plug directly onto the appropriate GPIO pins.



Minimal Single DS18B20 Connection

## Test

Because the Raspberry Pi acts like an embedded platform rather than a regular PC, it doesn't have a BIOS (Basic Input Output System) that goes through the various pieces of hardware when the Pi boots up and configures everything. Instead it has an optional text file named `config.txt`. This can be found in the `/boot` directory. To enable the Pi to use the GPIO pin to communicate with our temperature sensor we need to tell it to configure itself with the `w1-gpio` Onewire interface module.



Many thanks to 'Dan B' for pointing me in the right direction to get this sorted :-).

We can do this by editing the `/boot/config.txt` file using...

```
sudo nano /boot/config.txt
```

...and adding in the line...

```
dtoverlay=w1-gpio
```

...at the end of the file

After making this change we need to reboot our Pi to let the changes take effect;

```
sudo reboot
```

From the terminal as the 'pi' user run the command;

```
sudo modprobe w1-gpio
```

`modprobe w1-gpio` registers the new sensor connected to GPIO4 so that now the Raspberry Pi knows that there is a 1-Wire device connected to the GPIO connector (For more information on the `modprobe` command check out the [Glossary](#)).



`modprobe` is a Linux program used to add a loadable kernel module (LKM) to the Linux kernel or to remove a LKM from the kernel. It is commonly used to load drivers for automatically detected hardware.

Then run the command;

```
sudo modprobe w1-therm
```

`modprobe w1-therm` tells the Raspberry Pi to add the ability to measure temperature on the 1-Wire system.

Then we change into the `/sys/bus/w1/devices` directory and list the contents using the following commands;

```
cd /sys/bus/w1/devices  
ls
```

(For more information on the `cd` command check out the Glossary [here](#). Or to find out more about the `ls` command go [here](#))

This should list out the contents of the `/sys/bus/w1/devices` which should include a directory starting `28-`. The portion of the name following the `28-` is the unique serial number of the sensor.

We then change into that unique directory;

```
cd 28-xxxx (change xxxx to match the serial number of the directory)
```

We are then going to view the '`w1_slave`' file with the `cat` command using;

```
cat w1_slave
```



The [cat<sup>40</sup>](#) program takes the specified file (or files) and by default outputs the results to the screen (there are a multitude of different options for `cat`, more can be seen in the [Glossary](#)).

The output should look something like the following;

```
73 01 4b 46 7f ff 0d 10 41 : crc=41 YES
73 01 4b 46 7f ff 0d 10 41 t=23187
```

At the end of the first line we see a YES for a successful CRC check (CRC stands for Cyclic Redundancy Check, a good sign that things are going well). If we get a response like NO or FALSE or ERROR, it will be an indication that there is some kind of problem that needs addressing. Check the circuit connections and start troubleshooting.

At the end of the second line we can now find the current temperature. The `t=23187` is an indication that the temperature is 23.187 degrees Celsius (we need to divide the reported value by 1000).



To convert from degrees Celsius to degrees Fahrenheit, multiply by 9, then divide by 5, then add 32.

## Record

To record this data we will use a Python program that checks the sensor every minute and writes the temperature (with a time stamp) into our MySQL database.

### Database preparation

First we will set up our database table that will store our data.

Using the phpMyAdmin web interface that we set up, log on using the administrator (root) account and select the ‘measurements’ database that we created as part of the initial set-up.

The screenshot shows the phpMyAdmin interface on a local host. The database selected is 'measurements'. A modal window is open for creating a new table. The table name is 'temperature' and it has 'Number of columns: 2'. The 'Go' button is visible at the bottom right of the modal.

Create the MySQL Table

<sup>40</sup>[http://en.wikipedia.org/wiki/Cat\\_\(Unix\)](http://en.wikipedia.org/wiki/Cat_(Unix))

Enter in the name of the table and the number of columns that we are going to use for our measured values. In the screenshot above we can see that the name of the table is ‘temperature’ (how imaginative) and the number of columns is ‘2’.

We will use two columns so that we can store a temperature reading and the time it was recorded. Once we click on ‘Go’ we are presented with a list of options to configure our table’s columns. Don’t be intimidated by the number of options that are presented, we are going to keep the process as simple as practical.

For the first column we can enter the name of the ‘Column’ as ‘dtg’ (short for date time group) the ‘Type’ as ‘TIMESTAMP’ and the ‘Default’ value as ‘CURRENT\_TIMESTAMP’. For the second column we will enter the name ‘temperature’ and the ‘Type’ is ‘FLOAT’ (we won’t use a default value).

**Create Table**

**Table name:**

**Structure**

Column	dtg	temperature
Type	TIMESTAMP	FLOAT
Length/Values <sup>1</sup>		
Default <sup>2</sup>	CURRENT_TIMESTAMP	None

Configure the MySQL Table Columns

Scroll down a little and click on the ‘Save’ button and we’re done.

**Save** Or Add  column(s) **Go**

Save the MySQL Table Columns



## Why did we choose those particular settings for our table?

Our ‘dtg’ column needs to store a value of time that includes the date and the time, so either of the types ‘TIMESTAMP’ or ‘DATETIME’ would be suitable. Either of them stores the time in the format ‘YYYY-MM-DD HH:MM:SS’. The advantage of selecting TIMESTAMP in this case is that we can select the default value to be the current time which means that when we write our data to the table we only need to write the temperature value and the ‘dtg’ will be entered automatically for us. The disadvantage of using ‘TIMESTAMP’ is that it has a more limited range than DATETIME. TIMESTAMP can only have a range between ‘1970-01-01 00:00:01’ to ‘2038-01-19 03:14:07’.

Our temperature readings are generated (by our sensor) as an integer value that needs to be divided by 1000 to show degrees Centigrade. We could therefore store the value as an integer. However when we were selecting the data or in later processing we would then need to do the math to convert it to the correct value. It could be argued (successfully) that this would be a more efficient solution in terms of the amount of space taken to support the data on the Pi. However, I have a preference for storing the values as they would be used later and as a result we need to use a numerical format that supports numbers with decimal places. There are a range of options for defining the ranges for decimal numbers, but FLOAT allows us to ignore the options (at the expense of efficiency) and rely on our recorded values being somewhere between -3.402823466E+38 and 3.402823466E+38 (if our temperature falls outside those extremes we are in trouble).

## Record the temperature values

The following code (which is based on the code that is part of the great temperature sensing tutorial on Adafruit<sup>41</sup>) is a Python script which allows us to check the temperature reading from the sensor approximately every 10 seconds and write it to our database.

The full code can be found in the code samples bundled with this book (s\_temp.py).

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import os
import glob
import time
import MySQLdb as mdb

os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')

base_dir = '/sys/bus/w1/devices/'
device_folder = glob.glob(base_dir + '28*')[0]
```

<sup>41</sup><https://learn.adafruit.com/adafruits-raspberry-pi-lesson-11-ds18b20-temperature-sensing/>

```

device_file = device_folder + '/w1_slave'

def read_temp_raw():
    f = open(device_file, 'r')
    lines = f.readlines()
    f.close()
    return lines

def read_temp():
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find('t=')
    if equals_pos == -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0
        return temp_c

while True:

    try:
        pi_temp = read_temp()
        con = mdb.connect('localhost', \
                          'pi_insert', \
                          'xxxxxxxxxx', \
                          'measurements');

        cur = con.cursor()
        cur.execute("""INSERT INTO temperature(temperature) \
                      VALUES(%s)""", (pi_temp))
        con.commit()

    except mdb.Error, e:
        con.rollback()
        print "Error %d: %s" % (e.args[0],e.args[1])
        sys.exit(1)

    finally:
        if con:
            con.close()

    time.sleep(10)

```

This script can be saved in our home directory (/home/pi) and needs to be run as root (sudo) as follows;

```
sudo python s_temp.py
```

While we won't see much happening at the command line, if we use our web browser to go to the phpMyAdmin interface and select the 'measurements' database and then the 'temperature' table we will see a range of temperature measurements and their associated time of reading presented.

	+ Options	← T →	dtg	temperature
<input type="checkbox"/>	Edit  Inline Edit  Copy  Delete	2014-12-13 18:08:08	22	
<input type="checkbox"/>	Edit  Inline Edit  Copy  Delete	2014-12-13 18:08:19	21.937	
<input type="checkbox"/>	Edit  Inline Edit  Copy  Delete	2014-12-13 18:08:30	21.937	
<input type="checkbox"/>	Edit  Inline Edit  Copy  Delete	2014-12-13 18:08:41	21.937	
<input type="checkbox"/>	Edit  Inline Edit  Copy  Delete	2014-12-13 18:08:52	21.937	
<input type="checkbox"/>	Edit  Inline Edit  Copy  Delete	2014-12-13 18:09:03	21.937	
<input type="checkbox"/>	Edit  Inline Edit  Copy  Delete	2014-12-13 18:09:15	21.937	
<input type="checkbox"/>	Edit  Inline Edit  Copy  Delete	2014-12-13 18:09:26	21.937	
<input type="checkbox"/>	Edit  Inline Edit  Copy  Delete	2014-12-13 18:09:37	22	
<input type="checkbox"/>	Edit  Inline Edit  Copy  Delete	2014-12-13 18:09:48	22	
<input type="checkbox"/>	Edit  Inline Edit  Copy  Delete	2014-12-13 18:09:59	22	
<input type="checkbox"/>	Edit  Inline Edit  Copy  Delete	2014-12-13 18:10:10	22	
<input type="checkbox"/>	Edit  Inline Edit  Copy  Delete	2014-12-13 18:10:21	24.5	
<input type="checkbox"/>	Edit  Inline Edit  Copy  Delete	2014-12-13 18:10:32	24.937	

Save the MySQL Table Columns

## Code Explanation

The script starts by importing the modules that it's going to use for the process of reading and recording the temperature measurements;

```
import os
import glob
import time
import MySQLdb as mdb
```



Python code in one module gains access to the code in another module by the process of importing it. The `import` statement invokes the process and combines two operations; it searches for the named module, then it binds the results of that search to a name in the local scope.

The program then issues the `modprobe` commands that start the interface to the sensor;

```
os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')
```

Then we need to find the file (`w1_slave`) where the readings are being recorded in much the same way that we did it manually earlier;

```
base_dir = '/sys/bus/w1/devices/'
device_folder = glob.glob(base_dir + '28*')[0]
device_file = device_folder + '/w1_slave'
```

We then set the function for reading the temperature in a ‘raw’ form from the `w1_slave` file using the `read_temp_raw` function that fetches the two lines of messaging from the interface.

```
def read_temp_raw():
    f = open(device_file, 'r')
    lines = f.readlines()
    f.close()
    return lines
```

The `read_temp` function is then declared which checks for bad messages and keeps reading until it gets a message with ‘YES’ on end of the first line. Then the function returns the value of the temperature in degrees C.

```
def read_temp():
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0
    return temp_c
```

From here we enter into a while loop to continually read the temperature and insert the value into the MySQL database;

```

while True:

    try:
        pi_temp = read_temp()
        con = mdb.connect('localhost', \
                          'pi_insert', \
                          'xxxxxxxxxx', \
                          'measurements');
        cur = con.cursor()
        cur.execute("""INSERT INTO temperature(temperature) \
                      VALUES(%s)""", (pi_temp))
        con.commit()

    except mdb.Error, e:
        con.rollback()
        print "Error %d: %s" % (e.args[0],e.args[1])
        sys.exit(1)

    finally:
        if con:
            con.close()

    time.sleep(10)

```

The `while` statement takes an expression and executes the loop body while the expression evaluates to (boolean) “true”. `True` executes the loop body indefinitely. Note that most languages usually have some way of breaking out of the loop early. In the case of Python it’s the `break` statement (not that it’s used here).

## Explore

This section has a working solution for presenting temperature data but is a simple representation and is intended to provide a starting point for the display of data from a measurement process. Our data display techniques will become more advanced as we work out different things to measure. In the mean time, enjoy this simple effort.

## The Code

The following code is a PHP file that we can place on our Raspberry Pi’s web server (in the `/var/www` directory) that will allow us to view all of the results that have been recorded in the temperature directory on a graph;



This is the same code that is used in the set-up description and as such I won’t repeat the explanation of the code. The full code can be found in the code samples bundled with this book (`s_temp.php`).

```
<?php
$hostname = 'localhost';
$username = 'pi_select';
$password = 'xxxxxxxxxx';

try {
    $dbh = new PDO("mysql:host=$hostname;dbname=measurements",
                   $username, $password);

    /*** The SQL SELECT statement ***/
    $sth = $dbh->prepare(
        "SELECT `dtg`, `temperature` FROM `temperature`"
    );
    $sth->execute();

    /* Fetch all of the remaining rows in the result set */
    $result = $sth->fetchAll(PDO::FETCH_ASSOC);

    /*** close the database connection ***/
    $dbh = null;
}

catch(PDOException $e)
{
    echo $e->getMessage();
}

$json_data = json_encode($result);
?>

<!DOCTYPE html>
<meta charset="utf-8">
<style> /* set the CSS */

body { font: 12px Arial; }

path {
    stroke: steelblue;
    stroke-width: 2;
    fill: none;
}

.axis path,
.axis line {
    fill: none;
```

```
stroke: grey;
stroke-width: 1;
shape-rendering: crispEdges;
}

</style>
<body>

<!-- load the d3.js library -->
<script src="http://d3js.org/d3.v3.min.js"></script>

<script>

// Set the dimensions of the canvas / graph
var margin = {top: 30, right: 20, bottom: 30, left: 50},
    width = 800 - margin.left - margin.right,
    height = 270 - margin.top - margin.bottom;

// Parse the date / time
var parseDate = d3.time.format("%Y-%m-%d %H:%M:%S").parse;

// Set the ranges
var x = d3.time.scale().range([0, width]);
var y = d3.scale.linear().range([height, 0]);

// Define the axes
var xAxis = d3.svg.axis().scale(x)
    .orient("bottom");

var yAxis = d3.svg.axis().scale(y)
    .orient("left").ticks(5);

// Define the line
var valueline = d3.svg.line()
    .x(function(d) { return x(d.dtg); })
    .y(function(d) { return y(d.temperature); });

// Adds the svg canvas
var svg = d3.select("body")
    .append("svg")
        .attr("width", width + margin.left + margin.right)
        .attr("height", height + margin.top + margin.bottom)
    .append("g")
        .attr("transform",
              "translate(" + margin.left + "," + margin.top + ")");


```

```

// Get the data
<?php echo "data=".json_data." ?>
data.forEach(function(d) {
    d.dtg = parseDate(d.dtg);
    d.temperature = +d.temperature;
});

// Scale the range of the data
x.domain(d3.extent(data, function(d) { return d.dtg; }));
y.domain([0, d3.max(data, function(d) { return d.temperature; })]);

// Add the valueline path.
svg.append("path")
    .attr("d", valueline(data));

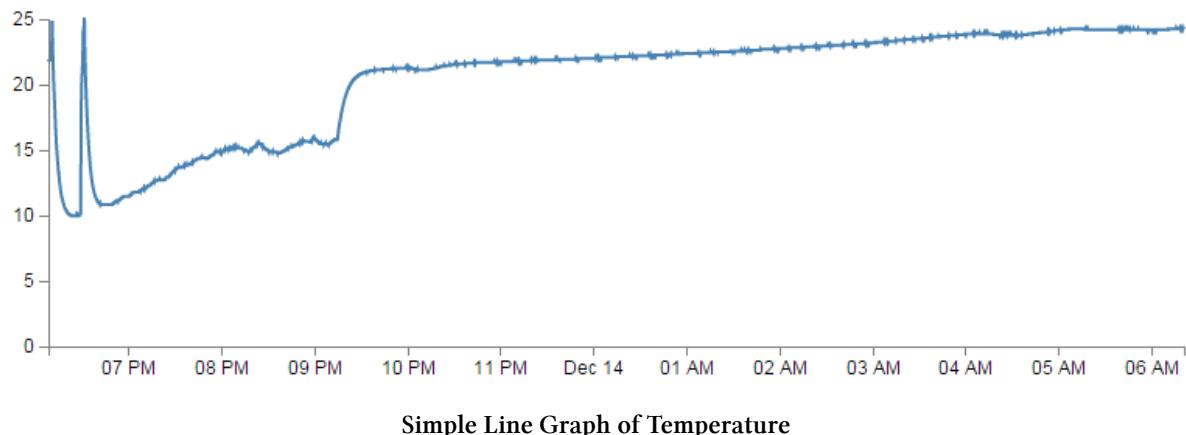
// Add the X Axis
svg.append("g")
    .attr("class", "x axis")
    .attr("transform", "translate(0," + height + ")")
    .call(xAxis);

// Add the Y Axis
svg.append("g")
    .attr("class", "y axis")
    .call(yAxis);

</script>
</body>

```

The graph that will look a little like this (except the data will be different of course).



This is a *VERY* simple graph (i.e, there is no title, labeling of axis or any real embellishment) and as such it has some limitations. For example it will automatically want to display *ALL* the recorded temperature data in the database. We might initially think that this would be a good

thing to do, but in this case there are over 3000 recordings trying to be displayed on a graph that is less than 800 pixels wide. Not only are we not going to see as much detail as could be possible, but the web browser can only cope with a finite amount of data being crammed into it. Eventually it will break.

So as an addendum to the code above we shall look at making a change to our database query to make a slightly more rational selection of data.

## Different MySQL Selection Options

Currently our SELECT statement looks like the following:

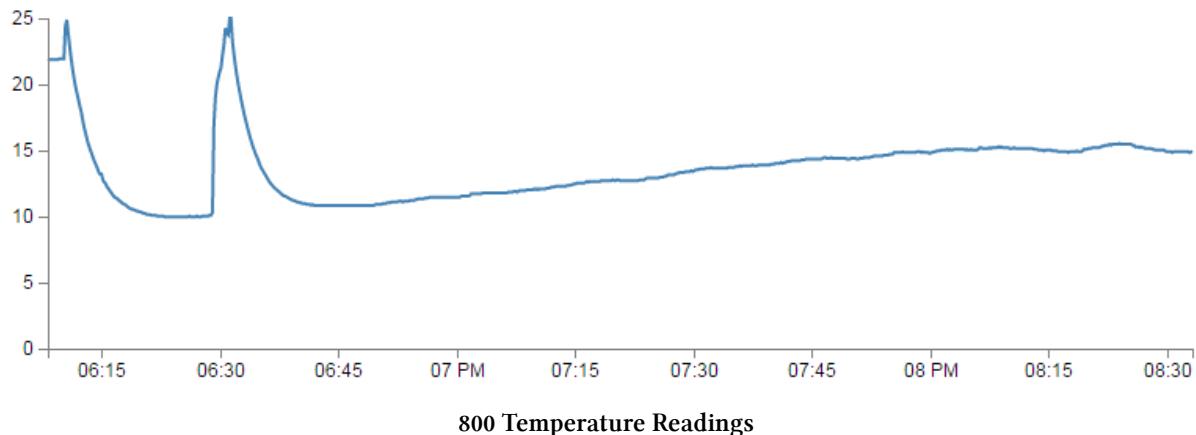
```
SELECT `dtg`, `temperature` FROM `temperature`
```

As described earlier, this query is telling the database to SELECT our date/time data (from the dtg column) and the temperature values (from the temperature column) FROM the table temperature. If there are 4 entries in the database, the query will return 4 readings. If there is 400,000 entries in the database, it will return 400,000 readings.

We can limit the number of returned rows to 800 with the query as follows;

```
SELECT `dtg`, `temperature` FROM `temperature` LIMIT 0 , 800
```

This adds in the **LIMIT 0,800** specifier which returns 800 rows starting at row ‘0’.

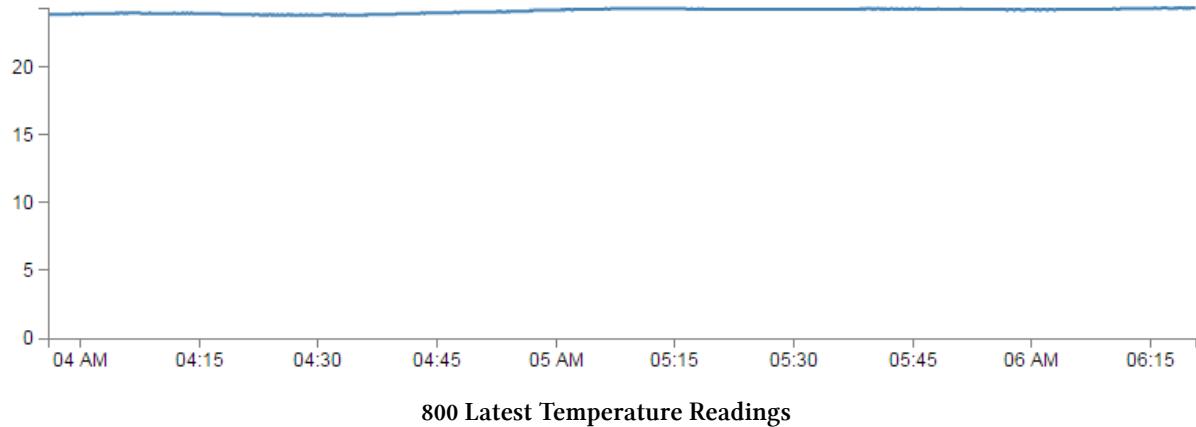


However, this is probably not satisfactory as in the case of the data we have recorded in our Python script, there is only a gap of 10 seconds or so between readings. This restricts us to just over 2 hours worth of recording and the values start when the recording started, so our returned values will never change.

We can improve on this by sorting the returned values and therefore take the most recent ones with the following query;

```
SELECT `dtg`, `temperature`
FROM `temperature`
ORDER BY `dtg` DESC
LIMIT 0,800
```

Here we order the returned rows by dtg descending. Which means the most recent date/time value is the one at row '0' and we will capture the most recent two hours worth of readings every time we run the query.



Of course in the case of the data that was in the database, this is a fairly 'booring' set with little variation.

We now have an efficient number of data points being returned to the web browser to graph, but we only see two hours worth of data. It could be argued that the fidelity of reading every 10 seconds is a little high, and if we were to return values for every minute that would be adequate. As an interesting exercise we can return that type of information using our current data set by using a slightly more sophisticated MySQL query;

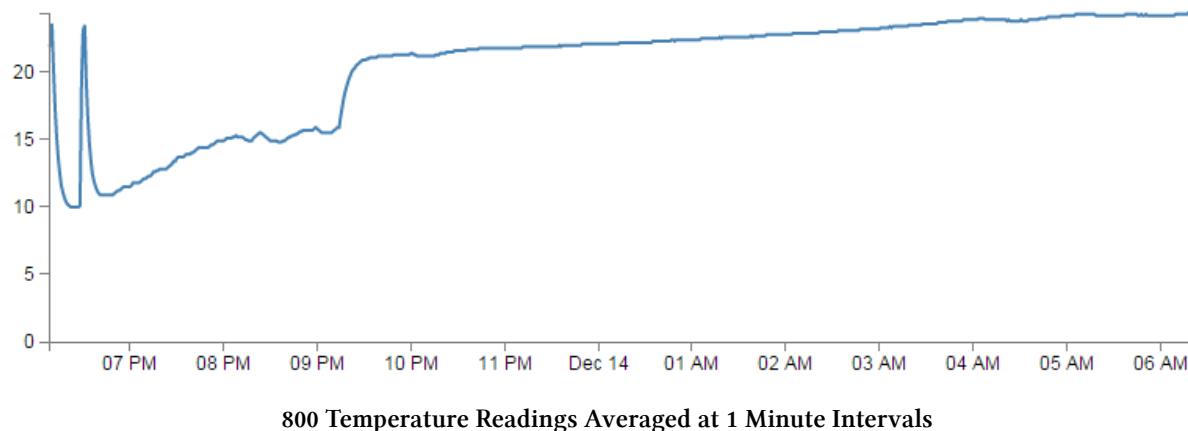
```
SELECT
ROUND(AVG(`temperature`),1) AS temperature,
TIMESTAMP(LEFT(`dtg`,16)) AS dtg
FROM `temperature`
GROUP BY LEFT(`dtg`,16)
ORDER BY `dtg` DESC
LIMIT 0,800
```

Here we are telling MySQL to average our temperature readings (AVG('temperature')) then round the number to 1 decimal place (ROUND(AVG('temperature'),1)) and label the column as 'temperature' (AS temperature). At the same time we take the left-most 16 characters of our dtg value (LEFT(dtg,16)) and then convert them back into a TIMESTAMP value (TIMESTAMP(LEFT('dtg',16))). This quirk allows us to eliminate the seconds from our 'dtg' values and replace it with '00'. We then label the column as 'dtg' (AS dtg ).

Now comes the interesting part. We group all the rows that have the same left-most 16 characters (in other words any dtg value that has the same year, month, day, hours and minutes). This has

no effect on the dtg values of course, but the temperature values for all the rows with identical dtg's get mashed together. And in this case we have already told MySQL that we want to average those values out with our earlier `ROUND(AVG('temperature'),1)` statement.

If we consider the resulting graph, it looks remarkably similar to our original one that included over 3000 points;



The end result has been a ‘smoothing’ of sorts. I can’t claim that it is any better or worse, but it certainly represents the way the temperature varied and does so in a way that won’t break the browser.

# Multiple Temperature Measurements

This project will measure the temperature at multiple points using DS18B20 sensors. This project will use the waterproof version of the sensors since they have more potential practical applications.

This project is a logical follow on to the [Single Temperature Measurement](#) project. The differences being the use of multiple sensors and with a slightly more sophisticated approach to recording and exploring our data. It is still a relatively simple hardware set-up.

## Measure

### Hardware required

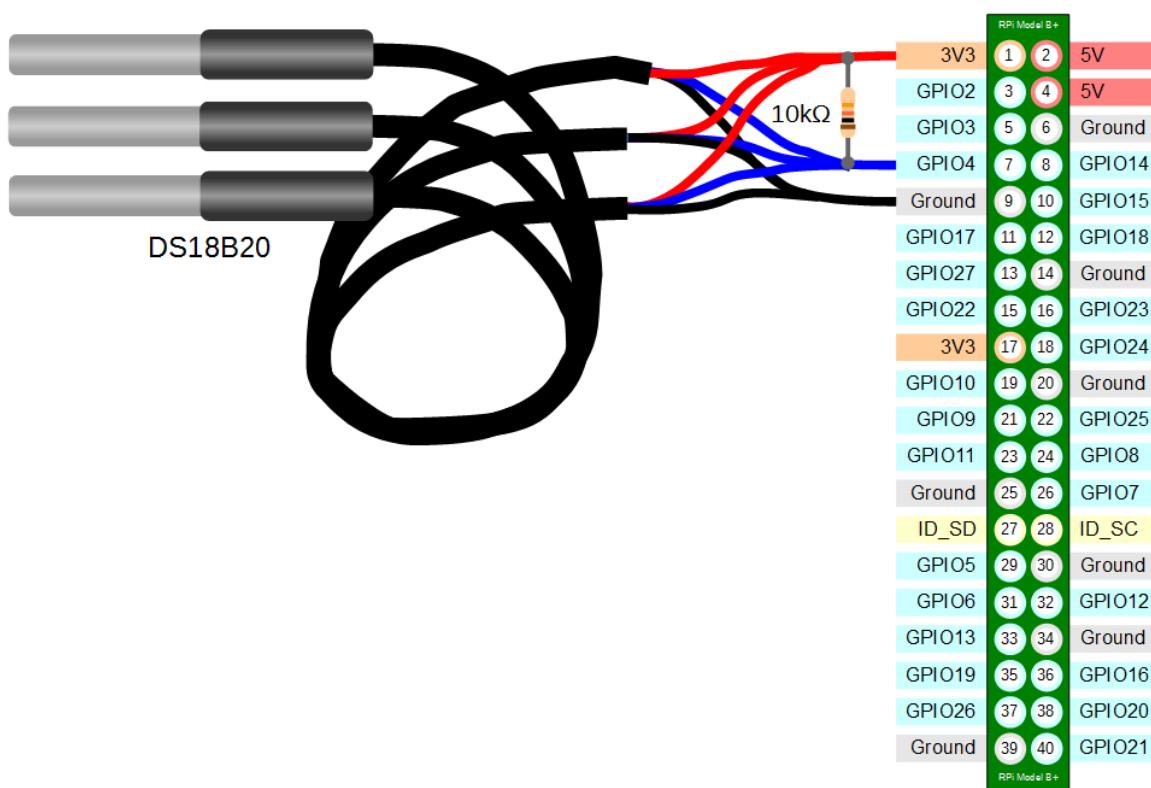
- DS18B20 sensors (the waterproof version)
- 10k Ohm resistor
- Jumper cables
- Solder
- Heatshrink

### Connect

The DS18B20 sensors needs to be connected with the black wires to ground, the red wires to the 3V3 pin and the blue or yellow (some sensors have blue and some have yellow) wires to the GPIO4 pin. A resistor between the value of 4.7k Ohms to 10k Ohms needs to be connected between the 3V3 and GPIO4 pins to act as a ‘pull-up’ resistor.

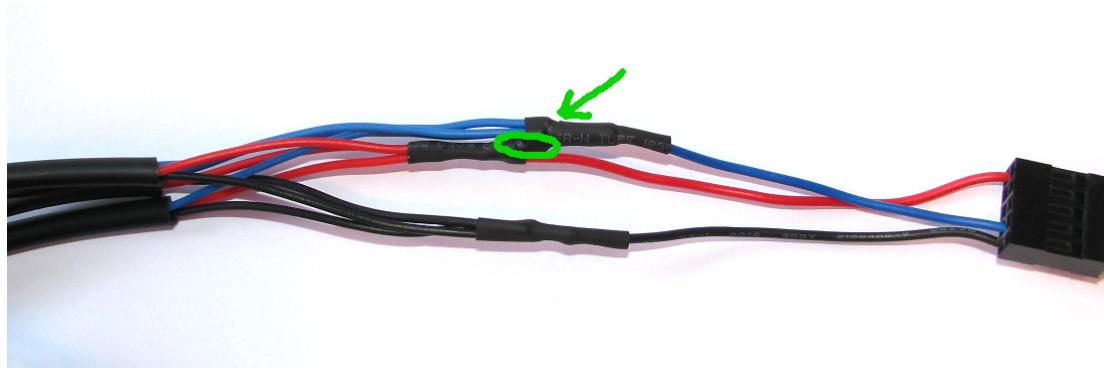
The Raspbian Operating System image that we are using only supports GPIO4 as a 1-Wire pin, so we need to ensure that this is the pin that we use for connecting our temperature sensor.

The following diagram is a simplified view of the connection.



Single DS18B20 Connection

To connect the sensor practically can be achieved in a number of ways. You could use a Pi Cobbler break out connector mounted on a bread board connected to the GPIO pins. But because the connection is relatively simple we could build a minimal configuration that will plug directly onto the appropriate GPIO pins. The resister is concealed under the heatshrink and indicated with the arrow.



Minimal Single DS18B20 Connection

This version uses a recovered header connector from a computers internal USB cable.

## Test

Because the Raspberry Pi acts like an embedded platform rather than a regular PC, it doesn't have a BIOS (Basic Input Output System) that goes through the various pieces of hardware when the Pi

boots up and configures everything. Instead it has an optional text file named `config.txt`. This can be found in the `/boot` directory. To enable the Pi to use the GPIO pin to communicate with our temperature sensor we need to tell it to configure itself with the `w1-gpio` Onewire interface module.



Many thanks to ‘Dan B’ for pointing me in the right direction to get this sorted :-).

We can do this by editing the `/boot/config.txt` file using...

```
sudo nano /boot/config.txt
```

...and adding in the line...

```
dtoverlay=w1-gpio
```

...at the end of the file

After making this change we need to reboot our Pi to let the changes take effect;

```
sudo reboot
```

From the terminal as the ‘pi’ user run the command;

```
sudo modprobe w1-gpio
```

`modprobe w1-gpio` registers the new sensor connected to GPIO4 so that now the Raspberry Pi knows that there is a 1-Wire device connected to the GPIO connector (For more information on the `modprobe` command check out the [Glossary](#)).



`modprobe` is a Linux program used to add a loadable kernel module (LKM) to the Linux kernel or to remove a LKM from the kernel. It is commonly used to load drivers for automatically detected hardware.

Then run the command;

```
sudo modprobe w1-therm
```

`modprobe w1-therm` tells the Raspberry Pi to add the ability to measure temperature on the 1-Wire system.

To allow the `w1_gpio` and `w1_therm` modules to load automatically at boot we can edit the the `/etc/modules` file and include both modules there where they will be started when the Pi boots up. To do this edit the `/etc/modules` file;

```
sudo nano /etc/modules
```

Add in the `w1_gpio` and `w1_therm` modules so that the file looks like the following;

```
# /etc/modules: kernel modules to load at boot time.  
#  
# This file contains the names of kernel modules that should be loaded  
# at boot time, one per line. Lines beginning with "#" are ignored.  
# Parameters can be specified after the module name.
```

`snd-bcm2835`

`w1-gpio`  
`w1-therm`

Save the file.

Then we change into the `/sys/bus/w1/devices` directory and list the contents using the following commands;

```
cd /sys/bus/w1/devices  
ls
```

(For more information on the `cd` command check out the Glossary [here](#). Or to find out more about the `ls` command go [here](#))

This should list out the contents of the `/sys/bus/w1/devices` which should include a number of directories starting `28-`. The number of directories should match the number of connected sensors. The portion of the name following the `28-` is the unique serial number of each of the sensors.

We then change into one of those directories;

```
cd 28-xxxx (change xxxx to match the serial number of one of the directories)
```

We are then going to view the '`w1_slave`' file with the `cat` command using;

```
cat w1_slave
```



The `cat`<sup>42</sup> program takes the specified file (or files) and by default outputs the results to the screen (there are a multitude of different options for `cat`, more can be seen in the [Glossary](#)).

The output should look something like the following;

```
73 01 4b 46 7f ff 0d 10 41 : crc=41 YES
73 01 4b 46 7f ff 0d 10 41 t=23187
```

At the end of the first line we see a YES for a successful CRC check (CRC stands for Cyclic Redundancy Check, a good sign that things are going well). If we get a response like NO or FALSE or ERROR, it will be an indication that there is some kind of problem that needs addressing. Check the circuit connections and start troubleshooting.

At the end of the second line we can now find the current temperature. The t=23187 is an indication that the temperature is 23.187 degrees Celsius (we need to divide the reported value by 1000).



To convert from degrees Celsius to degrees Fahrenheit, multiply by 9, then divide by 5, then add 32.

`cd` into each of the 28-xxxx directories in turn and run the `cat w1_slave` command to check that each is operating correctly. It may be useful at this stage to label the individual sensors with their unique serial numbers to make it easy to identify them correctly later.

## Record

To record this data we will use a Python program that checks all the sensors and writes the temperature and the sensor name into our MySQL database. At the same time a time stamp will be added automatically.

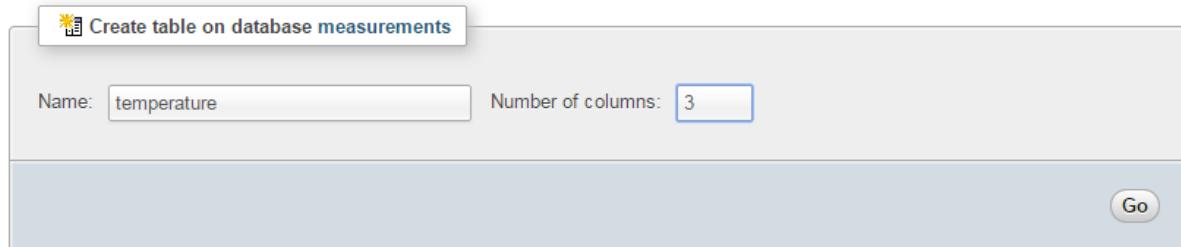
Our Python program will only write a single group of temperature readings to the database, but unlike the previous [single temperature measurement](#) example (which used `time.sleep(xx)`), this time we will execute the program at a regular interval using a clever feature used in Linux called `cron` (you can read a description of how to use the crontab (the cron-table) in the [Glossary](#)).

<sup>42</sup>[http://en.wikipedia.org/wiki/Cat\\_\(Unix\)](http://en.wikipedia.org/wiki/Cat_(Unix))

## Database preparation

First we will set up our database table that will store our data.

Using the phpMyAdmin web interface that we set up, log on using the administrator (root) account and select the ‘measurements’ database that we created as part of the initial set-up.



Create the MySQL Table

Enter in the name of the table and the number of columns that we are going to use for our measured values. In the screenshot above we can see that the name of the table is ‘temperature’ and the number of columns is ‘3’.

We will use three columns so that we can store a temperature reading, the time it was recorded and the unique ID of the sensor that recorded it.

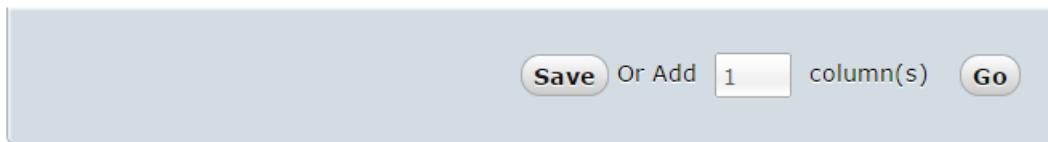
Once we click on ‘Go’ we are presented with a list of options to configure our table’s columns. Don’t be intimidated by the number of options that are presented, we are going to keep the process as simple as practical.

For the first column we can enter the name of the ‘Column’ as ‘dtg’ (short for date time group) the ‘Type’ as ‘TIMESTAMP’ and the ‘Default’ value as ‘CURRENT\_TIMESTAMP’. For the second column we will enter the name ‘temperature’ and the ‘Type’ is ‘FLOAT’ (we won’t use a default value). For the third column we will enter the name ‘sensor\_id’ and the type is ‘VARCHAR’ with a ‘Length/Values’ of 15.

Create Table			
Table name:			
temperature			
Structure			
Column	dtg	temperature	sensor_id
Type	TIMESTAMP	FLOAT	VARCHAR
Length/Values <sup>1</sup>			15
Default <sup>2</sup>	CURRENT_TIMESTAMP	None	None

Configure the MySQL Table Columns

Scroll down a little and click on the ‘Save’ button and we’re done.



### Save the MySQL Table Columns



### Why did we choose those particular settings for our table?

Our ‘dtg’ column needs to store a value of time that includes the date and the time, so the advantage of selecting TIMESTAMP in this case is that we can select the default value to be the current time which means that when we write our data to the table we only need to write the ‘temperature’ and ‘sensor\_id’ values and the ‘dtg’ will be entered automatically for us. The disadvantage of using ‘TIMESTAMP’ is that it has a more limited range than DATETIME. TIMESTAMP can only have a range between ‘1970-01-01 00:00:01’ to ‘2038-01-19 03:14:07’.

Our temperature readings are generated (by our sensor) as an integer value that needs to be divided by 1000 to show degrees Centigrade. We could therefore store the value as an integer. However when we were selecting the data or in later processing we would then need to do the math to convert it to the correct value. It could be argued (successfully) that this would be a more efficient solution in terms of the amount of space taken to support the data on the Pi. However, I have a preference for storing the values as they would be used later and as a result we need to use a numerical format that supports numbers with decimal places. There are a range of options for defining the ranges for decimal numbers, but FLOAT allows us to ignore the options (at the expense of efficiency) and rely on our recorded values being somewhere between -3.402823466E+38 and 3.402823466E+38 (if our temperature falls outside those extremes we are in trouble).

The sensor IDs are a combination of numbers, characters and letters, so we will use a variable type ‘VARCHAR’ which is for characters. We can also specify the maximum length of the information stored in the database to make things a little more efficient. In theory we could use the ‘CHAR’ type which is more efficient, but in this instance I prefer ‘VARCHAR’ which will allow the length of the recorded information to be flexible.

## Record the temperature values

The following Python code (which is based on the code that is part of the great temperature sensing tutorial on [iot-project<sup>43</sup>](#)) is a script which allows us to check the temperature reading from multiple sensors and write them to our database with a separate entry for each sensor.

The full code can be found in the code samples bundled with this book (m\_temp.py).

<sup>43</sup><http://iot-projects.com/index.php?id=connect-ds18b20-sensors-to-raspberry-pi>

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import os
import fnmatch
import time
import MySQLdb as mdb
import logging

logging.basicConfig(filename='/home/pi/DS18B20_error.log',
    level=logging.DEBUG,
    format='%(asctime)s %(levelname)s %(name)s %(message)s')
logger=logging.getLogger(__name__)

# Load the modules (not required if they are loaded at boot)
# os.system('modprobe w1-gpio')
# os.system('modprobe w1-therm')

# Function for storing readings into MySQL
def insertDB(IDs, temperature):

    try:

        con = mdb.connect('localhost',
                          'pi_insert',
                          'xxxxxxxxxx',
                          'measurements');

        cursor = con.cursor()

        for i in range(0,len(temperature)):
            sql = "INSERT INTO temperature(temperature, sensor_id) \
VALUES ('%s', '%s')" % \
            ( temperature[i], IDs[i])
            cursor.execute(sql)
            sql = []
        con.commit()

        con.close()

    except mdb.Error, e:
        logger.error(e)

# Get readings from sensors and store them in MySQL

temperature = []
IDs = []
```

```

for filename in os.listdir("/sys/bus/w1/devices"):
    if fnmatch.fnmatch(filename, '28-*'):
        with open("/sys/bus/w1/devices/" + filename + "/w1_slave") as f_obj:
            lines = f_obj.readlines()
            if lines[0].find("YES"):
                pok = lines[1].find('=')
                temperature.append(float(lines[1][pok+1:pok+6])/1000)
                IDs.append(filename)
            else:
                logger.error("Error reading sensor with ID: %s" % (filename))

if (len(temperature)>0):
    insertDB(IDs, temperature)

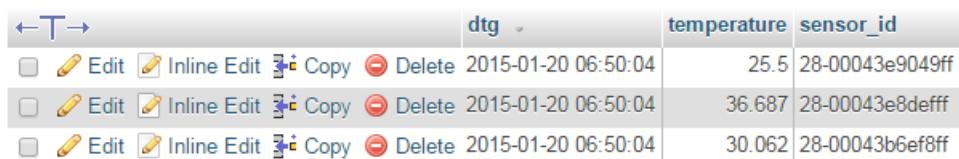
```

This script can be saved in our home directory (/home/pi) and can be run by typing:

```
python m_temp.py
```

While we won't see much happening at the command line, if we use our web browser to go to the phpMyAdmin interface and select the 'measurements' database and then the 'temperature' table we will see a range of temperature measurements for the different sensors and their associated time of reading.

Now you can be forgiven for thinking that this is not going to collect the sort of range of data that will let us 'Explore' very much, but let's do a quick explanation of the Python code first and then we'll work out how to record a lot more data :-).



The screenshot shows a MySQL table named 'temperature' with three rows of data. The columns are 'dtg' (Date and Time), 'temperature', and 'sensor\_id'. The data is as follows:

	dtg	temperature	sensor_id
<input type="checkbox"/>	2015-01-20 06:50:04	25.5	28-00043e9049ff
<input type="checkbox"/>	2015-01-20 06:50:04	36.687	28-00043e8defff
<input type="checkbox"/>	2015-01-20 06:50:04	30.062	28-00043b6ef8ff

Save the MySQL Table Columns

## Code Explanation

The script starts by importing the modules that it's going to use for the process of reading and recording the temperature measurements;

```
import os
import fnmatch
import time
import MySQLdb as mdb
import logging
```



Python code in one module gains access to the code in another module by the process of importing it. The `import` statement invokes the process and combines two operations; it searches for the named module, then it binds the results of that search to a name in the local scope.

Then the code sets up the `logging module`<sup>44</sup>. We are going to use the `basicConfig()` function to set up the default handler so that any debug messages are written to the file `/home/pi/DS18B20_error.log`.

```
logging.basicConfig(filename='/home/pi/DS18B20_error.log',
    level=logging.DEBUG,
    format='%(asctime)s %(levelname)s %(name)s %(message)s')
logger=logging.getLogger(__name__)
```

Later in the section where we are getting our readings or writing to the database we write to the log file if there is an error.

The program can then issue the `modprobe` commands that start the interface to the sensor;

```
# os.system('modprobe w1-gpio')
# os.system('modprobe w1-therm')
```

Our code has them commented out since we have already edited the `/etc/modules` file, but if you didn't want to start the modules at start-up (for whatever reasons), you can un-comment these.

We then declare the function that will insert the readings into the MySQL database;

```
def insertDB(IDs, temperature):
    try:
        con = mdb.connect('localhost',
                          'pi_insert',
                          'xxxxxxxxxx',
                          'measurements');
        cursor = con.cursor()
```

---

<sup>44</sup><http://pymotw.com/2/logging/>

```

for i in range(0, len(temperature)):
    sql = "INSERT INTO temperature(temperature, sensor_id) \
VALUES ('%s', '%s')" % \
    (temperature[i], IDs[i])
    cursor.execute(sql)
    sql = []
    con.commit()

con.close()

except mdb.Error, e:
    logger.error(e)

```

This is a neat piece of script that uses arrays to recall all the possible temperature and ID values. Then we have the main body of the script that finds all our possible sensors and reads the IDs and the temperatures;

```

temperature = []
IDs = []

for filename in os.listdir("/sys/bus/w1/devices"):
    if fnmatch.fnmatch(filename, '28-*'):
        with open("/sys/bus/w1/devices/" + filename + "/w1_slave") as f_obj:
            lines = f_obj.readlines()
            if lines[0].find("YES"):
                pok = lines[1].find('=')
                temperature.append(float(lines[1][pok+1:pok+6])/1000)
                IDs.append(filename)
            else:
                logger.error("Error reading sensor with ID: %s" % (filename))

if (len(temperature)>0):
    insertDB(IDs, temperature)

```

After declaring our two arrays temperature and IDs we start the for loop that checks all the file names in /sys/bus/w1/devices;

```
for filename in os.listdir("/sys/bus/w1/devices"):
```

If it finds a filename that starts with 28- then it processes it;

```
if fnmatch.fnmatch(filename, '28-*'):
```

First it opens the w1\_slave file in the 28-\* directory...

```
with open("/sys/bus/w1/devices/" + filename + "/w1_slave") as f_obj:
```

... then it pulls out the lines in the file;

```
lines = f_obj.readlines()
```

If it finds the word “YES” in the first line (line 0) of the file...

```
if lines[0].find("YES"):
```

...then it uses the position of the equals (=) sign in the second line (line 1)...

```
pok = lines[1].find('=')
```

... to pull out the characters following the ‘=’, and manipulate them to form the temperature in degrees Centigrade;

```
temperature.append(float(lines[1][pok+1:pok+6])/1000)
```

We then add the filename to the IDs array;

```
IDs.append(filename)
```

If we didn’t find a “yes” in the first line we log the error in the log file

```
logger.error("Error reading sensor with ID: %s" % (filename))
```

Then finally if we have been successful in reading at least one temperature value, we push the IDs and temperature array to the insertDB function;

```
if (len(temperature)>0):
    insertDB(IDs, temperature)
```

## Recording data on a regular basis with cron

As mentioned earlier, while our code is a thing of beauty, it only records a single entry for each sensor every time it is run.

What we need to implement is a schedule so that at a regular time, the program is run. This is achieved using cron via the crontab. While we will cover the requirements for this project here, you can read more about the [crontab](#) in the [Glossary](#).

To set up our schedule we need to edit the crontab file. This is done using the following command;

```
crontab -e
```

Once run it will open the crontab in the nano editor. We want to add in an entry at the end of the file that looks like the following;

```
*/* * * * /usr/bin/python /home/pi/m_temp.py
```

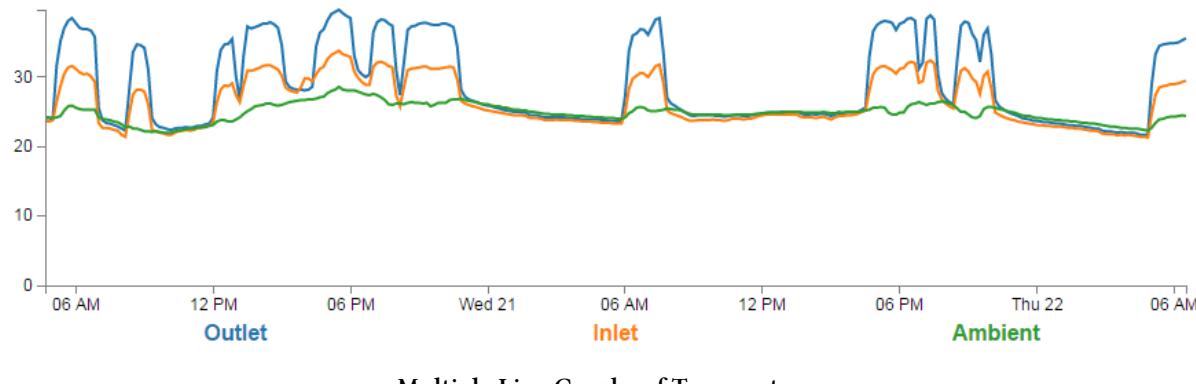
This instructs the computer that every minute of every hour of every day of every month we run the command `/usr/bin/python /home/pi/m_temp.py` (which if we were at the command line in the pi home directory we would run as `python m_temp.py`, but since we can't guarantee where we will be when running the script, we are supplying the full path to the python command and the `m_temp.py` script).

Save the file and the next time the computer boots up it will run our program on its designated schedule and we will have sensor entries written to our database every minute.

## Explore

This section has a working solution for presenting multiple streams of temperature data. This is a slightly more complex use of JavaScript and d3.js specifically but it is a great platform that demonstrates several powerful techniques for manipulating and presenting data.

The final form of the graph should look something like the following (depending on the number of sensors you are using, and the amount of data you have collected)



Multiple Line Graphs of Temperature

One of the neat things about this presentation is that it 'builds itself' in the sense that aside from us deciding what we want to label the specific temperature streams as, the code will organise all the colours and labels for us. Likewise, if the display is getting a bit messy we can click on the legend labels to show / hide the corresponding line.

## The Code

The following code is a PHP file that we can place on our Raspberry Pi's web server (in the /var/www directory) that will allow us to view all of the results that have been recorded in the temperature directory on a graph;



There are many sections of the code which have been explained already in the set-up section of the book that describes a simple line graph for a single temperature measurement. Where these occur we will be less thorough with the explanation of how the code works.

The full code can be found in the code samples bundled with this book (m\_temp.php).

```
<?php

$hostname = 'localhost';
$username = 'pi_select';
$password = 'xxxxxxxxxx';

try {
    $dbh = new PDO("mysql:host=$hostname;dbname=measurements", $username, $pa\ssword);

    /**
     * The SQL SELECT statement */
    $sth = $dbh->prepare("
        SELECT ROUND(AVG(`temperature`),1) AS temperature,
        TIMESTAMP(CONCAT(LEFT(`dtg`,15),'0')) AS date, sensor_id
        FROM `temperature`
        GROUP BY `sensor_id`, `date`
        ORDER BY `temperature`.`dtg` DESC
        LIMIT 0,900
    ");
    $sth->execute();

    /* Fetch all of the remaining rows in the result set */
    $result = $sth->fetchAll(PDO::FETCH_ASSOC);

    /**
     * close the database connection */
    $dbh = null;
}

catch(PDOException $e)
{
    echo $e->getMessage();
}
```

```
$json_data = json_encode($result);

?>

<!DOCTYPE html>
<meta charset="utf-8">
<style> /* set the CSS */

body { font: 12px Arial; }

path {
    stroke: steelblue;
    stroke-width: 2;
    fill: none;
}

.axis path,
.axis line {
    fill: none;
    stroke: grey;
    stroke-width: 1;
    shape-rendering: crispEdges;
}

.legend {
    font-size: 16px;
    font-weight: bold;
    text-anchor: middle;
}

</style>
<body>

<!-- load the d3.js library -->
<script src="http://d3js.org/d3.v3.min.js"></script>

<script>

// Set the dimensions of the canvas / graph
var margin = {top: 30, right: 20, bottom: 70, left: 50},
    width = 900 - margin.left - margin.right,
    height = 300 - margin.top - margin.bottom;

// Parse the date / time
```

```
var parseDate = d3.time.format("%Y-%m-%d %H:%M:%S").parse;

// Set the ranges
var x = d3.time.scale().range([0, width]);
var y = d3.scale.linear().range([height, 0]);

// Define the axes
var xAxis = d3.svg.axis().scale(x)
    .orient("bottom");

var yAxis = d3.svg.axis().scale(y)
    .orient("left").ticks(5);

// Define the line
var temperatureline = d3.svg.line()
    .x(function(d) { return x(d.date); })
    .y(function(d) { return y(d.temperature); });

// Adds the svg canvas
var svg = d3.select("body")
    .append("svg")
        .attr("width", width + margin.left + margin.right)
        .attr("height", height + margin.top + margin.bottom)
    .append("g")
        .attr("transform",
            "translate(" + margin.left + ", " + margin.top + ")");

// Get the data
<?php echo "data=".json_data.";" ?>

data.forEach(function(d) {
    d.date = parseDate(d.date);
    d.temperature = +d.temperature;
});

// Scale the range of the data
x.domain(d3.extent(data, function(d) { return d.date; }));
y.domain([0, d3.max(data, function(d) { return d.temperature; })]);

// Nest the entries by sensor_id
var dataNest = d3.nest()
    .key(function(d) {return d.sensor_id;})
    .entries(data);

var color = d3.scale.category10(); // set the colour scale
```

```

legendSpace = width/dataNest.length; // spacing for the legend

// Loop through each sensor_id / key
dataNest.forEach(function(d,i) {

    svg.append("path")
        .attr("class", "line")
        .style("stroke", function() { // Add the colours dynamically
            return d.color = color(d.key); })
        .attr("id", 'tag'+d.key.replace(/\s+/g, '')) // assign ID
        .attr("d", temperatureline(d.values));

    // Add the Legend
    svg.append("text")
        .attr("x", (legendSpace/2)+i*legendSpace) // space legend
        .attr("y", height + (margin.bottom/2)+ 5)
        .attr("class", "legend") // style the legend
        .style("fill", function() { // Add the colours dynamically
            return d.color = color(d.key); })
        .on("click", function(){
            // Determine if current line is visible
            var active = d.active ? false : true,
                newOpacity = active ? 0 : 1;
            // Hide or show the elements based on the ID
            d3.select("#tag"+d.key.replace(/\s+/g, '')).transition().duration(100)
                .style("opacity", newOpacity);
            // Update whether or not the elements are active
            d.active = active;
        })
        .text(
            function() {
                if (d.key == '28-00043b6ef8ff') {return "Inlet";}
                if (d.key == '28-00043e9049ff') {return "Ambient";}
                if (d.key == '28-00043e8defff') {return "Outlet";}
                else {return d.key;}
            });
});

// Add the X Axis
svg.append("g")
    .attr("class", "x axis")
    .attr("transform", "translate(0," + height + ")")
    .call(xAxis);

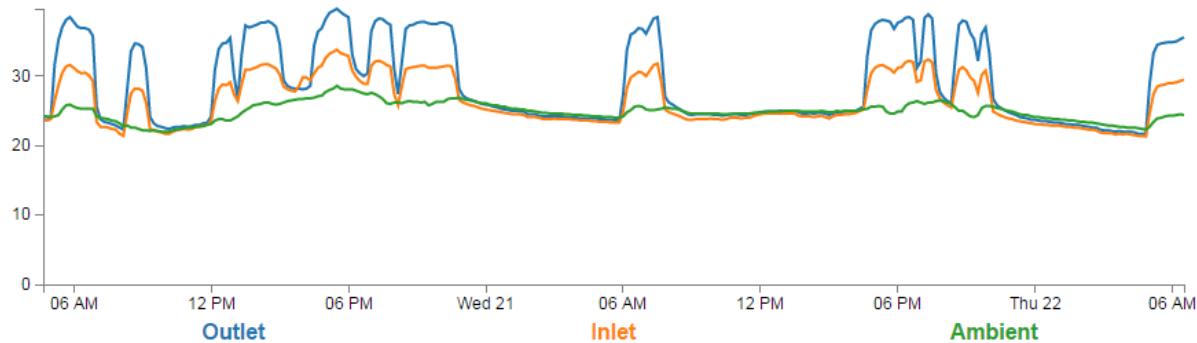
// Add the Y Axis

```

```
svg.append("g")
    .attr("class", "y_axis")
    .call(yAxis);

</script>
</body>
```

The graph that will look a little like this (except the data will be different of course).

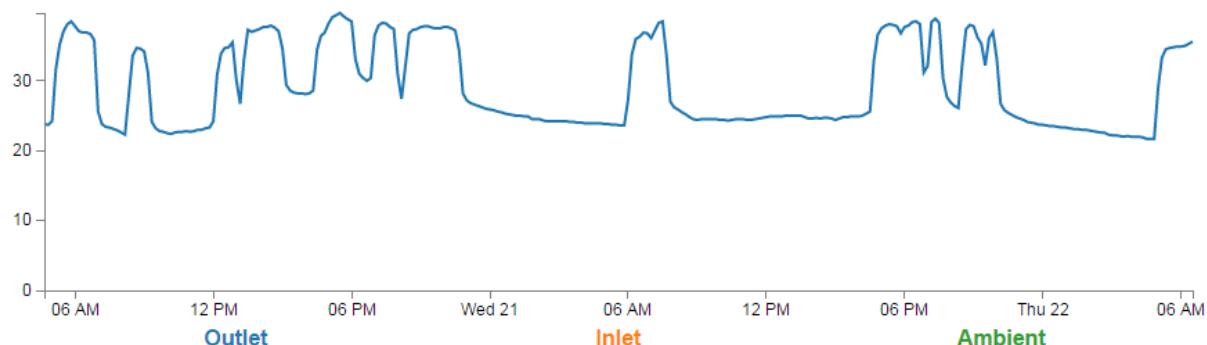


Multiple Temperature Line Graph

This is a fairly basic graph (i.e, there is no title or labelling of axis). It will automatically try to collect 900 measurements. So if (as is the case here) we have three sensors, this will result in three lines, each of which has 300 data points.

It does include some cool things though.

- It will automatically include as many lines as we have data for. So if we have 7 sensors, there will be 7 lines.
- Currently the graph is showing the three lines in the legend as ‘Outlet’, ‘Inlet’ and ‘Ambient’. This is because our code specifically assigns a name to a sensor ID. But, if we do not assign a specific label it will automagically use the sensor ID as the label.
- The colours for the lines and the legend will automatically be set as nicely distinct colours.
- We can click on a legend label and it will turn on / off the corresponding line to make it easier to read.



‘Inlet’ and ‘Ambient’ De-selected

## PHP

The PHP block at the start of the code is mostly the same as our example code for our single temperature measurement project. The significant difference however is in the select statement.

```
SELECT ROUND(AVG(`temperature`),1) AS temperature,
TIMESTAMP(CONCAT(LEFT(`dtg`,15), '0')) AS date, sensor_id
FROM `temperature`
GROUP BY `sensor_id`, `date`
ORDER BY `temperature`.`dtg` DESC
LIMIT 0,900
```

The difference is that we are now selecting three columns of information. The temperature, the date-time-group and the sensor ID.

temperature	date	sensor_id
25.7	2015-01-23 21:30:00	28-00043e9049ff
32.1	2015-01-23 21:30:00	28-00043e8defff
31.1	2015-01-23 21:30:00	28-00043b6ef8ff
24.8	2015-01-23 21:20:00	28-00043e9049ff
26.5	2015-01-23 21:20:00	28-00043e8defff
25.1	2015-01-23 21:20:00	28-00043b6ef8ff
24.7	2015-01-23 21:10:00	28-00043e9049ff
25.9	2015-01-23 21:10:00	28-00043e8defff
24.4	2015-01-23 21:10:00	28-00043b6ef8ff

temperature date and sensor\_id

In this project, we are going to need to ‘pivot’ the data that we are retrieving from our database so that it is produced in a multi-column format that the script can deal with easily. This is not always easy in programming, but it can be achieved using the d3 nest function which we will examine. Ultimately we want to be able to use the data in a format that looks a little like this;

Date	28-00043e9049ff	28-00043e8defff	28-00043b6ef8ff
2015-01-23 21:30:00	25.7	32.1	31.1
2015-01-23 21:20:00	24.8	26.5	25.1
2015-01-23 21:10:00	24.7	25.9	24.4

Pivoted sensor, temperature readings

We can see that the information is still the same, but there has been a degree of redundancy removed.

## JavaScript

The code is very similar to our single temperature measurement code and comparing both will show us that we are doing the same thing in each graph, but the manipulation of the data into a ‘pivoted’ or ‘nested form is a major deviation.

## Nesting the data

The following code nest's the data

```
var dataNest = d3.nest()
  .key(function(d) {return d.sensor_id;})
  .entries(data);
```

We declare our new array's name as dataNest and we initiate the nest function;

```
var dataNest = d3.nest()
```

We assign the key for our new array as sensor\_id. A 'key' is like a way of saying "*This is the thing we will be grouping on*". In other words our resultant array will have a single entry for each unique sensor\_id which will itself be an array of dates and values.

```
  .key(function(d) {return d.sensor_id;})
```

Then we tell the nest function which data array we will be using for our source of data.

```
}).entries(data);
```

Then we use the nested data to loop through our sensor IDs and draw the lines and the legend labels;

```
dataNest.forEach(function(d, i) {

  svg.append("path")
    .attr("class", "line")
    .style("stroke", function() { // Add the colours dynamically
      return d.color = color(d.key); })
    .attr("id", 'tag'+d.key.replace(/\s+/g, '')) // assign ID
    .attr("d", temperatureline(d.values));

  // Add the Legend
  svg.append("text")
    .attr("x", (legendSpace/2)+i*legendSpace) // space legend
    .attr("y", height + (margin.bottom/2)+ 5)
    .attr("class", "legend") // style the legend
    .style("fill", function() { // Add the colours dynamically
      return d.color = color(d.key); })
    .on("click", function(){
      // Determine if current line is visible
      var active = d.active ? false : true,
        newOpacity = active ? 0 : 1;
```

```

    // Hide or show the elements based on the ID
    d3.select("#tag"+d.key.replace(/\s+/g, ''))
      .transition().duration(100)
      .style("opacity", newOpacity);
    // Update whether or not the elements are active
    d.active = active;
  })
  .text(
    function() {
      if (d.key == '28-00043b6ef8ff') {return "Inlet";}
      if (d.key == '28-00043e9049ff') {return "Ambient";}
      if (d.key == '28-00043e8defff') {return "Outlet";}
      else {return d.key;}
    });
);

```

The `forEach` function being applied to `dataNest` means that it will take each of the keys that we have just declared with the `d3.nest` (each sensor ID) and use the values for each sensor ID to append a line using its values.

There is a small and subtle change that might otherwise go unnoticed, but is nonetheless significant. We include an `i` in the `forEach` function;

```
dataNest.forEach(function(d, i) {
```

This might not seem like much of a big deal, but declaring `i` allows us to access the index of the returned data. This means that each unique key (sensor ID) has a unique number.

Then the code can get on with the task of drawing our lines;

```

svg.append("path")
  .attr("class", "line")
  .style("stroke", function() { // Add the colours dynamically
    return d.color = color(d.key); })
  .attr("id", 'tag'+d.key.replace(/\s+/g, '') // assign ID
  .attr("d", temperatureline(d.values));

```

## Applying the colours

Making sure that the colours that are applied to our lines (and ultimately our legend text) is unique from line to line is actually pretty easy.

The set-up for this is captured in an earlier code snippet.

```
var color = d3.scale.category10(); // set the colour scale
```

This declares an [ordinal scale<sup>45</sup>](#) for our colours. This is a set of categorical colours (10 of them in this case) that can be invoked which are a nice mix of difference from each other and pleasant on the eye.

We then use the colour scale to assign a unique stroke (line colour) for each unique key (sensor ID) in our dataset.

```
.style("stroke", function() {
    return d.color = color(d.key); })
```

It seems easy when it's implemented, but in all reality, it is the product of some very clever thinking behind the scenes when designing d3.js and even picking the colours that are used.

Then we need to make sure that we can have a good reference between our lines and our legend labels. To do this we need to add assign an id to each legend text label.

```
.attr("id", 'tag'+d.key.replace(/\s+/g, ''))
```

Being able to use our key value as the id means that each label will have a unique identifier. “*What’s with adding the ‘tag’ piece of text to the id?*” I hear you ask. Good question. If our key starts with a number we could strike trouble (in fact I’m sure there are plenty of other ways we could strike trouble too, but this was one I came across). As well as that we include a little regular expression goodness to strip any spaces out of the key with `.replace(/\s+/g, '')`.



The `.replace` calls the regular expression action on our key. `\s` is the regex for “whitespace”, and `g` is the “global” flag, meaning match ALL `\s` (whitespaces). The `+` allows for any *contiguous* string of space characters to being replaced with the empty string (`' '`). This was a late addition to the example and kudos go to the participants in the [Stack Overflow question here<sup>46</sup>](#).

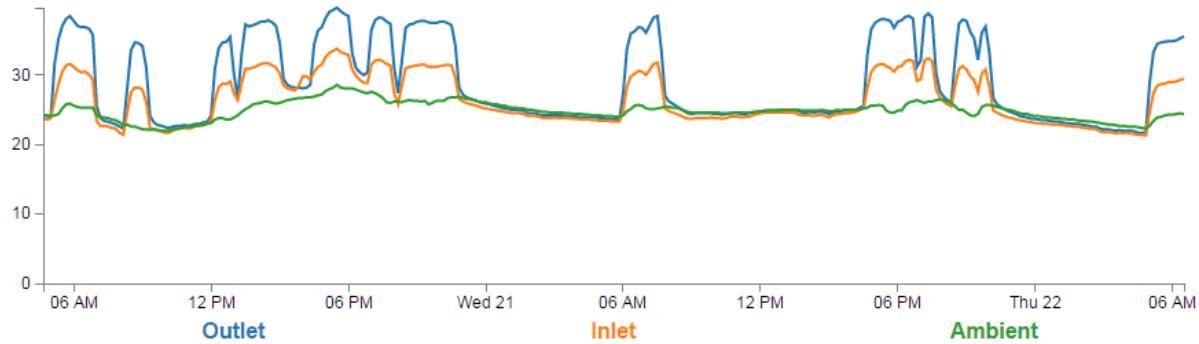
## Adding the legend

If we think about the process of adding a legend to our graph, what we’re trying to achieve is to take every unique data series we have (sensor ID) and add a relevant label showing which colour relates to which sensor. At the same time, we need to arrange the labels in such a way that they are presented in a manner that is not offensive to the eye. In the example I will go through I have chosen to arrange them neatly spaced along the bottom of the graph. so that the final result looks like the following;

---

<sup>45</sup><https://github.com/mbostock/d3/wiki/Ordinal-Scales>

<sup>46</sup><http://stackoverflow.com/questions/5963182/how-to-remove-spaces-from-a-string-using-javascript>



Multi-line graph with legend

Bear in mind that the end result will align the legend completely automatically. If there are three sensors it will be equally spaced, if it is six sensors they will be equally spaced. The following is a reasonable mechanism to facilitate this, but if the labels for the data values are of radically different lengths, the final result will look ‘odd’ likewise, if there are a LOT of data values, the legend will start to get crowded.

There are three broad categories of changes that we will want to make to our initial simple graph example code to make this possible;

1. Declare a style for the legend font
2. Change the area and margins for the graph to accommodate the additional text
3. Add the text

Declaring the style for the legend text is as easy as making an appropriate entry in the `<style>` section of the code. For this example we have the following;

```
.legend {
    font-size: 16px;
    font-weight: bold;
    text-anchor: middle;
}
```

To change the area and margins of the graph we can make the following small changes to the code.

```
var margin = {top: 30, right: 20, bottom: 70, left: 50},
    width = 900 - margin.left - margin.right,
    height = 300 - margin.top - margin.bottom;
```

The bottom margin is now 70 pixels high and the overall space for the area that the graph (including the margins) covers is increased to 300 pixels.

To add the legend text is slightly more work, but only slightly more.

One of the ‘structural’ changes we needed to put in was a piece of code that understood the physical layout of what we are trying to achieve;

```
legendSpace = width/dataNest.length; // spacing for the legend
```

This finds the spacing between each legend label by dividing the width of the graph area by the number of sensor IDs (key's).

The following code can then go ahead and add the legend;

```
// Add the Legend
svg.append("text")
    .attr("x", (legendSpace/2)+i*legendSpace) // space legend
    .attr("y", height + (margin.bottom/2)+ 5)
    .attr("class", "legend") // style the legend
    .style("fill", function() { // Add the colours dynamically
        return d.color = color(d.key); })
    .on("click", function(){
        // Determine if current line is visible
        var active = d.active ? false : true,
            newOpacity = active ? 0 : 1;
        // Hide or show the elements based on the ID
        d3.select("#tag"+d.key.replace(/\s+/g, ''))
            .transition().duration(100)
            .style("opacity", newOpacity);
        // Update whether or not the elements are active
        d.active = active;
    })
    .text(
        function() {
            if (d.key == '28-00043b6ef8ff') {return "Inlet";}
            if (d.key == '28-00043e9049ff') {return "Ambient";}
            if (d.key == '28-00043e8defff') {return "Outlet";}
            else {return d.key;}
        });

```

There are some slightly complicated things going on in here, so we'll make sure that they get explained.

Firstly we get all our positioning attributes so that our legend will go into the right place;

```
.attr("x", (legendSpace/2)+i*legendSpace) // space legend
    .attr("y", height + (margin.bottom/2)+ 5)
    .attr("class", "legend") // style the legend
```

The horizontal spacing for the labels is achieved by setting each label to the position set by the index associated with the label and the space available on the graph. To make it work out nicely we add half a legendSpace at the start (legendSpace/2) and then add the product of the index (i) and legendSpace (i\*legendSpace).

We position the legend vertically so that it is in the middle of the bottom margin (`height + (margin.bottom/2)+ 5`).

And we apply the same colour function to the text as we did to the lines earlier;

```
.style("fill", function() { // Add the colours dynamically
    return d.color = color(d.key); })
```

### Making it interactive

The last significant step we'll take in this example is to provide ourselves with a bit of control over how the graph looks. Even with the multiple colours, the graph could still be said to be 'busy'. To clean it up or at least to provide the ability to more clearly display the data that a user wants to see we will add code that will allow us to click on a legend label and this will toggle the corresponding graph line on or off.

```
.on("click", function(){
    // Determine if current line is visible
    var active = d.active ? false : true,
        newOpacity = active ? 0 : 1;
    // Hide or show the elements based on the ID
    d3.select("#tag"+d.key.replace(/\s+/g, ''))
        .transition().duration(100)
        .style("opacity", newOpacity);
    // Update whether or not the elements are active
    d.active = active;
})
```

We use the `.on("click", function(){` call to carry out some actions on the label if it is clicked on. We toggle the `.active` descriptor for our element with `var active = d.active ? false : true`. Then we set the value of `newOpacity` to either `0` or `1` depending on whether `active` is `false` or `true`.

From here we can select our label using its unique id and adjust it's opacity to either `0` (transparent) or `1` (opaque);

```
d3.select("#tag"+d.key.replace(/\s+/g, ''))
    .transition().duration(100)
    .style("opacity", newOpacity);
```

Just because we can, we also add in a transition statement so that the change in transparency doesn't occur in a flash (100 milli seconds in fact `(.duration(100))`).

Lastly we update our `d.active` variable to whatever the active state is so that it can toggle correctly the next time it is clicked on.

Since it's kind of difficult to represent interactivity in a book, head on over to the [live example on bl.ocks.org](#)<sup>47</sup> to see the toggling awesomeness that could be yours!

### Printing out custom labels

The only thing left to do is to decide what to print for our labels. If we wanted to simply show each sensor ID we could have the following;

```
.text(d.key);
```

This would produce the following at the bottom of the graph;



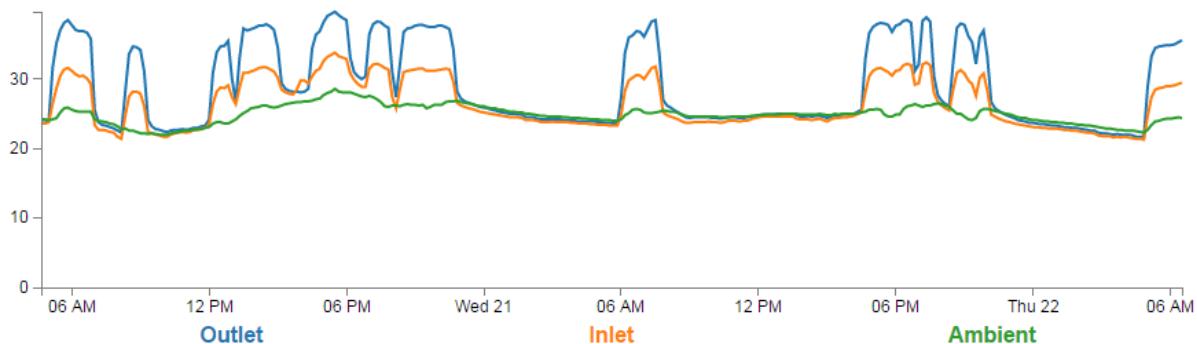
Multi-line graph with legend

But it makes more sense to put a real-world label in place so the user has a good idea about what they're looking at. To do this we can use an `if` statement to match up our sensors with a nice human readable representation of what is going on;

```
.text(
  function() {
    if (d.key == '28-00043b6ef8ff') {return "Inlet";}
    if (d.key == '28-00043e9049ff') {return "Ambient";}
    if (d.key == '28-00043e8defff') {return "Outlet";}
    else {return d.key;}
  });

```

The final result is a neat and tidy legend at the bottom of the graph;



Multi-line graph with legend

---

<sup>47</sup><http://bl.ocks.org/d3noob/e99a762017060ce81c76>

# System Information Measurement

This project will measure system information that shows how our Raspberry Pi is operating. This is useful information that will allow us to monitor the performance of our computing asset and to identify potential problems before they occur (or to discover the cause of problems when they do).

Specifically we are going to measure, record and display;

- System load average
- Memory (Ram) usage
- Disk usage (on our SD card)
- Temperature of the Raspberry Pi

## Measure

### Hardware required

Only the Raspberry Pi! All the readings are taken from the Pi itself.

### Measured Parameters

#### System Load

Load average is available in the file /proc/loadavg. We can print the contents of this file using cat;

```
cat /proc/loadavg
```

This will produce a line showing five (or maybe six depending on how you look at it) pieces of information that will look a little like the following;

```
0.14 0.11 0.13 1/196 16991
```

The first three numbers give the average load for 1 minute (0.14), 5 minutes (0.11) and 15 minutes (0.13) respectively. The next combination of two numbers separated by a slash (1/196) provides two(ish) pieces of information. Firstly, the number before the slash gives the number of threads running at that moment. This should be less than or equal to the CPUs in the system and in the case of the Raspberry Pi this should be less than or equal to 1. The number after the slash

indicates the total number of threads in the system. The last number is the process Id (the ‘pid’) of the thread that ran last.



From [Advanced Linux Programming](#)<sup>48</sup>:

*“Threads. Like processes are a mechanism to allow a program to do more than one thing at a time. As with processes, threads appear to run concurrently; the Linux kernel schedules them asynchronously, interrupting each thread from time to time to give others a chance to execute. Conceptually, a thread exists within a process. Threads are a finer-grained unit of execution than processes. When you invoke a program, Linux creates a new process and in that process creates a single thread, which runs the program sequentially. That thread can create additional threads; all these threads run the same program in the same process, but each thread may be executing a different part of the program at any given time.”*

We’re more interested in the system load numbers. Load average is an indication of whether the system resources (mainly the CPU) are adequately available for the processes (system load) that are running, runnable or in uninterruptible sleep states during the previous n minutes. A process is running when it has the full attention of the CPU. A runnable process is a process that is waiting for the CPU. A process is in uninterruptible sleep state when it is waiting for a resource and cannot be interrupted and will return from sleep only when the resource becomes available or a timeout occurs.

For example, a process may be waiting for disk or access to the network. Runnable processes indicate that we need more CPUs. Similarly processes in uninterruptible sleep state indicate Input/Output (I/O) bottlenecks. The load number at any time is the number of running, runnable and uninterruptible sleep state processes (we will call these collectively runnable processes) in the system. The load average is the average of the load number during the previous n minutes.

If, in a single CPU system such as our Raspberry Pi, the load average is 5, it is an undesirable situation because one process runs on the CPU and the other 4 have to wait for their turn. So the system is overloaded by 400%. In the above cat /proc/loadavg command output the load average for the last minute is 0.14, which indicates that the CPU is *underloaded* by 86%. Load average figures help in figuring out how much our system is able to cater to processing demands.

For our project we will record the value of system load at one of those intervals.

## Memory Used

The memory being used by the Raspberry Pi can be determined using the free command. If we type in the command as follows we will see an interesting array of information;

```
free -m
```

Produces ...

---

<sup>48</sup><http://www.advancedlinuxprogramming.com/>

	total	used	free	shared	buffers	cached
Mem:	437	385	51	0	85	197
-/+ buffers/cache:		102	335			
Swap:	99	0	99			

This is displaying the total amount of free and used physical and swap memory in the system, as well as the buffers used by the kernel. We also used the `-m` switch at the end of the command to display the information in megabytes.

The row labelled ‘Mem’, displays physical memory utilization, including the amount of memory allocated to buffers and caches.



A buffer, also called buffer memory, is usually defined as a portion of memory that is set aside as a temporary holding place for data that is being sent to or received from an external device, such as a HDD, keyboard, printer or network. Cache is a memory location to store frequently used data for faster access. Cache data can be used multiple times whereas buffer memory is used only once. And both are temporary stores for your data processing.

The next line of data, which begins with ‘-/+ buffers/cache’, shows the amount of physical memory currently devoted to system buffer cache. This is particularly meaningful with regards to applications, as all data accessed from files on the system pass through this cache. This cache can greatly speed up access to data by reducing or eliminating the need to read from or write to the SD card.

The last row, which begins with ‘Swap’, shows the total swap space as well as how much of it is currently in use and how much is still available.

For this project we will record the amount of memory used as a percentage of the total available.

## Disk Used

It would be a useful metric to know what the status of our available hard drive space was. To determine this we can use the `df` command. If we use the `df` command without any arguments as follows;

```
df
```

Produces ...

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
rootfs	7513804	2671756	4486552	38%	/
/dev/root	7513804	2671756	4486552	38%	/
devtmpfs	219744	0	219744	0%	/dev
tmpfs	44784	264	44520	1%	/run
tmpfs	5120	0	5120	0%	/run/lock
tmpfs	89560	0	89560	0%	/run/shm
/dev/mmcblk0p1	57288	9920	47368	18%	/boot

The first column shows the name of the disk partition as it appears in the /dev directory. The following columns show total space, blocks allocated and blocks available. The capacity column indicates the amount used as a percentage of total file system capacity.

The final column shows the mount point of the file system. This is the directory where the file system is mounted within the file system tree. Note that the root partition will always show a mount point of /. Other file systems can be mounted in any directory of a previously mounted file system.

For our purposes the root file system (/dev/root) would be ideal. We can reduce the amount of information that we have to deal with by running the df command and requesting information on a specific area. For example...

```
df /dev/root
```

It will produce ...

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
/dev/root	7513804	2671756	4486552	38%	/

In the project we will use the percentage used of the drive space as our metric.

## Raspberry Pi Temperature

The Raspberry Pi includes a temperature sensor in its CPU. The reading can be found by running the command

```
/opt/vc/bin/vcgencmd measure_temp
```

Which will produce a nice, human readable output similar to the following:

```
temp=39.0'C
```

This value shouldn't be taken as an indication of the ambient temperature of the area surrounding the Pi. Instead it is an indication of the temperature of the chip that contains the CPU. Monitoring this temperature could be useful since there is a maximum temperature at which it can operate reliably. Admittedly that maximum temperature is 85 degrees for the CPU (the BCM2835) and once it hits that figure it reduces the clock to reduce the temperature, but none the less, in some environments it will be a factor.

## Record

To record this data we will use a Python program that checks all the the values and writes them into our MySQL database along with the current time stamp.

Our Python program will only write a single group of readings to the database and we will execute the program at a regular interval using a [cron](#) job in the same way as the multiple temperature sensor project.

## Database preparation

First we will set up our database table that will store our data.

Using the phpMyAdmin web interface that we set up, log on using the administrator (root) account and select the 'measurements' database that we created as part of the initial set-up.

The screenshot shows a 'Create table on database measurements' dialog box. At the top, there's a title bar with the text 'Create table on database measurements'. Below this, there are two input fields: 'Name:' containing 'system\_info' and 'Number of columns:' containing '5'. At the bottom right of the dialog is a 'Go' button.

Create the MySQL Table

Enter in the name of the table and the number of columns that we are going to use for our measured values. In the screenshot above we can see that the name of the table is 'system\_info' and the number of columns is '5'.

We will use five columns so that we can store our four readings (system load, used ram, used disk and CPU temperature).

Once we click on 'Go' we are presented with a list of options to configure our table's columns. Again, we are going to keep the process as simple as practical and while we could be more economical with our data type selections, we'll err on the side of simplicity.

For the first column we can enter the name of the 'Column' as 'load' with a type of 'REAL'. Our second column is 'ram' and our third is 'disk'. Both of these should have a type of 'TINYINT'. Then we have a column for 'temperature' and the type is 'FLOAT'. Lastly we include the column 'dtg' (short for date time group) the type as 'TIMESTAMP' and the 'Default' value as 'CURRENT\_TIMESTAMP'.

Table name:			
system_info			
Column	Type	Length/Values <sup>1</sup>	Default <sup>2</sup>
load	REAL		None
ram	TINYINT		None
disk	TINYINT		None
temperature	REAL		None
dtg	TIMESTAMP		CURRENT_TIMESTAMP

#### Configure the MySQL Table Columns

Scroll down a little and click on the ‘Save’ button and we’re done.

<input style="border-radius: 10px; padding: 5px 10px;" type="button" value="Save"/> Or Add <input style="width: 20px;" type="text" value="1"/> column(s) <input style="border-radius: 10px; padding: 5px 10px;" type="button" value="Go"/>
--

#### Save the MySQL Table Columns



### Why did we choose those particular settings for our table?

Our system load readings will vary from 0 up and will hopefully spend most of their time as a decimal less than 1. However, sometimes it will exceed this. Our ‘temperature’ readings will also be a decimal value. Assigning them both a REAL data type allows us to recorded values where the number can have a maximum of 65 Digits, with 30 digits after decimal point (that should be plenty).

Both ‘ram’ and ‘disk’ will be integers, but since they will both be percentage values which will not exceed 100% we can use the ‘TINYINT’ type which allows values from -128 to 127 (or 0 to 255 if they’re unsigned).

Our ‘dtg’ column needs to store a value of time that includes the date and the time, so the advantage of selecting TIMESTAMP in this case is that we can select the default value to be the current time which means that when we write our data to the table we only need to write the ‘temperature’ and ‘sensor\_id’ values and the ‘dtg’ will be entered automatically for us. The disadvantage of using ‘TIMESTAMP’ is that it has a more limited range than DATETIME. TIMESTAMP can only have a range between ‘1970-01-01 00:00:01’ to ‘2038-01-19 03:14:07’.

## Record the system information values

The following Python code is a script which allows us to check the system readings from the Raspberry Pi and writes them to our database.

The full code can be found in the code samples bundled with this book (system\_info.py).

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import subprocess
import os
import MySQLdb as mdb

# Function for storing readings into MySQL
def insertDB(system_load, ram, disk, temperature):

    try:
        con = mdb.connect('localhost',
                          'pi_insert',
                          'xxxxxxxxxx',
                          'measurements');

        cursor = con.cursor()

        sql = "INSERT INTO system_info(`load`, `ram`, `disk`, `temperature`) \
VALUES ('%s', '%s', '%s', '%s')" % \
(system_load, ram, disk, temperature)
        cursor.execute(sql)
        con.commit()

        con.close()

    except mdb.Error, e:
        con.rollback()
        print "Error %d: %s" % (e.args[0],e.args[1])
        sys.exit(1)

# returns the system load over the past minute
def get_load():
    try:
        s = subprocess.check_output(["cat", "/proc/loadavg"])
        return float(s.split()[0])
    except:
        return 0

# Returns the used ram as a percentage of the total available
def get_ram():
```

```

try:
    s = subprocess.check_output(["free", "-m"])
    lines = s.split("\n")
    used_mem = float(lines[1].split()[2])
    total_mem = float(lines[1].split()[1])
    return (int((used_mem/total_mem)*100))
except:
    return 0

# Returns the percentage used disk space on the /dev/root partition
def get_disk():
    try:
        s = subprocess.check_output(["df", "/dev/root"])
        lines = s.split("\n")
        return int(lines[1].split("%")[0].split()[4])
    except:
        return 0

# Returns the temperature in degrees C of the CPU
def get_temperature():
    try:
        dir_path="/opt/vc/bin/vcgencmd"
        s = subprocess.check_output([dir_path, "measure_temp"])
        return float(s.split("=")[1][-3])
    except:
        return 0

got_load = str(get_load())
got_ram = str(get_ram())
got_disk = str(get_disk())
got_temperature = str(get_temperature())

insertDB(got_load, got_ram, got_disk, got_temperature)

```

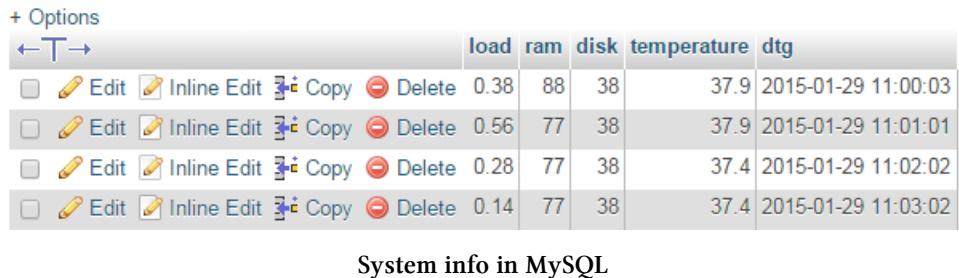
This script can be saved in our home directory (/home/pi) and can be run by typing:

```
python system_info.py
```

While we won't see much happening at the command line, if we use our web browser to go to the phpMyAdmin interface and select the 'measurements' database and then the 'system\_info' table we will see a range of information for the different system parameters and their associated time of reading.

As with our previous project recording multiple temperature points, this script only records a

single line of data whenever it is run. To make the collection more regular we will put in a cron job later to regularly check and record.



	+ Options	← →	load	ram	disk	temperature	dtg
	<input type="checkbox"/>	<input type="checkbox"/> Edit <input type="checkbox"/> Inline Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	0.38	88	38	37.9	2015-01-29 11:00:03
	<input type="checkbox"/>	<input type="checkbox"/> Edit <input type="checkbox"/> Inline Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	0.56	77	38	37.9	2015-01-29 11:01:01
	<input type="checkbox"/>	<input type="checkbox"/> Edit <input type="checkbox"/> Inline Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	0.28	77	38	37.4	2015-01-29 11:02:02
	<input type="checkbox"/>	<input type="checkbox"/> Edit <input type="checkbox"/> Inline Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	0.14	77	38	37.4	2015-01-29 11:03:02

System info in MySQL

## Code Explanation

The script starts by importing the modules that it's going to use for the process of reading and recording the temperature measurements;

```
import subprocess
import os
import MySQLdb as mdb
```



Python code in one module gains access to the code in another module by the process of importing it. The `import` statement invokes the process and combines two operations; it searches for the named module, then it binds the results of that search to a name in the local scope.

We then declare the function that will insert the readings into the MySQL database;

```
def insertDB(system_load, ram, disk, temperature):

    try:
        con = mdb.connect('localhost',
                          'pi_insert',
                          'xxxxxxxxxx',
                          'measurements');

        cursor = con.cursor()

        sql = "INSERT INTO system_info(`load`, `ram`, `disk`, `temperature`) \
VALUES ('%s', '%s', '%s', '%s')" % \
(system_load, ram, disk, temperature)
        cursor.execute(sql)
        con.commit()

        con.close()
```

```
except mdb.Error, e:  
    con.rollback()  
    print "Error %d: %s" % (e.args[0],e.args[1])  
    sys.exit(1)
```

This is a fairly simple insert of the values we will be collecting into the database.

Then we have our four functions that collect our system information;

## load

As described in the earlier [section](#) we will be extracting information out of the /proc/loadavg file. Specifically we will extract the average load for the last minute. We are going to accomplish this using the following code;

```
def get_load():  
    try:  
        s = subprocess.check_output(["cat", "/proc/loadavg"])  
        return float(s.split()[0])  
    except:  
        return 0
```

The code employs the subprocess module (which we loaded at the start of the script) which in this case is using the check\_output convenience function which will run the command (cat) and the argument (/proc/loadavg). It will return the output as a string (0.14 0.11 0.13 1/196 16991) that we can then manipulate. This string is stored in the variable s.

The following line returns the value from the function. The value is a floating number (decimal) and we are taking the first part (split()[0]) of the string (s) which by default is being split on any whitespace. In the case of our example string (0.14 0.11 0.13 1/196 16991) that would return the value 0.14.

If there is a problem retrieving the number it will be set to 0 (except: return 0).

## ram

As described in the earlier [section](#) we will be extracting information out of the results from running the free command with the -m argument. Specifically we will extract the used memory and total memory values and convert them to a percentage. We are going to accomplish this using the following code;

```
def get_ram():
    try:
        s = subprocess.check_output(["free", "-m"])
        lines = s.split("\n")
        used_mem = float(lines[1].split()[2])
        total_mem = float(lines[1].split()[1])
        return (int((used_mem/total_mem)*100))
    except:
        return 0
```

The code employs the `subprocess` module (which we loaded at the start of the script) which in this case is using the `check_output` convenience function which will run the command (`free`) and the argument (`-m`). It will store the returned multi-line output showing memory usage in the variable `s`. The output from this command (if it is run from the command line) would look a little like this;

	total	used	free	shared	buffers	cached
Mem:	437	385	51	0	85	197
-/+ buffers/cache:		102	335			
Swap:	99	0	99			

We then split that output string line by line and store the result in the array `lines` using `lines = s.split("\n")`.

Then we find the used and total memory by looking at the second line down (`lines[1]`) and extracting the appropriate column (`split()[2]` for used and `split()[1]` for total).

Then it's just some simple math to turn the memory variables (`used_mem` and `total_mem`) into a percentage.

If there is a problem retrieving the number it will be set to 0 (except: `return 0`).

## disk

As described in the earlier [section](#) we will be extracting information out of the results from running the `df` command with the `/dev/root` argument. Specifically we will extract the percentage used value. We will accomplish this using the following code;

```
def get_disk():
    try:
        s = subprocess.check_output(["df", "/dev/root"])
        lines = s.split("\n")
        return int(lines[1].split("%")[0].split()[4])
    except:
        return 0
```

The code employs the `subprocess` module (which we loaded at the start of the script) which in this case is using the `check_output` convenience function which will run the command (`df`) and the argument (`/dev/root`). It will store the returned multi-line output showing disk partition usage data in the variable `s`.

We then split that output string line by line and store the result in the array `lines` using `lines = s.split("\n")`.

Then, using the second line down (`lines[1]`)...

```
/dev/root      7513804 2671756  4486552  38% /
```

... we extract the percentage column (`split()[4]`) and remove the percentage sign from the number (`split("%")[0]`). The final value is returned as an integer.

If there is a problem retrieving the number it will be set to 0 (except: `return 0`).

## temperature

As described in the earlier [section](#) we will be extracting the temperature of the Raspberry Pis CPU the `vcgencmd` command with the `measure_temp` argument. We will accomplish this using the following code;

```
def get_temperature():
    try:
        dir_path="/opt/vc/bin/vcgencmd"
        s = subprocess.check_output([dir_path, "measure_temp"])
        return float(s.split("=')[1][:-3])
    except:
        return 0
```

The code employs the `subprocess` module (which we loaded at the start of the script) which in this case is using the `check_output` convenience function which will run the command (`vcgencmd`) and the argument (`/dev/root`). The `vcgencmd` command is referenced by its full path name which is initially stored as the variable `dir_path` and is then used in the subprocess command (this is only done for the convenience of not causing a line break in the code for the book by the way). The `measure_temp` argument returns the temperature in a human readable string (`temp=39.0'C`) which is stored in the variable `s`.

We are extracting the percentage value by splitting the line on the equals sign (`split("=")`), taking the text after the equals sign and trimming off the extra that is not required (`[1][:-3]`). The final value is returned as a real number.

If there is a problem retrieving the number it will be set to 0 (except: `return 0`).

## Main program

The main part of the program (if you can call it that) consists of only the following lines;

```
got_load = str(get_load())
got_ram = str(get_ram())
got_disk = str(get_disk())
got_temperature = str(get_temperature())

insertDB(got_load, got_ram, got_disk, got_temperature)
```

They serve to retrieve the value from each of our measurement functions and to then send the results to the function that writes the values to the database.

## Recording data on a regular basis with cron

As mentioned earlier, while our code is a thing of beauty, it only records a single entry for each sensor every time it is run.

What we need to implement is a schedule so that at a regular time, the program is run. This is achieved using cron via the crontab. While we will cover the requirements for this project here, you can read more about the [crontab](#) in the [Glossary](#).

To set up our schedule we need to edit the crontab file. This is done using the following command;

```
crontab -e
```

Once run it will open the crontab in the nano editor. We want to add in an entry at the end of the file that looks like the following;

```
*/* * * * /usr/bin/python /home/pi/system_info.py
```

This instructs the computer that every minute of every hour of every day of every month we run the command `/usr/bin/python /home/pi/system_info.py` (which if we were at the command line in the pi home directory we would run as `python system_info.py`, but since we can't guarantee where we will be when running the script, we are supplying the full path to the `python` command and the `system_info.py` script).

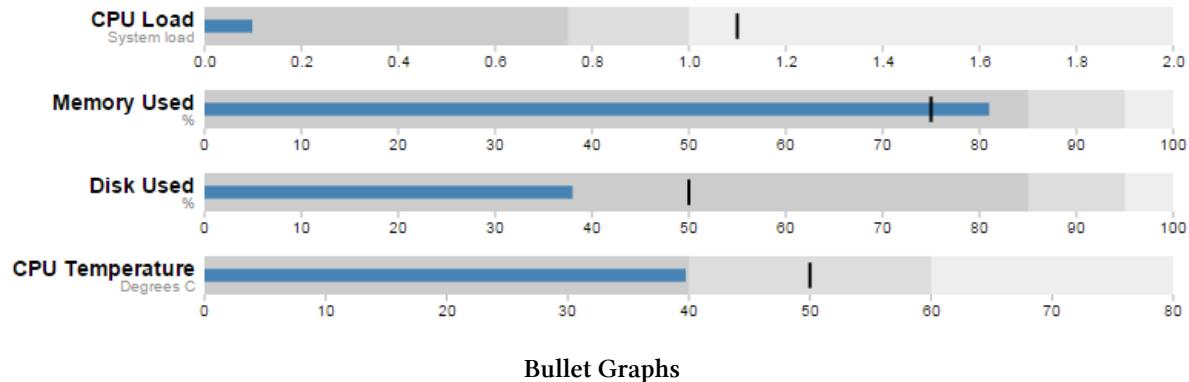
Save the file and the computer will start running the program on its designated schedule and we will have sensor entries written to our database every minute.

## Explore

This section has a working solution for presenting real-time, dynamic information from your Raspberry Pi. This is an interesting visualization, because it relies on two scripts in order to work properly. One displays the graphical image in our browser, and the other determines the current state of the data from our Pi, or more specifically from our database. The browser is set

up to regularly interrogate our data gathering script to get the values as they change. This is an effective demonstration of using dynamic visual effects with changing data.

The final form of the graph should look something like the following;



## The Bullet Graph

One of the first d3.js examples I ever came across (back when it was called Protovis) was one with bullet charts (or bullet graphs). They struck me straight away as an elegant way to represent data by providing direct information and context. With it we are able to show a measured value, labelling (and sub-labels), ranges and specific markers. As a double bonus it is a relatively simple task to get them to update when our data changes. If you are interested in seeing a bit more of an overview with some examples, check out the book '[D3 Tips and Tricks<sup>49</sup>](#)'.

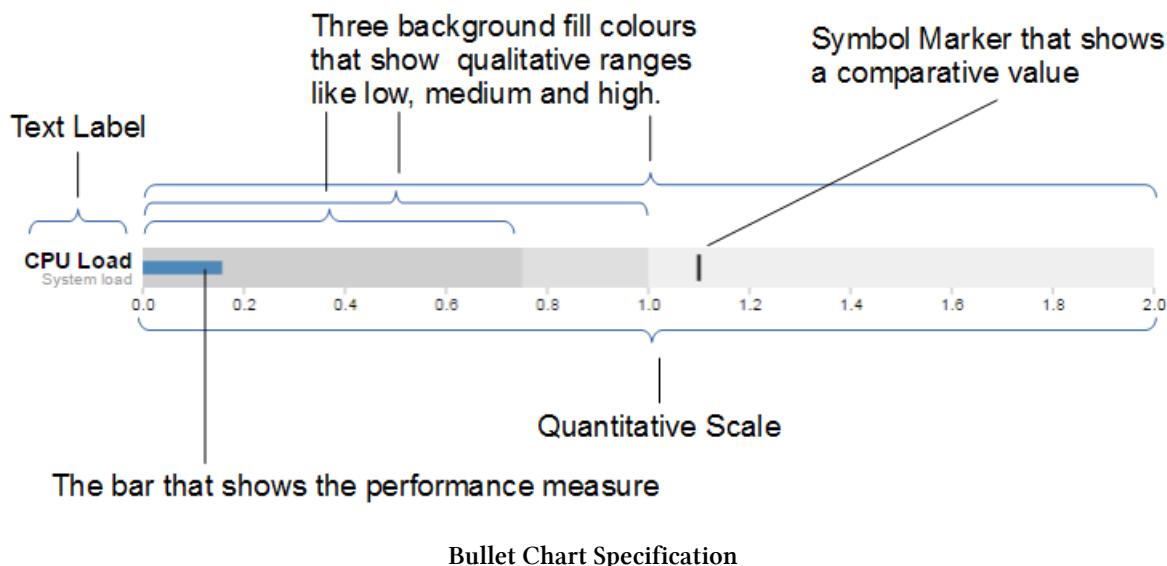
The [Bullet Graph Design Specification<sup>50</sup>](#) was laid down by Stephen Frew as part of his work with [Perceptual Edge<sup>51</sup>](#).

Using his specification we can break down the components of the chart as follows.

<sup>49</sup><https://leanpub.com/D3-Tips-and-Tricks>

<sup>50</sup>[http://www.perceptualedge.com/articles/misc/Bullet\\_Graph\\_Design\\_Spec.pdf](http://www.perceptualedge.com/articles/misc/Bullet_Graph_Design_Spec.pdf)

<sup>51</sup><http://www.perceptualedge.com/>



#### Text label:

Identifies the performance measure (the specific measured value) being represented.

#### Quantitative scale:

A scale that is an analogue of the scale on the x axis of a two dimensional xy graph.

#### Performance measure:

The specific measured value being displayed. In this case the system load on our CPU.

#### Comparative marker:

A reference symbol designating a measurement such as the previous day's high value (or similar).

#### Qualitative ranges:

These represent ranges such as low, medium and high or bad, satisfactory and good. Ideally there would be no fewer than two and no more than 5 of these (for the purposes of readability).

Understanding the specification for the chart is useful, because it's also reflected in the way that the data for the chart is structured.

For instance, If we take the CPU load example above, the data can be presented (in [JSON](#)) as follows;

```
[
  {
    "title": "CPU Load",
    "subtitle": "System Load",
    "ranges": [0.75, 1, 2],
    "measures": [0.17],
    "markers": [1.1]
  }
]
```

Here we can see all the components for the chart laid out and it's these values that we will load into our D3 script to display.

## The Code

As was outlined earlier, the code comes in two parts. The HTML / JavaScript display code and the PHP gather the data code.

First we'll look at the HTML web page.

### HTML / JavaScript

We'll move through the explanation of the code in a similar process to the other examples in the book. Where there are areas that we have covered before, I will gloss over some details on the understanding that you will have already seen them explained in an earlier section. This code can be downloaded as sys\_info.html with the code examples available with the book.

Here is the full code

```
<!DOCTYPE html>
<meta charset="utf-8">
<style>

<style>

body {
  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
  margin: auto;
  padding-top: 40px;
  position: relative;
  width: 800px;
}

.bullet { font: 10px sans-serif; }
.bullet .marker { stroke: #000; stroke-width: 2px; }
.bullet .tick line { stroke: #666; stroke-width: .5px; }
.bullet .range.s0 { fill: #eee; }
.bullet .range.s1 { fill: #ddd; }
.bullet .range.s2 { fill: #ccc; }
.bullet .measure.s0 { fill: steelblue; }
.bullet .title { font-size: 14px; font-weight: bold; }
.bullet .subtitle { fill: #999; }

</style>
```

```
<script src="http://d3js.org/d3.v3.min.js"></script>
<script src="bullet.js"></script>
<script>

var margin = {top: 5, right: 40, bottom: 20, left: 130},
    width = 800 - margin.left - margin.right,
    height = 50 - margin.top - margin.bottom;

var chart = d3.bullet()
    .width(width)
    .height(height);

d3.json("sys_info.php", function(error, data) {
  var svg = d3.select("body").selectAll("svg")
    .data(data)
    .enter().append("svg")
      .attr("class", "bullet")
      .attr("width", width + margin.left + margin.right)
      .attr("height", height + margin.top + margin.bottom)
    .append("g")
      .attr("transform",
            "translate(" + margin.left + "," + margin.top + ")");
  call(chart);

  var title = svg.append("g")
    .style("text-anchor", "end")
    .attr("transform", "translate(-6," + height / 2 + ")");

  title.append("text")
    .attr("class", "title")
    .text(function(d) { return d.title; });

  title.append("text")
    .attr("class", "subtitle")
    .attr("dy", "1em")
    .text(function(d) { return d.subtitle; });

  setInterval(function() {
    updateData();
  }, 60000);
});

function updateData() {
  d3.json("sys_info.php", function(error, data) {
    d3.select("body").selectAll("svg").datum(function (d, i) {
```

```

        d.ranges = data[i].ranges;
        d.measures = data[i].measures;
        d.markers = data[i].markers;
        return d;})
        .call(chart.duration(1000));
    });
}

</script>
</body>

```



It will become clearer in the process of going through the code below, but as a teaser, it is worth noting that while the code that we will modify is as presented above, we are employing a separate script `bullet.js` to enable the charts.

The first block of our code is the start of the file and sets up our HTML.

```

<!DOCTYPE html>
<meta charset="utf-8">
<style>

```

This leads into our style declarations.

```

body {
  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
  margin: auto;
  padding-top: 40px;
  position: relative;
  width: 800px;
}

.bullet { font: 10px sans-serif; }
.bullet .marker { stroke: #000; stroke-width: 2px; }
.bullet .tick line { stroke: #666; stroke-width: .5px; }
.bullet .range.s0 { fill: #eee; }
.bullet .range.s1 { fill: #ddd; }
.bullet .range.s2 { fill: #ccc; }
.bullet .measure.s0 { fill: steelblue; }
.bullet .title { font-size: 14px; font-weight: bold; }
.bullet .subtitle { fill: #999; }

```

We declare the (general) styling for the chart page in the first instance and then we move on to the more interesting styling for the bullet charts.

The first line `.bullet { font: 10px sans-serif; }` sets the font size.

The second line sets the colour and width of the symbol marker. Feel free to play around with the values in any of these properties to get a feel for what you can do. Try this for a start;

```
.bullet .marker { stroke: red; stroke-width: 10px; }
```

We then do a similar thing for the tick marks for the scale at the bottom of the graph.

The next three lines set the colours for the fill of the qualitative ranges.

```
.bullet .range.s0 { fill: #eee; }
.bullet .range.s1 { fill: #ddd; }
.bullet .range.s2 { fill: #ccc; }
```

You can have more or fewer ranges set here, but to use them you also need the appropriate values in your data file. We will explore how to change this later.

The next line designates the colour for the value being measured.

```
.bullet .measure.s0 { fill: steelblue; }
```

Like the qualitative ranges, we can have more of them, but in my personal opinion, it starts to get a bit confusing.

The final two lines lay out the styling for the label.

The next block of code loads the JavaScript files.

```
<script src="http://d3js.org/d3.v3.min.js"></script>
<script src="bullet.js"></script>
```

In this case it's d3 and bullet.js. We need to load bullet.js as a separate file since it exists outside the code base of the d3.js 'kernel'. The file itself can be found [here](#)<sup>52</sup> and it is part of a wider group of [plugins](#)<sup>53</sup> that are used by d3.js. Place the file in the same directory as the sys\_info.html page for simplicities sake.

Then we get into the JavaScript. The first thing we do is define the size of the area that we'll be working in.

```
var margin = {top: 5, right: 40, bottom: 20, left: 130},
    width = 800 - margin.left - margin.right,
    height = 50 - margin.top - margin.bottom;
```

Then we define the chart size using the variables that we have just set up.

---

<sup>52</sup><https://raw.githubusercontent.com/d3/d3-plugins/master/bullet/bullet.js>

<sup>53</sup><https://github.com/d3/d3-plugins>

```
var chart = d3.bullet()
  .width(width)
  .height(height);
```

The other important thing that occurs while setting up the chart is that we use the `d3.bullet` function call to do it. The `d3.bullet` function is the part that resides in the `bullet.js` file that we loaded earlier. The internal workings of `bullet.js` are a window into just how developers are able to craft extra code to allow additional functionality for `d3.js`.

Then we load our JSON data with our external script (which we haven't explained yet) called `sys_info.php`. As a brief spoiler, `sys_info.php` is a script that will query our database and return the latest values in a JSON format. Hence, when it is called as below, it will provide correctly formatted data.

```
d3.json("sys_info.php", function(error, data) {
```

The next block of code is the most important IMHO, since this is where the chart is drawn.

```
var svg = d3.select("body").selectAll("svg")
  .data(data)
  .enter().append("svg")
    .attr("class", "bullet")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
  .append("g")
    .attr("transform",
      "translate(" + margin.left + ", " + margin.top + ")")
  .call(chart);
```

However, to look at it you can be forgiven for wondering if it's doing anything at all.

We use our `.select` and `.selectAll` statements to designate where the chart will go (`d3.select("body").select`) and then load the data as `data` (`.data(data)`).

We add in a `svg` element (`.enter().append("svg")`) and assign the styling from our css section (`.attr("class", "bullet")`).

Then we set the size of the `svg` container for an individual bullet chart using `.attr("width", width + margin.left + margin.right)` and `.attr("height", height + margin.top + margin.bottom)`.

We then group all the elements that make up each individual bullet chart with `.append("g")` before placing the group in the right place with `.attr("transform", "translate(" + margin.left + ", " + margin.top + ")")`.

Then we wave the magic wand and call the `chart` function with `.call(chart)`; which will take all the information from our data file ( like the `ranges`, `measures` and `markers` values) and use the `bullet.js` script to create a chart.

The reason I made the comment about the process looking like magic is that the vast majority of the heavy lifting is done by the `bullet.js` file. Because it's abstracted away from the immediate code that we're writing, it looks simplistic, but like all good things, there needs to be a lot of complexity to make a process look simple.

We then add the titles.

```
var title = svg.append("g")
    .style("text-anchor", "end")
    .attr("transform", "translate(-6," + height / 2 + ")");

title.append("text")
    .attr("class", "title")
    .text(function(d) { return d.title; });

title.append("text")
    .attr("class", "subtitle")
    .attr("dy", "1em")
    .text(function(d) { return d.subtitle; });
```

We do this in stages. First we create a variable `title` which will append objects to the grouped element created above (`var title = svg.append("g")`). We apply a style (`.style("text-anchor", "end")`) and transform to the objects (`.attr("transform", "translate(-6," + height / 2 + ")");`).

Then we append the `title` and `subtitle` data (from our JSON file) to our chart with a modicum of styling and placement.

Lastly (inside the main part of the code) we set up a repeating function that calls another function (`updateData`) every 60000ms. (every minute)

```
setInterval(function() {
    updateData();
}, 60000);
```

Lastly we declare the function (`updateData`) which reads in our JSON file again, selects all the `svg` elements then updates all the `.ranges`, `.measures` and `.markers` data with whatever was in the file. Then it calls the `chart` function that updates the bullet charts (and it lets the change take 1000 milliseconds).

```

function updateData() {
    d3.json("sys_info.php", function(error, data) {
        d3.select("body").selectAll("svg")
            .datum(function(d, i) {
                d.ranges = data[i].ranges;
                d.measures = data[i].measures;
                d.markers = data[i].markers;
                return d;
            })
            .call(chart.duration(1000));
    });
}

```

## PHP

As mentioned earlier, our html code requires JSON formatted data to be gathered and we have named the file that will do this job for us `sys_info.php`. This file is per the code below and it can be downloaded as `sys_info.php` with the downloads available with the book.

We will want to put this code in the same directory as the `sys_info.py` and `bullet.js` files, so they can find each other easily.

```

<?php

$hostname = 'localhost';
$username = 'pi_select';
$password = 'xxxxxxxxxx';

try {
    $dbh = new PDO("mysql:host=$hostname;dbname=measurements",
                   $username, $password);

    /**
     * The SQL SELECT statement */
    $sth = $dbh->prepare("
        SELECT *
        FROM `system_info`
        ORDER BY `dtg` DESC
        LIMIT 0,1
    ");
    $sth->execute();

    /* Fetch all of the remaining rows in the result set */
    $result = $sth->fetchAll(PDO::FETCH_ASSOC);

    /**
     * close the database connection */
    $dbh = null;
}

```

```

}

catch(PDOException $e)
{
    echo $e->getMessage();
}

$bullet_json[0][ 'title' ] = "CPU Load";
$bullet_json[0][ 'subtitle' ] = "System load";
$bullet_json[0][ 'ranges' ] = array(0.75,1.0,2);
$bullet_json[0][ 'measures' ] = array($result[0][ 'load' ]);
$bullet_json[0][ 'markers' ] = array(1.1);

$bullet_json[1][ 'title' ] = "Memory Used";
$bullet_json[1][ 'subtitle' ] = "%";
$bullet_json[1][ 'ranges' ] = array(85,95,100);
$bullet_json[1][ 'measures' ] = array($result[0][ 'ram' ]);
$bullet_json[1][ 'markers' ] = array(75);

$bullet_json[2][ 'title' ] = "Disk Used";
$bullet_json[2][ 'subtitle' ] = "%";
$bullet_json[2][ 'ranges' ] = array(85,95,100);
$bullet_json[2][ 'measures' ] = array($result[0][ 'disk' ]);
$bullet_json[2][ 'markers' ] = array(50);

$bullet_json[3][ 'title' ] = "CPU Temperature";
$bullet_json[3][ 'subtitle' ] = "Degrees C";
$bullet_json[3][ 'ranges' ] = array(40,60,80);
$bullet_json[3][ 'measures' ] = array($result[0][ 'temperature' ]);
$bullet_json[3][ 'markers' ] = array(50);

echo json_encode($bullet_json);

?>

```

The PHP block at the start of the code is mostly the same as our example code for our single temperature measurement project. The difference however is in the select statement.

```

SELECT *
FROM `system_info`
ORDER BY `dtg` DESC
LIMIT 0,1

```

It is selecting all the columns in our table, but by ordering the rows by date/time with the most recent at the top and then limiting the returned rows to only one, we get the single, latest row of data returned.

+ Options	← T →	load	ram	disk	temperature	dtg
	Edit	Inline Edit	Copy	Delete	0.25	94 38 37.9 2015-01-31 08:59:02

Returned system data

Most of the remainder of the script assigns the appropriate values to our array of data `bullet_json`.

If we consider the required format of our JSON data...

```
[  
  {  
    "title": "CPU Load",  
    "subtitle": "System Load",  
    "ranges": [0.75, 1, 2],  
    "measures": [0.17],  
    "markers": [1.1]  
  }  
]
```

...we can see that in our code, we are adding in our title...

```
$bullet_json[0][ 'title' ] = "CPU Load";
```

... our subtitle...

```
$bullet_json[0][ 'subtitle' ] = "System load";
```

... our ranges...

```
$bullet_json[0][ 'ranges' ] = array(0.75, 1.0, 2);
```

... our system load value as returned from the MySQL query...

```
$bullet_json[0][ 'measures' ] = array($result[0][ 'load' ]);
```

... and our marker(s).

```
$bullet_json[0][ 'markers' ] = array(1.1);
```

This data is added for each chart that we will have displayed as a separate element in the `bullet_json` array.

Finally, we echo our JSON encoded data so that when `sys_info.php` is called, all `d3.js` ‘sees’ is correctly formatted data.

```
echo json_encode($bullet_json);
```

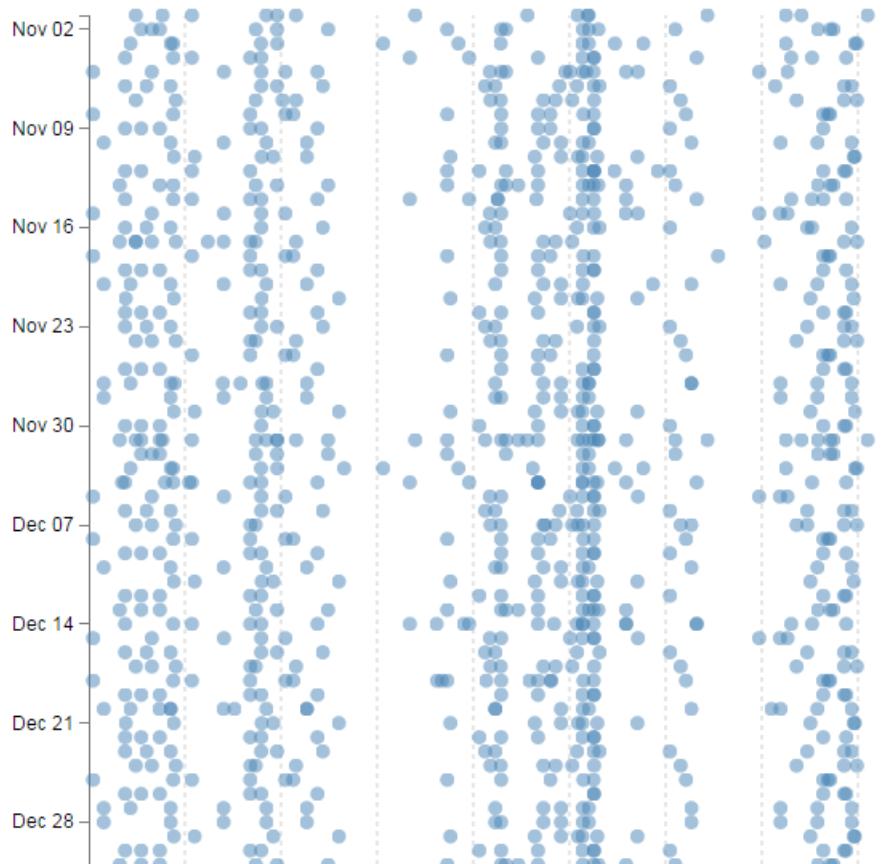
Now every 60 seconds, the d3.js code in the sys\_info.html script calls the sys\_info.php script that queries the MySQL database and gathers our latest system information. That information is collated and formatted and converted into a visual display that will update in a smooth ballet of motion.

As a way of testing that this portion of the code is working correctly, you can use your browser from an external computer and navigate to the sys\_info.php file. This will print out the JSON values directly in your browser window.

# Basic GPIO Input Sensors

This project will use a Hall effect sensor to measure a change in a magnetic field and use that change to trigger the recording of a reading to our database. This type of sensor outputs a logical ‘high’ (1) when triggered and as such we can use the same set-up on our Raspberry Pi for a range of sensors.

The practical application that we will examine is to determine when a cat flap has been opened and using the times that this occurs we will build up a visualization of the pattern of activity that the cat exhibits as it goes in and out the cat flap.



The ‘Catterplot’ Graph



## The Hall Effect

The Hall effect is the production of a voltage difference (the Hall voltage) across an electrical conductor, transverse to an electric current in the conductor and a magnetic field perpendicular to the current. It was discovered by Edwin Hall in 1879.

A Hall effect sensor is a transducer that varies its output voltage in response to a magnetic field. Hall effect sensors are used for a range of measurement functions including proximity switching, positioning, speed detection, and current sensing applications.

For this project we will use a Hall effect sensor combined with circuitry that allows the device to act as a switch. In particular we will use the pre-built sensor 'KY003' which incorporates a series 3144 integrated circuit that contains the hall effect sensor proper (and some supporting circuitry). When the sensor is exposed to a high density magnetic field the circuit will produce a logic 'high' output. The 'KY003' sensor is widely available at an extremely low cost (approximately \$2). Just give the part number a Googling for possible sources.

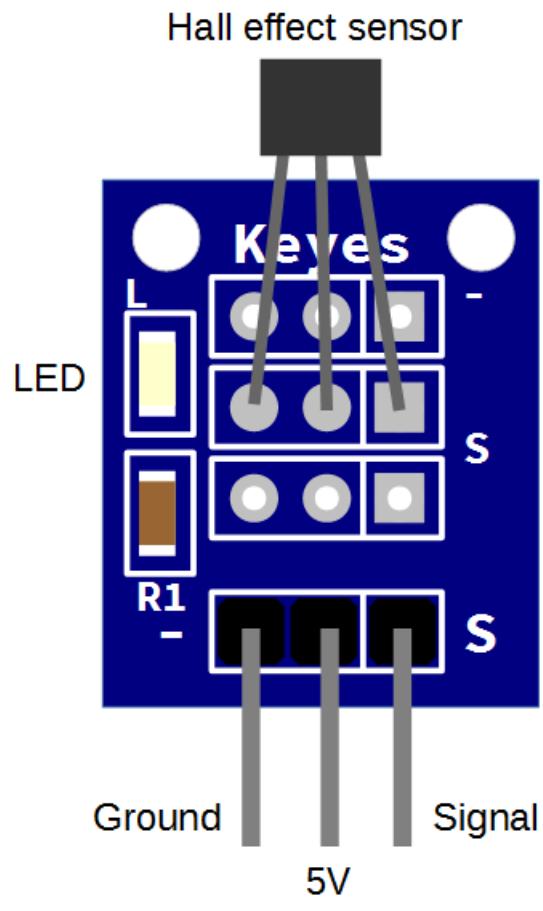
# Measure

## Hardware required

- KY003 Hall effect sensor
- 3 x Jumper cables
- A magnet (small Rare Earth type would be ideal)

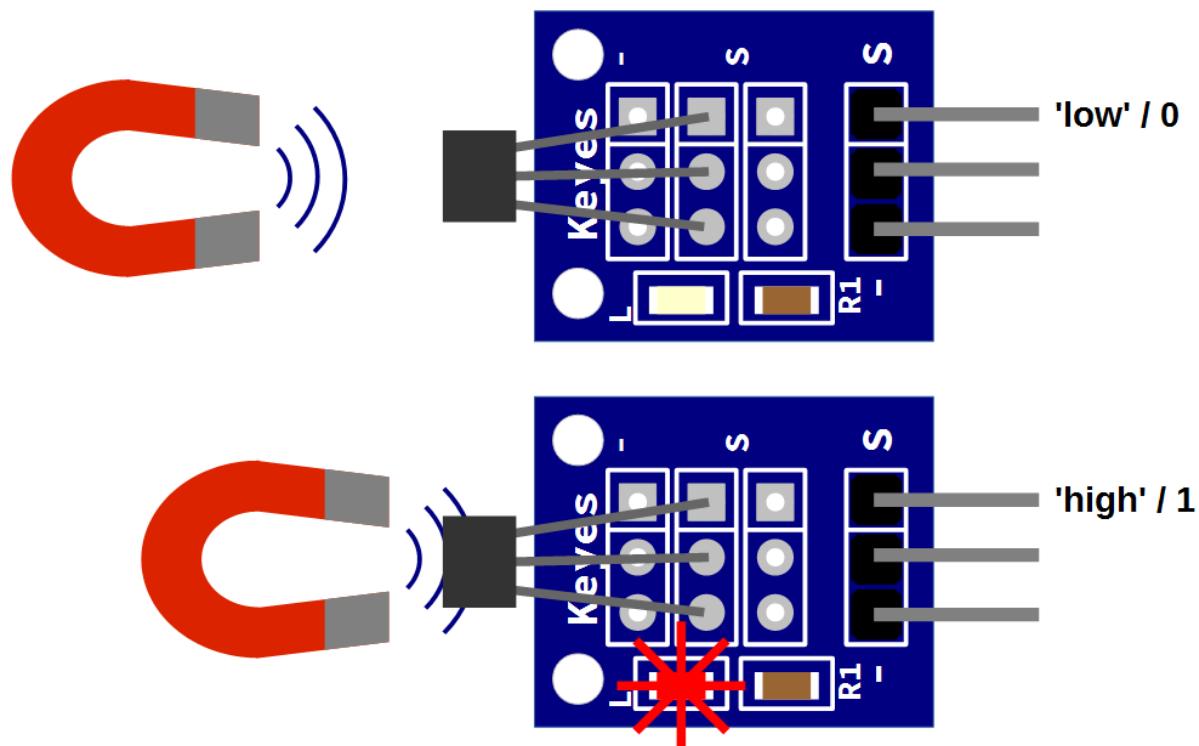
## The KY003 Hall Effect Sensor

The KY003 Hall effect sensor has three connections that we will need to connect to a 5V supply, a ‘Ground’ and a GPIO pin to read the signal from the sensor.



Keyes KY003 Hall Effect Sensor

It also incorporates a surface mounted LED that will illuminate when the sensor detects the presence of a magnetic field and is triggered.

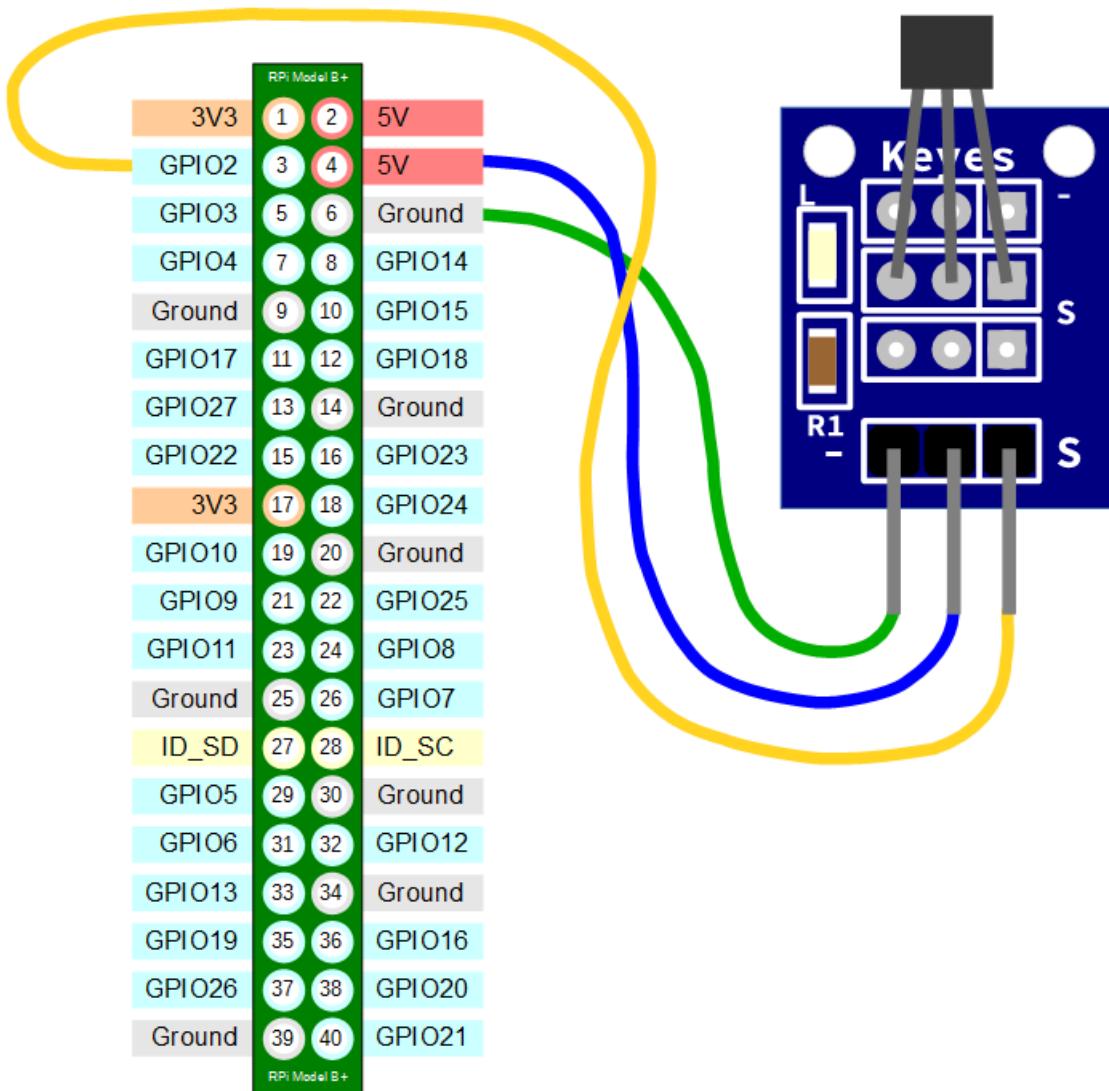


Triggering the Hall Effect Sensor

## Connect

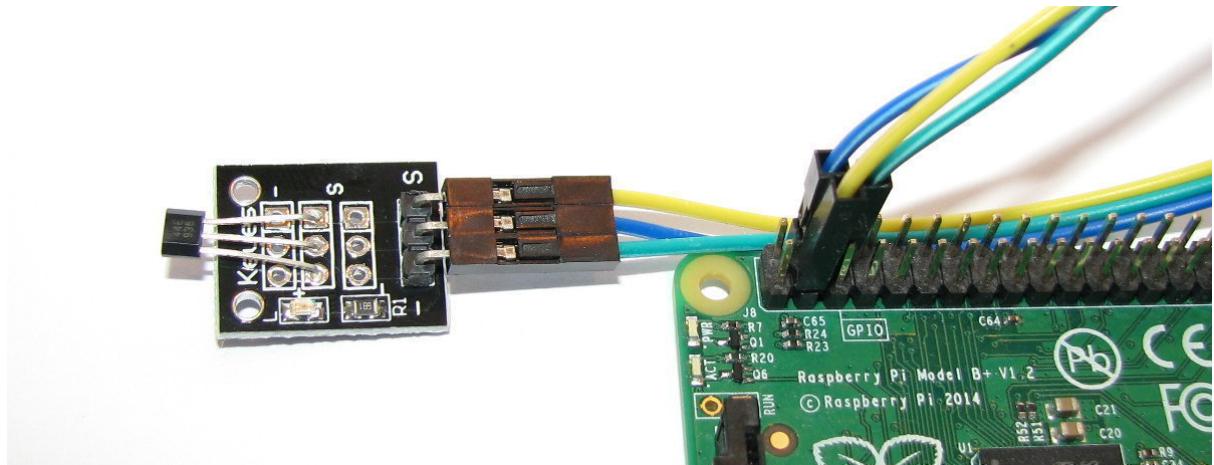
The KY003 sensor should be connected with ground pin to a ground connector, the 5V pin to a 5V connector and the signal pin to a GPIO pin on the on the Raspberry Pi's connector block. In the connection diagram below the ground is connected to pin 6, the 5V is connected to pin 4 and the signal is connected to pin 3 (GPIO 2)

The following diagram is a simplified view of the connection.



Hall Effect Sensor Connection

Connecting the sensor practically can be achieved in a number of ways. You could use a Pi Cobbler break out connector mounted on a bread board connected to the GPIO pins. But because the connection is relatively simple we could build a minimal configuration that will plug directly onto the appropriate GPIO pins using header connectors and jumper wire. The image below shows how simple this can be.



Physical Connection of Hall Effect Sensor

## Test

Once correctly connected, the sensor is ready to go. It can be simply tested by bringing a magnet into close proximity with the sensor and the LED should illuminate. When this occurs, we can make the assumption (at this stage) that the sensor is working and has placed ‘high’ or ‘1’ on the signal pin of the sensor and therefore is presenting a ‘high’ or 1 to GPIO 2 on the Pi.

## Record

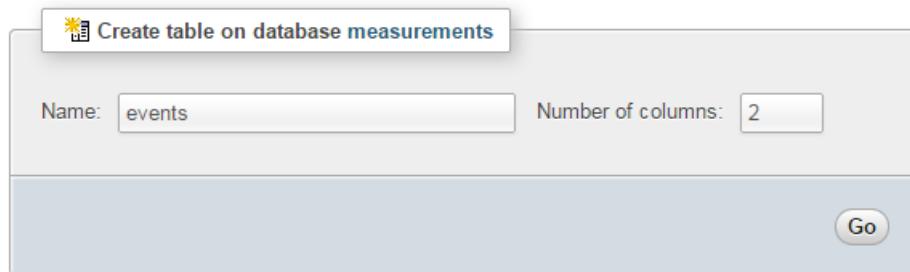
To record this data we will use a Python program that uses our sensor like a trigger and writes an identifier for the sensor into our MySQL database. At the same time a time stamp will be added automatically. You may be wondering why we’re not recording the value of the sensor (0 or 1) and that’s a valid question. What we are going to use our sensor for is to determine when it has been triggered. For all intents and purposes, we can make the assumption that our sensor represents a device that can be in one of two states. We are going to be interested when it changes state, not when it is steady. In particular we are going to be interested in when it changes from low to high (0 to 1) and therefore represents a ‘rising’ signal.

for this project to work, our Python script has to run continuously and will only write to our database when triggered.

## Database preparation

First we will set up our database table that will store our data.

Using the phpMyAdmin web interface that we set up, log on using the administrator (root) account and select the ‘measurements’ database that we created as part of the initial set-up.



Create the MySQL Table

Enter in the name of the table and the number of columns that we are going to use for our measured values. In the screenshot above we can see that the name of the table is ‘events’ and the number of columns is ‘2’.

We will use two columns so that we can store an event name and the time it was recorded.

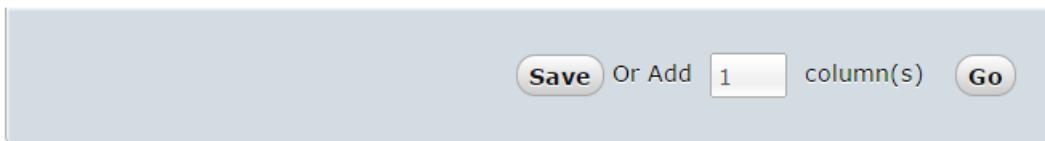
Once we click on ‘Go’ we are presented with a list of options to configure our table’s columns. Don’t be intimidated by the number of options that are presented, we are going to keep the process as simple as practical.

For the first column we can enter the name of the ‘Column’ as ‘dtg’ (short for date time group) the ‘Type’ as ‘TIMESTAMP’ and the ‘Default’ value as ‘CURRENT\_TIMESTAMP’. For the second column we will enter the name ‘event’ and the type is ‘VARCHAR’ with a ‘Length/Values’ of 30.

Table name:		
events		
Structure		
<b>Column</b>	dtg	event
<b>Type</b>	TIMESTAMP	VARCHAR
<b>Length/Values<sup>1</sup></b>	30	
<b>Default<sup>2</sup></b>	CURRENT_TIMESTAMP	None

Configure the MySQL Table Columns

Scroll down a little and click on the ‘Save’ button and we’re done.



### Save the MySQL Table Columns



### Why did we choose those particular settings for our table?

Our ‘dtg’ column needs to store a value of time that includes the date and the time, so the advantage of selecting TIMESTAMP in this case is that we can select the default value to be the current time which means that when we write our data to the table we only need to write the ‘event’ name and the ‘dtg’ will be entered automatically for us. The disadvantage of using ‘TIMESTAMP’ is that it has a more limited range than DATETIME. TIMESTAMP can only have a range between ‘1970-01-01 00:00:01’ to ‘2038-01-19 03:14:07’.

The event names are simply descriptions of the type of events we want to capture, so we will use a variable type ‘VARCHAR’ which is for characters. We can also specify the maximum length of the information stored in the database to make things a little more efficient. In theory we could use the ‘CHAR’ type which is more efficient, but in this instance I prefer ‘VARCHAR’ which will allow the length of the recorded information to be flexible.

## Record the events

The following Python code (which is based on the code that is part of the great blog series on [FIRSIM<sup>54</sup>](#)) is a script which allows us to check the state of a sensor and write an entry to our database when an event occurs.

The full code can be found in the code samples bundled with this book (events.py).

First a quick shout-out and thanks to the reader ‘salmon’ who [kindly suggested<sup>55</sup>](#) a significant change to the original code which changed it from a CPU killing ‘loop of doom’ to a far more well behaved script.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

# Import the python libraries
import RPi.GPIO as GPIO
import logging
import MySQLdb as mdb
import time

logging.basicConfig(filename='/home/event_error.log',
```

<sup>54</sup><http://firmsim.blogspot.com/2014/05/raspberry-pi-en-hall-magnetic-sensor.html>

<sup>55</sup><http://www.d3noob.org/2015/03/raspberry-pi-gpio-sensors-part-2-record.html?showComment=1425631191093#c1776224462758839663>

```
level=logging.DEBUG,
format='%(asctime)s %(levelname)s %(name)s %(message)s')
logger=logging.getLogger(__name__)

# Function called when GPIO.RISING
def storeFunction(channel):
    print("Signal detected")

con = mdb.connect('localhost', \
                  'pi_insert', \
                  'xxxxxxxxxx', \
                  'measurements');

try:
    cur = con.cursor()
    cur.execute("""INSERT INTO events(event) VALUES(%s)""", ('catflap'))
    con.commit()

except mdb.Error, e:
    logger.error(e)

finally:
    if con:
        con.close()

print "Sensor event monitoring (CTRL-C to exit)"

# use the BCM GPIO numbering
GPIO.setmode(GPIO.BCM)

# Define the BCM-PIN number
GPIO_PIR = 2

# Define pin as input with standard high signaal
GPIO.setup(GPIO_PIR, GPIO.IN, pull_up_down = GPIO.PUD_UP)

try:
    # Loop while true = true
    while True :

        # Wait for the trigger then call the function
        GPIO.wait_for_edge(GPIO_PIR, GPIO.RISING)
        storeFunction(2)
        time.sleep(1)

except KeyboardInterrupt:
```

```
# Reset the GPIO settings
GPIO.cleanup()
```

This script can be saved in our home directory (/home/pi) and can be run by typing:

```
sudo python events.py
```

The observant amongst you will notice that we need to run the program as the superuser (by invoking sudo before the command to run events.py). This is because the GPIO library requires the accessing of the GPIO pins to be done by the superuser.

Once the command is run the pi will generate a warning to let us know that there is a pull up resistor fitted to channel 2. That's fine. From here if we move our magnet towards and away from the sensor we should see the Signal detected notification appear in the terminal per below.

```
pi@raspberrypi ~ $ sudo python event.py
Sensor event monitoring (CTRL-C to exit)
event.py:23: RuntimeWarning: A physical pull up resistor is fitted on this ch\ \
annel!
    GPIO.setup(GPIO_PIR, GPIO.IN, pull_up_down = GPIO.PUD_UP)
Signal detected
Signal detected
```

We should then be able to check our MySQL database and see an entry for each time that the sensor triggered.

+ Options		dtg	event
	← →		
<input type="checkbox"/>		2015-02-25 09:18:50	catflap
<input type="checkbox"/>		2015-02-25 09:20:27	catflap
<input type="checkbox"/>		2015-02-25 09:20:28	catflap
<input type="checkbox"/>		2015-02-25 09:20:30	catflap

Save the MySQL Table Columns

## Code Explanation

The script starts by importing the modules that it's going to use for the process of reading and recording the measurements;

```
import RPi.GPIO as GPIO
import logging
import MySQLdb as mdb
import time
```



Python code in one module gains access to the code in another module by the process of importing it. The `import` statement invokes the process and combines two operations; it searches for the named module, then it binds the results of that search to a name in the local scope.

Then the code sets up the `logging module`<sup>56</sup>. We are going to use the `basicConfig()` function to set up the default handler so that any debug messages are written to the file `/home/pi/event_error.log`.

```
logging.basicConfig(filename='/home/event_error.log',
    level=logging.DEBUG,
    format='%(asctime)s %(levelname)s %(name)s %(message)s')
logger=logging.getLogger(__name__)
```

In the following function where we are getting our readings or writing to the database we write to the log file if there is an error.

Which brings us to our function `storeFunction` that will write information to our database when called later;

```
def storeFunction(channel):
    print("Signal detected")

    con = mdb.connect('localhost', \
                      'pi_insert', \
                      'xxxxxxxxxx', \
                      'measurements');

    try:
        cur = con.cursor()
        cur.execute("""INSERT INTO events(event) VALUES(%s)""", ('catflap'))
        con.commit()

    except mdb.Error, e:
        logger.error(e)

    finally:
        if con:
            con.close()
```

---

<sup>56</sup><http://pymotw.com/2/logging/>

This is very much a rinse and repeat of the function found in the single temperature project. We configure our connection details, connect and write the name of this particular sensor ('catflap') into the database (Remember that when we store an event name into the database the timestamp dtg is added to the record automatically.). We then have some house-keeping code that will log any errors and then close the connection to the database.

The program can then issues the GPIO commands that start the interface to the sensor and get it configured (the code below has been condensed for clarity);

```
GPIO.setmode(GPIO.BCM)
GPIO_PIR = 2
GPIO.setup(GPIO_PIR, GPIO.IN, pull_up_down = GPIO.PUD_UP)
```

The GPIO Python module was developed and is maintained by Ben Croston. You can find a wealth of information on its usage at the [official wiki site<sup>57</sup>](#).

GPIO.setmode allows us to tell which convention of numbering the IO pins on the Raspberry Pi we will use when we say that we're going to read a value off one of them. Observant Pi users will have noticed that the GPIO numbers don't match the pin numbers. Believe it or not there is a good reason for this and in the words of the good folks from [raspberrypi.org<sup>58</sup>](#);

*“While there are good reasons for software engineers to use the BCM numbering system (the GPIO pins can do more than just simple input and output), most beginners find the human readable numbering system more useful. Counting down the pins is simple, and you don’t need a reference or have to remember which is which. Take your pick though; as long as you use the same scheme within a program then all will be well.”*

In our case we are using the BCM numbering (`GPIO.setmode(GPIO.BCM)`).

We then assign the GPIO channel as GPIO 2 to a variable (`GPIO_PIR = 2`).

Then we configure how we are going to read the GPIO channel with `GPIO.setup`. We specify the channel with `GPIO_PIR`, whether the channel will be an input or an output (`GPIO.IN`) and we configure the Broadcom SCO to use a pull up resistor in software (`pull_up_down=GPIO.PUD_UP`).

Then the code executes an ‘endless’ loop that asks itself “Is True, True?”. While it is the program essentially pauses while it waits for a signal from the sensor via the GPIO channel.

---

<sup>57</sup>[http://sourceforge.net/p/raspberry-gpio-python/wiki/browse\\_pages/](http://sourceforge.net/p/raspberry-gpio-python/wiki/browse_pages/)

<sup>58</sup><http://www.raspberrypi.org/documentation/usage/gpio/>

```
try:
    # Loop while true = true
    while True :

        # Wait for the trigger then call the function
        GPIO.wait_for_edge(GPIO_PIR, GPIO.RISING)
        storeFunction(2)
        time.sleep(1)
```

Ultimately True stops being True when the program receives a ‘break’ which can occur with a ‘ctrl-c’ keypress.

In the mean time we’re using `wait_for_edge` to look for a specific type of event. We specify the channel (`GPIO_PIR`) we want to look for and the type of change in the channel which is a rising edge (`GPIO.RISING`) (which signifies a change in state (going from 0 to 1)). When this occurs the program progresses and the `storeFunction` function is called (which will record the event in the database). Lastly we let the program sleep for 1 second (`time.sleep(1)`) which reduces the occurrence of false events in the case of the cat flap swinging backwards and forwards when closing. This is the equivalent of ‘debouncing’ a switch.

Finally we use the keyboard interruption of the loop to do some housekeeping and reset our GPOI ports;

```
except KeyboardInterrupt:
    # Reset the GPIO settings
    GPIO.cleanup()
```

This results in any of the GPIO ports that have been used in the program being set back to input mode. This occurs because it is safer (for the equipment) to leave the ports as inputs which removes any extraneous voltages.

## Start the code automatically at boot

While we can run our script easily from the command line, this is not going to be convenient when we deploy our cat flap activity logger. The alternative is to automatically start the script using `rc.local` in a similar way that we did with ‘tightvncserver’ in our initial set-up.

We will add the following command into `rc.local`;

```
python /home/pi/events.py
```

This command looks slightly different for the way that we have been running the script so far (`sudo python events.py`). This is because we do not need to use `sudo` (since `rc.local` runs as as the root user already, and we need to specify the full path to the script (`/home/pi/events.py`) as there is no environment set up in a home directory or similar (i.e. we’re not starting from `/home/pi/`).

To do this we will edit the `rc.local` file with the following command;

```
sudo nano /etc/rc.local
```

Add in our lines so that the file looks like the following;

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
    printf "My IP address is %s\n" "$_IP"
fi

# Start tightvncserver
su - pi -c '/usr/bin/tightvncserver :1'

# Start the event monitoring script
python /home/pi/events.py

exit 0
```

(We can also add our own comment into the file to let future readers know what's going on)

That should be it. We should now be able to test that the service starts when the Pi boots by typing in;

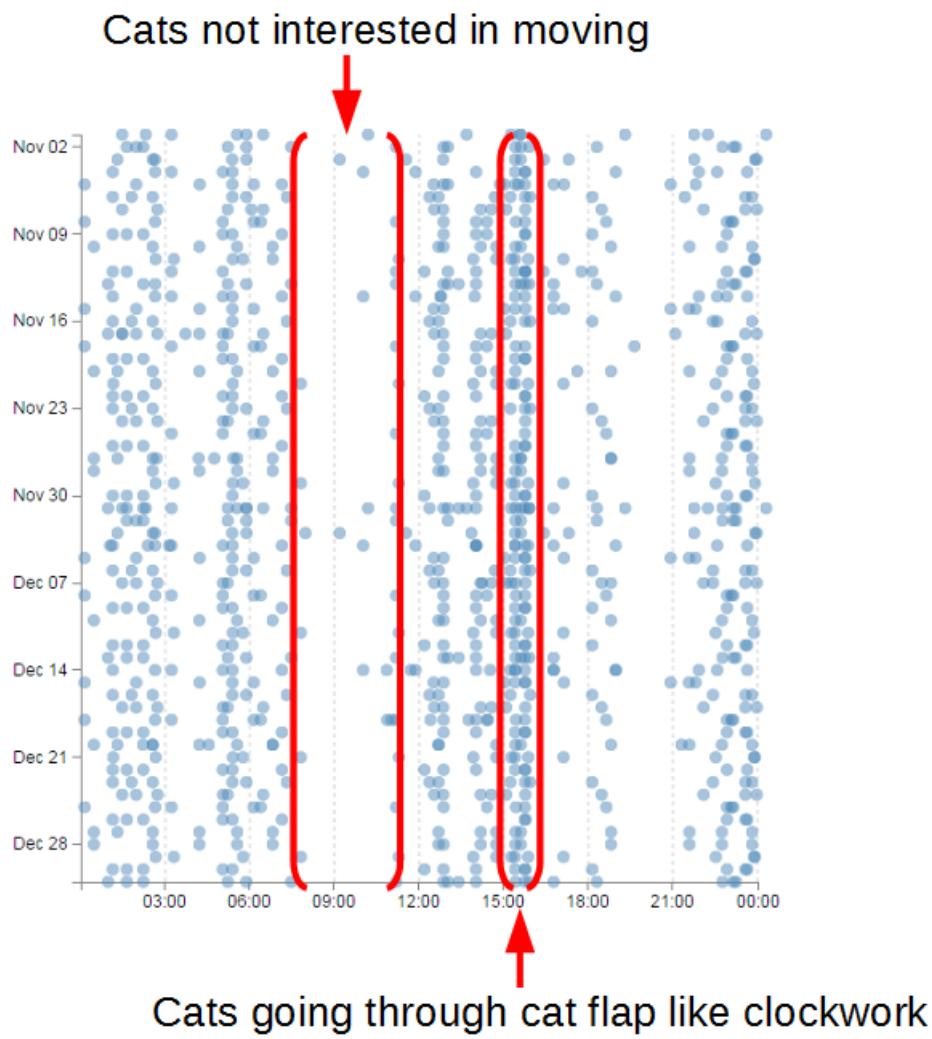
```
sudo reboot
```

Then check our MySQL database to see the events increment as we wave our magnet in front of the Hall effect sensor.

Nice job! We're measuring and recording a change in a magnetic field!

## Explore

This section has a working solution for presenting data from events,. This is done via a scatter-plot type matrix (hereby referred to as the ‘catter-plot’ as it involves measuring cats going through cat doors) that is slightly different to those that would normally be used. Typically a scatter-plot with use time on the x axis and a value on the Y axis. This example will use the time of day on the X axis, independent of the date and the Y axis will represent the date independent of the time of day. The end result is a scatter-plot where activities that occur on a specific day are seen on a horizontal line and the time of day that these activities occur can form a pattern that the brain can determine fairly easily.



The ‘Catterplot’ Graph

We can easily see that wherever the cats are between approximately 7:30 and 11am they’re not likely to be using the cat door. However, something happens at around 3:30pm and it’s almost certain that they will be coming through the cat flap.

This is a slightly more complex use of JavaScript and d3.js specifically but it is a great platform that demonstrates several powerful techniques for manipulating and presenting data.

It has the potential to be coupled with additional events that could be colour coded and / or they could be sized according to frequency.

## The Code

The following code is a PHP file that we can place on our Raspberry Pi's web server (in the `/var/www` directory) that will allow us to view all of the results that have been recorded in the temperature directory on a graph;



There are many sections of the code which have been explained already in the set-up section of the book that describes a simple line graph for a single temperature measurement. Where these occur we will be less thorough with the explanation of how the code works.

The full code can be found in the code samples bundled with this book (`events.php`).

```
<?php

$hostname = 'localhost';
$username = 'pi_select';
$password = 'xxxxxxxxxx';

try {
    $dbh = new PDO("mysql:host=$hostname;dbname=measurements",
                   $username, $password);

    /*** The SQL SELECT statement ***/
    $sth = $dbh->prepare("SELECT dtg
                          FROM `events`");
    $sth->execute();

    /* Fetch all of the remaining rows in the result set */
    $result = $sth->fetchAll(PDO::FETCH_ASSOC);

    /*** close the database connection ***/
    $dbh = null;
}

catch(PDOException $e)
{
    echo $e->getMessage();
}
```

```
$json_data = json_encode($result);

?>

<!DOCTYPE html>
<meta charset="utf-8">
<style>

body { font: 12px sans-serif; }

.axis path,
.axis line {
  fill: none;
  stroke: grey;
  shape-rendering: crispEdges;
}

.dot { stroke: none; fill: steelblue; }

.grid .tick { stroke: lightgrey; opacity: 0.7; }
.grid path { stroke-width: 0; }

</style>
<body>
<script src="http://d3js.org/d3.v3.min.js"></script>
<script>

// Get the data
<?php echo "data=". $json_data ."?" ?>

// Parse the date / time formats
parseDate = d3.time.format("%Y-%m-%d").parse;
parseTime = d3.time.format("%H:%M:%S").parse;

data.forEach(function(d) {
  dtgSplit = d.dtg.split(" ");      // split on the space
  d.date = parseDate(dtgSplit[0]); // get the date seperately
  d.time = parseTime(dtgSplit[1]); // get the time separately
});

// Get the number of days in the date range to calculate height
var oneDay = 24*60*60*1000; // hours*minutes*seconds*milliseconds
var dateStart = d3.min(data, function(d) { return d.date; });
var dateFinish = d3.max(data, function(d) { return d.date; });
var numberDays = Math.round(Math.abs((dateStart.getTime() -
```

```
        dateFinish.getTime()/(oneDay))));

var margin = {top: 40, right: 20, bottom: 30, left: 100},
    width = 600 - margin.left - margin.right,
    height = numberDays * 8;

var x = d3.time.scale().range([0, width]);
var y = d3.time.scale().range([0, height]);

var xAxis = d3.svg.axis()
    .scale(x)
    .orient("bottom")
    .ticks(7)
    .tickFormat(d3.time.format("%H:%M"));

var yAxis = d3.svg.axis()
    .scale(y)
    .orient("left")
    .ticks(7,0,0);

var svg = d3.select("body")
    .append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
    .append("g")
    .attr("transform", "translate(" + margin.left + ","
        + margin.top + ")");

// State the functions for the grid
function make_x_axis() {
    return d3.svg.axis()
        .scale(x)
        .orient("bottom")
        .ticks(7)
}

// Set the domains
x.domain([new Date(1899, 12, 01, 0, 0, 1),
          new Date(1899, 12, 02, 0, 0, 0)]);
y.domain(d3.extent(data, function(d) { return d.date; }));

// tickSize: Get or set the size of major, minor and end ticks
svg.append("g").classed("grid x_grid", true)
    .attr("transform", "translate(0," + height + ")")
    .style("stroke-dasharray", ("3, 3, 3"))
    .call(make_x_axis())
```

```
.tickSize(-height, 0, 0)
.tickFormat(""))

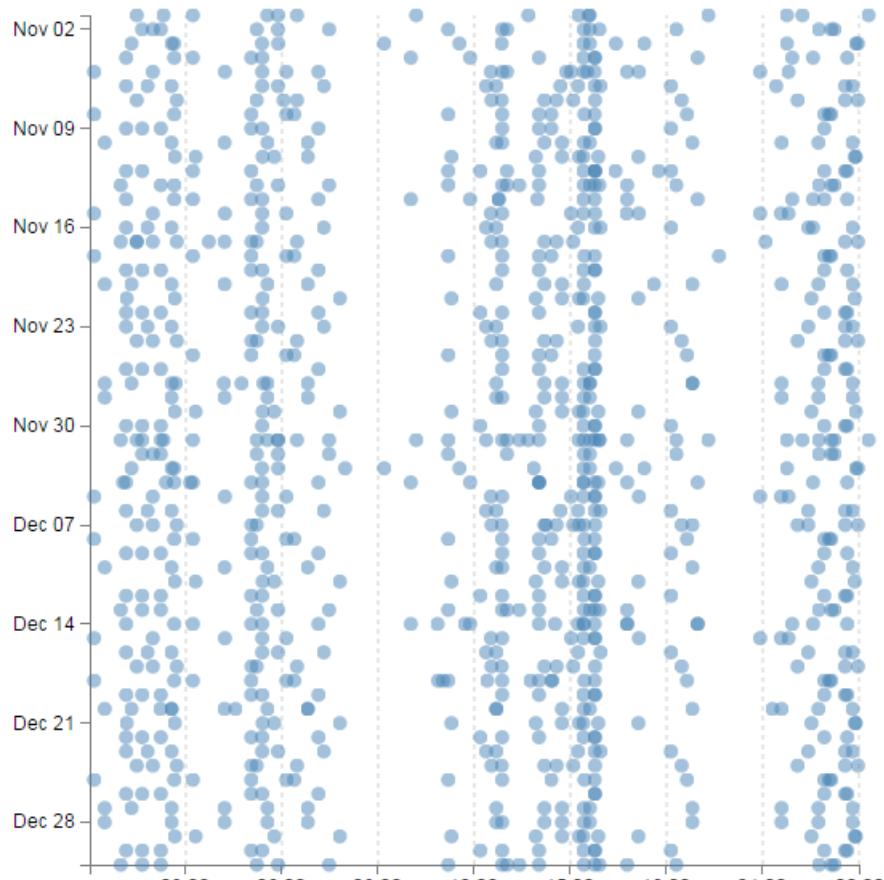
// Draw the Axes and the tick labels
svg.append("g")
    .attr("class", "x axis")
    .attr("transform", "translate(0," + height + ")");
    .call(xAxis)
    .selectAll("text")
        .style("text-anchor", "middle");

svg.append("g")
    .attr("class", "y axis")
    .call(yAxis)
    .selectAll("text")
        .style("text-anchor", "end");

// draw the plotted circles
svg.selectAll(".dot")
    .data(data)
    .enter().append("circle")
        .attr("class", "dot")
        .attr("r", 4.5)
        .style("opacity", 0.5)
        .attr("cx", function(d) { return x(d.time); })
        .attr("cy", function(d) { return y(d.date); });

</script>
</body>
```

The graph that will look a little like this (except the data will be different of course).



The 'Catterplot' Graph

This is a fairly basic graph (i.e, there is no title or labelling of axis).

The code will automatically try to collect as many events as are in the database, so depending on your requirements we may need to vary the query. As some means of compensation it will automatically increase the vertical size of the graph depending on how many days the data spans.

## PHP

The PHP block at the start of the code is mostly the same as our example code for our single temperature measurement project. The significant difference however is in the select statement.

```
SELECT dtg
FROM `events`
LIMIT 0,900
```

Here we are only returning a big list of date / time values.

## CSS (Styles)

There are a range of styles that are applied to the elements of the graphic.

```

body { font: 12px sans-serif; }

.axis path,
.axis line {
  fill: none;
  stroke: grey;
  shape-rendering: crispEdges;
}

.dot { stroke: none; fill: steelblue; }

.grid .tick { stroke: lightgrey; opacity: 0.7; }
.grid path { stroke-width: 0; }

```

We set a default text font and size, some formatting for our axes and grid lines and the colour and type of outline (none) that our dots for our events have.

## JavaScript

The code has very similar elements to our single temperature measurement script and comparing both will show us that we are doing similar things in each graph. Interestingly, this code mixes the sequence of some of the ‘blocks’ of code. This is in order to allow the dynamic adjustment of the vertical size of the graph.

The very first thing we do with our JavaScript is to use our old friend PHP to declare our data;

```
<?php echo "data=".json_data." ?>
```

Then we declare the two functions we will use to format our time values;

```

parseDate = d3.time.format("%Y-%m-%d").parse;
parseTime = d3.time.format("%H:%M:%S").parse;

```

`parseDate` will format any date values and `parseTime` will format any time values.

Then we cycle through our data using a `forEach` statement;

```

data.forEach(function(d) {
  dtgSplit = d.dtg.split(" "); // split on the space
  d.date = parseDate(dtgSplit[0]); // get the date seperately
  d.time = parseTime(dtgSplit[1]); // get the time seperately
});

```

In this loop we split our `dtg` value into date and time portions and then use our `parse` statements to ensure that they are correctly formatted.

We then do a little bit of date / time maths to work out how many days are between the first day in our range of data and the last day;

```
var oneDay = 24*60*60*1000; // hours*minutes*seconds*milliseconds
var dateStart = d3.min(data, function(d) { return d.date; });
var dateFinish = d3.max(data, function(d) { return d.date; });
var numberDays = Math.round(Math.abs((dateStart.getTime() -
dateFinish.getTime())/(oneDay)));
```

We set up the size of the graph and the margins (this is where we adjust the height of the graph depending on the number of days);

```
var margin = {top: 40, right: 20, bottom: 30, left: 100},
width = 600 - margin.left - margin.right,
height = numberDays * 8;
```

The scales and ranges for both axes are both time based in this example;

```
var x = d3.time.scale().range([0, width]);
var y = d3.time.scale().range([0, height]);
```

And we set up the x axis and y axis accordingly;

```
var xAxis = d3.svg.axis()
.scale(x)
.orient("bottom")
.ticks(7)
.tickFormat(d3.time.format("%H:%M"));

var yAxis = d3.svg.axis()
.scale(y)
.orient("left")
.ticks(7,0,0);
```

We then create our svg container with the appropriate width and height taking into account the margins;

```
var svg = d3.select("body")
.append("svg")
.attr("width", width + margin.left + margin.right)
.attr("height", height + margin.top + margin.bottom)
.append("g")
.attr("transform", "translate(" + margin.left + ","
+ margin.top + ")");
```

Then we declare a special function that we will use to make our grid;

```
function make_x_axis() {
    return d3.svg.axis()
        .scale(x)
        .orient("bottom")
        .ticks(7)
}
```

Essentially we will be creating another x axis with lines that extend the full height of the graph. Our domains are set in a bit of an unusual way since our `time` variables have no associated date. Therefore we tell the range to fall over a suitable time range that spans a single day

```
x.domain([new Date(1899, 12, 01, 0, 0, 1),
           new Date(1899, 12, 02, 0, 0, 0)]);
y.domain(d3.extent(data, function(d) { return d.date; }));
```

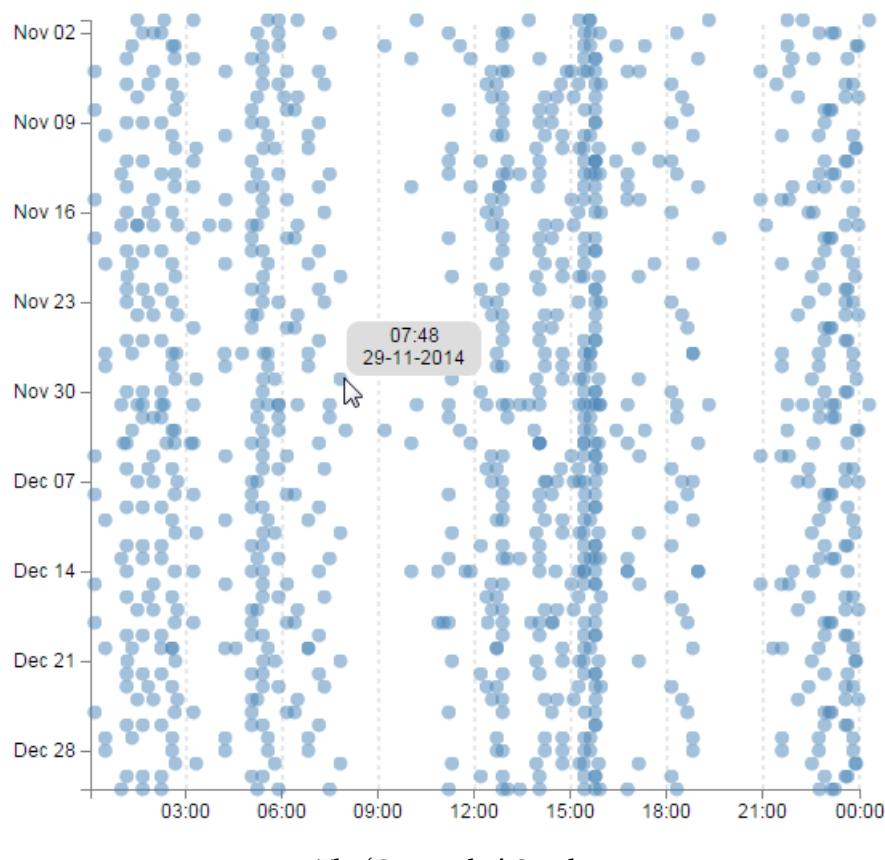
The y domain can exist as normal although is (unusually) a time scale and not ordinal.

Our grid is added as a separate axis with really tall ticks (`tickSize`), no text values (`tickFormat`) and with a dashed line (`stroke-dasharray`);

```
svg.append("g").classed("grid x_grid", true)
    .attr("transform", "translate(0," + height + ")")
    .style("stroke-dasharray", ("3, 3, 3"))
    .call(make_x_axis())
        .tickSize(-height, 0, 0)
        .tickFormat("")
```

Then we do the mundane adding of the axes and plotting the circles;

Wonderful! And as an added bonus you can also find the file ‘events-tips.php’ in the downloads with the book. This file is much the same as the one we have just explained, but also includes a tool-tip feature that shows the time and date of an individual point when our mouse moves over the top of it.



The 'Catterplot' Graph

If you want a closer explanation for this piece of code, download a copy of [D3 Tips and Tricks<sup>59</sup>](#) for this and a whole swag of other information.

---

<sup>59</sup><https://leanpub.com/D3-Tips-and-Tricks>

# Pressure and Temperature measurement with the BMP180

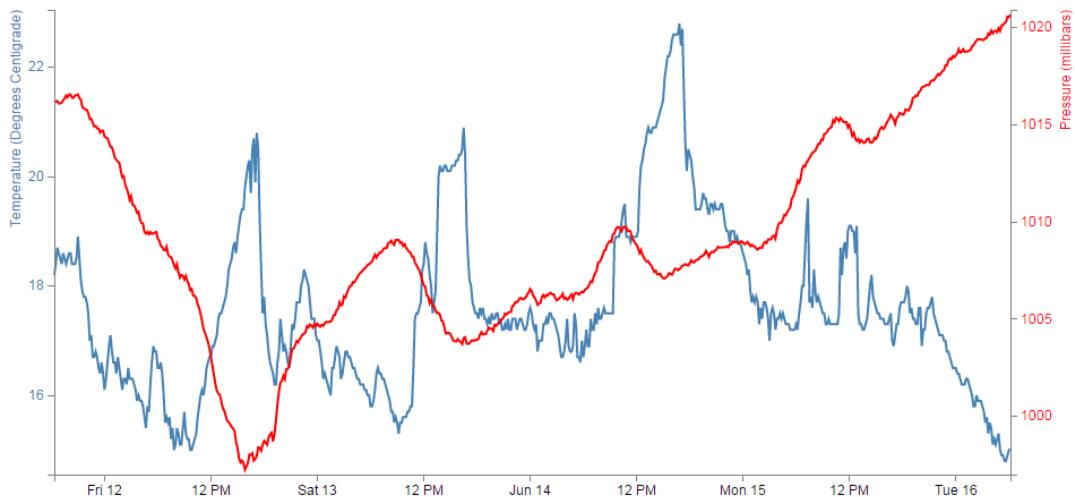
This project will use a BMP180 sensor manufactured by [Bosch Sensortec<sup>60</sup>](#) to measure pressure and temperature. The connection will use the I2C communications protocol.

**i** Inter-Integrated Circuit or I2C (pronounced as either I-squared-C or I-2-C) connection is generically referred to as a “two-wire interface”. It’s commonly used to attach low-speed peripherals to computing devices.

I2C can be used to connect up to 127 nodes via a bus which has two data wires, called SCL and SDA. SCL is the CLock line which is used to synchronize all data transfers over the I2C bus. SDA is the DAta line. The I2C bus works on a ‘Master - Slave’ system where in this case the master is the Raspberry Pi. Slaves can be integrated circuits such as sensors or micro controllers.

When the master wishes to communicate with a slave it sends a series of pulses down the SDA and SCL lines. The data that is sent includes a unique address that identifies the slave with which the master needs to interact. When data is being sent on the SDA line, clock pulses are sent on the SCL line to keep master and slave synchronised.

We will present the data by creating a graph with two lines representing pressure and temperature with different Y axes on the left and the right.



Dual Pressure and Temperature Graph

This project has drawn on a range of sources for information. Where not directly referenced in the text, check out the [Bibliography](#) at the end of the chapter.

<sup>60</sup>[http://www.bosch-sensortec.com/en/homepage/products\\_3/environmental\\_sensors\\_1/bmp180\\_1/bmp180](http://www.bosch-sensortec.com/en/homepage/products_3/environmental_sensors_1/bmp180_1/bmp180)

# Measure

## Hardware required

- A Raspberry Pi (huge range of sources)
- BMP180 based sensor board. Various types are available that will perform in the same way ([Adafruit<sup>61</sup>](#), [sparkfun<sup>62</sup>](#), [The Pi Hut<sup>63</sup>](#), [Deal Extreme<sup>64</sup>](#) (used in this project))
- Female to Female Dupont connector cables ([Deal Extreme<sup>65</sup>](#) or build your own!)

## The BMP180 Sensor

The BMP180 is the a digital barometric pressure sensor made by Bosch Sensortec to support applications in mobile devices, such as smart phones, tablet PCs and sports devices. It is an improvement on an earlier device the BMP085 in terms of size and expansion of digital interfaces.



If you Google ‘BMP180’ you will see that there are a wide range of small sensor boards that can be used with the Raspberry Pi to measure pressure and temperature. The boards are manufactured by different companies to provide connectivity / form factor and other options, but they all use the Bosch Sensortec BMP180 sensor which is a small (3.6 x 3.6 mm), silver unit that will probably be the most prominent component on the board.

The BMP180 is a sensor based on the [piezoresistive effect<sup>66</sup>](#) where a change in the electrical resistivity of a material occurs when mechanical strain is applied.



This is in contrast to the [piezoelectric effect<sup>67</sup>](#) which causes a change in electric potential.

The sensor communicates using the I2C protocol and has the ability to discriminate between pressure changes corresponding to 1m in altitude and temperature changes of 0.1 degree centigrade.

<sup>61</sup><http://www.adafruit.com/products/1603>

<sup>62</sup><https://www.sparkfun.com/products/11824>

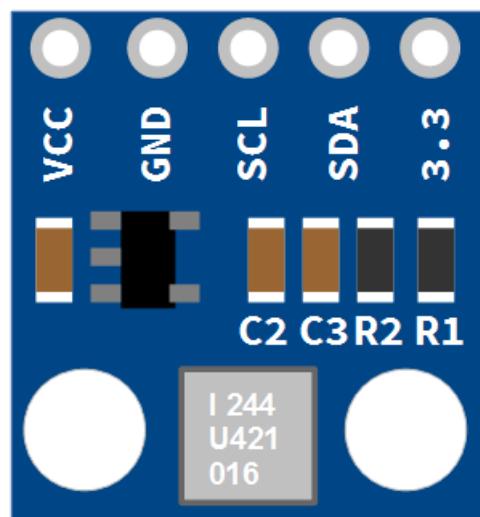
<sup>63</sup><http://thepihut.com/products/bmp180-barometric-pressure-temperature-altitude-sensor>

<sup>64</sup><http://www.dx.com/p/bmp180-bosch-temperature-air-pressure-module-deep-blue-294251>

<sup>65</sup><http://www.dx.com/p/8-pins-female-to-female-dupont-cable-for-raspberry-pi-multicolored-21cm-326450>

<sup>66</sup>[http://en.wikipedia.org/wiki/Piezoresistive\\_effect](http://en.wikipedia.org/wiki/Piezoresistive_effect)

<sup>67</sup><http://en.wikipedia.org/wiki/Piezoelectricity>



BMP180 Sensor Board

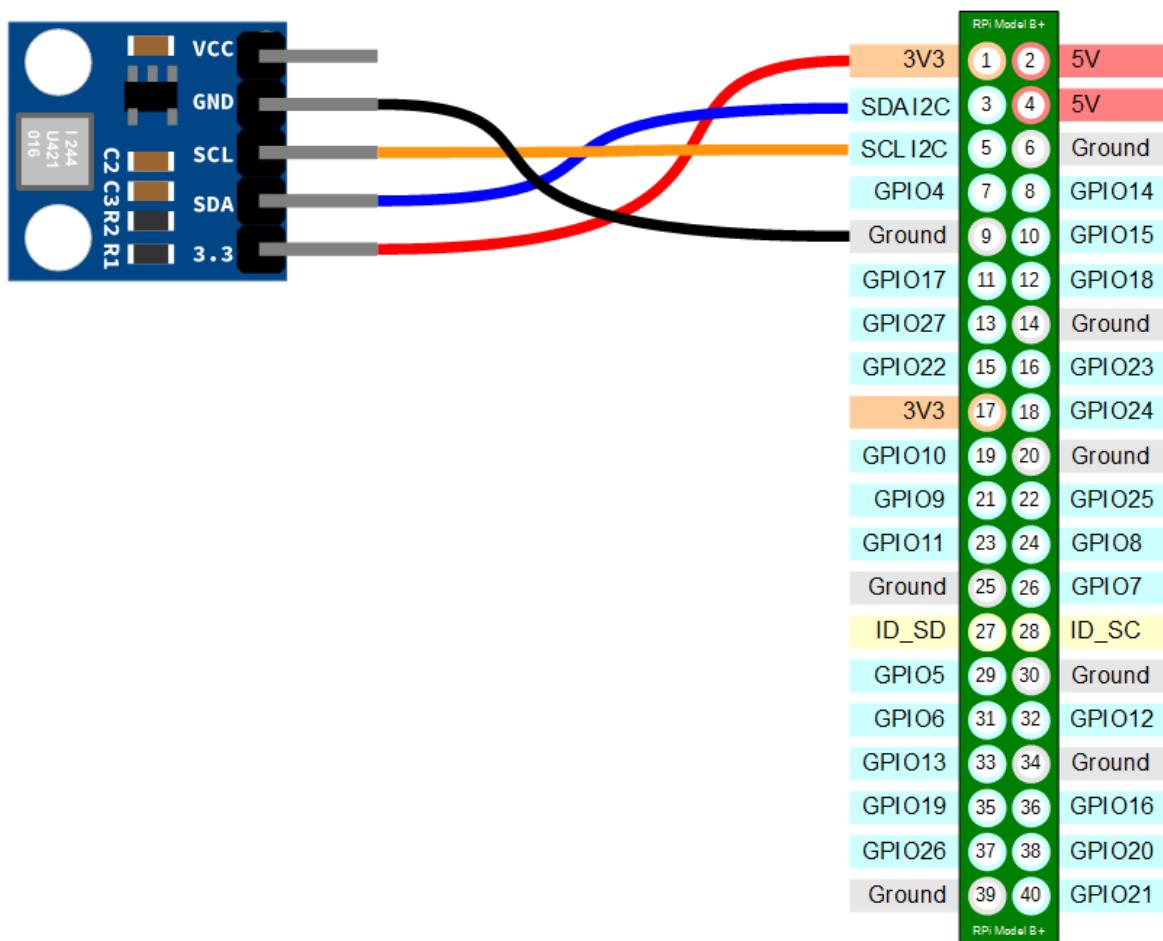
- i** There are a wide range of board configurations that support the BMP180. This just happens to be the one I used. For alternatives see the [hardware required section](#).

## Connect

The BMP180 sensor board should be connected with ground pin to a ground connector, the 3.3 pin (or the VCC pin) to a 3.3V connector, the SCL pin to the SCL I2C connector on pin 5 and the SDA pin to the SDA I2C connector on pin 3 on the Raspberry Pi's connector block. In the connection diagram below the ground is connected to pin 9 and the 3.3V is connected to pin 1.

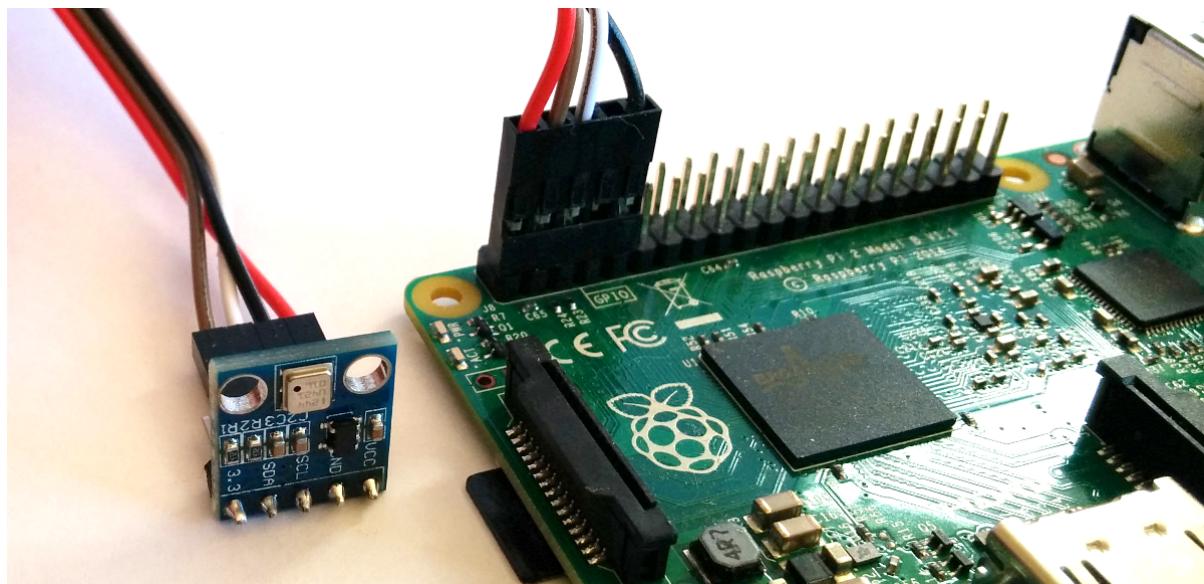
This board will support a connection to the 'VCC' connector of 5V, **but this is not advisable for the Raspberry Pi** as the resulting signal levels on the SDA connector may be higher than desired for the Pi's input. This connection can be safely used with an Arduino board.

The following diagram is a simplified view of the connection.



BMP180 Sensor Board Connection

Connecting the sensor practically can be achieved in a number of ways. You could use a Pi Cobbler break out connector mounted on a bread board connected to the appropriate pins. But because the connection is relatively simple we could build a minimal configuration that will plug directly onto the pins using header connectors and jumper wire. The image below shows how simple this can be.



Physical Connection of BMP180 based Sensor

## Test

Since the BMP180 uses the I2C protocol to communicate, we need to load the appropriate kernel support modules onto the Raspberry Pi to allow this to happen.

Firstly make sure that our software is up to date

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

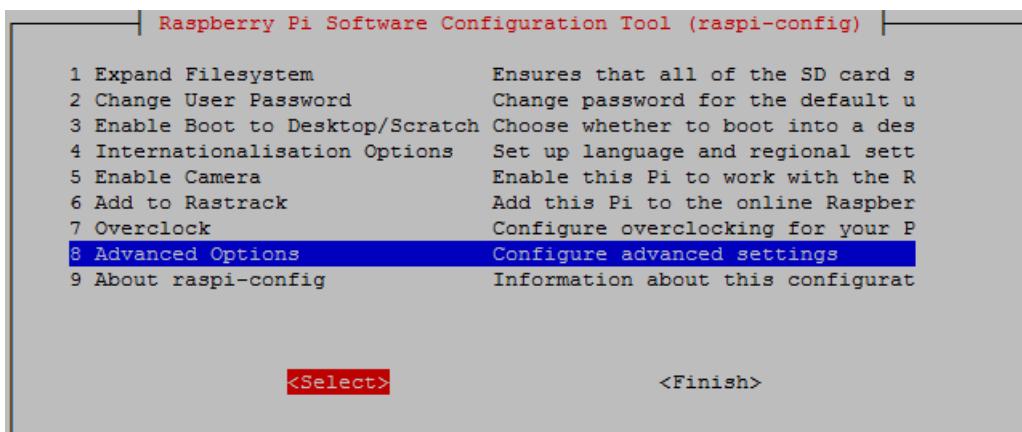
Since we are using the Raspbian distribution there is a simple method to start the process of configuring the Pi to use the I2C protocol.

We can start by running the command;

```
sudo raspi-config
```

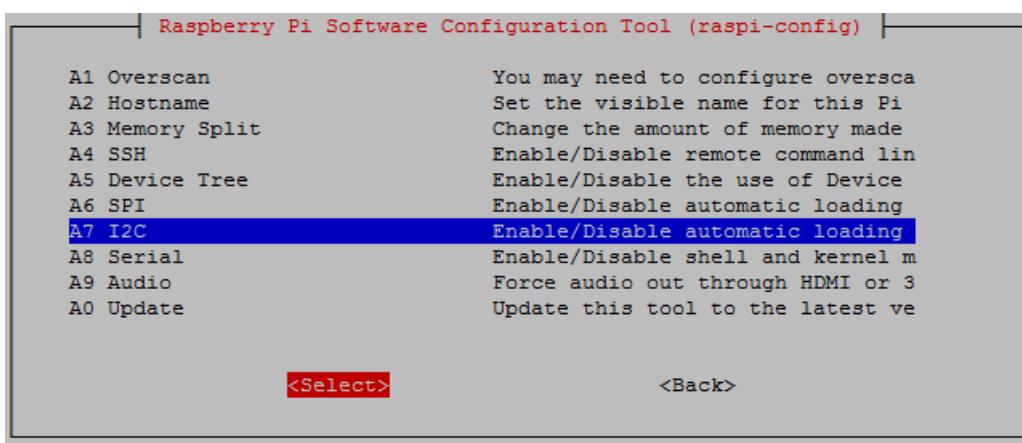
This will start the Raspberry Pi Software Configuration Tool.

On the first page select the Advanced Options with the space bar and then tab to select



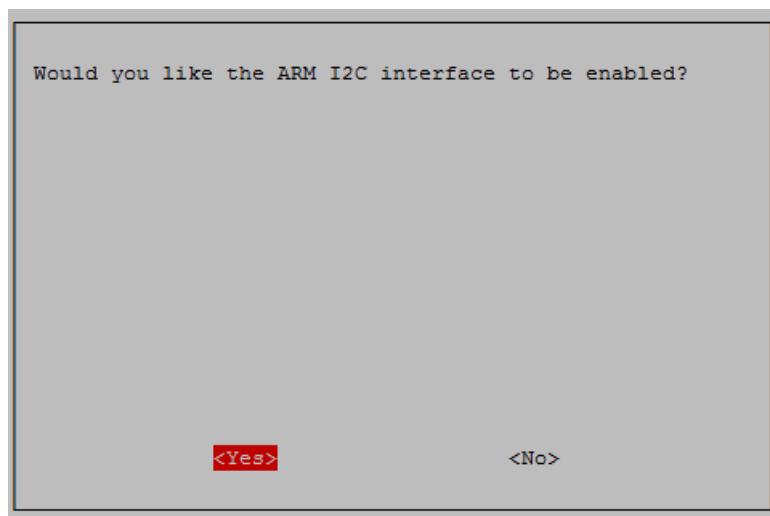
### Advanced Options

Then we select the I2C option for automatic loading of the kernel module;



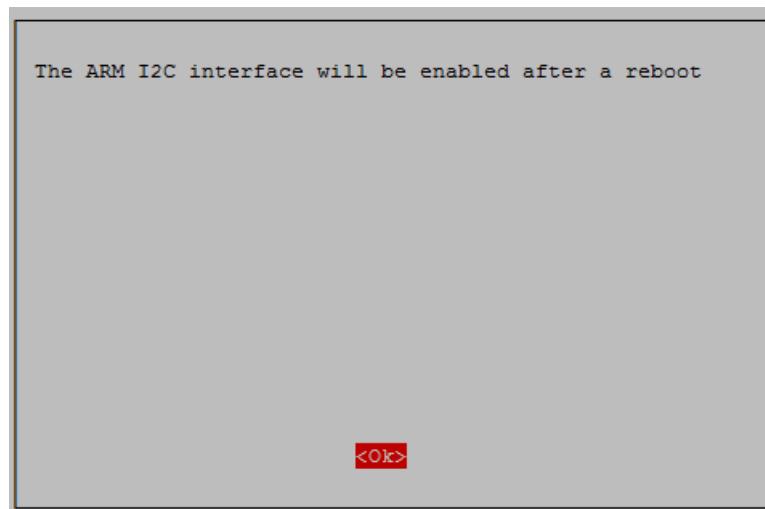
### Automatic Loading

Would we like the ARM I2C interface to be enabled? Yes we would;

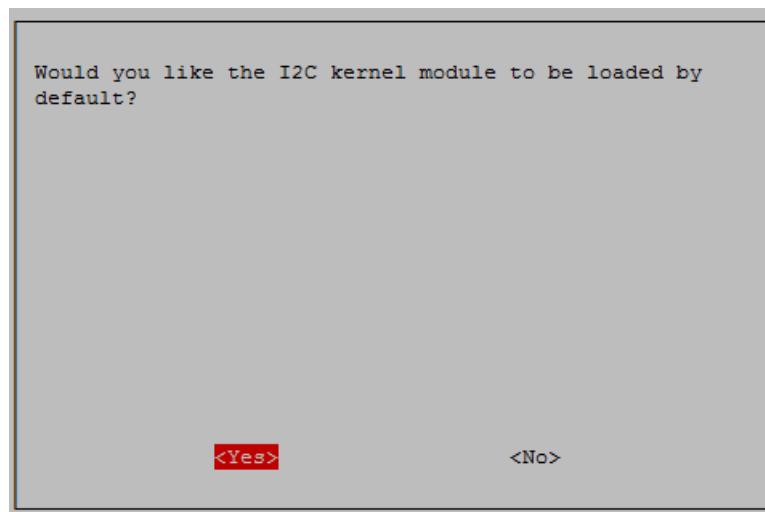


### Enable ARM I2C Interface

We are helpfully informed that the I2C interface will be enabled after the next reboot.

**Enabled After Reboot**

Would we like the I2C kernel module to be loaded by default? Yes we would;

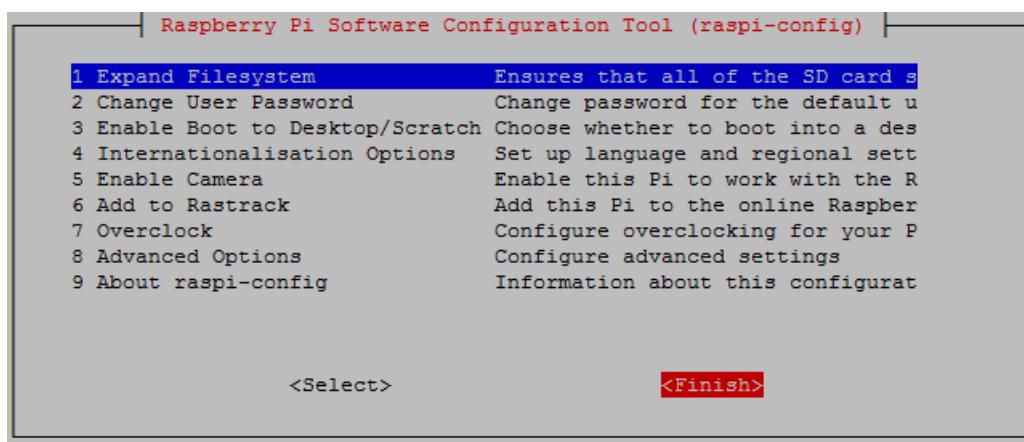
**I2C Kernel Module Loaded by Default**

We are helpfully informed that the I2C kernel module will be loaded by default after the next reboot.



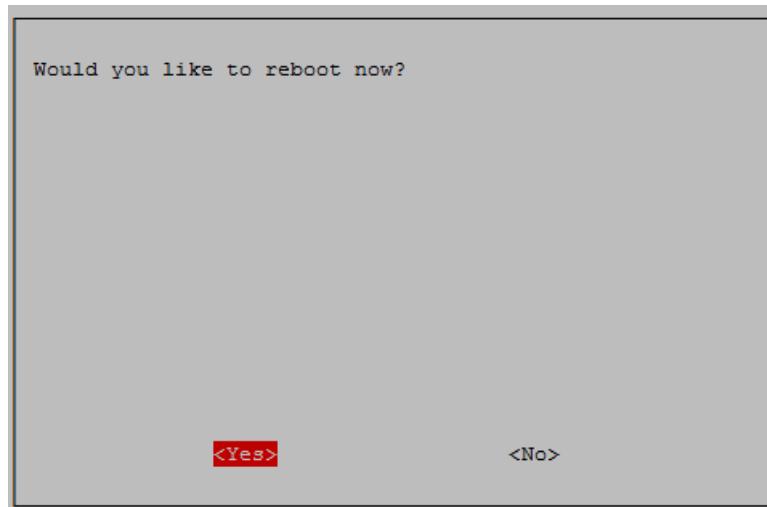
Loaded by Default

Press tab to select 'Finish'.



We're Finished

And yes, we would like to reboot.



Reboot

There's still some work to do to get things sorted. We need to edit the `/etc/modules` file using:

```
sudo nano /etc/modules
```

Where we need to add the following two lines to the end of the `/etc/modules` file:

```
i2c-bcm2708  
i2c-dev
```

Under some circumstances (depending on the kernel version we are using) we would also need to update the `/boot/config.txt` file. We can do this using;

```
sudo nano /boot/config.txt
```

Make sure that the following lines are in the file;

```
dtparam=i2c1=on  
dtparam=i2c_arm=on
```

The we should load tools for working with I2C devices using the following command;

```
sudo apt-get install i2c-tools
```

... and now we need to reboot to load the config.txt file from earlier

```
sudo reboot
```

We can now check to see if our sensor is working using;

```
sudo i2cdetect -y 1
```



If we were using an older B model of Raspberry Pi with 256MB of RAM, we would need to use `sudo i2cdetect -y 0`.

The output should look something like;

```
pi@raspberrypi ~ $ sudo i2cdetect -y 1
 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          - - - - - - - - - - - - - - - - - - - -
10:          - - - - - - - - - - - - - - - - - - - -
20:          - - - - - - - - - - - - - - - - - - - -
30:          - - - - - - - - - - - - - - - - - - - -
40:          - - - - - - - - - - - - - - - - - - - -
50:          - - - - - - - - - - - - - - - - - - - -
60:          - - - - - - - - - - - - - - - - - - - -
70:          - - - - - - - - - - - - - - - - 77
```

This shows us that we have detected our BMP180 on channel '77'.

Now we want to install Python libraries designed to read the values from the BMP180. The library we are going to use was designed specifically to work with the [Adafruit BMP085/BMP180 pressure sensors<sup>68</sup>](#). In carrying out this library development, Adafruit have invested a not inconsiderable amount of time and resources. In return please consider supporting Adafruit and open-source hardware by purchasing products from [Adafruit<sup>69</sup>](#)!

```
sudo apt-get install git build-essential python-dev python-smbus
```

Now we create a directory that we will use to download the Adafruit Python library (assuming that we're starting from our home directory);

<sup>68</sup><https://www.adafruit.com/products/1603>

<sup>69</sup><https://www.adafruit.com/>

```
mkdir bmp180  
cd bmp180
```

Now we will download the library into the `bmp180` directory (the following command retrieves the library from the github site);

```
git clone https://github.com/adafruit/Adafruit_Python_BMP.git
```

Now that we've downloaded it, we need to compile it so that it can be used;



A compiler is a program that translates human readable source code into computer executable machine code. The library that we receive from Github is in source code and in order for it to work correctly as an executable file on the hardware that the Raspberry Pi uses, it needs to be compiled into a form that will suit the Pi.

```
cd Adafruit_Python_BMP  
sudo python setup.py install
```

Once compilation has completed, we can test that everything has gone according to plan by running the test Python script that is installed from Github;

```
python /home/pi/bmp180/Adafruit_Python_BMP/examples/simpletest.py
```

This will hopefully produce an output similar to the following;

```
Temp = 22.10 *C  
Pressure = 101047.00 Pa  
Altitude = 23.42 m  
Sealevel Pressure = 101043.00 Pa
```

The readings for the altitude and sea level pressure are both likely to be inaccurate. That's because they are values that are calculated by the python library and not measured by the device. When the unit calculates the figures, the altitude reading requires an accurate figure for the current pressure at sea level at our location and the sea level pressure requires our altitude. Since in this simple example we never supplied either of these, the answers can hardly be expected to be accurate. Both of the currently returned figures are the results of the Adafruit\_Python\_BMP

library using default values of 101325 Pa for sea level pressure when calculating altitude and 0 m for altitude when calculating sea level pressure.

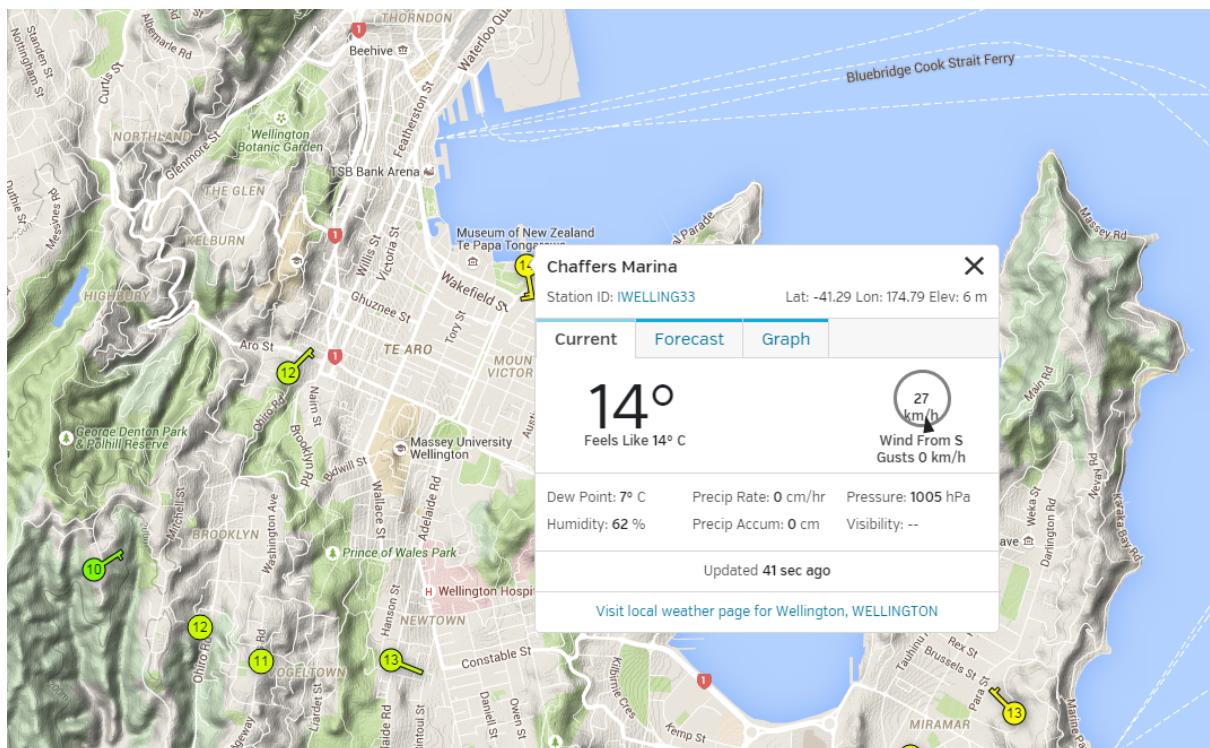
To use values of altitude and / or sea level pressure we will want to edit the `simpletest.py` program that we have just run and add them in there. If we open up the file using nano we see the following towards the end of the file(or at least it will look similar to the following, I have made a small formatting change to the final line so that it doesn't get mangled by exceeding the line width for the page);

```
print 'Temp = {0:0.2f} *C'.format(sensor.read_temperature())
print 'Pressure = {0:0.2f} mb'.format(sensor.read_pressure())
print 'Altitude = {0:0.2f} m'.format(sensor.read_altitude())
print 'Sealevel Pressure = {0:0.2f} Pa' \
    .format(sensor.read_sealevel_pressure())
```

At this point we need to know what our altitude is (in metres) and the pressure at our current location corrected to represent what it would be at seal level.

The altitude should be easy if you have a smart phone with a gps you should have a facility to use it to report it's altitude. Let's imagine that we have an altitude of 71m.

For the pressure we could use the map from the [Weather Underground<sup>70</sup>](#). Using the map, find the weather station closest to your location and click on it to discover what should be the corrected sea level pressure for your location.



Finding Pressure (Courtesy Weather Underground, [wunderground.com](http://wunderground.com/wundermap/))

<sup>70</sup><http://www.wunderground.com/wundermap/>

In the image above the pressure is listed as 1005hPa which is 100500Pa



The pressure on weather sites is commonly given as the sea level pressure to provide a common reference point for pressure readings. If this was not done, the pressure readings would be very difficult to interpret as each weather station would have a pressure that was dependent on the height about the mean sea level *and* the change in pressure due to atmospheric changes. An excellent overview of the reasoning for this is given by the [Eastern Illinois University](#)<sup>71</sup>.

Armed with the altitude and local sea level pressure we can enter them into the functions for `read_altitude` and `read_sealevel_pressure` as below;

```
print 'Temp = {0:0.2f} *C'.format(sensor.read_temperature())
print 'Pressure = {0:0.2f} mb'.format(sensor.read_pressure())
print 'Altitude = {0:0.2f} m'.format(sensor.read_altitude(100500))
print 'Sealevel Pressure = {0:0.2f} Pa' \
    .format(sensor.read_sealevel_pressure(71))
```

The next time we run the `simpletest.py` script we will get accurate readings for altitude and sea level pressure. Be aware however, that these readings will only be accurate so long as the sensor does not change in altitude (don't move it too far) or the pressure doesn't change (and it's always changing). So they're useful functions (and we'll use the seal level function in our [Explore](#) section), but we have to use them correctly.

---

<sup>71</sup>[http://www.ux1.eiu.edu/~cfjps/1400/pressure\\_wind.html](http://www.ux1.eiu.edu/~cfjps/1400/pressure_wind.html)

## Record

To record this data we will use a Python program that connects to our sensor, reads the values of temperature and sea level pressure and writes those values into our MySQL database. At the same time a time stamp will be added automatically.

This code will record the values when executed, but we will need to edit the crontab to allow the values to be recorded continuously (at a set interval).

## Database preparation

First we will set up our database table that will store our data.

Using the phpMyAdmin web interface that we set up, log on using the administrator (root) account and select the ‘measurements’ database that we created as part of the initial set-up.

The screenshot shows a 'Create table on database measurements' dialog box. It has two input fields: 'Name:' containing 'bmp180' and 'Number of columns:' containing '3'. A 'Go' button is located at the bottom right of the dialog.

Create the MySQL Table

Enter in the name of the table and the number of columns that we are going to use for our measured values. In the screenshot above we can see that the name of the table is ‘bmp180’ and the number of columns is ‘3’.

We will use three columns so that we can store the temperature, pressure (the corrected sea level pressure) and the time it was recorded (‘dtg’).

Once we click on ‘Go’ we are presented with a list of options to configure our table’s columns. Don’t be intimidated by the number of options that are presented, we are going to keep the process as simple as practical.

For the first column we can enter the name of the ‘Column’ as ‘dtg’ (short for date time group) the ‘Type’ as ‘TIMESTAMP’ and the ‘Default’ value as ‘CURRENT\_TIMESTAMP’. For the second column we will enter the name ‘temperature’ and the type is ‘FLOAT’. For the third column we will enter the name ‘pressure’ and the type is also ‘FLOAT’.

**Create Table**

Table name:

**Structure**

Column	dtg	temperature	pressure
Type	TIMESTAMP	FLOAT	INT
Length/Values <sup>1</sup>			
Default <sup>2</sup>	CURRENT_TIMESTAMP	None	None

Configure the MySQL Table Columns

Scroll down a little and click on the ‘Save’ button and we’re done.

**Save the MySQL Table Columns**

**Save** Or Add  column(s) **Go**



## Why did we choose those particular settings for our table?

Our ‘dtg’ column needs to store a value of time that includes the date and the time, so the advantage of selecting TIMESTAMP in this case is that we can select the default value to be the current time. This means that when we write our data to the table we only need to write the ‘temperature’ and ‘pressure’ and the ‘dtg’ will be entered automatically for us. The disadvantage of using ‘TIMESTAMP’ is that it has a more limited range than DATETIME. TIMESTAMP can only have a range between ‘1970-01-01 00:00:01’ to ‘2038-01-19 03:14:07’.

Our temperature readings are generated (by the Python library) as a value with decimal places. As a result we need to use a numerical format that supports numbers with decimal places. There are a range of options for defining the ranges for decimal numbers, but FLOAT allows us to ignore the options (at the expense of efficiency) and rely on our recorded values being somewhere between -3.402823466E+38 and 3.402823466E+38 (if our temperature falls outside those extremes we are in trouble).

Our pressure readings are generated (by the Python library) as an integer value. As a result we need to use a numerical format that supports numbers as integers. There are a range of options for defining the ranges for integers and INT allows us to use a format that can record values between -2,147,483,648 and 2,147,483,647 (again, if our pressure readings fall outside those extremes we are in trouble).

## Record the readings

The following Python code is a script which allows us to check the state of our BMP180 sensor, return values for temperature and sea level pressure (based on a height of our sensor of 71m)

and write those values to our database.

The full code can be found in the code samples bundled with this book (bmp180.py).

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import MySQLdb as mdb
import logging
import Adafruit_BMP.BMP085 as BMP085

# Setup logging
logging.basicConfig(filename='/home/pi/bmp180_error.log',
    format='%(asctime)s %(levelname)s %(name)s %(message)s')
logger=logging.getLogger(__name__)

# Function for storing readings into MySQL
def insertDB(temperature,pressure):

    try:

        con = mdb.connect('localhost',
                          'pi_insert',
                          'xxxxxxxxxx',
                          'measurements');

        cursor = con.cursor()

        sql = "INSERT INTO bmp180(temperature, pressure) \
VALUES ('%s', '%s')" % \
( temperature, pressure)
        cursor.execute(sql)
        sql = []
        con.commit()

        con.close()

    except mdb.Error, e:
        logger.error(e)

# Get readings from sensor and store them in MySQL
sensor = BMP085.BMP085()

temperature = sensor.read_temperature()
pressure = sensor.read_sealevel_pressure(71)

insertDB(temperature,pressure)
```

This script can be saved in our home directory (/home/pi) as bmp180.py and can be run by typing;

```
python bmp180.py
```

Once the command is run we should be able to check our MySQL database and see an entry for the times that the sensor was checked along with the corresponding temperature and pressure readings.

	dtg	temperature	pressure
<input type="checkbox"/> Copy Delete	2015-06-15 20:50:01	17.6	101704
<input type="checkbox"/> Copy Delete	2015-06-15 20:40:01	17.5	101705
<input type="checkbox"/> Copy Delete	2015-06-15 20:30:01	17.2	101697
<input type="checkbox"/> Copy Delete	2015-06-15 20:20:02	17.2	101696
<input type="checkbox"/> Copy Delete	2015-06-15 20:10:01	16.9	101691
<input type="checkbox"/> Copy Delete	2015-06-15 20:00:02	17.6	101675

Save the MySQL Table Columns

## Code Explanation

The script starts by importing the modules that it's going to use for the process of reading and recording the measurements;

```
import MySQLdb as mdb
import logging
import Adafruit_BMP.BMP085 as BMP085
```



Python code in one module gains access to the code in another module by the process of importing it. The `import` statement invokes the process and combines two operations; it searches for the named module, then it binds the results of that search to a name in the local scope.

Then the code sets up the `logging module`<sup>72</sup> so that any failures in writing information to the database are written to the file /home/pi/bmp180\_error.log.

```
logging.basicConfig(filename='/home/pi/bmp180_error.log',
                    format='%(asctime)s %(levelname)s %(name)s %(message)s')
logger=logging.getLogger(__name__)
```

In the following function where we are getting our readings or writing to the database we write to the log file if there is an error. Otherwise it will accept values of temperature and pressure and write them to our database.

<sup>72</sup><http://pymotw.com/2/logging/>

```

def insertDB(temperature,pressure):

    try:

        con = mdb.connect('localhost',
                          'pi_insert',
                          'xxxxxxxxxx',
                          'measurements');

        cursor = con.cursor()

        sql = "INSERT INTO bmp180(temperature, pressure) \
VALUES ('%s', '%s')" % \
( temperature, pressure)
        cursor.execute(sql)
        sql = []
        con.commit()

        con.close()

    except mdb.Error, e:
        logger.error(e)

```

Which brings us to the main part of our code;

```

sensor = BMP085.BMP085()

temperature = sensor.read_temperature()
pressure = sensor.read_sealevel_pressure(71)

insertDB(temperature,pressure)

```

Here we declare our sensor (`sensor = BMP085.BMP085()`) then read our temperature value (`sensor.read_temperature()`) and pressure (`sensor.read_sealevel_pressure(71)`). Note here that we are including the '71' as an argument for the altitude when reading the sea level pressure and therefore recording it calibrated for the height of the sensor.

Finally we call the function to insert the values into the database (`insertDB(temperature,pressure)`). It's a fairly simple block of code that is made simple by virtue of the work that has been put into the associated libraries like Adafruit\_BMP.BMP085.

## Recording data on a regular basis with cron

While our code is a thing of simple elegance, it only records each time it is run.

What we need to implement is a schedule so that at a regular time, the program is run. This is achieved using cron via the crontab. While we will cover the requirements for this project here, you can read more about the [crontab](#) in the [Glossary](#).

To set up our schedule we need to edit the crontab file. This is done using the following command;

```
crontab -e
```

Once run it will open the crontab in the nano editor. We want to add in an entry at the end of the file that looks like the following;

```
*/10 * * * * /usr/bin/python /home/pi/bmp180.py
```

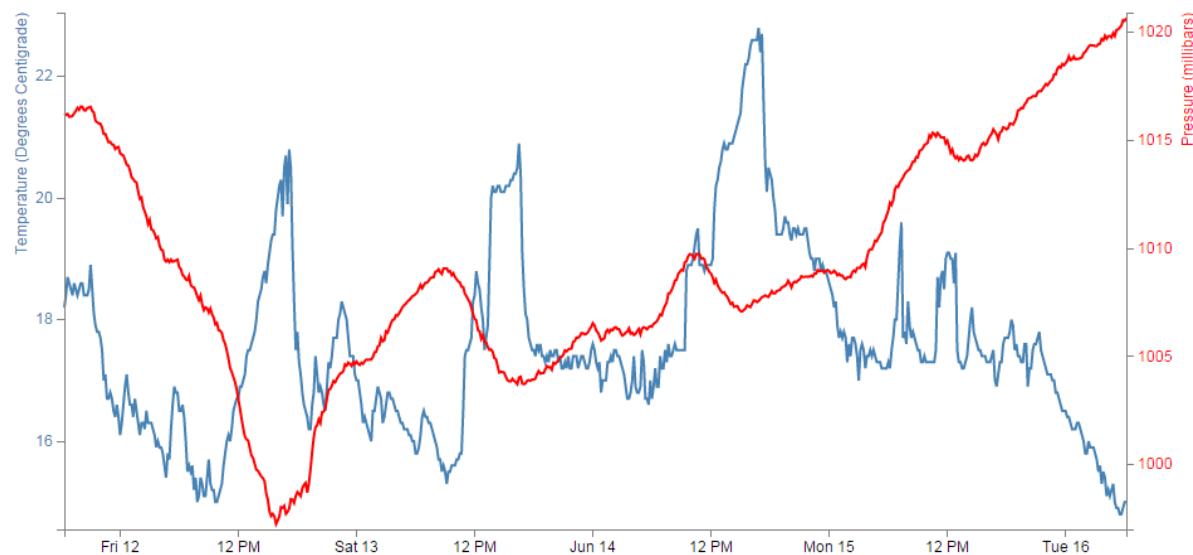
This instructs the computer that every 10 minutes we will run the command `/usr/bin/python /home/pi/bmp180.py` (which if we were at the command line in the pi home directory we would run as `python bmp180.py`, but since we can't guarantee where the script will be running from, we are supplying the full path to the `python` command and the `bmp180.py` script).

Save the file and the computer will start running the program on its designated schedule and we will have temperature and pressure entries written to our database every 10 minutes.

Job done! We're measuring and recording temperature and pressure!

## Explore

To explore our temperature and pressure data we will use a web based graph that shows both sets of data as lines superimposed on the same graph with separate Y axes on either side to provide value references.



The Dual Line Temperature and Pressure Graph

The graph will be drawn using the [d3.js<sup>73</sup>](#) JavaScript library and the data will be retrieved via a PHP script that queries our MySQL database.

## The Code

The following code is a PHP file that we can place on our Raspberry Pi's web server (in the `/var/www` directory) that will allow us to view all of the results that have been recorded in the temperature directory on a graph;



There are many sections of the code which have been explained already in the set-up section of the book that describes a simple line graph for a single temperature measurement. Where these occur we will be less thorough with the explanation of how the code works.

The full code can be found in the code samples bundled with this book (`bmp180.php`).

---

<sup>73</sup><http://d3js.org/>

```
<?php

$hostname = 'localhost';
$username = 'pi_select';
$password = 'xxxxxxxxxx';

try {
    $dbh = new PDO("mysql:host=$hostname;dbname=measurements",
                   $username, $password);

    /*** The SQL SELECT statement ***/
    $sth = $dbh->prepare(
        "SELECT `dtg` AS date,
        `temperature` AS temperature,
        `pressure` AS pressure
        FROM `bmp180`
        ORDER BY date DESC
        LIMIT 0,900
    ");
    $sth->execute();

    /* Fetch all of the remaining rows in the result set */
    $result = $sth->fetchAll(PDO::FETCH_ASSOC);

    /*** close the database connection ***/
    $dbh = null;
}

catch(PDOException $e)
{
    echo $e->getMessage();
}

$json_data = json_encode($result);

?>

<!DOCTYPE html>
<meta charset="utf-8">
<style>

body { font: 12px Arial; }

path {
    stroke-width: 2;
    fill: none;
}
```

```
.axis path,
.axis line {
  fill: none;
  stroke: grey;
  stroke-width: 1;
  shape-rendering: crispEdges;
}

</style>
<body>
<script src="http://d3js.org/d3.v3.min.js"></script>

<script>

var margin = {top: 30, right: 55, bottom: 30, left: 60},
  width = 960 - margin.left - margin.right,
  height = 470 - margin.top - margin.bottom;

// Parse the date / time
var parseDate = d3.time.format("%Y-%m-%d %H:%M:%S").parse;

// specify the scales for each set of data
var x = d3.time.scale().range([0, width]);
var y0 = d3.scale.linear().range([height, 0]);
var y1 = d3.scale.linear().range([height, 0]);

// axis formatting
var xAxis = d3.svg.axis().scale(x)
  .orient("bottom");
var yAxisLeft = d3.svg.axis().scale(y0)
  .orient("left").ticks(5);
var yAxisRight = d3.svg.axis().scale(y1)
  .orient("right").ticks(5).tickFormat(d3.format(".0f"));

// line functions
var temperatureLine = d3.svg.line()
  .x(function(d) { return x(d.date); })
  .y(function(d) { return y0(d.temperature); });
var pressureLine = d3.svg.line()
  .x(function(d) { return x(d.date); })
  .y(function(d) { return y1(d.pressure); });

// seetup the svg area
var svg = d3.select("body")
  .append("svg")
  .attr("width", width + margin.left + margin.right)
```

```

        .attr("height", height + margin.top + margin.bottom)
.append("g")
    .attr("transform",
          "translate(" + margin.left + ", " + margin.top + ")");
}

// Get the data
<?php echo "data=".json_data." ?>

// wrangle the data into the correct formats and units
data.forEach(function(d) {
    d.date = parseDate(d.date);
    d.temperature = +d.temperature;
    d.pressure = +d.pressure/100;
});

// Scale the range of the data
x.domain(d3.extent(data, function(d) { return d.date; }));
y0.domain([
    d3.min(data, function(d) {return Math.min(d.temperature); })-.25,
    d3.max(data, function(d) {return Math.max(d.temperature); })+.25]);
y1.domain([
    d3.min(data, function(d) {return Math.min(d.pressure); })-.25,
    d3.max(data, function(d) {return Math.max(d.pressure); })+.25]);

svg.append("path")      // Add the temperature line.
    .style("stroke", "steelblue")
    .attr("d", temperatureLine(data));

svg.append("path")      // Add the pressure line.
    .style("stroke", "red")
    .attr("d", pressureLine(data));

svg.append("g")         // Add the X Axis
    .attr("class", "x axis")
    .attr("transform", "translate(0, " + height + ")")
    .call(xAxis);

svg.append("g")         // Add the temperature axis
    .attr("class", "y axis")
    .style("fill", "steelblue")
    .call(yAxisLeft);

svg.append("g")         // Add the pressure axis
    .attr("class", "y axis")
    .attr("transform", "translate(" + width + ", 0)")
    .style("fill", "red")

```

```

    .call(yAxisRight);

svg.append("text")      // Add the text label for the temperature axis
    .attr("transform", "rotate(-90)")
    .attr("x", 0)
    .attr("y", -30)
    .style("fill", "steelblue")
    .style("text-anchor", "end")
    .text("Temperature (Degrees Centigrade)");

svg.append("text")      // Add the text label for the pressure axis
    .attr("transform", "rotate(-90)")
    .attr("x", 0)
    .attr("y", width + 53)
    .style("fill", "red")
    .style("text-anchor", "end")
    .text("Pressure (millibars)");

</script>
</body>

```

This graph contains some fairly common elements that in a couple of cases are just arranged slightly differently to what we have explored in the single temperature measurement project.

The code will automatically try to collect 900 data points with a minimum time between each of 10 minutes. Depending on your requirements (any your recording times) you may want to vary the query.

## PHP

The PHP block at the start of the code is mostly the same as our example code for our single temperature measurement project. The significant difference however is in the select statement.

```

SELECT `dtg` AS date,
`temperature` AS temperature,
`pressure` AS pressure
FROM `bmp180`
ORDER BY date DESC
LIMIT 0,900

```

The query collects three columns of values. The Date Time Group (date), the temperature (temperature) and the pressure (pressure). There are a range of different ways that the values could be gathered and this is a very simple mechanism that grabs the latest 900 entries irrespective of the time that they were entered. In theory (so long as our `bmp180.py` script is working correctly) we are recording a set of readings every 10 minutes which will make for a graph that will span just over 6 days, so feel free to play about with the query to find a set of returned values that suit the data that you're wanting to portray.

## CSS (Styles)

There are a range of styles that are applied to the elements of the graphic.

```
body { font: 12px Arial; }

path {
  stroke-width: 2;
  fill: none;
}

.axis path,
.axis line {
  fill: none;
  stroke: grey;
  stroke-width: 1;
  shape-rendering: crispEdges;
}
```

We set a default text font and size, the width and fill state for our graph lines and some formatting for our axes.

## JavaScript

The code has very similar elements to our single temperature measurement script and comparing both will show us that we are doing similar things in each graph.

We set up the margins and declare the `parseDate` function in much the same way as the single temperature script.

Then the first significant change occurs that will permeate through the code. Where previously we have dealt with a single Y axis, now we have to declare and differentiate between two separate Y axes. The first place that this occurs is where we scale our domains and set our ranges. This will now look like the following;

```
var x = d3.time.scale().range([0, width]);
var y0 = d3.scale.linear().range([height, 0]);
var y1 = d3.scale.linear().range([height, 0]);
```

Here `y0` will serve as the scale for the temperature (on the left of the graph) and `y1` will serve for pressure on the right.

Then when we declare our axes formatting functions we have to do the same thing;

```
var xAxis = d3.svg.axis().scale(x)
    .orient("bottom");
var yAxisLeft = d3.svg.axis().scale(y0)
    .orient("left").ticks(5);
var yAxisRight = d3.svg.axis().scale(y1)
    .orient("right").ticks(5).tickFormat(d3.format(".0f"));
```

yAxisLeft uses y0 and yAxisRight uses y1. We should also note that there is a slight addition to the formatting of the right hand axis. Because the values for pressure will be over 1000 d3 will add in a comma separator for every factor of 1000. In this case I find that the comma can be a bit distracting (the largest value should in theory be only just over 1000), so the d3.format(".0f") fixes the shown values to have a fixed precision of 0 numbers after the decimal place (.0f) and by omitting the comma from the formatting statement it will not be used. For more details of formatting options in general check out the excellent page [here in the D3 Wiki](#)<sup>74</sup>.

The other point of difference is that yAxisRight has the ticks orientated on the right and yAxisLeft has the ticks on the left.

Because we have two lines we need to declare two different functions for them;

```
var temperatureLine = d3.svg.line()
    .x(function(d) { return x(d.date); })
    .y(function(d) { return y0(d.temperature); });
var pressureLine = d3.svg.line()
    .x(function(d) { return x(d.date); })
    .y(function(d) { return y1(d.pressure); });
```

The SVG area is set up in the same way and then we get our data with a PHP call;

```
<?php echo "data=".json_data.";" ?>
```

Then we cycle through our data using a forEach statement;

```
data.forEach(function(d) {
    d.date = parseDate(d.date);
    d.temperature = +d.temperature;
    d.pressure = +d.pressure/100;
});
```

In this loop we parse our date/time values so that the code knows how to use them appropriately (as time units rather than numeric values or plain text strings). Then we then do a little bit of maths to ensure that our temperature values are recognised as numbers and we divide our pressure readings (which are in pascals) by 100 so that they can be represented as millibars (since this is the more common format that people recognise and 100Pa = 1mb).

The domains for all the datasets are then established and again we need to include both Y axes;

---

<sup>74</sup>[https://github.com/mbostock/d3/wiki/Formatting#d3\\_format](https://github.com/mbostock/d3/wiki/Formatting#d3_format)

```

x.domain(d3.extent(data, function(d) { return d.date; }));
y0.domain([
  d3.min(data, function(d) {return Math.min(d.temperature); })-.25,
  d3.max(data, function(d) {return Math.max(d.temperature); })+.25]);
y1.domain([
  d3.min(data, function(d) {return Math.min(d.pressure); })-.25,
  d3.max(data, function(d) {return Math.max(d.pressure); })+.25]);

```

Here we've set the domain of both sets of Y axis data sets to be between the maximum and minimum of their highest and lowest values and we've tacked on an extra .25 to the top and bottom of the ranges to move the top and bottom of both lines just off the edges of the graph.

We then append both lines in a normal fashion and add the X and temperature axes in a familiar way. However, we do something a little different when we add the pressure axis;

```

svg.append("g")
  .attr("class", "y axis")
  .attr("transform", "translate(" + width + " ,0)")
  .style("fill", "red")
  .call(yAxisRight);

```

We add in a `transform` attribute that moves (translates) the axis to the right hand side of the graph (using the `width` of the graph).

We then also utilise a `transform` attribute for both of the labels that we add adjacent to each Y axis. In this case however we employ the `rotate` operator to pivot the text so that it seems a bit more in keeping with the axis.

```

svg.append("text")
  .attr("transform", "rotate(-90)")
  .attr("x", 0)
  .attr("y", -30)
  .style("fill", "steelblue")
  .style("text-anchor", "end")
  .text("Temperature (Degrees Centigrade)");

svg.append("text")
  .attr("transform", "rotate(-90)")
  .attr("x", 0)
  .attr("y", width + 53)
  .style("fill", "red")
  .style("text-anchor", "end")
  .text("Pressure (millibars)");

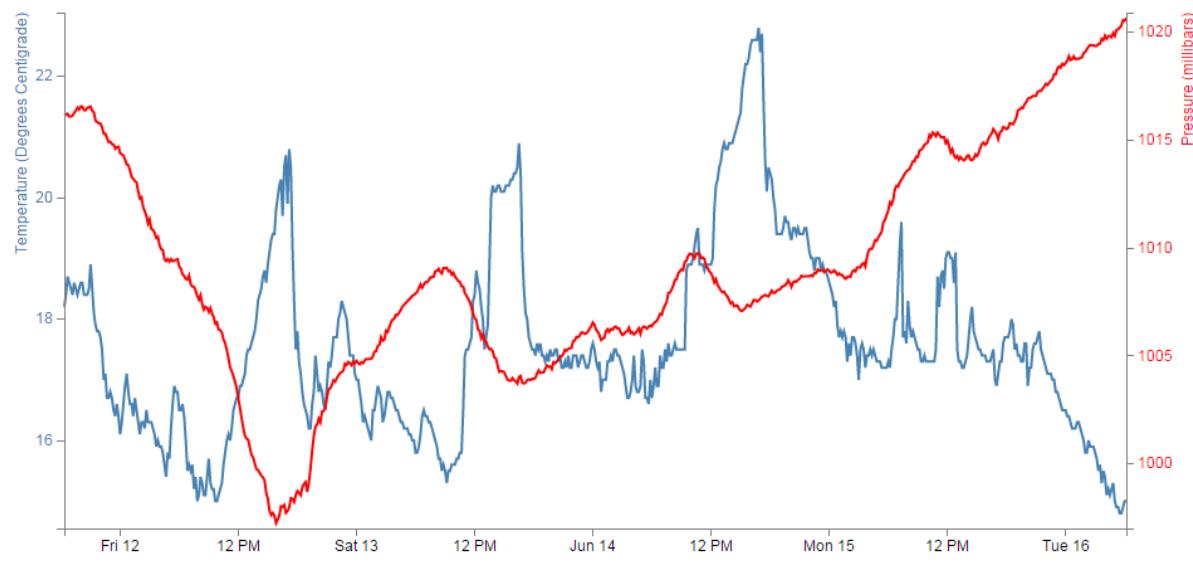
```

We also move the rotated text to an appropriate position using `x` and `y` attributes. For more information on manipulating text elements with D3 check out this section from D3 Tips and Tricks<sup>75</sup>.

---

<sup>75</sup><https://leanpub.com/D3-Tips-and-Tricks/read#leanpub-auto-text>

And there we have it. Our temperature and pressure being read and presented in a stylish dual line graph.



**Dual Line Temperature and Pressure Graph**

If you want a closer explanation for this piece of code, download a copy of D3 Tips and Tricks<sup>76</sup> for this and a whole swag of other information.

## Bibliography

- Using the BMP085/180 with Raspberry Pi or Beaglebone Black (Adafruit Learning System)<sup>77</sup>
- BMP180 Barometric Pressure/Temperature/Altitude Sensor- 5V ready (Adafruit Industries)<sup>78</sup>
- Adafruit Python library for accessing the BMP series pressure and temperature sensors (github)<sup>79</sup>
- BMP180 Barometric Pressure Sensor Hookup (learn.sparkfun.com)<sup>80</sup>
- Sensors - Pressure, Temperature and Altitude with the BMP180 (The Pi Hut)<sup>81</sup>
- I2C - BMP180 Temperature and Pressure Sensor (Eric Friedrich, github.com/raspberrypi-aa)<sup>82</sup>
- BMP180 Digital pressure sensor data sheet (Bosch)<sup>83</sup>
- Air pressure and wind (Eastern Illinois University)<sup>84</sup>
- Hypsometric equation (Wikipedia)<sup>85</sup>

<sup>76</sup><https://leanpub.com/D3-Tips-and-Tricks>

<sup>77</sup><https://learn.adafruit.com/using-the-bmp085-with-raspberry-pi?view=all>

<sup>78</sup><http://www.adafruit.com/product/1603>

<sup>79</sup>[https://github.com/adafruit/Adafruit\\_Python\\_BMP](https://github.com/adafruit/Adafruit_Python_BMP)

<sup>80</sup><https://learn.sparkfun.com/tutorials/bmp180-barometric-pressure-sensor-hookup->

<sup>81</sup><http://thepihut.com/blogs/raspberry-pi-tutorials/18025084-sensors-pressure-temperature-and-altitude-with-the-bmp180>

<sup>82</sup><http://raspberrypi-aa.github.io/session3/i2c-temp-pressure.html>

<sup>83</sup><http://www.adafruit.com/datasheets/BST-BMP180-DS000-09.pdf>

<sup>84</sup>[http://www.ux1.eiu.edu/~cfjps/1400/pressure\\_wind.html](http://www.ux1.eiu.edu/~cfjps/1400/pressure_wind.html)

<sup>85</sup>[http://en.wikipedia.org/wiki/Hypsometric\\_equation](http://en.wikipedia.org/wiki/Hypsometric_equation)

# Connecting Analog Sensors to the Raspberry Pi

The Raspberry Pi is a marvel of connectivity. It's 40 pin header and associated peripheral ports provide a spectacular range of options to interface with the world outside the Raspberry Pi. However, one feature that the Pi doesn't have built in is the facility to accept an analog input.



## Analog vs. analogue

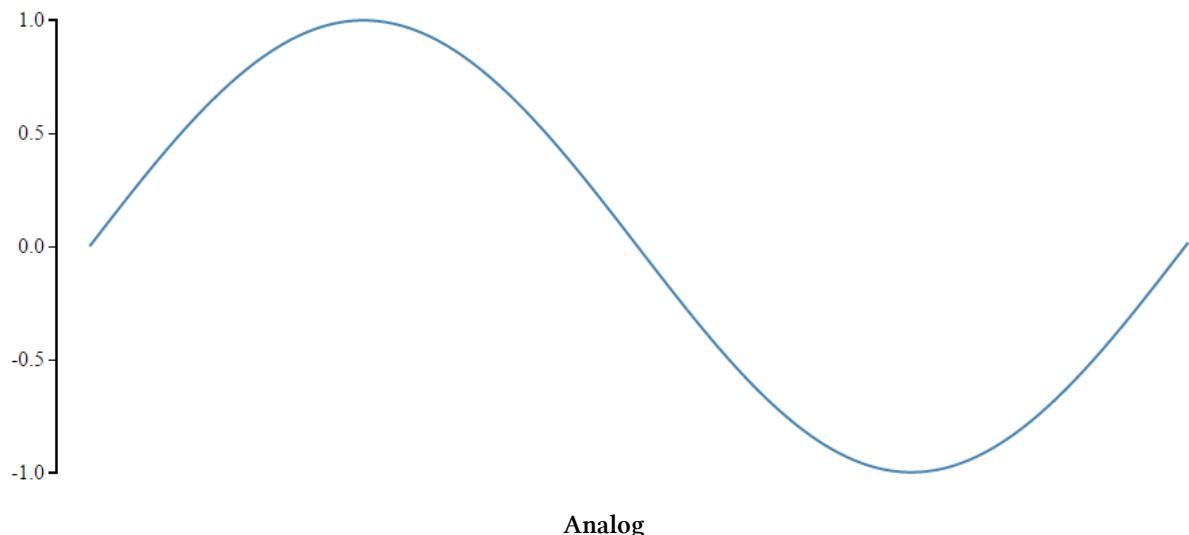
With the word traditionally spelled analogue, American English tends to drop the silent -ue in some contexts, making analog. The spellings are largely interchangeable, though analog is usually used in relation to electronics, while analogue is often used in the sense something that bears analogy to something else. Frankly I'm torn. I'm going with 'analog' in the text for this chapter, but my instincts tell me 'analogue'. I'm sure the Internet has an opinion.

## Analog and Digital

Signals (or even information in general) can be broken down into two different types; Analog and digital.

### Analog

An analog signal is one that has an infinitely variable range of values that can change over time.

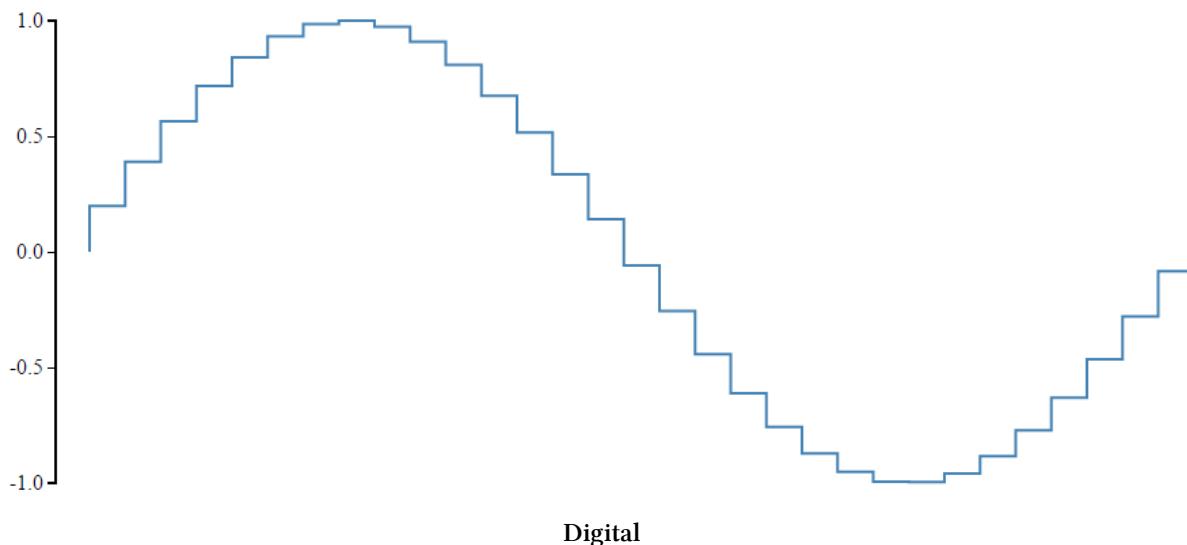


If we consider the question of how much light is shining outside we could imagine that the level of brightnesses varies between the blackness of a moonless night and a overcast sky to a cloudless day with the sun high in the sky.

These are rough approximations of dark and light, but between the two extremes is a range of brightness levels which are always changing. If we wanted to measure how bright it was at any particular time we could set ourselves a numeric range of 0 representing the middle of the night and 100 representing the middle of the day and the number that represented the brightness at any particular time would be somewhere between those two numbers. Typically in electronics an analog signal is a voltage that will be anywhere on that variable range between two limits.

## Digital

A digital signal represents information as discrete values.



For example at it's most fundamental the light level outside could be described as dark or light. Represented numerically this could be dark = 0 and bright = 1. While this is perfectly valid, we would often prefer to have a little more granularity in our measurement and so we can increase the number of discrete steps that represent light levels to match our expectations of the type of information we're interested in. if we add another couple of levels in we could have light that was dark = 0, dim = 0.33, glowing = 0.66 and bright = 1. We can continue to improve the resolution of our numerical perception of the level of light in a process that is called Analog to Digital Conversion or ADC.

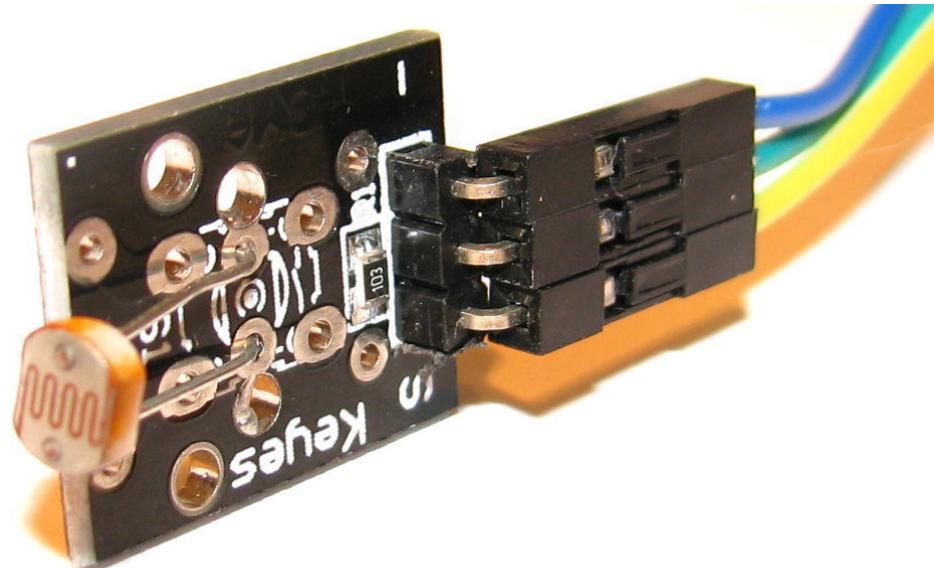
## Analog to Digital Conversion (ADC)

Luckily there are a wide range of options available to convert analog signals into digital ones. Therefore, people wanting to input an analog signal into a Raspberry Pi can simply include a separate ADC into their project and it will work wonderfully. That's what we're going to do here using the [ADS1015 from Adafruit<sup>86</sup>](#). The ADS1015 has a 12bit resolution giving it the ability to convert an analog signal into one of 4096 discrete levels.

<sup>86</sup><http://www.adafruit.com/products/1083>

## The Sensor

While this project is more about the conversion of analog signals into digital ones, this project will use a Keyes KY-018<sup>87</sup> sensor based on a Light Dependent Resistor (LDR) to produce a variable resistance in the presence of different light levels. An applied voltage (from the Pi) returns a variable voltage from the LDR. It is this variable voltage that is then digitised with the ADC.



Keyes KY-018 Analog Light Sensor

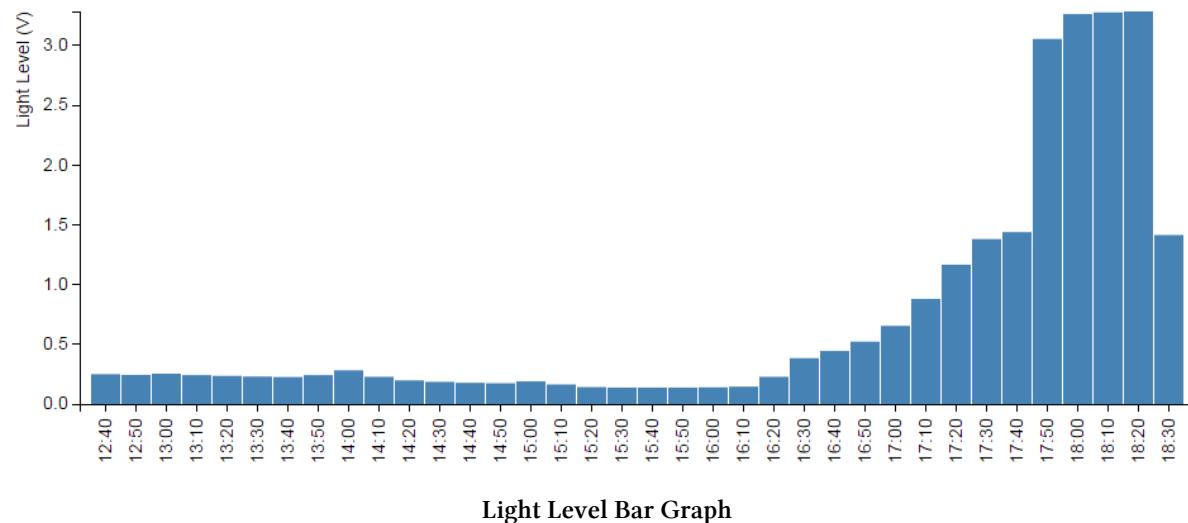
In essence there are a range of different sensors that could be used. I have successfully also connected the Keyes analog hall effect sensor (KY-035, which senses magnetic fields) and there will be others in that range that will work.

---

<sup>87</sup>[https://tkkrlab.nl/wiki/Arduino\\_KY-018\\_Photo\\_resistor\\_module](https://tkkrlab.nl/wiki/Arduino_KY-018_Photo_resistor_module)

## Data Visualization

We will present the data by creating a bar graph showing the light level as measured every 10 minutes for the last 6 hours.



This project has drawn on a range of sources for information. Where not directly referenced in the text, check out the [Bibliography](#) at the end of the chapter.

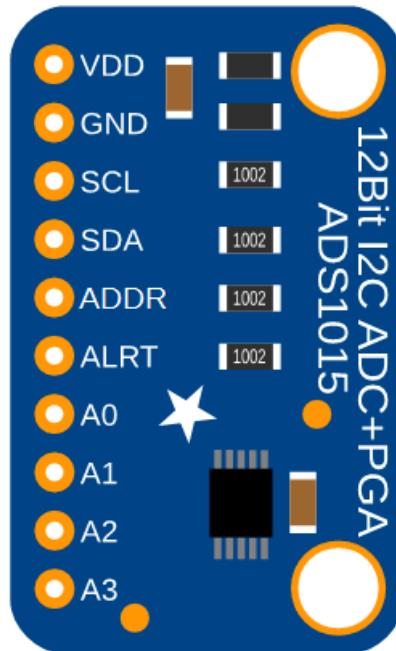
# Measure

## Hardware required

- A Raspberry Pi (huge range of sources)
- ADS1015 Analog to Digital Converter from Adafruit<sup>88</sup>.
- Female to Female Dupont connector cables (Deal Extreme<sup>89</sup> or build your own!)
- Photoresistor module KY-018<sup>90</sup> from Keyes<sup>91</sup>.

## The ADS1015 Analog to Digital Converter

The ADS1015 is actually a component *on* our ADC board. This component is manufactured by integrated circuits manufacturer Texas Instruments. The circuit board that we're using in the project is from Adafruit. It incorporates some interconnection circuitry to make the signals as stable as practical and to provide a convenient physical interface (via header pins). The ADS1015 provides 12-bit (4096 levels) precision at up to 3300 readings per second (the rate is programmable). The board can be configured to accept four sensors of the type we will be using (single-ended), or two differential channels (which use two varying signals instead of a single signal and a ground). There is also a programmable gain amplifier built in with up to x16 gain, to help amplify smaller signals to the full range. The ADC can operate on a voltage range from 2V to 5V (which is applied to the VDD pin).



ADS1015 Sensor Board

<sup>88</sup><http://www.adafruit.com/products/1083>

<sup>89</sup><http://www.dx.com/p/8-pins-female-to-female-dupont-cable-for-raspberry-pi-multicolored-21cm-326450>

<sup>90</sup>[https://tkkrlab.nl/wiki/Arduino\\_KY-018\\_Photo\\_resistor\\_module](https://tkkrlab.nl/wiki/Arduino_KY-018_Photo_resistor_module)

<sup>91</sup><http://en.keyes-robot.com/index.aspx>

If all this sounds a bit electrickery, don't sweat it. The aim here is to provide ourselves with enough information to get us started and if we feel like pressing on and learning more we will :-).

The ADS1015 will send the digital levels to the Pi via the I2C communications protocol. The address that this connection is made on can be changed to one of four options so you can have up to 4 ADS1015's connected for up 16 sensor inputs!



Inter-Integrated Circuit or I2C (pronounced as either I-squared-C or I-2-C) connection is generically referred to as a "two-wire interface". It's commonly used to attach low-speed peripherals to computing devices.

I2C can be used to connect up to 127 nodes via a bus which has two data wires, called SCL and SDA. SCL is the CLock line which is used to synchronize all data transfers over the I2C bus. SDA is the DAta line. The I2C bus works on a 'Master - Slave' system where in this case the master is the Raspberry Pi. Slaves can be integrated circuits such as sensors or micro controllers.

When the master wishes to communicate with a slave it sends a series of pulses down the SDA and SCL lines. The data that is sent includes a unique address that identifies the slave with which the master needs to interact. When data is being sent on the SDA line, clock pulses are sent on the SCL line to keep master and slave synchronised..

## The Light Dependant Resistor (LDR or Photoresistor) Sensor

Our sensor will use an LDR to produce a variable resistance in the presence of different light levels.

In the dark, their resistance is very high, sometimes up to  $1M\Omega$ , but when the LDR sensor is exposed to light, the resistance drops dramatically, even down to a few ohms, depending on the light intensity. LDRs have a sensitivity that varies with the wavelength of the light applied and are non-linear devices.

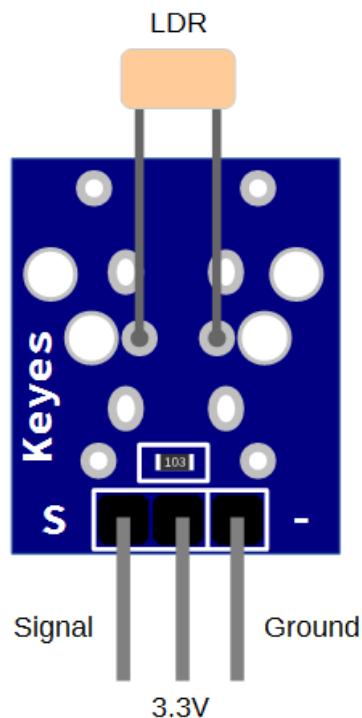
They are widely used in cameras, solar garden lights, clocks, mini night-lights, and a variety of light control devices.

Specifications from a [typical LDR<sup>92</sup>](#) show that as illumination increases, the resistance of an LDR decreases.

Light Level	Resistance
Moonlight	1,000,000 Ohms
60W bulb at 1m	6,000 Ohms
Fluorescent Lighting	1,000 Ohms
Bright sunlight	1 Ohm

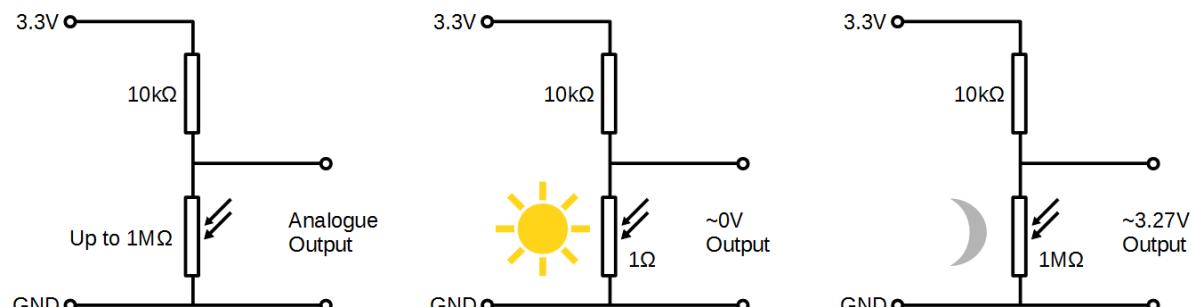
The Keyes KY-018 sensor board comprises an LDR and a fixed resistor with header pins for connecting the ground, the reference voltage (we will use the 3.3V from the Pi) and the sensors analog voltage output.

<sup>92</sup><http://kennarar.vma.is/thor/v2011/vgr402/ldr.pdf>



Keyes KY-018 Photoresister Sensor Board

If we consider a simplified circuit of our sensor, the LDR in series with a fixed resistor allows the variation in resistance to develop a variation in output voltage.



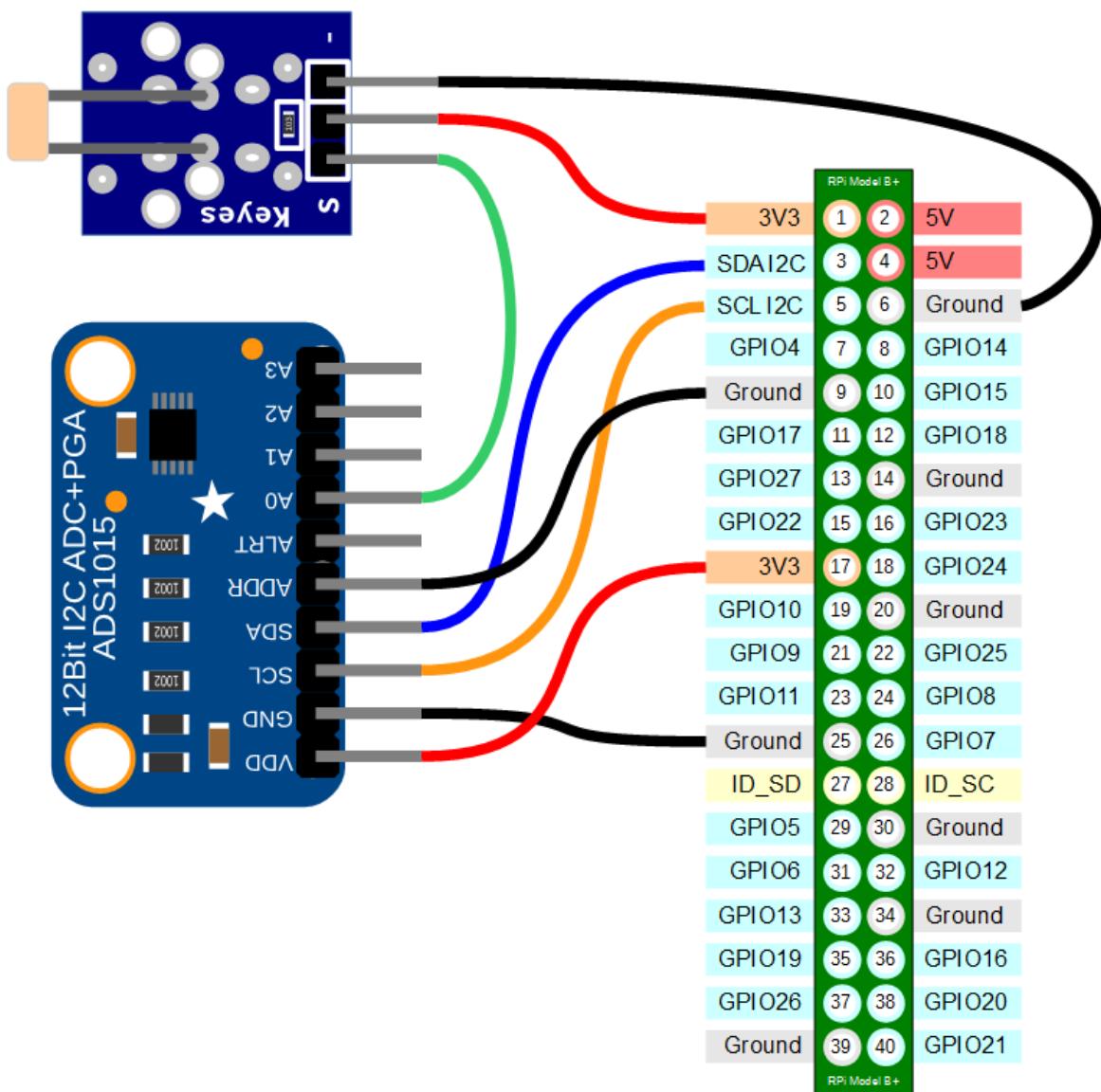
LDR Sensor Output Voltage

## Connect

The LDR sensor board should be connected with ground pin (labelled '-') to a ground connector, the reference voltage pin (in the case of the board shown below the centre pin) to a 3.3V connector and the signal output pin (labelled 'S') to the A0 pin on the ADS1015 ADC.

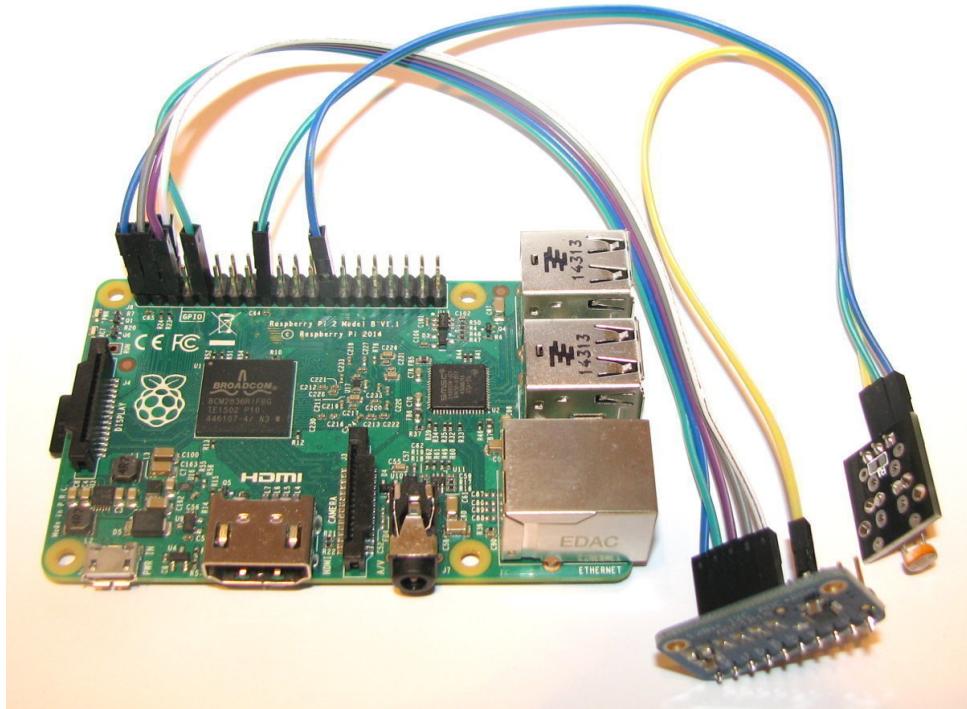
The ADS1015 board should have the VDD pin connected to a 3.3V pin, the GND to a ground pin, the SCL pin to the SCL I2C connector on pin 5 and the SDA pin to the SDA I2C connector on pin 3 and lastly the ADDR pin should be connected to ground.

Both boards will support a connection to the 'VDD' and reference voltage connector of 5V, **but this is not advisable for the Raspberry Pi** as the resulting signal levels on the SDA connector *may* be higher than desired for the Pi's input. This connection can be safely used with an Arduino board.



LDR Sensor Board Connection

Connecting the sensor practically can be achieved in a number of ways. You could use a Pi Cobbler break out connector mounted on a bread board connected to the appropriate pins. But because the connection is relatively simple we could build a minimal configuration that will plug directly onto the pins using Dupont header connectors and jumper wire. The image below shows how simple this can be.



Physical Connection of ADS1015 and LDR Sensor

## Test

Since the ADS1015 uses the I2C protocol to communicate, we need to load the appropriate kernel support modules onto the Raspberry Pi to allow this to happen.

Firstly make sure that our software is up to date

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

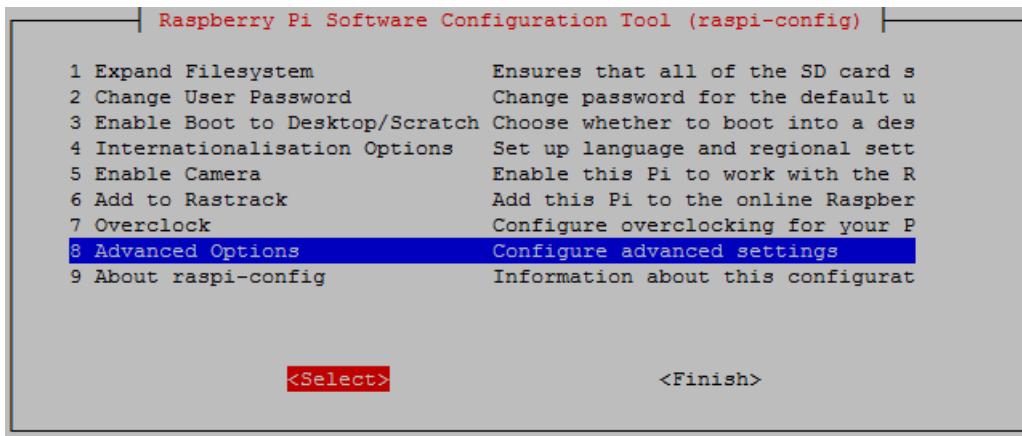
Since we are using the Raspbian distribution there is a simple method to start the process of configuring the Pi to use the I2C protocol.

We can start by running the command;

```
sudo raspi-config
```

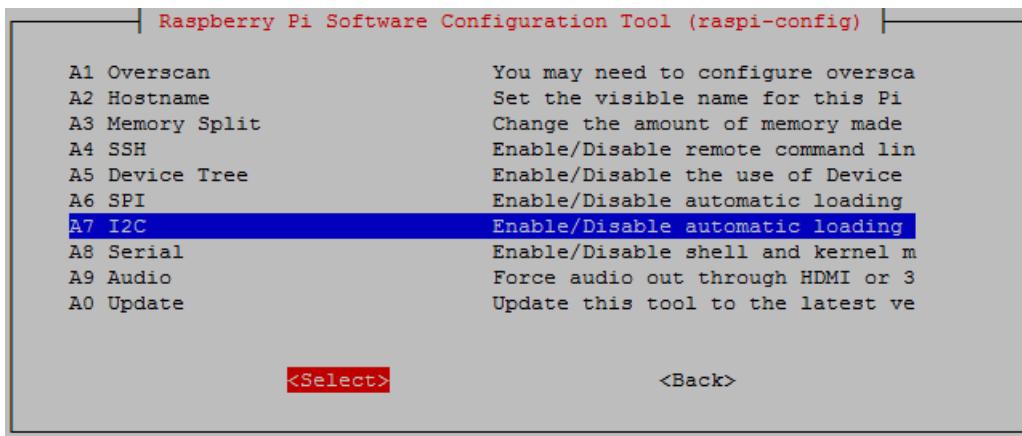
This will start the Raspberry Pi Software Configuration Tool.

On the first page select the Advanced Options with the space bar and then tab to select



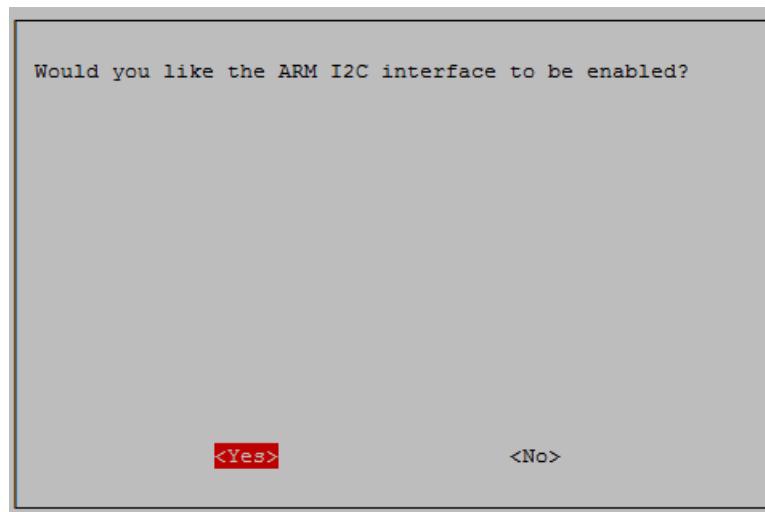
### Advanced Options

Then we select the I2C option for automatic loading of the kernel module;



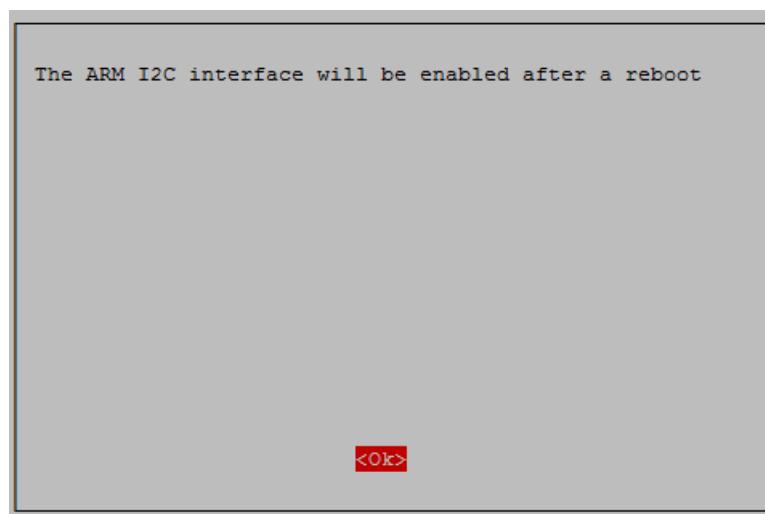
### Automatic Loading

Would we like the ARM I2C interface to be enabled? Yes we would;



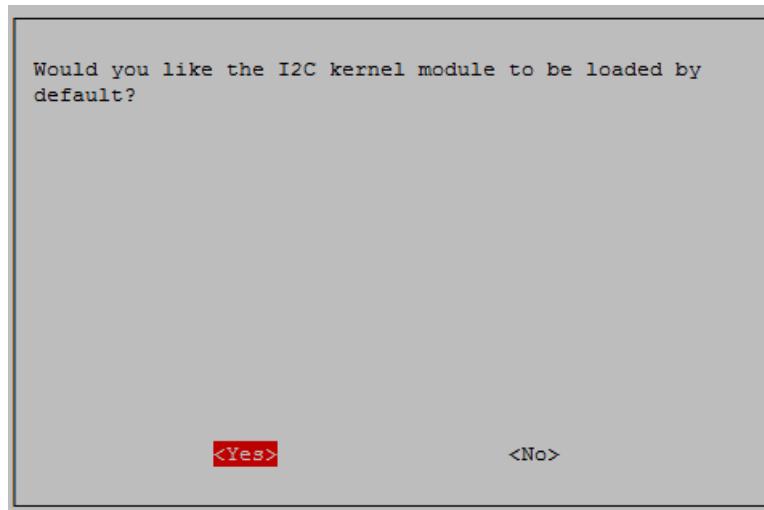
Enable ARM I2C Interface

We are helpfully informed that the I2C interface will be enabled after the next reboot.



Enabled After Reboot

Would we like the I2C kernel module to be loaded by default? Yes we would;



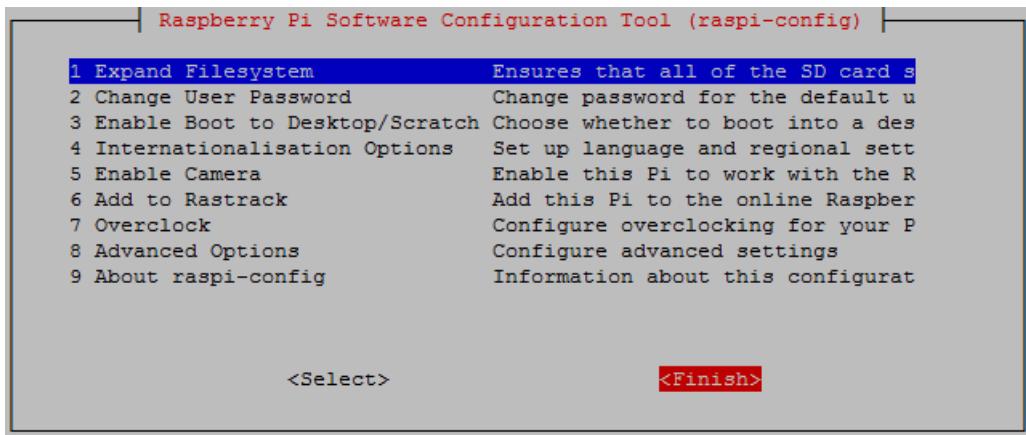
I2C Kernel Module Loaded by Default

We are helpfully informed that the I2C kernel module will be loaded by default after the next reboot.



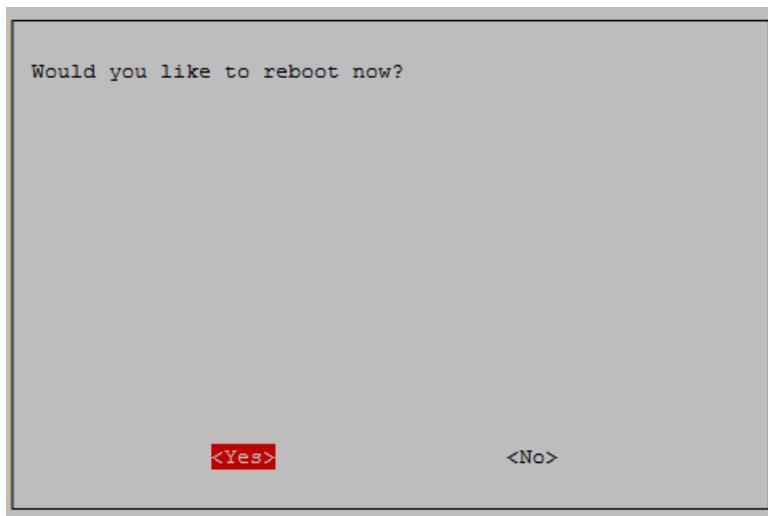
Loaded by Default

Press tab to select 'Finish'.



### We're Finished

And yes, we would like to reboot.



### Reboot

There's still some work to do to get things sorted. We need to edit the /etc/modules file using:

```
sudo nano /etc/modules
```

Where we need to add the following two lines to the end of the /etc/modules file:

```
i2c-bcm2708  
i2c-dev
```

Under some circumstances (depending on the kernel version we are using) we would also need to update the /boot/config.txt file. We can do this using;

```
sudo nano /boot/config.txt
```

Make sure that the following lines are in the file;

```
dtparam=i2c1=on  
dtparam=i2c_arm=on
```

Then we should load tools for working with I2C devices using the following command;

```
sudo apt-get install i2c-tools
```

... and now we need to reboot to load the config.txt file from earlier

```
sudo reboot
```

We can now check to see if our sensor is working using:

```
sudo i2cdetect -y 1
```



If we were using an older B model of Raspberry Pi with 256MB of RAM, we would need to use `sudo i2cdetect -y 0`.

The output should look something like;

```
pi@raspberrypi ~ $ sudo i2cdetect -y 1  
 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  
00:          -- -- -- -- -- -- -- -- -- -- -- --  
10:          -- -- -- -- -- -- -- -- -- -- -- --  
20:          -- -- -- -- -- -- -- -- -- -- -- --  
30:          -- -- -- -- -- -- -- -- -- -- -- --  
40:          -- -- -- -- -- 48 -- -- -- -- -- --  
50:          -- -- -- -- -- -- -- -- -- -- -- --  
60:          -- -- -- -- -- -- -- -- -- -- -- --  
70:          -- -- -- -- -- -- -- -- -- -- -- --
```

This shows us that we have detected our ADS1015 on address ‘48’. The ADS1015 can support four different addresses as shown on page 17 of the [data sheet<sup>93</sup>](#). The address is selected by what the ADDR (short for address!) pin on the board is connected to;

ADDR PIN	ADDRESS
Ground	48
VDD	49
SDA	4A
SCL	4B

As we noted earlier, this means we can connect up to four ADS1015’s on the same I2C bus.

Now we want to install Python libraries designed to read the values from the ADS1015. The library we are going to use was designed specifically to work with the [Adafruit ADS1015/ADS1115 ADCs<sup>94</sup>](#). In carrying out this library development, Adafruit have invested a not inconsiderable amount of time and resources. In return please consider supporting Adafruit and open-source hardware by purchasing products from [Adafruit<sup>95</sup>](#)!

```
sudo apt-get install git build-essential python-dev python-smbus
```

Now we create a directory that we will use to download the Adafruit Python library (assuming that we’re starting from our home directory);

```
mkdir adc
cd adc
```

Now we will download the library into the adc directory (the following command retrieves the library from the github site);

```
git clone https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code
```

Now that we’ve downloaded it, we can run an example program as follows

```
cd /home/pi/adc/Adafruit-Raspberry-Pi-Python-Code/Adafruit_ADS1x15
python ads1x15_ex_singleended.py
```

---

<sup>93</sup><http://www.adafruit.com/datasheets/ads1015.pdf>

<sup>94</sup><http://www.adafruit.com/product/1083>

<sup>95</sup><https://www.adafruit.com/>

This will hopefully produce an output similar to the following;

1.558000

This is an indication of the voltage that the analog sensor is producing.

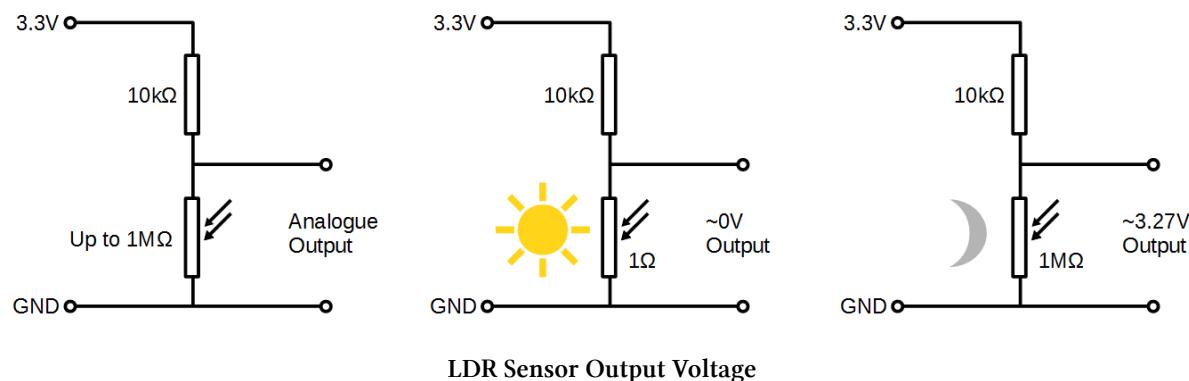
To test that the sensor is responding correctly shine a light on it and re-run the command;

```
python ads1x15_ex_singleended.py
```

With a brighter light we should have a lower voltage;

0.44000

If we remember back to our earlier diagram showing the type of connection this helps put the changes into context



At this point we have successfully read and displayed an analog signal from a sensor using the Raspberry Pi.

## Record

To record this data we will use a Python program that connects to our sensor via the ADC, reads the value of voltage and writes that value into our MySQL database. At the same time a time stamp will be added automatically.

This code will record the value when executed, but we will need to edit the crontab to allow the value to be recorded continuously (at a set interval).

## Database preparation

First we will set up our database table that will store our data.

Using the phpMyAdmin web interface that we set up, log on using the administrator (root) account and select the ‘measurements’ database that we created as part of the initial set-up.

The screenshot shows the 'Create table on database measurements' page in phpMyAdmin. There is a form with the following fields:  
 - Name: light  
 - Number of columns: 2  
 - Go button (highlighted in red)

Create the MySQL Table

Enter in the name of the table and the number of columns that we are going to use for our measured values. In the screenshot above we can see that the name of the table is ‘light’ and the number of columns is ‘2’.

We will use two columns so that we can store the voltage that corresponds to the light level and the time it was recorded (‘dtg’).

Once we click on ‘Go’ we are presented with a list of options to configure our table’s columns. Don’t be intimidated by the number of options that are presented, we are going to keep the process as simple as practical.

For the first column we can enter the name of the ‘Column’ as ‘dtg’ (short for date time group) the ‘Type’ as ‘TIMESTAMP’ and the ‘Default’ value as ‘CURRENT\_TIMESTAMP’. For the second column we will enter the name ‘temperature’ and the type is ‘FLOAT’. For the third column we will enter the name ‘pressure’ and the type is also ‘FLOAT’.

The screenshot shows the 'Configure the MySQL Table Columns' page in phpMyAdmin. There is a form with the following fields:  
 - Name: light  
 - Number of columns: 2  
 - Go button (highlighted in red)

Configure the MySQL Table Columns

Scroll down a little and click on the ‘Save’ button and we’re done.



Save the MySQL Table Columns



### Why did we choose those particular settings for our table?

Our ‘dtg’ column needs to store a value of time that includes the date and the time, so the advantage of selecting TIMESTAMP in this case is that we can select the default value to be the current time. This means that when we write our data to the table we only need to write the ‘temperature’ and ‘pressure’ and the ‘dtg’ will be entered automatically for us. The disadvantage of using ‘TIMESTAMP’ is that it has a more limited range than DATETIME. TIMESTAMP can only have a range between ‘1970-01-01 00:00:01’ to ‘2038-01-19 03:14:07’.

Our voltage readings are generated (by the Python library) as a value with decimal places. As a result we need to use a numerical format that supports numbers with decimal places. There are a range of options for defining the ranges for decimal numbers, but FLOAT allows us to ignore the options (at the expense of efficiency) and rely on our recorded values being somewhere between -3.402823466E+38 and 3.402823466E+38 (if our temperature falls outside those extremes we are in trouble).

## Record the readings

To utilise the Python libraries and to run our Python script from our home directory we will want to change into the the directory where the library is and copy the appropriate file from the downloaded directory to our home directory then we can change back into our home directory;

```
cd /home/pi/adc/Adafruit-Raspberry-Pi-Python-Code/Adafruit_ADS1x15
cp Adafruit_ADS1x15.py /home/pi/
cd /home/pi
```

The following Python code is a script which allows us to check the state of our LDR sensor, return the value for voltage that corresponds to the amount of light (via the ADC) and writes that value to our database.

The full code can be found in the code samples bundled with this book (adc.py).

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import MySQLdb as mdb
import logging
from Adafruit_ADS1x15 import ADS1x15

# Setup logging
logging.basicConfig(filename='/home/pi/adc_error.log',
    format='%(asctime)s %(levelname)s %(name)s %(message)s')
logger=logging.getLogger(__name__)

# Function for storing readings into MySQL
def insertDB(level):

    try:

        con = mdb.connect('localhost',
                          'pi_insert',
                          'xxxxxxxxxx',
                          'measurements');
        cursor = con.cursor()

        sql = "INSERT INTO light(level) \
VALUES ('%s')" % \
(level)
        cursor.execute(sql)
        sql = []
        con.commit()

        con.close()

    except mdb.Error, e:
        logger.error(e)

# Get readings from sensor and store them in MySQL

ADS1015 = 0x00 # 12-bit ADC
gain = 4096 # +/- 4.096V
sps = 250 # 250 samples per second

# Initialise the ADC
adc = ADS1x15(ic=ADS1015)

# Read channel 0 in single-ended mode using the settings above
level = adc.readADCSingleEnded(0, gain, sps) / 1000
```

```
insertDB(level)
```

This script can be saved in our home directory (/home/pi) as adc.py and can be run by typing:

```
python adc.py
```

Once the command is run we should be able to check our MySQL database and see an entry for the times that the sensor was checked along with the corresponding level readings.

	dtg	level
<input type="checkbox"/>	Edit <input type="checkbox"/> Inline Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	2015-06-23 22:10:05 2.284
<input type="checkbox"/>	Edit <input type="checkbox"/> Inline Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	2015-06-23 22:15:50 2.28
<input type="checkbox"/>	Edit <input type="checkbox"/> Inline Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	2015-06-23 22:23:57 2.282
<input type="checkbox"/>	Edit <input type="checkbox"/> Inline Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	2015-06-23 22:24:07 2.28
<input type="checkbox"/>	Edit <input type="checkbox"/> Inline Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	2015-06-23 22:24:09 2.282
<input type="checkbox"/>	Edit <input type="checkbox"/> Inline Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	2015-06-23 22:24:23 2.282

Save the MySQL Table Columns

## Code Explanation

The script starts by importing the modules that it's going to use for the process of reading and recording the measurements;

```
import MySQLdb as mdb
import logging
from Adafruit_ADS1x15 import ADS1x15
```



Python code in one module gains access to the code in another module by the process of importing it. The `import` statement invokes the process and combines two operations; it searches for the named module, then it binds the results of that search to a name in the local scope.

Then the code sets up the `logging module`<sup>96</sup> so that any failures in writing information to the database are written to the file /home/pi/adc\_error.log.

---

<sup>96</sup><http://pymotw.com/2/logging/>

```
logging.basicConfig(filename='/home/pi/adc_error.log',
    format='%(asctime)s %(levelname)s %(name)s %(message)s')
logger=logging.getLogger(__name__)
```

In the following function where we are getting our reading or writing to the database we write to the log file if there is an error. Otherwise it will accept value of level and write it to our database.

```
def insertDB(level):

    try:

        con = mdb.connect('localhost',
                          'pi_insert',
                          'xxxxxxxxxx',
                          'measurements');
        cursor = con.cursor()

        sql = "INSERT INTO light(level) \
VALUES ('%s')" % \
(level)
        cursor.execute(sql)
        sql = []
        con.commit()

        con.close()

    except mdb.Error, e:
        logger.error(e)
```

Which brings us to the main part of our code;

```
ADS1015 = 0x00 # 12-bit ADC
gain = 4096     # +/- 4.096V
sps = 250       # 250 samples per second

adc = ADS1x15(ic=ADS1015)

level = adc.readADCSingleEnded(0, gain, sps) / 1000

insertDB(level)
```

First we declare the variables that we will use when collecting our data. We set the model number of the ADS1x15 we are using to '0x00' to correspond to the ADS1015. Then we can set the amount of gain (gain = 4096) and the number of samples per second the ADC will carry out (sps =

250). These are the default values that are found in the sample program and while there is a wide range of options available depending on our application, in this case the default will be adequate.

Then we initialise the ADC (`adc = ADS1x15(ic=ADS1015)`) then read in our level using the declared parameter values (`level = adc.readADCSingleEnded(0, gain, sps) / 1000`).

Finally we call the function to insert the value into the database (`insertDB(level)`).

It's a fairly simple block of code that is made simple by virtue of the work that has been put into the associated libraries from Adafruit.

## Recording data on a regular basis with cron

While our code is a thing of simple elegance, it only records each time it is run.

What we need to implement is a schedule so that at a regular time, the program is run. This is achieved using cron via the crontab. While we will cover the requirements for this project here, you can read more about the [crontab](#) in the [Glossary](#).

To set up our schedule we need to edit the crontab file. This is done using the following command;

```
crontab -e
```

Once run it will open the crontab in the nano editor. We want to add in an entry at the end of the file that looks like the following;

```
*/10 * * * * /usr/bin/python /home/pi/adc.py
```

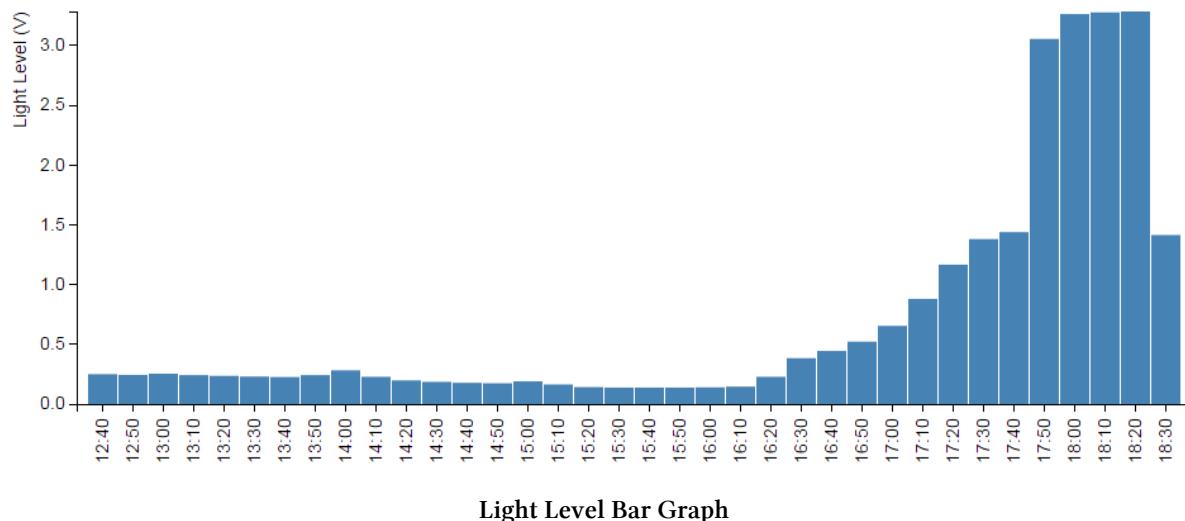
This instructs the computer that every 10 minutes we will run the command `/usr/bin/python /home/pi/adc.py` (which if we were at the command line in the pi home directory we would run as `python adc.py`, but since we can't guarantee where the script will be running from, we are supplying the full path to the `python` command and the `adc.py` script).

Save the file and the computer will start running the program on its designated schedule and we will have level entries written to our database every 10 minutes.

Job done! We're measuring and recording our analog sensor!

## Explore

To explore our light level data we will use a web based graph that shows the voltage level that has come from our ADC as bars on a bar graph.



Light Level Bar Graph

The graph will be drawn using the [d3.js<sup>97</sup>](#) JavaScript library and the data will be retrieved via a PHP script that queries our MySQL database.

The astute reader will probably be thinking “Why don’t we just use a line graph similar to the one in the simple temperature measurement project?”. That would be a very good question. The answer is that you already know how to make a simple line graph, but bar graphs are still a mystery! So read on.... (but make a line graph as well)

## The Code

The following code is a PHP file that we can place on our Raspberry Pi’s web server (in the `/var/www` directory) that will allow us to view the last 6 hours of data in 10 minute blocks.



There are many sections of the code which have been explained already in the set-up section of the book that describes a simple line graph for a single temperature measurement. Where these occur we will be less thorough with the explanation of how the code works.

The full code can be found in the code samples bundled with this book (`adc-bar.php`).

---

<sup>97</sup><http://d3js.org/>

```

<?php

$hostname = 'localhost';
$username = 'pi_select';
$password = 'xxxxxxxxxx';

try {
    $dbh = new PDO("mysql:host=$hostname;dbname=measurements",
                   $username, $password);

    /*** The SQL SELECT statement ***/
    $sth = $dbh->prepare("
        SELECT * FROM (
            SELECT `dtg` AS date,
                   `level` AS value
            FROM `light`
            ORDER BY date DESC
            LIMIT 0,36
        ) sub
        ORDER BY date ASC
    ");
    $sth->execute();

    /* Fetch all of the remaining rows in the result set */
    $result = $sth->fetchAll(PDO::FETCH_ASSOC);

    /*** close the database connection ***/
    $dbh = null;
}

catch(PDOException $e)
{
    echo $e->getMessage();
}

$json_data = json_encode($result);

?>

<!DOCTYPE html>
<meta charset="utf-8">

<head>
    <style>

        .axis {
            font: 14px sans-serif;
        }
    </style>

```

```
.axis path,
.axis line {
  fill: none;
  stroke: #000;
  shape-rendering: crispEdges;
}

</style>
</head>

<body>

<script src="http://d3js.org/d3.v3.min.js"></script>

<script>

var margin = {top: 20, right: 20, bottom: 70, left: 60},
  width = 960 - margin.left - margin.right,
  height = 400 - margin.top - margin.bottom;

// Parse the date / time
var parseDate = d3.time.format("%Y-%m-%d %H:%M:%S").parse;

// specify the scale/range for each dimension
var x = d3.scale.ordinal().rangeRoundBands([0, width], .05);
var y = d3.scale.linear().range([height, 0]);

// axis formatting
var xAxis = d3.svg.axis()
  .scale(x)
  .orient("bottom")
  .tickFormat(d3.time.format("%H:%M"));
var yAxis = d3.svg.axis()
  .scale(y)
  .orient("left")
  .ticks(10);

// setup the svg area
var svg = d3.select("body").append("svg")
  .attr("width", width + margin.left + margin.right)
  .attr("height", height + margin.top + margin.bottom)
.append("g")
  .attr("transform",
    "translate(" + margin.left + "," + margin.top + ")");


```

```
// Get the data
<?php echo "data=".json_data." ?>

// wrangle the data into the correct formats and units
data.forEach(function(d) {
    d.date = parseDate(d.date);
    d.value = +d.value;
});

// Scale the range of the data
x.domain(data.map(function(d) { return d.date; }));
y.domain([0, d3.max(data, function(d) { return d.value; })]);

// Add the X Axis
svg.append("g")
    .attr("class", "x axis")
    .attr("transform", "translate(0," + height + ")")
    .call(xAxis)
    .selectAll("text")
        .style("text-anchor", "end")
        .attr("dx", "-.8em")
        .attr("dy", "-.35em")
        .attr("transform", "rotate(-90)");
    
// Add the Y Axis and the Y axis label
svg.append("g")
    .attr("class", "y axis")
    .call(yAxis)
    .append("text")
        .attr("transform", "rotate(-90)")
        .attr("y", -50)
        .attr("dy", ".71em")
        .style("text-anchor", "end")
        .text("Light Level (V)");

// Add the bars
svg.selectAll("bar")
    .data(data)
    .enter().append("rect")
        .style("fill", "steelblue")
        .attr("x", function(d) { return x(d.date); })
        .attr("width", x.rangeBand())
        .attr("y", function(d) { return y(d.value); })
        .attr("height", function(d) { return height - y(d.value); });

</script>
```

```
</body>
```

This graph contains some of the common elements that what we have explored in the single temperature measurement project. However, in this code we introduce the concept of using filled rectangles to form a bar graph.

The code will automatically try to collect 36 data points from a data set that is recording a value every 10 minutes. As a result this will present 6 hours worth of data.

## PHP

The PHP block at the start of the code is mostly the same as our example code for our single temperature measurement project. The significant difference however is in the select statement.

```
SELECT * FROM (
    SELECT `dtg` AS date,
    `level` AS value
    FROM `light`
    ORDER BY date DESC
    LIMIT 0,36
) sub
ORDER BY date ASC
```

The query only collects two columns of values. The Date Time Group (date) and the light level (level). The major difference in this query is that it has one query nested inside another.

This query...

```
SELECT `dtg` AS date,
`level` AS value
FROM `light`
ORDER BY date DESC
LIMIT 0,36
```

.... is wrapped in this query.

```
SELECT * FROM (
    subquery
) sub
ORDER BY date ASC
```

This might seem slightly unusual, but it is done so that we can select the last 36 data points and arrange those points from oldest to newest.

The subquery (the nested one) selects the 36 points we want and by ordering them by date in descending (DESC) order we make sure we have the latest ones. But then they would be in an

order that if we were to plot them they would appear to go from newest to oldest in our graph. So what we do is wrap that query in parentheses and give it an alias (`sub`) and then we select all of those results and order them by date ascending (ASC). It seems a little ‘hacky’ and there would be alternative ways to do it either in PHP or JavaScript, but it has to happen somewhere, so there it is. Any interested readers who want to suggest different avenues please forward them through and I will publish them here with suitable attribution :-).

## CSS (Styles)

The styles that are applied to the elements of the graphic in the CSS area are all done for the axes.

```
.axis {
    font: 14px sans-serif;
}

.axis path,
.axis line {
    fill: none;
    stroke: #000;
    shape-rendering: crispEdges;
}
```

## JavaScript

The code has very similar elements to our single temperature measurement script and comparing both will show us that we are doing similar things in the code.

The things that are for all intents the same as the single temperature code are;

- Setting up the margins
- Declaring the `parseDate` function
- Setting the scales/ranges
- Formatting the axes
- Setting up the SVG area
- Loading the data and
- Wrangling the data into the correct format
- Scaling the range of the data

There are only three blocks of code left and two of them are adding the X and Y axes. The last is adding the bars themselves, so the code is pretty darned similar. However, those three blocks are a little different, so let’s describe what’s going on;

Firstly we add the X axis;

```
svg.append("g")
  .attr("class", "x axis")
  .attr("transform", "translate(0," + height + ")")
  .call(xAxis)
  .selectAll("text")
    .style("text-anchor", "end")
    .attr("dx", "-.8em")
    .attr("dy", "-.35em")
    .attr("transform", "rotate(-90)" );
```

This is placed in the correct position `.attr("transform", "translate(0," + height + ")")` and the text is positioned (using dx and dy) and rotated (`.attr("transform", "rotate(-90)" );`) so that it is aligned vertically.

Then when add the Y axis and its label;

```
svg.append("g")
  .attr("class", "y axis")
  .call(yAxis)
  .append("text")
    .attr("transform", "rotate(-90)" )
    .attr("y", -50)
    .attr("dy", ".71em")
    .style("text-anchor", "end")
    .text("Light Level (V)");
```

We also move the rotated text for the label to an appropriate position using y attribute.

At first the rotation and movement of text elements can seem confusing. In this case it's easy to see the rotation by 90 degrees (`.attr("transform", "rotate(-90)" )`), but the subsequent translation in the y direction (`.attr("y", -50)`) actually ends up moving the text in the x direction. This is because the rotate function call has also rotated our axes for the text. Therefore where a movement would have been upwards, after a rotation by -90 degrees, it is now to the left.

For more information on manipulating text elements with D3 check out [this section from D3 Tips and Tricks<sup>98</sup>](#).

Lastly we add the code that appends the bars;

---

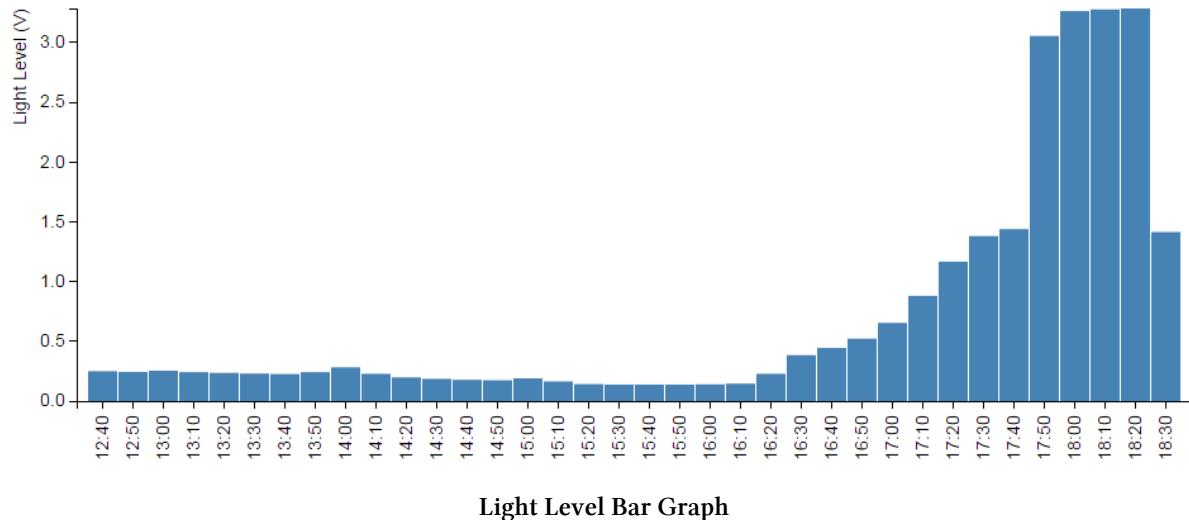
<sup>98</sup><https://leanpub.com/D3-Tips-and-Tricks/read#leanpub-auto-text>

```
svg.selectAll("bar")
  .data(data)
  .enter().append("rect")
    .style("fill", "steelblue")
    .attr("x", function(d) { return x(d.date); })
    .attr("width", x.rangeBand())
    .attr("y", function(d) { return y(d.value); })
    .attr("height", function(d) { return height - y(d.value); });
```

This block of code creates the bars (`selectAll("bar")`) and associates each of them with a data set (`.data(data)`).

We then append a rectangle (`.append("rect")`) with values for x/y position and height/width as configured in our earlier code.

And there we have it. Our temperature and pressure being read and presented in a stylish dual line graph.



If you want a closer explanation for this piece of code, download a copy of [D3 Tips and Tricks](#)<sup>99</sup> for this and a whole swag of other information.

---

<sup>99</sup><https://leanpub.com/D3-Tips-and-Tricks>

## Bibliography

RPi and I2C Analog-Digital Converter ([OpenLabTools<sup>100</sup>](#))

ADS1015 12-Bit ADC - 4 Channel with Programmable Gain Amplifier([Adafruit<sup>101</sup>](#))

ADS1015 datasheet ([Adafruit<sup>102</sup>](#))

Adafruit 4-Channel ADC Breakouts ([Adafruit Learning System<sup>103</sup>](#))

Analog Sensors On The Raspberry Pi Using An MCP3008 ([Matt, raspberrypi-spy.co.uk<sup>104</sup>](#))

Analog Input Board for the Espiresso Pressure Sensor ([int03.co.uk<sup>105</sup>](#))

Arduino KY-018 Photo resistor module ([TkkrLab<sup>106</sup>](#))

[Shenzhen KEYES DIY Robot co., Ltd<sup>107</sup>](#)

Light dependant resister datasheet ([Sunroom Technologies<sup>108</sup>](#))

---

<sup>100</sup>[http://openlabtools.eng.cam.ac.uk/Resources/Datalog/RPi\\_ADS1115/](http://openlabtools.eng.cam.ac.uk/Resources/Datalog/RPi_ADS1115/)

<sup>101</sup><http://www.adafruit.com/products/1083>

<sup>102</sup><http://www.adafruit.com/datasheets/ads1015.pdf>

<sup>103</sup><https://learn.adafruit.com/adafruit-4-channel-adc-breakouts?view=all>

<sup>104</sup><http://www.raspberrypi-spy.co.uk/2013/10/analogue-sensors-on-the-raspberry-pi-using-an-mcp3008/>

<sup>105</sup><http://int03.co.uk/http://int03.co.uk/blog/2014/12/17/analogue-input-board-for-the-pressure-sensor-espresso/>

<sup>106</sup>[https://tkkrlab.nl/wiki/Arduino\\_KY-018\\_Photo\\_resistor\\_module](https://tkkrlab.nl/wiki/Arduino_KY-018_Photo_resistor_module)

<sup>107</sup><http://en.keyes-robot.com/index.aspx>

<sup>108</sup><http://kennarar.vma.is/thor/v2011/vgr402/ldr.pdf>

# Web Scraping

The term ‘web scraping’ refers to the process of programmatically retrieving data from a web page. There are a wide range of ways that it can be accomplished and there is a degree of responsibility that needs to be taken to do it in a way that doesn’t violate [Wheaton’s law<sup>109</sup>](#).

**i** To spell it out a little more clearly, checking a web page every day to discover and record a changing value is unlikely to cause anyone any angst. Checking a web page every second is probably unnecessary and might be annoying to the person who runs the web page. Checking all the web pages of an organisation to plagiarise their data could result in [legal action<sup>110</sup>](#).

## OK, so what *is* web scraping?

Web scraping is the act of retrieving some form of information from a web page. The example we will work through is where we want to be able to collect, store and compare the number of readers of the books [R Programming for Data Science<sup>111</sup>](#) and [The Elements of Data Analytic Style<sup>112</sup>](#) by Messrs Roger Peng and Jeff Leek respectively.

The data we're interested in!

30789 READERS 147 PAGES

R Programming for Data Science

Roger D. Peng

This book brings the fundamentals of R programming to you, using the same material developed as part of the industry-leading Johns Hopkins Data Science Specialization. The skills taught in this book will lay the foundation for you to begin your journey learning data science. See the packages below to obtain datasets, R code files, and video lectures.

FREE SAMPLE

Download

UPDATED 12 DAYS AGO

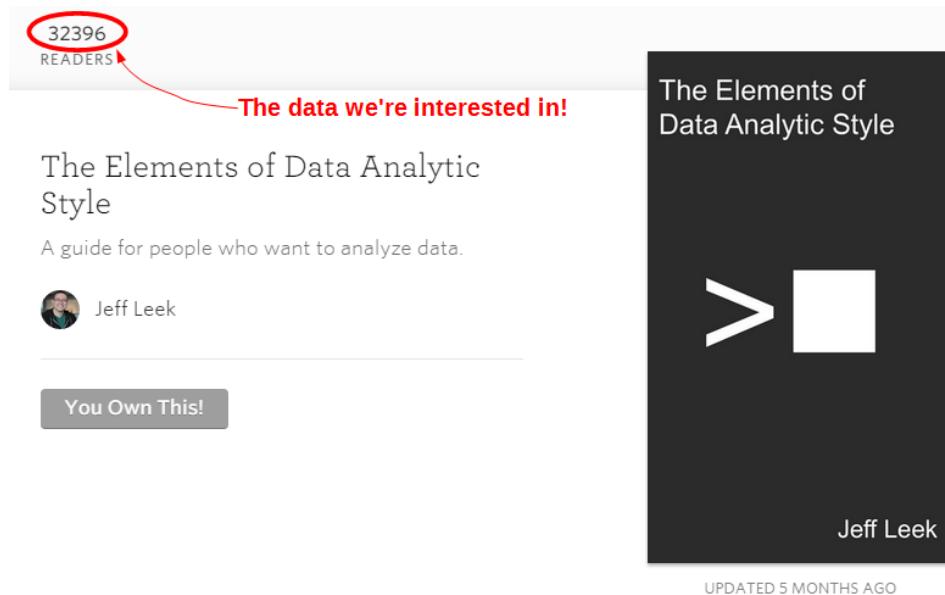
### R Programming for Data Science by Roger Peng

<sup>109</sup><http://knowyourmeme.com/memes/wheatons-law>

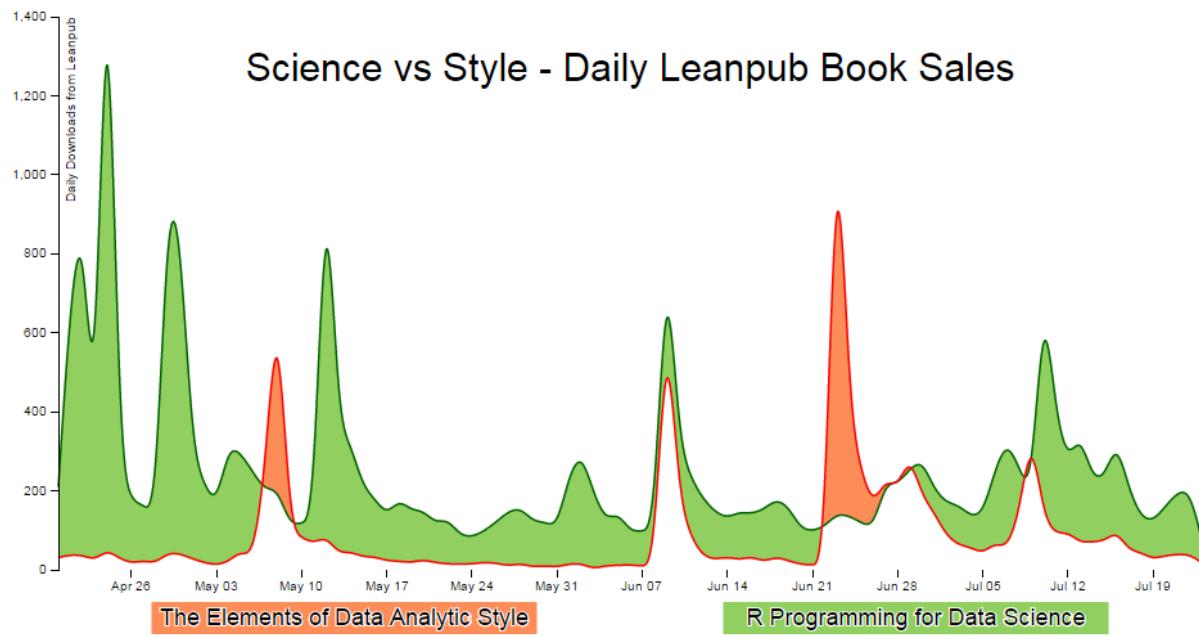
<sup>110</sup>[https://en.wikipedia.org/wiki/Web\\_scraping#Legal\\_issues](https://en.wikipedia.org/wiki/Web_scraping#Legal_issues)

<sup>111</sup><https://leanpub.com/rprogramming>

<sup>112</sup><https://leanpub.com/datastyle>



We will check the check the main page of each book every day, store the number of readers in a database and after a period of time we will be able to compare the number or readers that each book gains in a day using a difference chart.





### Why these two books?

Both Roger and Jeff work at the Johns Hopkins Bloomberg School of Public Health where Roger is an Associate Professor of Biostatistics, and Jeff is an Associate Professor of Biostatistics and Oncology. While both are doing amazing work to improve peoples health and well-being (amongst other things), their success in publishing means that there is a really interesting set of data that might reflect a [degree of competition<sup>113</sup>](#). So it's definitely a good natured selection and hopefully someone interested in the Raspberry Pi will also find some interesting reading in *R Programming for Data Science*, *Exploratory Data Analysis with R* and *The Elements of Data Analytic Style*.



### It's tricky

Fair warning. This example has some trickier elements in it. Using regular expressions to extract data has broken many a strong willed person. The difference chart that we'll build to explore the information takes a bit of thinking about if you want to truly understand the technique. Don't let any of this put you off, but stay vigilant!

## Measure

### Hardware required

Only the Raspberry Pi and an internet connection! All the data that we're going to source will come from the internet.

### Software required

The technique that we'll use to scrape the pages will use the PHP 'curl' library. cURL is a command line tool for transferring data with a URL syntax, such as (but not limited to) FTP, HTTP, HTTPS, POP3, SFTP and SMTP. The integration of cURL into PHP give us a programmatic way to load the contents of a web page using PHP.

However, since it isn't a standard part of the PHP installation (it is a separate library) we need to install it separately.

Before we get to the installation of the library, we will need to ensure that our Linux distribution is up to date. To do this type in the following line which will find the latest lists of available software;

```
sudo apt-get update
```

You should see a list of text scroll up while the Pi is downloading the latest information.

Then we want to upgrade our software to latest versions from those lists using;

<sup>113</sup><https://twitter.com/d3noob/status/611227825685725184/photo/1>

```
sudo apt-get upgrade
```



Without the update / upgrade, the following error may occur;

```
PHP Fatal error: Call to undefined function curl_init()
```

We will now be ready to install the PHP cURL library. We can do this from the command line as follows;

```
sudo apt-get install php5-curl  
sudo ls/etc/init.d/apache2 restart
```

The line to restart apache2 is required so that the `php-curl` functionality can be integrated into the web server.

## Let the scraping begin

As stated earlier, the process we are going to go through is using PHP to load the web page we're interested in and then we're going to parse out the specific portion of the page that we're wanting to capture. The two 'helpers' that we're going to use are `cURL` to load the page and regular expressions to parse out the information.

The full code that we can use to demonstrate how we will do this is as follows;

```
<?php  
  
$books = array(  
    array('https://leanpub.com/rprogramming',  
          'R Programming for Data Science'),  
    array('https://leanpub.com/dastyle',  
          'The Elements of Data Analytic Style')  
);  
  
for ($x = 0; $x <= sizeof($books)-1; $x++) {  
  
    $file_string = file_get_contents_curl($books[$x][0]);  
  
    $regex_pre =  
        '/<ul class=\\"book-details-list\\">\n<li class=\\"detail\\">\n';
```

```
$regex_apre = '\n<p>Readers</p>/s';
$regex_actual = '<span>(.*)</span>';
$regex = $regex_pre.$regex_actual.$regex_apre;

preg_match($regex,$file_string,$title);
$downloads = $title[1];

echo $downloads." ".$books[$x][1]."<br>";

}

function file_get_contents_curl($url) {
    $ch = curl_init();

    curl_setopt($ch, CURLOPT_SSL_VERIFYPeer, false);
    curl_setopt($ch, CURLOPT_AUTOREFERER, TRUE);
    curl_setopt($ch, CURLOPT_HEADER, 0);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt($ch, CURLOPT_FOLLOWLOCATION, TRUE);

    $data = curl_exec($ch);
    curl_close($ch);

    return $data;
}

?>
```



The full code can be found in the code samples bundled with this book as `scrape.php`.

The first thing that our file does is to store the details of the books that we are going to look up in an array called `$books`:

```
$books = array(
    array('https://leanpub.com/rprogramming',
          'R Programming for Data Science'),
    array('https://leanpub.com/dastyle',
          'The Elements of Data Analytic Style')
);
```

Then we loop through the array one book at a time;

```
for ($x = 0; $x <= sizeof($books)-1; $x++) {
```

Inside the loop we call the function `file_get_contents_curl` with the argument of the URL of the book's page `$books[$x][0]` (from the array);

```
$file_string = file_get_contents_curl($books[$x][0]);
```

The function `file_get_contents_curl` is (as the name suggests) a function to retrieve the contents of a web page using cURL. The function is as follows;

```
function file_get_contents_curl($url) {
    $ch = curl_init();

    curl_setopt($ch, CURLOPT_SSL_VERIFYPeer, false);
    curl_setopt($ch, CURLOPT_AUTOREFERER, TRUE);
    curl_setopt($ch, CURLOPT_HEADER, 0);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt($ch, CURLOPT_FOLLOWLOCATION, TRUE);

    $data = curl_exec($ch);
    curl_close($ch);

    return $data;
}
```

`'curl_init()'` initialises a cURL session before we start setting our options for the transfer.

`CURLOPT_SSL_VERIFYPeer, false` stops cURL from verifying the peer's certificate. This allows the retrieval of an 'https' page without verifying that the page is secure (or at least we ignore the status of the security certificate that it has).

`CURLOPT_AUTOREFERER, TRUE` automatically sets the Referrer: field in requests where it follows a 'Location: ' redirect (this takes into account the situation where the page that we're trying to download is actually being re-directed).

`CURLOPT_HEADER, 0` does not include the header in the output.

`CURLOPT_RETURNTRANSFER, 1` returns the transfer as a string.

`CURLOPT_URL` is the URL to fetch.

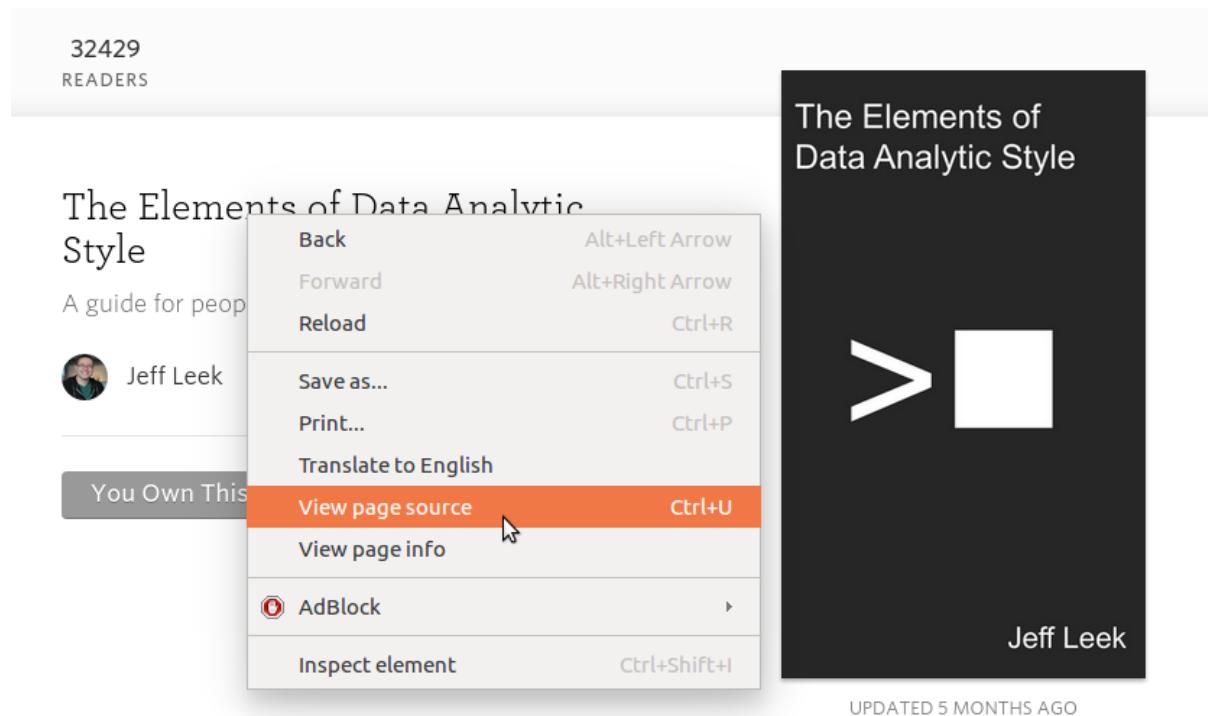
`CURLOPT_FOLLOWLOCATION, TRUE` means we will follow any 'Location: ' header that the server forwards as part of the HTTP header.

Then we store the file as a string into the variable `$data` before closing the connection and returning the string with the function call.

The variable `$file_string` now has the contents of the web page in it. This web page is comprised of all the HTML blocks and code that a web page requires to be displayed. Within that variable is a section of text that will look a little like the following;

```
<div id='book-metadata'>
<div class='large-container'>
<ul class='book-details-list'>
<li class='detail'>
<span>32429</span>
<p>Readers</p>
</li>
<li class='detail'>
```

Within this portion we can see our value for the number of readers! We can examine the underlying code for a page while in a web browser. Just right click on a portion of the page and a dialogue box should appear with an option to ‘View page source...’



### Viewing the page source

Armed with this information we are going to work out how to parse just the number of readers using regular expressions.

A regular expression is a special text string that describes a search pattern. We can compare regular expressions to wildcards where notations such as '\*' represent any number of characters. However, they are not wildcards and although there are some direct similarities, they cannot be used in the same way.

To try and make understanding the regular expression that we will use a little easier, I have broken it into three separate parts which we combine just before use.

There is the actual section of the HTML that holds the number of readers;

```
$regex_actual = '<span>( .*)</span>';
```

Here we are looking for text that falls between two `<span>` tags that consists of a single character (.) and any number (including zero) of additional characters (\*). This should find and return our number of readers.

However, because there are numerous `<span>` tags in the code, we want to make sure that we only extract the correct block of numbers. To do that we can define additional unique combinations of characters that occur before and / or after our numbers.

The section that appears before the span should match the following;

```
$regex_pre =
'<ul class=\`book-details-list\`>\n<li class=\`detail\`\>\n';
```

(The \n characters are representing new-line characters.)

The section that follows the spans will look like the following;

```
$regex_apre = '\n<p>Readers</p>/s';
```

There's a new-line followed by a paragraph tag with the word 'Readers' in it.

Then we combine the three strings to make our regex;

```
$regex = $regex_pre.$regex_actual.$regex_apre;
```

I'll be honest with you dear readers. Regular expressions are something of an art form. Those who can understand them are capable of magic that I can only wonder at and those (like myself) who are noobish can only hope to try to follow the rules<sup>114</sup> in the hope that it all becomes clear in the end. Don't despair however. Experiment a bit with the code, ask for assistance when you're struggling ([stackoverflow](#)<sup>115</sup>).

Finally we carry out our match (`preg_match`) and return our values in the array `$title`;

```
preg_match($regex,$file_string,$title);
$title = $title[1];

echo $downloads." ".$books[$x][1]."<br>";
```

In the code above we print out the number of readers followed by the title of the book so that it appears something like the following;

---

<sup>114</sup>[https://en.wikipedia.org/wiki/Regular\\_expression](https://en.wikipedia.org/wiki/Regular_expression)

<sup>115</sup><http://stackoverflow.com/questions/tagged/regex>

31095 R Programming for Data Science  
32426 The Elements of Data Analytic Style

This can be tested by either running the file from the command line;

```
php scrape.php
```

... or by browsing to the file in a web browser.

## Record

To record this data we will use a PHP script that checks the reader numbers and writes them and the book name into our MySQL database. At the same time a time stamp will be added automatically.

Our PHP script will write a group of reader numbers to the database and we will execute the program at a regular interval using [cron](#) (you can read a description of how to use the crontab (the cron-table) in the [Glossary](#)).

## Database preparation

First we will set up our database table that will store our data.

Using the phpMyAdmin web interface that we set up, log on using the administrator (root) account and select the ‘measurements’ database that we created as part of the initial set-up.

The screenshot shows a 'Create table' dialog box. The title bar says 'Create table on database measurements'. Below the title, there are two input fields: 'Name:' with the value 'downloads' and 'Number of columns:' with the value '3'. At the bottom right of the dialog is a 'Go' button.

Create the MySQL Table

Enter in the name of the table and the number of columns that we are going to use for our measured values. In the screenshot above we can see that the name of the table is ‘downloads’ and the number of columns is ‘3’.

We will use three columns so that we can store the number of readers, the time it the number was recorded and the name of the book.

Once we click on ‘Go’ we are presented with a list of options to configure our table’s columns. Don’t be intimidated by the number of options that are presented, we are going to keep the process as simple as practical.

For the first column we can enter the name of the ‘Column’ as ‘dtg’ (short for date time group) the ‘Type’ as ‘TIMESTAMP’ and the ‘Default’ value as ‘CURRENT\_TIMESTAMP’. For the second column we will enter the name ‘downloaded’ and the ‘Type’ is ‘INT’ (we won’t use a default value). For the third column we will enter the name ‘book\_name’ and the type is ‘VARCHAR’ with a ‘Length/Values’ of 60.

**Create Table**

**Table name:** downloads

**Structure**

Column	dtg	downloaded	book_name
Type	TIMESTAMP	INT	VARCHAR
Length/Values <sup>1</sup>			60
Default <sup>2</sup>	CURRENT_TIMESTAMP	None	None

Configure the MySQL Table Columns

Scroll down a little and click on the ‘Save’ button and we’re done.

Save the MySQL Table Columns

**Save** Or Add 1 column(s) **Go**



## Why did we choose those particular settings for our table?

Our ‘dtg’ column needs to store a value of time that includes the date and the time, so the advantage of selecting TIMESTAMP in this case is that we can select the default value to be the current time which means that when we write our data to the table we only need to write the ‘downloaded’ and ‘book\_name’ values and the ‘dtg’ will be entered automatically for us. The disadvantage of using ‘TIMESTAMP’ is that it has a more limited range than DATETIME. TIMESTAMP can only have a range between ‘1970-01-01 00:00:01’ to ‘2038-01-19 03:14:07’.

Our downloaded values are always going to be integers. There are a range of options for defining the ranges for integers, but INT allows us to ignore the options (at the expense of efficiency) and rely on our recorded values being somewhere between -2147483648 and 2147483647 (if our reader numbers fall outside those extremes we are talking about the best selling book of all time).

The book names are a combination of numbers, characters and letters, so we will use a variable type ‘VARCHAR’ which is for characters. We can also specify the maximum length of the information stored in the database to make things a little more efficient. In theory we could use the ‘CHAR’ type which is more efficient, but in this instance I prefer ‘VARCHAR’ which will allow the length of the recorded information to be flexible.

## Record the reader numbers

The following PHP script (which is based on the code from the ‘scrape.php’ script described above) allows us to check the reader numbers from multiple books and write them to our database with a separate entry for each book.

The full code can be found in the code samples bundled with this book (scrape-book.php).

```
<?php

$hostname = 'localhost';
$username = 'pi_insert';
$password = 'xxxxxxxxxx';
$dbname   = 'measurements';

$books = array(
    array('https://leanpub.com/rprogramming',
          'R Programming for Data Science'),
    array('https://leanpub.com/dastyle',
          'The Elements of Data Analytic Style')
);

for ($x = 0; $x <= sizeof($books)-1; $x++) {

    $file_string = file_get_contents_curl($books[$x][0]);
```

```

$regex_pre =
'<ul class=\'book-details-list\'>\n<li class=\'detail\'>\n';
$regex_apre = '\n<p>Readers</p>/s';
$regex_actual = '<span>(.*)</span>';
$regex = $regex_pre.$regex_actual.$regex_apre;

preg_match($regex,$file_string,$title);
$downloads = $title[1];

$link = new PDO("mysql:host=$hostname;dbname=$dbname",
    $username,$password);

$statement = $link->prepare(
    "INSERT INTO downloads(book_name, downloaded) VALUES(?, ?)");

$statement->execute(array($books[$x][1], $downloads));

}

function file_get_contents_curl($url) {
    $ch = curl_init();

    curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
    curl_setopt($ch, CURLOPT_AUTOREFERER, TRUE);
    curl_setopt($ch, CURLOPT_HEADER, 0);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt($ch, CURLOPT_FOLLOWLOCATION, TRUE);

    $data = curl_exec($ch);
    curl_close($ch);

    return $data;
}

?>

```

This script can be saved in our home directory (/home/pi) and can be run by typing:

```
php scrape-books.php
```

While we won't see much happening at the command line, if we use our web browser to go to the phpMyAdmin interface and select the 'measurements' database and then the 'downloads'

table we will see values of readers for the different books and their associated time of recording.

Now you can be forgiven for thinking that this is not going to collect the sort of range of data that will let us ‘Explore’ very much, but let’s do a quick explanation of the PHP script first and then we’ll work out how to record a lot more data :-).

	+ Options	← T →	dtg	downloaded	book_name
<input type="checkbox"/>	Edit  Inline Edit  Copy  Delete	2015-08-03 15:08:57	31175	R Programming for Data Science	
<input type="checkbox"/>	Edit  Inline Edit  Copy  Delete	2015-08-03 15:08:58	32434	The Elements of Data Analytic Style	

Save the MySQL Table Columns

## Code Explanation

An observant reader will notice that this script is essentially a repeat of the ‘[scrape.php](#)’ script with the addition of a few lines to write the associated values to a database. Well done you! As a result, we’ll only need to explain the additional lines.

The script starts by declaring the variables needed to write the values to the database;

```
$hostname = 'localhost';
$username = 'pi_insert';
$password = 'xxxxxxxxxx';
$dbname = 'measurements';
```

Then inside the loop that we use for reading our book names we set up the parameters for connecting to our database;

```
$link = new PDO("mysql:host=$hostname;dbname=$dbname",
    $username,$password);
```

We then prepare the insert statement;

```
$statement = $link->prepare(
    "INSERT INTO downloads(book_name, downloaded) VALUES(?, ?)");
```

... and then we execute the statement;

```
$statement->execute(array($books[$x][1], $downloads));
```

Since these lines are inside the loop going through the array, we capture a unique record of the readers of the book every time the script is run.

## Recording data on a regular basis with cron

As mentioned earlier, while our code is a thing of simplicity and elegance, it only records a single entry for each book every time it is run.

What we need to implement is a schedule so that at a regular time, the program is run. This is achieved using cron via the crontab. While we will cover the requirements for this project here, you can read more about the [crontab](#) in the [Glossary](#).

To set up our schedule we need to edit the crontab file. This is done using the following command;

```
crontab -e
```

Once run it will open the crontab in the nano editor. We want to add in an entry at the end of the file that looks like the following;

```
@daily /usr/bin/php /home/pi/scrape-books.php
```

This instructs the computer that every day it will run the command `/usr/bin/python /home/pi/scrape-books.php` (which if we were at the command line in the pi home directory we would run as `php scrape-books.php`, but since we can't guarantee where we will be when running the script, we are supplying the full path to the `php` command and the `scrape-books.php` script).

Save the file and the next time the day rotates past midnight (the default for '@daily') it will run our program on its designated schedule and we will have reader numbers written to our database every day.

## Explore

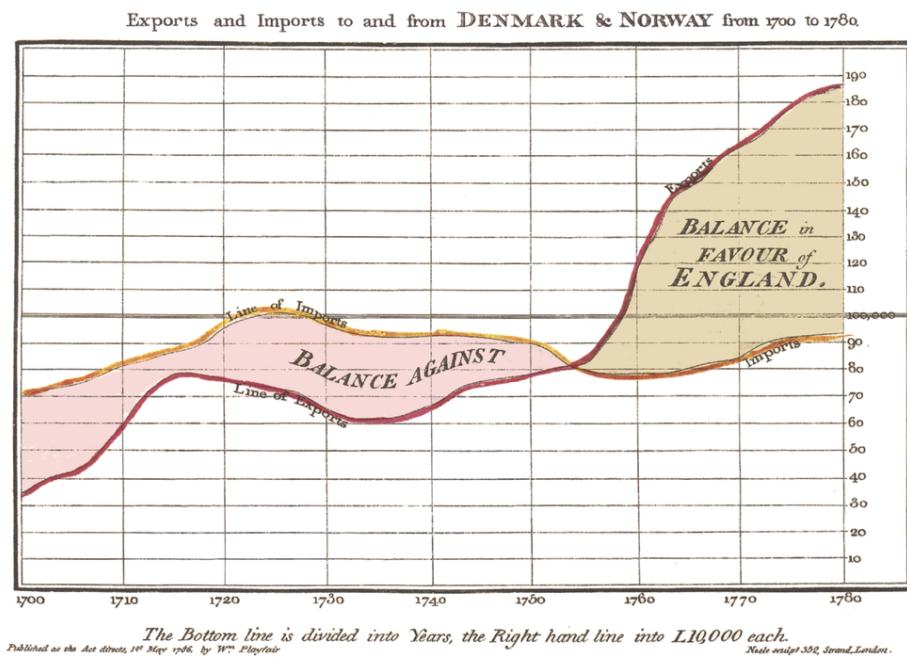
This section presents our data in a difference chart which is a variation on a [bivariate area chart](#)<sup>116</sup>. This is a line chart that includes two lines that are interlinked by filling the space between the lines. A difference chart as demonstrated in the example [here](#)<sup>117</sup> by Mike Bostock is able to highlight the differences between the lines by filling the area between them with different colours depending on which line is the greater value.

As Mike points out in his example, this technique harks back at least as far as [William Playfair](#)<sup>118</sup> when he was describing the time series of exports and imports of Denmark and Norway in 1786.

<sup>116</sup><http://www.informit.com/articles/article.aspx?p=709139&seqNum=5>

<sup>117</sup><http://bl.ocks.org/mbostock/3894205>

<sup>118</sup>[https://en.wikipedia.org/wiki/William\\_Playfair](https://en.wikipedia.org/wiki/William_Playfair)

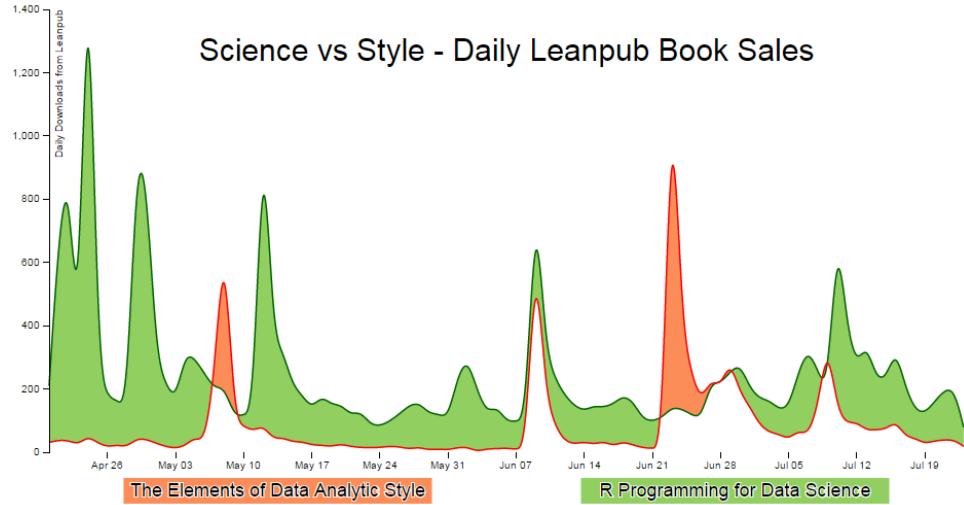


*The Bottom line is divided into Years, the Right hand line into £10,000 each.  
Published as the Act directs, 1st May 1786, by W<sup>m</sup> Playfair  
No. 10, next west of 382, Strand, London.*

### William Playfair's Time Series of Exports and Imports of Denmark and Norway

All that remains is for us to work out how d3.js can help us out by doing the job programmatically. The example that I use here is based on that of [Mike Bostock's](#)<sup>119</sup>. As an bonus extra, we will add of a few niceties in the form of a legend, a title, and some minor changes after explaining the graph.

We will start with a simple example of the code and we will add blocks to finally arrive at the example with Legends and title.



Science vs Style - Daily Leanpub Book Sales

<sup>119</sup><http://bl.ocks.org/mbostock/3894205>

## The Code

The following is the code for the simple difference chart. A live version (that imports a csv file) is available online at [bl.ocks.org<sup>120</sup>](http://bl.ocks.org/d3noob/8beea1d918ff4104f9ab) or [GitHub<sup>121</sup>](https://gist.github.com/d3noob/8beea1d918ff4104f9ab). The version used here is available as the file ‘scrape-diff.php’ as a download with the book Raspberry Pi: Measure, Record, Explore (in a zip file) when you [download the book from Leanpub<sup>122</sup>](#).

```
<?php

$hostname = 'localhost';
$username = 'pi_select';
$password = 'xxxxxxxxxx';

try {
    $dbh = new PDO("mysql:host=$hostname;dbname=measurements",
                   $username, $password);

    /*** The SQL SELECT statement ***/

    $sth = $dbh->prepare("
SELECT *
FROM   downloads
ORDER BY dtg, book_name
");
    $sth->execute();

    /* Fetch all of the remaining rows in the result set */
    $result = $sth->fetchAll(PDO::FETCH_ASSOC);

    /*** close the database connection ***/
    $dbh = null;

}

catch(PDOException $e)
{
    echo $e->getMessage();
}

$json_data = json_encode($result);

?>
```

---

<sup>120</sup><http://bl.ocks.org/d3noob/8beea1d918ff4104f9ab>

<sup>121</sup><https://gist.github.com/d3noob/8beea1d918ff4104f9ab>

<sup>122</sup><https://leanpub.com/RPiMRE>

```
<!DOCTYPE html>
<meta charset="utf-8">
<style>

body { font: 10px sans-serif; }

text.shadow {
  stroke: white;
  stroke-width: 2px;
  opacity: 0.9;
}

.axis path,
.axis line {
  fill: none;
  stroke: #000;
  shape-rendering: crispEdges;
}

.x.axis path { display: none; }

.area.above { fill: rgb(252,141,89); }
.area.below { fill: rgb(145,207,96); }

.line {
  fill: none;
  stroke: #000;
  stroke-width: 1.5px;
}

</style>
<body>
<script src="http://d3js.org/d3.v3.min.js"></script>
<script>

var title = "Science vs Style - Daily Leanpub Book Sales";

var margin = {top: 20, right: 20, bottom: 50, left: 50},
    width = 960 - margin.left - margin.right,
    height = 500 - margin.top - margin.bottom;

var parsedtg = d3.time.format("%Y-%m-%d %H:%M:%S").parse;

var x = d3.time.scale().range([0, width]);
var y = d3.scale.linear().range([height, 0]);
```

```
var xAxis = d3.svg.axis().scale(x).orient("bottom");
var yAxis = d3.svg.axis().scale(y).orient("left");

var lineScience = d3.svg.area()
  .interpolate("basis")
  .x(function(d) { return x(d.dtg); })
  .y(function(d) { return y(d["Science"]); });

var lineStyle = d3.svg.area()
  .interpolate("basis")
  .x(function(d) { return x(d.dtg); })
  .y(function(d) { return y(d["Style"]); });

var area = d3.svg.area()
  .interpolate("basis")
  .x(function(d) { return x(d.dtg); })
  .y1(function(d) { return y(d["Science"]); });

var svg = d3.select("body").append("svg")
  .attr("width", width + margin.left + margin.right)
  .attr("height", height + margin.top + margin.bottom)
.append("g")
  .attr("transform",
    "translate(" + margin.left + "," + margin.top + ")");

<?php echo "dataNest=".json_data." ?>

dataNest.forEach(function(d) {
  d.dtg = parsedtg(d.dtg);
  d.downloaded = +d.downloaded;
});

var data = d3.nest()
  .key(function(d) {return d.dtg;})
  .entries(dataNest);

data.forEach(function(d) {
  d.dtg = d.values[0]['dtg'];
  d["Science"] = d.values[0]['downloaded'];
  d["Style"] = d.values[1]['downloaded'];
});

for(i=data.length-1;i>0;i--) {
  data[i].Science = data[i].Science -data[(i-1)].Science ;
  data[i].Style = data[i].Style -data[(i-1)].Style ;
}
```

```
data.shift(); // Removes the first element in the array

x.domain(d3.extent(data, function(d) { return d.dtg; }));
y.domain([
//    d3.min(data, function(d) {
//        return Math.min(d["Science"], d["Style"]); }),
//    d3.max(data, function(d) {
//        return Math.max(d["Science"], d["Style"]); })
0,1400
]);
svg.datum(data);

svg.append("clipPath")
    .attr("id", "clip-above")
.append("path")
    .attr("d", area.y0(0));

svg.append("clipPath")
    .attr("id", "clip-below")
.append("path")
    .attr("d", area.y0(height));

svg.append("path")
    .attr("class", "area above")
    .attr("clip-path", "url(#clip-above)")
    .attr("d", area.y0(function(d) { return y(d["Style"]); })); 

svg.append("path")
    .attr("class", "area below")
    .attr("clip-path", "url(#clip-below)")
    .attr("d", area.y0(function(d) { return y(d["Style"]); })); 

svg.append("path")
    .attr("class", "line")
    .style("stroke", "darkgreen")
    .attr("d", lineScience);

svg.append("path")
    .attr("class", "line")
    .style("stroke", "red")
    .attr("d", lineStyle);

svg.append("g")
    .attr("class", "x axis")
```

```

    .attr("transform", "translate(0," + height + ")")
    .call(xAxis);

  svg.append("g")
    .attr("class", "y axis")
    .call(yAxis);

</script>
</body>

```

## Description

The graph has some portions that are common to the simple graphs that have been worked through in the earlier chapters.

The PHP block at the start of the code is mostly the same as our example code for our single temperature measurement project. The significant difference however is in the select statement.

```

SELECT *
FROM downloads
ORDER BY dtg, book_name

```

Here, all the data is to be used (hence the \*) and we order by date time group and book name.

We start the HTML file, load some styling for the upcoming elements, set up the margins, time formatting scales, ranges and axes.

Because the graph is composed of two lines we need to declare two separate line functions;

```

var lineScience = d3.svg.area()
  .interpolate("basis")
  .x(function(d) { return x(d.dtg); })
  .y(function(d) { return y(d["Science"]); });

var lineStyle = d3.svg.area()
  .interpolate("basis")
  .x(function(d) { return x(d.dtg); })
  .y(function(d) { return y(d["Style"]); });

```

To fill an area we declare an area function using one of the lines as the baseline ( $y_1$ ) and when it comes time to fill the area later in the script we declare  $y_0$  separately to define the area to be filled as an intersection of two paths.

```
var area = d3.svg.area()
  .interpolate("basis")
  .x(function(d) { return x(d.dtg); })
  .y1(function(d) { return y(d["Science"]); });
```

In this instance we are using the green ‘Science’ line as the y1 line.

The svg area is then set up using the height, width and margin values and we load our csv files with our number of downloads for each book. We then carry out a standard `forEach` to ensure that the time and numerical values are formatted correctly.

## Nesting the data

The data that we are starting with is stored and formatted in a way that we could reasonably expect data to be available in this instance. This mechanism makes it easier to add additional books for recording if desired.

dtg	downloaded	book_name
2015-04-19 00:00:00	5481	R Programming for Data Science
2015-04-19 00:00:00	23751	The Elements of Data Analytic Style
2015-04-20 00:00:00	5691	R Programming for Data Science
2015-04-20 00:00:00	23782	The Elements of Data Analytic Style
2015-04-21 00:00:00	6379	R Programming for Data Science
2015-04-21 00:00:00	23820	The Elements of Data Analytic Style
2015-04-22 00:00:00	7281	R Programming for Data Science
2015-04-22 00:00:00	23857	The Elements of Data Analytic Style
2015-04-23 00:00:00	7554	R Programming for Data Science
2015-04-23 00:00:00	23881	The Elements of Data Analytic Style
2015-04-24 00:00:00	9331	R Programming for Data Science
2015-04-24 00:00:00	23932	The Elements of Data Analytic Style
2015-04-25 00:00:00	9614	R Programming for Data Science

### Data storage

In this case, we will need to ‘pivot’ the data to produce a multi-column representation where we have a single row for each date, and the number of downloads for each book as separate columns as follows;

dtg	book 1	book 2
2015-04-19	5481	23751
2015-04-20	5691	23782
2015-04-21	6379	23820

This can be achieved using the `d3 nest` function.

```
var data = d3.nest()
  .key(function(d) {return d.dtg;})
  .entries(dataNest);
```

We declare our new array's name as data and we initiate the nest function;

```
var data = d3.nest()
```

We assign the key for our new array as dtg. A 'key' is like a way of saying "*This is the thing we will be grouping on*". In other words our resultant array will have a single entry for each unique date (dtg) which will have the values of the number of downloaded books associated with it.

```
  .key(function(d) {return d.dtg;})
```

Then we tell the nest function which data array we will be using for our source of data.

```
}).entries(dataNest);
```

## Wrangle the data

Once we have our pivoted data we can format it in a way that will suit the code for the visualisation. This involves storing the values for the 'Science' and 'Style' variables as part of a named index.

```
data.forEach(function(d) {
  d.dtg = d.values[0]['dtg'];
  d["Science"] = d.values[0]['downloaded'];
  d["Style"] = d.values[1]['downloaded'];
});
```

We then loop through the 'Science' and 'Style' array to convert the incrementing value of the total number of downloads into a value of the number that have been downloaded each day;

```
for(i=data.length-1;i>0;i--) {
  data[i].Science = data[i].Science - data[(i-1)].Science ;
  data[i].Style = data[i].Style - data[(i-1)].Style ;
}
```

Finally because we are adjusting from total downloaded to daily values we are left with an orphan value that we need to remove from the front of the array;

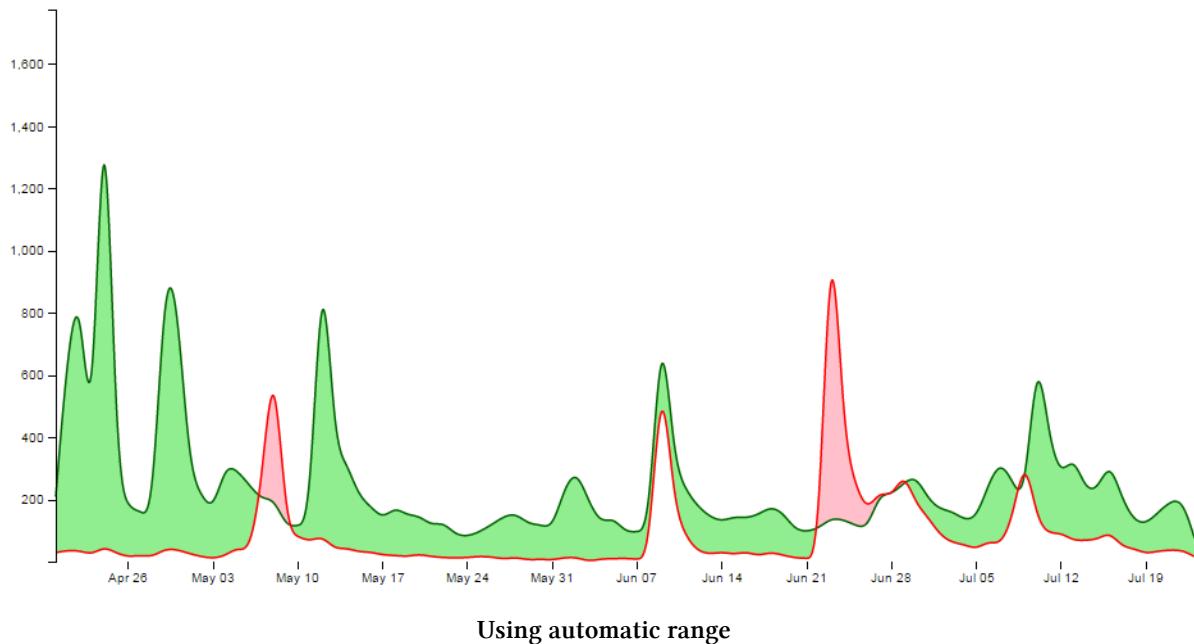
```
data.shift();
```

## Cheating with the domain

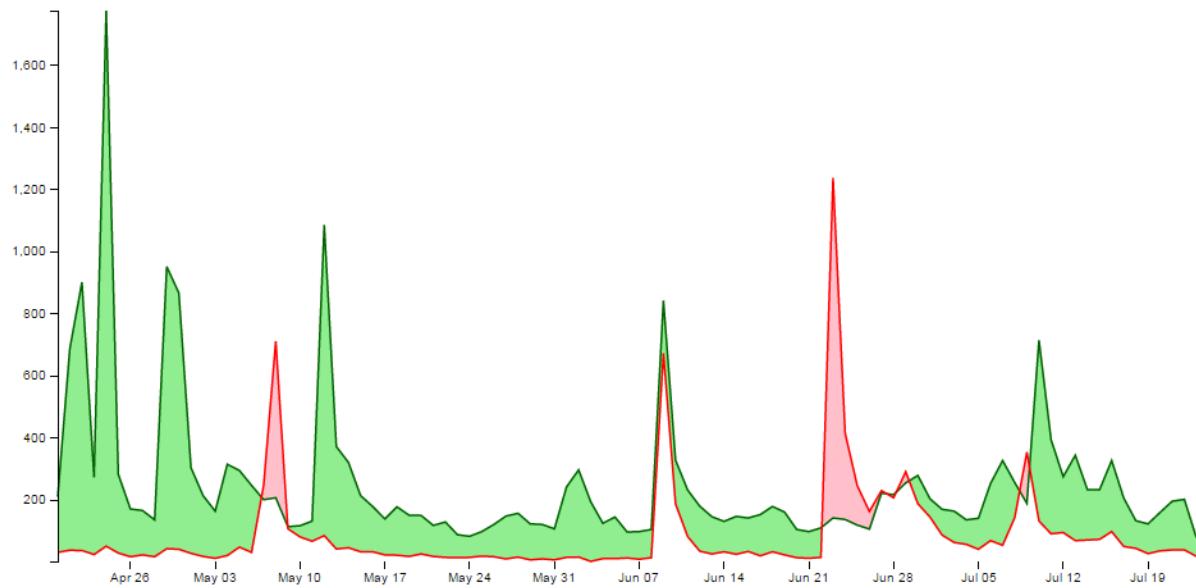
The observant d3.js reader will have noticed that the setting of the y domain has a large section commented out;

```
x.domain(d3.extent(data, function(d) { return d.dtg; }));
y.domain([
//    d3.min(data, function(d) {
//        return Math.min(d["Science"], d["Style"]); }),
//    d3.max(data, function(d) {
//        return Math.max(d["Science"], d["Style"]); })
0,1400
]);
```

That's because I want to be able to provide an ideal way for the graph to represent the data in an appropriate range, but because we are using the `basis smoothing` modifier, and the data is ‘peaky’, there is a tendency for the y scale to be fairly broad and the resultant graph looks a little lost;



Alternatively, we could remove the smoothing and let the true data be shown;



Using automatic range and removing the basis smoothing

It should be argued that this is a truer representation of the data, but in this case I feel comfortable sacrificing accuracy for aesthetics (what have I become?).

Therefore, the domain for the y axis is set manually to between 0 and 1400, but feel free to remove that at the point when you introduce your own data :-).

### **data VS datum**

One small line gets its own section. That line is;

```
svg.datum(data);
```

A casual d3.js user could be forgiven for thinking that this doesn't seem too fearsome a line, but it has hidden depths.

As Mike Bostock explains [here<sup>123</sup>](#), if we want to bind data to elements as a group we would be \*.data, but if we want to bind that data to individual elements, we should use \*.datum.

It's a function of how the data is stored. If there is an expectation that the data will be dynamic then data is the way to go since it has the feature of preparing enter and exit selections. If the data is static (it won't be changing) then datum is the way to go.

In our case we are assigning data to individual elements and as a result we will be using datum.

### **Setting up the clipPaths**

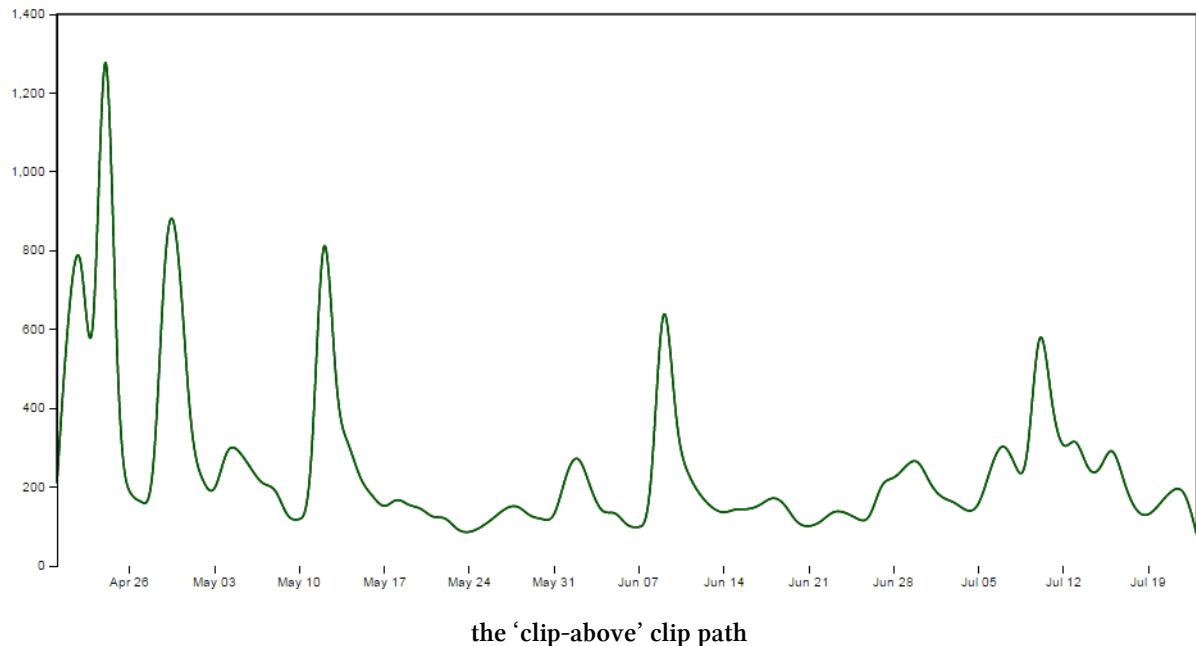
The `clipPath124` operator is used to define an area that is used to create a shape by intersecting one area with another.

---

<sup>123</sup><http://bost.ocks.org/mike/selection/#data>

<sup>124</sup><https://developer.mozilla.org/en-US/docs/Web/SVG/Element/clipPath>

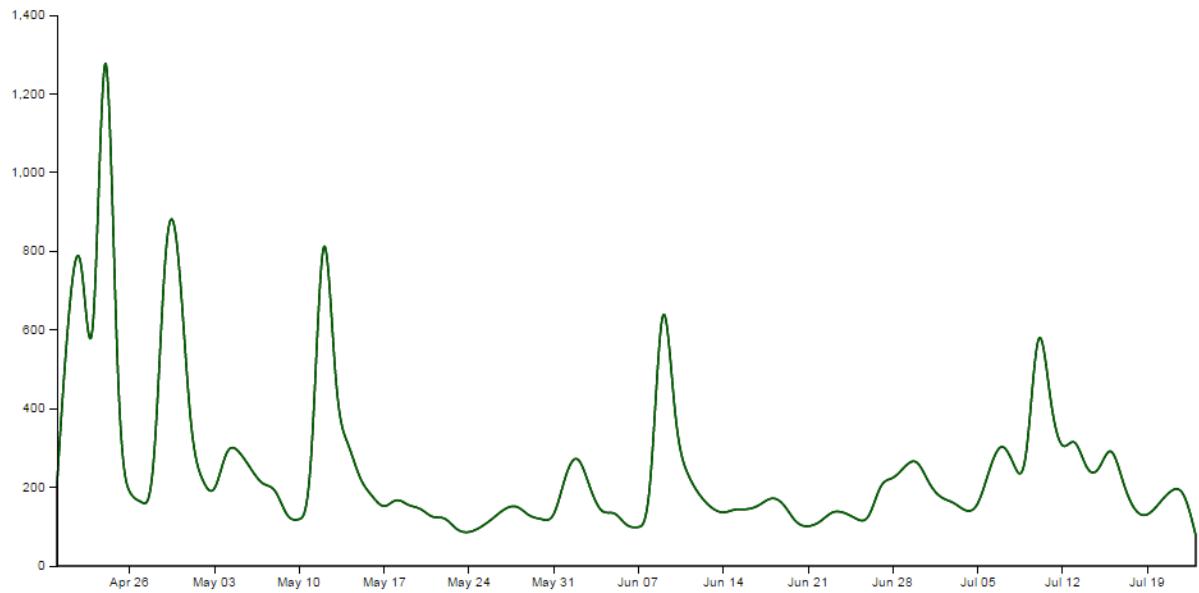
In our case we are going to set up two clip paths. One is the area above the green ‘Science’ line (which we defined earlier as being the `y1` component of an area selection);



This is declared via this portion of the code;

```
svg.append("clipPath")
  .attr("id", "clip-above")
  .append("path")
  .attr("d", area.y0(0));
```

Then we set up the clip path that will exist for the area below the green ‘Science’ line ;



This is declared via this portion of the code;

```
svg.append("clipPath")
    .attr("id", "clip-below")
    .append("path")
    .attr("d", area.y0(height));
```

Each of these paths has an 'id' which can be subsequently used by the following code.

## Clipping and adding the areas

Now we come to clipping our shape and filling it with the appropriate colour.

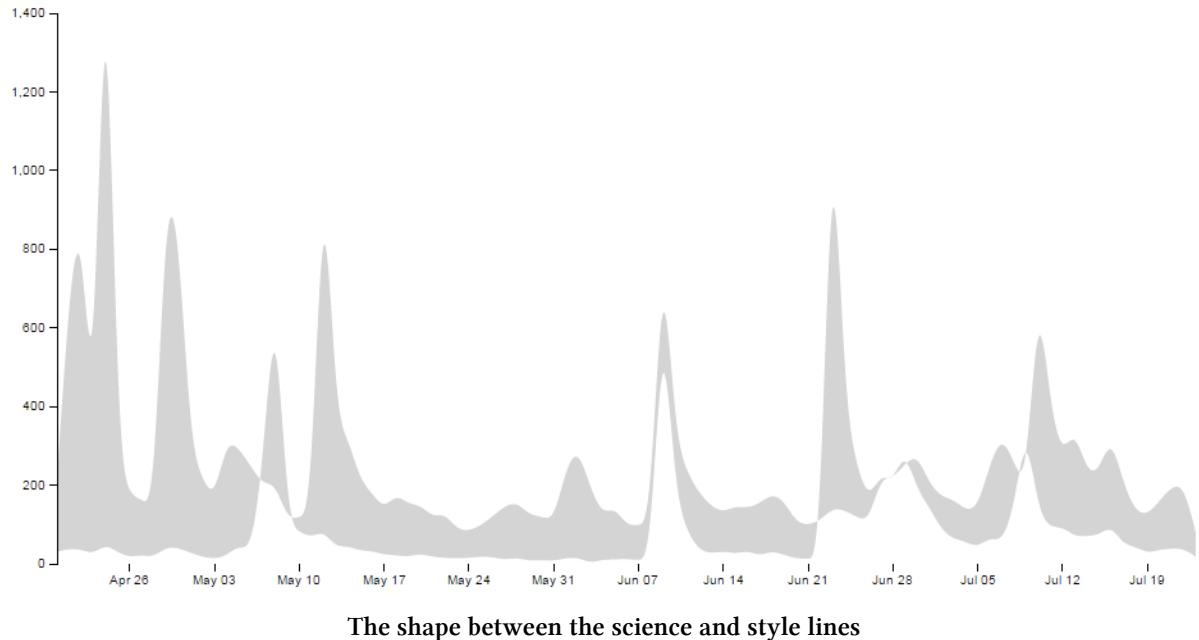
We do this by having a shape that represents the area between the two lines and applying our clip path for the values above and below our reference line (the green 'Science' line). Where the two intersect, we fill it with the appropriate colour. The code to fill the area above the reference line is as follows;

```
svg.append("path")
    .attr("class", "area_above")
    .attr("clip-path", "url(#clip-above)")
    .attr("d", area.y0(function(d) { return y(d["Style"]); }));
```

Here we have two lines that are defining the shape between the two science and style lines;

```
svg.append("path")
...
...
.attr("d", area.y0(function(d) { return y(d["Style"]); }));
```

If we were to look at the shape that this produces it would look as follows (greyed out for highlighting);



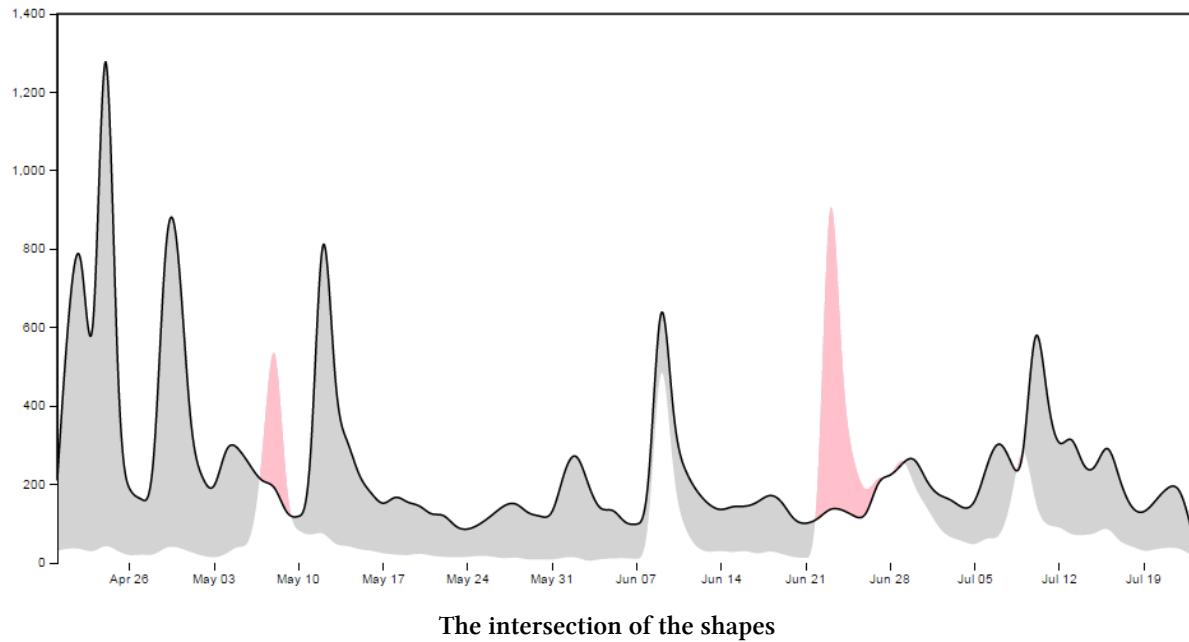
We apply a class to the shape so that is filled with the colour that we want;

```
.attr("class", "area_above")
```

.. and apply the clip path so that only the areas that intersect the two shapes are filled with the appropriate colour;

```
.attr("clip-path", "url(#clip-above)")
```

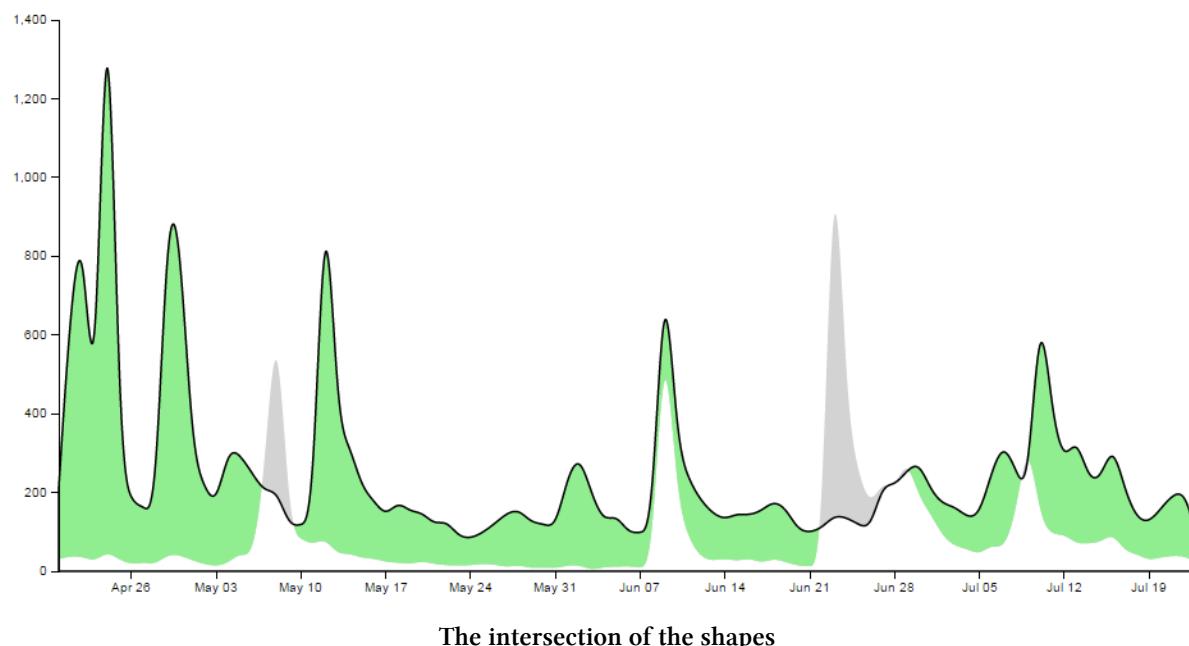
Here the intersection of those two shapes is shown as pink;



Then we do the same for the area below;

```
svg.append("path")
  .attr("class", "area below")
  .attr("clip-path", "url(#clip-below)")
  .attr("d", area.y0(function(d) { return y(d["Style"]); }));
```

With the corresponding areas showing the intersection of the two shapes coloured differently;



## Draw the lines and the axes

The final part of our basic difference chart is to draw in the lines over the top so that they are highlighted and to add in the axes;

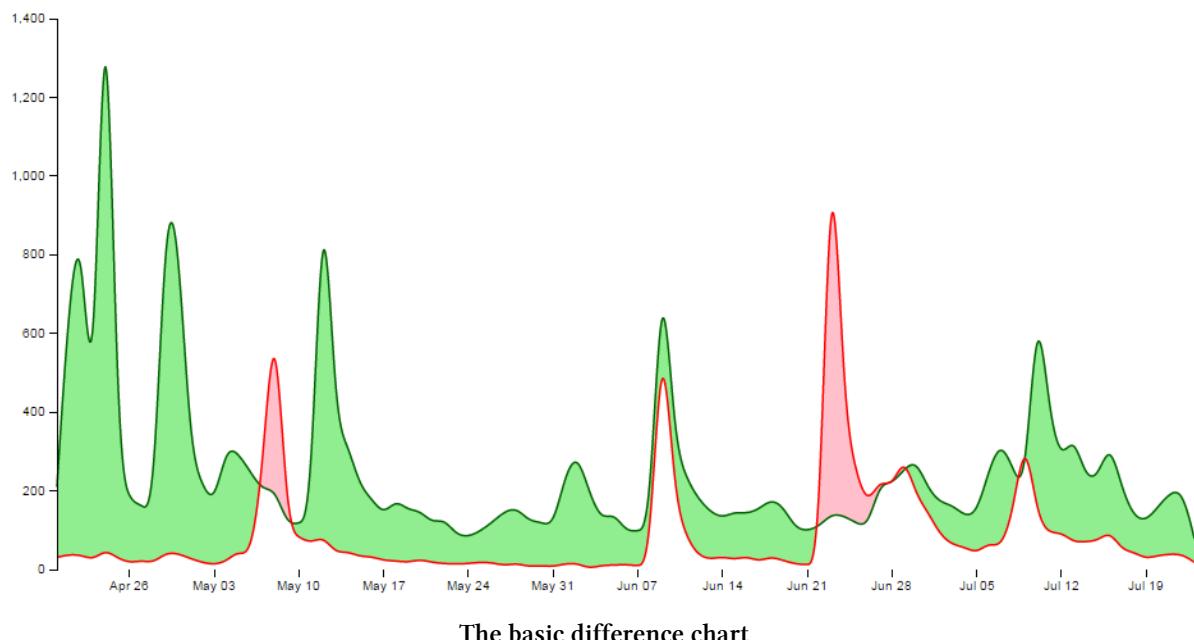
```
svg.append("path")
  .attr("class", "line")
  .style("stroke", "darkgreen")
  .attr("d", lineScience);

svg.append("path")
  .attr("class", "line")
  .style("stroke", "red")
  .attr("d", lineStyle);

svg.append("g")
  .attr("class", "x_axis")
  .attr("transform", "translate(0," + height + ")")
  .call(xAxis);

svg.append("g")
  .attr("class", "y_axis")
  .call(yAxis);
```

Et viola! we have our difference chart!



As mentioned earlier, the code for the simple difference chart using a csv data file is available online at [bl.ocks.org<sup>125</sup>](http://bl.ocks.org/d3noob/8beea1d918ff4104f9ab) or [GitHub<sup>126</sup>](https://gist.github.com/d3noob/8beea1d918ff4104f9ab). It is also available as the file ‘scrape-diff.php’ as a download

<sup>125</sup><http://bl.ocks.org/d3noob/8beea1d918ff4104f9ab>

<sup>126</sup><https://gist.github.com/d3noob/8beea1d918ff4104f9ab>

with the book (in a zip file) when you download the book from Leanpub<sup>127</sup>.

## Adding a bit more to our difference chart.

The chart itself is a thing of beauty, but given the subject matter (it's describing two books after all) we should include a bit more information on what it is we're looking at and provide some links so that a fascinated viewer of the graphs can read the books!

### Add a Y axis label

Because it's not immediately obvious what we're looking at on the Y axis we should add in a nice subtle label on the Y axis;

```
svg.append("g")
  .attr("class", "y_axis")
  .call(yAxis)
  .append("text")
    .attr("transform", "rotate(-90)")
    .attr("y", 6)
    .attr("dy", ".71em")
    .style("text-anchor", "end")
    .text("Daily Downloads from Leanpub");
```

### Add a title

Every graph should have a title. The following code adds this to the top(ish) centre of the chart and provides a white drop-shadow for readability;

```
// ***** Title Block *****
svg.append("text") // Title shadow
  .attr("x", (width / 2))
  .attr("y", 50)
  .attr("text-anchor", "middle")
  .style("font-size", "30px")
  .attr("class", "shadow")
  .text(title);

svg.append("text") // Title
  .attr("x", (width / 2))
  .attr("y", 50)
  .attr("text-anchor", "middle")
  .style("font-size", "30px")
  .style("stroke", "none")
  .text(title);
```

---

<sup>127</sup><https://leanpub.com/RPiMRE>

## Adding the legend

A respectable legend in this case should provide visual context of what it is describing in relation to the graph (by way of colour) and should actually name the book. We can also go a little bit further and provide a link to the books in the legend so that potential readers can access them easily.

Firstly the rectangles filled with the right colour, sized appropriately and arranged just right;

```
var block = 300; // rectangle width and position

svg.append("rect") // Style Legend Rectangle
    .attr("x", ((width / 2)/2)-(block/2))
    .attr("y", height+(margin.bottom/2) )
    .attr("width", block)
    .attr("height", "25")
    .attr("class", "area above");

svg.append("rect") // Science Legend Rectangle
    .attr("x", ((width / 2)/2)+(width / 2)-(block/2))
    .attr("y", height+(margin.bottom/2) )
    .attr("width", block)
    .attr("height", "25")
    .attr("class", "area below");
```

Then we add the text (with a drop-shadow) and a link;

```
svg.append("text") // Style Legend Text shadow
    .attr("x", ((width / 2)/2))
    .attr("y", height+(margin.bottom/2) + 5)
    .attr("dy", ".71em")
    .attr("text-anchor", "middle")
    .style("font-size", "18px")
    .attr("class", "shadow")
    .text("The Elements of Data Analytic Style");

svg.append("text") // Science Legend Text shadow
    .attr("x", ((width / 2)/2)+(width / 2))
    .attr("y", height+(margin.bottom/2) + 5)
    .attr("dy", ".71em")
    .attr("text-anchor", "middle")
    .style("font-size", "18px")
    .attr("class", "shadow")
    .text("R Programming for Data Science");

svg.append("a")
```

```

.attr("xlink:href", "https://leanpub.com/dastyle")
.append("text") // Style Legend Text
.attr("x", ((width / 2)/2))
.attr("y", height+(margin.bottom/2) + 5)
.attr("dy", ".71em")
.attr("text-anchor", "middle")
.style("font-size", "18px")
.style("stroke", "none")
.text("The Elements of Data Analytic Style");

svg.append("a")
.attr("xlink:href", "https://leanpub.com/rprogramming")
.append("text") // Science Legend Text
.attr("x", ((width / 2)/2)+(width / 2))
.attr("y", height+(margin.bottom/2) + 5)
.attr("dy", ".71em")
.attr("text-anchor", "middle")
.style("font-size", "18px")
.style("stroke", "none")
.text("R Programming for Data Science");

```

I'll be the first to admit that this could be done more efficiently with some styling via css, but then it would leave nothing for the reader to try :-).

## Link the areas

As a last touch we can include the links to the respective books in the shading for the graph itself;

```

svg.append("a")
.attr("xlink:href", "https://leanpub.com/dastyle")
.append("path")
.attr("class", "area above")
.attr("clip-path", "url(#clip-above)")
.attr("d", area.y0(function(d) { return y(d["Style"]); }));

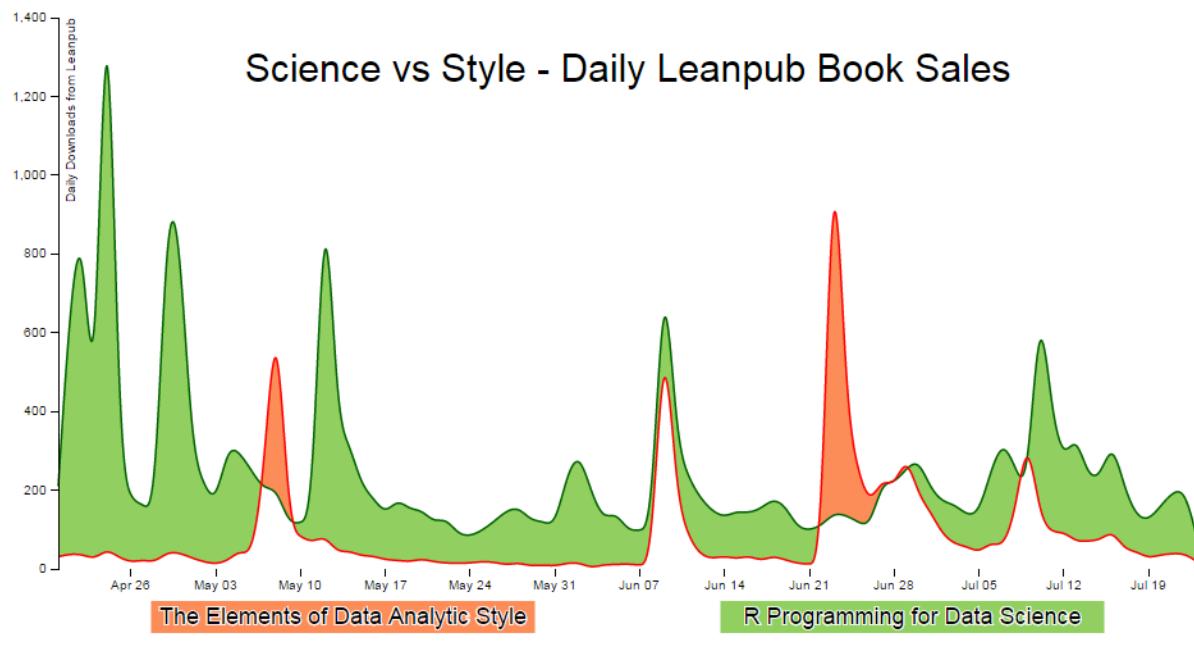
svg.append("a")
.attr("xlink:href", "https://leanpub.com/rprogramming")
.append("path")
.attr("class", "area below")
.attr("clip-path", "url(#clip-below)")
.attr("d", area.y0(function(d) { return y(d["Style"]); }));

```

Perhaps not strictly required, but a nice touch none the less.

## The final result

And here it is;



The code for the full difference chart using a csv file for data is available online at [bl.ocks.org<sup>128</sup>](http://bl.ocks.org/d3noob/f6d8588a0aa3a7503f57) or [GitHub<sup>129</sup>](https://gist.github.com/d3noob/f6d8588a0aa3a7503f57). It is also available as the file ‘scrape-diff-full.php’ as a download with the book (in a zip file) when you [download the book from Leanpub<sup>130</sup>](#).

<sup>128</sup><http://bl.ocks.org/d3noob/f6d8588a0aa3a7503f57>

<sup>129</sup><https://gist.github.com/d3noob/f6d8588a0aa3a7503f57>

<sup>130</sup><https://leanpub.com/RPiMRE>

# Raspberry Pi Tips and Tricks

## Changing the default keyboard layout

By default the Raspbian operating system comes configured to recognise and use a keyboard with a Great Britain (GB) character set. If we want to change the default keyboard to something else (for example a US keyboard) we will need to edit the keyboard configuration file.

So, from the command line execute the following command;

```
sudo nano /etc/default/keyboard
```

This will present the keyboard configuration file which will look something like the following;

```
# KEYBOARD CONFIGURATION FILE

# Consult the keyboard(5) manual page.

XKBMODEL="pc105"
XKBLAYOUT="gb"
XKBVARIANT=""
XKBOPTIONS=""

BACKSPACE="guess"
```

Edit the file to implement the United States character keymapping by changing the file to the following;

```
# KEYBOARD CONFIGURATION FILE

# Consult the keyboard(5) manual page.

XKBMODEL="pc105"
XKBLAYOUT="us"
XKBVARIANT=""
XKBOPTIONS=""

BACKSPACE="guess"
```

Then we need to reboot the Pi to let the changes take effect;

```
sudo reboot
```

## Changing the default local time

By default Raspbian is configured to use [UTC<sup>131</sup>](#) as its local time. This can be reconfigured by executing the following command from the terminal;

```
sudo dpkg-reconfigure tzdata
```

You will be asked to pick the continent or ocean of the time zone you are in and then the specific country.

Once your selection is complete, reboot to allow the changes to take effect;

```
sudo reboot
```

Then we can check what time the Raspberry Pi thinks it is with the date command;

```
date
```

The response should be something like;

```
Sun Dec 28 16:47:16 NZDT 2014
```

(if you have set your local time to be New Zealand time (who wouldn't?)).

---

<sup>131</sup>[http://en.wikipedia.org/wiki/Coordinated\\_Universal\\_Time](http://en.wikipedia.org/wiki/Coordinated_Universal_Time)

## Access the Pi with a 'name' or an IP address

Very few people will be used to accessing a web page by typing in the IP address of the server which they are trying to connect to. We can add a host name to the desktop computer we use to access the Pi so that we can use its name instead of the IP address.

Many thanks to the reddit<sup>132</sup> user ‘asterisk\_man’ for suggesting that it be included<sup>133</sup> and for providing the details on how to get it done.

Making the assumption that we are accessing the Pi from a Windows machine and I have only tested this on Windows 7 (if there are significant differences on other flavours, please let me know what needs to be added). You need to have administrator permissions to carry this out, so if you haven’t got them, you aren’t going to be able to edit the following file. Browse to the C:\Windows\System32\drivers\etc directory. There we should find a file called hosts. It should probably look a little like the following;

```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97    rhino.acme.com          # source server
#      38.25.63.10      x.acme.com            # x client host

# localhost name resolution is handled within DNS itself.
#      127.0.0.1        localhost
#      ::1              localhost

127.0.0.1      localhost
```

As the instructions in the file say, we want to add in the IP address of our Raspberry Pi and the name that we are going to give it.

In the case of the example I will use, the IP address will be 10.1.1.8 and I will pick a Top Level Domain (TLD) name that should not be resolvable in the outside world (that could get confusing). Taking ‘asterisk\_man’s example I will call it ‘pi’ and specify a TLD of ‘.local’.

<sup>132</sup><http://www.reddit.com/>

<sup>133</sup>[http://www.reddit.com/r/raspberry\\_pi/comments/2y04zr/set\\_a\\_static\\_ip\\_address\\_on\\_your\\_raspberry\\_pi/](http://www.reddit.com/r/raspberry_pi/comments/2y04zr/set_a_static_ip_address_on_your_raspberry_pi/)



The ‘.local’ domain, has been reserved as a Special Use Domain Name (SUDN) specifically for the purpose of internal network usage. It will never be configured as a Fully Qualified Domain Name (FQDN) therefore your custom local names should never conflict with external addresses.

The hosts file should therefore look like;

```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97      rhino.acme.com          # source server
#      38.25.63.10      x.acme.com               # x client host

# localhost name resolution is handled within DNS itself.
#      127.0.0.1      localhost
#      ::1            localhost

127.0.0.1      localhost
10.1.1.8      pi.local
```

Now you can type ‘`http://pi.local`’ into your web browser and you should see the default web page for your Pi!



## It works!

This is the default web page for this server.

The web server software is running but no content has been added, yet.

Browsing to the Pi’s web page with a host name

If you’re wanting to do the equivalent from a Linux box, the hosts file is in the ‘/etc’ directory. Apple users might want to consider using the ‘Bonjour’ service, but as I’m not an apple guy I will have to leave you to work that out on your own.

Many thanks asterisk\_man.

## Transfer files easily to / from the Pi

When working on simple files on the Pi it's easy enough to be able to copy the code you're working on from a desktop machine and paste it into a file that we're editing on the Pi. However, it could be argued that this is not a simple mechanism and it should be a simple job to use a GUI to accomplish the same task.

This is possible using programs that will allow access via SSH File Transfer Protocol or SFTP.



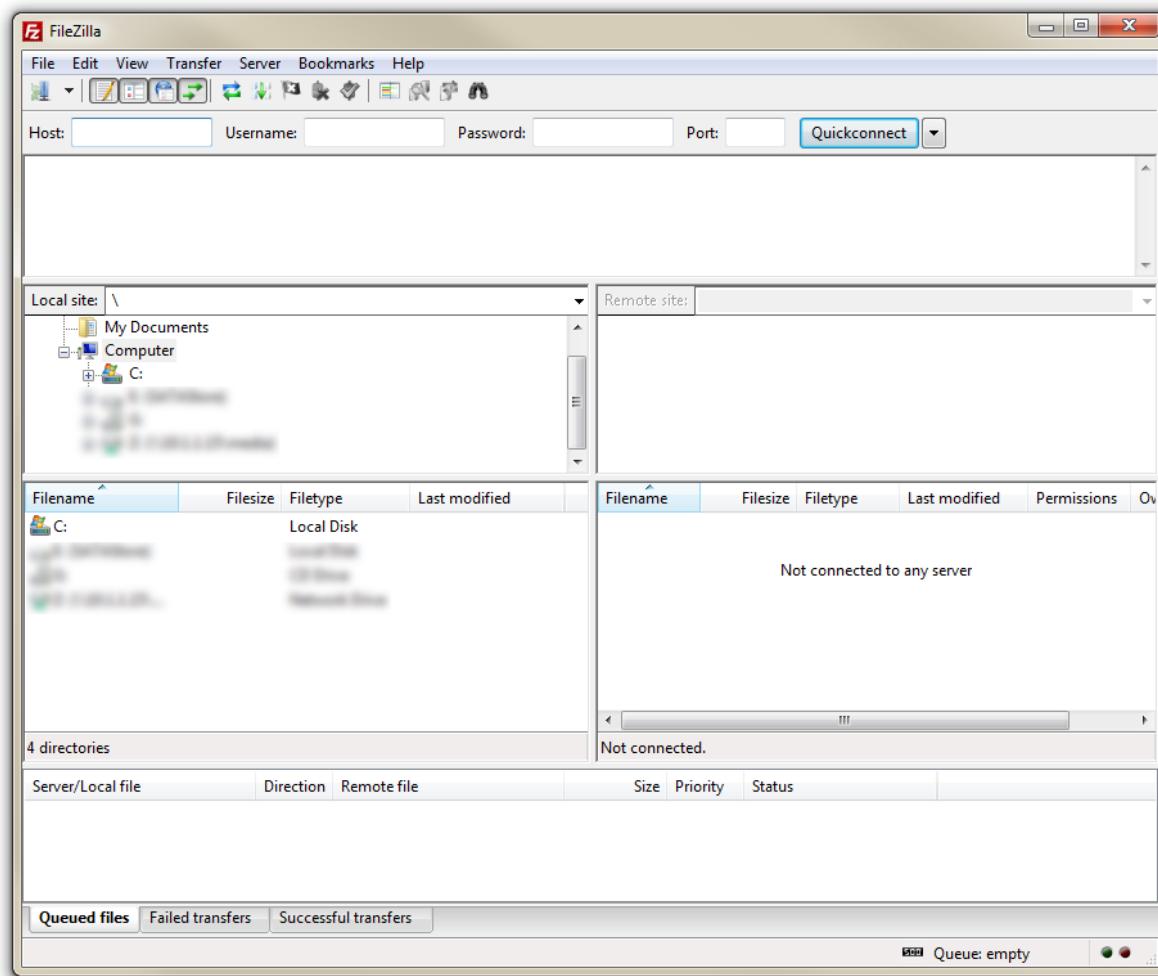
'SSH' stands for Secure Shell and while it is possible to use FTP without SSH, it's easy enough to use SFTP, so there's really no excuse to not use a secure mechanism.

The program that we'll use to make this all happen is 'FileZilla'. It can be downloaded from the site '<https://filezilla-project.org/>'. It is Open Source software distributed free of charge, so there's a lot to like about the option.

Since we already have SSH running on the Pi (it does so by default in Raspbian) the only software we need to install is the client software on our desktop.

From the FileZilla site, download and install the client software appropriate for your desktop (the example I will be using here will be for Windows 7 64 bit). As much as I appreciate that Open Source projects need support to keep themselves operating, I recommend that you download the zipped software and not the installer, since the installer requires you to avoid installing additional software that you might not necessarily want.

Once installed, you should be presented with the main interface that looks like this;

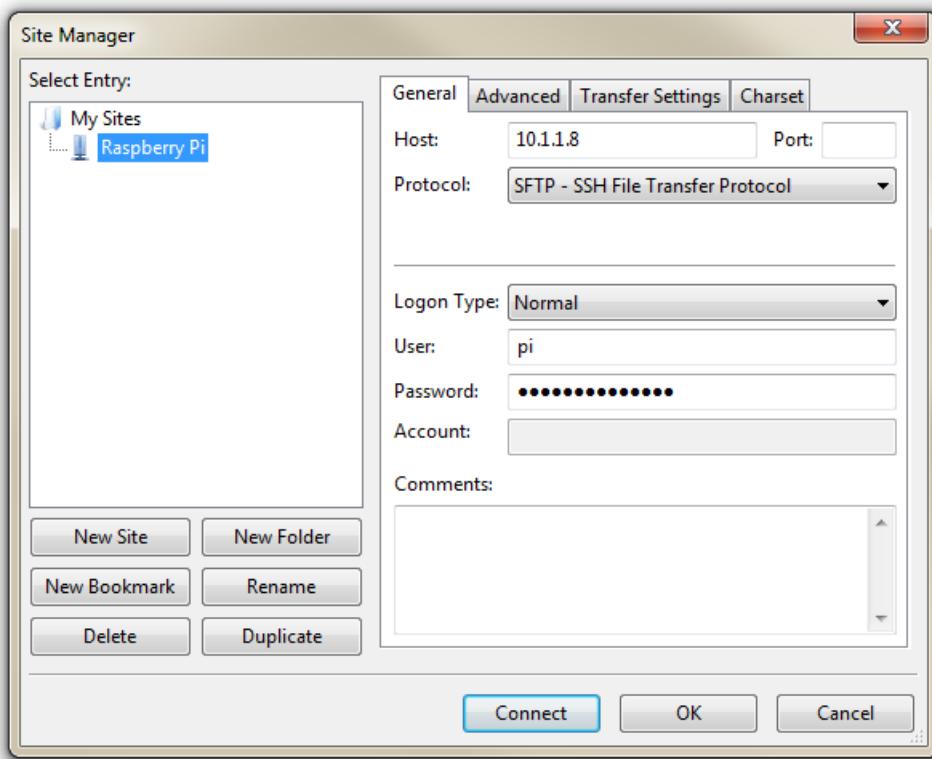


FileZilla Starting Interface

Now select File and then Site Manager from the main menus to start to set up our connection.

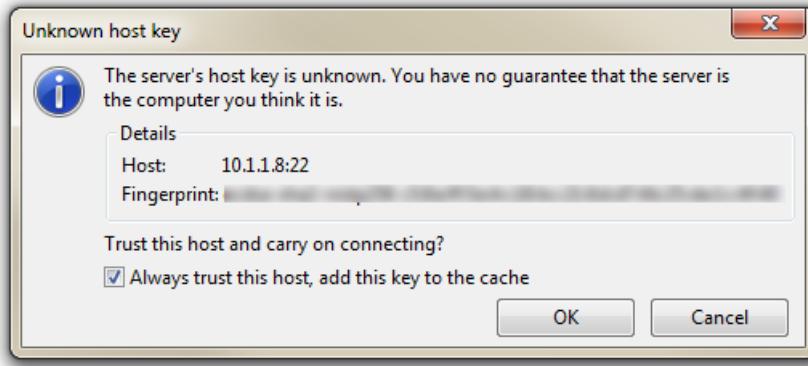
We will now see a dialogue box that will allow us to enter the connection details for our Raspberry Pi. Click on the 'New Site' button and we can fill in the following;

- In the 'Host' entry box type in the IP address of our Raspberry Pi.
- Change 'Protocol' to 'SFTP SSH File Transfer Protocol'.
- Change 'Logon Type' to Normal
- Enter our login name in the 'User' box (showing the default 'pi' user here).
- Type in the users password into the 'Password' box.
- If you want, click on 'Rename' to change the name of the connection to make it easier to understand when you have multiple connections.



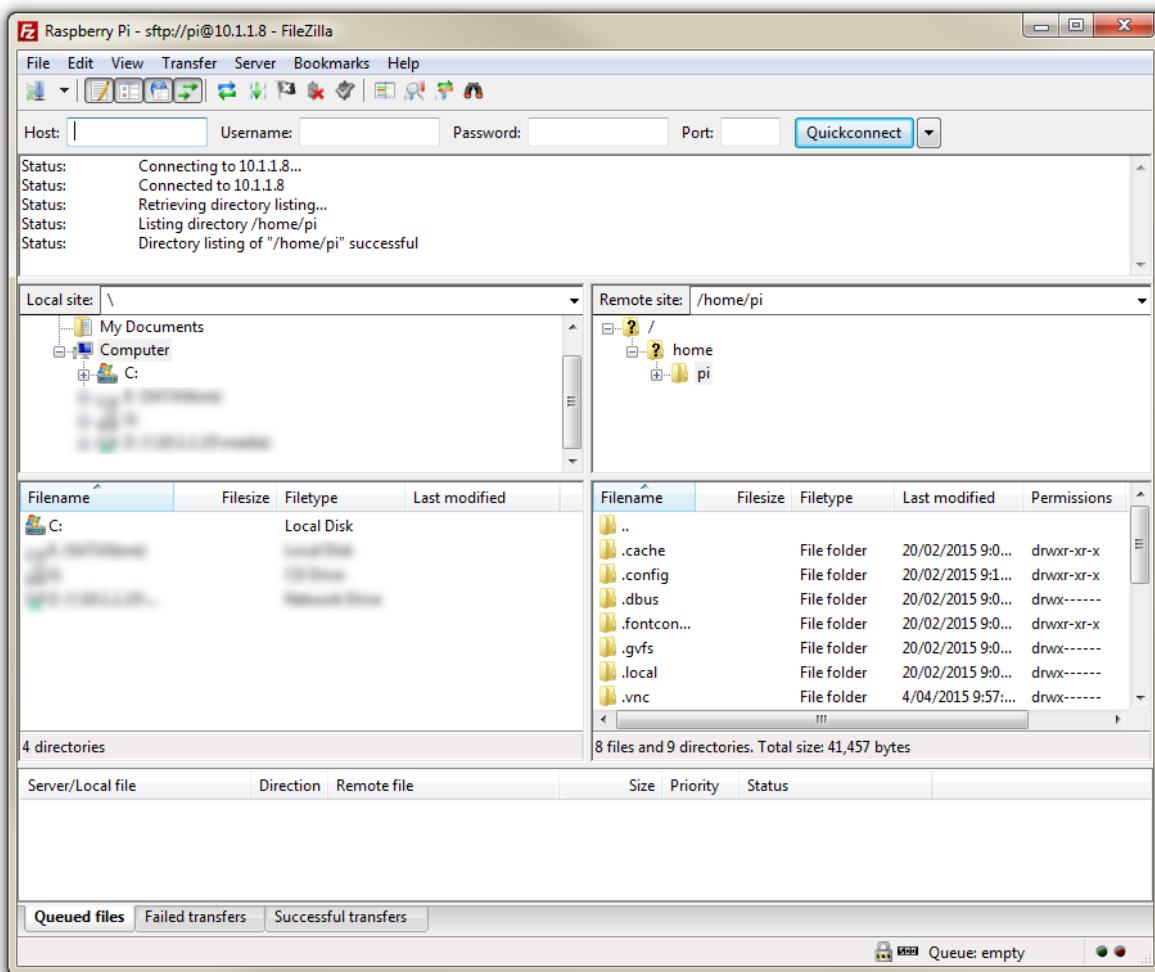
FileZilla New Site Dialogue Box Details

Once complete we can click on 'Connect' to initiate the connection. The first time we do this we will receive a warning that will allow us to confirm that we are wanting to carry out the connection. If we're happy enough to trust this host on an on-going basis, feel free to tick the 'Always trust this host, add this key to the cache' box.



FileZilla Connection Warning

Then click on 'OK' and the connection should be made.



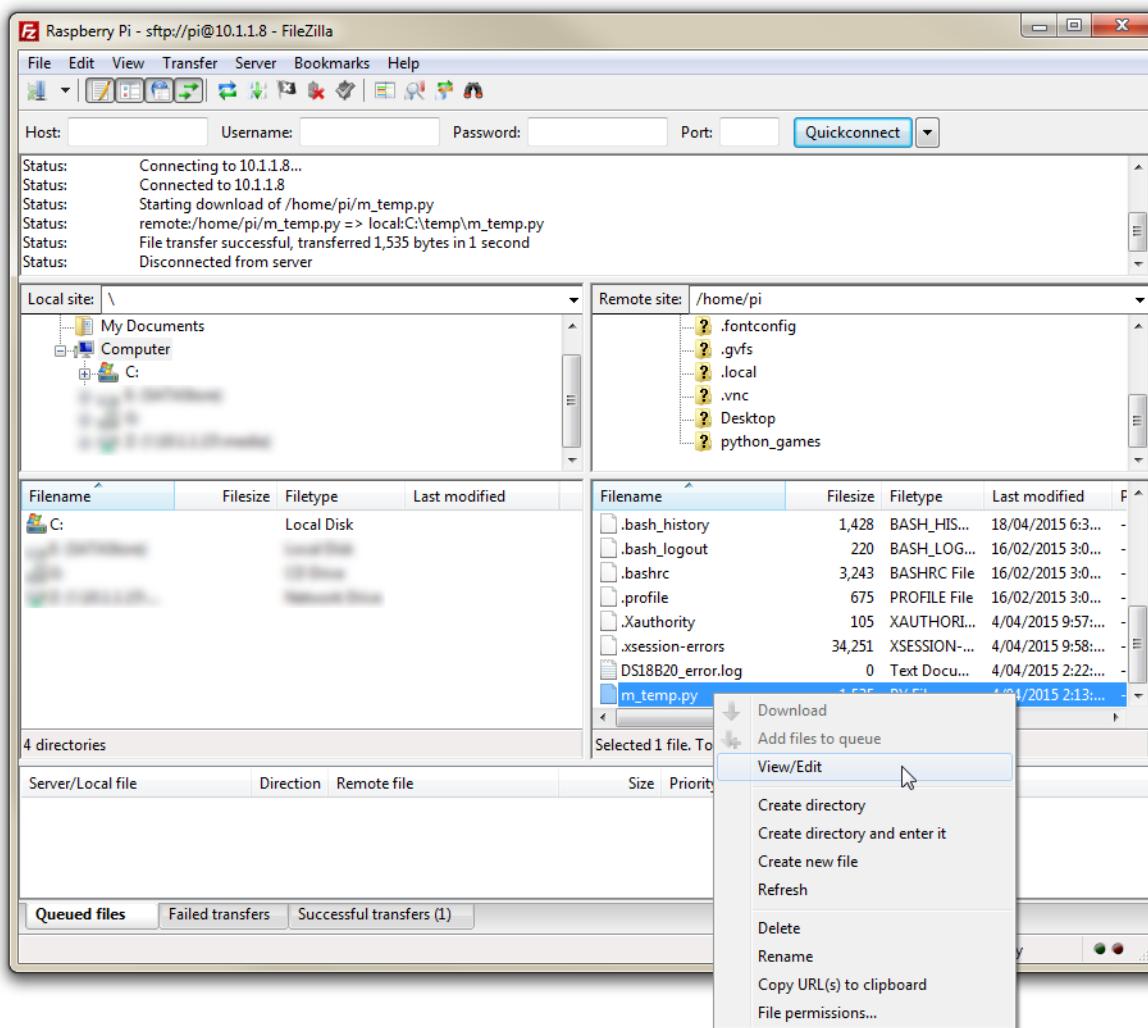
FileZilla Connected

The GUI will now show the local (Desktop) files on the left and the remote (Pi) files on the right hand side of the window. We can use these two windows to drag and drop files between the computers.

## Bonus: Edit Files on the Pi from your Desktop Editor

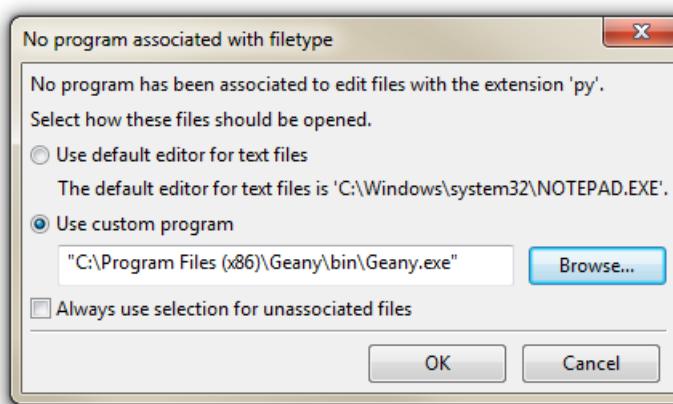
Once we are able to see the files on the Pi in FileZilla, we can go one step further and edit them on our Desktop machine in our favourite editor. As an example, I can use a TightVNC session to log into a remote desktop session on the Pi and edit a file in a nice GUI editor, but there is a great deal of overhead involved in supporting a desktop on the Pi and then routing that via the network. Alternatively, I can use Putty to connect via the terminal and I can use 'nano' or similar to edit the file, but as marvellous as those programs are, they will never be as familiar to me as the editor that I use most of the time on my Desktop. In my case I am a fan of 'Geany', so I would like to be able to edit a file on the Pi from my desktop using Geany.

To demonstrate how to do this we can go to a suitable file on the Pi window ('m\_temp.py' has been used here as an example) and right click on it;



Right click on file to edit in FileZilla

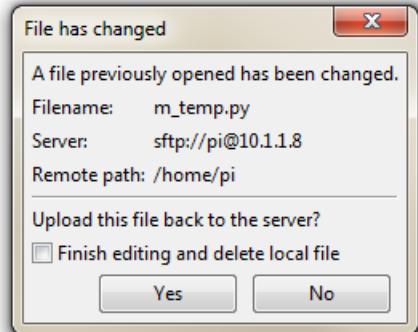
We can then select 'View/Edit'



Associate the File Type to an Editor

If the file has no associated program for editing, we can select one specifically, then click on 'OK'.

The file will then open in your favourite editor and can be worked on to our hearts content. Once we are finished editing the file and save it we will be presented with another dialogue box that will ask us if we want to update the file back on the server ‘Pi’.



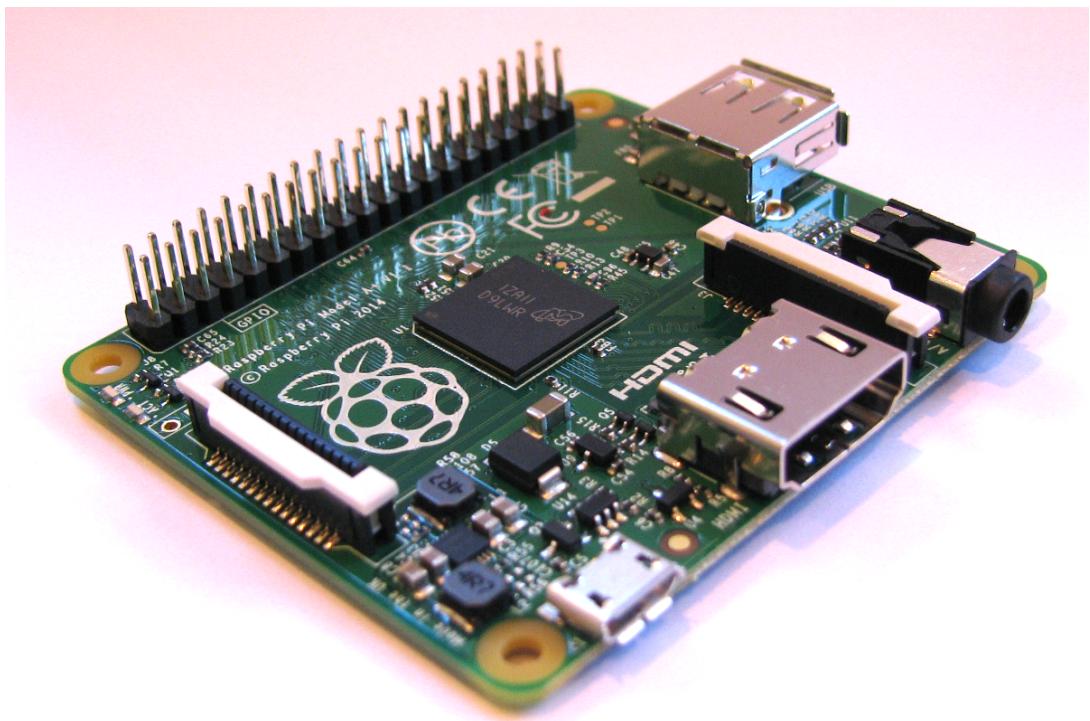
#### Upload the Edited File

The answer is ‘Yes’, and the reason we do this is that when we go to edit the file it creates a local copy on the Desktop, we edit it and then when we go to save it, it uploads the file back to the remote (‘Pi’) server.

# Hardware

The following is a list of various hardware components used in the projects we'll work through.

## Raspberry Pi A+

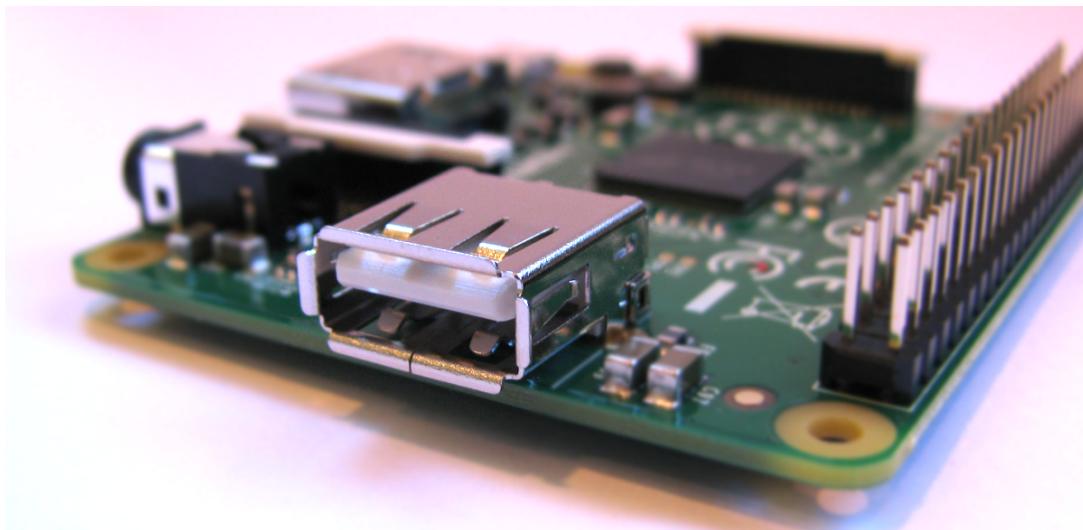


Raspberry Pi A+

The model A+ of the Raspberry Pi is the most modern version of the lower-spec model of the Raspberry Pi line. It replaced the original Model A in November 2014. It is 65 x 56 x 10mm, weighs 23g and is powered by a Broadcom BCM2835 ARM11 700Mhz with 256MB RAM.

## USB Port

It includes 1 x USB Port (with a maximum output of 1.2A)



Raspberry Pi A+ USB Ports

## Video Out

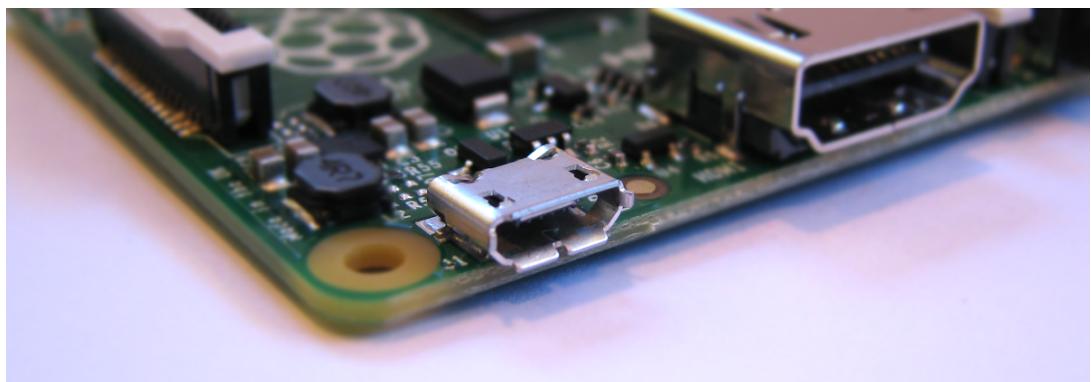
Integrated Videocore 4 graphics GPU capable of playing full 1080p HD video via a HDMI video output connector. HDMI standards rev 1.3 & 1.4 are supported with 14 HDMI resolutions from  $640 \times 350$  to  $1920 \times 1200$  plus various PAL and NTSC standards.



Raspberry Pi A+ HDMI Video Output

## USB Power Input Jack

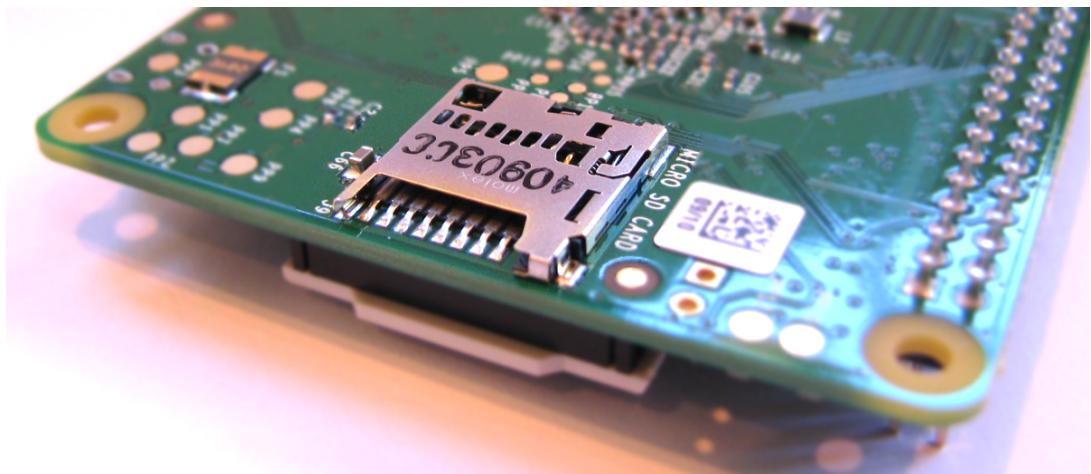
The board includes a 5V 2A Micro USB Power Input Jack.



Raspberry Pi A+ USB Power Input

## MicroSD Flash Memory Card Slot

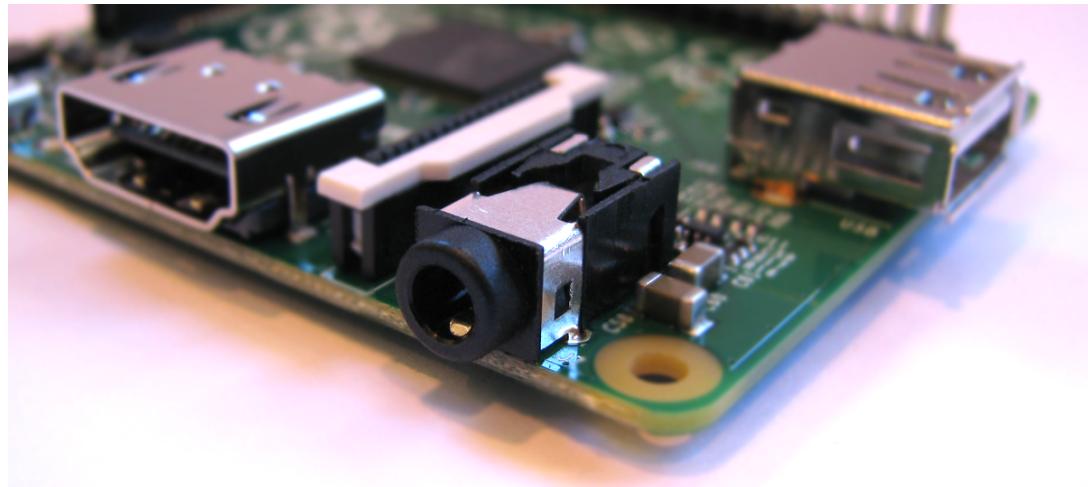
The A+ Raspberry Pi includes a push-push microSD card socket. This is on the ‘underside’ of the board.



Raspberry Pi A+ MicroSD Card Socket

## Stereo and Composite Video Output

The A+ includes a 4-pole (TRRS<sup>134</sup>) type connector that can provide stereo sound if you plug in a standard headphone jack and composite video output with stereo audio if you use a TRRS adapter.



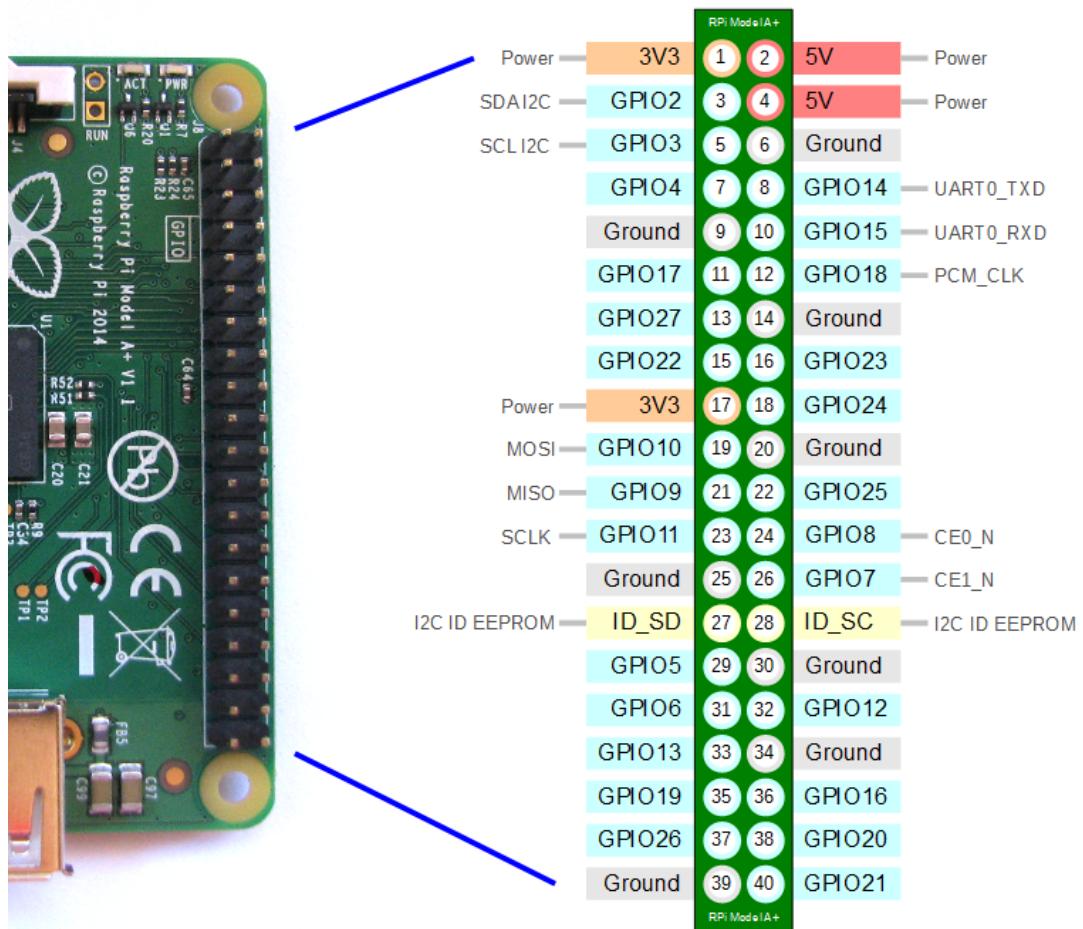
Raspberry Pi A+ A/V Connector

---

<sup>134</sup><http://www.cablechick.com.au/blog/understanding-trrs-and-audio-jacks/>

## 40 Pin Header

The Raspberry Pi A+ includes a 40-pin, 2.54mm header expansion slot (Which allows for peripheral connection and expansion boards).



Raspberry Pi A+ GPIO Connector

## Raspberry Pi B+

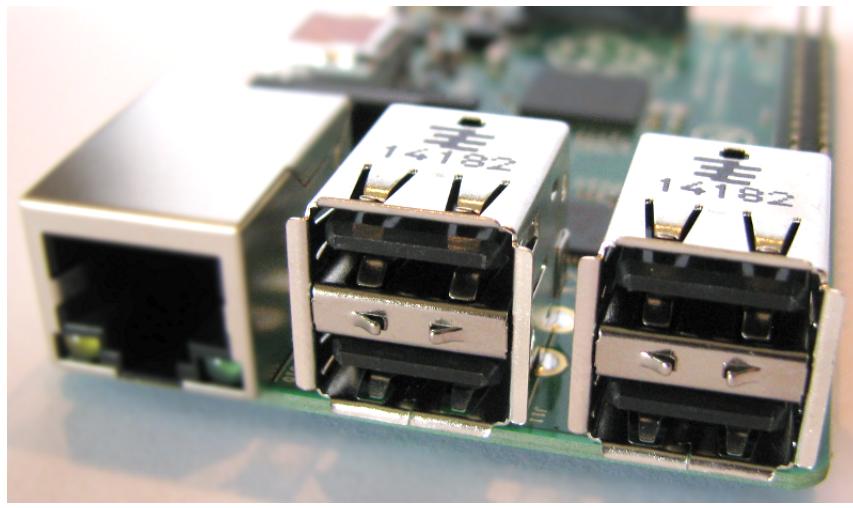


Raspberry Pi B+

The model B+ of the Raspberry Pi is the highest-spec variant of the Raspberry Pi line. It replaced the original Model B in July 2014. It is 85 x 56 x 17mm, weighs 45g and is powered by a Broadcom BCM2835 ARM11 700Mhz with 512MB RAM.

### USB Ports

It includes 4 x USB Ports (with a maximum output of 1.2A)

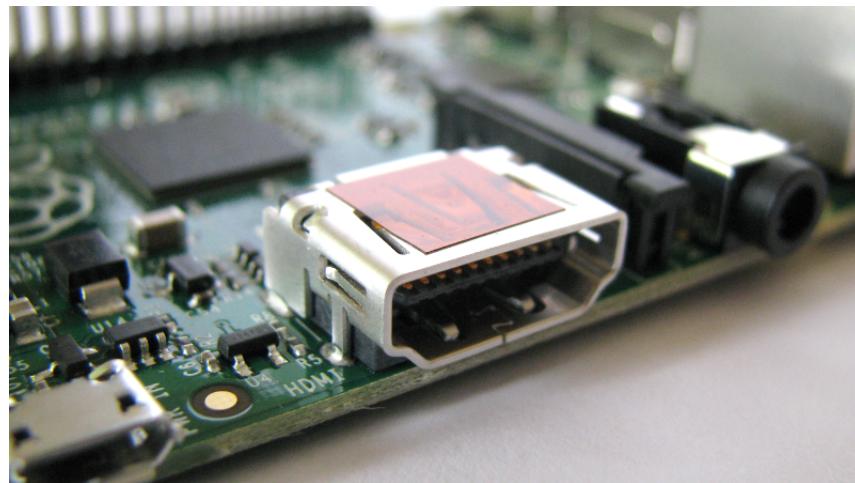


Raspberry Pi B+ USB Ports

### Video Out

Integrated Videocore 4 graphics GPU capable of playing full 1080p HD video via a HDMI video output connector. HDMI standards rev 1.3 & 1.4 are supported with 14 HDMI resolutions from

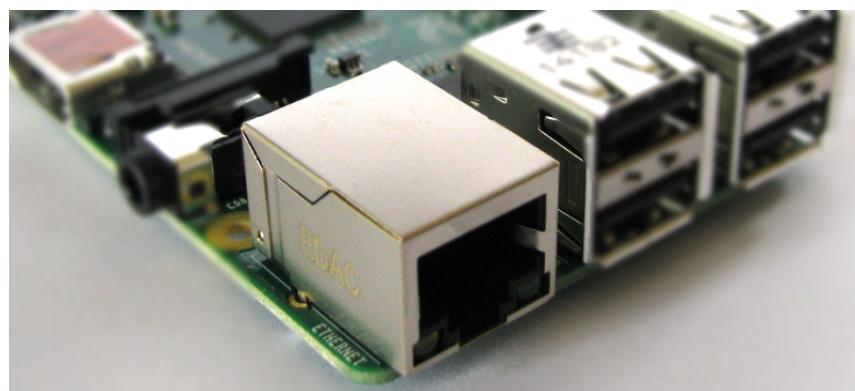
640×350 to 1920×1200 plus various PAL and NTSC standards.



Raspberry Pi B+ HDMI Video Output

## Ethernet Network Connection

There is an integrated 10/100Mb Ethernet Port for network access.



Raspberry Pi B+ Ethernet Connector

## USB Power Input Jack

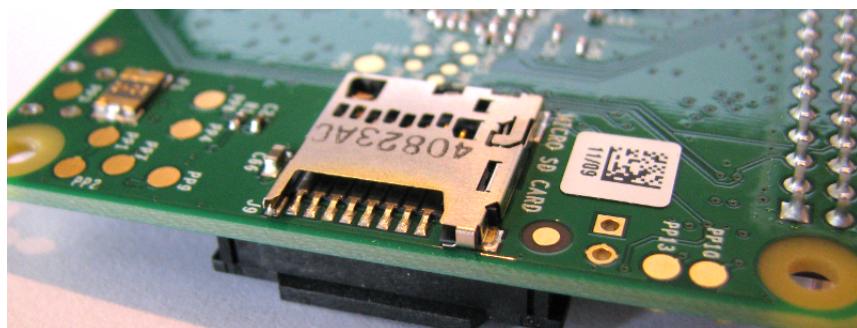
The board includes a 5V 2A Micro USB Power Input Jack.



Raspberry Pi B+ USB Power Input

## MicroSD Flash Memory Card Slot

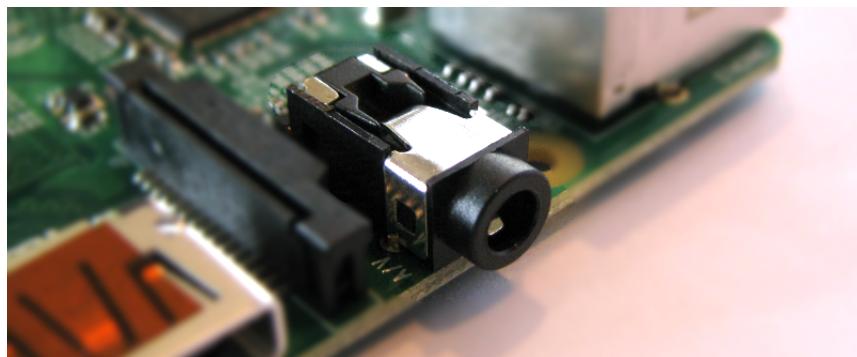
The B+ Raspberry Pi includes a push-push microSD card socket. This is on the ‘underside’ of the board.



Raspberry Pi B+ MicroSD Card Socket

## Stereo and Composite Video Output

The B+ includes a 4-pole (TRRS<sup>135</sup>) type connector that can provide stereo sound if you plug in a standard headphone jack and composite video output with stereo audio if you use a TRRS adapter.



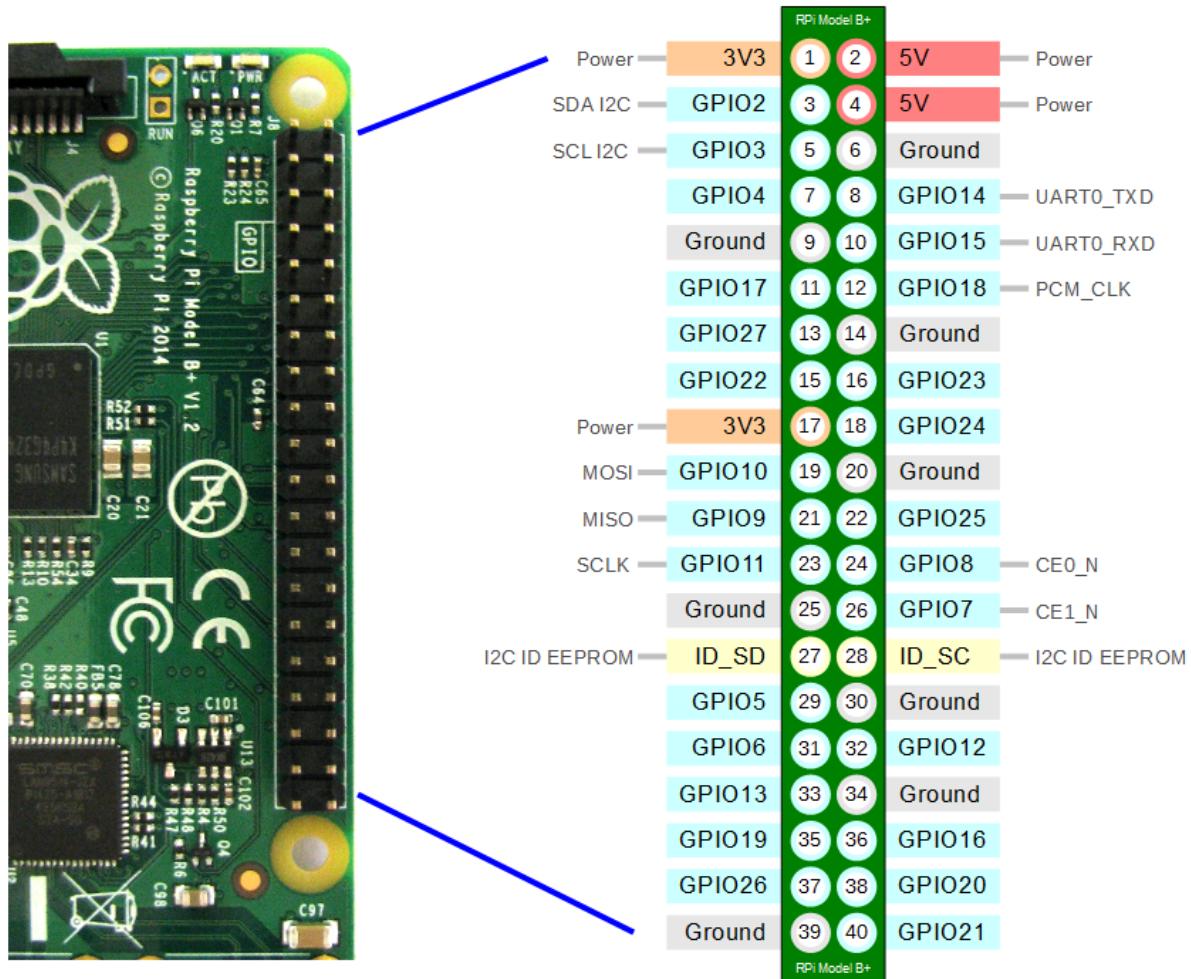
Raspberry Pi B+ A/V Connector

---

<sup>135</sup><http://www.cablechick.com.au/blog/understanding-trrs-and-audio-jacks/>

## 40 Pin Header

The Raspberry Pi B+ includes a 40-pin, 2.54mm header expansion slot (Which allows for peripheral connection and expansion boards).



Raspberry Pi B+ GPIO Connector

## Raspberry Pi B

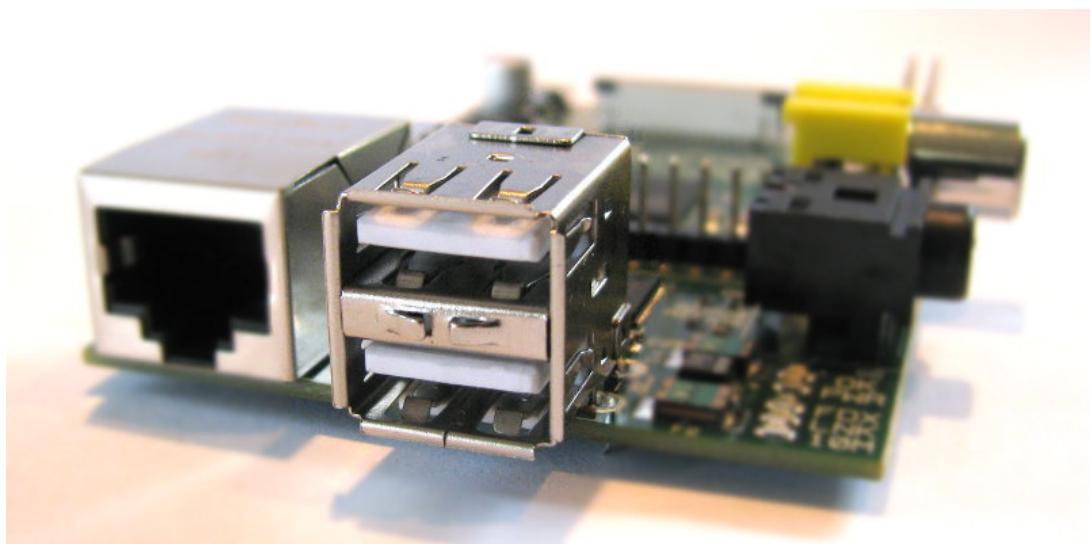


Raspberry Pi B

The model B of the Raspberry Pi is the precursor to the B+ variant of the Raspberry Pi line. It was replaced by the model B+ in July 2014. It is 85mm x 56mm (which does not include protruding connectors), weighs 45g and is powered by a Broadcom BCM2835 ARM11 700Mhz with 512MB RAM on variants supplied after October 2012 (Revision 2) or 256MB prior to that time (Revision 1).

## USB Ports

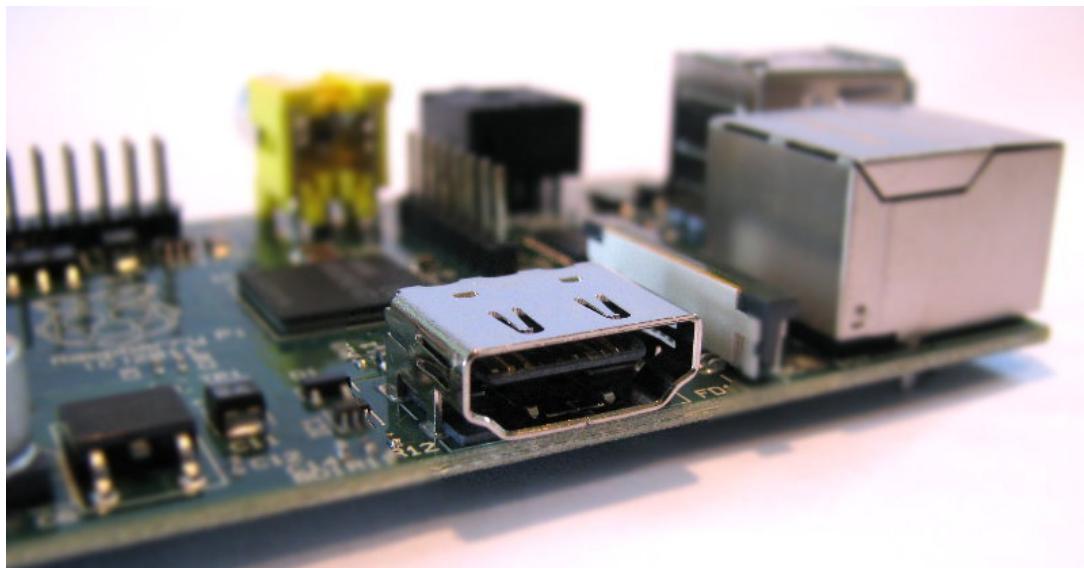
It includes 2 x USB Ports (with a maximum output of 1.2A)



Raspberry Pi B USB Ports

## HDMI Video Out

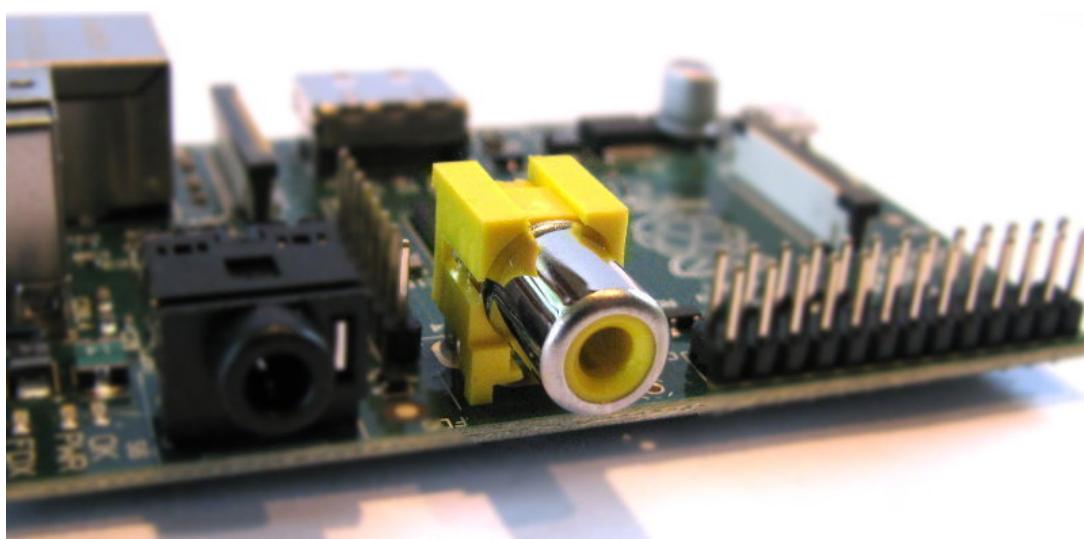
Integrated Videocore 4 graphics GPU capable of playing full 1080p HD video via a HDMI video output connector. HDMI standards rev 1.3 & 1.4 are supported with 14 HDMI resolutions from  $640 \times 350$  to  $1920 \times 1200$  plus various PAL and NTSC standards.



Raspberry Pi B HDMI Video Output

## Composite Video Out

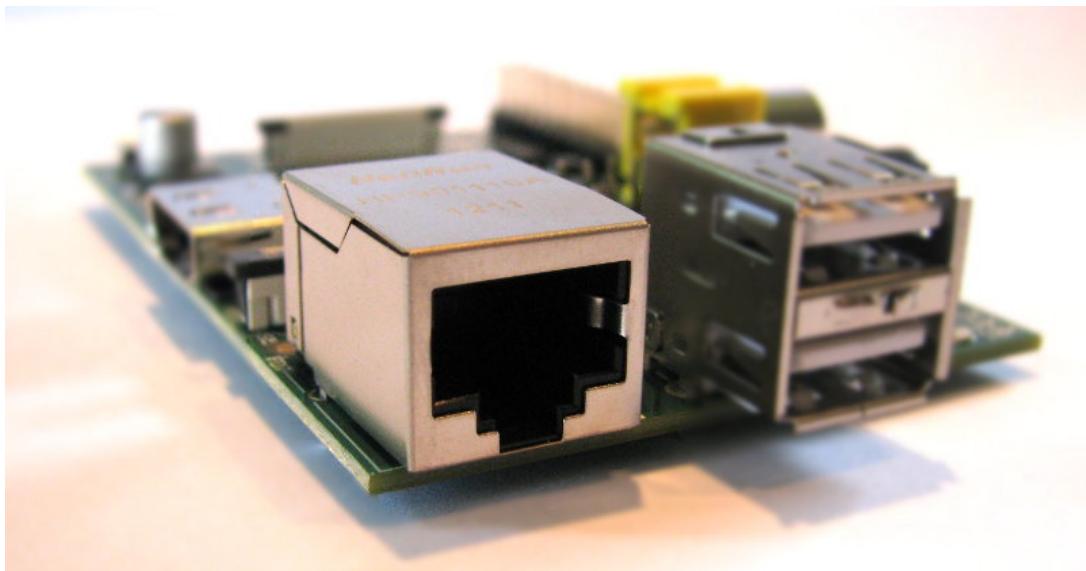
An RCA Composite video connector capable of supplying either NTSC or PAL video.



Raspberry Pi B Composite Video Output

## Ethernet Network Connection

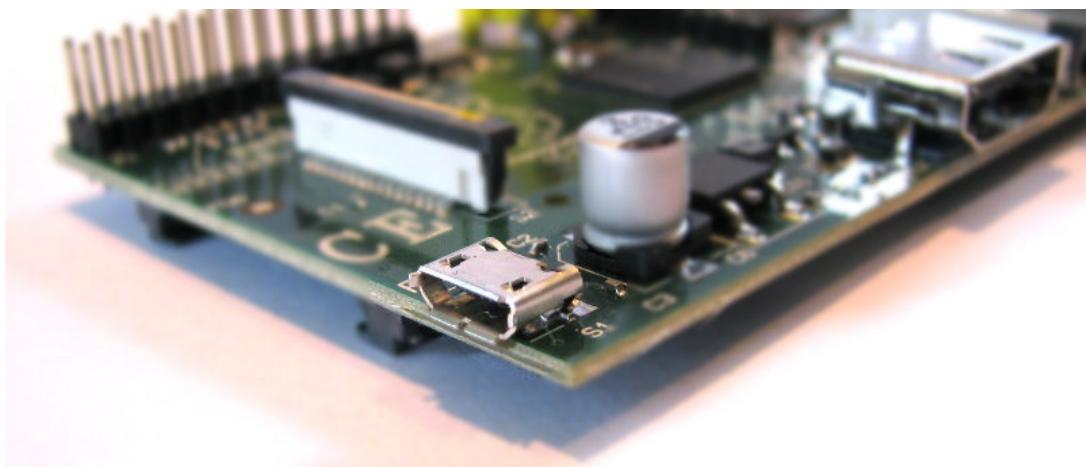
There is an integrated 10/100Mb Ethernet Port for network access.



Raspberry Pi B Ethernet Connector

## USB Power Input Jack

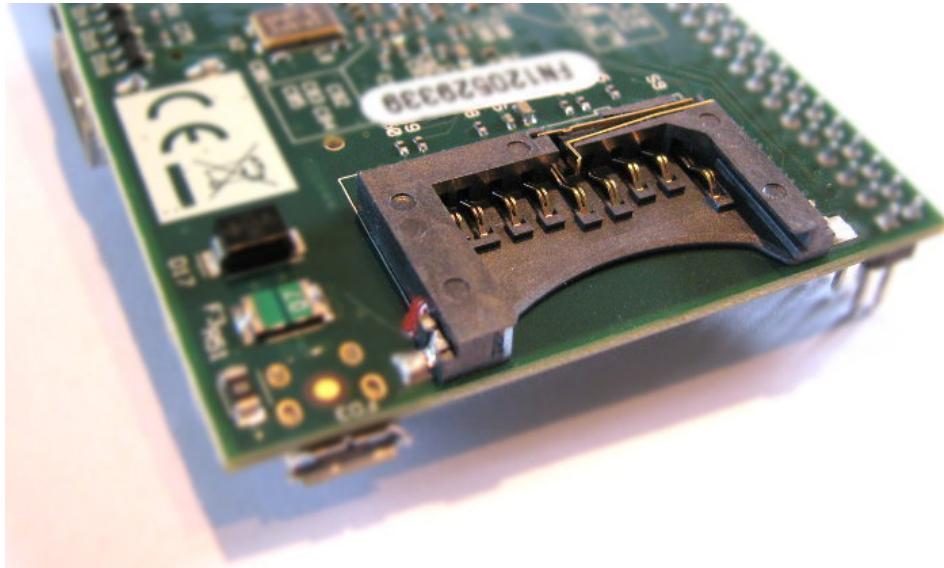
The board includes a 5V 2A Micro USB Power Input Jack.



Raspberry Pi B Micro USB Power Input

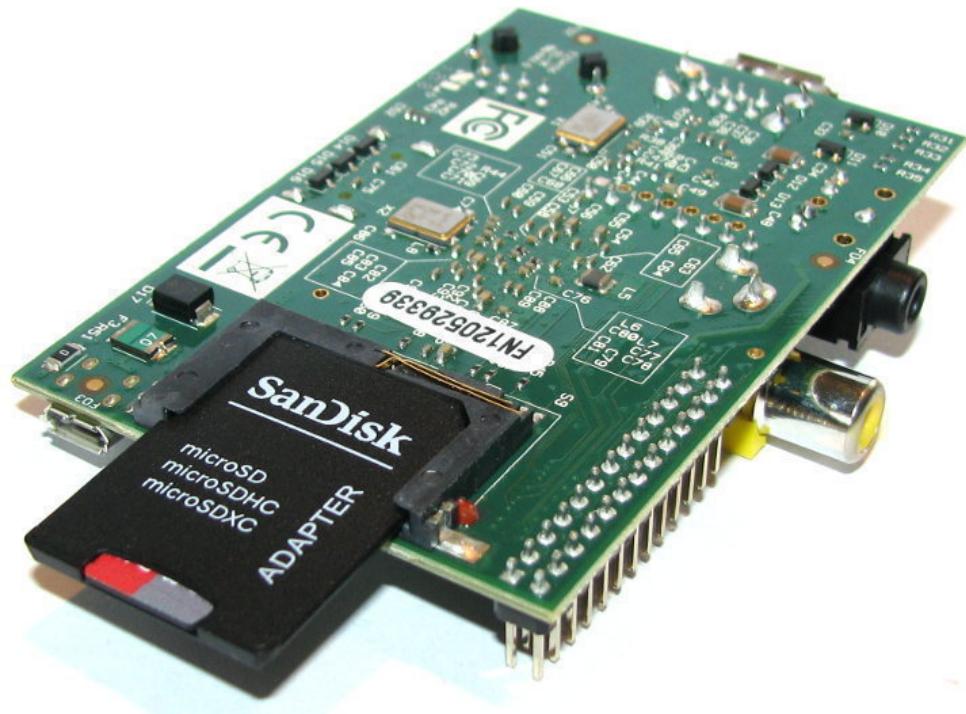
## SD Flash Memory Card Slot

The B Raspberry Pi includes a full size SD/MMC/SDIO memory card slot. This is on the ‘underside’ of the board.



Raspberry Pi B SD Card Socket

When a full size SD card is fitted it protrudes some considerable distance from the edge of the board.

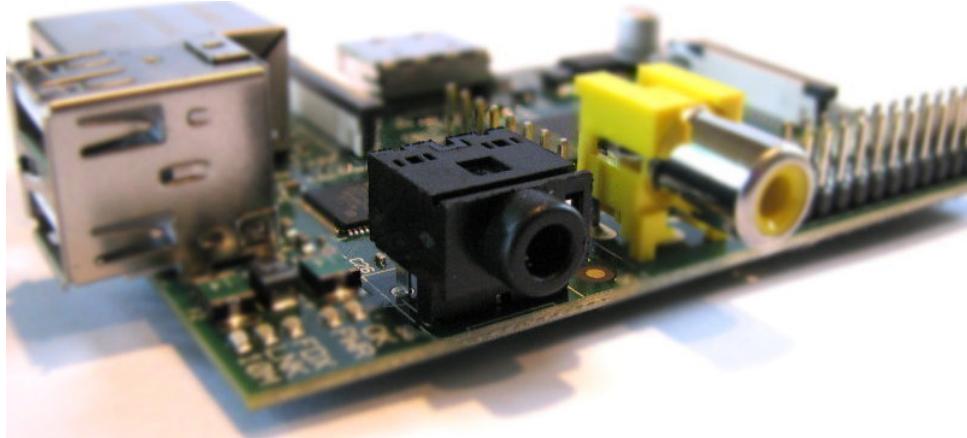


Raspberry Pi B with SD Card Fitted

There are low profile adapters that will allow microSD cards to be used that avoid this overhang.

## Audio Output

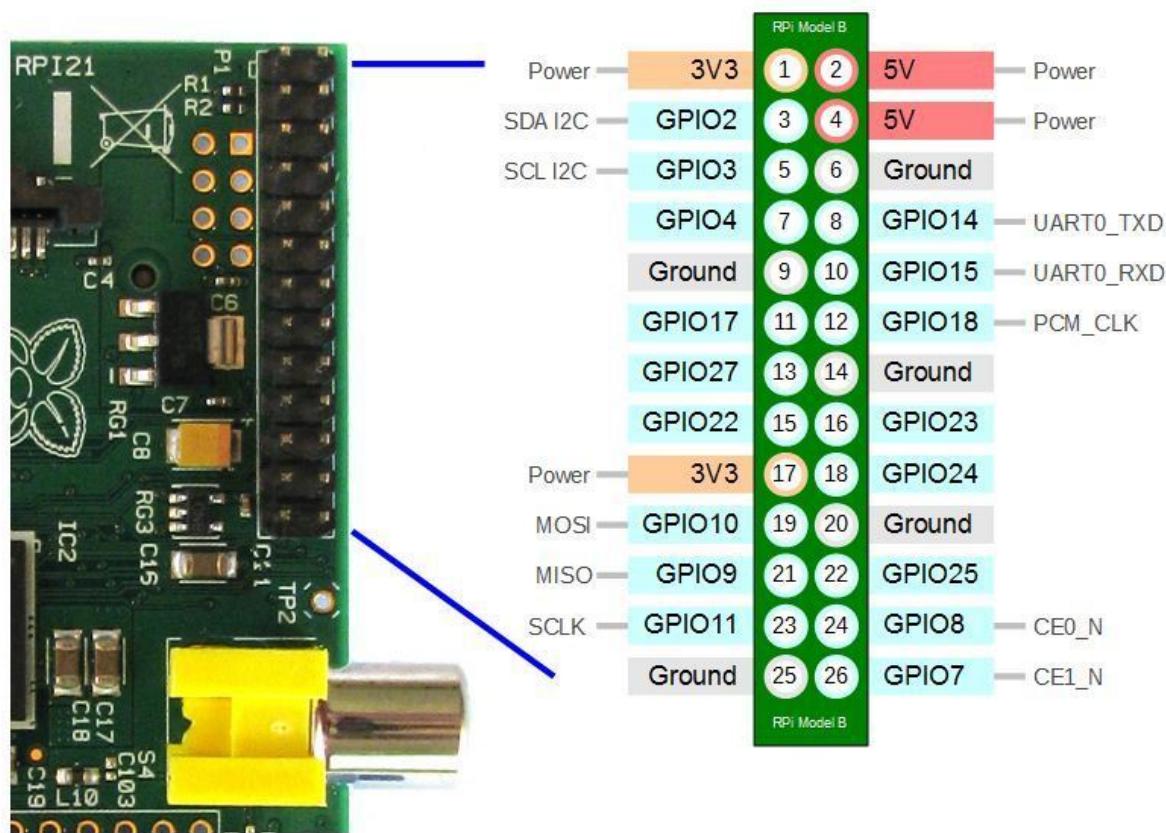
The B model includes a 3.5mm stereo jack connector for audio output.



Raspberry Pi B Audio Connector

## 26 Pin Header

The Raspberry Pi model B includes a 26-pin, 2.54mm header expansion slot (Which allows for peripheral connection and expansion boards).



Raspberry Pi B+ GPIO Connector

## Cases

### Multicomp MC-RP002-CLR

This is a popular and low priced case that comes in two main halves with separate feet and mounting screws.

It is designed for the B+ and 2B models.



Multicomp MC-RP002-CLR

Accessories in their little baggie.



Adhesive feet and mounting screws

**Side views.**



RJ45 and USB ports



Slot for 40 pin ribbon cable



Micro USB, HDMI and stereo / composite ports



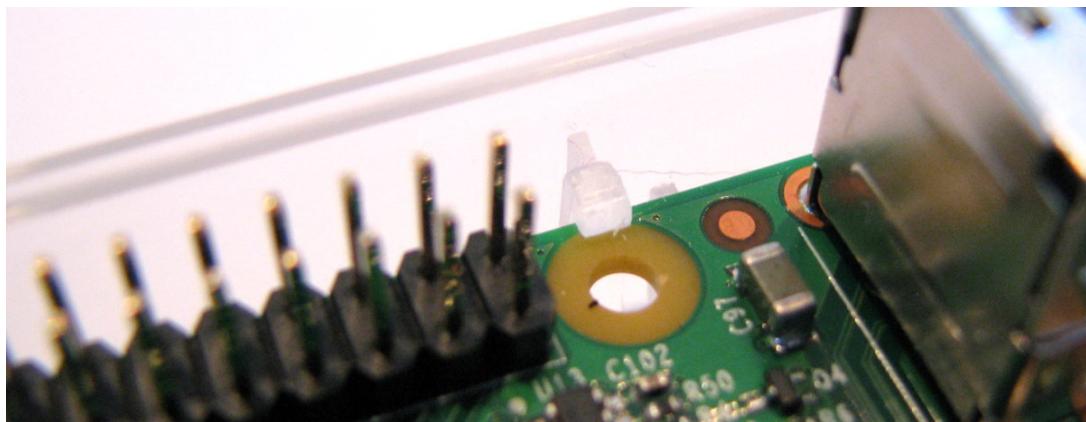
MicroSD card fitted snugly

## Fitting the Raspberry Pi



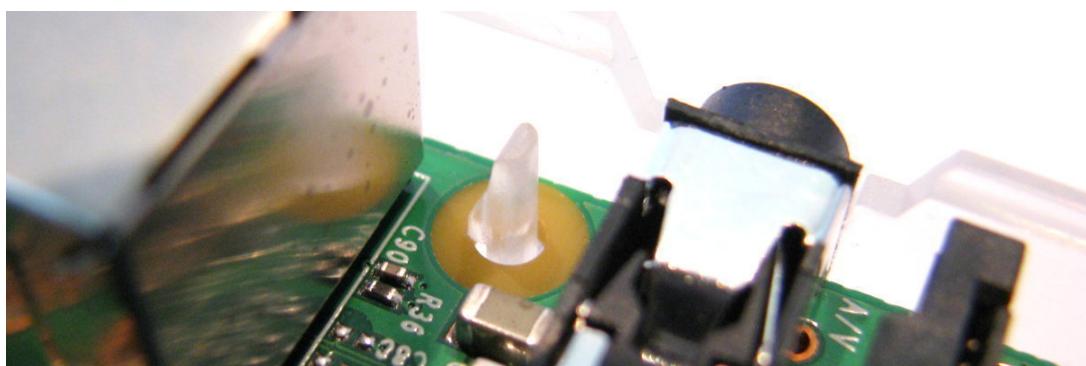
Pay attention to the sequence described here when fitting the board. Trying to do it the other way around can snap off one or more of the little plastic catches.

The Pi is fitted to the case by sliding the board under the two small plastic catches on the side with the 40 pin header...



Board under the catches

... then the other side of the board can be lowered into place with two plastic locating / latching pins going through the mounting holes on the micro USB / HDMI side of the board.



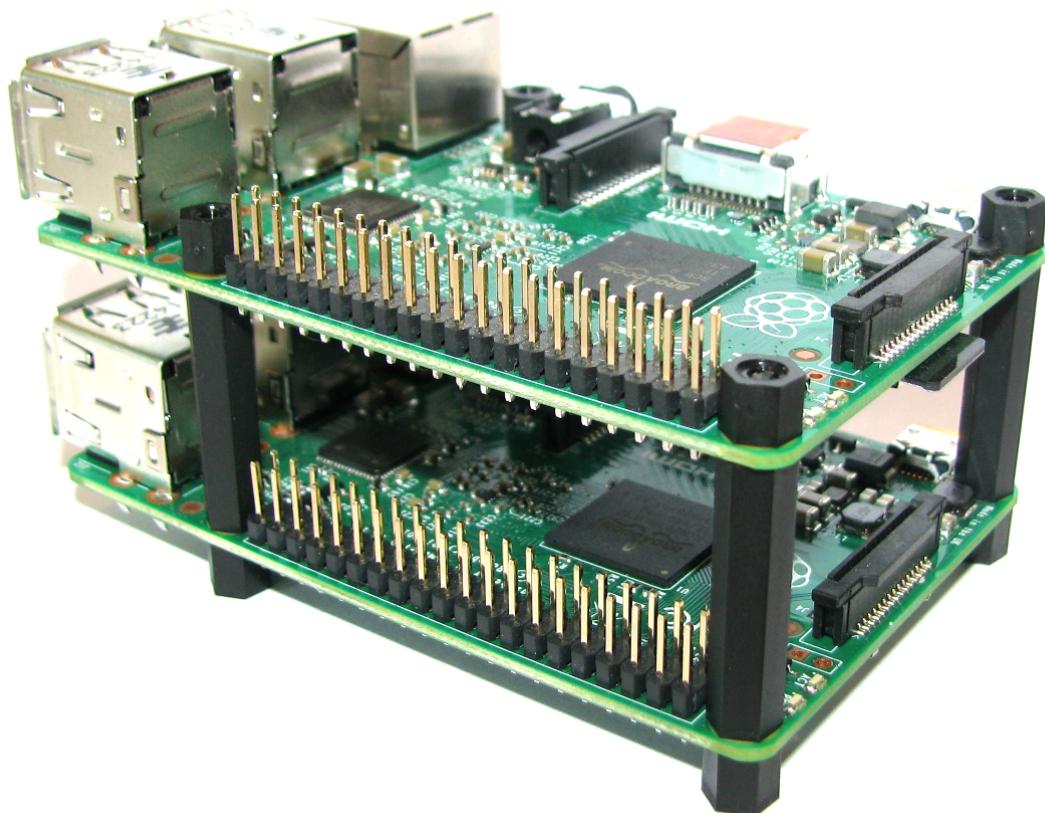
Plastic locating pin through board mounting hole

## DIY Open Multi-stack Pi

It could be argued (quite successfully IMHO) that there is an aesthetic beauty to the Raspberry Pi in its natural state and therefore there seems to be little reason to hide it in a case when not required for protective reasons.

Likewise, with the Raspberry Pi being available at a very reasonable price, there seems to be little to no reason to limit ourselves to only running one device.

Without further ado I present the Open Multi-stack Raspberry Pi case.



**Open Multi-stack Pi**

While this might look like Raspberry Pis stacked together with nylon standoffs in reality its.... OK, it *is* Raspberry Pis stacked together with nylon standoffs. But that doesn't mean that its a bad idea.

What we're looking at here is a combination of four M3 x 6 + 6 Nylon spacer hex pillars on the bottom.



**M3 x 6 + 6 Nylon Spacer Hex Pillars**

That's a 6mm long Hex section (with an internal M3 threaded hole) with a 6mm long, M3 threaded screw extension. The screw end can be pushed through the mounting holes on the Raspberry Pi board and these can in turn be attached to four M3 x 25 + 6 Nylon spacer hex

pillars.



M3 x 25 + 6 Nylon Spacer Hex Pillars

The 25mm length allows enough space to set another Raspberry Pi on top through the threaded section of the 25mm long hex units. This way we can secure the top Raspberry Pi using 5mm long M3 threaded spacers.



These Spacers are available from a range of sources. I have found success in buying them from Deal Extreme, but they are widely available from many suppliers;

- [M3 x 5 Nylon Plastic Hexa Pillar Spacer Supporter Black \(20 PCS\)<sup>136</sup>](http://www.dx.com/p/zndiy-bry-r203-305-m3-x-5-nylon-plastic-hexa-pillar-spacer-supporter-black-20-pcs-328434)
- [M3 x 6 + 6 Nylon Spacer Hex Pillars Black \(20 PCS\)<sup>137</sup>](http://www.dx.com/p/zndiy-bry-r201-306-m3-x-6-6-nylon-spacer-hex-pillars-for-r-c-multicopters-black-20-pcs-335780)
- [M3 x 25 + 6 Nylon Spacer Hex Nylon Pillars Black \(20 PCS\)<sup>138</sup>](http://www.dx.com/p/zndiy-bry-m3-x-25-6-nylon-spacer-hex-nylon-pillars-for-multicopter-rc-model-black-20-pcs-336489)

<sup>136</sup><http://www.dx.com/p/zndiy-bry-r203-305-m3-x-5-nylon-plastic-hexa-pillar-spacer-supporter-black-20-pcs-328434>

<sup>137</sup><http://www.dx.com/p/zndiy-bry-r201-306-m3-x-6-6-nylon-spacer-hex-pillars-for-r-c-multicopters-black-20-pcs-335780>

<sup>138</sup><http://www.dx.com/p/zndiy-bry-m3-x-25-6-nylon-spacer-hex-nylon-pillars-for-multicopter-rc-model-black-20-pcs-336489>

## Sensors

### DS18B20 Programmable Resolution 1-Wire Digital Thermometer

The DS18B20<sup>139</sup> is a digital thermometer that provides Celsius temperature measurements. The DS18B20 communicates over a 1-Wire bus that by definition requires only one data line (and ground) for communication. It has an operating temperature range of -55°C to +125°C and is accurate to ±0.5°C over the range of -10°C to +85°C.

Each DS18B20 has a unique 64-bit serial code, which allows multiple DS18B20s to function on the same 1-Wire bus. Thus, it is simple to use one microprocessor to control many DS18B20s distributed over a large area.

The sensor connections are: red (VCC), blue or yellow (DATA), black (GND) (there are two variations of the sensor).

---

<sup>139</sup><http://www.maximintegrated.com/en/products/analog/sensors-and-sensor-interface/DS18B20.html>

## Accessories

### VGA to HDMI Adapter

The Raspberry Pi comes with a modern video audio interface in the form of an HDMI connector. However, that's not to say that everyone will have a monitor (or TV) with an HDMI input ready to use. The ubiquitous connection type that has been used for many years on computers has been the [VGA<sup>140</sup>](#) (Video Graphics Array) connector. It consists of a three-row, 15-pin D-sub connector. Since it may be a simple job to find an older model monitor with a VGA connector, what is needed to connect the Raspberry Pi to a monitor with a VGA connector is a HDMI male to VGA female adapter cable (assuming that our monitor has a VGA cable attached).

I have used a simple model from [DealExtreme<sup>141</sup>](#) which can be found [here<sup>142</sup>](#) and at the time of writing costs around \$7 (USD) which includes free shipping.



HDMI Male to VGA Female Adapter

While the HDMI interface supports audio, the standard VGA connector does not. There are work-arounds to allow audio through, but they are beyond the scope of this book.

### In-line switch for USB power supply

The Raspberry Pi does not have a built in off/on switch, so we are left to find our own way to switch power off and on to the unit. In most cases this can be as crude as switching our USB power supply on or off at the wall. However, a useful solution is to use a USB cable that incorporates a switch for turning the power off or on.

<sup>140</sup>[http://en.wikipedia.org/wiki/VGA\\_connector](http://en.wikipedia.org/wiki/VGA_connector)

<sup>141</sup><http://www.dx.com/>

<sup>142</sup><http://www.dx.com/p/1080p-hdmi-male-to-vga-female-adapter-cable-w-mini-hdmi-male-to-hdmi-female-adapter-black-193337>



**USB Cable with Switch**

There are several variations of the same theme on the market, so let your fingers do the Googling to find one that will suit your needs.

## **Multiple Outlet USB Power Supply**

If we find ourselves needing to provide power to multiple Raspberry Pis it is worth thinking about the convenience of utilising a device that can provide multiple supply points (cables). It is also reasonable to consider the power requirements of the devices and the capability of a supply to maintain a suitable current to our Raspberry Pis and their connected devices (WiFi dongles, etc).

With that in mind, we would ideally want to be able to supply up to 2A per device.

I have been happy using an [Anker 5 port charger<sup>143</sup>](#) for supplying up to 5 devices with a theoretical total current draw of 8 amps.

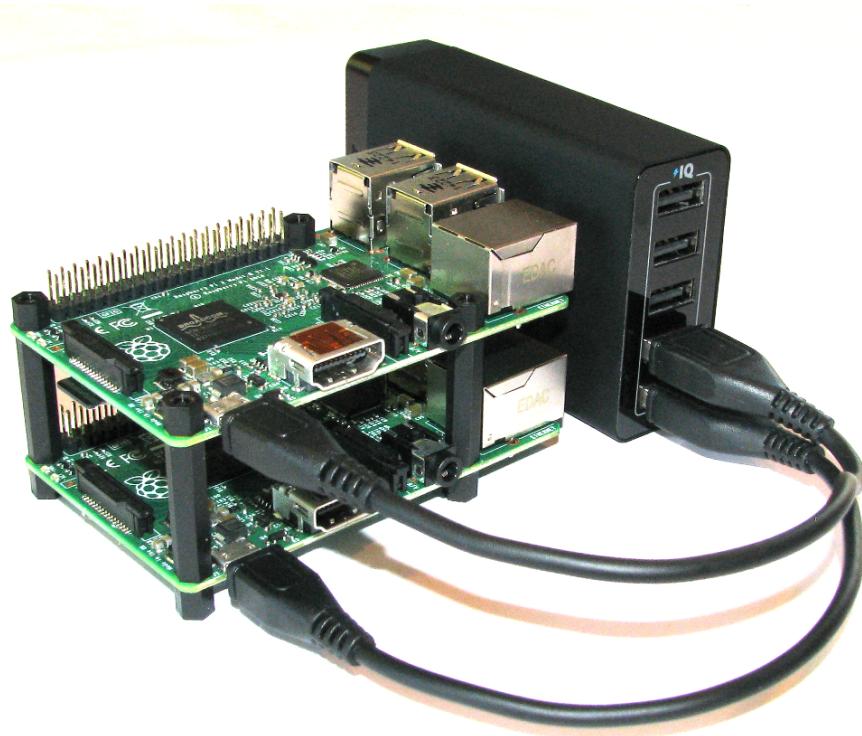
---

<sup>143</sup><http://www.anker.com/product/71AN7105SS-BA>



Anker 5 port charger

This is a convenient device, which, when combined with some short USB to microUSB cables can be an effective and efficient means to supply a number of Raspberry Pis.



Anker 5 port charger and Raspberry Pis

# Linux Command Glossary

The following are commands used in the book and a short explanation of what they do.

## **apt-get**

The apt part of apt-get stands for ‘Advanced Packaging Tool’. The program is a process for managing software packages installed on Linux machines, or more specifically [Debian<sup>144</sup>](#) based Linux machines (Sine those based on ‘[redhat<sup>145</sup>](#)’ typically use their rpm (red hat package management (or more lately the recursively named ‘rpm package management’) system). As Raspbian is based on Debian, so we are using apt-get.

APT simplifies the process of managing software on Unix-like computer systems by automating the retrieval, configuration and installation of software packages. This was historically a process best described as ‘dependency hell’ where the requirements for different packages could mean a manual installation of a simple software application could lead a user into a morass of despair.

Common apt-get usage that we will use when using the Raspberry Pi include (and bear in mind that in most of these cases we will be prefixing the command with [sudo](#) to give ourselves the appropriate permissions);

### **apt-get update**

```
apt-get update
```

This will resynchronize our local list of packages files, updating information about new and recently changed packages. If an apt-get upgrade is planned, an apt-get update (see below) should always be performed first.

### **apt-get upgrade**

```
apt-get upgrade
```

The apt-get upgrade command will install the newest versions of all packages currently installed on the system. If a package is currently installed and a new version is available, it

---

<sup>144</sup><https://www.debian.org/>

<sup>145</sup><http://www.redhat.com/>

will be retrieved and upgraded. Any new versions of current packages that cannot be upgraded without changing the install status of another package will be left as they are.

As mentioned above, an `apt-get update` should always be performed first so that `apt-get upgrade` knows which new versions of packages are available.

## **apt-get install**

```
apt-get install *package_name*
```

The `apt-get install` command installs or upgrades one (or more) packages. All additional (dependency) packages required will also be retrieved and installed.

## **cat**

The `cat` command is used for a range of functions. The name ‘cat’ is short for ‘catenate’, which is to say to connect things in a series. This is certainly one of its common uses, but a better overview would be to say that the `cat` command is used to;

- Display text files at the command line
- Copy text files into a new document
- Join one text file to the end of another text file, combining them

For example, to display a text file (`foo.txt`) on the screen we can use the following command;

```
cat foo.txt
```

As a result the contents of the file ‘`foo.txt`’ will be sent to the screen (be aware, if the file is sufficiently large, it will simply dump the contents in a long scrolling waterfall of text).

We could just as easily display two files one after the other (catenated) as follows;

```
cat foo.txt bar.txt
```

Instead of having the file sent to the screen, we can specify that `cat` send our file to a new (renamed) file as follows;

```
cat foo.txt > newfile.txt
```

This could be thought of an a equivalent to a file copy action and uses the redirection symbol `>`.

But taking the process one step further we can take our original two files and combine them into a single file with;

```
cat foo.txt bar.txt > newfile.txt
```

Finally we could use `cat` to append a file to an already existing file similar to the following;

```
cat another.txt >> existing.txt
```

Here we use the redirection operator `>>` to add the contents of the file `another.txt` to the already existing file `existing.txt`.

## cd

The `cd` command is one of the commands used most often in Linux. It is used to move around in the directory structure of the file system.

For example, when we first log into the Raspberry Pi as the ‘pi’ user we will find ourselves in the `/home/pi` directory. If we wanted to change into the `/home` directory (go up a level) we could use the command;

```
cd /home
```

Alternatively, since this change was only one level up, we could have used the following shorthand method of going up a level;

```
cd ..
```

Now that we are in the `/home` directory, to change back into the `pi` directory we can either use the following command (which fully qualifies the directory location);

```
cd /home/pi
```

Or, since we are already in the `home` directory and the `pi` directory is also in the `home` directory, we can simply provide the destination without the full structure as follows;

```
cd pi
```

Take some time to get familiar with the concept of moving around the directory structure from the command line as it is an important skill to establish early in Linux.

## chmod

The `chmod` command allows us to set or modify a file's permissions.

Every file on the computer has an associated set of permissions. Permissions tell the operating system what can be done with that file and by whom. There are three things you can (or can't) do with a given file:

- read it,
- write (modify) it and
- execute it.

Linux permissions specify what can the owner do, what can the owner group do and what everybody else can do with the file. For any given owner, we need three bits to specify access permissions: the first to denote read (r) access, the second to denote (w) access and the third to denote execute (x) access.

We also have three levels of ownership: 'owner', 'group' and 'all' so we need a triplet for each, resulting in nine bits. Each bit can be set or clear. (not set) We mark a set bit by its corresponding operation letter (r, w or x) and a clear bit by a dash (-) and put them all on a row. An example might be `rwxr-xr--`. This means is that the owner can do anything with the file (`rwx`), the group owners can read and execute the file (`r-x`) and the rest of the world can only read it (`r--`) (Usually in Linux there is also another bit that leads this 9-bit pattern, but we will ignore this in the mean time.).

Permission	Symbolic	3-bit	Octal
read, write and execute	rwx	111	7
read and write	rw-	110	6
read and execute	r-w	101	5
read only	r--	100	4
write and execute	-wx	011	3
write only	-w-	010	2
execute only	--x	001	1
none	---	000	0

Another interesting thing to note is that permissions take a different slant for directories. Here's what they mean:

- read determines if a user can view the directory's contents, i.e. do ls in it.
- write determines if a user can create new files or delete file in the directory. (Note here that this essentially means that a user with write access to a directory can delete files in the directory even if he/she doesn't have write permissions for the file! So be careful with this.)
- execute determines if the user can cd into the directory.

As an example, consider the following command (which would most likely be prefixed with sudo);

```
chmod 775 /var/www
```

Here the permissions for the /var/www directory are set so that the owner can read from, write to and change into the directory. Group owners can also read from, write to and change into the directory. All others can read from and change into the directory, but they cannot create or delete a file within it.

Remembering that only the owner of a file may use chmod to alter a file's permissions.

## chown

The chown command changes the user and/or group ownership of given files. If only a user name is given, that user is made the owner of each given file, and the files' group is not changed. If the owner is followed by a colon and a group name (with no space in between them) the group ownership of the files is changed as well. If a colon but no group name follows the user name, that user is made the owner of the files and the group of the files is changed to that user's login group. If the colon and group are given, but the owner is omitted, only the group of the files is changed.

As an example, consider the following command (which would most likely be prefixed with sudo);

```
chown www-data:www-data /var/www
```

Here the user `www-data` is made the owner of the directory `www` (in the `/var` directory). Additionally, the group ownership of that directory is given to the group `www-data`.

## crontab

The `crontab` command give the user the ability to schedule tasks to be run at a specific time or with a specific interval. `crontab` is a concatenation of “cron table” because it uses the job scheduler `cron` to execute tasks. `cron` is named after “Khronos,” the Greek personification of time. The schedule is called the `crontab`, which is also the name of the program used to edit that schedule.

As an example, consider that we wish to run a Python script every day at 6am. The following command will let us edit the `crontab`:

```
crontab -e
```

Once run it will open the `crontab` in the `nano` editor. The file will look as follows:

```
# 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).
#
# 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
```

The default file obviously includes some explanation of how to format an entry in the crontab. In our case we wish to add in an entry that told the script to start at 6 hours and 0 minutes each day. The crontab accepts six pieces of information that will allow that action to be performed. each of those pieces is separated by a space.

1. A number (or range of numbers), m, that represents the minute of the hour;
2. A number (or range of numbers), h, that represents the hour of the day;
3. A number (or range of numbers), dom, that represents the day of the month;
4. A number (or list, or range), or name (or list of names), mon, that represents the month of the year;
5. A number (or list, or range), or name (or list of names), dow, that represents the day of the week; and
6. command, which is the command to be run, exactly as it would appear on the command line.

Assuming that we want to run a Python script called ‘m\_temp.py’ which was in the ‘pi’ home directory the line that we would want to add would be as follows;

```
0 6 * * * /usr/bin/python /home/pi/m_temp.py
```

So at minute 0, hour 6, every day of the month, every month, every day of the week we run the command /usr/bin/python /home/pi/m\_temp.py (which if we were at the command line in the pi home directory we would run as python m\_temp.py, but since we can’t guarantee where we will be when running the script, we are supplying the full path to the python command and the m\_temp.py script).

If we want to run the same command every 2 hours we can use the \*/2 notation, so that our line in the crontab would look like the following;

```
* */2 * * * /usr/bin/python /home/pi/m_temp.py
```

## **ifconfig**

The ifconfig command can be used to view the configuration of, or to configure a network interface.

Used with no arguments it will display information about all the operational interfaces. For example running;

```
ifconfig
```

Will produce something similar to the following on a simple Raspberry Pi.

```

eth0      Link encap:Ethernet HWaddr 76:12:45:56:47:53
          UP BROADCAST MULTICAST MTU:1500 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

lo       Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          UP LOOPBACK RUNNING MTU:65536 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet HWaddr 09:87:65:54:43:32
          inet addr:10.1.1.8 Bcast:10.1.1.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:3978 errors:0 dropped:898 overruns:0 frame:0
          TX packets:347 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:859773 (839.6 KiB) TX bytes:39625 (38.6 KiB)

```

The output above is broken into three sections; eth0, lo and wlan0.

- eth0 is the first Ethernet interface and in our case represents the RJ45 network port on the Raspberry Pi (in this specific case on a B+ model). If we had more than one Ethernet interface, they would be named eth1, eth2, etc.
- lo is the loopback interface. This is a special network interface that the system uses to communicate with itself. You can notice that it has the IP address 127.0.0.1 assigned to it that we have described as designating the ‘localhost’.
- wlan0 is the name of the first wireless network interface on the computer. This reflects our wireless USB adapter. Any additional wireless interfaces would be named wlan1, wlan2, etc.

To view the configuration of all the network interfaces (not just the active ones), we can specify the -a option, as follows;

```
ifconfig -a
```

Or to view a specific interface we can name it directly;

```
ifconfig wlan0
```

If we wanted to disable a network interface we can use `ifconfig` to configure it as so;

```
sudo ifconfig eth0 down
```

If we were to run `ifconfig` after disabling the interface as above it would no longer be reported as ‘active’.

## ls

The `ls` command lists the contents of the directory that we are currently in. This is another important Linux command to understand and use well and there are a large number of options that can be used.

To demonstrate, after logging into the Pi as the ‘pi’ user, if we change directory to the `/var` directory with the command...

```
cd /var
```

If we then execute the `ls` command ...

```
ls
```

... we should see the following;

```
pi@raspberrypi ~ % ls  
backups cache lib local lock log mail opt run spool swap tmp www
```

ls command

This is the default listing of the contents of the `/var` directory.

If we use the `-l` option we can show the total files in the directory and subdirectories, the names of the files in the current directory, their permissions, the number of subdirectories in directories listed, the size of the file, and the date of last modification.

```
ls -l
```

```
pi@raspberrypi /var $ ls -l
total 102440
drwxr-xr-x  2 root      root          4096 Jul 28 03:12 backups
drwxr-xr-x 12 root      root          4096 Dec 28 05:57 cache
drwxr-xr-x 43 root      root          4096 Dec 28 05:57 lib
drwxrwxr-x  2 root      uucp          4096 Jul 28 03:12 local
lrvwxrwxrwx  1 root      root          9 Dec 22 00:06 lock -> /run/lock
drwxr-xr-x 11 root      root          4096 Dec 28 19:42 log
drwxrwxr-x  2 root      mail          4096 Dec 22 00:06 mail
drwxr-xr-x  2 root      root          4096 Dec 22 00:06 opt
lrvwxrwxrwx  1 root      root          4 Dec 22 00:06 run -> /run
drwxr-xr-x  4 root      root          4096 Dec 22 00:09 spool
-rw-----  1 root      root          104857600 Dec 25 01:28 swap
drwxrwxrwt  2 root      root          4096 Dec 28 05:45 tmp
drwxrwxr-x  2 www-data  www-data    4096 Dec 28 05:42 www
```

ls -l command

Or, we could use a combination of options such as `-ltr` to list files sorted by the time they were last modified in reverse order (most recently modified files last).

```
ls -ltr
```

```
pi@raspberrypi /var $ ls -ltr
total 102440
drwxrwxr-x  2 root      uucp          4096 Jul 28 03:12 local
drwxr-xr-x  2 root      root          4096 Jul 28 03:12 backups
drwxr-xr-x  2 root      root          4096 Dec 22 00:06 opt
drwxrwxr-x  2 root      mail          4096 Dec 22 00:06 mail
lrvwxrwxrwx  1 root      root          4 Dec 22 00:06 run -> /run
lrvwxrwxrwx  1 root      root          9 Dec 22 00:06 lock -> /run/lock
drwxr-xr-x  4 root      root          4096 Dec 22 00:09 spool
-rw-----  1 root      root          104857600 Dec 25 01:28 swap
drwxrwxr-x  2 www-data  www-data    4096 Dec 28 05:42 www
drwxrwxrwt  2 root      root          4096 Dec 28 05:45 tmp
drwxr-xr-x 12 root     root          4096 Dec 28 05:57 cache
drwxr-xr-x 43 root     root          4096 Dec 28 05:57 lib
drwxr-xr-x 11 root     root          4096 Dec 28 19:42 log
```

ls -l command

Take the opportunity to Google the `ls` command and to example and play with some of the options presented on the internet. There are a large number and they are worth becoming familiar with.

## modprobe

The `modprobe` command allows us to add (or remove) modules to the [Linux Kernel<sup>146</sup>](#) (The code that forms the core of the Linux operating system). The Linux kernel is designed with a monolithic structure, but with the ability to be able to change kernel modules while running. In the case of the command...

```
sudo modprobe w1-therm
```

<sup>146</sup><http://www.howtogeek.com/howto/31632/what-is-the-linux-kernel-and-what-does-it-do/>

... the module `w1-therm` is added to support measurement of temperature via the 1-Wire bus (it is necessary to do this as the superuser via `sudo`).

## **sudo**

The `sudo` command allows a user to execute a command as the ‘superuser’. This allows the user to run programs or give commands that should only be executed with a degree of caution as they could potentially affect the normal operation of the computer. However, a user can only use this command if they have the correct permissions to do so. In the case of the user ‘pi’, it has those permissions. ‘`sudo`’ is shorthand for ‘superuser do’.

Under normal situations the use of `sudo` would require a user to enter their password, but by default the Raspbian operating system has the ‘pi’ user configured in the ‘/etc/sudoers’ file to avoid entering the password every time.

## **usermod**

The `usermod` command is used by a superuser (typically via `sudo`) to change a user’s system account settings. There are a wide range of options available for use. The following example is one used in the early stages of setting up the Raspberry Pi;

```
sudo usermod -a -G www-data pi
```

The `-a` option adds the user to an additional ‘supplementary’ group (`-a` must only be used in a situation where it is followed by the use of a `-G` option). The `-G` option lists the supplementary group to which a user will be added. In this case the group is `www-data`. Finally the user which is getting added to the supplementary group is `pi`.

# Appendices

## Raspberry Pi Quick Set-up

### Download Raspbian Image

Download the latest version of the Raspbian Operating System from the raspberrypi.org page; <http://www.raspberrypi.org/downloads/>.

### Writing Raspbian to the SD Card

Once we have the image file we need to get it onto our SD card.

Download and install [Win32DiskImager<sup>147</sup>](#).

Start the Win32 Disk Imager program and select the **correct** drive letter for your SD card and the Raspbian disk image that you downloaded. Then select ‘Write’ and the disk imager will write the image to the SD card.

Once the process is finished exit the disk imager and eject the card from the computer.

### Installing Raspbian

Insert the SD card into the slot on the Raspberry Pi and turn on the power.

Using the Raspberry Pi Software Configuration Tool tool you can first ensure that all of the SD card storage is available to the Operating System. Once this has been completed leave the other settings where they are and select finish.

Once the reboot is complete you login using the default username `pi` and password `raspberry`.

### Software Updates

Type in the following line which will find the latest lists of available software;

```
sudo apt-get update
```

Upgrade the software to latest versions using;

---

<sup>147</sup><http://sourceforge.net/projects/win32diskimager/>

```
sudo apt-get upgrade
```

## Static IP Address

Configure your home network to make a static IP address available for the RPi. Note your netmask and default gateway.

On the RPi, edit the file /etc/network/interfaces.

```
sudo nano /etc/network/interfaces
```

Change the line that tells the network interface to use DHCP (`iface eth0 inet dhcp`) to use the static address that we decided on earlier along with information on the netmask to use and the default gateway. So replace the line...

```
iface eth0 inet dhcp
```

... with the following lines (and don't forget to put YOUR address, netmask and gateway in the file, not necessarily the ones below);

```
iface eth0 inet static  
address 10.1.1.8  
netmask 255.255.255.0  
gateway 10.1.1.1
```

To allow the changes to become operative type in;

```
sudo reboot
```

## Remote access via TightVNC

### On Windows

To install TightVNC for Windows, go to the [downloads page<sup>148</sup>](http://www.tightvnc.com/download.php) and select the appropriate version for your operating system.

<sup>148</sup><http://www.tightvnc.com/download.php>

Work through the installation process answering all the questions until you get to the screen asking what setup type to choose.

We only want to install the viewer for the software. Click on ‘Custom’ and then click on the ‘TightVNC Server’ drop-down and select ‘Entire feature will be unavailable’, then select ‘Next’.

The ‘Select Additional Tasks’ selections can be left at their defaults.

Then click on ‘Install’.

Click on ‘Finish’ and you should be done.

## On the Raspberry Pi.

From the command line, type;

```
sudo apt-get install tightvncserver
```

Now we can run the program by typing in;

```
tightvncserver
```

You will be prompted to enter a password that we will use on our Windows client software to authenticate that we are the right people connecting. (there is a maximum length of password of 8 characters).

You will be asked if you want to have a ‘view-only’ password that would allow a client to look at, but not interact with the remote desktop.

The software will then assign us a desktop and print out a messages that we will need to note saying something like;

```
New 'X' desktop is raspberrypi:1
```

the :1 informs us of the number of the desktop session that we will be looking at.

Now on the Windows desktop, start the TightVNC Viewer program. We will see a dialogue box asking which remote host we want to connect to. In this box we will put the IP address of our Raspberry Pi followed by a colon (:) and the number of the desktop that was allocated in the tightvncserver program (10.1.1.8:1).

We will be prompted for the password that we set to access the remote desktop;

In theory we will then be connected to a remote desktop view of the Raspberry Pi from our Windows desktop.

## Starting TightVNC at boot.

We will add the following command into `rc.local`;

```
su - pi -c '/usr/bin/tightvncserver :1'
```

To do this we will edit the `/etc/rc.local` file with the following command;

```
sudo nano /etc/rc.local
```

Add in our lines so that the file looks like the following;

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
    printf "My IP address is %s\n" "$_IP"
fi

# Start tightvncserver
su - pi -c '/usr/bin/tightvncserver :1'

exit 0
```

Test that the service starts when the Pi boots by typing in;

```
sudo reboot
```

When the Raspberry has finished starting up again, we should be able to see in the list of text that shows up while the boot sequence is starting the line New 'X' desktop is raspberrypi:1. Assuming that this is the case, power off the Raspberry Pi;

```
sudo poweroff
```

Now physically turn off the power to the Pi.

Unplug the keyboard / mouse and the video from the unit so that there is only the power connector and the network cable left plugged in.

Now turn the power back on.

(We will need to wait for 30 seconds or so while it boots up)

Start the TightVNC Viewer program on the Windows desktop and we will be able to see a remote view of the Raspberry Pi's desktop

## **Copying and Pasting between Windows and the Raspberry Pi via TightVNC**

On the Raspberry Pi, install 'autocutsel';

```
sudo apt-get install autocutsel
```

Then edit the 'xstartup' file;

```
nano /home/pi/.vnc/xstartup
```

... to add in the line autocutsel -fork;

```
#!/bin/sh

xrdb $HOME/.Xresources
xsetroot -solid grey
autocutsel -fork
#x-terminal-emulator -geometry 80x24+10+10 -ls -title "$VNCDESKTOP Desktop" &
#x-window-manager &
# Fix to make GNOME work
export XKL_XMODMAP_DISABLE=1
/etc/X11/Xsession
```

Reboot the Raspberry Pi for the changes to take effect.

```
sudo reboot
```

## Remote access via SSH

### Setting up the Server (Raspberry Pi)

This is definitely one of the easiest set-up steps since SSH is already installed on Raspbian.

Check that SSH is already installed on Raspbian by typing the following from the command line;

```
/etc/init.d/ssh status
```

The Pi should respond with the message that the program `sshd` is running.

If for some reason SSH is *not* installed on your Pi, install with the command;

```
sudo apt-get install ssh
```

### Setting up the Client (Windows)

Download PuTTY from [here<sup>149</sup>](#).

Save the file somewhere logical and run the program.

Click on the ‘Selection’ option. On this page we want to change the ‘Action of mouse’ option from the default of ‘Compromise (Middle extends, Right paste)’ to ‘Windows (Middle extends, Right brings up menu)’.

Now select the ‘Session’ Category on the left hand menu. Here we want to enter our static IP address that we set up earlier and enter a name for it as a saved session. Then click on ‘Save’.

Select the raspberry Pi Session and click on the ‘Open’ button.

Click on the ‘Yes’ button to confirm that you trust the host that you’re trying to connect to.

A new terminal window will be shown with a prompt to `login as:`. Enter our user name (‘pi’) and then our password (if it’s still the default it is ‘raspberry’).

<sup>149</sup><http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>

## Setting up a WiFi Network Connection

Starting this section makes the assumption that you have a working wireless adapter and advises that the following configuration changes take place with the keyboard / mouse and monitor connected to the Raspberry Pi (I.e. not via a remote desktop).

Start with the Pi powered off and install the USB adapter into a convenient USB connection. Turn the power on and let the device be recognised.

Edit the `/etc/network/interfaces` file to set up the static IP address on the wireless interface.

```
sudo nano /etc/network/interfaces
```

This time we will edit the `interfaces` file using the ‘ssid’ (the network name) of the network that we are going to connect to and the password for the network;

```
auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0
auto wlan

iface wlan0 inet static
    address 10.1.1.8
    netmask 255.255.255.0
    gateway 10.1.1.1
    wpa-ssid "homenetwork"
    wpa-psk "h0mepassw0rd"
```

To allow the changes to become operative reboot the Pi;

```
sudo reboot
```

Once we have rebooted, we can check the status of our network interfaces by typing in;

```
ifconfig
```

... or disconnect the wired connection and attempt to connect to the Raspberry Pi via the remote desktop.

## Web Server and PHP

At the same time as setting up a web server on the Pi we will install PHP and the library for connecting Python and MySQL.

At the command line run the following command;

```
sudo apt-get install apache2 php5 libapache2-mod-php5 python-mysqldb
```

Once complete we need to restart our web server with the following command;

```
sudo service apache2 restart
```

We can now test our web server from the windows desktop machine.

Open up a web browser and type in the IP address of the Raspberry Pi into the URL bar at the top and confirm that we can see the test web page.

### Adjust permissions for web files

Make the ‘www-data’ group and user the owner of the /var/www directory;

```
sudo chown www-data:www-data /var/www
```

Allow the ‘www-data’ group permission to write to the directory;

```
sudo chmod 775 /var/www
```

Add the ‘pi’ user to the ‘www-data’ group;

```
sudo usermod -a -G www-data pi
```

Reboot the Raspberry Pi to let the changes take effect;

```
sudo reboot
```

## Database

From the command line run the following command;

```
sudo apt-get install mysql-server
```

You will be prompted (twice) to enter a root password for your database. Note it down somewhere safe;

Enter the following from the command line;

```
sudo apt-get install mysql-client php5-mysql
```

## phpMyAdmin

To begin installation run the following from the command line;

```
sudo apt-get install phpmyadmin
```

You will receive a prompt to ask what sort of web server we are using. Select ‘apache2’ (with the space bar) and tab to ‘OK’ to continue.

We will then be prompted to configure the database for use with phpMyAdmin. We want the program to look after it for us, so select ‘Yes’ and continue.

We will then be prompted for the password for the administrative account for the MySQL database. This is the root password for MySQL that we set up earlier. Enter it and tab to ‘OK’ to continue.

We will then be prompted for a password for phpMyAdmin to access MySQL. Use the same password as the MySQL root password to save confusion. Note it down. Then tab to ‘OK’ to continue (and confirm).

Edit the web server (Apache) configuration to access phpMyAdmin by executing the following command from the command line;

```
sudo nano /etc/apache2/apache2.conf
```

Get to the bottom of the file by pressing ctrl-v a few times and there add the line;

```
Include /etc/phpmyadmin/apache.conf
```

Save the file and then restart Apache2;

```
sudo service apache2 restart
```

In a browser on the Windows (or on the Raspberry Pi) desktop and enter the IP address followed by /phpmyadmin (in the case of our example 10.1.1.8/phpmyadmin) it should start up phpMyAdmin in the browser.

Enter the username as ‘root’ and the MySQL root password that we set earlier, it will open up the phpMyAdmin interface.

## Allow remote database access

By default the Raspberry Pi’s Operating System is set up to deny that access and if this is something that you want to allow this is what you will need to do.

On the Raspberry Pi edit the configuration file ‘my.cnf’ in the directory /etc/mysql/.

```
sudo nano /etc/mysql/my.cnf
```

Find the section [mysqld] and edit the bind-address line and place a ‘#’ in front of the line to disable it as a configuration option.

```
#bind-address      = 127.0.0.1
```

Then restart the MySQL service;

```
sudo service mysql restart
```

## Add users to the database

Click on the ‘Privileges’ tab in phpMyAdmin we can see the range of users that are already set up.

We will create an additional two users. One that can only read (select) data from our database and another that can put data into (insert) the database.

From the ‘Privileges’ tab, select ‘Add a new user’;

Enter a user name ('pi\_select') and password then scroll down a little and tick 'SELECT' in the 'Global privileges' section.

Then press the 'Create User' button and we've created a user.

For the second user with the ability to insert data (let's call the user 'pi\_insert'), go through the same process, but tick the 'SELECT' *and* the 'INSERT' options for the data.

## Create a database

Go to the 'Databases' tab, enter a name for a new database (I've called one 'measurements') and click on 'Create'.

Congratulations! We're set up and ready to go.

## Understanding JavaScript Object Notation (JSON)

One of the most useful things you might want to learn when understanding how to present your data with D3 is how to *structure* your data so that it is easy to use.

There are several different types of data that can be requested by D3 including text, Extensible Markup Language (xml), HyperText Markup Language (html), Comma Separated Values (csv), Tab Separated Values (tsv) and JavaScript Object Notation (json).

Comma separated values and tab separated values are fairly well understood forms of data. They are expressed as rows and columns of information that are separated using a known character. While these forms of data are simple to understand, it is not easy to incorporate a hierarchy structure to the data, and when you try, it isn't natural and makes managing the data difficult.

JavaScript Object Notation (JSON) presents a different mechanism for storing data. A light weight description could read "JSON is a text-based open standard designed to present human-readable data. It is derived from the JavaScript scripting language, but it is language and platform independent."

When I first started using JSON, I struggled with the concept of how it was structured, in spite of some fine descriptions on the web (start with <http://www.json.org/><sup>150</sup> in my humble opinion). So the following is how I came to think of and understand JSON.



**Fair Warning:** This advice is no substitute for the correct explanation of the topic of data structures that I'm sure you could receive from a reputable educational site or institution. It's just the way I like to think of it :-). It's also just the way that I *started* to understand JSON. There is plenty to learn and understand once you grasp the basics. So this isn't a complete guide. Just the beginnings.

In the following steps we'll go through a process that (hopefully) demonstrates that we can transform identifiers that would represent the closing price for a stock of \$58.3 on 2013-03-14 into more traditional x,y coordinates.

I think of data as having an identifier and a value.

`identifier: value`

If a point on a graph is located at the x,y coordinates 150,25 then the identifier 'x' has a value 150.

`"x": 150`

If the x axis was a time-line, the true value for 'x' could be "2013-03-14".

---

<sup>150</sup><http://www.json.org/>

```
"x": "2013-03-14"
```

This example might look similar to those seen by users of d3.js, since if we're using date / time format we can let D3 sort out the messy parts like what coordinates to provide for the screen.

And there's no reason why we couldn't give the 'x' identifier a more human readable label such as "date". So our data would look like;

```
"date": "2013-03-14"
```

This is only one part of our original  $x,y = 150,25$  data set. The same way that the x value represented a position on the x axis that was really a date, the y value represents a position on the y axis that is really another number. It only gets converted to 25 when we need to plot a position on a graph at 150,25. If the 'y' component represents the closing price of a stock we could take the same principles used to transform...

```
"x": 150
```

... into ...

```
"date": "2013-03-14"
```

... to change ....

```
"y": 25
```

... into ...

```
"close": 58.3
```

This might sound slightly confusing, so try to think of it this way. We want to plot a point on a graph at 150,25, but the data that this position is derived from is really "2013-03-14", 58.3. D3 can look after all the scaling and determination of the range so that the point gets plotted at 150,25 and our originating data can now be represented as;

```
"date": "2013-03-14", "close": 58.3
```

This represents two separate pieces of data. Each of which has an identifier ("date" or "close") and a value ("2013-03-14" and 58.3)

If we wanted to have a series of these data points that represented several days of closing prices, we would store them as an array of identifiers and values similar to this;

```
{
  "date": "2013-03-14", close: 58.13 },
  { "date": "2013-03-15", close: 53.98 },
  { "date": "2013-03-16", close: 67.00 },
  { "date": "2013-03-17", close: 89.70 },
  { "date": "2013-03-18", close: 99.00 }
```

Each of the individual elements of the array is enclosed in curly brackets and separated by commas.



I am making the assumption that you are familiar with the concept of what an ‘array’ is. If this is an unfamiliar word, in the context of data, then I strongly recommend that you do some Goggling to build up some familiarity with the principle.

Now that we have an array, we can apply the same rules to it as we did the the item that had a single value. We can give it an identifier all of its own. In this case we will call it “data”. Now we can use our identifier: value analogy to use “data” as the identifier and the array as the value.

```
{
  "data": [
    { "date": "2013-03-14", close: 58.13 },
    { "date": "2013-03-15", close: 53.98 },
    { "date": "2013-03-16", close: 67.00 },
    { "date": "2013-03-17", close: 89.70 },
    { "date": "2013-03-18", close: 99.00 }
  ]
}
```

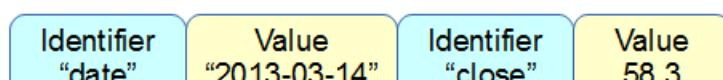
The array has been enclosed in square brackets to designate it as an array and the entire identifier: value sequence has been encapsulated with curly braces (much the same way that the subset “date”, “close” values were enclosed with curly braces).

If we try to convey the same principle in a more graphical format, we could show our initial identifier and value for the x component like so;



Single identifier and value

The we can add our additional component for the y value;



Single identifier and value

We can then add several of these combinations together in an array;

Identifier “date”	Value “2013-03-14”	Identifier “close”	Value 58.3
Identifier “date”	Value “2013-03-15”	Identifier “close”	Value 53.98
Identifier “date”	Value “2013-03-16”	Identifier “close”	Value 67.00
Identifier “date”	Value “2013-03-17”	Identifier “close”	Value 89.70
Identifier “date”	Value “2013-03-18”	Identifier “close”	Value 99.00”

Single identifier and value

Then the array becomes a value for another identifier “data”;

Identifier “data”	Value			
	Identifier “date”	Value “2013-03-14”	Identifier “close”	Value 58.3
	Identifier “date”	Value “2013-03-15”	Identifier “close”	Value 53.98
	Identifier “date”	Value “2013-03-16”	Identifier “close”	Value 67.00
	Identifier “date”	Value “2013-03-17”	Identifier “close”	Value 89.70
	Identifier “date”	Value “2013-03-18”	Identifier “close”	Value 99.00”

Single identifier and value

More complex JSON files will have multiple levels of identifiers and values arranged in complex hierarchies which can be difficult to interpret. However, laying out the data in a logical way in a text file is an excellent way to start to make sense of the data.