Learning Python with a Raspberry Pi

By Bradley Miles

Develop a user guide aimed at 13 year old students teaching them the programming language "Python" and basic electronics using a Raspberry Pi and a GPIO breadboard.

This project was created for one goal

To encourage the next generation to study Computer Science at GCSE, A-level and beyond.

Contents

Lesson 1—"Introduction"

What is a Raspberry Pi? 6

What are the GPIO Pins? 7

What is Python? 8

Installing Raspbian 9

Logging In 10

Configuring Raspberry Pi 11

Overclocking 12

Starting the Raspbian GUI 13

Terminal 14

Installing GPIO & Sound 15

Lesson 2—"Light it Up"

What is a breadboard? 16

Flash.py Tutorial 18

Flash.py Completed Code 20

Variableflash.py Tutorial 22

Variableflash.py Completed Code 23

SOS.py	Tutorial	24
SOS.py	Completed Code	25
Morsecode.py	Tutorial	26
Morsecode.py	Completed Code	27
Lesson 3—"Button It"		
Multipleflsh.py	Tutorial & Completed Code	32
Mybutton1.py	Tutorial	33
Mybutton1.py	Completed Code	34
Mybutton2.py	Tutorial & Completed Code	35
Mybutton3.py	Tutorial & Completed Code	36
Trafficlight.py	Tutorial	37
Trafficlight.py	Completed Code	38
Lesson 4—"Sound & SSH"		
JukeBox.py	Tutorial	42
JukeBox.py	Completed Code	43
SSH		44

Lesson 5—"ADC"

Temperaturesensor.py Tutorial 50

Temperaturesensor.py Completed Code 51

Lesson 6—"Games"

Minecraft 54

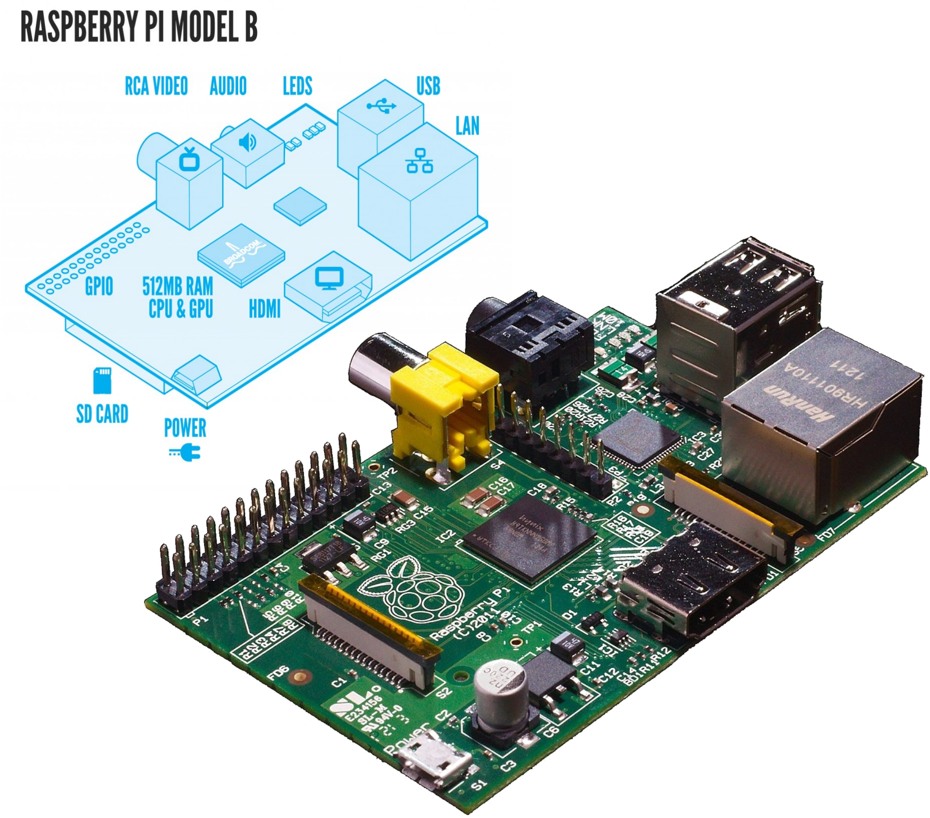
Quake 3 56

Lesson 1—"Introduction"

What is a raspberry pi?

The Raspberry Pi is a credit-card sized computer which can be used for many of the things that your desktop PC does, like word-processing and games.

However one key aspect that makes the Raspberry Pi so brilliant for schools is its ability to execute "Python" coded programmes. This allows us along with the General Purpose Input Output (GPIO) pins to create programs that can control anything from a single LED to opening your garage door.

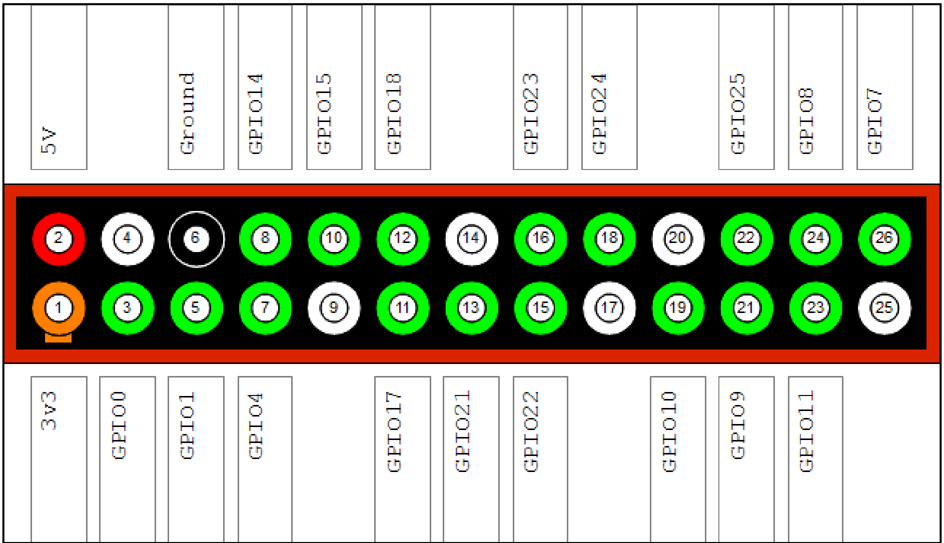


What are the GPIO Pins?

Visible in the two pictures to the left, the GPIO pins are located on the Raspberry Pi in the top left corner.

There are a total of 26 pins:

1x	**3.3V Power**
1x	**5V Power**
17x	**GPIO Pins**
7x	**Ground**



N.B. When we come to programming in a later lesson we will name the GPIO pins by their BCM number.

What is Python?

Python is a free to use programming language that runs on Windows, Linux/Unix, Mac OS X and has even been ported to Java and .NET virtual machines.

"Python is a programming language that lets you work more quickly and integrate your systems more effectively. You can learn to use Python and see almost immediate gains in productivity and lower maintenance costs."

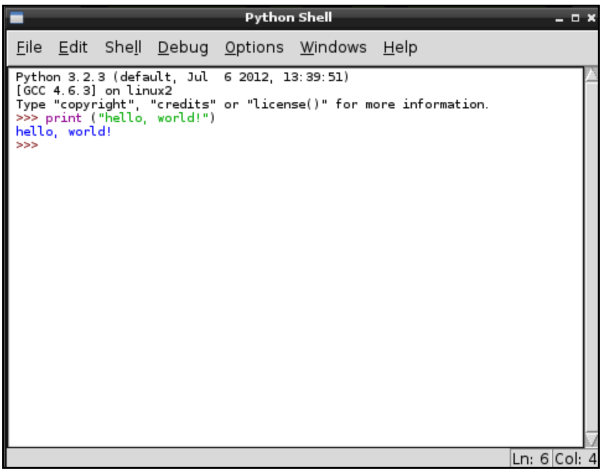
http://www.python.org

IDLE

In order to create our python-run programmes we will first need to write them. This is where IDLE comes in...

IDLE is a special text editor software — like Microsoft Word — however it understands the language Python. This enables us to write in a language that our Raspberry Pi will be able to understand and interpret.

N.B. This is what IDLE 3 looks like on a Raspberry Pi.

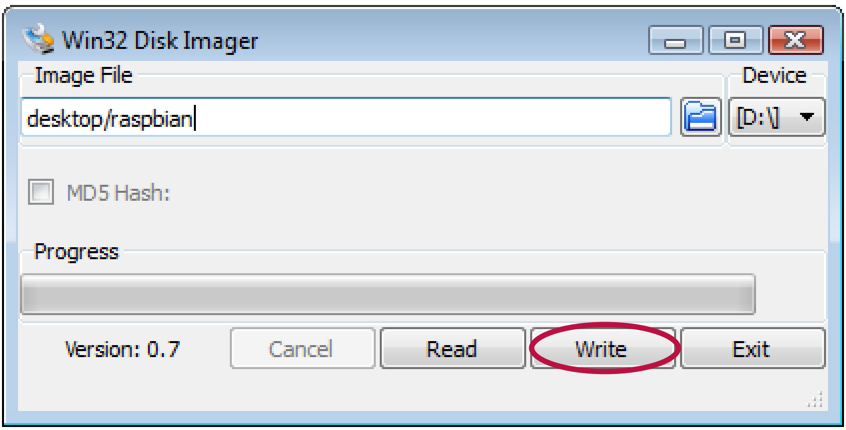


Installing 'Raspbian'

Raspbian is the desired operating system for the Raspberry Pi. In order to download and install the operating system onto our Raspberry Pi; you will need the following:

- Raspbian (http://www.raspberrypi.org/downloads)
- Win32DiskImager (http://sourceforge.net/projects/win32diskimager)
- A USB memory card reader

1. **Download both Raspbian and Win32DiskImager and save somewhere easily accessible**
2. **Plug the USB memory card reader into your computer**
3. **Open Win32DiskImager**
4. **Find the location of the image file and the memory card**
5. **Click "Write"**



Logging In

Now it is time to turn on our Raspberry Pi. When the memory card, HDMI lead, Ethernet cable, mouse and keyboard are plugged in, plug in the power lead.

As soon as you do this. You screen should be black and filled with white text. This will be visible every time you turn on your raspberry pi.

Wait until your screen reads "raspberrypi login:"

Username = pi	**[ENTER]**
Password = raspberry	**[ENTER]**

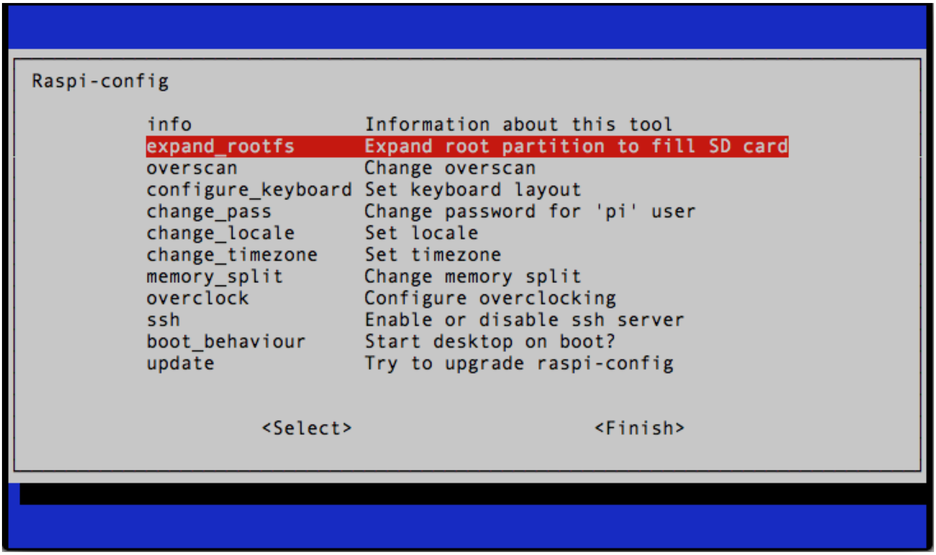
```
Debian GNU/Linux wheezy/sid raspberrypi tty1

raspberrypi login: pi
Password:
Last login: Tue Aug 21 21:24:50 EDT 2012 on tty1
Linux raspberrypi 3.1.9+ #168 PREEMPT Sat Jul 14 18:56:31 BST 2012 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.

Type 'startx' to launch a graphical session

pi@raspberrypi ~ $
```

Configuring Raspberry Pi

Unfortunately, unlike most operating systems the majority of the configurations for Raspbian have to be done manually.

One such as example that would normally be done automatically is expanding the partitions in your hard drive. This allows us to use all of the available space in the memory card.

1. Type the line: "sudo raspi-config"



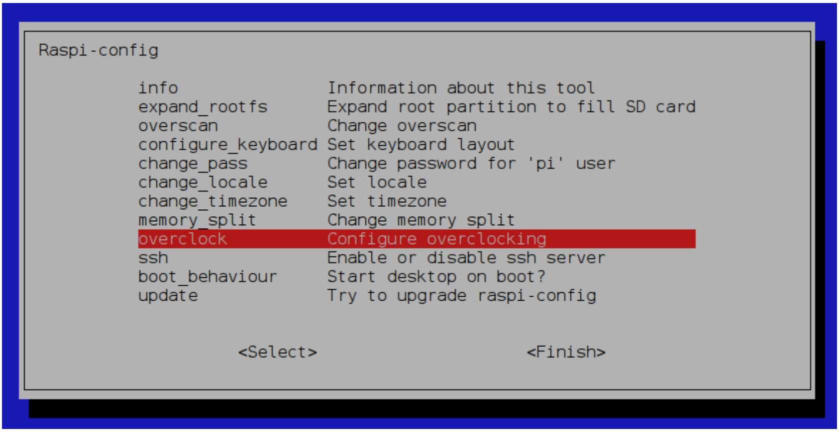
2. Use the arrow keys to move down to "expand_rootfs"

3. Press the enter button

Overclocking

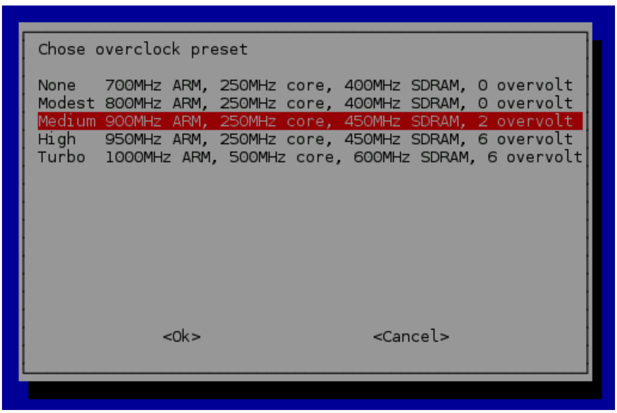
Overclocking is used to increase the capabilities of the hardware on the Raspberry Pi. When you overclock a machine you increase the amount of voltage travelling through the circuits and therefore increase both the processing speed and the RAM's speed.

1. Type the line: "sudo raspi-config"



2. Use the arrow keys to move down to "overclock" [ENTER]

3. Select "Medium" [ENTER]

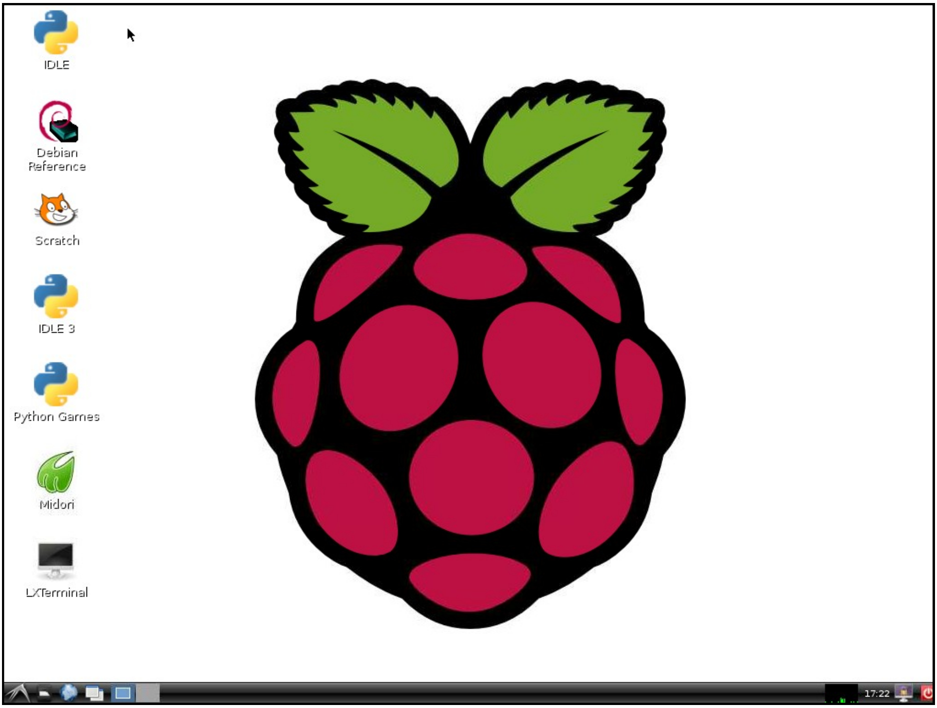


N.B. The higher the overclocking, the more likely you are to corrupt your memory card.

Starting the Raspbian GUI

GUI stands for Graphical User Interface and is a type of operating system. It is the most common type of user interface as it is a very 'friendly' way for people to interact with the computer. It makes use of pictures, graphics, icons and pointers — hence the name 'Graphical' User Interface.

1. Type the line: "startx"



Welcome to the Raspberry Pi

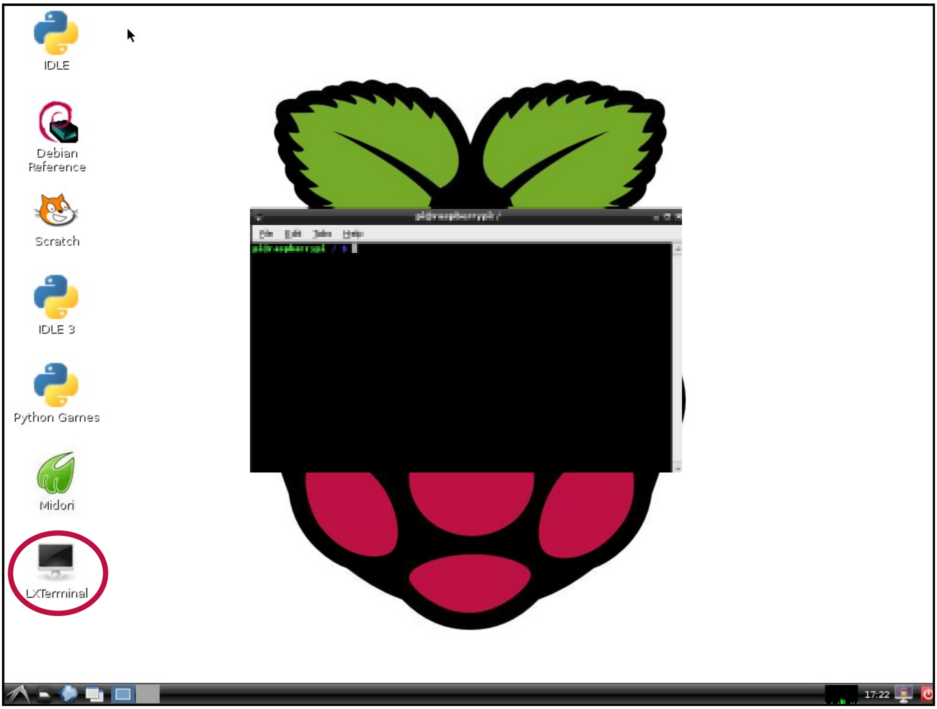
Terminal

Terminal in Raspbian is the equivalent of Command Prompt in Windows.

We will use Terminal to execute the programs that we will create later in this guide.

To run a program simply type: “python” + “ “ + (name of program)

To end a running program: Ctrl + C



Installing GPIO & Sound

We are about to install a number of plug-ins; one will allow the Raspberry Pi to understand signals from the GPIO pins, and the other two will enable the Raspberry Pi to output mp3 files and interpret sound.

1. **Open Terminal**
2. **Type the following exactly…**

GPIO

```
gunzip RPi.GPIO-0.4.2a.tar.gz
tar -xvf RPi.GPIO-0.4.2a.tar
sudo apt-get install python-dev
cd RPi.GPIO-0.4.2a
sudo python setup.py install
```

SOUND

```
sudo apt-get install mpg321
sudo apt-get install alsa-utils
```

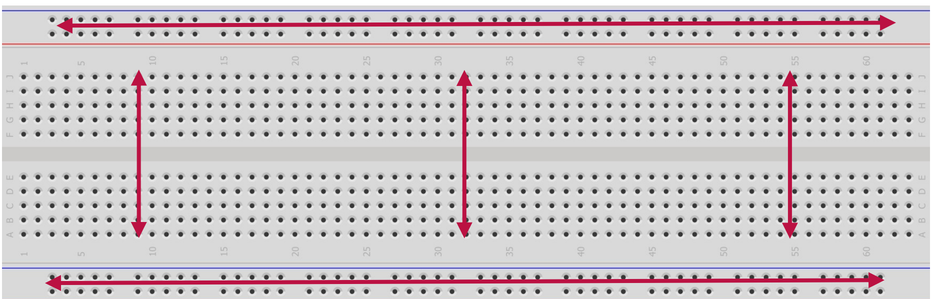
Lesson 2—"Light it Up"

In this lesson we are going to start using the basics of the GPIO pins and also take our first step into understanding the programming language python.

We are going to try and get a number of systems set up that incorporate a single LED. But first we will need to set up our breadboard.

What is a Breadboard?

A breadboard is used for making an experimental model of an electric circuit. We are going to be using a 840 contact solderless breadboard (plugboard). The reason for this is that you can simply push hardware in and out of the board without the need of soldering. This will ensure that if any mistakes are made you have not permanently damaged the hardware.



N.B. The current in the centre pins flows vertically, whereas the current in the pins at the top and bottom flows horizontally.

Setting Up

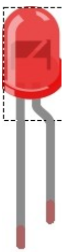
To set up for this lesson you will need:

x1 Red LED

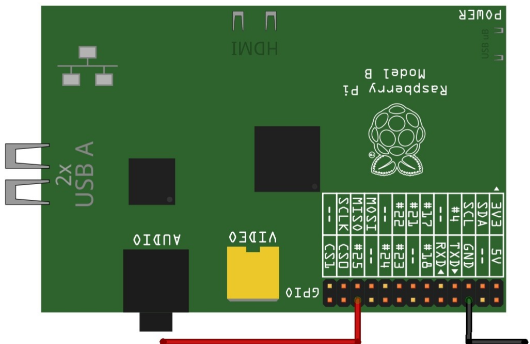
x1 13K Resistor (Brown, Orange, Orange, Gold)

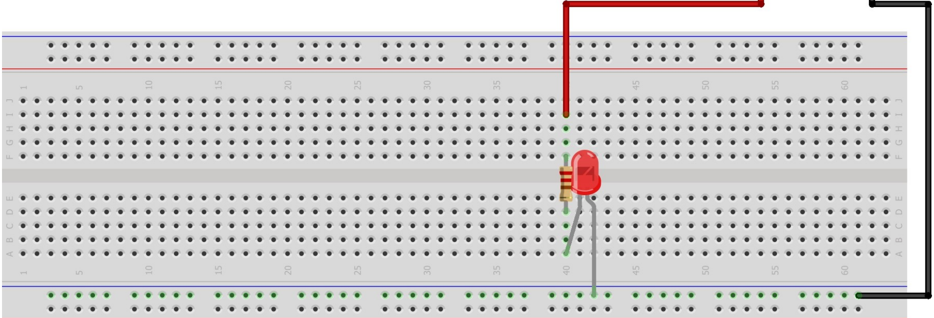
x2 Male to female jumper wire

Now set up the hardware as shown below.



N.B. The short pin of an LED is its negative and must be placed in the horizontally flowing current.





Flash.py – Tutorial

Importing:

The code "import" in python is used to introduce packages into your program. Because packages can be huge we store them in a separate location and use the import code to minimise the amount of code on your current project.

For Flash.py we will import

```
import RPi.GPIO as gpio

import time
```

Setup GPIO Pins:

To ensure that the Raspberry Pi understands whether a piece of hardware from the breadboard is an import or an export; we have to tell it.

For Flash.py we have one output — The LED

```
#name pins by their BCM number

gpio.setmode(gpio.BCM)

gpio.setup(25, gpio.OUT) #output from pin 25
```

Commenting:

#any text written after a "#" is known as a comment.

This text is not read by the computer; it is there simply to try and explain what the written code is doing to any human reading it.

While Loops:

Loops are one of the most useful tools available when creating programs. Computers are great at repetitive tasks, as unlike humans; they don't get bored. As a result you can program a computer to repeat a task while something is true and then stop when it become false.

For Flash.py however the loop will always be True.

```
while True: #forever loop
```

Indents:

Now that we have created our loop, lets put something in it! In order to place something within a loop in Python we must use indents.

> **Indents are essential in Python as they tell the computer what is or is not in the loop.**

For Flash.py lets turn the LED on and then off

```
    #turns light on for 1 second
    gpio.output(25, gpio.HIGH)
    time.sleep(1)
    #turns light off for 1 second
    gpio.output(25, gpio.LOW)
    time.sleep(1)
```

Flash.py – Completed Code

```
import RPi.GPIO as gpio
import time

#name pins by their BCM number
gpio.setmode(gpio.BCM)
gpio.setup(25, gpio.OUT) #output from pin 25

while True: #forever loop

    #turns light on for 1 second
    gpio.output(25, gpio.HIGH)
    time.sleep(1)
    #turns light off for 1 second
    gpio.output(25, gpio.LOW)
    time.sleep(1)
```

<u>Saving:</u>

In order for the Raspberry Pi to understand your program as a python run program you need to name it with the '.py' tag on the end. Each of the titles in this guide have suitable names for each of the programs; however if you wish to name them yourself just ensure that you name each one with the '.py' tag.

Now run Flash.py in Terminal (see page 14).

To run a program simply type: "python" + " " + (name of program)

To end a running program: Ctrl + C

Variableflash.py – Tutorial

```
import RPi.GPIO as gpio
import time
#setup pin 25 as an output
gpio.setmode(gpio.BCM)
gpio.setup(25, gpio.OUT)
```

Keyboard Input:

Coding user inputs are surprisingly easy in Python but are once again are very useful. These inputs allows the user to interact with the computer and change the conditions of the program.

For Variableflash.py we will input the on and off time for the LED

```
on = raw_input("Enter on time: ") #x = input
off = raw_input("Enter off time: ") #y = input
```

Numeric Types:

Numeric types are what are used so that the computer understands what type of number it is reading

In Python there are 4 Numeric Types:

Int	**Integers (Whole Numbers)**
Float	**Decimals**
Long	**Recurring Decimals**
Complex	**Imaginary Roots**

Variableflash.py – Completed Code

```
import RPi.GPIO as gpio
import time

#name pins by their BCM number
gpio.setmode(gpio.BCM)
gpio.setup(25, gpio.OUT) #output from pin 25

on = raw_input("Enter on time: ") #x = input
off = raw_input("Enter off time: ") #y = input

while True: #forever loop
    #turns light on for 1 second
    gpio.output(25, gpio.HIGH)
    time.sleep(float(on))
    #turns light off for 1 second
    gpio.output(25, gpio.LOW)
    time.sleep(float(off))
```

SOS.py – Tutorial

Morse Code

Morse Code was developed to transmit text via a series of on-off tones, lights or clicks.

Each character is represented by a unique sequence of dots and dashes or 'Dits' and 'Dahs'.

The best know message to transmit is "SOS" - the international distress signal.

Lets try this using Python...

Morse Code Alphabet

The International morse code characters:

A	.-	N	-.	0	-----
B	-...	O	---	1	.----
C	-.-.	P	.--.	2	..---
D	-..	Q	--.-	3	...--
E	.	R	.-.	4	....-
F	..-.	S	...	5	.....
G	--.	T	-	6	-....
H	....	U	..-	7	--...
I	..	V	...-	8	---..
J	.---	W	.--	9	----.
K	-.-	X	-..-	Fullstop	.-.-.-
L	.-..	Y	-.--	Comma	--..--
M	--	Z	--..	Query	..--..

For Loops:

We have already looked at While Loops and discovered that we can create a 'forever loop'. However if we wish to create a loop with a limit then we use a For loop.

For SOS.py we will use it to generate the Morse code for each letter

```
for i in range(3): #repeat x3
```

SOS.py – Completed Code

```
import RPi.GPIO as gpio
import time
gpio.setmode(gpio.BCM)
gpio.setup(25, gpio.OUT) #output from pin 25
while True: #forever loop
    for i in range(3): #short burst - repeat x3
        gpio.output(25, gpio.HIGH)
        time.sleep(0.2)
        gpio.output(25, gpio.LOW)
        time.sleep(0.2)
    time.sleep(0.6) #delay time between letter
    for i in range(3): #long burst - repeat x3
        gpio.output(25, gpio.HIGH)
        time.sleep(0.6)
        gpio.output(25, gpio.LOW)
        time.sleep(0.2)
    time.sleep(0.6) #delay time between letter
    for i in range(3): #short burst - repeat 3 times
        gpio.output(25, gpio.HIGH)
        time.sleep(0.2)
        gpio.output(25, gpio.LOW)
        time.sleep(0.2)
    time.sleep(0.6) #delay time between letter
    time.sleep(1.4) #delay time between word
```

Morsecode.py – Tutorial

Upper Case Tag:

The '.upper()' tag convers all of the characters of any string into upper case characters.

```
message = message.upper()
```

Length Function:

We use to the 'Len' function to count the length of something. This could be used to count the number of characters in a string or even count the number of items in a list.

```
lengthy = len(message)
```

Output:

Coding outputs in Python is just as simple as coding inputs. We use outputs to return anything; could be a message or something that the computer has calculated.

For Morsecode.py we return each character of the total message

```
print (message[k]) #displays message
```

IF Statement (Equal To):

IF Statements are used to check whether something is True or not. They are equally as important as loops and enable the user to program a question into the system. Such as does x = y?

For Morsecode.py we will use it to convert a letter to Morse Code

```
if message[k] == "A":
     letter = ".-"
```

Morsecode.py – Completed Code

```
import RPi.GPIO as gpio
import time
gpio.setmode(gpio.BCM)
gpio.setup(25, gpio.OUT) #output from pin 25
while True: #forever loop
    message = raw_input("Enter message: ")
    message = message.upper() #converts to upper case
    lengthy = len(message)
    for k in range(0,lengthy):

        print (message[k]) #displays message

        if message[k] == "A":
            letter = ".-"
        elif message[k] == "B":
            letter = "-..."
        elif message[k] == "C":
            letter = "-.-."
        elif message[k] == "D":
            letter = "-.."
        elif message[k] == "E":
            letter = "."
        elif message[k] == "F":
            letter = "..-."
```

```
        elif message[k] == "G":
            letter = "--."
        elif message[k] == "H":
            letter = "...."
        elif message[k] == "I":
            letter = ".."
        elif message[k] == "J":
            letter = ".---"
        elif message[k] == "K":
            letter = "-.-"
        elif message[k] == "L":
            letter = ".-.."
        elif message[k] == "M":
            letter = "--"
        elif message[k] == "N":
            letter = "-."
        elif message[k] == "O":
            letter = "---"
        elif message[k] == "P":
            letter = ".--."
        elif message[k] == "Q":
            letter = "--.-"
        elif message[k] == "R":
            letter = ".-."
        elif message[k] == "S":
            letter = "..."
```

```
        elif message[k] == "T":
            letter = "-"
        elif message[k] == "U":
            letter = "..-"
        elif message[k] == "V":
            letter = "...-"
        elif message[k] == "W":
            letter = ".--"
        elif message[k] == "X":
            letter = "-..-"
        elif message[k] == "Y":
            letter = "-.--"
        elif message[k] == "Z":
            letter = "--.."
        elif message[k] == " ":
            letter = ""
        print (letter) #displays character
        lengthyy = len(letter)

        for t in range(0,lengthyy):
             if letter[t] == ".": #dit = short flash
                gpio.output(25, gpio.HIGH)
                time.sleep(0.2)
                gpio.output(25, gpio.LOW)
                time.sleep(0.2)
```

```
        elif letter[t] == "-": #dah = long flash
            gpio.output(25, gpio.HIGH)
            time.sleep(0.6)
            gpio.output(25, gpio.LOW)
            time.sleep(0.2)
        else:
            time.sleep(1.4) #delay time after word

    time.sleep(0.6) #delay time after letter
```

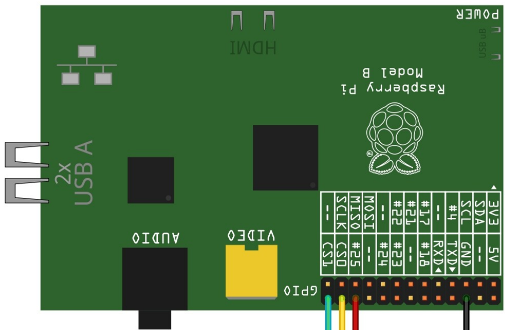
Lesson 3—"Button It"

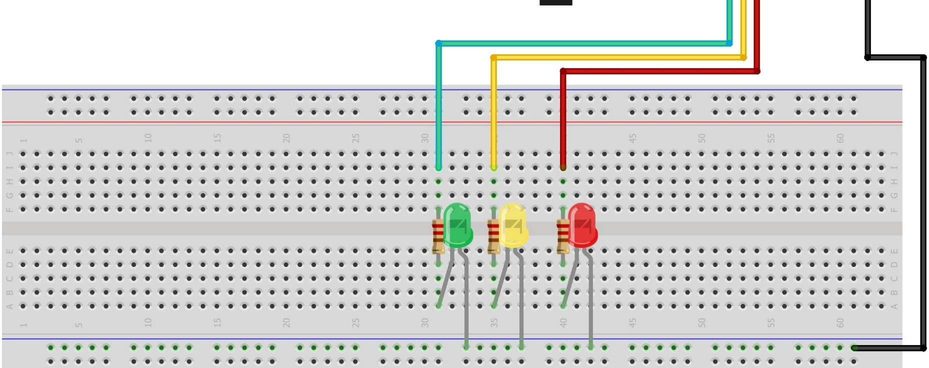
Now that we have learnt the fundamentals of Python, lets use what we have learnt in more complicated scenarios; with inputs and multiple outputs.

To set up for this lesson you will need:

- **x3 Red LED**
- **x2 Yellow LED**
- **x3 Green LED**
- **x10 Male to female jumper wire**
- **x1 Male to Male jumper wire**
- **x8 13K Resistor (Brown, Orange, Orange, Gold)**
- **x1 Push Button**

Now set up the hardware as shown below.





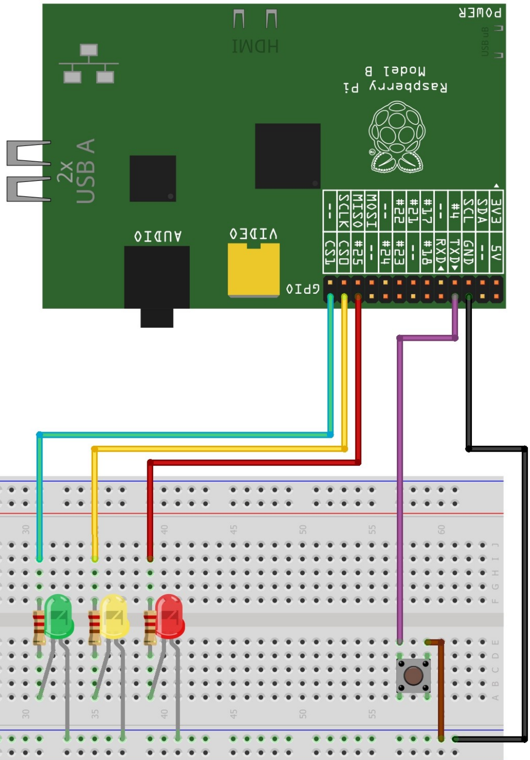
Multipleflash.py – Tutorial & Complete Code

In order to create this program no more further knowledge of Python is required. We are simply using the same material to controlling three LEDs as we used to control one LED.

```
import RPi.GPIO as gpio

import time

#setup pin 7, 8 & 25 as outputs

gpio.setmode(gpio.BCM)

gpio.setup(7, gpio.OUT)

gpio.setup(8, gpio.OUT)

gpio.setup(25, gpio.OUT)

while True: #forever loop

     #turns light on for 0.5 seconds

     gpio.output(7, gpio.HIGH)

     time.sleep(0.5)

     gpio.output(7, gpio.LOW) #turns light off


     #turns light on for 0.5 seconds

     gpio.output(8, gpio.HIGH)

     time.sleep(0.5)

     gpio.output(8, gpio.LOW) #turns light off


     #turns light on for 0.5 seconds

     gpio.output(25, gpio.HIGH)

     time.sleep(0.5)

     gpio.output(25, gpio.LOW) #turns light off
```

Mybutton.py – Tutorial

Now add the push button to your already existing hardware.



Hardware Inputs:

We have already learnt about Keyboard Inputs in a previous lesson, and hardware inputs are pretty much identical. The only difference is that in order for the Raspberry Pi to understand the hardware as an input; you have to state it when you set up the GPIO pins.

```
GPIO.setup(14,GPIO.IN)

input = GPIO.input(14)
```

Mybutton1.py – Complete Code

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(14,GPIO.IN)
input = GPIO.input(14)

while True:
    if GPIO.input(14) == True:
        print("Button Pressed")
```

You should notice that after running this program and after pressing the push button that tens of "Button Pressed" are displayed on the screen.

This is because within the button there is a tiny spring. After pressing the button the first time the spring will continue to bounce for quite some time. The system will register each bounce as a separate button press.

This is not what we want and therefore a new type of IF Statement is required.

Mybutton2.py – Tutorial & Complete Code

IF Statement (Great Than/Less Than):

We have already learnt that IF Statements can be used check whether something is equal to something else. However IF Statements also work for other mathematical symbols; such as greater than (>) and less than (<)

```
if prev_input < input:
```

For Mybutton2.py we will use it to stop the system from registering the spring bouncing as multiple button presses.

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(14,GPIO.IN)

prev_input = 0
while True:
    input = GPIO.input(14)
    if prev_input < input:
        print("Button Pressed")
    prev_input = input
    time.sleep(0.5)
```

Mybutton3.py – Tutorial & Complete Code

Calling Programs:

The term ‘calling’ refers to running a piece of code from a different location in a program. This is used to stop the repeating of code in; instead you can just “call” the code using one line.

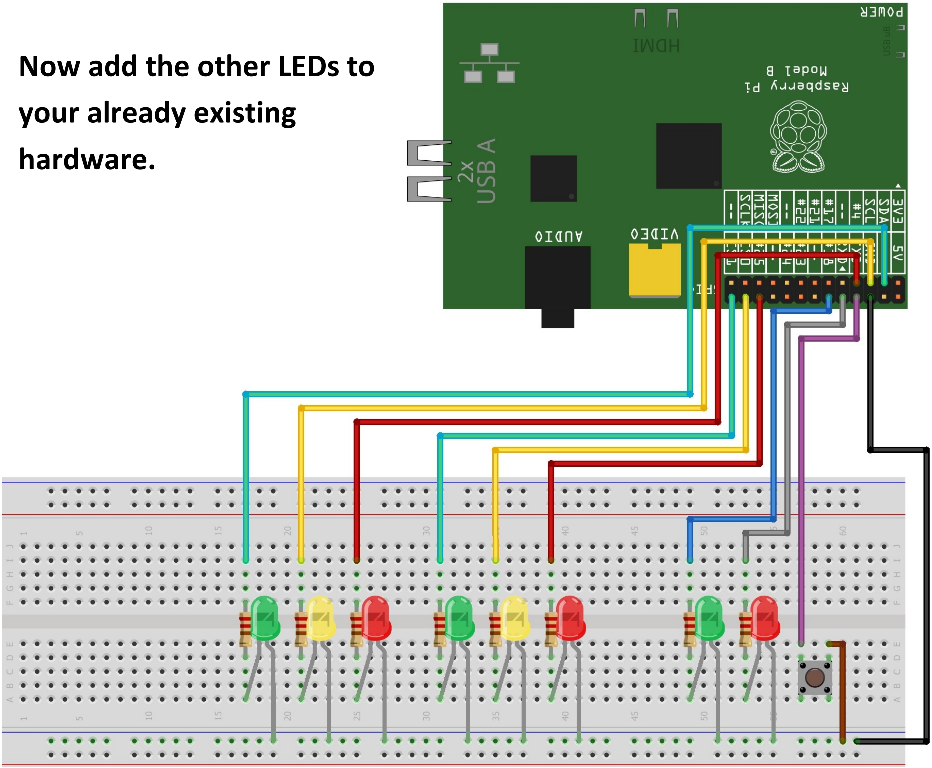
For Mybutton3.py we will call a program that we created earlier “multipleflash.py”.

```
import RPi.GPIO as GPIO
import time
import os

GPIO.setmode(GPIO.BCM)
GPIO.setup(14,GPIO.IN)

while True:
    if (GPIO.input(14)):
        #call multipleflash.py program
        os.system("python /home/pi/
                                multipleflash.py")
```

Have a try calling some of the other programs we created earlier.

Trafficlight.py – Tutorial

Now add the other LEDs to your already existing hardware.



GPIO Reset:

In order to reset all of the GPIO pins; turning any pins that is already turned on to off; simply use the line:

```
gpio.cleanup()
```

CHALLENGE

Using the knowledge that you have already learnt. Develop a traffic light system that cycles through each lane of traffic and has the ability to allow pedestrians to cross safely.

Trafficlight.py – Complete Code

```
import time
import RPi.GPIO as gpio
gpio.setmode(gpio.BCM)
gpio.cleanup()
gpio.setup(7, gpio.OUT) # green LED side 1
gpio.setup(8, gpio.OUT) # Yellow LED side 1
gpio.setup(25, gpio.OUT) # Red LED side 1
gpio.setup(0, gpio.OUT) # green LED side 2
gpio.setup(1, gpio.OUT) # Yellow LED side 2
gpio.setup(4, gpio.OUT) # Red LED side 2

gpio.setup(14, gpio.IN) # Crosswalk Button
gpio.setup(18, gpio.OUT) # Crosswalk green LED
gpio.setup(15, gpio.OUT) # Crosswalk red LED

carA = 0
flashA = 0

while True:
    if gpio.input(14) == True:
        carA = 1
    else:
        carA = 0
```

```
if carA == 0:
    print("Light cycle")

    #Light control cycle
    time.sleep(2)
    gpio.output(15, gpio.HIGH)
    gpio.output(18, gpio.LOW)

    gpio.output(0, gpio.LOW)
    gpio.output(25, gpio.LOW)
    gpio.output(4, gpio.HIGH) # turn red(2) on
    time.sleep(1)
    gpio.output(7, gpio.HIGH) # turn green(1) on
    while flashA < 5:
        gpio.output(8, gpio.HIGH)
        time.sleep(0.5)
        gpio.output(8, gpio.LOW)
        time.sleep(0.5)
        flashA = flashA + 1
    flashA = 0
    gpio.output(8, gpio.LOW)
    time.sleep(10)
    gpio.output(7, gpio.LOW)
    gpio.output(4, gpio.LOW)
    gpio.output(25, gpio.HIGH) # turn red(1) on
```

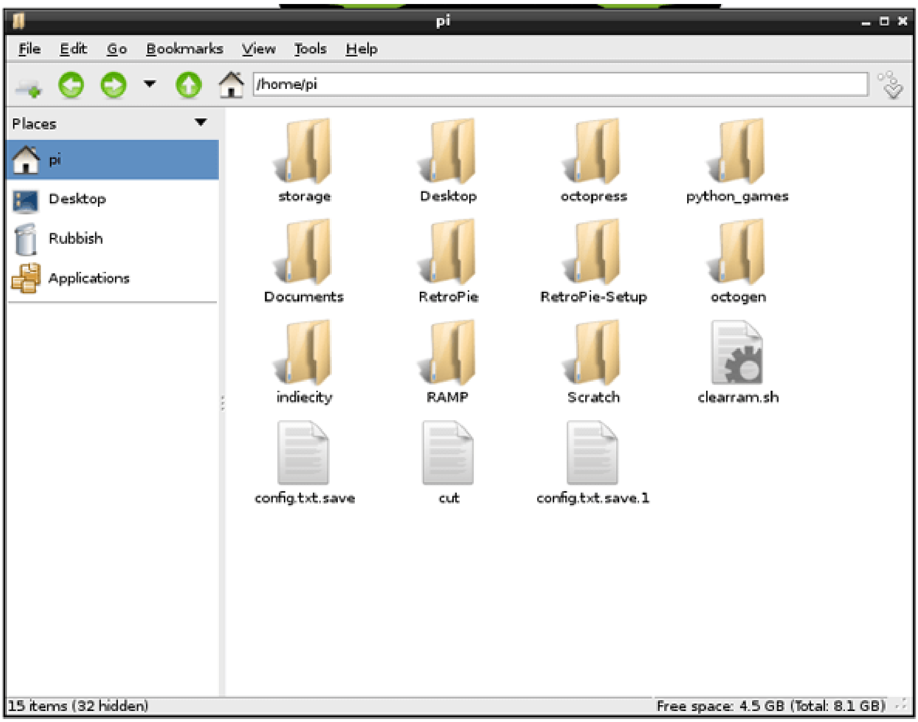
```
    time.sleep(1)
    gpio.output(0, gpio.HIGH) # turn green(2) on
        while flashA < 5:
            gpio.output(1, gpio.HIGH)
            time.sleep(0.5)
            gpio.output(1, gpio.LOW)
            time.sleep(0.5)
            flashA = flashA + 1
        flashA = 0
        gpio.output(1, gpio.LOW)
        time.sleep(8)
    if carA == 1:
        print("Crosswalk")
        #Crosswalk control cycle
        gpio.output(7, gpio.LOW)
        gpio.output(8, gpio.LOW)
        gpio.output(25, gpio.HIGH)
        gpio.output(0, gpio.LOW)
        gpio.output(1, gpio.LOW)
        gpio.output(4, gpio.HIGH)
        time.sleep(1)

        gpio.output(18, gpio.HIGH)
        gpio.output(15, gpio.LOW)
        time.sleep(8)
```

Lesson 4—"Sound & SSH"

Playing a mp3 music file on a Raspberry Pi is not as simple as just double clicking on the file. This is because the Raspberry Pi does not recognise the file unless you run it through the 'mpg321' plugin. Creating a program to do this however is surprisingly simple.

Besides the 'mpg321' plugin which you should have already installed on page 15; no further software or hardware is required.

What you will need however is to create a new folder called "music" under the /home/pi/ directory and populate that folder with any music that you wish.



Jukebox.py – Tutorial

Glob:

Glob is a very useful command and can be used to list all of the files in a chosen directory. If there is a specific type of file that you are looking for then you can program the glob to command to only return your specific file type.

For Jukebox.py we will use it to list all of the mp3 files in our music folder:

```
print glob.glob("/home/pi/music/*.mp3")
```

Root Directory **File Type**

```
song = raw_input("What do you want to play: ")
```

Mpg321:

Mpg321 is an external plugin which we call using the 'os.system' (see page 36). Mpg321 allows the Raspberry Pi to understand, read and output mp3 files as audible sound via the 3.5mm jack socket.

```
os.system("mpg321 /home/pi/music/" + song + ".mp3")
```

Call Software **Root Directory** **Song name** **File Type**

Jukebox.py – Completed Code

```
import os
import glob


while True:
    print glob.glob("/home/pi/music/
*.mp3")
    song = raw_input("What do you want
to play: ")
os.system("mpg321 /home/pi/music/" +
song + ".mp3")
```

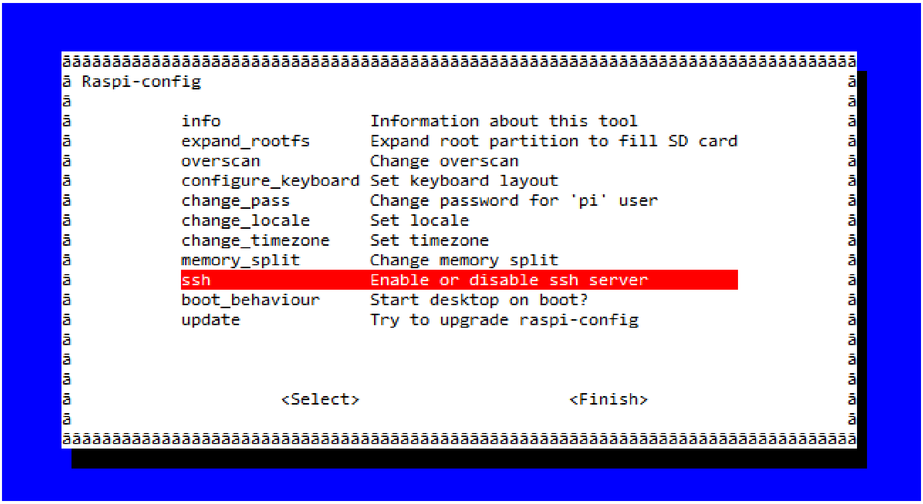
SSH

SSH stands for secure shell and is used to control devices remotely from another computer. This means that we will be able to control our Raspberry Pi via the internet from another computer.

First we need to ensure that SSH is enabled on our Raspberry Pi.

We do this via ‘raspi-config’:

1. **Open Terminal**
2. **Type the code ‘sudo raspi-config’**



3. **Use the arrow keys to find ‘SSH’ [ENTER]**
4. **Click Enable [ENTER]**
5. **Click Finish [ENTER]**

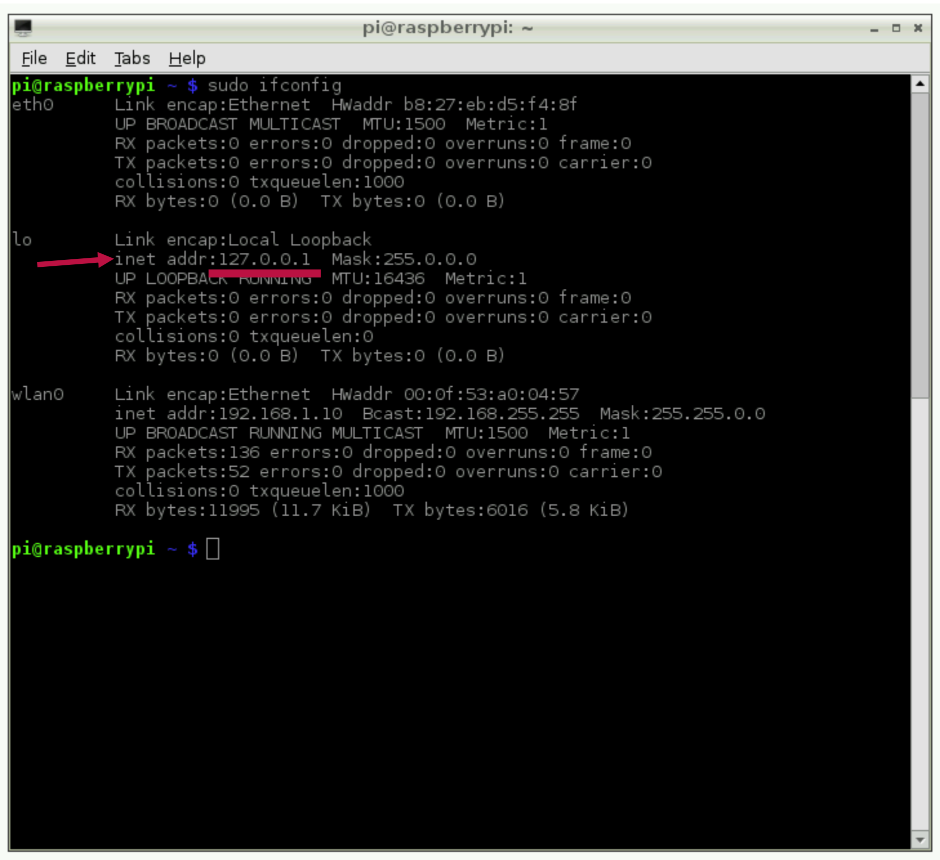
Next we need to find out the IP address for our Raspberry Pi.

1. **Open Terminal**
2. **Type the code 'sudo ifconfig'**

A lot of text will now be visible on the screen however we are only interested in the line that begins with inet addr:

After inet addr: there will be a series of numbers — this is the IP address for the Raspberry Pi.

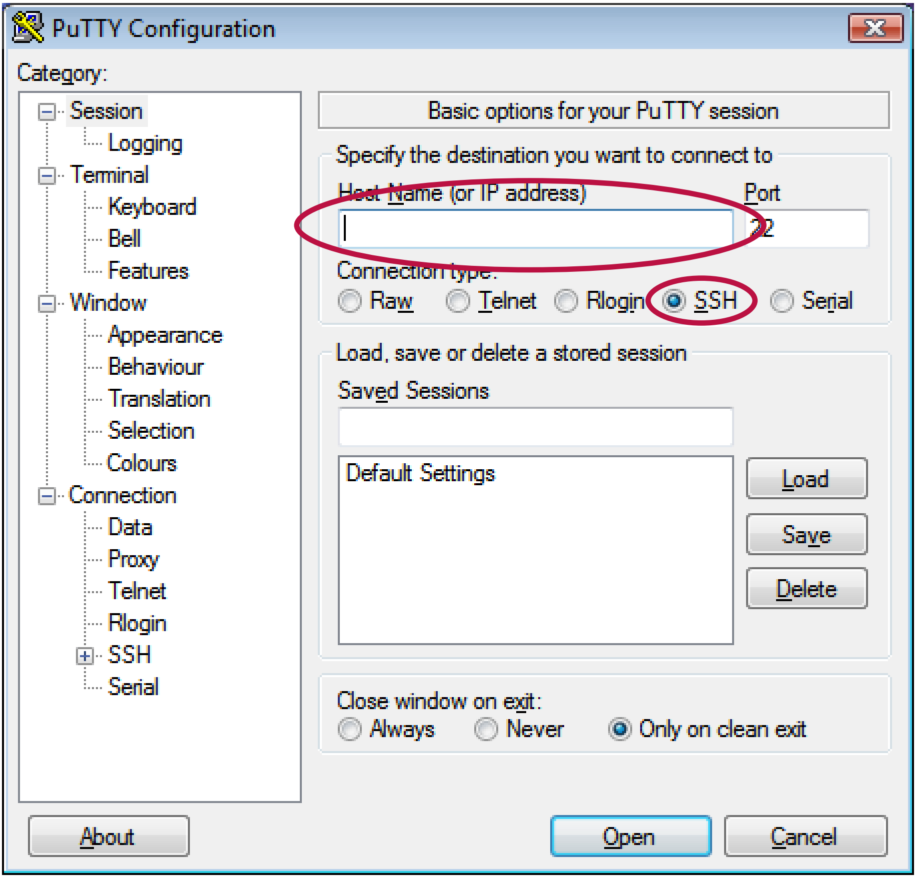
REMEMBER THE IP ADDRESS



In order to form the link between the Raspberry Pi and your desktop computer you will need two additional pieces of software.

1. **PuTTY (http://putty.en.softonic.com/)**
2. **Xming (http://sourceforge.net/projects/xming)**

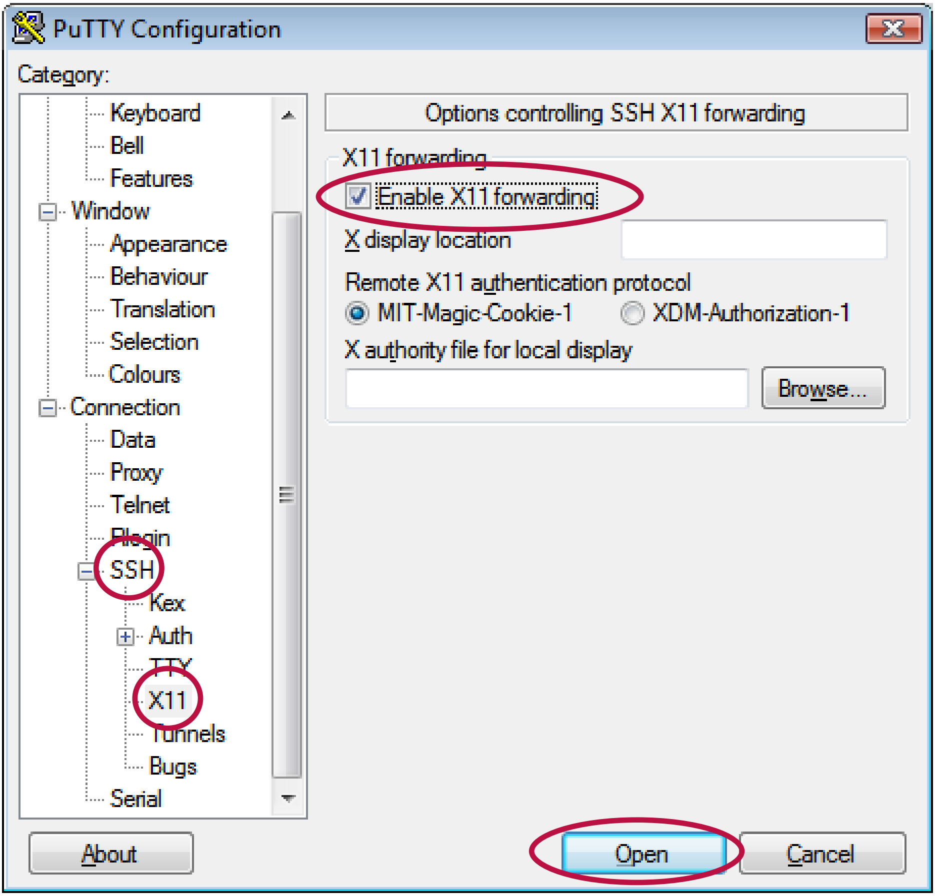
1. **Enter the IP address from the Raspberry Pi**
2. **Check that Connection Type = SSH**



2. Expand “SSH”

3. Click on “X11”

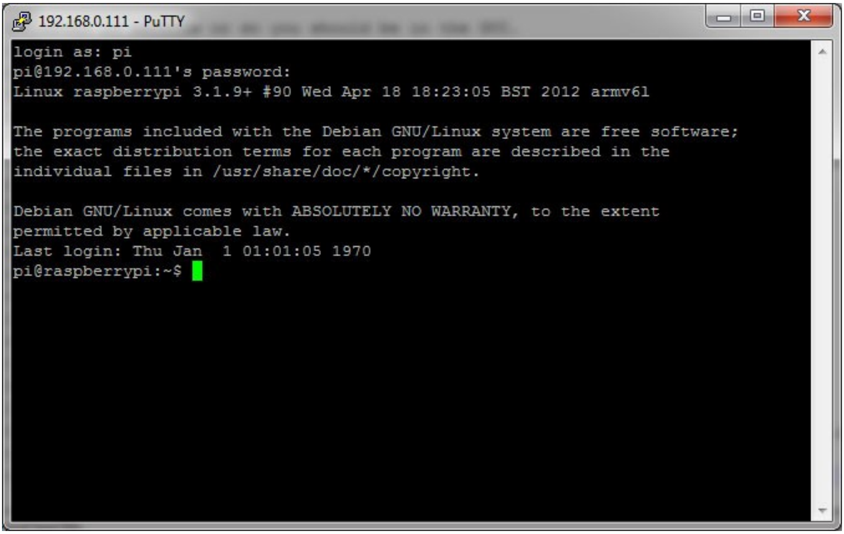
4. Tick “Enable X11 forwarding”



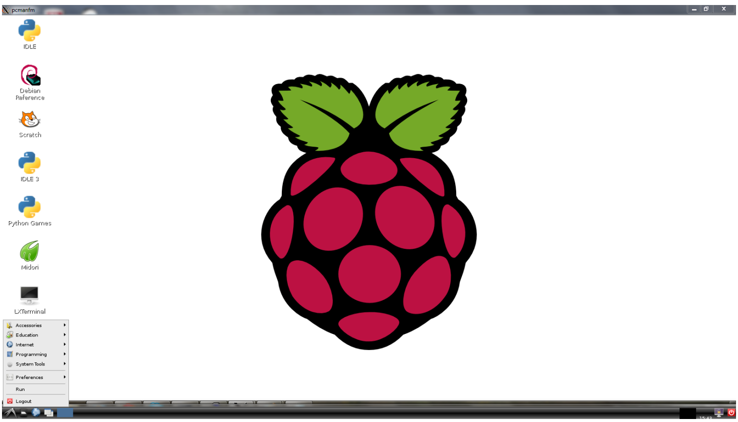
5. Click ‘Open’

7. Enter the username and password as if you were logging into the Raspberry Pi.

8. Type the code ‘startlxde’



9. Return to Xming and you should see the Raspberry Pi desktop.



Now try and play some music off your Raspberry Pi from you desktop computer.

Lesson 5—"ADC"

In this lesson we are going to try and measure them temperature of the room using a temperature sensor. However; the GPIO pins on a Raspberry Pi only read in digital signals and unfortunately temperature must be read as an analogue signal. As a result we need an Analogue to Digital Converter.

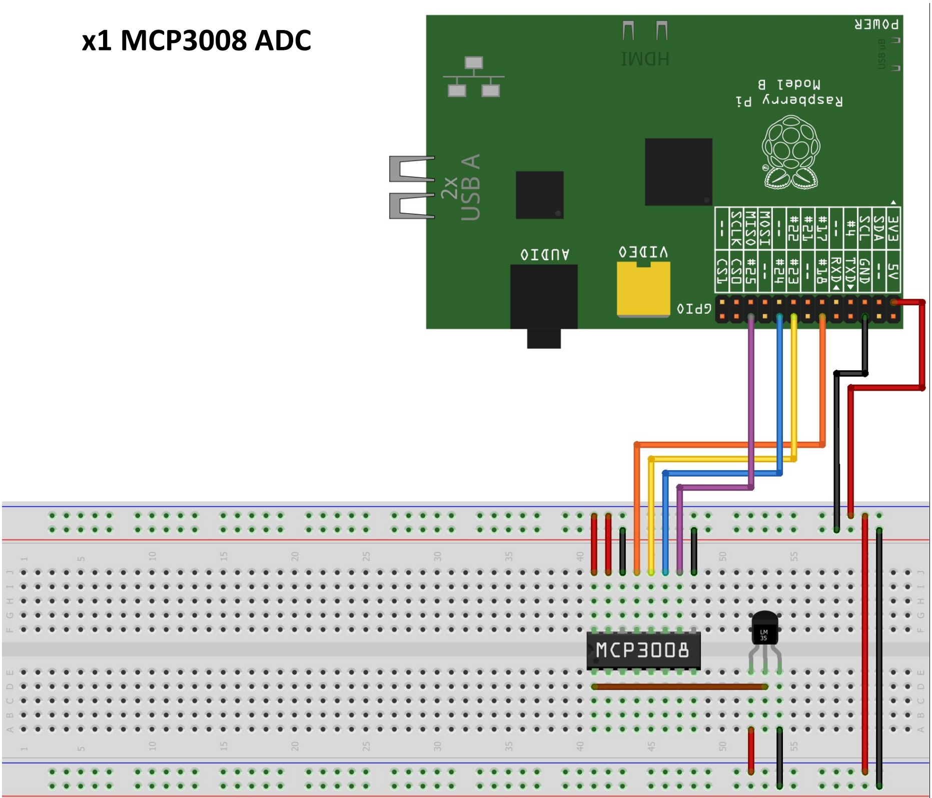
To set up for this lesson you will need:

x6 Male to female jumper wire

x9 Male to Male jumper wire

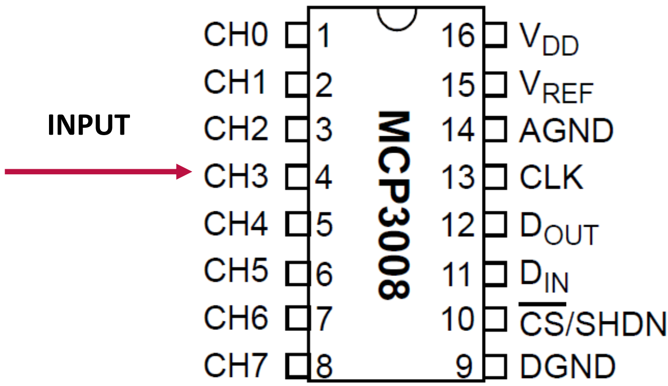
x1 TMP36 Temperature Sensor

x1 MCP3008 ADC

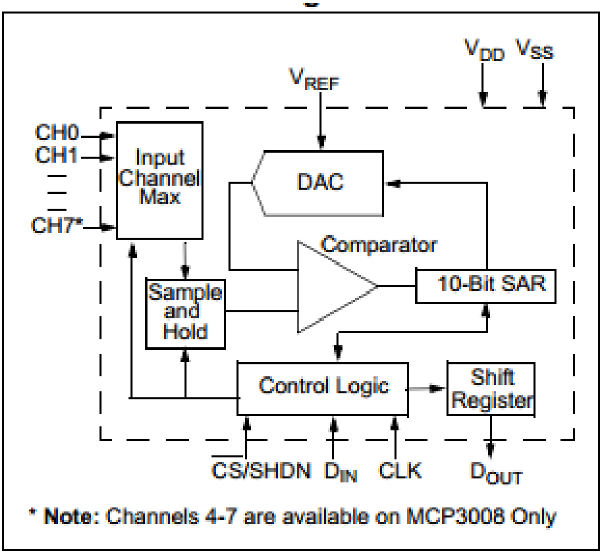


Temperaturesensor.py – Tutorial

ADC stands for Analogue to Digital Converter. It is used to convert analogue information into digital 'binary' information. ADCs are used in thousands of devices such as microphones (converting analogue sound waves into digital waves) and in most sensors. The ADC that we are going to be using in order to measure the temperature of the room is an MCP3008.



We will not go into detail exactly how the data is converted, But this is a map of the internal structure of the MCP3008.



Temperaturesensor.py – Completed Code

```
import time

import os

import RPi.GPIO as GPIO

import sys

GPIO.setmode(GPIO.BCM)



# read SPI data from MCP3008 chip, 8 possible ADC's (0
to 7)

def readadc(adcnum, clockpin, mosipin, misopin, cspin):

        if ((adcnum > 7) or (adcnum < 0)):

                return -1

        GPIO.output(cspin, True)

        GPIO.output(clockpin, False)  # start clock low

        GPIO.output(cspin, False)     # bring CS low



        commandout = adcnum

        commandout |= 0x18 # start bit + single-ended
bit

        commandout <<= 3   # we only need to send 5
bits here
```

```
    for i in range(5):
        if (commandout & 0x80):
            GPIO.output(mosipin, True)
        else:
            GPIO.output(mosipin, False)
        commandout <<= 1
        GPIO.output(clockpin, True)
        GPIO.output(clockpin, False)

    adcout = 0
    # read in one empty bit, one null bit and 10 ADC bits
    for i in range(12):
        GPIO.output(clockpin, True)
        GPIO.output(clockpin, False)
        adcout <<= 1
        if (GPIO.input(misopin)):
            adcout |= 0x1

    GPIO.output(cspin, True)

    adcout >>= 1       # first bit is 'null' so drop it
    return adcout
```

```
# change these as desired - they're the pins connected
from the
# SPI port on the ADC to the Cobbler
SPICLK = 18
SPIMISO = 23
SPIMOSI = 24
SPICS = 25
# set up the SPI interface pins
GPIO.setup(SPIMOSI, GPIO.OUT)
GPIO.setup(SPIMISO, GPIO.IN)
GPIO.setup(SPICLK, GPIO.OUT)
GPIO.setup(SPICS, GPIO.OUT)
# 10k trim pot connected to adc #0
potentiometer_adc = 0;

while True:
     # read the analogue pin
     value = readadc(potentiometer_adc, SPICLK, SPIMOSI,
SPIMISO, SPICS)

     print "%s,%s,%s" % (time.time(), value,
100.*3.3*value/1024.-50.)
     sys.stdout.flush()

     time.sleep(5)
```

Lesson 6—"Games"

Minecraft

Minecraft is a indie game that was created by one Swedish programmer Markus "Notch" Persson. Since being developed by 'Mojang' and being released for PC on the 18th November 2011 it has sold over 10 million copies. Over time Minecraft has been released for other gaming platforms, but only until recently — 11th February 2013 — has it become available for the Raspberry Pi.

We are about to download and install the pi edition of Minecraft onto the Raspberry Pi.

1. **Open Terminal**
2. **To download the files—type the code**

   "wget https://dl.dropbox.com/s/hqk8wsdzlyyujli/minecraft-pi-0.1.tar.gz "
3. **Find the files in the '/home/pi' directory**
4. **Right click on the root folder and select 'Xarchiver'**

5. Select 'mcpi' -> right click -> Extract

Extract to:	'/home/pi/Desktop'
Files:	'All Files'
Options:	- Overwrite Existing Files
	- Extract files with full patch

6. Click 'Extract'
7. Return to Terminal

<u>Changing directory:</u>

In the past we have always run a program from the '/home/pi' directory. However if we want to execute a program from within a different directory; we have to change our current directory in terminal.

In order to do this type 'cd' followed by the name of the directory.

8. For Minecraft type -> 'cd Desktop'

   'cd mcpi'

N.B. This can also be written as:

'cd Desktop/mcpi'

N.B. If the name of the directory has any spaces. 'Raspberry Pi' Then speech marks must be used. 'cd "Raspberry Pi"'

9. Type the code './Minecraft-pi'
10. To move back a directory; type the code 'cd ..'

Quake 3–N.B. This installation will take over an hour

Quake 3 is the 'Call of Duty' of the year 2000. Originally released for Windows on the 2nd December 1999; since then has become a classic in the gaming world. Quake 3 is a multiplayer first-person shooter video game placed in a futuristic world.

We are about to download and install the Quake 3 onto the Raspberry Pi.

1. **Open Terminal**
2. **'sudo apt-get install git gcc build-essential libsdl1.2-dev'**

Now we install the Quake 3 source code:

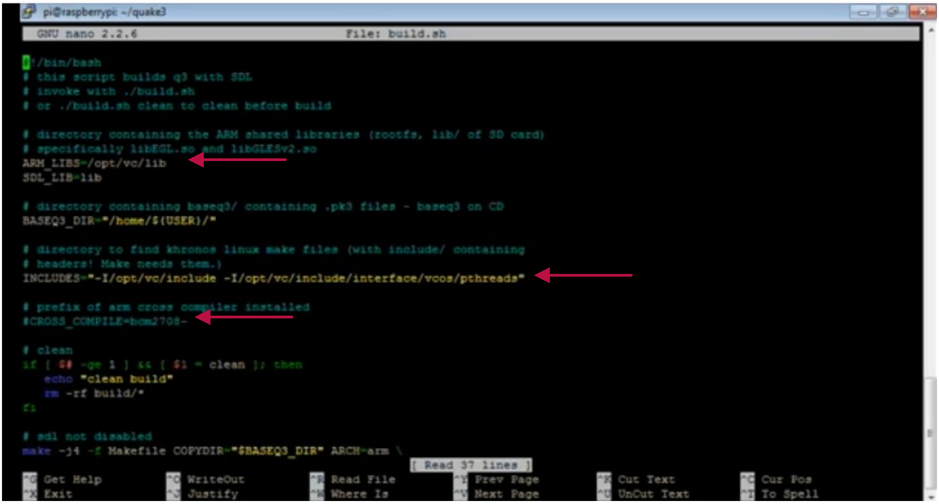
3. **'git clone https://github.com/raspberrypi/quake3.git '**

Change Directory:

4. **'cd quake3'**

Edit Script that alternates the compiling process:

5. **'nano build.sh'**



6. Edit line 8, 16 and 19.

(8) = ARM_LIBS=/opt/vc/lib

(16) = INCLUDES="-I/opt/vc/include -I/opt/vc/include/interface/vcos/pthreads"

(19) = #CROSS_COMPILE=bcm2708-

7. **Ctrl + X**

Now we start the compiling process: (This will take about an hour)

8. './build.sh'

Change Directory:

9. **'cd build'**

   'cd release-linux-arm'

   'cd baseq3'

Download Pak Files: (Must be saved in 'baseq3' directory)

10. **'wget http://dl.dropbox.com/u/1816557/Q3%20Demo%20Paks.zip'**

Unzip Pak Files:

11. **'unzip Q3\Demo\Paks.zip'**

Change Directory:

12. **'cd ..'**

Must be in 'quake3/build/release-linux-arm' directory

Run the game:

13. **sudo ./ioquake3.arm'**