WORK SHOP ON RASPBERRY PI

Course Handouts

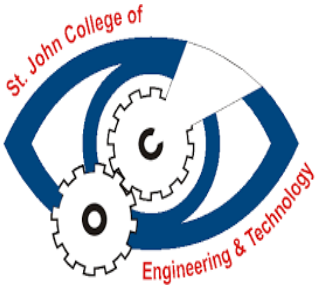
- **MS. DHVANI SHAH**

[Assistant Professor IT Department]

Organised by

Department of Information Technology

St. John College Of Engineering and Technology, Palghar



St. John College Of Engineering and Technology

Palghar-401404, India

September 2015

Course Content

Sr.No	Chapter	Learning Outcome
1.	Overview of Raspberry Pi	Students have knowledge of Raspberry pi
2.	Installation of Raspberry Pi Operating System	Students will learn installation part of Raspberry Pi operating System
3.	Basics of Python	Students will understand basic skills of python programming
4.	Remote Control of Pi	Students will analyze and implement the Remote Control techniques through Pi board.
5.	LED On & Off	Students will learn and implement the LED on and off Programme through Pi board.
6.	LED Blink	Students will learn and implement the LED blinking programme through Pi board.
7.	Push Button	Students will study and implement the Push button programme through Pi board.
8.	User Inputs	Student will study and apply knowledge of user input programme of Raspberry pi
9.	SOS Buzzer	Student will study and apply knowledge of SOS buzzer to implement programme of SOS Buzzer by using Raspberry pi
10.	Temperature Sensor	Student will study and apply knowledge of SOS buzzer to implement programme of SOS Buzzer by using Raspberry pi
11.	Light Dependent Resistor	Student will study and apply knowledge of SOS buzzer to implement programme of SOS Buzzer by using Raspberry pi
12.	Passive Infrared Sensor	Student will study and apply knowledge of SOS buzzer to implement programme of SOS Buzzer by using Raspberry pi
13.	PiCamera Module	Student will study and apply knowledge of SOS buzzer to implement programme of SOS Buzzer by using Raspberry pi
14.	Interfacing PiCamera, PIR and Buzzer	Student will study and apply knowledge of SOS buzzer to implement programme of SOS Buzzer by using Raspberry pi
15.	Playing Music	Student will study and apply knowledge of SOS buzzer to implement programme of playing Music by using Raspberry pi
16.	Auto run during start-up of Raspberry Pi and execution of multiple programs simultaneously	Student will study and apply knowledge of SOS buzzer to implement programme of SOS Buzzer by using Raspberry pi

17.	E-mail Notifier	Student will study and apply knowledge of SOS buzzer to implement programme of SOS Buzzer by using Raspberry pi
18.	Socket Programming	Student will study and apply knowledge of SOS buzzer to implement programme of SOS Buzzer by using Raspberry pi
19.	Chat Server	Student will study and apply knowledge of SOS buzzer to implement programme of SOS Buzzer by using Raspberry pi
20.	File Transfer between two Raspberry Pi	Student will study and apply knowledge of SOS buzzer to implement programme of SOS Buzzer by using Raspberry pi
21.	Raspberry Pi as a Server	Student will study and apply knowledge of SOS buzzer to implement programme of SOS Buzzer by using Raspberry pi
22.	Interfacing Raspberry Pi with GPS	Student will study and apply knowledge of SOS buzzer to implement programme of SOS Buzzer by using Raspberry pi
23.	Controlling Mains using Raspberry Pi	Student will study and apply knowledge of SOS buzzer to implement programme of SOS Buzzer by using Raspberry pi
24.	Project Ideas	

Chapter 1

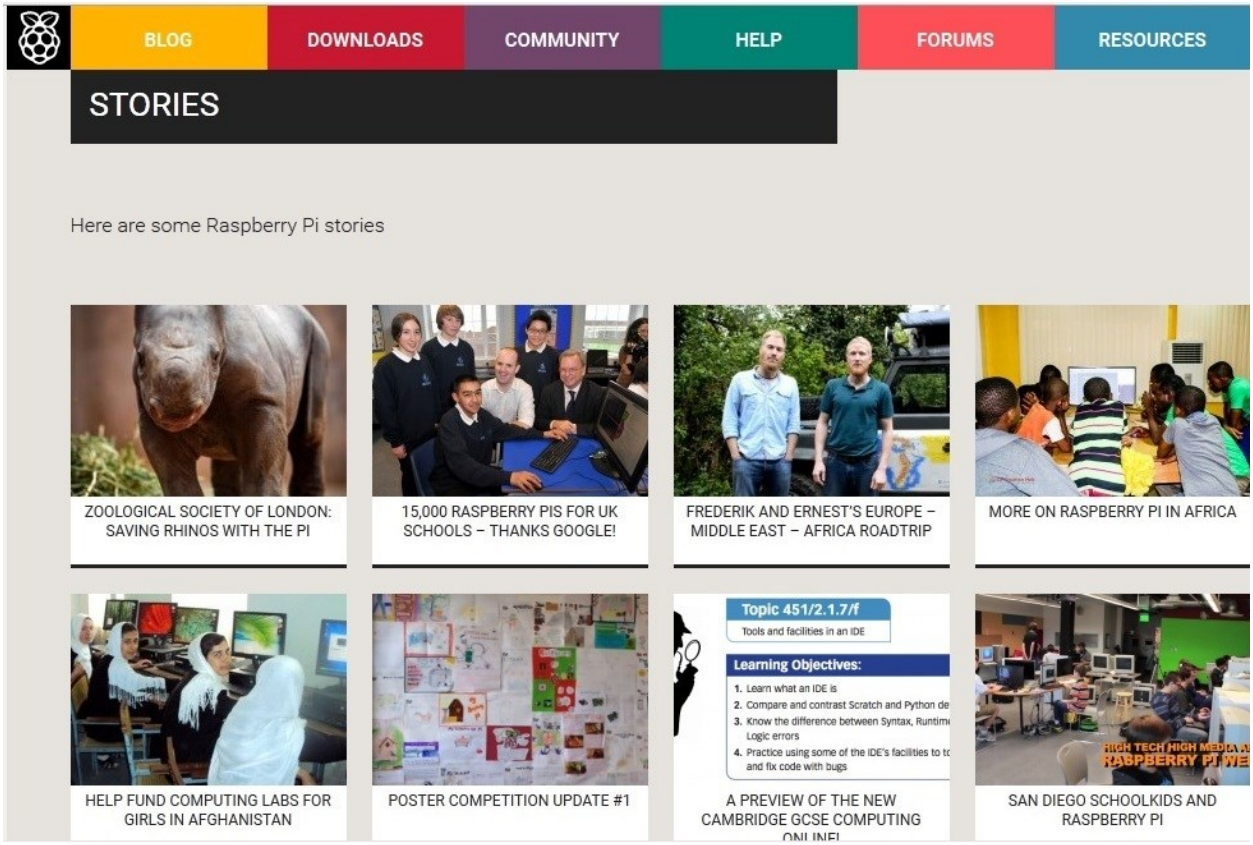
OVERVIEW OF RASPBERRY PI

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.

What's more, the Raspberry Pi has the ability to interact with the outside world, and has been used in a wide array of digital maker projects, from music machines and parent detectors to weather stations and tweeting birdhouses with infra-red cameras. We want to see the Raspberry Pi being used by kids all over the world to learn to program and understand how computers work.

1.1 Raspberry Pi Foundation

The Raspberry Pi Foundation is a registered **educational charity** (registration number 1129409) based in the UK. Our Foundation's goal is to advance the education of adults and children, particularly in the field of computers, computer science and related subjects.



1.2 Eben Upton: The Raspberry Pi Pioneer

He just wanted to help some kids learn to code. Five million units later, his $35 computer has sparked a revolution.

Eben Upton
Age: 36
Occupation: Founder, Raspberry Pi Foundation
Location: Cambridge, England
Hero Moment: Dismayed at students' poor coding skills, he created a capable yet ultracheap computer to get them hooked on programming.

Photo: Dan Burn-Forti

In 1988, 10-year-old Eben Upton bought his first computer: a beat-up, second-hand BBC Micro. This little machine was ubiquitous in British schools at the time, not least because it ran a simple programming language called BASIC, which Upton quickly mastered and used to create various games.

Fast-forward almost two decades. Upton had just been appointed director of studies in computer science at the University of Cambridge's St John's College. And he noticed a big problem: Applications to study computer science at the university were declining, and few incoming students had any programming skills at all.

Like many of his contemporaries, Upton, who holds a Ph.D. in computer science from Cambridge and had already founded two software companies, owed his career to a childhood spent twiddling bits. "Kids today don't have that experience," he says. Instead of diving in deep with programmable computers like the BBC Micro, most of today's youth only scratch the

surface of computing, despite all the hours they log with their phones, game consoles, and Web browsers.

Upton's solution was to create a miniature computer, barely bigger than a credit card, which cost just US $35. It was essentially a circuit board fitted with a smart phone processor chip, 256 megabytes of RAM (since boosted to 1 gigabyte), and some attractive options for connectivity: USB ports, a slot for a memory card, general-purpose input-output pins to link it with other devices, an Ethernet port, and HDMI and analog ports to connect it to a television or computer monitor. "It's trying to be cheap, it's trying to be robust, and it's trying to be fun," says Upton.

When the Raspberry Pi launched on 29 February 2012, Upton figured that getting a thousand units into the hands of the right kids would help to end the drought of qualified candidates applying to study computer science at Cambridge. "Ten thousand would have been a great outcome for us," he recalls thinking. Instead, demand exploded: To date, more than 5 million units have sold, including the stripped-down Model A+, which boasts a $20 price tag, and the more powerful Pi2, which debuted last month. Pi's low cost means that pretty much everyone can own one.

"Overly sleek things can be bad for education," says Alan Mycroft, a professor of computing at the University of Cambridge's Computer Laboratory and a trustee of the Raspberry Pi Foundation. This charity, which Upton set up, takes the profit from Pi sales and plows it into education initiatives and other outreach activities.

Mycroft credits Upton for helping inspire a new generation of student coders, hackers, and hobbyists. "He's brought inspiration, leadership, and persistence to it—Eben just won't let go."

Growing up in the small town of Likely in northern England, Upton didn't have a techy household—his father was a professor of English, his mother an English teacher. In part because of his own background, he's thrilled that the Raspberry Pi is providing those without IT-savvy parents an opportunity to learn about computing.

That egalitarian spirit pervades the Raspberry Pi headquarters as well. Right after Upton politely asked the office manager to make me a cup of tea, he bounded after her to apologize. "I'll do it," he said. "You've got better things to do, and I was being an ass."

Raspberry Pi's success can be tied to its vibrant user community. In June 2012, Pi enthusiast Ben Nuttall organized the first "Raspberry Jam"—part fan-club meet, part hackathon—and since then there have been hundreds more of these community-led events. "They're full of kids learning to solder," Upton says happily. He was so impressed that he hired Nuttall to work for the Raspberry Pi Foundation.

The foundation and the business now have a staff of about 20. Many of them came from the Cambridge offices of U.S. chipmaker Broadcom, where Upton began working in 2006 as a chip architect. Upton—who's now allowed to spend most of his time on Raspberry Pi—took on some of those who lost their jobs at Broadcom last year during a round of layoffs. That makes for a close-knit family atmosphere at the Pi offices, where Upton's wife, Liz, is head of communications. "It's not a nine-to-five job," notes Mycroft. "It's something that has consumed much of their lives for the last five or so years."

Although the Pi sells globally, penetration in the developing world remains patchy. The foundation has set up showcase projects that include using the computer as a local server for remote villages that lack high-bandwidth mobile or landline connectivity. The computers come loaded with useful information—Wikipedia entries, books, videos—and are then stationed in the middle of town so that villagers and schools can access the database wirelessly.

Back home, the Pi is also playing a growing role in schools. At St John's College School, just a 10-minute walk from Pi headquarters, a lunchtime programming club is buzzing with more than 25 pupils busily writing code and tinkering with the computers. "They learn by hacking and experimenting, which fixes it in their memories much better than if you give them a traditional course," says Graham Hastings, the school's head of information and communications technology.

He also uses the Raspberry Pi in lessons for physical-computing applications—controlling model lighthouses or fairground rides, and as part of science experiments. "The great thing about the Raspberry Pi is that it's not about making money," says Hastings. The interface boxes and software required to connect conventional computers to devices are expensive and need frequent upgrades. But the cheap Pi, with its open-source software, has changed that: "I've been freed from the tyranny of the upgrade cycle," he says.

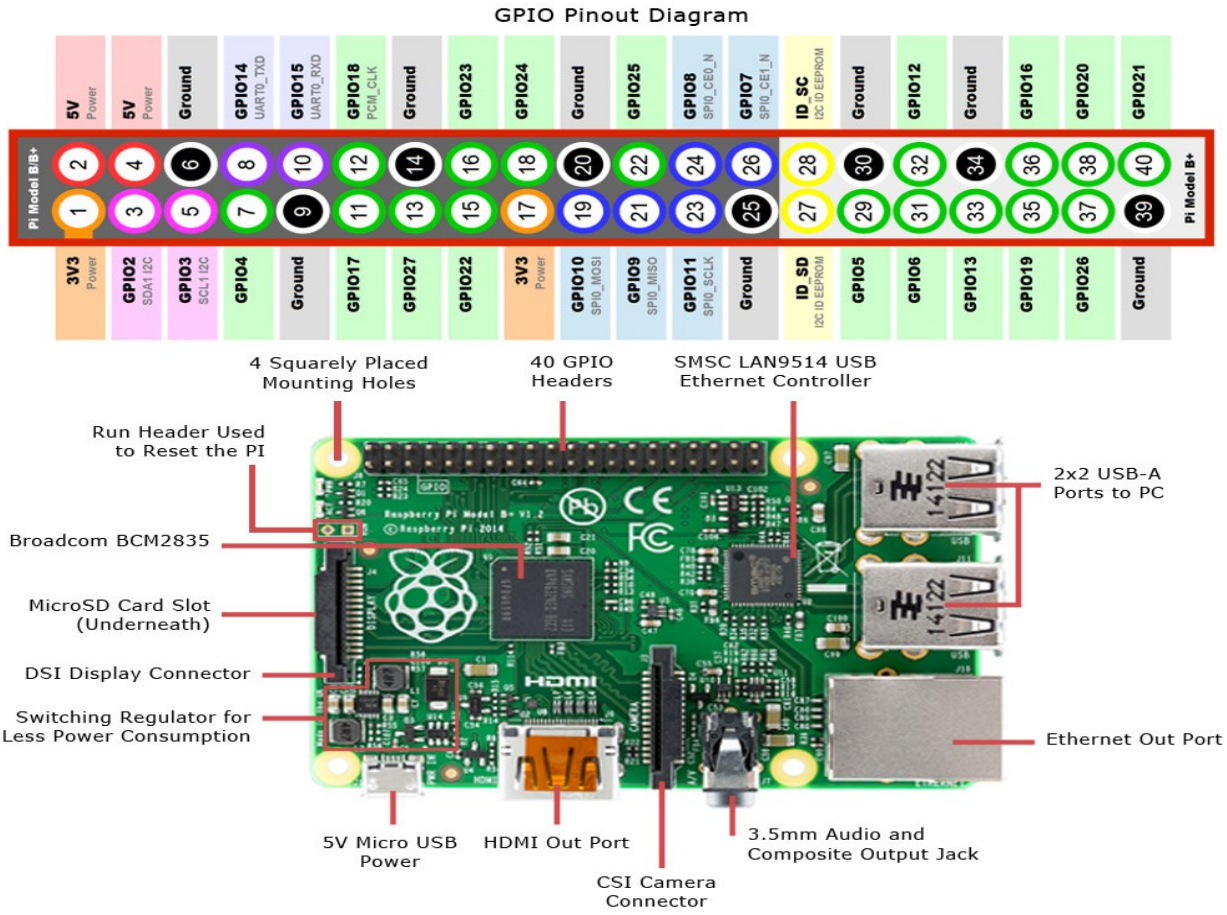
The Pi computers can run Scratch, free education software that portrays programming commands as jigsaw pieces, allowing children to quickly click them together into working programs. Then they typically graduate to Python, "one of the few languages that genuinely spans the range from trivial introductory level to professional software engineer," says Upton.

With millions of Pi computers now in circulation, Upton says the main barrier to getting more kids coding is a lack of skills among teachers. "A vastly greater investment in teacher training is required, and it's the government's job to do it—that's what governments are for. If we're serious about it, we've got to pony up the cash."

The Raspberry Pi Foundation aims to help with its Pi academy, which since April 2014 has been offering free computer training for teachers. The foundation plans to roll out more education initiatives and is building corporate partnerships to bring the Raspberry Pi into more schools. Meanwhile, the U.K. government recently launched a revamped curriculum for students studying information and communication technology that places much greater emphasis on programming.

1.3 Hardware on Pi board:

- **10/100 Ethernet Socket**
- **HDMI Socket**
- **USB 2.0 Socket**
- **RCA Video Socket**
- **SD Card socket**
- **Powered From Micro USB socket**
- **3.5 mm Audio out Jack**
- **Header footprint for Camera Connection**



1.4 Raspberry Pi GPIO Pin Details:

GPIO Numbers

Raspberry Pi B
Rev 1 P1 GPIO Header

	Pin No.		
3.3V	1	2	5V
GPIO0	3	4	5V
GPIO1	5	6	GND
GPIO4	7	8	GPIO14
GND	9	10	GPIO15
GPIO17	11	12	GPIO18
GPIO21	13	14	GND
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10	19	20	GND
GPIO9	21	22	GPIO25
GPIO11	23	24	GPIO8
GND	25	26	GPIO7

Raspberry Pi A/B
Rev 2 P1 GPIO Header

	Pin No.		
3.3V	1	2	5V
GPIO2	3	4	5V
GPIO3	5	6	GND
GPIO4	7	8	GPIO14
GND	9	10	GPIO15
GPIO17	11	12	GPIO18
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10	19	20	GND
GPIO9	21	22	GPIO25
GPIO11	23	24	GPIO8
GND	25	26	GPIO7

Raspberry Pi B+
B+ J8 GPIO Header

	Pin No.		
3.3V	1	2	5V
GPIO2	3	4	5V
GPIO3	5	6	GND
GPIO4	7	8	GPIO14
GND	9	10	GPIO15
GPIO17	11	12	GPIO18
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10	19	20	GND
GPIO9	21	22	GPIO25
GPIO11	23	24	GPIO8
GND	25	26	GPIO7
DNC	27	28	DNC
GPIO5	29	30	GND
GPIO6	31	32	GPIO12
GPIO13	33	34	GND
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
GND	39	40	GPIO21

Key

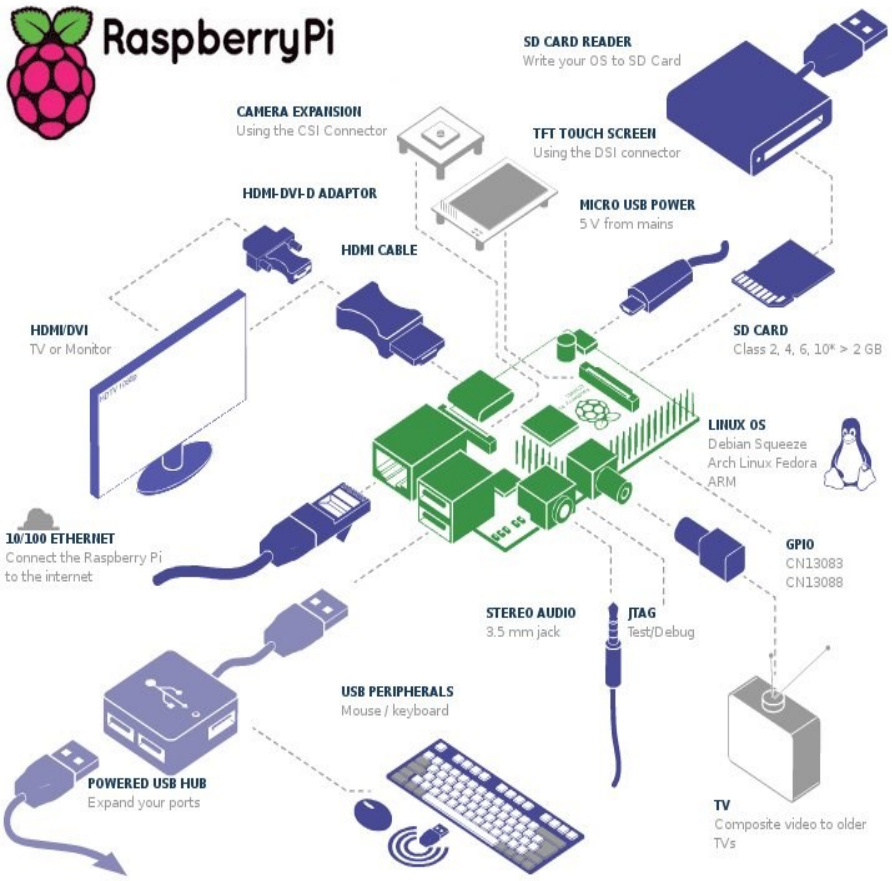
Power +	UART
GND	SPI
I²C	GPIO

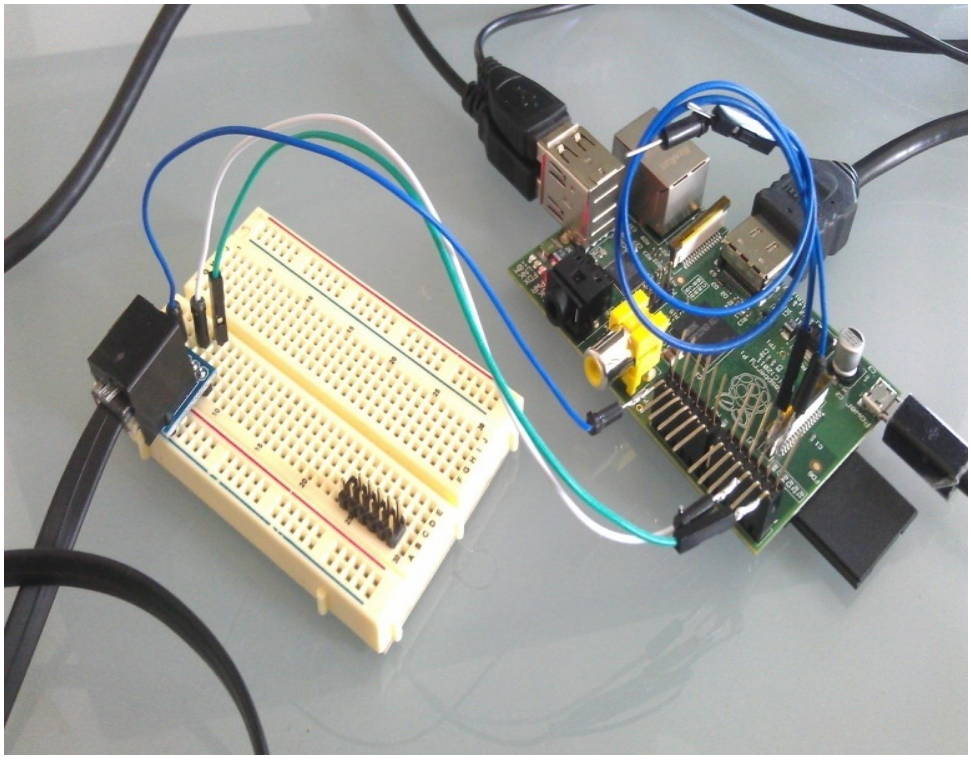
UART- Universal asynchronous receiver/transmitter is a piece of computer hardware that translates data between parallel and serial forms.

SPI – **Serial Peripheral Interface** bus is a synchronous serial communication interface specification used for short distance communication, primarily in embedded systems

GPIO- **General Purpose Input output** is a generic pin on an integrated circuit whose behaviour, including whether it is an input or output pin, can be controlled by the user at run time.

1.5 Raspberry Pi connections





1.6 Raspberry Pi Model comparison

Feature	Model A	Model B	Model A+	Model B+	Model Pi 2 [B]
Release Date	February 2012	N/A	November 2014	July 2014	February 2015
BRCM2835 SoC	Yes	Yes	Yes	Yes	Yes
Standard SoC Speed CPU	700 MHz	700 MHz	700 MHz	700 MHz	900 MHz Quad core
RAM	256 MB	512 MB*	256 MB	512 MB	1 GB (shared with GPU)
Storage	Full SD	Full SD	Micro SD	Micro SD	Micro SD
Ethernet 10/100	No	Yes	No	Yes	Yes
HDMI output port	Yes	Yes	Yes	Yes	Yes
Composite Video output	Yes	Yes	On 3.5 mm jack	On 3.5 mm jack	On 3.5 mm jack
No of USB2.0 Port	1	2	1	4	4
No of available GPIO	**26**	**26**	26	40	40
No of Camera interface port	1	1	1	1	1

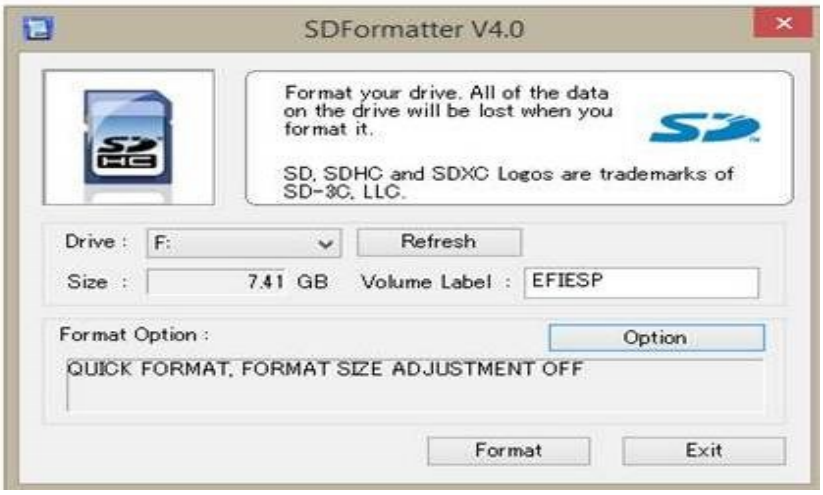
Chapter -2

INSTALLATION OF OPERATING SYSTEM OF RASPBERRY PI

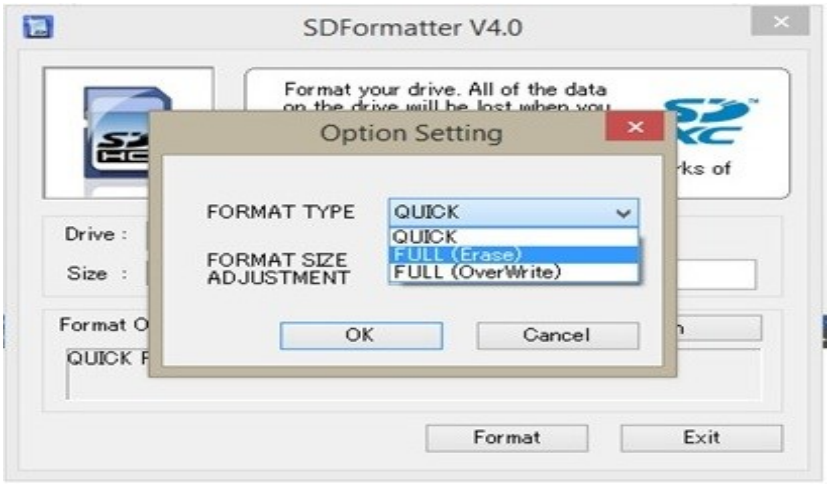
Installation part of Raspberry Pi

By using card reader we have perform following task for SD Card formatting.

A] SD Card formatting:



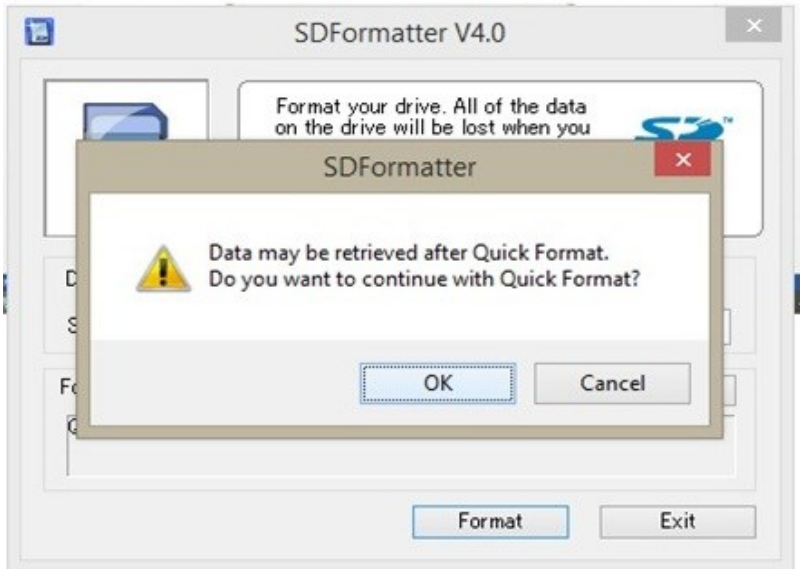
Step 1: View of SD Card



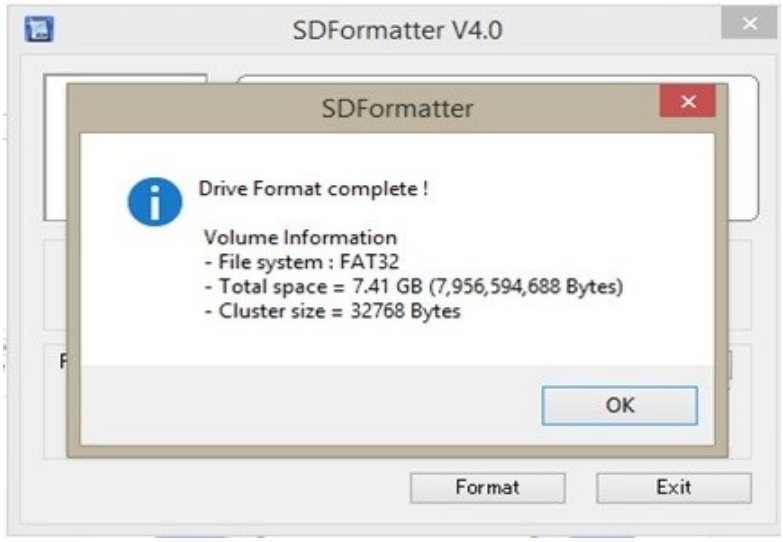
Step 2: Select option for formatting of SD card.


SDFormatter V4.0
Format your drive. All of the data on the drive will be lost when you format it.
SD, SDHC and SDXC Logos are trademarks of SD-3C, LLC.
Drive : F:
Refresh
Size : 7.41 GB
Volume Label : EFIESP
Format Option :
Option
QUICK FORMAT, FORMAT SIZE ADJUSTMENT OFF
Format
Exit

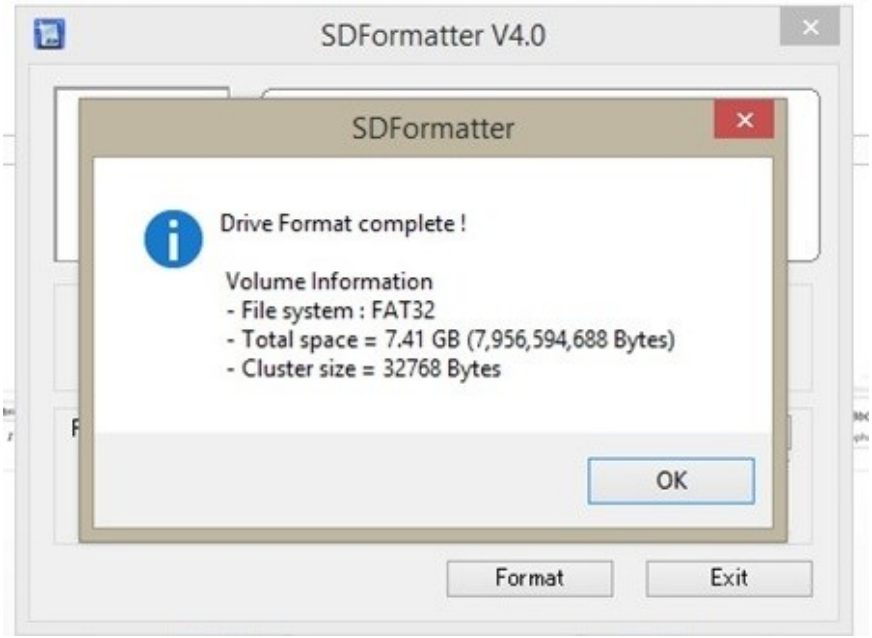
Step 3


SDFormatter V4.0
Format your drive. All of the data on the drive will be lost when you
SDFormatter
Data may be retrieved after Quick Format.
Do you want to continue with Quick Format?
OK
Cancel
Format
Exit

Step 4



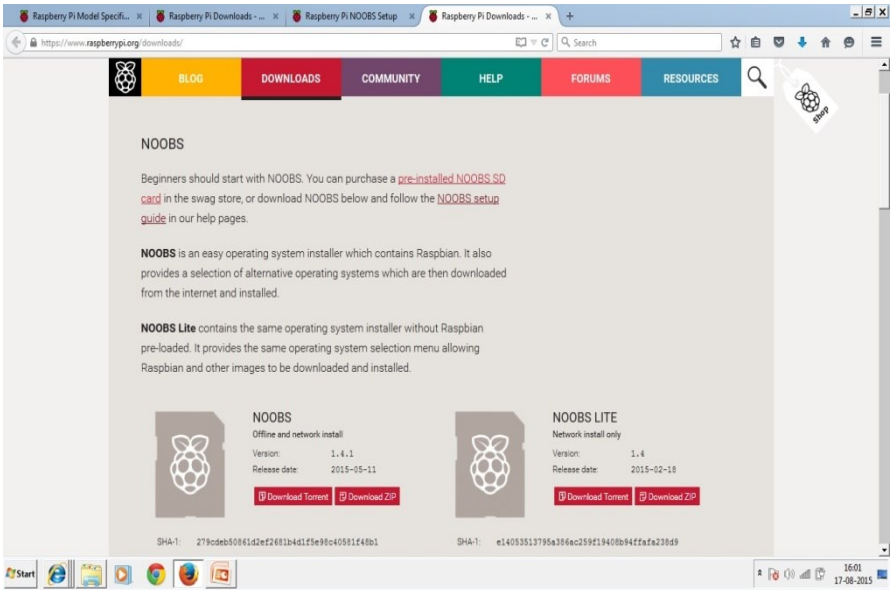
Step 5: System shows pop-up window



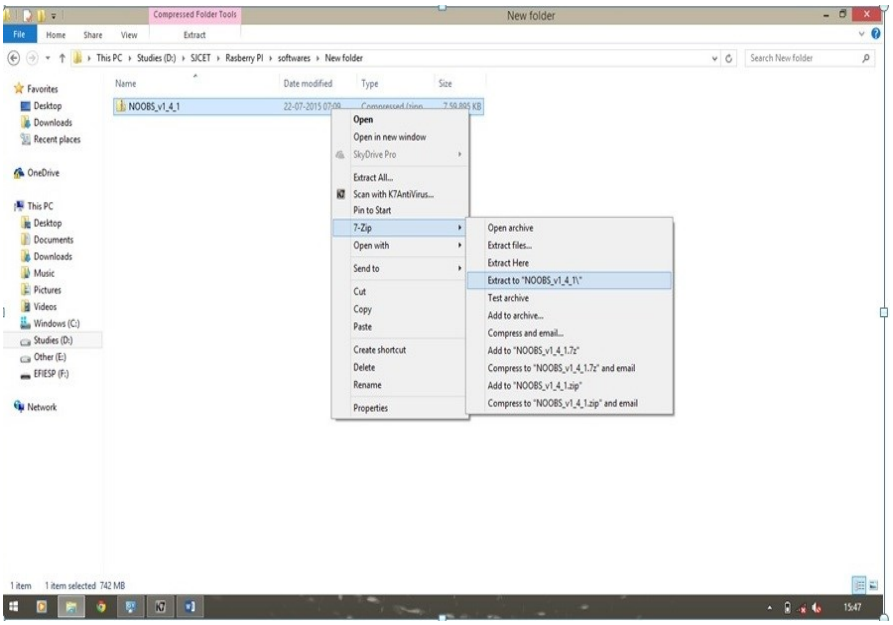
Step 6: User must click on Ok button

B] URL: for Download NOOBS setup file

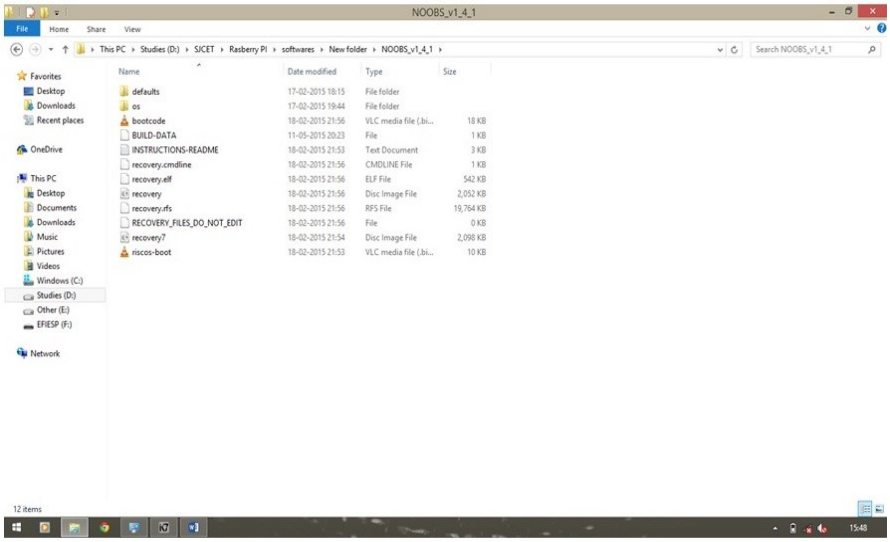
https://www.raspberrypi.org/downloads/



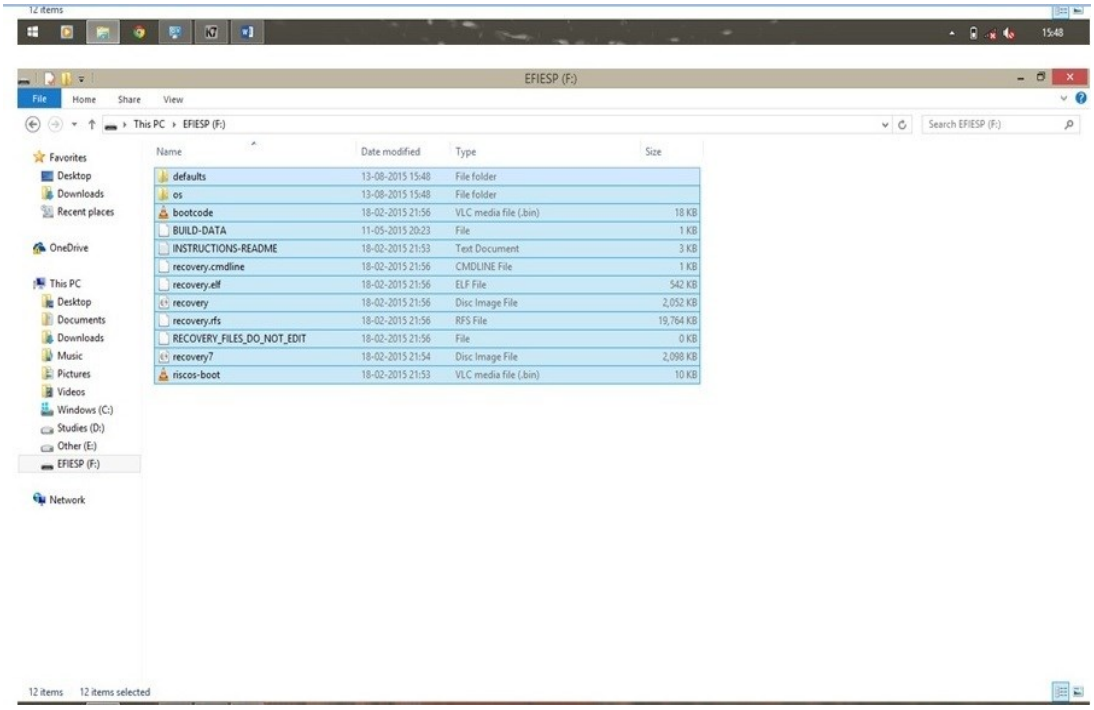
Step7: Download setup files of NOOBS [Raspberry Pi]



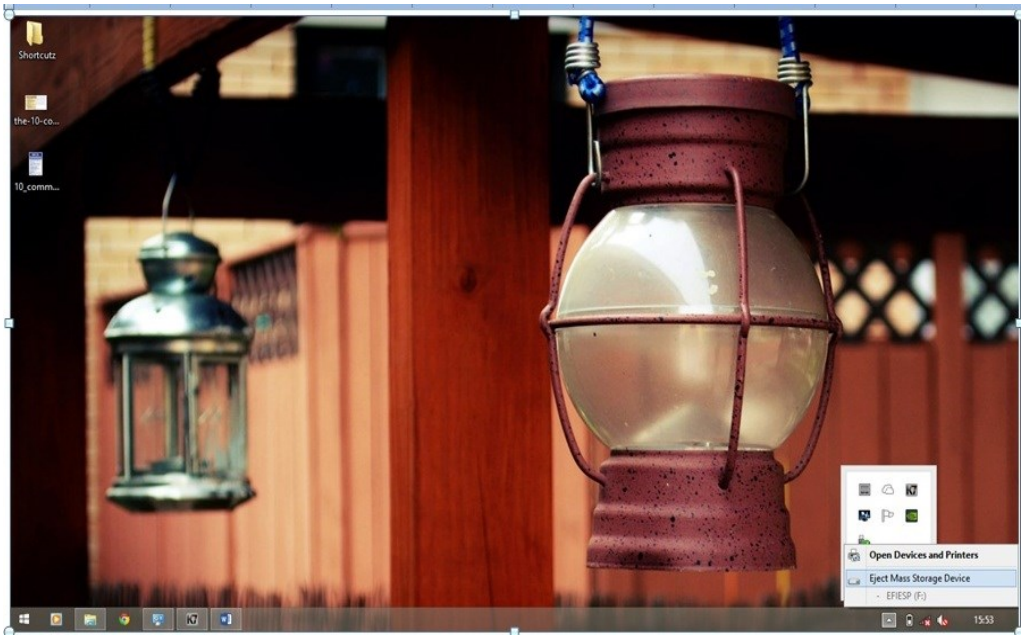
Step 8**:** Extract Files from NOOBS Setup ZIP folder



Step 9: Copy and paste Extracted Files from NOOBS Setup ZIP folder

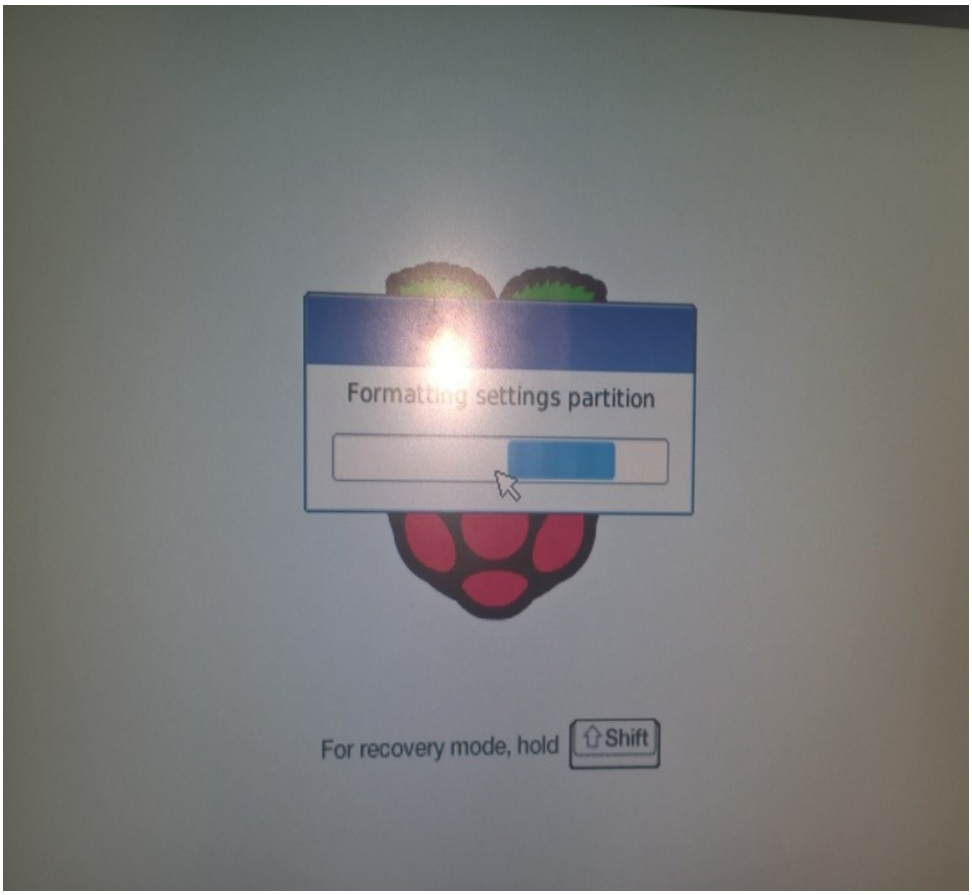


Step10: Copy and paste Extracted Files from NOOBS Setup ZIP folder

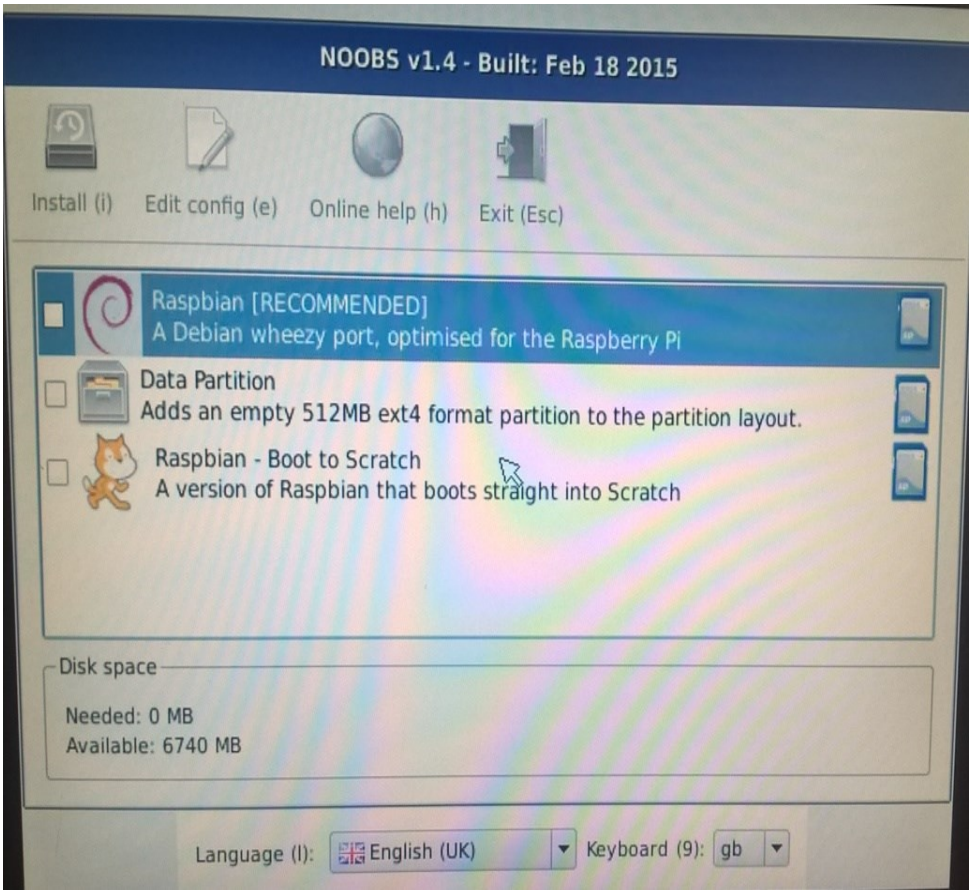


Step 11: Remove safely your SD card from PC

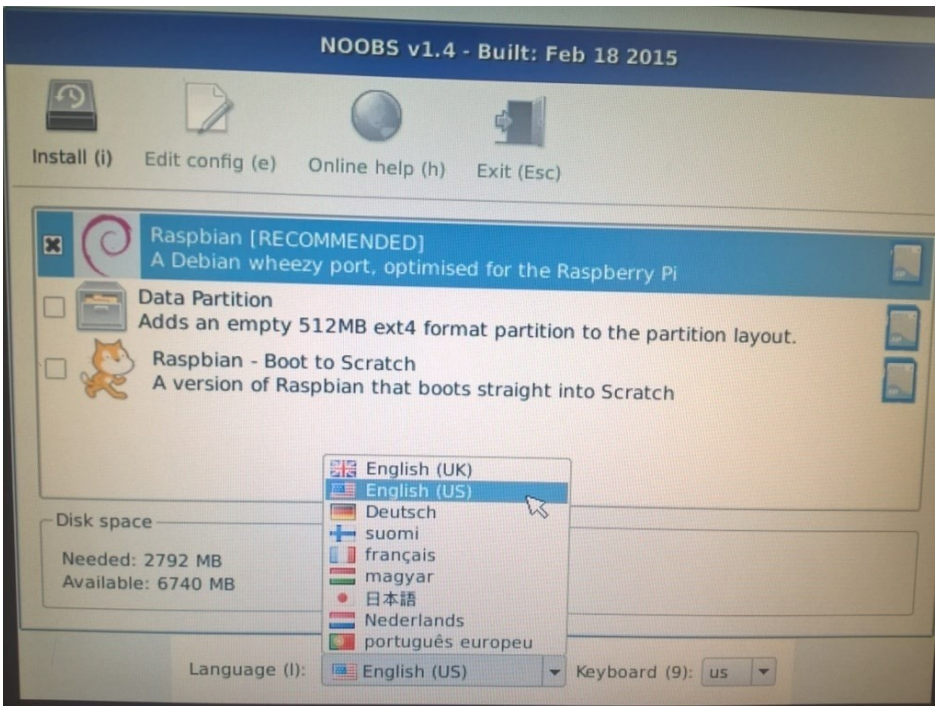
C] Booting of Raspberry Pi through SD card



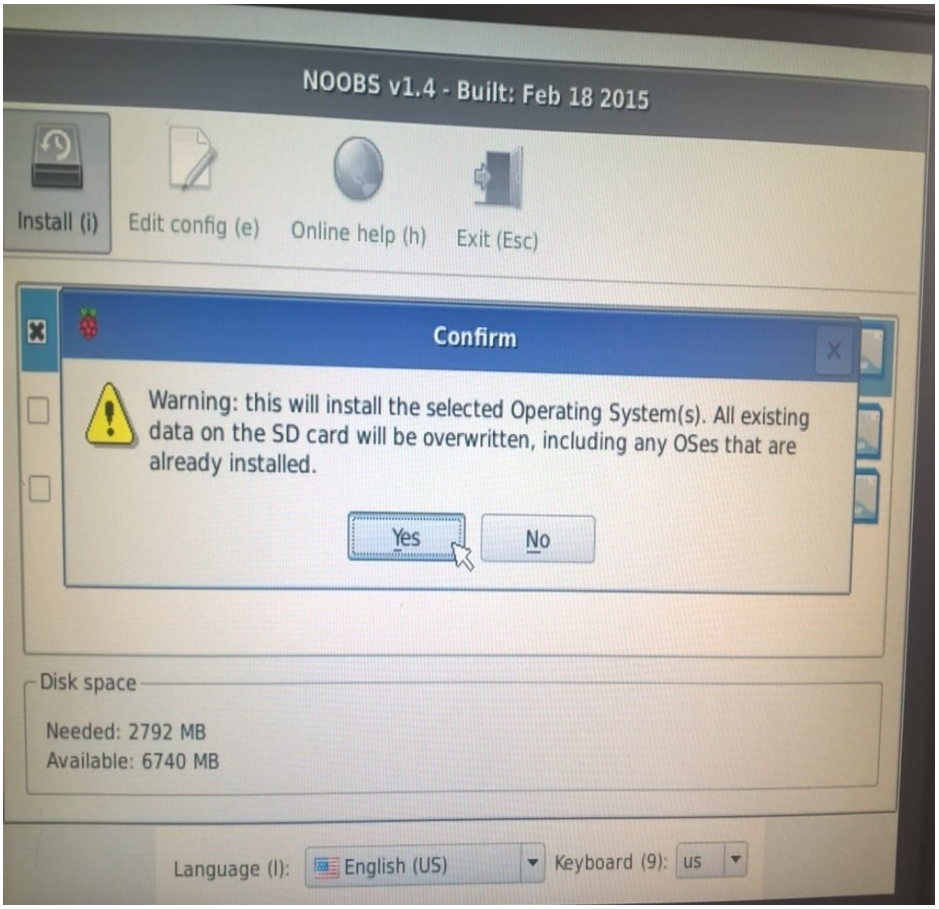
Step 12: First Screen look like this



Step 13: Select Raspbian Option

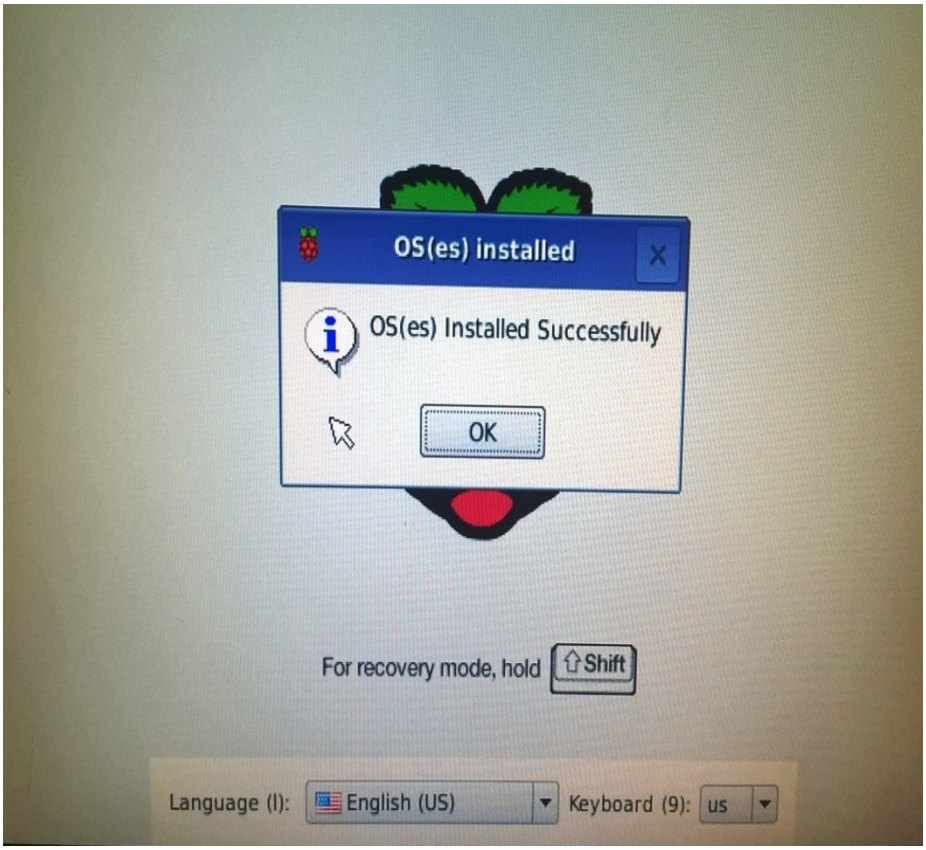


Step14: Select Language

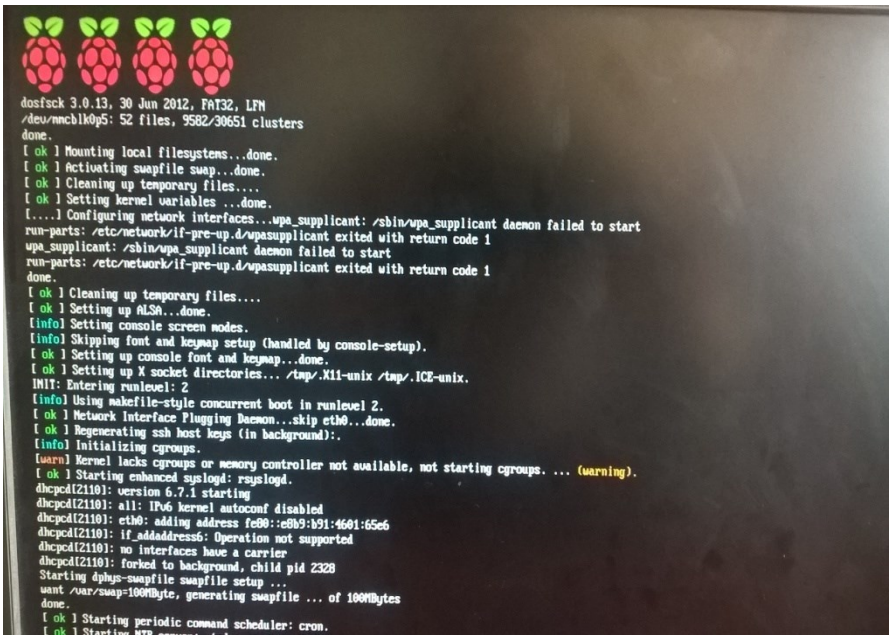


Step 15: Warning Massage from Pi click on 'Yes' Button

Step 16: Installation starts

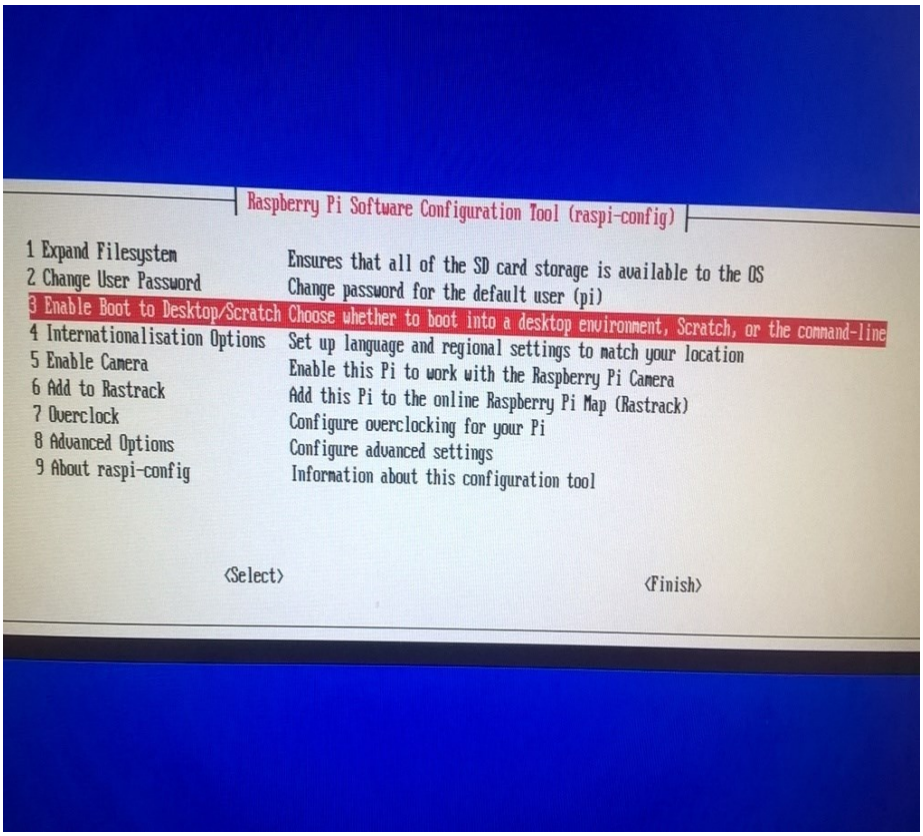


Step 17: OS Installation Pop up Massage from Pi

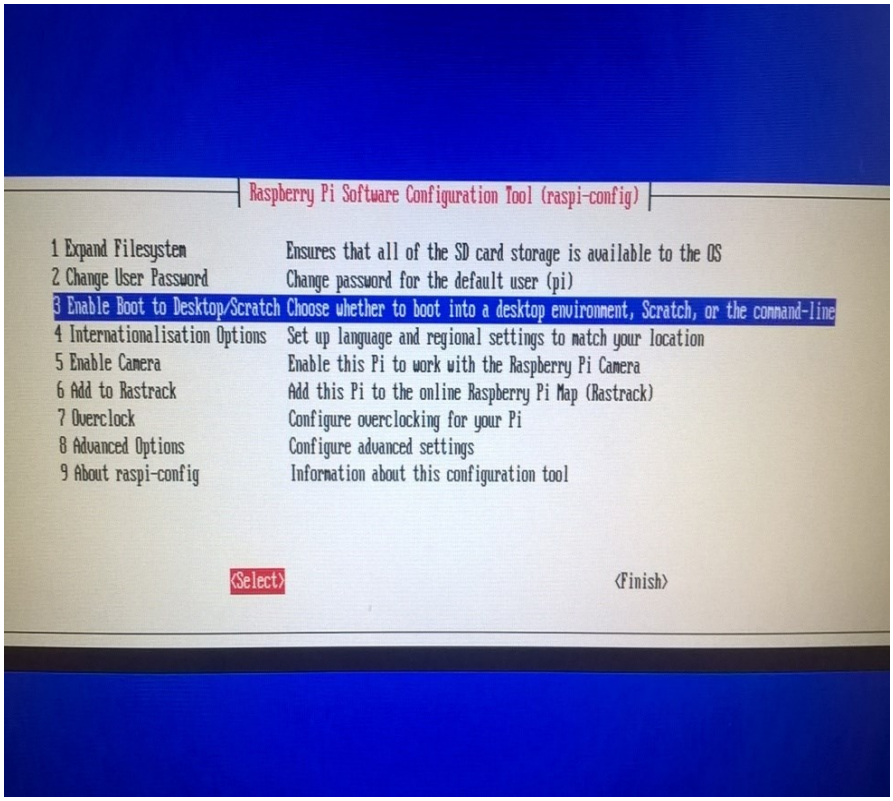


Step 18: First time Bootable screen of Pi

D] Raspberry Pi software configuration

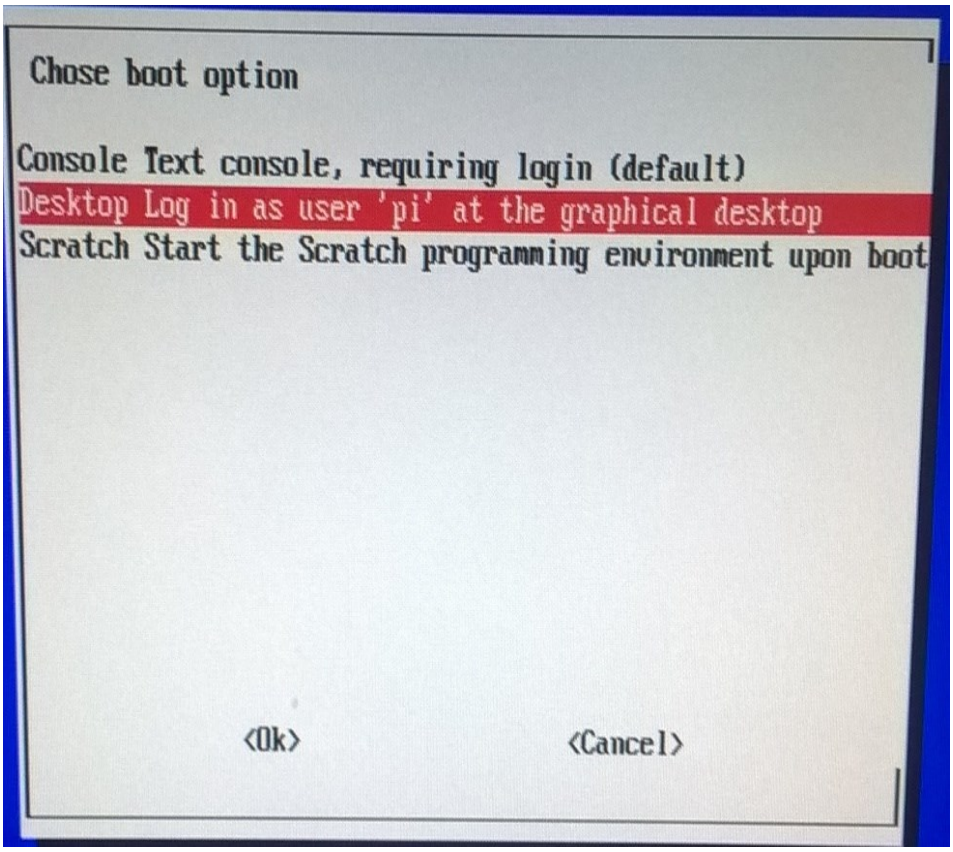


Step 19: Pi screen shows to choose Software Configuration select 3 option

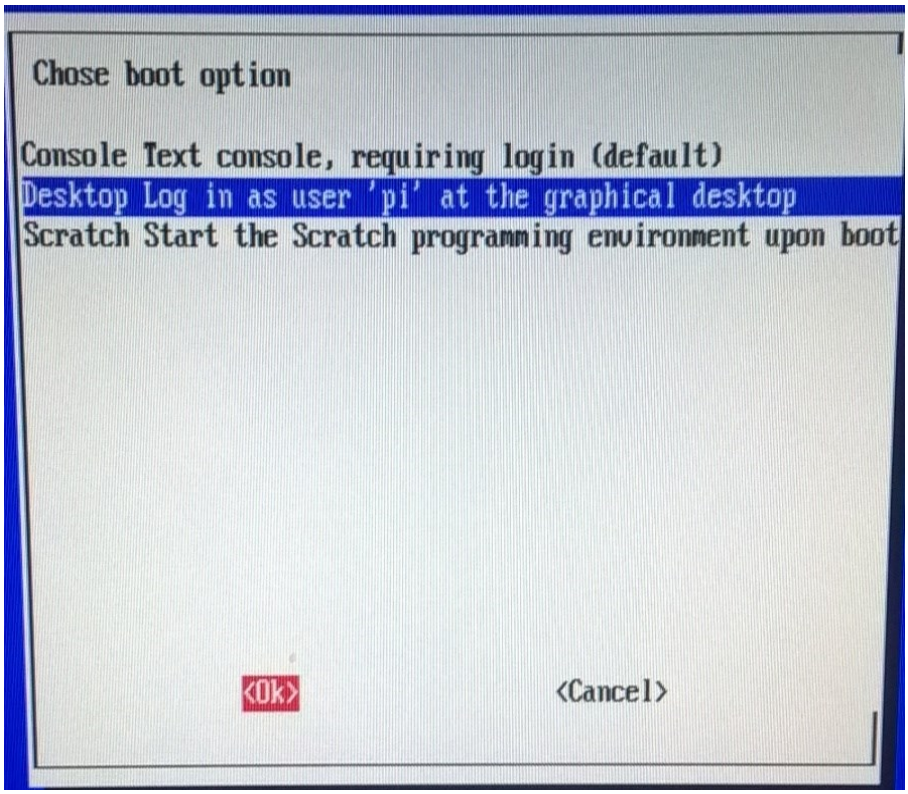


Step 20: After user selection it becomes Blue colour

E] User ID selection

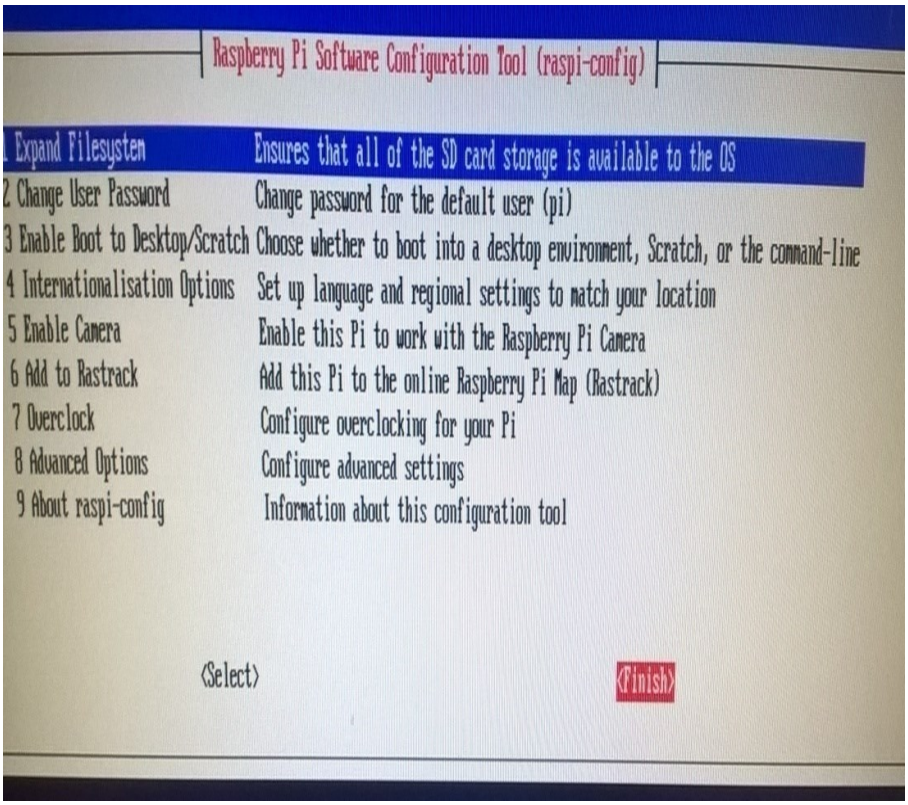


Step 21: Raspberry Pi shows screen for selection ‘User ID’ as login of OS

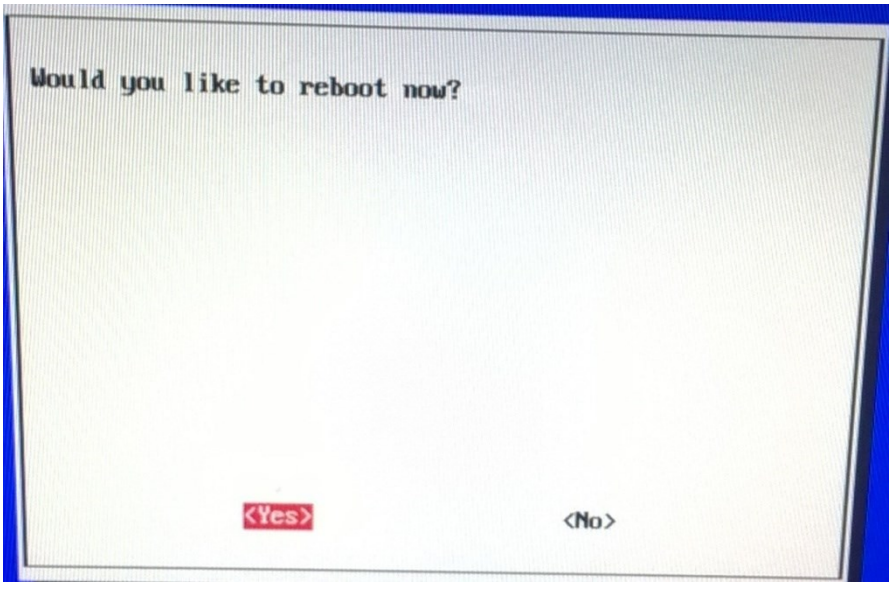


Step 22: Select second option

F] Expand file system with SD card

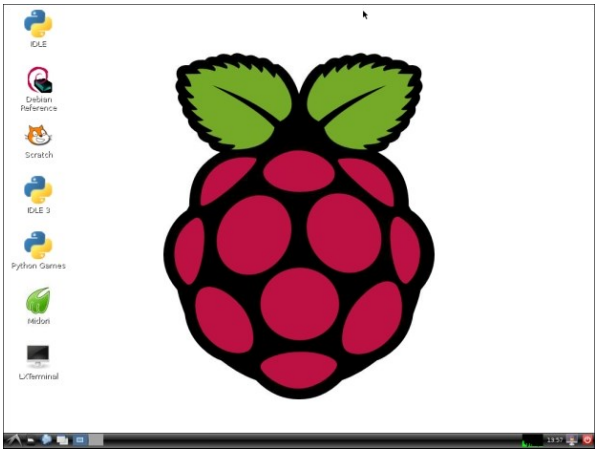


Step 23: Raspberry Pi screen shows screen select option one



Step 24: Select ‘Yes’ Button to boot your pi.

After the reboot, RPi GUI environment will start and you are ready to play with your Pi.



Chapter 4 : Remote control of Raspberry Pi

1. Remote Control with ssh

In this lesson you will learn how to remote control your Raspberry Pi over your local network using Secure Shell (SSH).

A common reason for remote controlling your Pi from another computer is that you may be using your Pi solely to control some electronics and therefore not need a keyboard, mouse and monitor, other than for setting it up.

Enabling SSH:

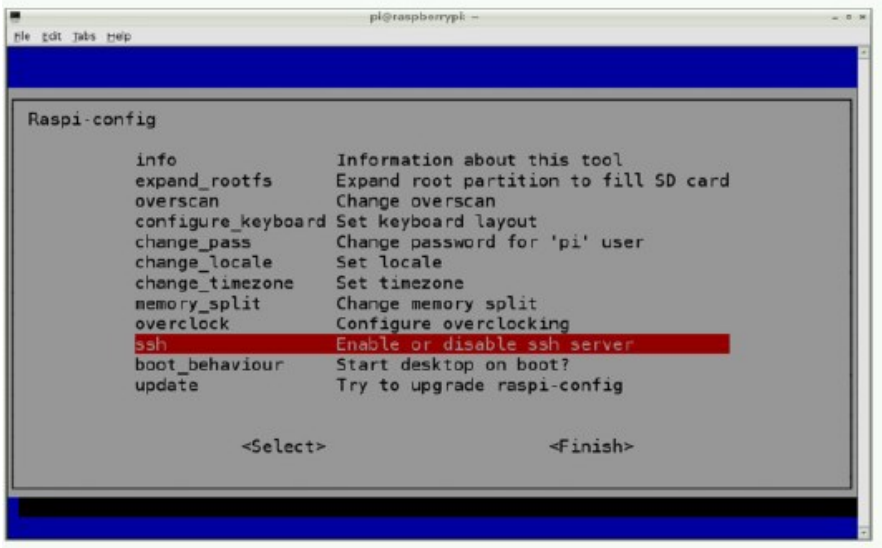
Secure Shell (SSH) is a feature of Linux that allows you to effectively open a terminal session on your Raspberry Pi from the command line of your host computer.

To use SSH, you need to first enable your Pi for using it. The easiest way to do this is to use Raspi Config.

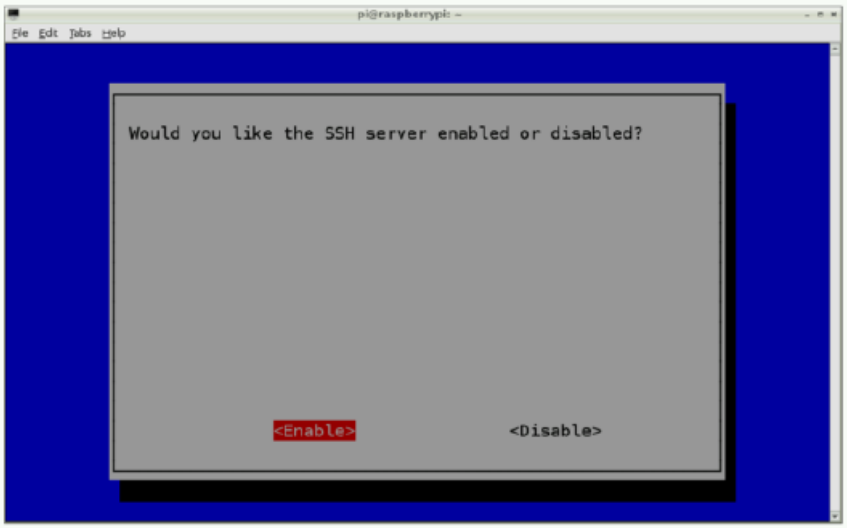
Open LX Terminal on your Pi and enter the following command to start Raspi Config:

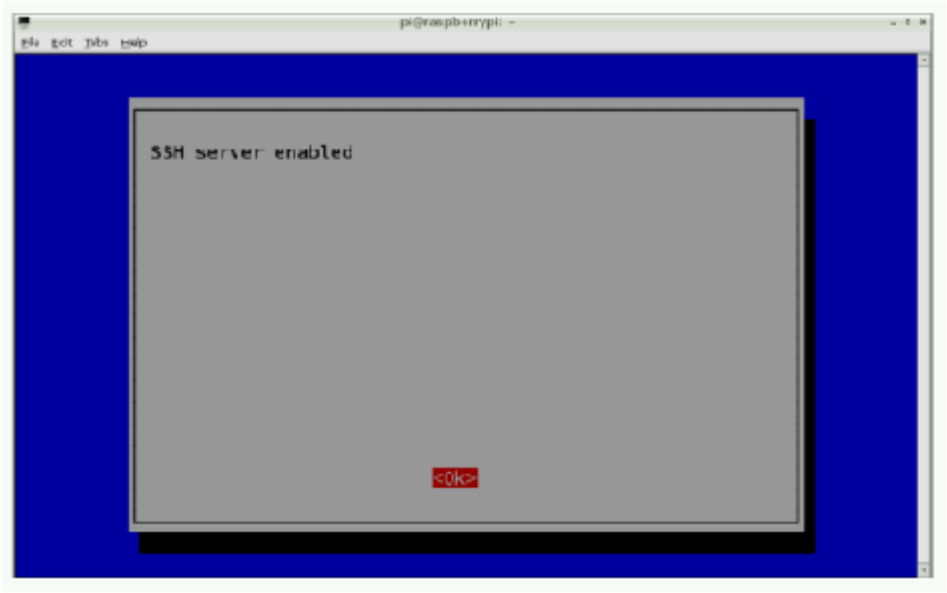
```
sudo raspi-config
```

Scroll down to `ssh`:



Hit Enter and then select Enable:



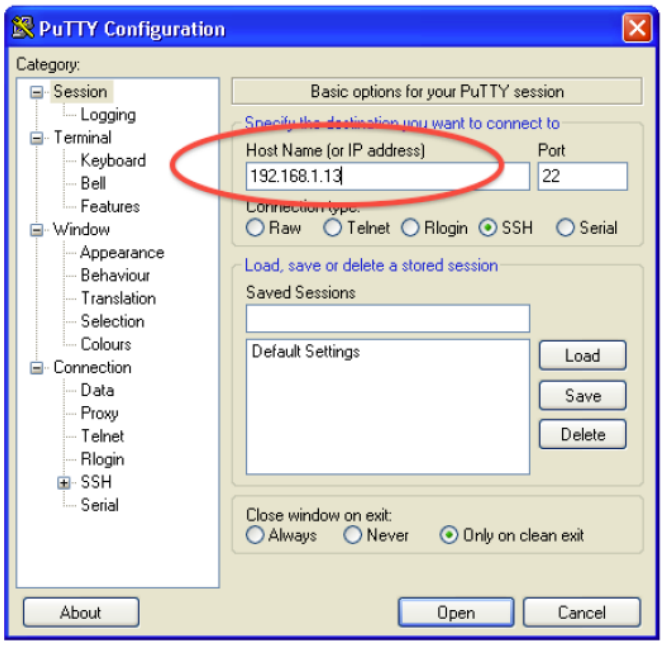


SSH under Windows

If you use windows, then you will need to download a free program called “putty” from here:

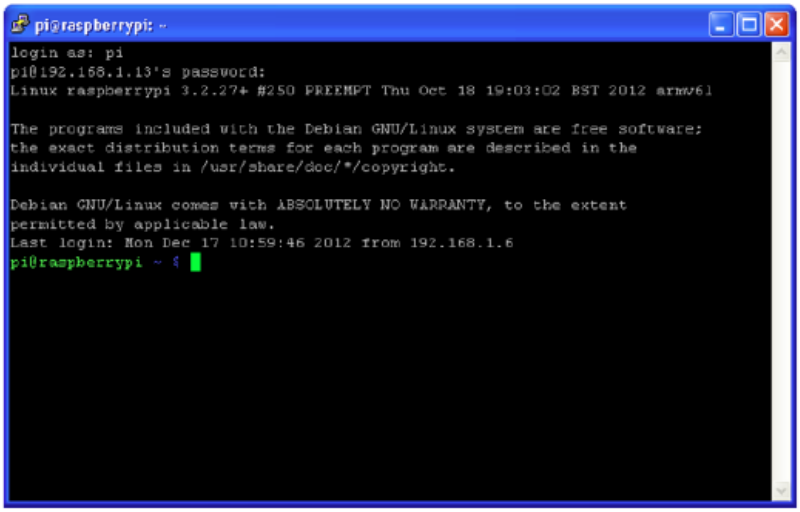
http://www.putty.org/ (http://adafru.it/aUb).

Having downloaded and installed Putty (its a single file called putty.exe), run the program.



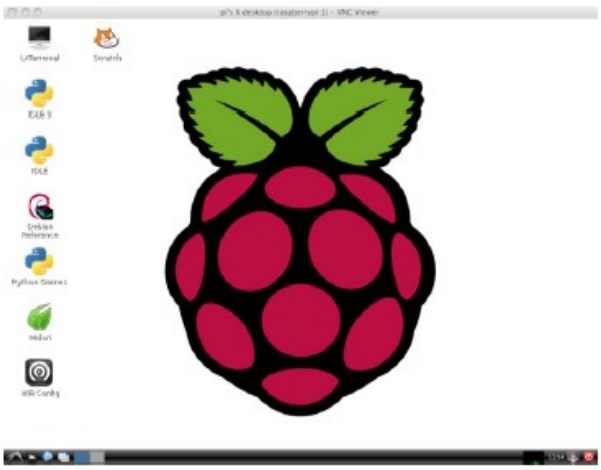
Enter the IP address that you found earlier and click “Open”. This will give you a warning (the first time) and then prompt you for the user (“pi”) and password (“raspberry”).

The SSH window will be ready for use.



2. Remote control with VNC

In this lesson we will explain how to install and use VNC on your raspberry Pi. This will allow you to see your Raspberry Pi's desktop remotely in a graphical way, using the mouse as if you were sitting in front of your Pi.



To use VNC (Virtual Network Connection), you have to install some software on your Pi. There are a number of VNC server applications, and the one we are going to use is called “tightvnc”.

We can install the VNC server software using the SSH connection that we established earlier.

Enter the following command into your SSH terminal:

```
sudo apt-get update
```

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

You will be prompted to confirm installation by typing “Y'. We now need to run the VNC Server, so enter the following command into your SSH window:

```
vncserver :1
```

You will be prompted to enter and confirm a password. It would make sense to use “raspberry” for this, but passwords are limited to 8 characters, so I use “raspberr”. Note that this is the password that you will need to use to connect to the Raspberry Pi remotely.

You will also be asked if you want to create a separate “read-only” password – say no.

From now on, the only command that you need to type within your SSH to start the VNC server

will be:

The VNC server is now running and so we can attempt to connect to it, but first we must switch to the computer from which we want to control the Pi and setup a VNC client to connect to the Pi.

vncserver :1

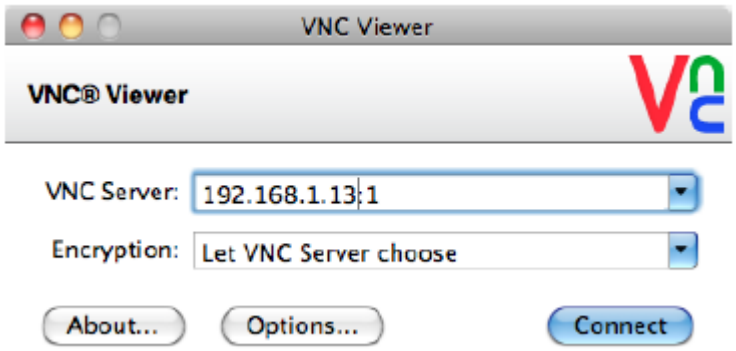
The VNC server is now running and so we can attempt to connect to it, but first we must switch to the computer from which we want to control the Pi and setup a VNC client to connect to the Pi.

Using a VNC Client

Again, there are many VNC clients, of which “VNCViewer”

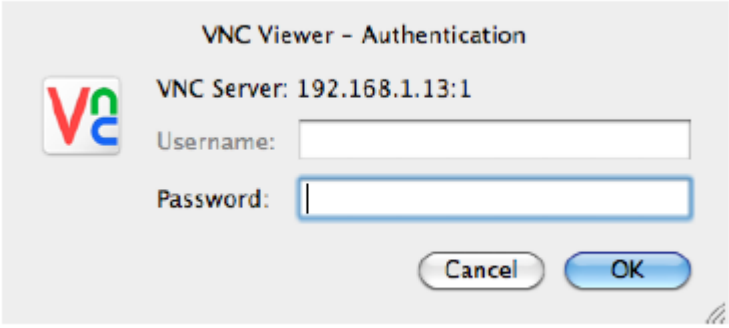
(http://www.realvnc.com (http://adafru.it/aU4)) is available for most platforms and I have found it to work well with TightVNC.

When you first run VNCViewer, you will see the following:



Enter the IP address of your Raspberry Pi, append : 1 (to indicate the port) and click on “Connect”. You will then get a warning message. Just click 'Continue'.

The following window will then popup for you to enter your password ("raspberry").



Finally, the VNC window itself should appear. You will be able to use the mouse and do everything as if you were using the Pi's keyboard mouse and monitor, except through your other computer.

As with SSH, since this is working over your network, your Pi could be situated anywhere, as long as it is connected to your network. It does, then you either have to connect with SSH and restart the VNC Server or arrange for the VNC Server to run automatically after the Raspberry Pi reboots.

There are several different methods of arranging for some code to be run as the Pi starts. The method described below is probably the easiest to use. You can adapt it to run other commands instead of starting the VNC server.

Step 1.

Open a Terminal session on the Pi, or connect using SSH. A new terminal or SSH session will automatically start you off in your home directory of /home/pi. If you are not in this directory, change to it by typing:

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

Then cd to the .config directory by typing:

```
$ cd .config
```

Note the '.' at the start of the folder name. This makes it a hidden folder that will not show up

when you type 'ls'.

Step 2.

Issue the command below to create a new directory inside .config called 'autostart'.

```
$ mkdir autostart
```

cd into that new directory by typing:

```
$ cd autostart
```

All that remains is to edit a new configuration file. So type the following command to open the nano editor on the new file

```
$ nano tightvnc.desktop
```

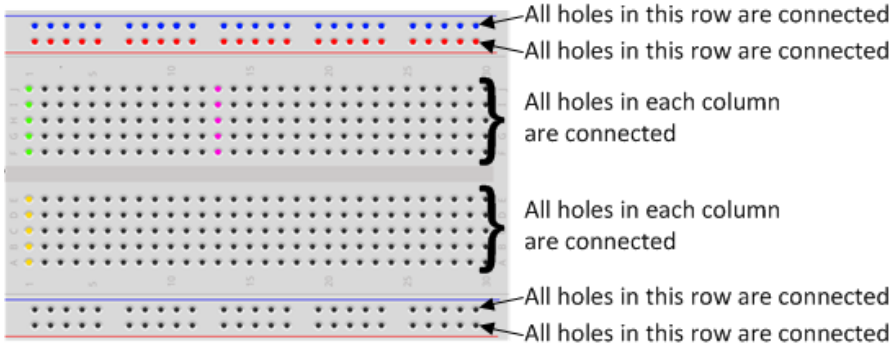
Edit the contents of the file with the following text.

```
[Desktop Entry]
Type=Application
Name=TightVNC
Exec=vncserver :1
StartupNotify=false
```

Type ctrl-X and then Y to save the changes to the file.

Thats all there is to it. The next time you reboot the VNC server will restart automatically.

Breadboard Basics:



The LED:

THE LED

A Red LED stands for Light Emitting Diode, and glows when electricity is passed through it.

When you pick up the LED, you will notice that one leg is longer than the other. The longer leg known as the 'anode'), is always connected to the positive supply of the circuit. The shorter leg known as the 'cathode') is connected to the negative side of the power supply, known as 'ground'.

LEDs will only work if power is supplied the correct way round (i.e. if the 'polarity' is correct). You will not break the LEDs if you connect them the wrong way round – they will just not light. If you find that they do not light in your circuit, it may be because they have been connected the wrong way round.

THE RESISTOR

330 Ohm Resistor You must ALWAYS use resistors to connect LEDs up to the GPIO pins of the Raspberry Pi. The Raspberry Pi can only supply a small current (about 60mA). The LEDs will want to draw more, and if allowed to they will burn out the Raspberry Pi. Therefore putting the resistors in the circuit will ensure that only this small current will flow and the Pi will not be damaged.

Resistors are a way of limiting the amount of electricity going through a circuit; specifically, they limit the amount of 'current' that is allowed to flow. The measure of resistance is called the Ohm (Ω), and the larger the resistance, the more it limits the current. The value of a resistor is marked with coloured bands along the length of the resistor body.

You will be using a 330Ω resistor. You can identify the 330Ω resistors by the colour bands along the body. The colour coding will depend on how many bands are on the resistors supplied:

If there are four colour bands, they will be Orange, Orange, Brown, and then Gold.

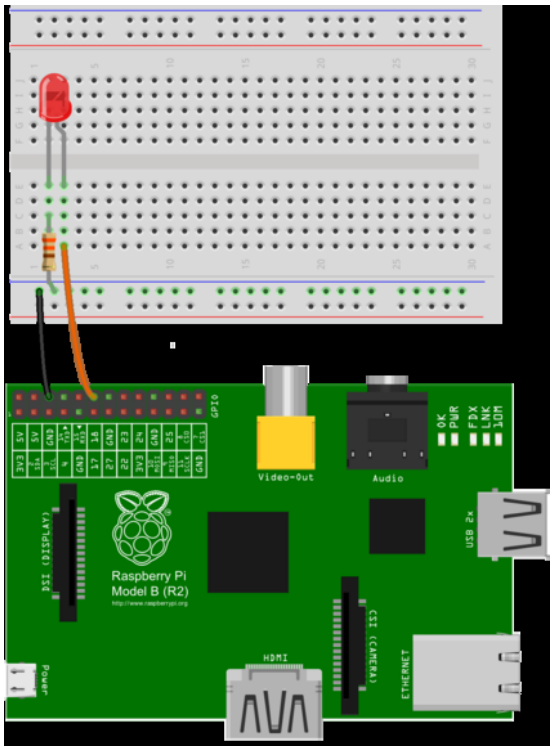
If there are five bands, then the colours will be Orange, Orange, Black, Black, Brown.

It does not matter which way round you connect the resistors. Current flows in both ways through them.

BUILDING THE CIRCUIT

The circuit consists of a power supply (the Pi), an LED that lights when the power is applied, and a resistor to limit the current that can flow through the circuit.

You will be using one of the 'ground' (GND) pins to act like the 'negative' or 0 volt ends of a battery. The 'positive' end of the battery will be provided by a GPIO pin. Here we will be using pin 18. When they are 'taken high', which means it outputs 3.3 volts, the LED will light. Now take a look at the circuit diagram below.



You should turn your Pi off for the next bit, just in case you accidentally short something out.

Use one of the jumper wires to connect a ground pin to the rail, marked with blue, on the breadboard. The female end goes on the Pi's pin, and the male end goes into a hole on the breadboard.

Then connect the resistor from the same row on the breadboard to a column on the breadboard, as shown above.

Next, push the LEDs legs into the breadboard, with the long leg (with the kink) on the right.

Lastly, complete the circuit by connecting pin 18 to the right hand leg of the LED. This is shown here with the orange wire.

LED ON-OFF Code:

Off

```
#!/usr/bin/python
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.cleanup()
GPIO.setwarnings(False)
GPIO.setup(17,GPIO.OUT)
GPIO.setup(27,GPIO.OUT)
print "Lights off"
GPIO.output(17,GPIO.LOW)
GPIO.output(27,GPIO.LOW)
```

On

```
#!/usr/bin/python
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.cleanup()
GPIO.setwarnings(False)
GPIO.setup(17,GPIO.OUT)
GPIO.setup(27,GPIO.OUT)
print "Lights on"
GPIO.output(17,GPIO.HIGH)
GPIO.output(27,GPIO.HIGH)
```

LED Blink

```
#!/usr/bin/python
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(17,GPIO.OUT)
GPIO.setup(27,GPIO.OUT)
```

```
#Turn LEDs on
GPIO.output(17,GPIO.HIGH)
GPIO.output(27,GPIO.HIGH)
time.sleep(1)
#Turn LEDs off
GPIO.output(17,GPIO.LOW)
GPIO.output(27,GPIO.LOW)
time.sleep(1)
#Turn LEDs on
GPIO.output(17,GPIO.HIGH)
GPIO.output(27,GPIO.HIGH)
time.sleep(1)
#Turn LEDs off
GPIO.output(17,GPIO.LOW)
GPIO.output(27,GPIO.LOW)
GPIO.cleanup
```

Blink forever

```
#!/usr/bin/python
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(17,GPIO.OUT)
GPIO.setup(27,GPIO.OUT)
while 1:
      GPIO.output(17,GPIO.HIGH)
      GPIO.output(27,GPIO.HIGH)
      time.sleep(1)
      GPIO.output(17,GPIO.LOW)
      GPIO.output(27,GPIO.LOW)
      time.sleep(1)
```

`import RPi.GPIO as GPIO`	The first line tells the Python interpreter (the thing that runs the Python code) that it will be using a 'library' that will tell it how to work with the Raspberry Pi's GPIO pins. A 'library' gives a programming language extra commands that can be used to do something different that it previously did not know how to do. This is like adding a new channel to your TV so you can watch something different.
`import time`	Imports the Time library so that we can pause the script later on.
`GPIO.setmode(GPIO.BCM)`	Each pin on the Pi has several different names, so you need to tell the program which naming convention is to be used.
`GPIO.setwarnings(False)`	This tells Python not to print GPIO warning messages to the screen.

`GPIO.setup(17,GPIO.OUT)`	This line tells the Python interpreter that pin 18 is going to be used for outputting information, which means you are going to be able to turn the pin 'on' and 'off'.
`print "LED on"`	This line prints some information to the terminal.
`GPIO.output(17,GPIO.HIGH)`	This turns the GPIO pin 'on'. What this actually means is that the pin is made to provide power of 3.3volts. This is enough to turn the LED in our circuit on.
`time.sleep(1)`	Pauses the Python program for 1 second.
`print "LED off"`	This line prints some information to the terminal.
`GPIO.output(17,GPIO.LOW)`	This turns the GPIO pin 'off', meaning that the pin is no longer supplying any power.

Push Button

```
#!/usr/bin/python
import os
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(10, GPIO.IN)

while True:
   if ( GPIO.input(10) == False ):
      print("Button Pressed")
      os.system('date')
      print GPIO.input(10)
      time.sleep(5)
   else:
      os.system('clear')
      print ("Waiting for you to press a button")
time.sleep(1)
```

User Inputs

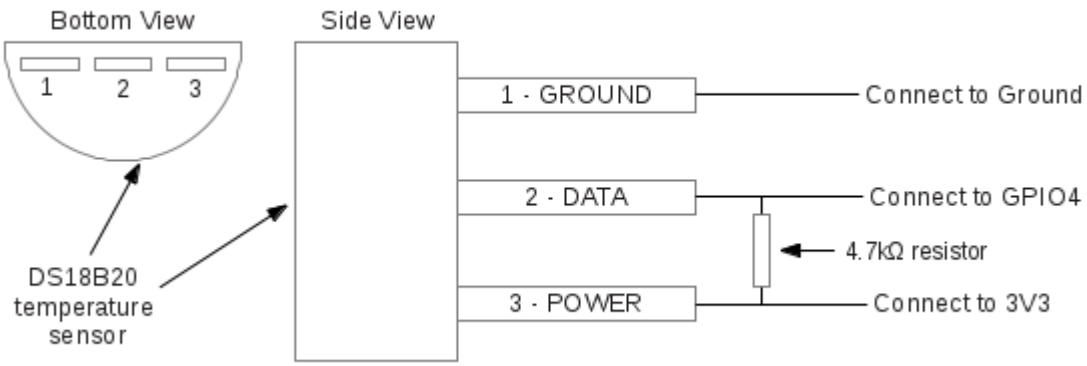
```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import os
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(17,GPIO.OUT)
GPIO.setup(27,GPIO.OUT)
#Setup variables for user input
led_choice = 0
count = 0
os.system('clear')
print "Which LED would you like to blink"
print "1: Red?"
print "2: Blue?"
led_choice = input("Choose your option: ")
if led_choice == 1:
      os.system('clear')
      print "You picked the Red LED"
      count = input("How many times would you like it to blink?: ")
      while count > 0:
            GPIO.output(27,GPIO.HIGH)
            time.sleep(1)
            GPIO.output(27,GPIO.LOW)
            time.sleep(1)
            count = count - 1
if led_choice == 2:
      os.system('clear')
      print "You picked the Red LED"
      count = input("How many times would you like it to blink?: ")
      while count > 0:
            GPIO.output(17,GPIO.HIGH)
            time.sleep(1)
            GPIO.output(17,GPIO.LOW)
            time.sleep(1)
            count = count - 1
```

SOS Buzzer

```
#!/usr/bin/python
import os
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(22,GPIO.OUT)
loop_count = 0
def morsecode ():
      #Dot Dot Dot
      GPIO.output(22,GPIO.HIGH)
      time.sleep(.1)
      GPIO.output(22,GPIO.LOW)
      time.sleep(.1)
      GPIO.output(22,GPIO.HIGH)
      time.sleep(.1)
      GPIO.output(22,GPIO.LOW)
      time.sleep(.1)
      GPIO.output(22,GPIO.HIGH)
      time.sleep(.1)
      #Dash Dash Dah
      GPIO.output(22,GPIO.LOW)
      time.sleep(.2)
      GPIO.output(22,GPIO.HIGH)
      time.sleep(.2)
      GPIO.output(22,GPIO.LOW)
      time.sleep(.2)
      GPIO.output(22,GPIO.HIGH)
      time.sleep(.2)
      GPIO.output(22,GPIO.LOW)
      time.sleep(.2)
      GPIO.output(22,GPIO.HIGH)
      time.sleep(.2)
      GPIO.output(22,GPIO.LOW)
      time.sleep(.2)
      #Dot Dot Dot
      GPIO.output(22,GPIO.HIGH)
      time.sleep(.1)
      GPIO.output(22,GPIO.LOW)
      time.sleep(.1)
      GPIO.output(22,GPIO.HIGH)
```

```
        time.sleep(.1)
        GPIO.output(22,GPIO.LOW)
        time.sleep(.1)
        GPIO.output(22,GPIO.HIGH)
        time.sleep(.1)
        GPIO.output(22,GPIO.LOW)
        time.sleep(.7)
os.system('clear')
print "Morse Code"
loop_count = input("How many times would you like SOS to loop?: ")
while loop_count > 0:
        loop_count = loop_count - 1
        morsecode ()
```

Temperature Sensor

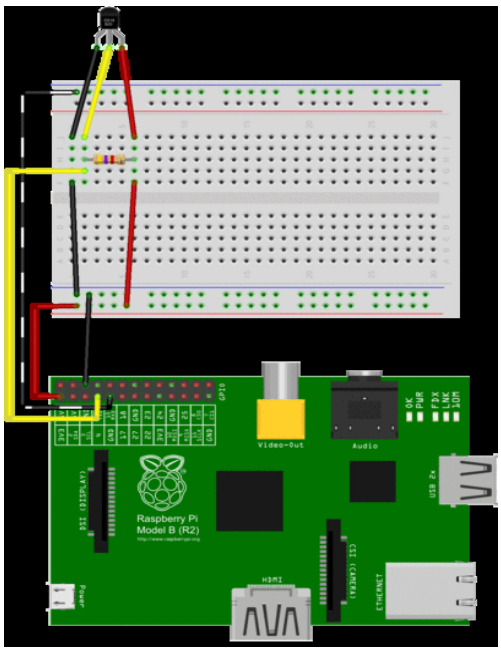


1-Wire is really just a simple system of devices such as weather monitoring devices, garden monitoring devices, or home automation devices that are connected through a 1-Wire network to a computer (or other controller – more on that later). Your computer then receives signals from the devices, allowing you to track rainfall, temperature, humidity, soil moisture levels, lightning, and a host of other information.

So how does it work? Well, a 1-Wire network consists of a master controller which is connected to one or many slave devices. The master controller is typically a computer or microcontroller with an external 1-Wire interface such as our Serial Adaptor or USB Adaptor. All of the actual monitoring devices (lightning detector, moisture meter, motion detector, barometer, etc.) are slave devices. The master communicates with one or more slave device(s) using the serial 1-Wire protocol developed by Dallas Semiconductor, sending and receiving signals over a single data line plus ground reference. The 1-Wire protocol synchronizes the slave devices to the master. The master initiates and controls all activities on the 1-Wire network.

One key feature of the Dallas system is that every 1-Wire slave device has a guaranteed unique address.

Circuit Diagram:



CONFIGURING YOUR PI

Before you can use any 1-wire devices you must first tell the Raspberry Pi how to read them. Open a Terminal Window and type the following to edit the Raspberry Pi's configuration file:

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

Look to see whether there is a line that has dtoverlay=w1-gpio in it. If not, add the following to the end of the file:

```
dtoverlay=w1-gpio
```

Now reboot the Pi:

```
sudo reboot
```

To test the configuration, set-up the circuit above to connect the DS18B20 and type the following into a terminal window:

```
sudo modprobe w1-gpio
sudo modprobe w1-therm
cd /sys/bus/w1/devices
ls
```

This will list all the devices that are connected to the 1-wire interface. The Dallas DS18B20 sensor starts with '28-' followed by a long number. Type in the following, replacing the 'xxxx' with the text following the '28-':

```
cd 28-xxxx
cat w1_slave
```

In response, you should get the following showing that the DS18B20 is working:

a3 01 4b 46 7f ff 0e 10 d8 : crc=d8 YES

a3 01 4b 46 7f ff 0e 10 d8 t=32768

READING FROM THE SENSOR

Create a new python script either from a terminal window (with nano 3-temperature.py) or from IDLE, the 'Interactive Development Environment'. Enter the following example code:

```
# Import Libraries
import os
import glob
import time

# Initialize the GPIO Pins
os.system('modprobe w1-gpio')  # Turns on the GPIO module
os.system('modprobe w1-therm') # Turns on the Temperature module

# Finds the correct device file that holds the temperature data
base_dir = '/sys/bus/w1/devices/'
device_folder = glob.glob(base_dir + '28*')[0]
device_file = device_folder + '/w1_slave'

# A function that reads the sensors data
def read_temp_raw():
```

```
  f = open(device_file, 'r') # Opens the temperature device file
  lines = f.readlines() # Returns the text
  f.close()
  return lines

# Convert the value of the sensor into a temperature
def read_temp():
  lines = read_temp_raw() # Read the temperature 'device file'

  # While the first line does not contain 'YES', wait for 0.2s
  # and then read the device file again.
  while lines[0].strip()[-3:] != 'YES':
    time.sleep(0.2)
    lines = read_temp_raw()

  # Look for the position of the '=' in the second line of the
  # device file.
  equals_pos = lines[1].find('t=')

  # If the '=' is found, convert the rest of the line after the
  # '=' into degrees Celsius, then degrees Fahrenheit
  if equals_pos != -1:
    temp_string = lines[1][equals_pos+2:]
    temp_c = float(temp_string) / 1000.0
    temp_f = temp_c * 9.0 / 5.0 + 32.0
    return temp_c, temp_f

# Print out the temperature until the program is stopped.
while True:
  print(read_temp())
  time.sleep(1)
```

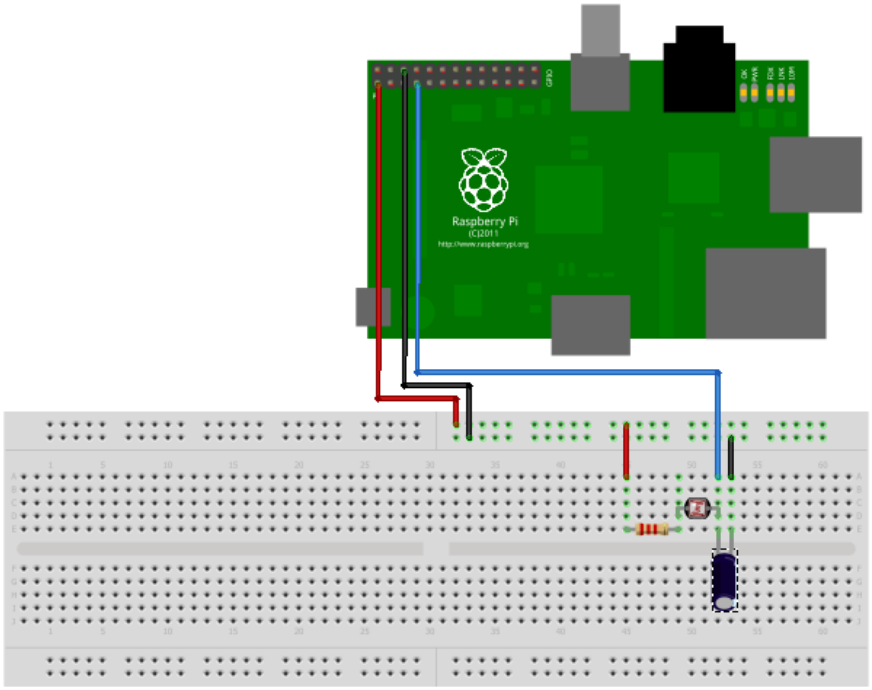
To run the code, you need to be the Super User, so run the code with:

```
sudo python temperature.py
```

Light Dependent Resistor

```
#!/usr/bin/env python
import os
import datetime
import time
import RPi.GPIO as GPIO
GPIO.setwarnings(False)
DEBUG = 1
GPIO.setmode(GPIO.BCM)
 def RCtime (RCpin):
        reading = 0
        GPIO.setup(RCpin, GPIO.OUT)
        GPIO.output(RCpin, GPIO.LOW)
        time.sleep(.1)
        GPIO.setup(RCpin, GPIO.IN)
        # This takes about 1 millisecond per loop cycle
        while (GPIO.input(RCpin) == GPIO.LOW):
                reading += 1
        return reading
while True:
             GetDateTime = datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S")
             LDRReading = RCtime(3)
             print RCtime(3)
             # Open a file
             fo = open("/home/pi/Desktop/gpio_python_code/foo.txt", "wb")
             fo.write (GetDateTime)
             LDRReading = str(LDRReading)
             fo.write ("\n")
             fo.write (LDRReading)
             # Close opend file
             fo.close()
             time.sleep(1)
```

LDR




```
#!/usr/bin/env python
import os
import datetime
import time
import RPi.GPIO as GPIO
GPIO.setwarnings(False)
DEBUG = 1
GPIO.setmode(GPIO.BCM)
def RCtime (RCpin):
        reading = 0
        GPIO.setup(RCpin, GPIO.OUT)
        GPIO.output(RCpin, GPIO.LOW)
        time.sleep(.1)

        GPIO.setup(RCpin, GPIO.IN)
        # This takes about 1 millisecond per loop cycle
        while (GPIO.input(RCpin) == GPIO.LOW):
                reading += 1
        return reading

while True:
             GetDateTime = datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S")
             LDRReading = RCtime(3)
             print RCtime(3)
```

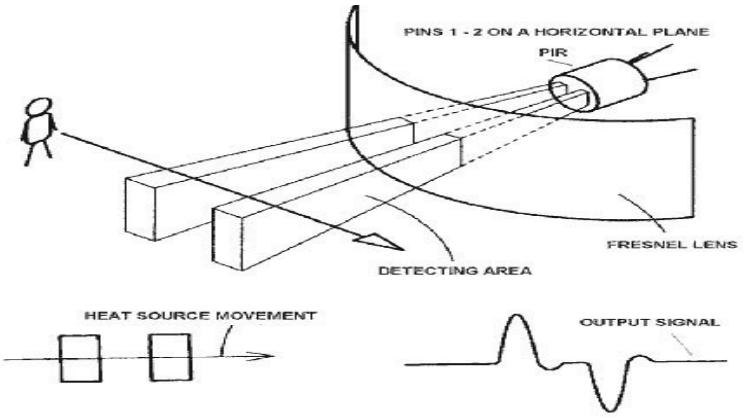
```
 # Open a file
fo = open("/home/pi/Desktop/gpio_python_code/foo.txt", "wb")
fo.write (GetDateTime)
LDRReading = str(LDRReading)
fo.write ("\n")
fo.write (LDRReading)

# Close opend file
fo.close()
time.sleep(1)
```

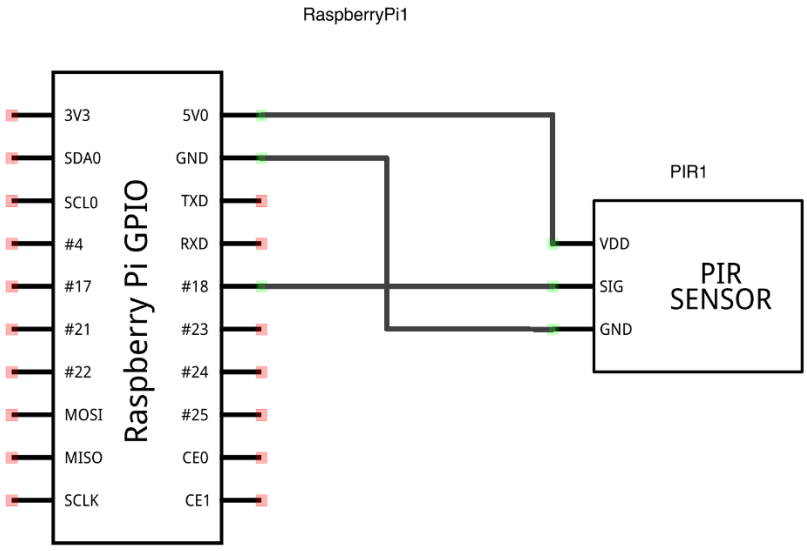
Passive Infrared Sensor

How PIRs Work

PIR sensors are more complicated than many of the other sensors explained in these tutorials (like photocells, FSRs and tilt switches) because there are multiple variables that affect the sensors input and output. To begin explaining how a basic sensor works, we'll use this rather nice diagram (if anyone knows where it originates plz let me know).

The PIR sensor itself has two slots in it, each slot is made of a special material that is sensitive to IR. The lens used here is not really doing much and so we see that the two slots can 'see' out past some distance (basically the sensitivity of the sensor). When the sensor is idle, both slots detect the same amount of IR, the ambient amount radiated from the room or walls or outdoors. When a warm body like a human or animal passes by, it first intercepts one half of the PIR sensor, which causes a positive differential change between the two halves. When the warm body leaves the sensing area, the reverse happens, whereby the sensor generates a negative differential change. These change pulses are what is detected.





```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(27,GPIO.OUT)
GPIO_PIR = 7
print "PIR Module Test (CTRL-C to exit)"
# Set pin as input
GPIO.setup(GPIO_PIR,GPIO.IN)      # Echo
Current_State  = 0
Previous_State = 0
try:
  print "Waiting for PIR to settle ..."
  # Loop until PIR output is 0
  while GPIO.input(GPIO_PIR)==1:
    Current_State  = 0
  print "  Ready"
  # Loop until users quits with CTRL-C
  while True :
    # Read PIR state
    Current_State = GPIO.input(GPIO_PIR)
    if Current_State==1 and Previous_State==0:
      # PIR is triggered
      print "  Motion detected!"
      # Record previous state
      GPIO.output(27,GPIO.HIGH)
      time.sleep(1)
      GPIO.output(27,GPIO.LOW)
      Previous_State=1
    elif Current_State==0 and Previous_State==1:
      # PIR has returned to ready state
      print "  Ready"
      Previous_State=0
    # Wait for 10 milliseconds
    time.sleep(0.01)
except KeyboardInterrupt:
  print "  Quit"
  # Reset GPIO settings
  GPIO.cleanup()
```

PiCamera Module:

class picamera.PiCamera

Provides a pure Python interface to the Raspberry Pi's camera module.

Upon construction, this class initializes the camera. As there is only a single camera supported by the Raspberry Pi, this means that only a single instance of this class can exist at any given time (it is effectively a singleton class although it is not implemented as such).

No preview or recording is started automatically upon construction. Use the capture() method to capture image, the start_recording() method to begin recording video, or the start_preview() method to start live display of the camera's input.

Start a preview for 10 seconds with the default settings:

```
import time
import picamera
camera = picamera.PiCamera()
try:
    camera.start_preview()
    time.sleep(10)
    camera.stop_preview()
finally:
    camera.close()
```

Note that you should always ensure you call close() on the PiCamera object to clean up resources. The following example demonstrates that the context manager protocol can also be used to achieve this:

```
import time
import picamera
with picamera.PiCamera() as camera:
    camera.start_preview()
    time.sleep(10)
    camera.stop_preview()
```

The next example demonstrates setting the camera resolution (this can only be done when the camera is not recording or previewing) to 640x480, then starting a preview and a recording to a disk file:

```
import picamera
with picamera.PiCamera() as camera:
    camera.resolution = (640, 480)
    camera.start_preview()
    camera.start_recording('foo.h264')
    camera.wait_recording(60)
    camera.stop_recording()
    camera.stop_preview()
```

Note that wait_recording() is used above instead of time.sleep(). This method checks for errors (e.g. out of disk space) while the recording is running and raises an exception if one occurs. If time.sleep() was used instead the exception would be raised by stop_recording() but only after the full waiting time had run.

Capturing an image

capture(output, format=None, **options)

Capture an image from the camera, storing it in output.

If output is a string, it will be treated as a filename for a new file which the image will be written to. Otherwise, output is assumed to a be a file-like object and the image data is appended to it (the implementation only assumes the object has a write() method - no other methods will be called).

If format is None (the default), the method will attempt to guess the required image format from the extension of output (if it's a string), or from the name attribute of output (if it has one). In the case that the format cannot be determined, a PiCameraValueError will be raised.

If format is not None, it must be a string specifying the format that you want the image written to. The format can be a MIME-type or one of the following strings:

'jpeg' - Write a JPEG file

'png' - Write a PNG file

'gif' - Write a GIF file

'bmp' - Write a bitmap file

```
import time
import picamera
camera=picamera.PiCamera()
try:
	camera.start_preview()
	time.sleep(10)
	camera.capture('image.jpg')
	camera.stop_preview()
finally:
	camera.close()
```

The next example demonstrates capturing a series of images as a numbered series with a one minute delay between each capture using the capture_continuous() method:

capture_continuous(output, format=None, use_video_port=False, **options)[source]

Capture images continuously from the camera as an infinite iterator.

This method returns an infinite iterator of images captured continuously from the camera. If output is a string, each captured image is stored in a file named after output after substitution of two values with the format() method. Those two values are:

{counter} - a simple incrementor that starts at 1 and increases by 1 for each image taken

{timestamp} - a datetime instance

The table below contains several example values of *output* and the sequence of filenames those values could produce:

***output* Value**	**Filenames**
`'image{counter}.jpg'`	image1.jpg, image2.jpg, image3.jpg, ...
`'image{counter:02}.jpg'`	image01.jpg, image02.jpg, image03.jpg, ...
`'image{timestamp}.jpg'`	image2013-10-05 12:07:12.346743.jpg, image2013-10-05 12:07:32.498539, ...
`'image{timestamp:%H-%M-%S-%f}.jpg'`	image12-10-02-561527.jpg, image12-10-14-905398.jpg
`'{timestamp:%H%M%S}-{counter:03d}.jpg'`	121002-001.jpg, 121013-002.jpg, 121014-003.jpg, ...

The use_video_port parameter controls whether the camera's image or video port is used to capture images. It defaults to False which means that the camera's image port is used. This port is slow but produces better quality pictures. If you need rapid capture up to the rate of video frames, set this to True.

For example, to capture 6 images with a one second delay between them, writing the output to a series of JPEG files named image01.jpg, image02.jpg, etc. one could do the following:

```
import time
import picamera
with picamera.PiCamera() as camera:
    camera.resolution = (1280, 720)
    camera.start_preview()
    time.sleep(1)
    for i, filename in
enumerate(camera.capture_continuous('image{counter:02d}.jpg')):
        print('Captured image %s' % filename)
        if i == 5:
            break
        time.sleep(1)
    camera.stop_preview()
```

This example demonstrates capturing low resolution JPEGs extremely rapidly using the video-port capability of the capture_sequence() method. The framerate of the captures is displayed afterward:

capture_sequence(outputs, format='jpeg', use_video_port=False, **options)[source]

Capture a sequence of consecutive images from the camera.

This method accepts a sequence or iterator of outputs each of which must either be a string specifying a filename for output, or a file-like object with a write method. For each item in the sequence or iterator of outputs, the camera captures a single image as fast as it can.

The format and options parameters are the same as in capture(), but format defaults to 'jpeg'. The format is _not_ derived from the filenames in outputs by this method.

The use_video_port parameter controls whether the camera's image or video port is used to capture images. It defaults to False which means that the camera's image port is used. This port is slow but produces better quality pictures. If you need rapid capture up to the rate of video frames, set this to True.

For example, to capture 20 consecutive images:

```
import time
import picamera
with picamera.PiCamera() as camera:
    camera.resolution = (640, 480)
    camera.start_preview()
    start = time.time()
    camera.capture_sequence((
        'image%03d.jpg' % i
        for i in range(20)
        ), use_video_port=True)
    print('Captured 20 images at %.2ffps' % (20 / (time.time() - start)))
    camera.stop_preview()
```

This example demonstrates capturing an image in raw RGB format:

```
import time
import picamera
with picamera.PiCamera() as camera:
    camera.resolution = (1024, 768)
    camera.raw_format = 'rgb'
    camera.start_preview()
    time.sleep(2)
    camera.capture('image.data', 'raw')
```

Interfacing PiCamera, PIR and Buzzer

```
import picamera
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(18,GPIO.OUT)
GPIO_PIR = 7
GPIO.setup(22,GPIO.OUT)


def morsecode ():
 #Dot Dot Dot
        GPIO.output(22,GPIO.HIGH)
        time.sleep(.1)
        GPIO.output(22,GPIO.LOW)
        time.sleep(.1)
        GPIO.output(22,GPIO.HIGH)
        time.sleep(.1)
        GPIO.output(22,GPIO.LOW)
        time.sleep(.1)
        GPIO.output(22,GPIO.HIGH)
#Dash Dash Dah
        GPIO.output(22,GPIO.LOW)
        time.sleep(.2)
        GPIO.output(22,GPIO.HIGH)
        time.sleep(.2)
        GPIO.output(22,GPIO.LOW)
        time.sleep(.2)
        GPIO.output(22,GPIO.HIGH)
        time.sleep(.2)
#Dash Dash Dah
        GPIO.output(22,GPIO.LOW)
        time.sleep(.2)
        GPIO.output(22,GPIO.HIGH)
        time.sleep(.2)
        GPIO.output(22,GPIO.LOW)
        time.sleep(.2)
        GPIO.output(22,GPIO.HIGH)
        time.sleep(.2)
        GPIO.output(22,GPIO.LOW)
        time.sleep(.2)
        GPIO.output(22,GPIO.HIGH)
```

```
        time.sleep(.2)
        GPIO.output(22,GPIO.LOW)
        time.sleep(.2)
def camclick():
                GPIO.output(18,GPIO.HIGH)
                with picamera.PiCamera() as camera:
                        camera.resolution = (1280,720)
                        camera.start_preview()
                        time.sleep(1)
                        for i, filename in
enumerate(camera.capture_continuous('image{timestamp}.jpg')):
                                print('Captured image %s' % filename)
                                if i==3:
                                      break
                                time.sleep(.1)
                         camera.stop_preview()
                GPIO.output(18, GPIO.LOW)
print "PIR Module Test (CTRL-C to exit)"
# Set pin as input
GPIO.setup(GPIO_PIR,GPIO.IN)
 # Echo
Current_State  = 0
Previous_State = 0
try:
  print "Waiting for PIR to settle ..."
  # Loop until PIR output is 0
  while GPIO.input(GPIO_PIR)==1:
    Current_State  = 0
  print "  Ready"
  # Loop until users quits with CTRL-C
  while True :
    # Read PIR state
    Current_State = GPIO.input(GPIO_PIR)
    if Current_State==1 and Previous_State==0:
      # PIR is triggered
      print "  Motion detected!"
      # Record previous state
      camclick()
      morsecode()
      Previous_State=1
    elif Current_State==0 and Previous_State==1:
      # PIR has returned to ready state
      print "  Ready"
```

```
      Previous_State=0
  # Wait for 10 milliseconds
    time.sleep(0.01)
except KeyboardInterrupt:
  print "  Quit"
  # Reset GPIO settings
  GPIO.cleanup()
```

Playing Music

```
import pygame.mixer

pygame.mixer.init()

drum = pygame.mixer.Sound("samples/drum_tom_mid_hard.wav")

while True:

    drum.play()
```

Here, we import the audio mixer module of the PyGame library and initialise it.

Then we create a reference to one of the sample sound files.

The while True is a continuous loop containing a command to play the sound file. It will keep playing the drum sound repeatedly until the program is terminated.

If you can't hear the sound, or it's coming out of the wrong speakers, you'll need to change your audio configuration.

Return to the terminal window and type the following command:

```
amixer cset numid=3 1
```

to switch audio to the headphone jack, or

```
amixer cset numid=3 2
```

to switch to HDMI

WIRE UP FIRST BUTTON

Now we've configured the audio and tested playing sound in Python, we'll connect the GPIO button.

Firstly, observe the following GPIO diagram. You'll be using a single ground pin (marked GND) and several GPIO pins (marked GPIO):

Note that if you have an older Raspberry Pi model you'll only have 26 pins but they have the same layout, starting at the top row (3V3 and 5V and ending at GND and GPIO7).

1. Find a ground pin (marked GND) on the diagram of the Raspberry Pi's pin layout above.

2. Attach a wire to a ground pin on the Raspberry Pi and connect it to the ground rail on your breadboard like so:

3. Place the button on the breadboard and connect one of its feet to the ground rail.

4. Connect the button's other foot (on the same side) to GPIO pin 2 like so:

CONNECT FIRST BUTTON TO SOUND FILE

Now we've connected a GPIO button, we'll make the sound play when the button is pressed.

1. Return to the code window and amend the code to add the GPIO library and set up pin 2, so it looks like this:

```
import pygame.mixer
```

```
from RPi import GPIO

pygame.mixer.init()

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

GPIO.setup(2, GPIO.IN, GPIO.PUD_DOWN)

drum = pygame.mixer.Sound("samples/drum_tom_mid_hard.wav")
```

This sets up GPIO pin 2 as an input, so you can trigger an event with the button.

Now add a function for playing the sound. This will be the code than runs when the button is pressed. You'll need to add this after the drum = line:

```
def play(pin):
    print("playing")
    drum.play()
```

The print will tell you when the function has been called, so you know what's going on.

Create a GPIO event that will run the play function when the button is pressed. Add the following line

```
GPIO.add_event_detect(2, GPIO.FALLING, play, 100)
```

The arguments passed to the function are:

the GPIO pin number (2)

the type of voltage change (FALLING)

the function to be used as the callback (play)

the amount of time allowed between button presses (100 milliseconds)

This uses an advanced feature called a threaded callback which means it will run the code in the function when it detects the button has been pressed. We set up the event on GPIO pin 2, which is what our button is connected to. The event/callback feature passes in the pin number to the function - but we're not using it for anything yet.

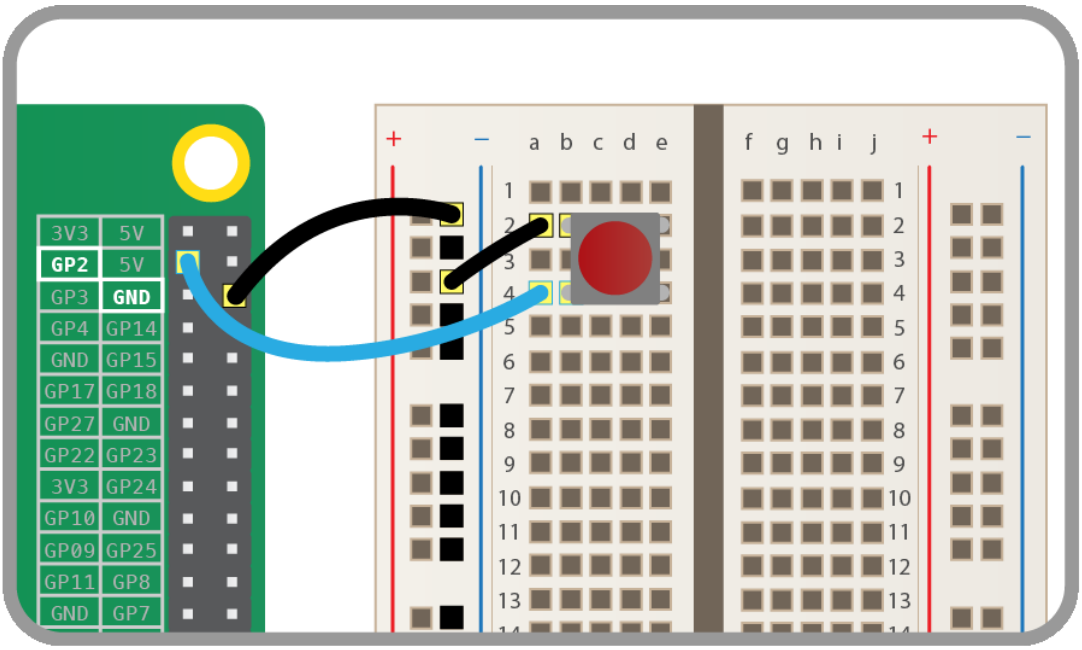
Add a line to print ready once it's all been set up, and add a while True loop to wait for a button press:

```
print("ready")
while True:
      pass
```

This goes at the very end of the file.

Run the program again - and once you see ready printed to the screen, press the button and you should hear the drum sound played. Each time you press the button it should print playing to the screen and play the sound.

If you do not see the word playing when you press the button, check you have it wired to the ground rail and pin 2, and that the cables are securely connected.



ADD SECOND BUTTON

Now that we've added an event for the first button to trigger the drum sound, we'll connect a second button and map that to a different sound.

1. Add a second button to the breadboard and wire it up to the ground rail and to GPIO pin 3 like so:

In the code, add a cymbal sound in the same way:

2. `drum = pygame.mixer.Sound("samples/drum_tom_mid_hard.wav")`

`cymbal = pygame.mixer.Sound("samples/drum_cymbal_open.wav")`

We also need to perform the GPIO.setup() on pin 3 as well as pin 2. Rather than just copy this line, we'll automate the process.

Normally we'd use a list like pins = [2, 3, 4, 5] and use a loop to run the setup for each item in the list. However, as this time we also need a list of sounds that correspond to GPIO pins, we'll use another data structure called a dictionary which is used to store relationships between items.

After the cymbal line, create a dictionary mapping the two GPIO pins to their respective sounds, like so:

`sound_pins = { 2: drum, 3: cymbal}`

This means you can look up which sound to play by passing in the pin number; for examplesound_pins[2] yields drum and sound_pins[3] yields cymbal.

Now where we previously had the GPIO.setup() line for pin 2, we'll use a loop to set up all the pins in the sound_pins dictionary.

Use the following loop to set up each pin:

8. for pin in sound_pins:

GPIO.setup(pin, GPIO.IN, GPIO.PUD_DOWN)

Looping over a dictionary like this yields the dictionary keys (the left hand side), and passing a key into a dictionary yields the corresponding value. For the pin setup all we need is the pin numbers.

The other thing we want to do for each pin is set up the event detection.

We could write a new function to play each particular sound, and add each event detection separately mapped to the appropriate function. However, it is possible to use a single function that determines which sound to play according to which pin was triggered.

Move the previously used add_event_detect() line into this loop, replacing the hard-coded value of 2 for the loop variable pin:

```
for pin in sound_pins:
    GPIO.setup(pin, GPIO.IN, GPIO.PUD_DOWN)
    GPIO.add_event_detect(pin, GPIO.FALLING, play, 100)
```

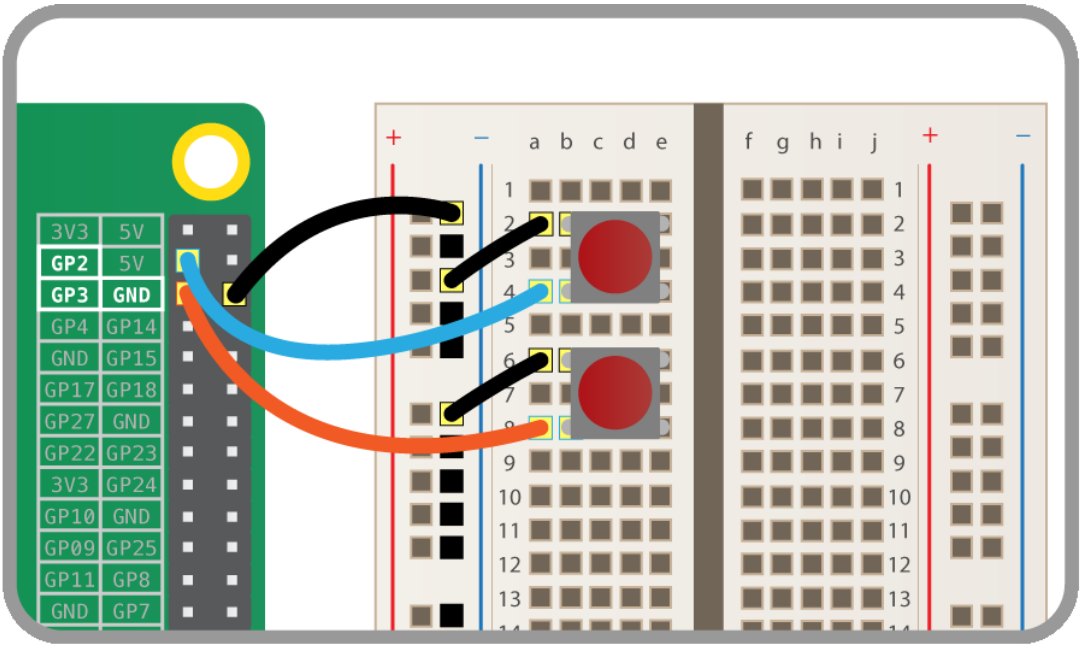
Now the play function will be called when pins 2 or 3 are triggered (by their connected buttons being pressed). However, we need to amend the play function to make it play the right sound according to which button was pressed.

Amend the play function like so:

```
def play(pin):
    sound = sound_pins[pin]
    print("playing note from pin %s" % pin)
    sound.play()
```

Because the event detection callback sends the GPIO pin number to the play function, we can look it up in the sound_pins dictionary, set sound to the appropriate sound and then play it.

Now run the program again and when you see the ready message, try pressing each button. Different sounds should be played when each button is pressed.



CONNECT MORE BUTTONS

Now we've done all the hard work, it's really easy to connect more buttons to make more sounds!

For each extra button, all you need to do is:

1. Connect the button to the breadboard and wire it to the ground rail and another GPIO pin. Make sure it's a pin marked GPIO in the pin diagram above.

2. Add a reference to the new sound file:

3. `bell = pygame.mixer.Sound("sounds/elec_bell.wav")`

`snare = pygame.mixer.Sound("sounds/elec_hi_snare.wav")`

4. Add the pin number and sound variable to the sound_pins dictionary:

5. `sound_pins = {`

6. `  2: drum,`

7. `    3: cymbal,`

8. `  4: bell,`

9. `    14: snare,`

`}`

And that's it! Re-run the program and the new buttons should make new sounds!

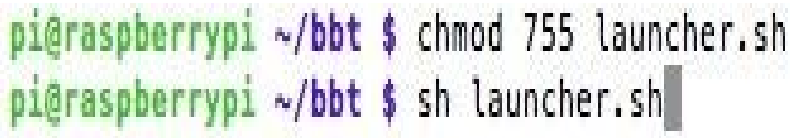
Auto run during start-up of RPi and execution of multiple programs simultaneously:

Step 1: Make a launcher sript

The python script is called : bbt.py and lives in a directory called bbt that is in the root directory. You can substitute your own director/Python script name.

We will use the Linux crontab to run the Python script.

Step 2: Make it executable



We need to make the launcher script an executable, which we do with this command

```
chmod 755 launcher.sh
```

Now test it, by typing in:

```
sh launcher.sh
```

This should run your Python code.

Step 3: Add to your crontab

crontab is a background (daemon) process that lets you execute scripts at specific times. It's essential to Python and Raspberry Pi.

Type in:

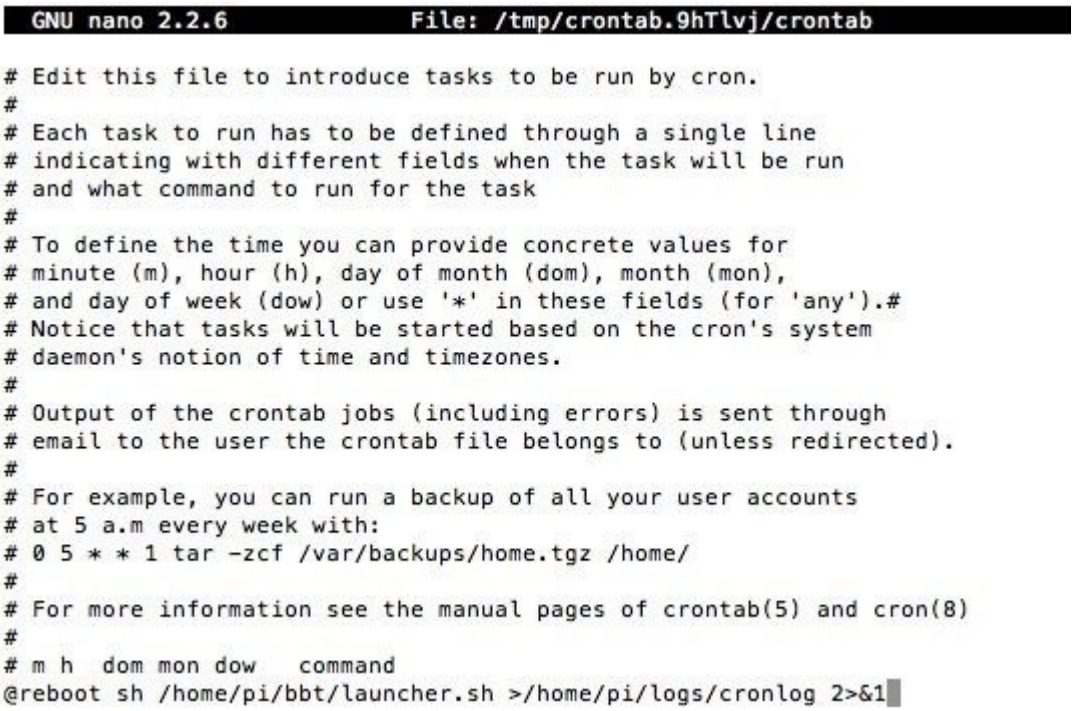
```
sudo crontab -e
```

This will brings up a crontab window.

Now, enter the line:

```
@reboot sh /home/pi/bbt/launcher.sh >/home/pi/logs/cronlog 2>&1
```

What this does is rather than executing the launcher script at a specific time, it will execute it once upon startup.



Step 4: Reboot and see if it works

Unplug the power or just type in:

```
sudo reboot
```

Wait for startup and see if your script automatically launches.

If it doesn't work, check out the log file:

```
cd logs
```

```
cat cronlog
```

This will show you any errors that you might have.

E-mail Notifier

Step 1: Prepare Python

In order for our Python code to work, we'll want to make sure a few libraries are installed.

First, from either the keyboard/monitor or SSH console type in:

```
sudo apt-get install python-pip
```

This part will probably take a little while.

Then you can install the IMAPClient Python library, which lets Python talk to most e-mail services:

```
sudo pip install imapclient
```

Link to download IMAPClient if the above step does not work:

https://pypi.python.org/pypi?:action=show_md5&digest=b980e22489c114c45df6f028297f2e25

tar -xvzf filename.tar.gz

f: this must be the last flag of the command, and the tar file must be immediately after. It tells tar the name and path of the compressed file.

z: tells tar to decompress the archive using gzip

x: tar can collect files or extract them. x does the latter.

v: makes tar talk a lot. Verbose output shows you all the files being extracted.

Then run the following in the LXTerminal:

```
sudo python setup.py install
```

Step 2: Wire LEDs

Step 3: Python Script

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


from imapclient import IMAPClient
import time
import RPi.GPIO as GPIO
DEBUG = True
HOSTNAME = 'imap.gmail.com'
USERNAME = 'your username here'
PASSWORD = 'your password here'
MAILBOX = 'Inbox'
NEWMAIL_OFFSET = 1   # my unread messages never goes to zero, yours might
MAIL_CHECK_FREQ = 60 # check mail every 60 seconds
```

```
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GREEN_LED = 18
RED_LED = 23
GPIO.setup(GREEN_LED, GPIO.OUT)
GPIO.setup(RED_LED, GPIO.OUT)
def loop():
    server = IMAPClient(HOSTNAME, use_uid=True, ssl=True)
    server.login(USERNAME, PASSWORD)

    if DEBUG:
        print('Logging in as ' + USERNAME)
        select_info = server.select_folder(MAILBOX)
        print('%d messages in INBOX' % select_info['EXISTS'])
  folder_status = server.folder_status(MAILBOX, 'UNSEEN')
    newmails = int(folder_status['UNSEEN'])
    if DEBUG:
        print "You have", newmails, "new emails!"
    if newmails > NEWMAIL_OFFSET:
        GPIO.output(GREEN_LED, True)
        GPIO.output(RED_LED, False)
    else:
        GPIO.output(GREEN_LED, False)
        GPIO.output(RED_LED, True)
    time.sleep(MAIL_CHECK_FREQ)
if __name__ == '__main__':
    try:
        print 'Press Ctrl-C to quit.'
        while True:
            loop()
  finally:
        GPIO.cleanup()
```

Next up, we'll mark the file as executable, so that it can run as a standalone program:

```
chmod +x checkmail.py
```

Finally you can run the script! Type in:

```
sudo ./checkmail.py
```

Socket Programming:

Term	Description
domain	The family of protocols that is used as the transport mechanism. These values are constants such as AF_INET, PF_INET, PF_UNIX, PF_X25, and so on.
type	The type of communications between the two endpoints, typically SOCK_STREAM for connection-oriented protocols and SOCK_DGRAM for connectionless protocols.
protocol	Typically zero, this may be used to identify a variant of a protocol within a domain and type.
hostname	The identifier of a network interface: ▫ A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation ▫ A string "<broadcast>", which specifies an INADDR_BROADCAST address. ▫ A zero-length string, which specifies INADDR_ANY, or ▫ An Integer, interpreted as a binary address in host byte order.
port	Each server listens for clients calling on one or more ports. A port may be a Fixnum port number, a string containing a port number, or the name of a service.

The socket Module:

To create a socket, you must use the socket.socket() function available in socket module, which has the general syntax –

```
s = socket.socket (socket_family, socket_type, protocol=0)
```

Here is the description of the parameters –

- socket_family: This is either AF_UNIX or AF_INET, as explained earlier.
- socket_type: This is either SOCK_STREAM or SOCK_DGRAM.
- protocol: This is usually left out, defaulting to 0.

Once you have socket object, then you can use required functions to create your client or server program. Following is the list of functions required –

Server Socket Methods

Method	Description
s.bind()	This method binds address (hostname, port number pair) to socket.
s.listen()	This method sets up and start TCP listener.
s.accept()	This passively accept TCP client connection, waiting until connection arrives (blocking).

Client Socket Methods

Method	Description
s.connect()	This method actively initiates TCP server connection.

General Socket Methods

Method	Description
s.recv()	This method receives TCP message
s.send()	This method transmits TCP message
s.recvfrom()	This method receives UDP message
s.sendto()	This method transmits UDP message
s.close()	This method closes socket
socket.gethostname()	Returns the hostname.

A simple echo server

To write Internet servers, we use the socket function available in socket module to create a socket object. A socket object is then used to call other functions to setup a socket server.

Now call bind(hostname, port) function to specify a port for your service on the given host.

Next, call the accept method of the returned object. This method waits until a client connects to the port you specified, and then returns a connection object that represents the connection to that client.

SERVER

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

import socket

host = ''

port = 50000

backlog = 5

size = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.bind((host,port))

s.listen(backlog)

while 1:

    client, address = s.accept()

    data = client.recv(size)

    if data:

        client.send(data)

    client.close()
```

A simple echo client

Let us write a very simple client program which opens a connection to a given port 12345 and given host. This is very simple to create a socket client using Python's socket module function.

The socket.connect(hostname, port) opens a TCP connection to hostname on the port. Once you have a socket open, you can read from it like any IO object. When done, remember to close it, as you would close a file.

The following code is a very simple client that connects to a given host and port, reads any available data from the socket, and then exits −

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

import socket

host = 'localhost'

port = 50000

size = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.connect((host,port))

s.send('Hello, world')

data = s.recv(size)
```

```
s.close()
print 'Received:', data
```

Chat Server

Server Code:

```
#!/usr/bin/env python
import socket
host=''
port=50000
backlog=5
size=1024
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((host,port))
s.listen(backlog)
client,address=s.accept()
while 1:
            data=client.recv(size)
            print 'Received:' ,data
            p=raw_input()
            client.send(p)
```

File Transfer Protocol

Step 1: Install proftpd service

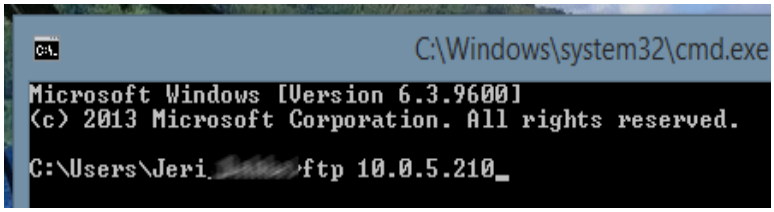
```
sudo apt-get install proftpd
```

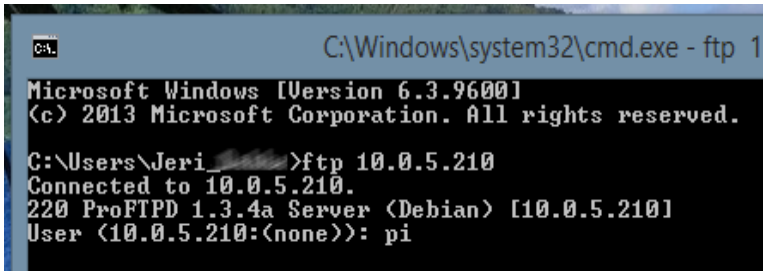
Step 2: Select 'Standalone' option

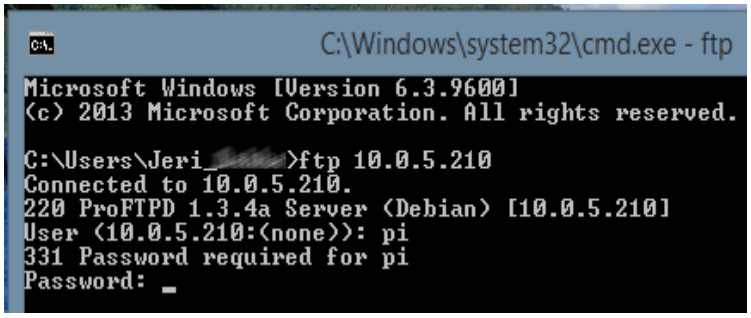
Step 3: To check status of proftpd

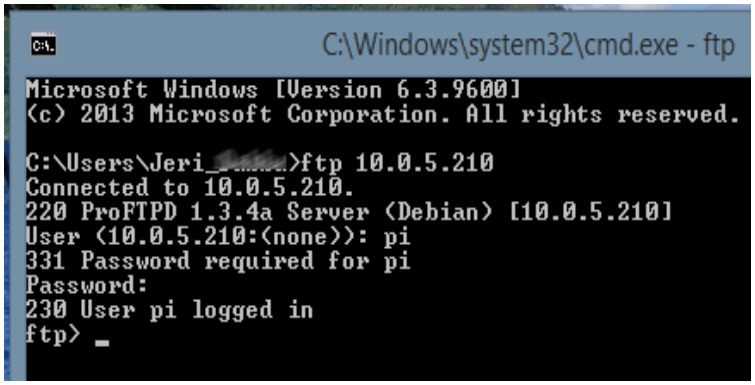
```
sudo service proftpd status
```

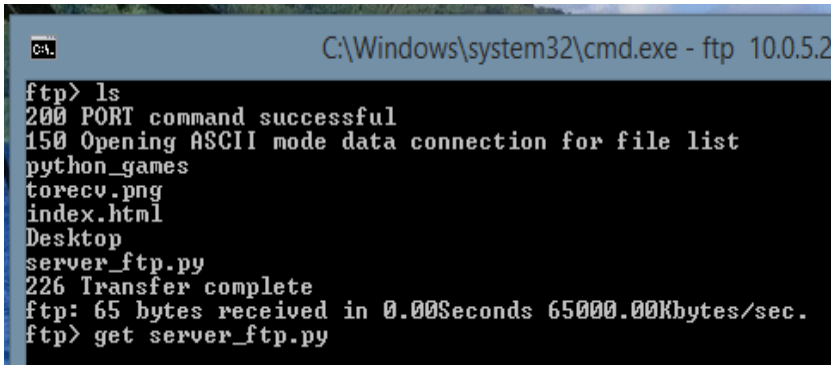
Step 4: Open cmd from your laptop and give the following command

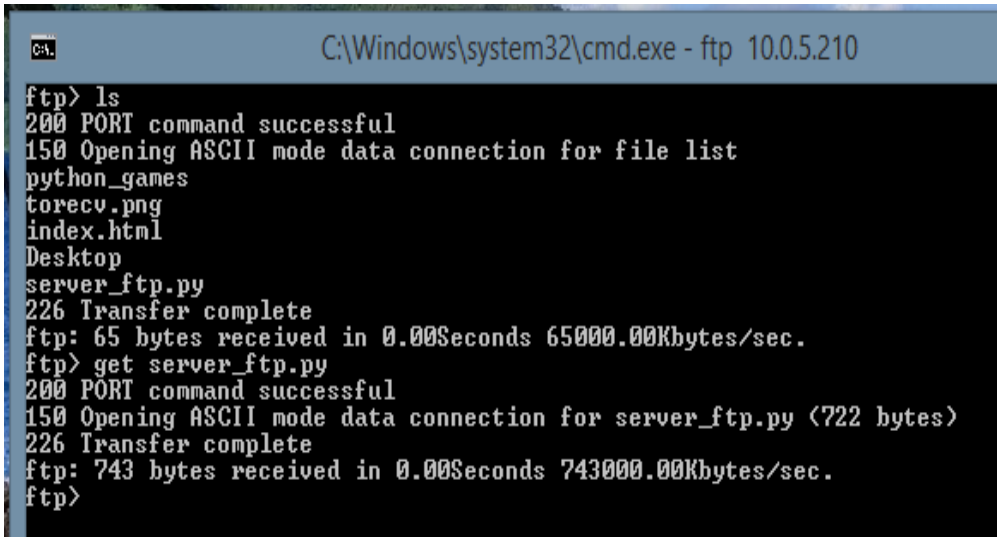












Raspberry Pi as a LAMP

Apache is a popular web server application you can install on the Raspberry Pi to allow it to serve web pages.

On its own, Apache can serve HTML files over HTTP, and with additional modules can serve dynamic web pages using scripting languages such as PHP.

INSTALL APACHE

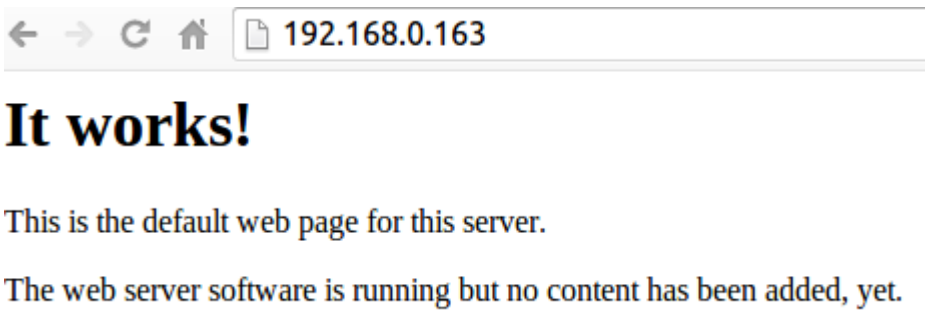
First install the apache2 package by typing the following command in to the Terminal:

```
sudo apt-get install apache2 -y
```

TEST THE WEB SERVER

By default, Apache puts a test HTML file in the web folder. This default web page is served when you browse to http://localhost/ on the Pi itself, or http://192.168.1.10 (whatever the Pi's IP address is) from another computer on the network. To find the Pi's IP address, type hostname -I at the command line (or read more about finding your IP address).

Browse to the default web page either on the Pi or from another computer on the network and you should see the following:



This means you have Apache working!

CHANGING THE DEFAULT WEB PAGE

This default web page is just a HTML file on the filesystem. It is located at /var/www/index.html. navigate to this directory in the Terminal and have a look at what's inside:

```
cd /var/www
```

```
ls -al
```

This will show you:

total 12

drwxr-xr-x 2 root root 4096 Jan 8 01:29 .

drwxr-xr-x 12 root root 4096 Jan 8 01:28 ..

-rw-r--r-- 1 root root 177 Jan 8 01:29 index.html

This shows that there is one file in /var/www/ called index.html. The . refers to the directory itself /var/www/ and the .. refers to the parent directory /var/.

WHAT THE COLUMNS MEAN

The permissions of the file or directory

The number of files in the directory (or 1 if it's a file).

The user which owns the file or directory

The group which owns the file or directory

The file size

The last modification date & time

As you can see, by default the www directory and index.html file are both owned by the root user. In order to edit the file, you must gain root permissions. Either change the owner to your own user (sudo chown pi: index.html) before editing, or edit with sudo, e.g. sudo nano index.html.

Try editing this file and refreshing the browser to see the web page change.

YOUR OWN WEBSITE

If you know HTML you can put your own HTML files and other assets in this directory and serve them as a website on your local network.

ADDITIONAL - INSTALL PHP

To allow your Apache server to process PHP files, you'll need to install PHP5 and the PHP5 module for Apache. Type the following command to install these:

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

Now remove the index.html file:

```
sudo rm index.html
```

and create the file index.php:

```
sudo nano index.php
```

and put some PHP content in it:

```
<?php echo "hello world";
```

Now save and refresh your browser. You should see "hello world". This is not dynamic but still served by PHP. Try something dynamic:

```
<?php echo date('Y-m-d H:i:s');
```

or show your PHP info:

```
<?php phpinfo();
```

INSTALL MYSQL

MySQL (pronounced My Sequel or My S-Q-L) is a popular database engine. Like PHP, its overwhelming presence on web servers enhanced its popularity. This is why projects like WordPress use it, and why those projects are so popular.

Install the MySQL Server and PHP-MySQL packages by entering the following command into the terminal:

sudo apt-get install mysql-server php5-mysql -y

When installing MySQL you will be asked for a root password. You'll need to remember this to allow your website to access the database.

DOWNLOAD WORDPRESS

You can download WordPress from wordpress.org using the wget command. Helpfully, a copy of the latest version of WordPress is always available at wordpress.org/latest.tar.gz and wordpress.org/latest.zip, so you can grab the latest version without having to look it up on the website. At the time of writing, this is version 4.0.

Navigate to /var/www/, and download WordPress to this location. You'll need to empty the folder first (be sure to check you're not deleting files you need before running rm); change the ownership of this folder to the pi user too.

```
cd /var/www

sudo chown pi: .

sudo rm *

wget http://wordpress.org/latest.tar.gz
```

Now extract the tarball, move the contents of the folder it extracted (wordpress) to the current directory and remove the (now empty) folder and the tarball to tidy up:

```
tar xzf latest.tar.gz

mv wordpress/* .

rm -rf wordpress latest.tar.gz
```

Running the ls or (tree -L 1) command here will show you the contents of a WordPress project:

```
├── index.php
```

```
├── license.txt
├── readme.html
├── wp-activate.php
├── wp-admin
├── wp-blog-header.php
├── wp-comments-post.php
├── wp-config-sample.php
├── wp-content
├── wp-cron.php
├── wp-includes
├── wp-links-opml.php
├── wp-load.php
├── wp-login.php
├── wp-mail.php
├── wp-settings.php
├── wp-signup.php
├── wp-trackback.php
└── xmlrpc.php
```

This is the source of a default WordPress installation. The files you edit to customise your installation belong in the wp-content folder.

SET UP YOUR WORDPRESS DATABASE

To get your WordPress site set up, you need a database. Run the mysql command in the terminal and provide your login credentials (e.g. username root, password password):

```
mysql -uroot -ppassword
```

Here I have provided my password (the word password) on the command line; there is no space between -p and your password.

Alternatively you can simply supply an empty -p flag and wait to be asked for a password:

```
mysql -uroot -p
```

Now you will be prompted to enter the root user password you created earlier.

Once you're connected to MySQL, you can create the database your WordPress installation will use:

```
mysql> create database wordpress;
```

Note the semi-colon ending the statement. On success you should see the following message:

Query OK, 1 row affected (0.00 sec)

Exit out of the MySQL prompt with Ctrl + D.

WORDPRESS CONFIGURATION

You need to find out your Pi's IP address to access it in the browser, so in a terminal type the command hostname -I.

Navigate to http://YOUR-IP-ADDRESS e.g. http://192.168.1.5 in the web browser on your Pi.

You should see a WordPress error page; this is good! Click the big button marked Create a Configuration File followed by the Let's go! button on the next page.

Now fill out the basic site information as follows:

Database Name: wordpress

User Name: root

Password: <YOUR PASSWORD>

Database Host: localhost

Table Prefix: wp_

Upon successful database connection, you will be given the contents of your wp-config.php file.

Interfacing RPi with GPS

Now the Raspberry PI has no onboard Real time clock – which means it needs to use an NTP server to get the time when it starts. Usually you would use the default settings and allow the PI to connect to the net for it's time. Now this is fine if you have a working net connection but what if you are not connected to the net? You might be in the field running the PI on batteries.

Now the obvious solution here is to use GPS as a time source. GPS works by having a constellation of satellites in orbit and each one carries a highly accurate atomic clock & broadcast both their current position and the time. A GPS receiver then receives these signals and, as long as it has enough satellites and workout where you are by comparing the times from those clocks.

Plug the GPS module to the RPi and run lsusb

```
pi@raspberrypi:~$ sudo lsusb
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.
Bus 001 Device 004: ID 067b:2303 Prolific Technology, Inc. PL2303 Serial Port
Bus 001 Device 002: ID 0424:9512 Standard Microsystems Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

There the Prolific Technology entry is the GPS appearing as a serial port. If you look in /var/log/syslog you will also notice it will have created the port as /dev/ttyUSB0 as it's the first serial port.

Using the PI as a GPS Receiver

Now the next step is to get the pi receiving data from the satellites. Now there is a suite of tools available for Linux called gpsd which we'll install:

```
pi@raspberrypi:~$ sudo apt-get install gpsd gpsd-clients python-gps
```

Next we need to start the daemon:

```
pi@raspberrypi:~$ sudo gpsd /dev/ttyUSB0 -F /var/run/gpsd.sock
```

Ignore any messages from the console or in the log files, you may see it complaining about IPv6 but you can ignore that.

Viewing whats in the sky & your location

Now GPS doesn't work indoors – as it needs a clear view of the sky so for this I've placed the PI on the window sill. Next I ssh into the pi and run cgps.

```
pi@raspberrypi:~$ cgps -s
```

The -s flag is there to tell the command not to write raw data to the screen as well as the processed data.

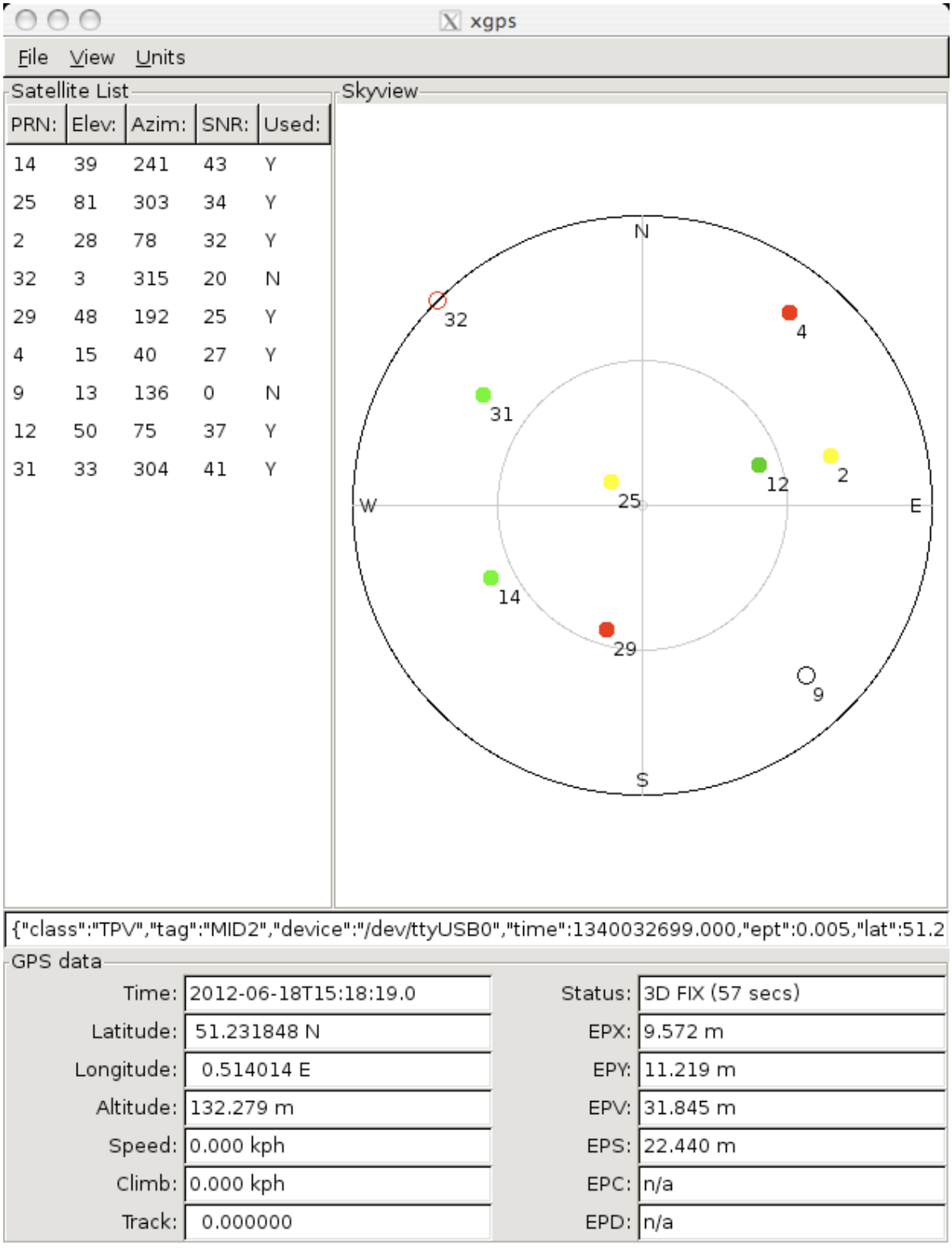
You should then get the following output:

```
Time:       2012-06-18T15:05:10.0Z     ||PRN:   Elev:  Azim:  SNR:  Used:
Latitude:    51.231848 N               ||  14    43    249    40      Y
Longitude:    0.514014 E               ||  25    75    283    37      Y
Altitude:   132.3 m                    ||   2    26    085    31      Y
Speed:      0.0 kph                    ||  12    56    070    18      Y
Heading:    0.0 deg (true)             ||   9    19    133    22      Y
Climb:      0.0 m/min                  ||  27    09    133    17      Y
Status:     3D FIX (1 secs)            ||   4    17    045    31      Y
GPS Type:                              ||  32    05    321    20      Y
Longitude Err:   +/- 8 m               ||  29    41    192    18      Y
Latitude Err:    +/- 9 m               ||  31    28    304    42      Y
Altitude Err:    +/- 27 m              ||
Course Err:      n/a                   ||
Speed Err:       +/- 68 kph            ||
                                       ||
```

Viewing GPS under X-Windows

Now above I showed how the GPS looks from an SSH connection but you can get a graphical display as well using the xgps client thats also been installed. Now if you have a monitor connected to the pi simply open a terminal and run xgps. However as I've not got a monitor against the window I've used ssh to connect to it from another machine. To get this to work you need to add -Y to the ssh command.

```
pi@raspberrypi:~$ xgps
```



Setting the computer time using GPS

Now we have a working GPS we can now get the PI to use it for setting the time. To do this we need to configure ntp to use the GPS satellites as a time source. Now you should already have ntp installed but if not then you need to install it:

```
pi@raspberrypi:~$ sudo apt-get install ntp
```

Next you need to edit the file: /etc/ntp.conf and add a few lines to it defining the GPS. This can be either before or after the existing lines beginning with server:

```
# gps ntp
server 127.127.28.0 minpoll 4
fudge  127.127.28.0 time1 0.183 refid NMEA
server 127.127.28.1 minpoll 4 prefer
fudge  127.127.28.1 refid PPS
```

Now restart ntp:

```
pi@raspberrypi:~$ sudo service ntp restart
```

```
pi@raspberrypi:~$ ntpq -p
remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*ns1.luns.net.uk 33.117.170.50    2 u   54   64    7   65.454    2.666   5.800
+resntp-b-vip.lo 127.151.91.34    3 u   45   64   17   55.704   -5.169   8.482
+bart.nexellent. 194.242.34.149   2 u   17   64   17   76.585   -4.271  57.595
+v01.s01.be.it2g 193.190.230.65   2 u   20   64   37   86.464   -2.374 228.460
xSHM(0)          .NMEA.           0 l   11   16  377    0.000  144.714   3.026
SHM(1)          .PPS.            0 l    -   16    0    0.000    0.000   0.000
```

A couple of notes:

You might find that ntp doesn't connect to the gps at first. It appears that it starts gpsd up without the link to the serial port. What I find I have to do is:

```
pi@raspberrypi:~$ sudo killall gpsd
```

```
pi@raspberrypi:~$ sudo gpsd /dev/ttyUSB0 -F /var/run/gpsd.sock
```

```
pi@raspberrypi:~$ sudo service ntp restart
```

Controlling Mains using Raspberry Pi

Don't try this unless you know what you are doing. Mains electricity can kill you.

What's a relay?

A relay is a device which allows a low power circuit to switch a higher power circuit. In the most popular relays, the switch is controlled by an electromagnet switch (solenoid). When there's current flowing through the low power circuit, an electromagnetic solenoid switch flips from one position to another, switching on/off the higher power circuit (depending on how it's set up). It makes an unmistakeably characteristic "click" sound. relay-click Click the play button to hear the relay click sound.