TABLE OF CONTENTS

1	**Displaying different LED patterns with Raspberry Pi.**

```
import RPi.GPIO as GPIO
import time

x=1
numTimes=int(input("Enter tottal number of times to blink"))
speed=float(input("Enter length of each blink(seconds) : "))

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(5,GPIO.OUT)
GPIO.setup(10,GPIO.OUT)
GPIO.setup(19,GPIO.OUT)
G PIO.setup(26,GPIO.OUT)
GPIO.setup(29,GPIO.OUT)

def Blink(numTimes,speed):
    for i in range(0,numTimes):
        GPIO.output(5,True)
        print ("Iteration ", (i+1))


        GPIO.output(10,True)
        print ("Iteration ", (i+1))


        GPIO.output(19,True)
        print ("Iteration ", (i+1))


        GPIO.output(26,True)
        print ("Iteration ", (i+1))


        GPIO.output(29,True)
        print ("Iteration ", (i+1))


        GPIO.output(29,False)
        print ("Iteration ", (i+1))
        time.sleep(speed)

        GPIO.output(26,False)
        print ("Iteration ", (i+1))
```

```
        time.sleep(speed)

        GPIO.output(19,False)
        print ("Iteration ", (i+1))
        time.sleep(speed)

        GPIO.output(10,False)
        print ("Iteration ", (i+1))
        time.sleep(speed)

        GPIO.output(5,False)
        print ("Iteration ", (i+1))
        time.sleep(speed)

Blink(numTimes,speed)
print("Done")
```

2	**Displaying Time over 4-Digit 7-Segment Display using Raspberry Pi.**

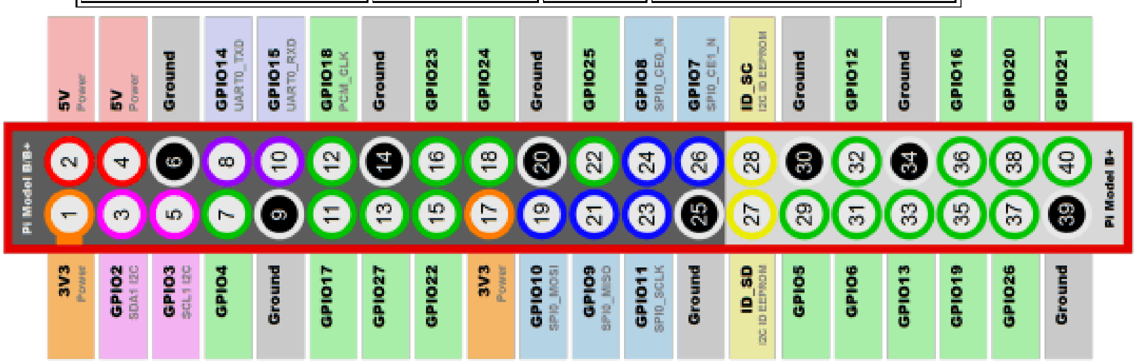
Control 4 digits-7 segments LED display with TM1637 controller

Connection scheme Raspberry Pi

Pi to

1. **TM1637 Board Pin**	**Function**	**RPI Pin**	**Raspberry Function**
2. GND	Ground	14	GND
3. VCC	+ 5V Power	4	5V
4. DI0	Data In	18	GPIO 24
5. CLK	Clock	16	GPIO 23

Connect the LED to your Raspberry according the following diagram:



TM1637 script

In order to control the LED, we use a special script with pre-defined functions. Various functions are available in the script, for example you can display numbers and adjust the intensity of the LEDs. Download the script with the command:

```
wget https://raspberrytips.nl/files/tm1637.py
```

Code:

```
import sys
import time
import datetime
import RPi.GPIO as GPIO
import tm1637

#CLK -> GPIO23 (Pin 16)
#Di0 -> GPIO24 (Pin 18)

Display = tm1637.TM1637(23,24,tm1637.BRIGHT_TYPICAL)

Display.Clear()
Display.SetBrightnes(1)

while(True):
   now = datetime.datetime.now()
   hour = now.hour
   minute = now.minute
   second = now.second
   currenttime = [ int(hour / 10), hour % 10, int(minute / 10), minute % 10 ]

   Display.Show(currenttime)
   Display.ShowDoublepoint(second % 2)

   time.sleep(1)
```

3	**Raspberry Pi Based Oscilloscope**

Project Requirements

The requirement for this project can be classified into two:

1. Hardware Requirements
2. Software Requirements

Hardware requirements

To build this project, the following components/part are required;

1. Raspberry pi 2 (or any other model)
2. 8 or 16GB SD Card
3. LAN/Ethernet Cable
4. Power Supply or USB cable
5. ADS1115 ADC
6. LDR (Optional as its meant for test)
7. 10k or 1k resistor
8. Jumper wires
9. Breadboard
10. Monitor or any other way of seeing the pi's Desktop(VNC inclusive)

Software Requirements

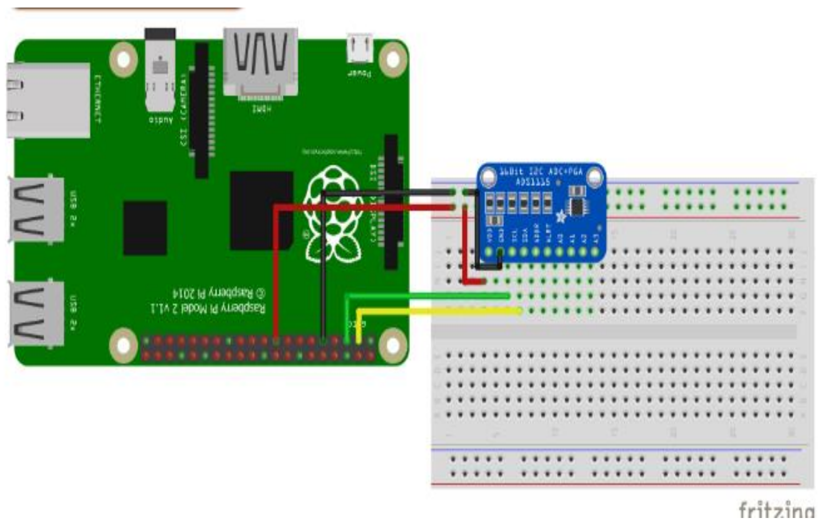
The software requirements for this project are basically the python modules (***matplotlib and drawnow***) that will be used for data visualization and the Adafruit module for interfacing with the ADS1115 ADC chip. I will show how to install these modules on the Raspberry Pi as we proceed.

While this tutorial will work irrespective of the raspberry pi OS used, I will be using the Raspberry Pi stretch OS and I will assume you are familiar with setting up the Raspberry Pi with the Raspbian stretch OS, and you know how to SSH into the raspberry pi using a terminal software like putty. If you have issues with any of this, there are tons of Raspberry Pi Tutorials on this website that can help.

With all the hardware components in place, let's create the schematics and connect the components together.

Circuit Diagram:

To convert the analog input signals to digital signals which can be visualized with the Raspberry Pi, we will be using the **ADS1115 ADC chip**. This chip becomes important because the Raspberry Pi, unlike Arduino and most micro-controllers, does not have an on-board analog to digital converter(ADC). While we could have used any raspberry pi compatible ADC chip, I prefer this chip due to its high resolution(16bits) and its well documented datasheet and use instructions by Adafruit. You can also check our Raspberry Pi ADC tutorial to learn more about it.



ADS1115 and Raspberry Pi Connections:

VDD – 3.3v

GND – GND

SDA – SDA

SCL – SCL

With the connections all done, power up your pi and proceed to install the dependencies mentioned below.

Install Dependencies for Raspberry Pi Oscilloscope:

Before we start writing the python script to pull data from the ADC and plot it on a live graph, we need to **enable the I2C communication interface** of the raspberry pi and install the software requirements that were mentioned earlier. This will be done in below steps so its easy to follow:

Step 1: Enable Raspberry Pi I2C interface

To enable the I2C, from the terminal, run;

```
sudo raspi-config
```

When the configuration panels open, select interface options, select I2C and click enable.

Step 2: Update the Raspberry pi

The first thing I do before starting any project is updating the Pi. Through this, I am sure every thing on the OS is up to date and I won't experience compatibility issue with any latest software I choose to install on the Pi. To do this, run below two commands:

sudo apt-get update

sudo apt-get upgrade

Step 3: Install the Adafruit ADS1115 library for ADC

With the update done, we are now ready to install the dependencies starting with the Adafruit python module for the ADS115 chip. Ensure you are in the Raspberry Pi home directory by running;

cd ~

then install the build-essentials by running;

sudo apt-get install build-essential python-dev python-smbus git

Next, clone the Adafruit git folder for the library by running;

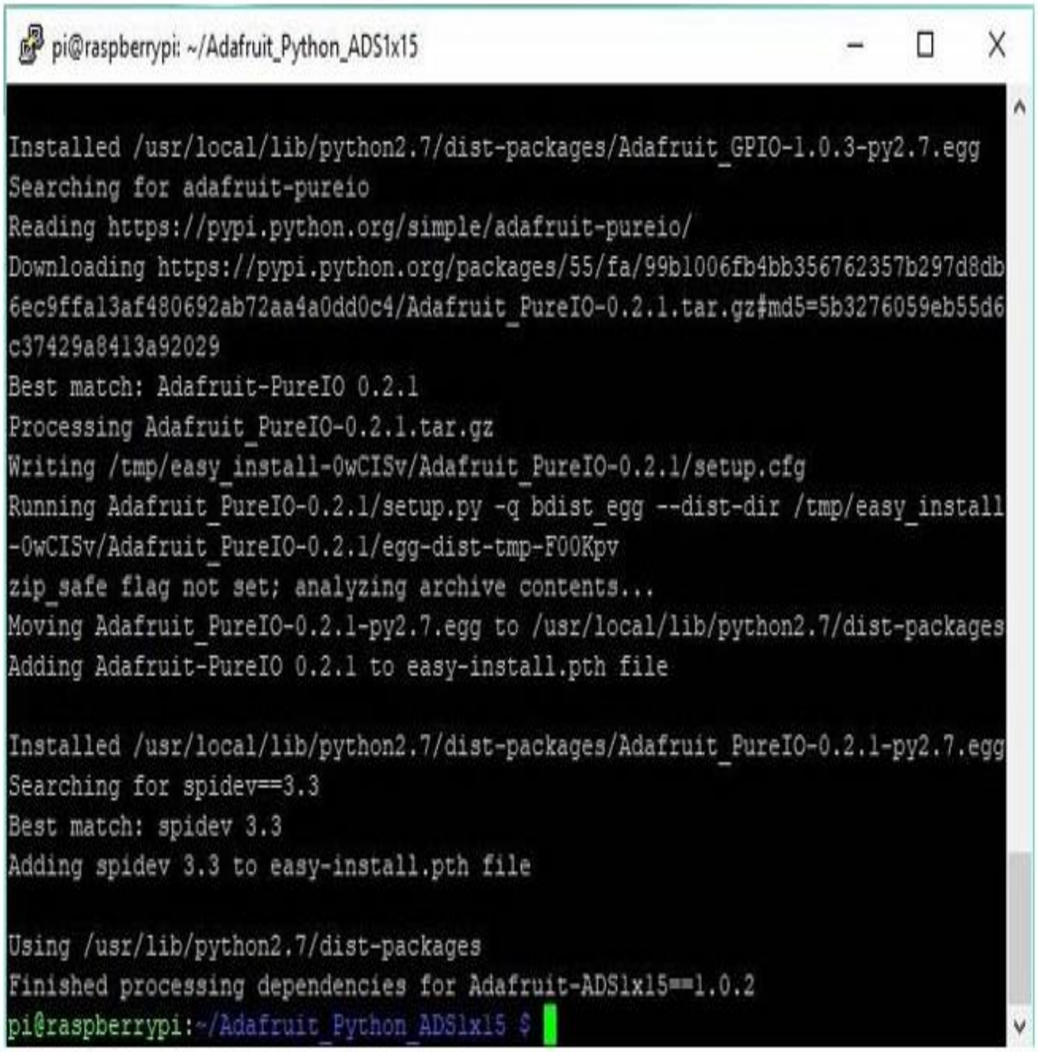
git clone https://github.com/adafruit/Adafruit_Python_ADS1x15.git

Change into the cloned file's directory and run the setup file;

cd Adafruit_Python_ADS1x15

sudo python setup.py install

After installation, your screen should look like the image below.



Step 4: Test the library and 12C communication.

Before we proceed with the rest of the project, it is important to test the library and ensure the ADC can communicate with the raspberry pi over I2C. To do this we will use an example script that comes with the library.

While still in the Adafruit_Python_ADS1x15 folder, change directory to the examples directory by running;

```
cd examples
```

Next, run the sampletest.py example which displays the value of the four channels on the ADC in a tabular form.

Run the example using:

python simpletest.py

If the I2C module is enabled and connections good, you should see the data as shown in the image below.

```
pi@raspberrypi:~ $ cd Adafruit_Python_ADS1x15
pi@raspberrypi:~/Adafruit_Python_ADS1x15 $ cd examples
pi@raspberrypi:~/Adafruit_Python_ADS1x15/examples $ python simpletest.py
Reading ADS1x15 values, press Ctrl-C to quit...
|     0 |     1 |     2 |     3 |
-------------------------------------
|  4699 |  4584 |  4625 |  4665 |
|  4583 |  4587 |  4601 |  4614 |
|  4563 |  4604 |  4600 |  4612 |
|  4601 |  4630 |  4609 |  4585 |
|  4614 |  4606 |  4577 |  4636 |
|  4616 |  4580 |  4621 |  4630 |
|  4566 |  4630 |  4618 |  4631 |
|  4614 |  4619 |  4615 |  4620 |
|  4577 |  4622 |  4609 |  4625 |
|  4624 |  4615 |  4626 |  4648 |
|  4636 |  4660 |  4656 |  4607 |
|  4609 |  4616 |  4629 |  4651 |
```

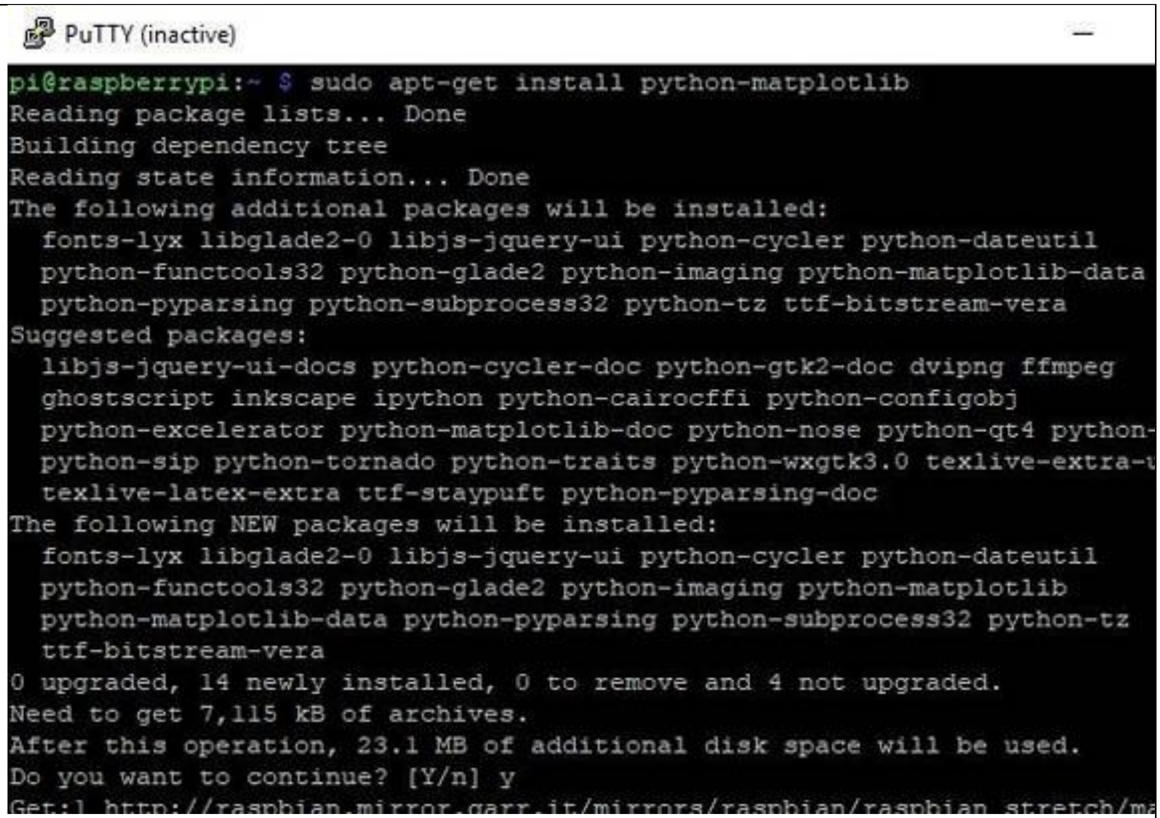
If an error occurs, check to ensure the ADC is well connected to the PI and I2C communication is enabled on the Pi.

Step 5: Install *Matplotlib*

To visualize the data we need to install the *matplotlib* module which is used to plot all kind of graphs in python. This can be done by running;

sudo apt-get install python-matplotlib

You should see an outcome like the image below.



Step6: Install the *Drawnow* python module

Lastly, we need to install the *drawnow* python module. This module helps us provide live updates to the data plot.

We will be installing *drawnow* via the python package installer; *pip*, so we need to ensure it is installed. This can be done by running;

sudo apt-get install python-pip12

We can then use pip to install the *drawnow* package by running:

sudo pip install drawnow

You should get an outcome like the image below after running it.

```
pi@raspberrypi:~ $ sudo pip install drawnow
Collecting drawnow
  Downloading drawnow-0.71.3.tar.gz
Requirement already satisfied: matplotlib>=1.5 in /usr/lib/python2.7/dist-pac
es (from drawnow)
Building wheels for collected packages: drawnow
  Running setup.py bdist_wheel for drawnow ... done
  Stored in directory: /root/.cache/pip/wheels/83/90/79/cc7449a69f925bfbee33f
582fa58febee3e2d0944ccb058
Successfully built drawnow
Installing collected packages: drawnow
Successfully installed drawnow-0.71.3
pi@raspberrypi:~ $
```

With all the dependencies installed, we are now ready to write the code.

Python Code for Raspberry Pi Oscilloscope:

The python code for this **Pi Oscilloscope** is fairly simple especially if you are familiar with the python *matplotlib* module. Before showing us the whole code, I will try to break it into part and explain what each part of the code is doing so you can have enough knowledge to extend the code to do more stuffs.

At this stage it is important to switch to a monitor or use the VNC viewer, anything through which you can see your Raspberry Pi's desktop, as the graph being plotted won't show on the terminal.

With the monitor as the interface **open a new python file**. You can call it any name you want, but I will call it scope.py.

sudo nano scope.py

With the file created, the first thing we do is import the modules we will be using;

import time

import matplotlib.pyplot as plt

from drawnow import *

import Adafruit_ADS1x15

Next, we **create an instance of the ADS1x15 library** specifying the ADS1115 ADC

adc = Adafruit_ADS1x15.ADS1115()

Next, we set the gain of the ADC. There are different ranges of gain and should be chosen based on the voltage you are expecting at the input of the ADC. For this tutorial, we are estimating a 0 – 4.09v so we will be using a gain of 1. For more info on gain you can check the ADS1015/ADS1115 datasheet.

```
GAIN = 1
```

Next, we need to create the array variables that will be used to store the data to be plotted and another one to serve as count.

```
Val = [ ]
cnt = 0
```

Next, we make know our intentions of making the plot interactive known so as to **enable us plot the data live**.

```
plt.ion()
```

Next, we start continuous ADC conversion **specifying the ADC channel**, in this case, channel 0 and we also specify the gain.

It should be noted that all the four ADC channels on the ADS1115 can be read at the same time, but 1 channel is enough for this demonstration.

```
adc.start_adc(0, gain=GAIN)
```

Next we create a function *def makeFig*, to **create and set the attributes of the graph** which will hold our live plot. We first of all set the limits of the y-axis using *ylim*, after which we input the title of the plot, and the label name before we specify the data that will be plotted and its plot style and color using *plt.plot()*. We can also state the channel (as channel 0 was stated) so we can identify each signal when the four channels of the ADC are being used. *plt.legend* is used to specify where we want the information about that signal(e.g Channel 0) displayed on the figure.

```
plt.ylim(-5000,5000)
plt.title('Osciloscope')
plt.grid(True)
plt.ylabel('ADC outputs')
```

```
plt.plot(val, 'ro-', label='lux')
plt.legend(loc='lower right')
```

Next we write the *while* loop which will be used constantly read data from the ADC and update the plot accordingly.

The first thing we do is **read the ADC conversion value**

```
value = adc.get_last_result()
```

Next we print the value on the terminal just to give us another way of confirming the plotted data. We wait a few seconds after printing then we append the data to the list (val) created to store the data for that channel.

```
print('Channel 0: {0}'.format(value))
time.sleep(0.5)
val.append(int(value))
```

We then call *drawnow* to update the plot.

```
drawnow(makeFig)
```

To ensure the latest data is what is available on the plot, we delete the data at index 0 after every 50 data counts.

```
cnt = cnt+1
if(cnt>50):
val.pop(0)
```

That's all!

The **complete Python code** is given at the end of this tutorial.

Raspberry Pi Oscilloscope in Action:

Copy the complete python code and paste in the python file we created earlier, remember we will need a monitor to view the plot so all of this should be done by either VNC or with a connected monitor or screen.

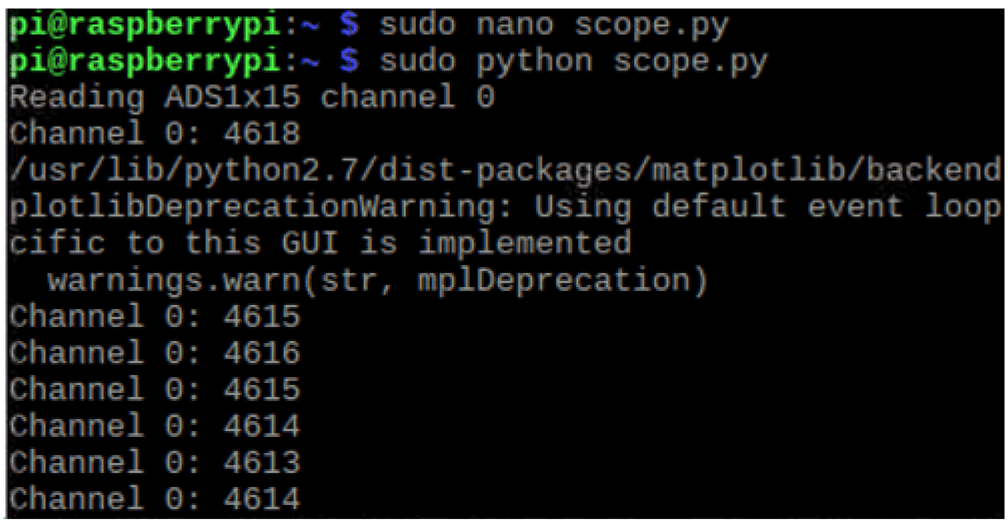
Save the code and run using;

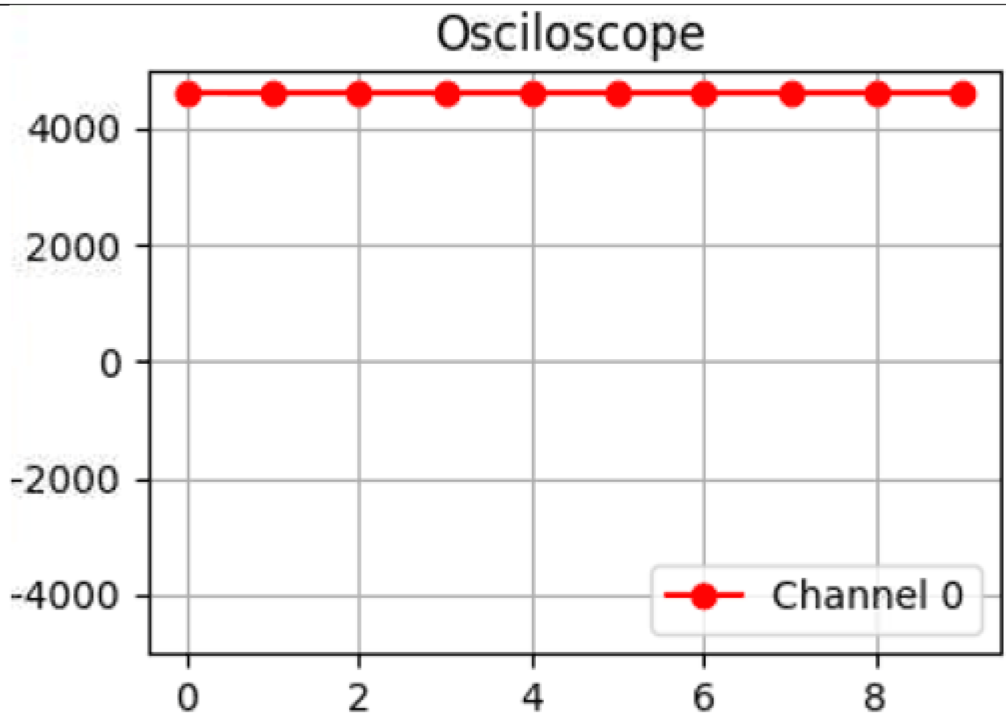
```
sudo python scope.py
```

If you used a different name other than scope.py, don't forget to change this to match.

After a few minutes, you should see the ADC data being printed on the terminal. Occasionally you may get a warning from *matplotlib* (as shown in the image below) which should be suppressed but it doesn't affect the data being displayed or the plot in anyway. To suppress the warning however, the following lines of code can be added after the import lines in our code.

```
Import warnings

import matplotlib.cbook

warnings.filterwarnings("ignore", category=matplotlib.cbook.mplDeprecation)
```





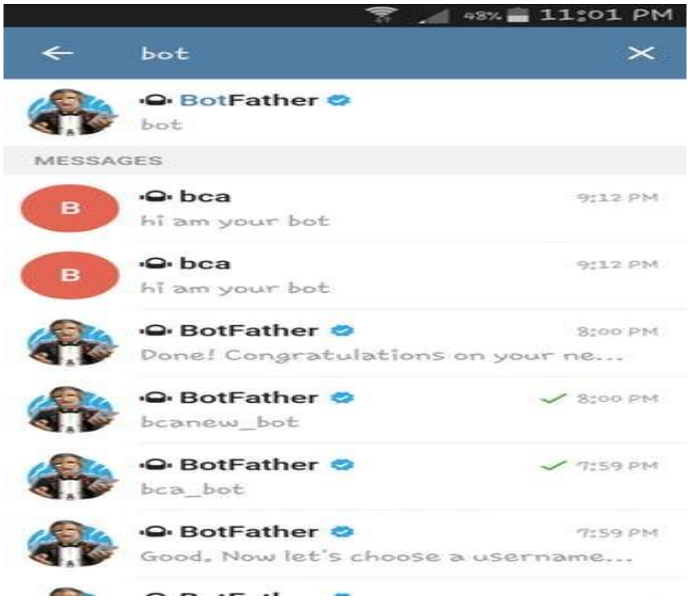
Code:

```
import time
import matplotlib.pyplot as plt
#import numpy
from drawnow import *
# Import the ADS1x15 module.
import Adafruit_ADS1x15
# Create an ADS1115 ADC (16-bit) instance.
adc = Adafruit_ADS1x15.ADS1115()

GAIN = 1
val = [ ]
cnt = 0
plt.ion()
# Start continuous ADC conversions on channel 0 using the previous gain value.
adc.start_adc(0, gain=GAIN)
print('Reading ADS1x15 channel 0')
#create the figure function
def makeFig():
    plt.ylim(-5000,5000)
    plt.title('Osciloscope')
    plt.grid(True)
    plt.ylabel('ADC outputs')
    plt.plot(val, 'ro-', label='Channel 0')
    plt.legend(loc='lower right')
```

```
while (True):
    # Read the last ADC conversion value and print it out.
    value = adc.get_last_result()
    print('Channel 0: {0}'.format(value))
    # Sleep for half a second.
    time.sleep(0.5)
    val.append(int(value))
    drawnow(makeFig)
    plt.pause(.000001)
    cnt = cnt+1
    if(cnt>50):
        val.pop(0)
```

4	**Controlling Raspberry Pi with Telegram.**

Step 1: Open Telegram app in your system or mobile
Open Telegram app in your system or mobile\
1.2 Start "BotFather"

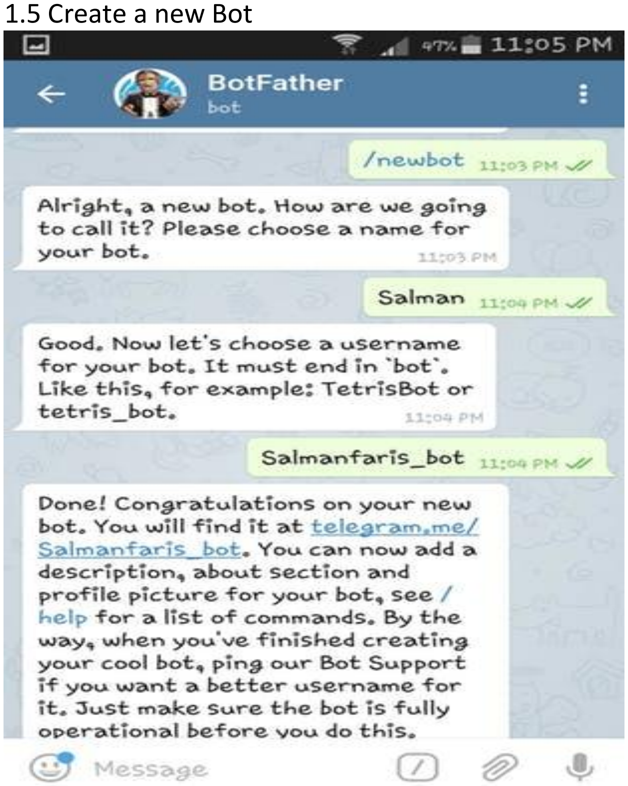


1.3 Open "BotFather"

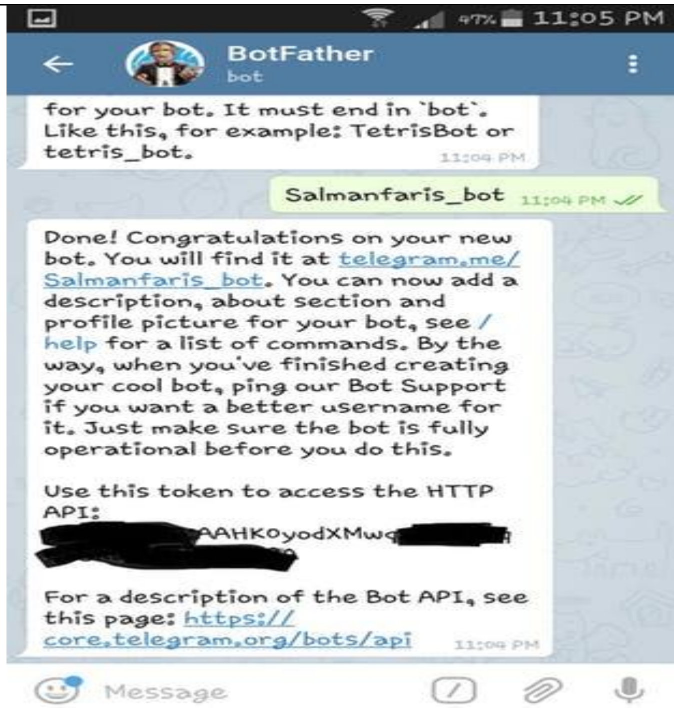
1.4 Start "BotFather"

/start

1.5 Create a new Bot



1.6 Obtain access token



3.3 Install "Python Package Index"

sudo apt-get install python-pip

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Jan  4 16:25:26 2017 from 192.168.100.4
pi@raspberrypi:~ $ sudo apt-get install python-pip
```

Note: Make sure Pi has internet access

```
Last login: Wed Jan  4 16:25:26 2017 from 192.168.100.4
pi@raspberrypi:~ $ sudo apt-get install python-pip
Reading package lists... Done
Building dependency tree
Reading state information... Done
python-pip is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 81 not upgraded.
pi@raspberrypi:~ $
```

3.4 Install "telepot"

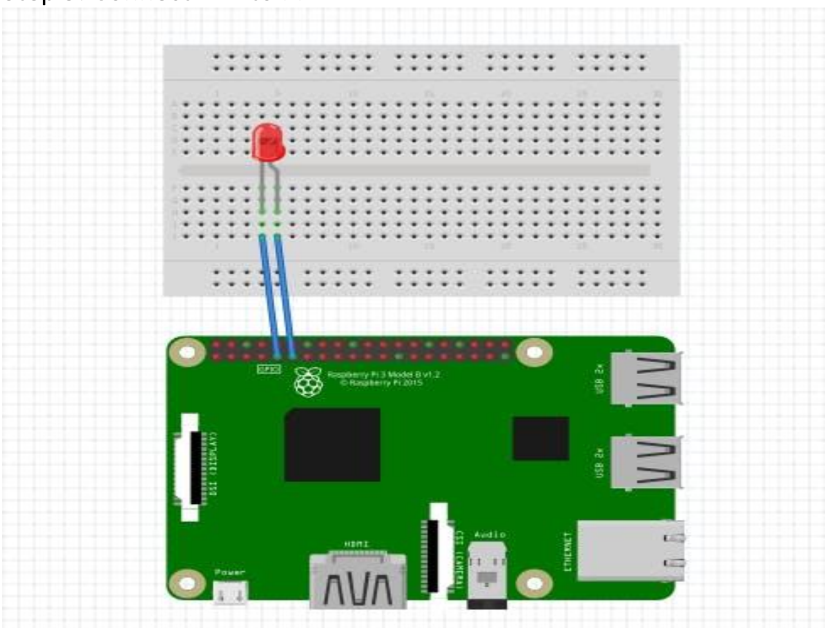
sudo pip install telepot

```
pi@raspberrypi:~ $ sudo pip install telepot
```

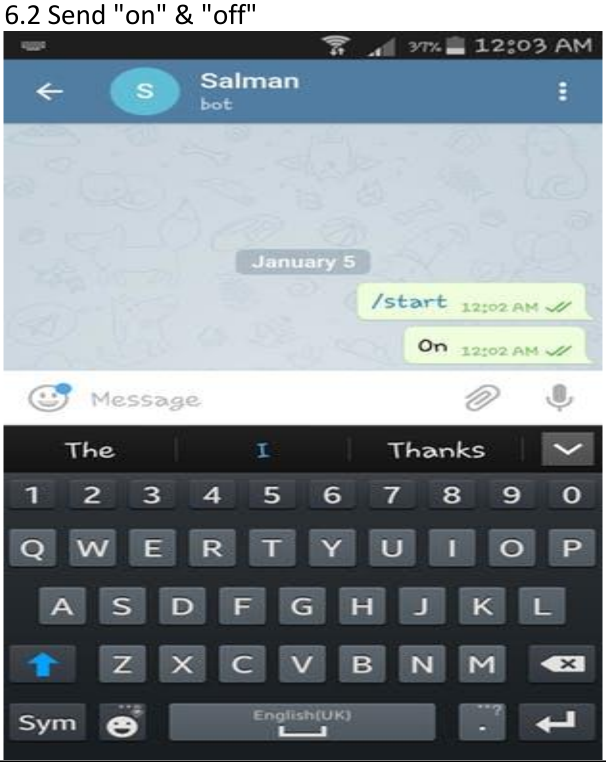
Step 4: Run the Python Code

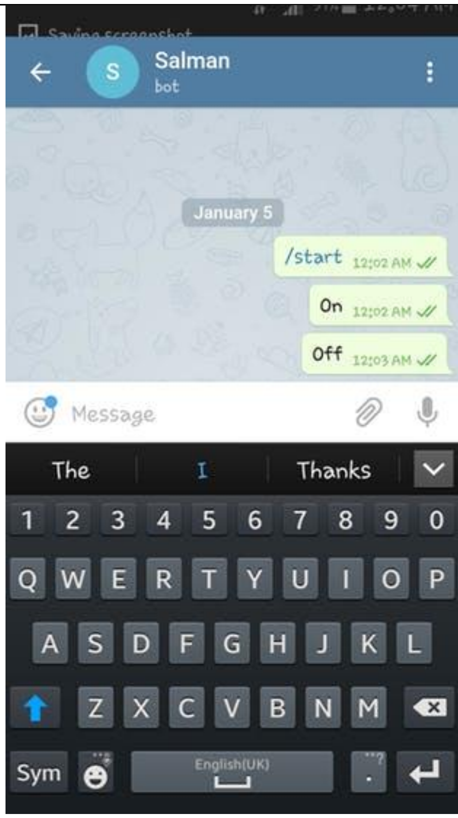
4.1 Clone the git

git clone https://github.com/salmanfarisvp/TelegramBot.git

	4.2 Paste your Bot Token here bot = telepot.Bot('Bot Token') Note: 1.6 for more details 4.3 Run the Code python telegrambot.py All set, now time to connect the Pi and LED. Step 5: Connect LED to Pi  Step 6: Send Command 6.1 Start our Bot 

6.2 Send "on" & "off"





Look at your Pi, you can see the LED on and off when you send "on" and "off" to our bot.
Code:

```
import sys
import time
import random
import datetime
import telepot
import RPi.GPIO as GPIO

#LED
def on(pin):
	GPIO.output(pin,GPIO.HIGH)
	return
def off(pin):
	GPIO.output(pin,GPIO.LOW)
	return
# to use Raspberry Pi board pin numbers
GPIO.setmode(GPIO.BOARD)
# set up GPIO output channel
GPIO.setup(11, GPIO.OUT)
```

```
def handle(msg):
    chat_id = msg['chat']['id']
    command = msg['text']

    print 'Got command: %s' % command

    if command == 'on':
       bot.sendMessage(chat_id, on(11))
    elif command =='off':
       bot.sendMessage(chat_id, off(11))

bot = telepot.Bot('Bot Token')
bot.message_loop(handle)
print 'I am listening...'

while 1:
     time.sleep(10)
```

5 Setting up Wireless Access Point using Raspberry Pi

Required Components:

The following components will be needed to set up a raspberry pi as a wireless access point:

1. Raspberry Pi 2
2. 8GB SD card
3. WiFi USB dongle
4. Ethernet cable
5. Power supply for the Pi.
6. Monitor (optional)
7. Keyboard (optional)
8. Mouse (optional)

Steps for Setting up Raspberry Pi as Wireless Access Point:

Step 1: Update the Pi

As usual, we update the raspberry pi to ensure we have the latest version of everything. This is done using;

```
sudo apt-get update
```

followed by;

```
sudo apt-get upgrade
```

With the update done, reboot your pi to effect changes.

Step 2: Install “*dnsmasq*” and “*hostapd*”

Next, we install the software that makes it possible to setup the pi as a wireless access point and also the software that helps assign network address to devices that connect to the AP. We do this by running;

```
sudo apt-get install dnsmasq
```

followed by;

```
sudo apt-get install hostapd
```

or you could combine it by running;

```
sudo apt-get install dnsmasq hostapd
```

Step 3: Stop the software from Running

Since we don’t have the software configured yet there is no point running it, so we disable them from running in the underground. To do this we run the following commands to stop the *systemd* operation.

```
sudo systemctl stop dnsmasq

sudo systemctl stop hostapd
```

Step 4: Configure a Static IP address for the wireless Port

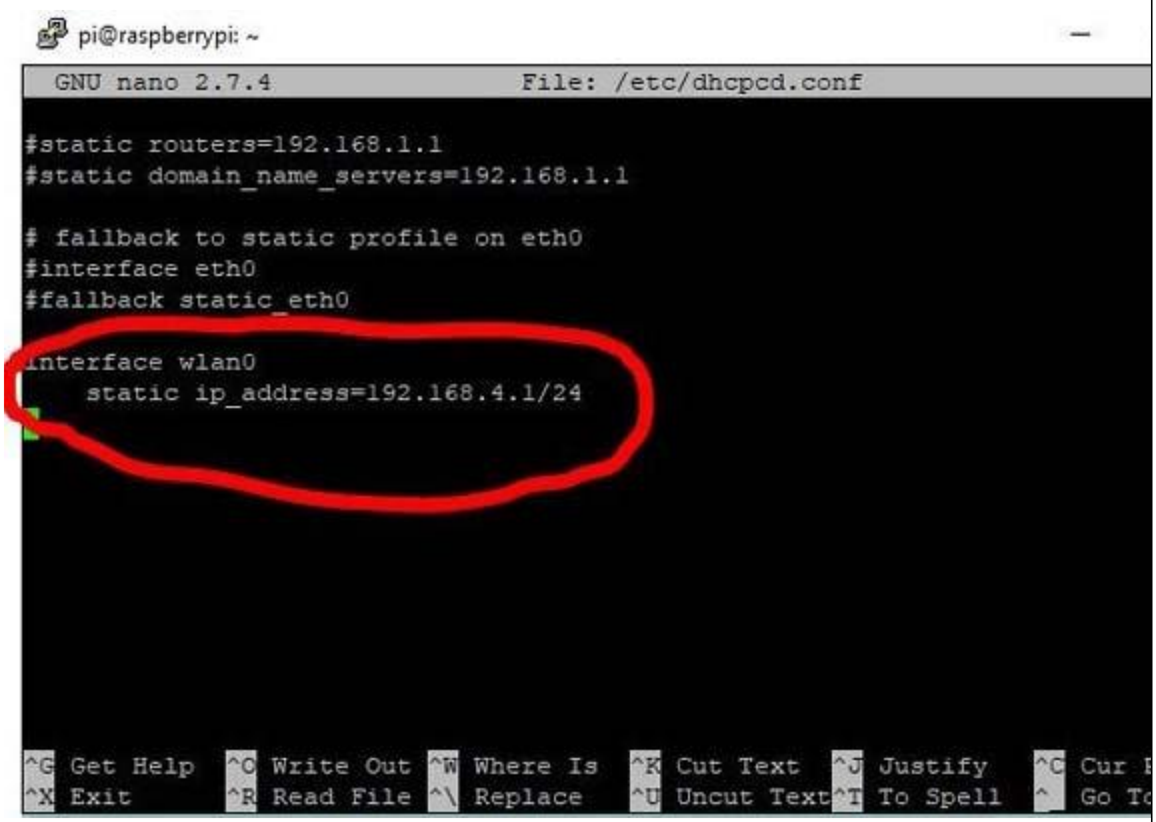
Confirm the *wlan* port on which the wireless device being used is connected. For my Pi, the wireless is on wlan0. **Setting up the Raspberry Pi to act as a server** requires us to assign a static IP address to the wireless port. This can be done by editing the *dhcpcd* config file. To edit the configuration file, run;

```
sudo nano /etc/dhcpcd.conf
```

Scroll to the bottom of the config file and add the following lines.

```
interface wlan0

static ip_address=192.168.1.200/24 #machine ip address
```

After adding the lines, the config file should look like the image below.



Note: This IP address can be changed to suit your preferred configuration.

Save the file and exit using; ctrl+x followed by Y

Restart the *dhcpcd* service to effect the changes made to the configuration using;

```
sudo service dhcpcd restart
```

Step 5: Configure the *dhcpcd* server

With a static IP address now configured for the Raspberry Pi wlan, the next thing is for us to configure the *dhcpcd* server and provide it with the **range of IP addresses to be**

assigned to devices that connect to the wireless access point. To do this, we need to edit the configuration file of the *dnsmasq* software but the config file of the software contains way too much info and a lot could go wrong If not properly edited, so instead of editing, we will be creating a new config file with just the amount of information that is needed to make the wireless access point fully functional.

Before creating the new config file, we keep the old on safe by moving and renaming it.

```
sudo mv /etc/dnsmasq.conf  /etc/dnsmasq.conf.old
```

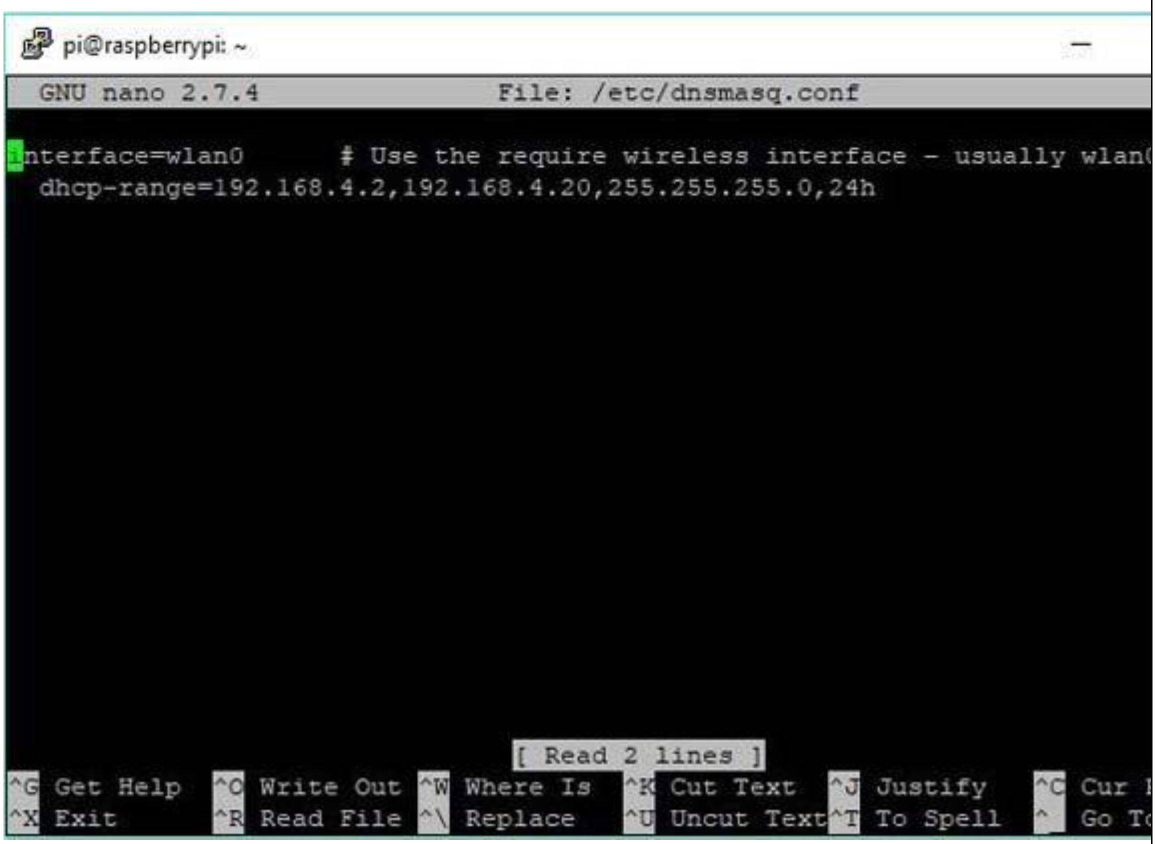
Then launch the editor to create a new configuration file;

```
sudo nano /etc/dnsmasq.conf
```

with the editor launched, copy the lines below and paste in or type directly into it.

```
interface = wlan0  #indicate the communication interface which is usually wlan0 for wire less
dhcp-range = 192.168.1.201, 192.168.1.220, 255.255.255.0,24h #start addr(other than ma chine ip assigned above), end addr, subnet mask, mask
```

the content of the file should look like the image below.



Save the file and exit. The content of this config file is just to specify the range of IP address that can be assigned to devices connected to the wireless access point.

With this done, we will be able to give an identity to devices on our network.

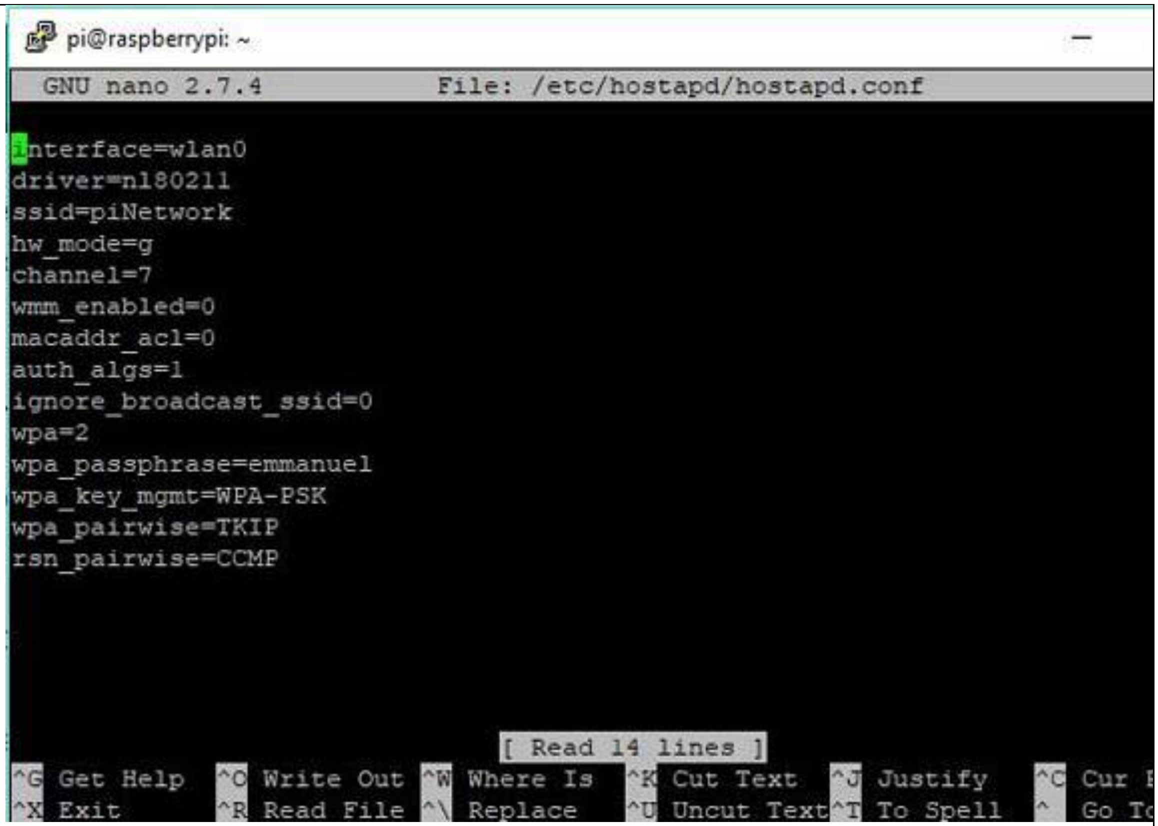
The next set of steps will help us configure the access point host software, setup the ssid, select the encrytpion etc.

Step 6: Configure *hostapd* for SSID and Password

We need to edit the *hostapd* config file(run ***sudo nano /etc/hostapd/hostapd.conf***) to add the various parameters for the wireless network being **setup including the ssid and password**. Its should be noted that the password (passphrase) should be between 8 and 64 characters. Anything lesser won't work.

```
interface=wlan0
driver=nl80211
ssid=piNetwork
hw_mode=g
channel=7
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=mumbai123 # use a very secure password and not this
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

The content of the file should look like the image below.



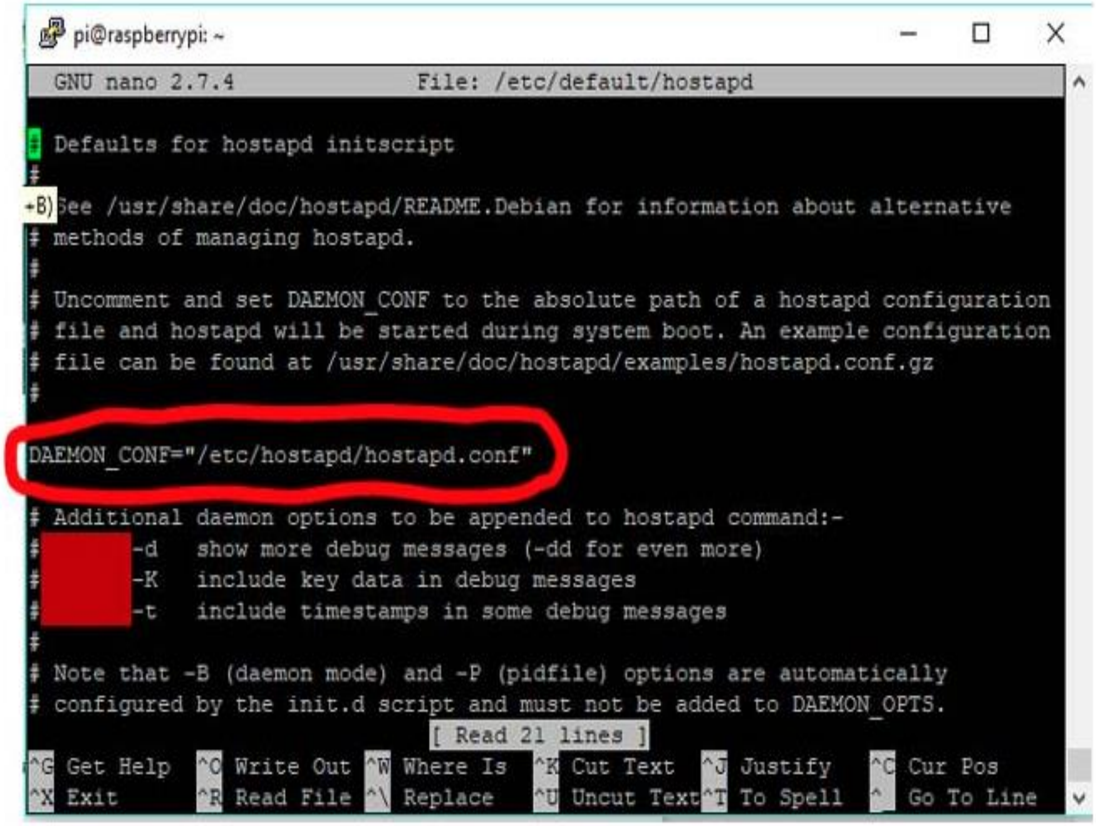
Feel free to change the ssid and password to suit your needs and desire.

Save the config file and exit.

After the config file has been saved, we need to point the hostapd software to where the config file has been saved. To do this, run;

```
sudo nano /etc/default/hostapd
```

find the line with *daemon_conf* commented out as shown in the image below.



Uncomment the DAEMON_CONF line and add the line below in between the quotes in front of the "equal to" sign.

```
/etc/hostapd/hostapd.conf
```

Step 7: Fire it up

Since we disabled the two software initially, to allow us configure them properly, we need to restart the system after configuration to effect the changes.

Use;

```
sudo systemctl start hostapd
sudo systemctl start dnsmasq
```

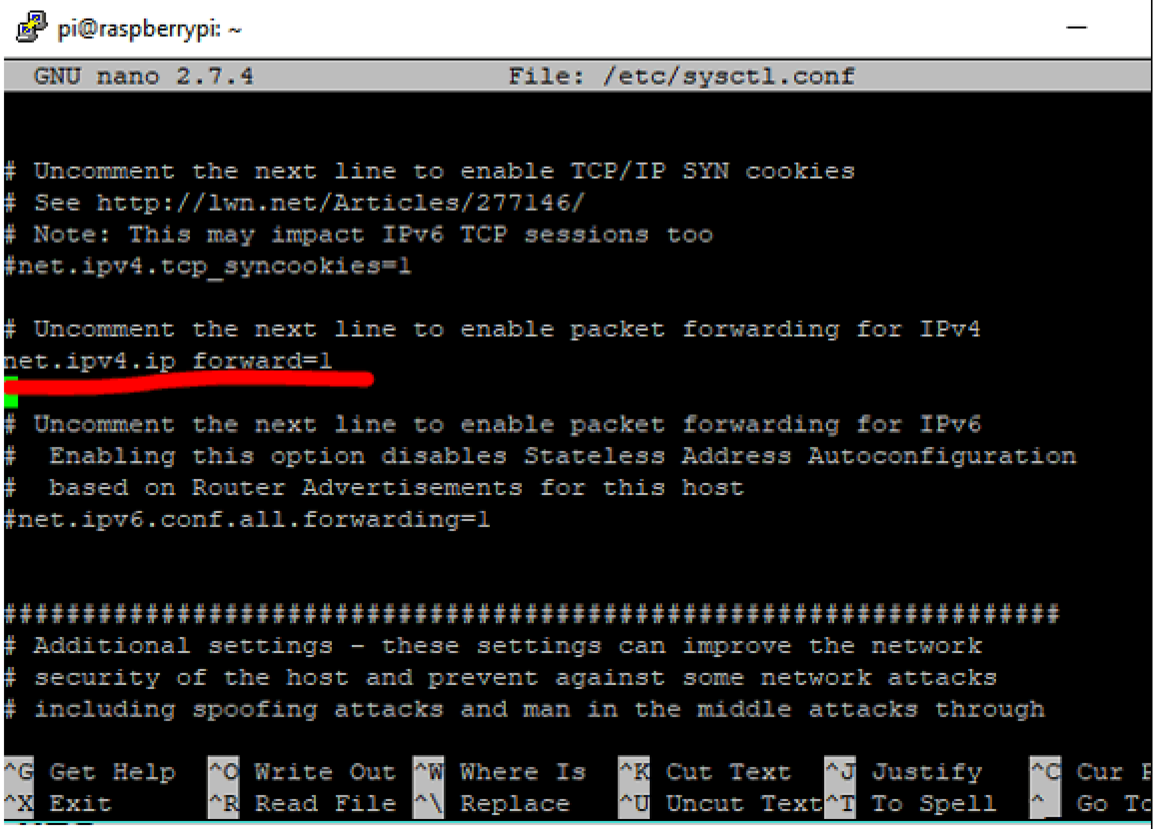
Step 8: Routing and masquerade for outbound traffic

We need to add routing and masquerade for outbound traffic.

To do this, we need to edit the config file of the *systemctl* by running:

sudo nano /etc/sysctl.conf

Uncomment this line ***net.ipv4.ip_forward=1***(highlighted in the image below)



Save the config file and exit using ctrl+x followed by y.

Next we move to masquerading the outbound traffic. This can be done by making some changes to the iptable rule. To do this, run the following commands:

sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

then save the Iptables rule using:

sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"

Step 9: Create Wireless Access Point on startup:

For most wireless access point application, it is often desired that the access point comes up as soon as the system boots. To implement this on the raspberry pi, one of the easiest ways is to add instructions to run the software in the *rc.local* file so we put commands to install the iptable rules on boot in the rc.local file.

To edit the rc.local file, run:

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

and add the following lines at the bottom of the system, just before the exit 0 statement

```
iptables-restore < /etc/iptables.ipv4.nat
```

Step 9: Reboot! and Use

At this stage, we need to reboot the system to effect all the changes and test the wireless access point starting up on boot with the iptables rule updated.

Reboot the system using:

```
sudo reboot
```

As soon as the system comes back on, you should be able to access the wireless access point using any Wi-Fi enabled device and the password used during the setup.

Accessing the Internet from the Raspberry Pi's Wi-Fi Hotspot

To implement this, we need to put a "bridge" in between the wireless device and the Ethernet device on the Raspberry Pi (the wireless access point) to pass all traffic between the two interfaces. To set this up, we will use the *bridge-utils* software. Install *hostapd* and *bridge-utils*. While we have installed*hostapd* before, run the installation again to clear all doubts.

```
sudo apt-get install hostapd bridge-utils
```

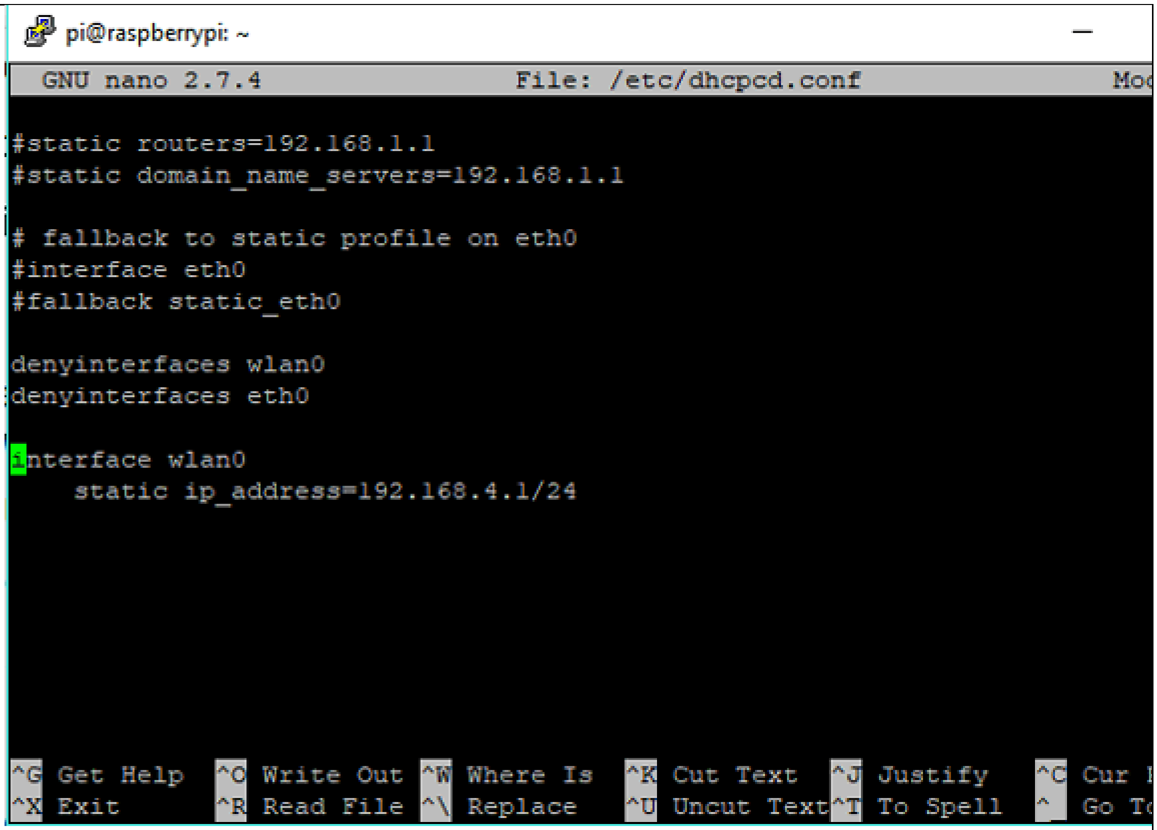
Next, we stop hostapd so as to configure the software.

```
sudo systemctl stop hostapd
```

When a bridge is created, a higher level construct is created over the two ports being bridged and the bridge thus becomes the network device. To prevent conflicts, we need to stop the allocation of IP addresses by the DHCP client running on the Raspberry Pi to the eth0 and wlan0 ports. This will be done by editing the config file of the dhcpcd client to include ***denyinterfaces wlan0*** and ***denyinterfaces eth0*** as shown in the image below.

The file can be edited by running the command;

```
sudo nano /etc/dhcpcd.conf
```



Note: *From this point on, ensure you don't disconnect the Ethernet cable from your PC if you are running in headless mode as you may not be able to connect via SSH again since we have disabled the Ethernet port. If working with a monitor, you have nothing to fear.*

Next, we create a new bridge called br0

sudo brctl addbr br0

Next, we connect the ethernet port (eth0) to the bridge (br0) using;

sudo brctl addif br0 eth0

(Note: if eth0 doesn't exists use ifconfig command to list all Ethernet adapters and use th e name from list)

Next, we edit the interfaces file using ***sudo nano /etc/network/interfaces*** so various devices can work with the bridge. Edit the interfaces file to include the information below;

#Bridge setup

auto br0

```
iface br0 inet manual

bridge_ports eth0 wlan0
```

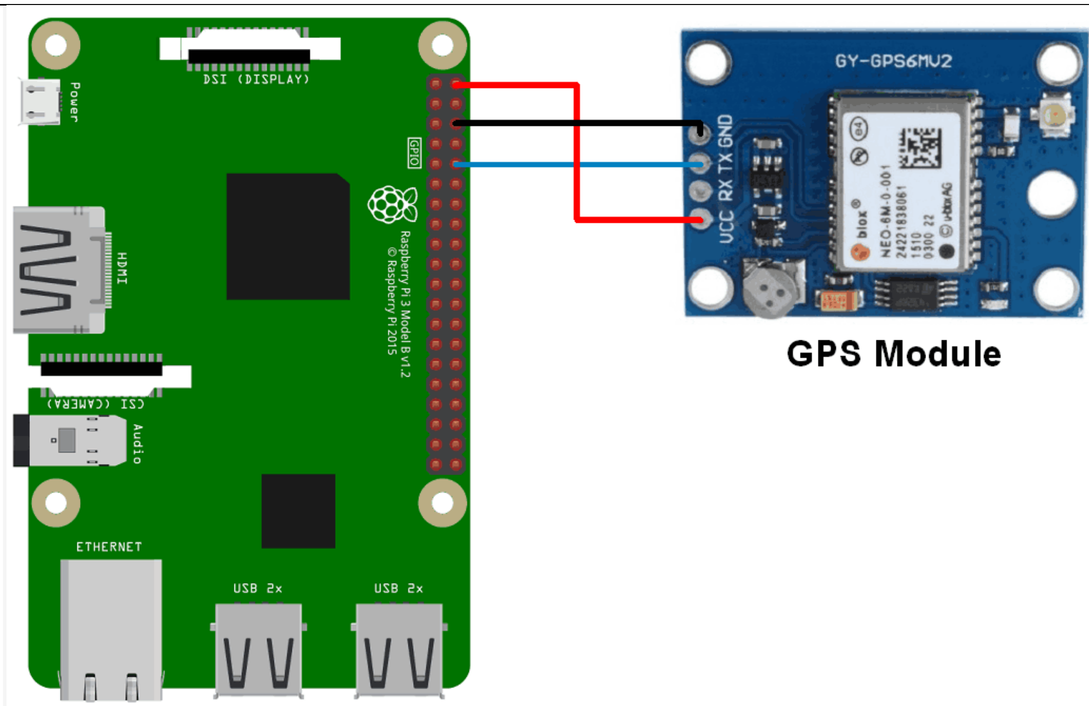
Lastly we edit the hostapd.conf file to include the bridge configuration. This can be done by running the command: ***sudo nano /etc/hostapd/hostapd.conf*** and editing the file to contain the information below. Note the bridge was added below the wlan0 interface and the driver line was commented out.

```
interface=wlan0

bridge=br0

ssid=piNetwork

hw_mode=g

channel=7

wmm_enabled=0

macaddr_acl=0

auth_algs=1

ignore_broadcast_ssid=0

wpa=2

wpa_passphrase=mcctest1

wpa_key_mgmt=WPA-PSK

wpa_pairwise=TKIP

rsn_pairwise=CCMP
```

With this done, save the config file and exit.

To effect the changes made to the Raspberry Pi, **reboot** the system. Once it comes back up, you should now be **able to access the internet by connecting to the Wireless access point created by the Raspberry Pi**. This of course will only work if internet access is available to the pi via the Ethernet port.

6 **Raspberry Pi GPS Module Interfacing.**




```
sudo nano /boot/config.txt
################
dtparam=spi=on
dtoverlay=pi3-disable-bt
core_freq=250
enable_uart=1
force_turbo=1
##############
sudo systemctl stop serial-getty@ttyS0.service
sudo systemctl disable serial-getty@ttyS0.service

sudo systemctl enable serial-getty@ttyAMA0.service

sudo apt-get install minicom
sudo pip install pynmea2

sudo cat /dev/ttyAMA0
```

code:

```
import time
import serial
import string
```

```
import pynmea2
import RPi.GPIO as gpio
gpio.setmode(gpio.BCM)
port = "/dev/ttyAMA0" # the serial port to which the pi is connected.

#create a serial object
ser = serial.Serial(port, baudrate = 9600, timeout = 0.5)
while 1:
    try:
        data = ser.readline()
#           print data
    except:
            print("loading")
#wait for the serial port to churn out data

    if data[0:6] == '$GPGGA':
            msg = pynmea2.parse(data)
            print msg
            time.sleep(2)
```

7 **Interfacing Raspberry Pi with 16x2 LCD using I2C module.**

Step 1 – Connect LCD Screen to the Pi

The I2c module can be powered with either 5V or 3.3V but the screen works best if it provided with 5V. However the Pi’s GPIO pins aren’t 5V tolerant so the I2C signals need to be level shifted. To do this I used an I2C level shifter.

This requires a high level voltage (5V) and a low level voltage (3.3V) which the device uses as a reference. The HV pins can be connected to the screen and two of the LV pins to the Pi’s I2C interface.

Level Shifter	Pi	I2C Backpack
LV	3.3V	–
LV1	SDA	–
LV2	SCL	–
GND	GND	GND
HV	5V	VCC
HV1		SDA
HV2		SCL

While experimenting I found that it worked fine without the level shifting but I couldn’t be certain this wasn’t going to damage the Pi at some point. So it’s probably best to play it safe!

Step 2 – Download the Example Python Script

The example script will allow you to send text to the screen via I2C. It is very similar to my scripts for the normal 16×2 screen. To download the script directly to your Pi you can use :

```
wget https://bitbucket.org/MattHawkinsUK/rpispy-misc/raw/master/python/lcd_i2c.py
```

Step 3 – Enable the I2C Interface

In order to use I2C devices you must enable the interface on your Raspberry Pi. This can be done by following my "Enabling The I2C Interface On The Raspberry Pi" tutorial. By default the I2C backpack will show up on address 0x27.

Step 4 – Run LCD Script

The script can be run using the following command :

```
sudo python lcd_i2c.py
```

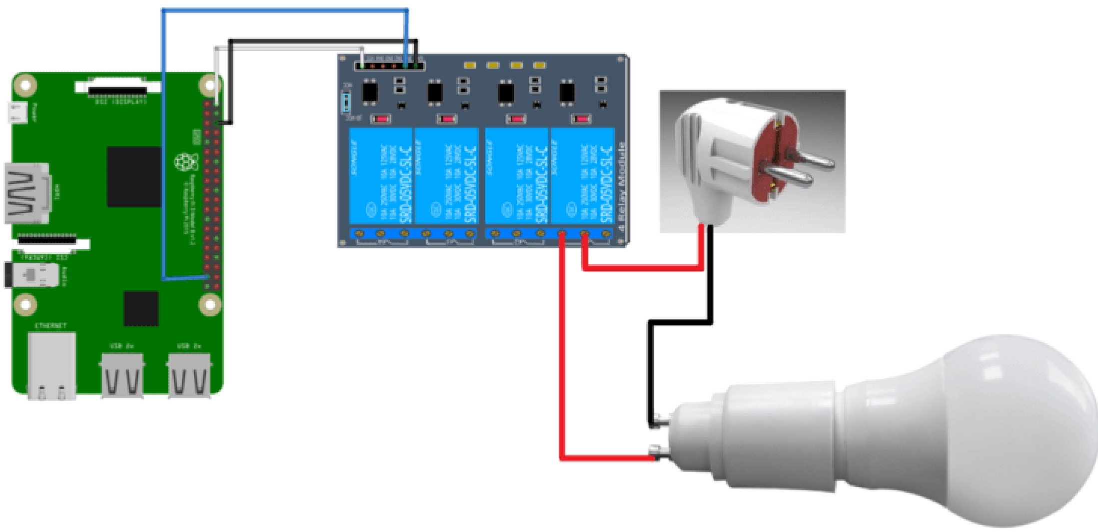
Code:

```
import smbus
import time

# Define some device parameters
I2C_ADDR  = 0x27 # I2C device address
LCD_WIDTH = 16   # Maximum characters per line

# Define some device constants
LCD_CHR = 1 # Mode - Sending data
LCD_CMD = 0 # Mode - Sending command

LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line
LCD_LINE_3 = 0x94 # LCD RAM address for the 3rd line
LCD_LINE_4 = 0xD4 # LCD RAM address for the 4th line

LCD_BACKLIGHT  = 0x08  # On
#LCD_BACKLIGHT = 0x00  # Off

ENABLE = 0b00000100 # Enable bit

# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005

#Open I2C interface
#bus = smbus.SMBus(0)  # Rev 1 Pi uses 0
bus = smbus.SMBus(1) # Rev 2 Pi uses 1

def lcd_init():
```

```
  # Initialise display
  lcd_byte(0x33,LCD_CMD) # 110011 Initialise
  lcd_byte(0x32,LCD_CMD) # 110010 Initialise
  lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction
  lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off
  lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  time.sleep(E_DELAY)

def lcd_byte(bits, mode):
  # Send byte to data pins
  # bits = the data
  # mode = 1 for data
  #        0 for command

  bits_high = mode | (bits & 0xF0) | LCD_BACKLIGHT
  bits_low = mode | ((bits<<4) & 0xF0) | LCD_BACKLIGHT

  # High bits
  bus.write_byte(I2C_ADDR, bits_high)
  lcd_toggle_enable(bits_high)

  # Low bits
  bus.write_byte(I2C_ADDR, bits_low)
  lcd_toggle_enable(bits_low)

def lcd_toggle_enable(bits):
  # Toggle enable
  time.sleep(E_DELAY)
  bus.write_byte(I2C_ADDR, (bits | ENABLE))
  time.sleep(E_PULSE)
  bus.write_byte(I2C_ADDR,(bits & ~ENABLE))
  time.sleep(E_DELAY)

def lcd_string(message,line):
  # Send string to display

  message = message.ljust(LCD_WIDTH," ")

  lcd_byte(line, LCD_CMD)

  for i in range(LCD_WIDTH):
    lcd_byte(ord(message[i]),LCD_CHR)

def main():
  # Main program block
```

```
  # Initialise display
  lcd_init()

  while True:

    # Send some test
    lcd_string("RPiSpy         <",LCD_LINE_1)
    lcd_string("I2C LCD        <",LCD_LINE_2)

    time.sleep(3)

    # Send some more text
    lcd_string(">         RPiSpy",LCD_LINE_1)
    lcd_string(">        I2C LCD",LCD_LINE_2)

    time.sleep(3)

if __name__ == '__main__':

  try:
    main()
  except KeyboardInterrupt:
    pass
  finally:
    lcd_byte(0x01, LCD_CMD)
```

8	**IoT based Web Controlled Home Automation using Raspberry Pi**



Code:

```
import RPi.GPIO as GPIO
from time import sleep
relay_pin = 26
GPIO.setmode(GPIO.BOARD)
GPIO.setup(relay_pin, GPIO.OUT)
GPIO.output(relay_pin, 1)

try:
    while True:
        GPIO.output(relay_pin, 0)
        sleep(5)
        GPIO.output(relay_pin, 1)
        sleep(5)
except KeyboardInterrupt:
    pass
GPIO.cleanup()
```

9	Interfacing Raspberry Pi with Pi Camera.

To capture image:

```
import picamera
from time import sleep

#create object for PiCamera class
camera = picamera.PiCamera()
#set resolution
camera.resolution = (1024, 768)
camera.brightness = 60
camera.start_preview()
#add text on image
camera.annotate_text = 'Hi Pi User'
sleep(5)
#store image
camera.capture('image1.jpeg')
camera.stop_preview()
```

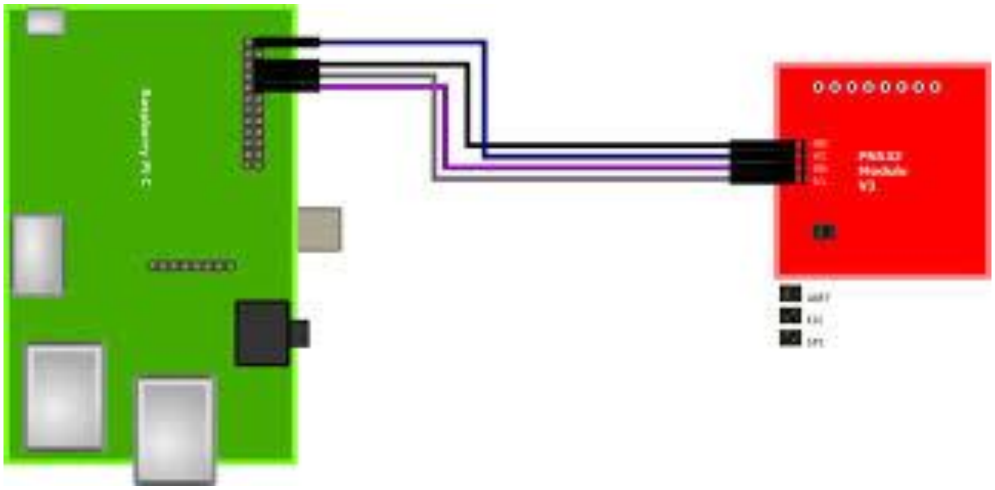
To capture video:

```
import picamera
from time import sleep
camera = picamera.PiCamera()
camera.resolution = (640, 480)
print()
#start recording using pi camera
camera.start_recording("/home/pi/demo.h264")
#wait for video to record
camera.wait_recording(20)
#stop recording
camera.stop_recording()
camera.close()
print("video recording stopped")
```

To Play the video:

```
Omxplayer demo.h264
```

10	**Interfacing Raspberry Pi with RFID.**



I2C Communication Instructions for Raspberry Pi

1. Open I2C of the Raspberry Pi :

```
sudo raspi-config
```

Select **5 Interfacing Options** -> **I2C** -> **yes**.

2. **Install some dependent packages**

```
sudo apt-get update
sudo apt-get install libusb-dev libpcsclite-dev i2c-tools
```

3. **Download and unzip the source code package of libnfc**

```
cd ~
wget http://dl.bintray.com/nfc-tools/sources/libnfc-1.7.1.tar.bz2
tar -xf libnfc-1.7.1.tar.bz2
```

4. **Compile and install**

```
cd libnfc-1.7.1
./configure --prefix=/usr --sysconfdir=/etc
make
sudo make install
```

5. **Write the configuration file for NFC communication**

```
cd /etc
sudo mkdir nfc
sudo nano /etc/nfc/libnfc.conf
```

Check the following details of the file *etc/nfc/libnfc.conf*:

```
# Allow device auto-detection (default: true)
# Note: if this auto-detection is disabled, user has to set manually a device
# configuration using file or environment variable
allow_autoscan = true

# Allow intrusive auto-detection (default: false)
# Warning: intrusive auto-detection can seriously disturb other devices
# This option is not recommended, user should prefer to add manually his device.
allow_intrusive_scan = false

# Set log level (default: error)
# Valid log levels are (in order of verbosity): 0 (none), 1 (error), 2 (info), 3 (debug)
# Note: if you compiled with --enable-debug option, the default log level is "debug"
log_level = 1

# Manually set default device (no default)
# To set a default device, you must set both name and connstring for your device
# Note: if autoscan is enabled, default device will be the first device available in
device list.
#device.name = "_PN532_SPI"
#device.connstring = "pn532_spi:/dev/spidev0.0:500000"
```

device.name = "_PN532_I2c"
device.connstring = "pn532_i2c:/dev/i2c-1"

6. **Wiring**

Toggle the switch to the **I2C mode**

SEL 0	SEL 1
H	L

Connect the devices:

PN532	Raspberry
5V	5V 4
GND	GND 6
SDA	SDA0 3
SCL	SCL0 5

7. Run *i2cdetect –yes 1* to check whether the I2C device is recognized.

If yes, it means both the module and the wiring work well.
Then type in *nfc-list* to check the NFC module:

pi@raspberrypi: ~

```
pi@raspberrypi:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- 24 -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi:~ $ nfc-list
nfc-list uses libnfc 1.7.1
NFC device: pn532_i2c:/dev/i2c-1 opened
pi@raspberrypi:~ $
```

Run *nfc-poll* to scan the RFID tag and you can read information on the card:

```
pi@raspberrypi:~ $ nfc-list
nfc-list uses libnfc 1.7.1
NFC device: pn532_i2c:/dev/i2c-1 opened
pi@raspberrypi:~ $ nfc-poll
nfc-poll uses libnfc 1.7.1
NFC reader: pn532_i2c:/dev/i2c-1 opened
NFC device will poll during 30000 ms (20 pollings of 300 ms for 5 modulation
ISO/IEC 14443A (106 kbps) target:
    ATQA (SENS_RES): 00  04
       UID (NFCID1): f4  55  4e  b8
      SAK (SEL_RES): 08
nfc_initiator_target_is_present: Target Released
Waiting for card removing...done.
pi@raspberrypi:~ $
```

SPI Communication Instructions for Raspberry Pi

1. Open SPI of the Raspberry Pi:

sudo raspi-config

Select **9 Advanced Options** -> **SPI** -> **yes**.

2. Install some dependent packages

sudo apt-get update
sudo apt-get install libusb-dev libpcsclite-dev i2c-tools

3. Download and unzip the source code package of libnfc

cd ~
wget http://dl.bintray.com/nfc-tools/sources/libnfc-1.7.1.tar.bz2
tar -xf libnfc-1.7.1.tar.bz2

4. Compile and install

cd libnfc-1.7.1
./configure --prefix=/usr --sysconfdir=/etc
make
sudo make install

5. Write the configuration file for NFC communication

```
cd /etc

sudo mkdir nfc
sudo nano /etc/nfc/libnfc.conf
```

Check the following details of the file *etc/nfc/libnfc.conf*:

```
# Allow device auto-detection (default: true)
# Note: if this auto-detection is disabled, user has to set manually a device
# configuration using file or environment variable
allow_autoscan = true

# Allow intrusive auto-detection (default: false)
# Warning: intrusive auto-detection can seriously disturb other devices
# This option is not recommended, user should prefer to add manually his device.
allow_intrusive_scan = false

# Set log level (default: error)
# Valid log levels are (in order of verbosity): 0 (none), 1 (error), 2 (info), 3 (debug)
# Note: if you compiled with --enable-debug option, the default log level is "debug"
log_level = 1

# Manually set default device (no default)
# To set a default device, you must set both name and connstring for your device
# Note: if autoscan is enabled, default device will be the first device available in
device list.
device.name = "_PN532_SPI"
device.connstring = "pn532_spi:/dev/spidev0.0:500000"
#device.name = "_PN532_I2c"
#device.connstring = "pn532_i2c:/dev/i2c-1"
```

6. Wiring

Toggle the switch to the **SPI mode**

SEL0	SEL1
L	H

Connect the devices:

PN532	Raspberry
5V	5V
GND	GND
SCK	SCKL
MISO	MISO
MOSI	MOSI
NSS	CE0

7. Run *ls /dev/spidev0. to check whether the SPI is opened or not.**
If yes, it means both the module and the wiring work well.
Then type in *nfc-list* to check the NFC module:
/dev/spidev0.0 /dev/spidev0.1
If two devices are detected, it means the SPI is already opened.
Then type in *nfc-list* to check the NFC module:

```
pi@raspberrypi:~ $ nfc-list
nfc-list uses libnfc 1.7.1
NFC device: pn532_spi:/dev/spidev0.0 opened
pi@raspberrypi:~ $
```

For Raspberry Pi 3, you may be appear the following error

```
pi@raspberrypi:/etc $ sudo nano /etc/nfc/libnfc.conf
pi@raspberrypi:/etc $ nfc-list
nfc-list uses libnfc 1.7.1
error   libnfc.driver.pn532_spi Unable to wait for SPI data. (RX)
pn53x_check_communication: Timeout
error   libnfc.driver.pn532_spi Unable to wait for SPI data. (RX)
nfc-list: ERROR: Unable to open NFC device: pn532_spi:/dev/spidev0.0:500000
pi@raspberrypi:/etc $
```

You should modifiy the *libnfc.conf*

sudo nano /etc/nfc/libnfc.conf

then modify 500000 to 50000:

device.connstring = "pn532_spi:/dev/spidev0.0:50000"

Run *nfc-poll* to scan the RFID tag and you can read information on the card:

```
pi@raspberrypi:~ $ nfc-poll
nfc-poll uses libnfc 1.7.1
NFC reader: pn532_spi:/dev/spidev0.0 opened
NFC device will poll during 30000 ms (20 pollings of 300 ms for 5 modulations)
ISO/IEC 14443A (106 kbps) target:
    ATQA (SENS_RES): 00  04
       UID (NFCID1): f4  55  4e  b8
      SAK (SEL_RES): 08
nfc_initiator_target_is_present: Target Released
Waiting for card removing...done.
pi@raspberrypi:~ $
```

Code:

```
import subprocess
import time

def nfc_raw():
	lines=subprocess.check_output("/usr/bin/nfc-poll",
stderr=open('/dev/null','w'))
	return lines

def read_nfc():
	lines=nfc_raw()
	return lines

try:
	while True:
		myLines=read_nfc()
		buffer=[]
		for line in myLines.splitlines():
			line_content=line.split()
			if(not line_content[0] =='UID'):
				pass
			else:
				buffer.append(line_content)
		str=buffer[0]
		id_str=str[2]+str[3]+str[4]+str[5]
		print (id_str)

except KeyboardInterrupt:
    pass
```

11	**Installing Windows 10 IoT Core on Raspberry Pi.**

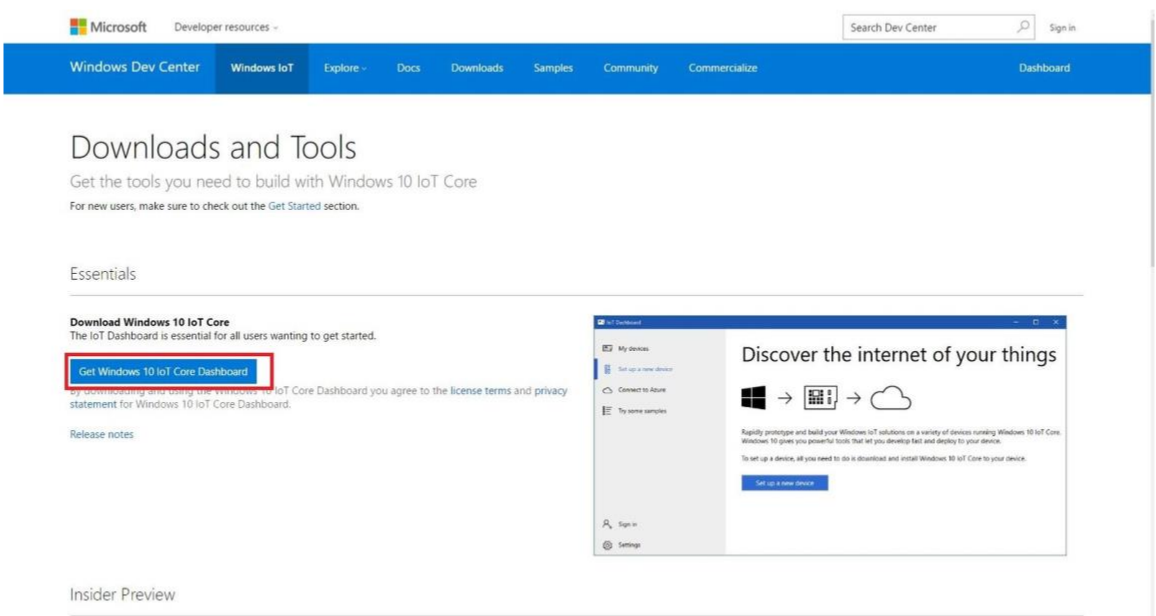
To get up and running you need a few bits and pieces:

1. Raspberry Pi 3.
2. 5V 2A microUSB power supply.
3. 8GB or larger Class 10 microSD card with full-size SD adapter.
4. HDMI cable.
5. Access to a PC.
6. USB WiFi adapter (older models of Raspberry Pi) or Ethernet cable.

At this point, the HDMI cable is only to plug the Raspberry Pi into a display so you can make sure your install worked. Some Raspberry Pi starter kits include everything you need, but the list above covers the power, display, and something to install Windows 10 IoT Core on.

Go to the Windows 10 developer center.

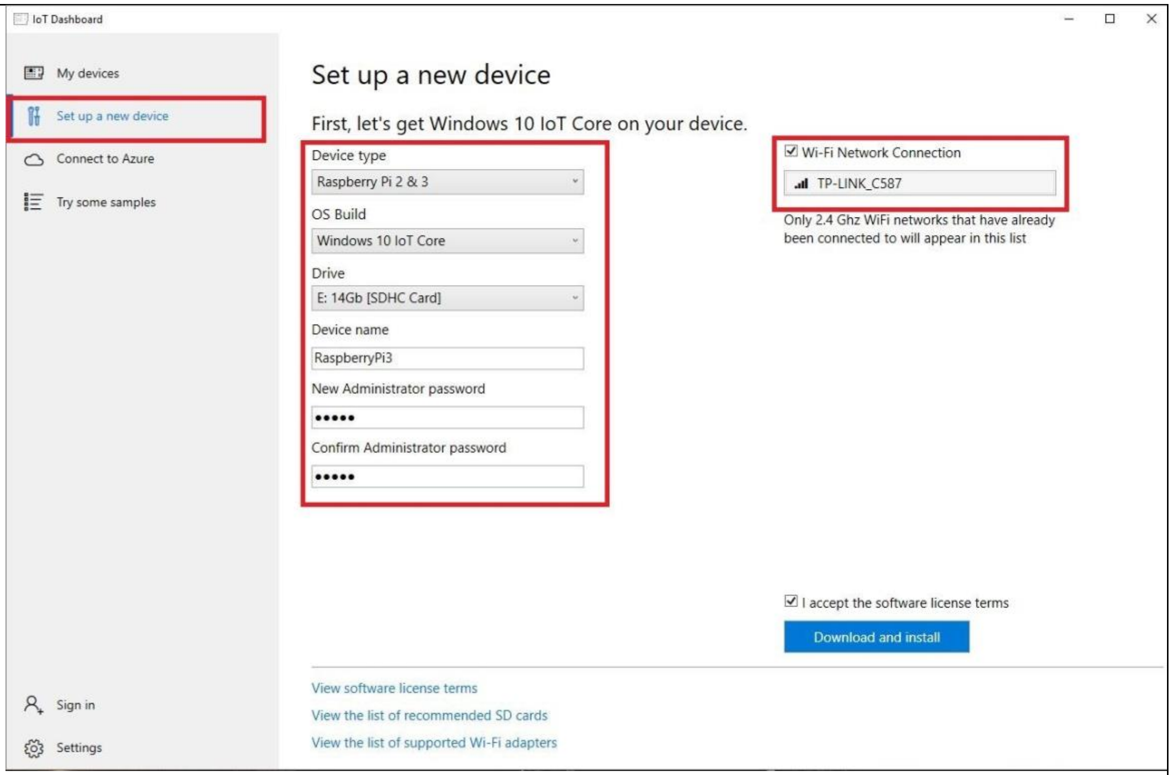
Click **Get Windows 10 IoT Core Dashboard** to download the necessary application.



Install the application and open it.

Select **set up a new device** from the sidebar.

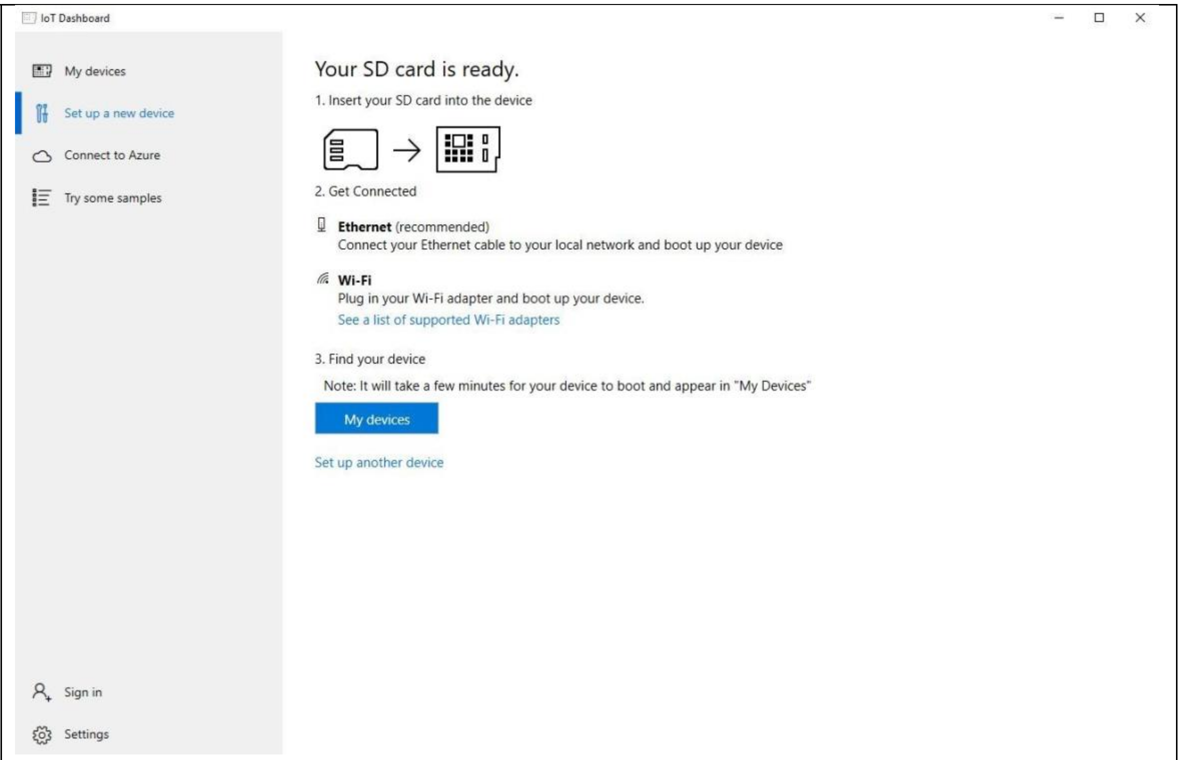
Select the options as shown in the image below. Make sure you select the correct drive for your microSD card and give your device a name and admin password.



Select the WiFi network connection you want your Raspberry Pi to connect to, if required. Only networks your PC connects to will be shown.

Click **download and install**.

The application will now download the necessary files from Microsoft and flash them to your microSD card. It'll take a little while, but the dashboard will show you the progress.



Once the image has been installed on the microSD card, it's time to eject it from your PC and go over to the Raspberry Pi. First connect up the micro USB cable and power supply, HDMI cable and USB WiFi adapter or Ethernet cable. Connect the HDMI cable to your chosen display, insert the microSD card into the Raspberry Pi and power it up.