UNIVERSITY OF MUMBAI

Teacher's Reference Manual

Subject: Internet of Things

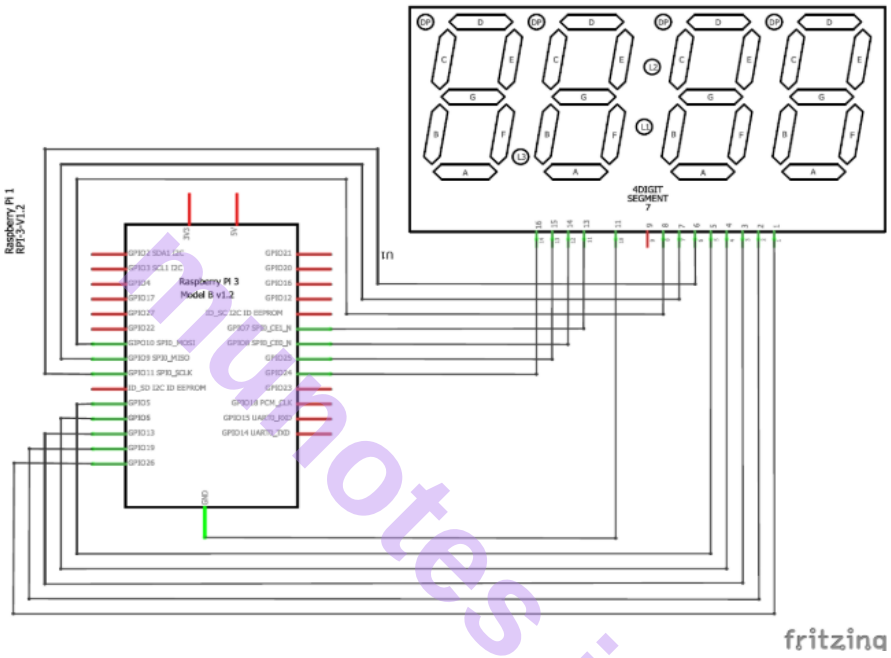
with effect from the academic year

2018 – 2019

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

Let us see how, how we can connect this 4-digit 7-segment module with our Raspberry Pi. The 7-segment module has 16 pins as shown below. You module might have lesser, but don't worry it will still have the following for sure

1. 7 or 8 segment pins (here pins starting from 1 to 8)
2. Ground pin (here pin 11)
3. 4 digit pins (here pins 13 to 16)



The following table will also help you in making the connections and verifying it to be as per the schematics shown above.

S.No	Rsp Pi GPIO number	Rsp Pi PIN number	7-Segment name	7-Seg pin number (in this module)
1	GPIO 26	PIN 37	Segment a	1
2	GPIO 19	PIN 35	Segment b	2
3	GPIO 13	PIN 33	Segment c	3
4	GPIO 6	PIN 31	Segment d	4
5	GPIO 5	PIN 29	Segment e	5

6	GPIO 11	PIN 23	Segment f	6
7	GPIO 9	PIN 21	Segment g	7
8	GPIO 10	PIN 19	Segment DP	8
9	GPIO 7	PIN 26	Digit 1	13
10	GPIO 8	PIN 24	Digit 2	14
11	GPIO 25	PIN 22	Digit 3	15
12	GPIO 24	PIN 18	Digit 4	16
13	Ground	Ground	Ground	11

Programming your Raspberry Pi:

First we are going to **import GPIO file from library**, below function enables us to program GPIO pins of PI. We are also renaming "GPIO" to "IO", so in the program whenever we want to refer to GPIO pins we will use the word 'IO'. We have also imported *time* and *datetime* to read the value of time from Rsp Pi.

```
import RPi.GPIO as GPIO
import time, datetime
```

Sometimes, when the GPIO pins, which we are trying to use, might be doing some other functions. In that case, we will receive warnings while executing the program. Below command tells the PI to **ignore the warnings** and proceed with the program.

```
IO.setwarnings(False)
```

We can refer the GPIO pins of PI, either by pin number on board or by their function number. Like 'PIN 29' on the board is 'GPIO5'. So we tell here either we are going to represent the pin here by '29' or '5'. GPIO.BCM means we will represent using 5 for GPIO5 pin 29.

```
IO.setmode (GPIO.BCM)
```

As always we should begin by **initialising the pins**, here both the **segment pins and the digit pins are output pins**. For programming purpose we form arrays for segment pins and initialize them to '0' after declaring them as GPIO.OUT

```
segment8 = (26,19,13,6,5,11,9,10)
for segment in segment8:
    GPIO.setup(segment, GPIO.OUT)
    GPIO.output(segment, 0)
```

Similarly for the digit pins we declare them as output pins and make them '0' by default

```
#Digit 1
    GPIO.setup(7, GPIO.OUT)
    GPIO.output(7, 0) #Off initially
    #Digit 2
    GPIO.setup(8, GPIO.OUT)
    GPIO.output(8, 0) #Off initially
    #Digit 3
    GPIO.setup(25, GPIO.OUT)
    GPIO.output(25, 0) #Off initially
    #Digit 4
    GPIO.setup(24, GPIO.OUT)
    GPIO.output(24, 0) #Off initially
```

We have to **form arrays to display each number on a seven segment display**. To display one number we have to control all 7 segment pins (dot pin excluded), that is they either has to be turned off or turned on. For example to display the number 5 we have make the following arrangement

S.No	Rsp Pi GPIO number	7-Segment name	Status to display '5'. (0-> OFF, 1->ON)
1	GPIO 26	Segment a	1
2	GPIO 19	Segment b	1

3	GPIO 13	Segment c	0
4	GPIO 6	Segment d	1
5	GPIO 5	Segment e	1
6	GPIO 11	Segment f	0
7	GPIO 9	Segment g	1

Similarly we have **sequence number for all numbers** and alphabets. You can write on your own or use the chart below.

Number	g f e d c b a	Hexadecimal
0	0 1 1 1 1 1 1	3F
1	0 0 0 0 1 1 0	06
2	1 0 1 1 0 1 1	5B
3	1 0 0 1 1 1 1	4F
4	1 1 0 0 1 1 0	66
5	1 1 0 1 1 0 1	6D
6	1 1 1 1 1 0 1	7D
7	0 0 0 0 1 1 1	07
8	1 1 1 1 1 1 1	7F
9	1 1 0 1 1 1 1	6F

With these data we can form the **arrays for each number** in our python program as shown below.

```
null = [0,0,0,0,0,0,0]

zero = [1,1,1,1,1,1,0]

one = [0,1,1,0,0,0,0]

two = [1,1,0,1,1,0,1]

three = [1,1,1,1,0,0,1]

four = [0,1,1,0,0,1,1]
```

```
five = [1,0,1,1,0,1,1]
six = [1,0,1,1,1,1,1]
seven = [1,1,1,0,0,0,0]
eight = [1,1,1,1,1,1,1]
nine = [1,1,1,1,0,1,1]
```

If you follow the program there will be a function to display each character to our 7-segment display but, lets skip this for now and get into the *while* infinite loop. Where **read the present time from Raspberry Pi and split the value of time between four variables**. For example if the time is 10.45 then the variable h1 will have 1, h2 will have 0, m1 will have 4vand m2 will have 5.

```
now = datetime.datetime.now()
hour = now.hour
minute = now.minute
h1 = hour/10
h2 = hour % 10
m1 = minute /10
m2 = minute % 10
print (h1,h2,m1,m2)
```


We have to display these four variable values on our four digits respectively. To write a value of variable to a digit we can use the following lines. Here we are display on digit 1 by making it go high then the function ***print_segment (variable)*** will be called to display the value in variable on the segment display. You might be wondering why we have a delay after that and why we turn this digit off after this.

```
GPIO.output(7, 1) #Turn on Digit One
print_segment (h1) #Print h1 on segment
time.sleep(delay_time)
GPIO.output(7, 0) #Turn off Digit One
```

The reason is, as we know we can display only one digit at a time, but we have four digits to be displayed and only if all the four digits are displayed the complete four digit number will be visible for the user.

The last thing to learn it to know how the *print_segment(variable)* function works. Inside this function we use the arrays that we have declared so far. So whatever variable that we send to this function should have the value between (0-9), the variable character will receive this value and compare it for real value. Here the variable is compared with '1'. Similarly we compare with all number from 0 to 9. If it is a match we use the arrays and assign each value to its respective segment pins as shown below.

```
def print_segment(charactor):
  if charactor == 1:
    for i in range(7):
      GPIO.output(segment8[i], one[i])
```

Display time on 4-Digit 7-segment using Raspberry Pi:

Use the schematic and code given here to make the connections and program your raspberry pi accordingly. After everything is done just launch the program and you should find the current time being displayed in the seven segment display. But, there are few things that you have to check before this

1. Make sure you have set your Raspberry Pi with current time just in case if it running on offline time.
2. Power your Raspberry pi with a Adapter and not with your Laptop/computer because the amount of current drawn by the 7-segment display is high and your USB port cannot source it.

Code:

```
import RPi.GPIO as GPIO
import time, datetime
now = datetime.datetime.now()
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

 #GPIO ports for the 7seg pins
segment8 =  (26,19,13,6,5,11,9,10)


for segment in segment8:
   GPIO.setup(segment, GPIO.OUT)
   GPIO.output(segment, 0)
```

```
    #Digit 1
    GPIO.setup(7, GPIO.OUT)
    GPIO.output(7, 0) #Off initially
    #Digit 2
    GPIO.setup(8, GPIO.OUT)
    GPIO.output(8, 0) #Off initially
    #Digit 3
    GPIO.setup(25, GPIO.OUT)
    GPIO.output(25, 0) #Off initially
    #Digit 4
    GPIO.setup(24, GPIO.OUT)
    GPIO.output(24, 0) #Off initially

null = [0,0,0,0,0,0,0]
zero = [1,1,1,1,1,1,0]
one = [0,1,1,0,0,0,0]
two = [1,1,0,1,1,0,1]
three = [1,1,1,1,0,0,1]
four = [0,1,1,0,0,1,1]
five = [1,0,1,1,0,1,1]
six = [1,0,1,1,1,1,1]
seven = [1,1,1,0,0,0,0]
eight = [1,1,1,1,1,1,1]
nine = [1,1,1,1,0,1,1]

def print_segment(charector):
    if charector == 1:
        for i in range(7):
            GPIO.output(segment8[i], one[i])

    if charector == 2:
        for i in range(7):
            GPIO.output(segment8[i], two[i])

    if charector == 3:
        for i in range(7):
            GPIO.output(segment8[i], three[i])

    if charector == 4:
        for i in range(7):
            GPIO.output(segment8[i], four[i])

    if charector == 5:
        for i in range(7):
            GPIO.output(segment8[i], five[i])

    if charector == 6:
        for i in range(7):
            GPIO.output(segment8[i], six[i])
```

```
    if charector == 7:
        for i in range(7):
            GPIO.output(segment8[i], seven[i])
    if charector == 8:
        for i in range(7):
            GPIO.output(segment8[i], eight[i])
    if charector == 9:
        for i in range(7):
            GPIO.output(segment8[i], nine[i])
    if charector == 0:
        for i in range(7):
            GPIO.output(segment8[i], zero[i])

    return;
while 1:
    now = datetime.datetime.now()
    hour = now.hour
    minute = now.minute
    h1 = hour/10
    h2 = hour % 10
    m1 = minute /10
    m2 = minute % 10
    print (h1,h2,m1,m2)

    delay_time = 0.001 #delay to create virtual effect


    GPIO.output(7, 1) #Turn on Digit One
    print_segment (h1) #Print h1 on segment
    time.sleep(delay_time)
    GPIO.output(7, 0) #Turn off Digit One
    GPIO.output(8, 1) #Turn on Digit One
    print_segment (h2) #Print h1 on segment
    GPIO.output(10, 1) #Display point On
    time.sleep(delay_time)
    GPIO.output(10, 0) #Display point Off
    GPIO.output(8, 0) #Turn off Digit One
    GPIO.output(25, 1) #Turn on Digit One
    print_segment (m1) #Print h1 on segment
    time.sleep(delay_time)
    GPIO.output(25, 0) #Turn off Digit One
```

```
GPIO.output(24, 1) #Turn on Digit One
print_segment (m2) #Print h1 on segment
time.sleep(delay_time)
GPIO.output(24, 0) #Turn off Digit One

#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

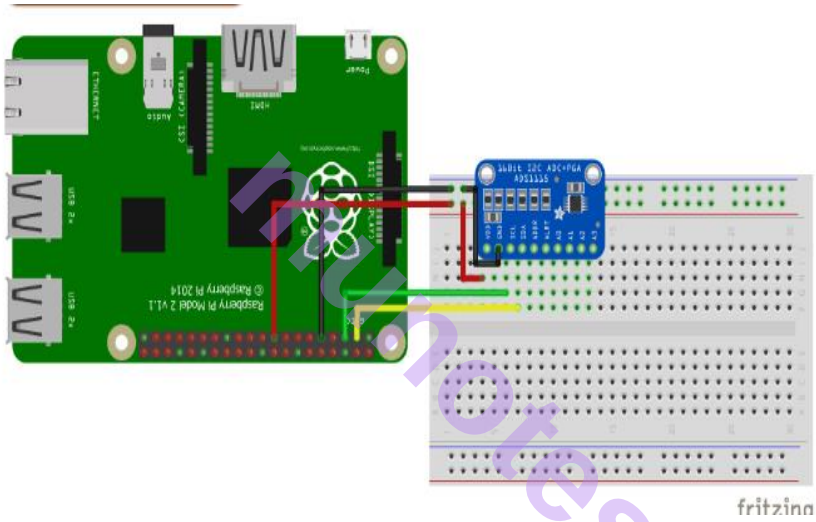
The software requirements for this project are basically the python modules (***matplotlib and drawnow***) that will be used for data visualization and the Adafruit module for interfacing with the ADS1115 ADC chip. I will show how to install these modules on the Raspberry Pi as we proceed.

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;

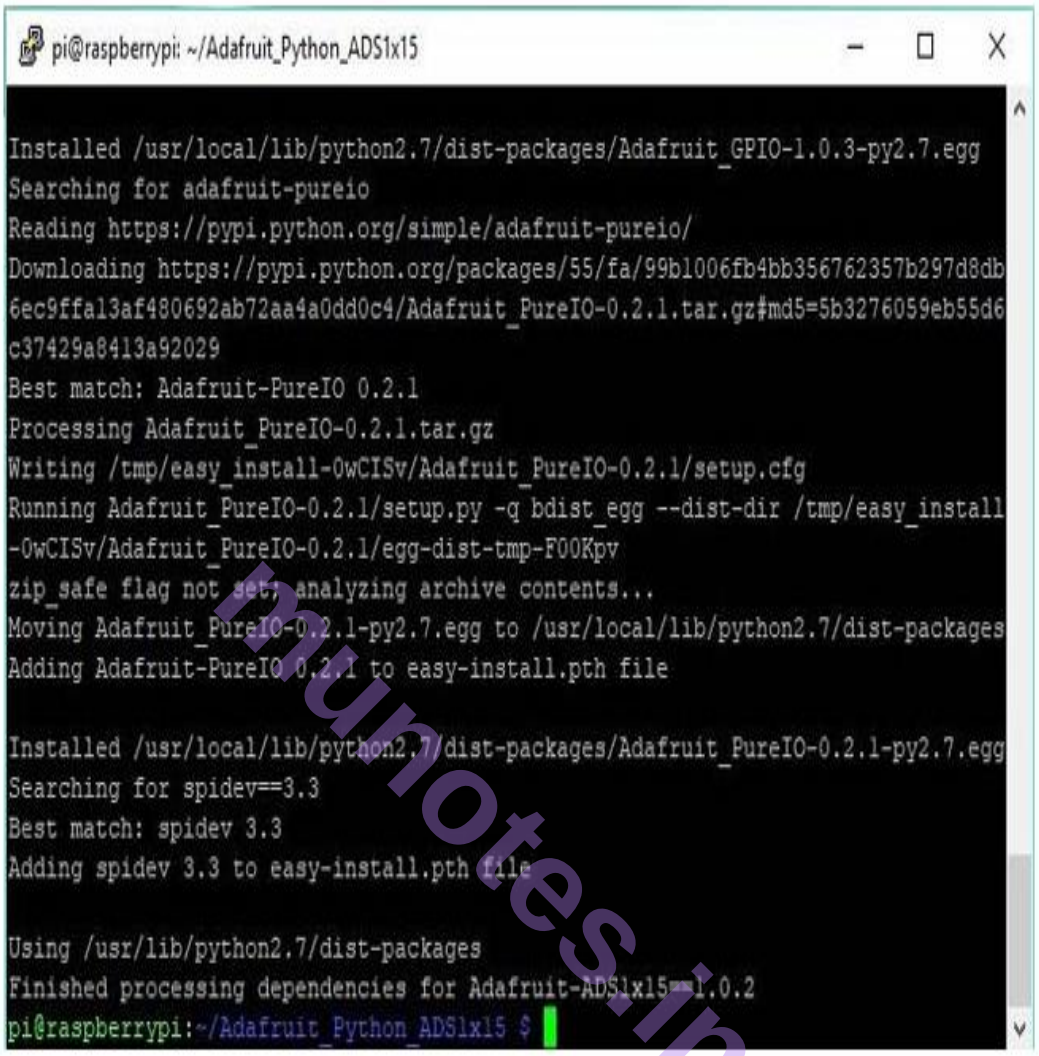
git clone https://github.com/adafruit/Adafruit_Python_ADS1x15.git

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

cd Adafruit_Python_ADS1x1z

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 |
```

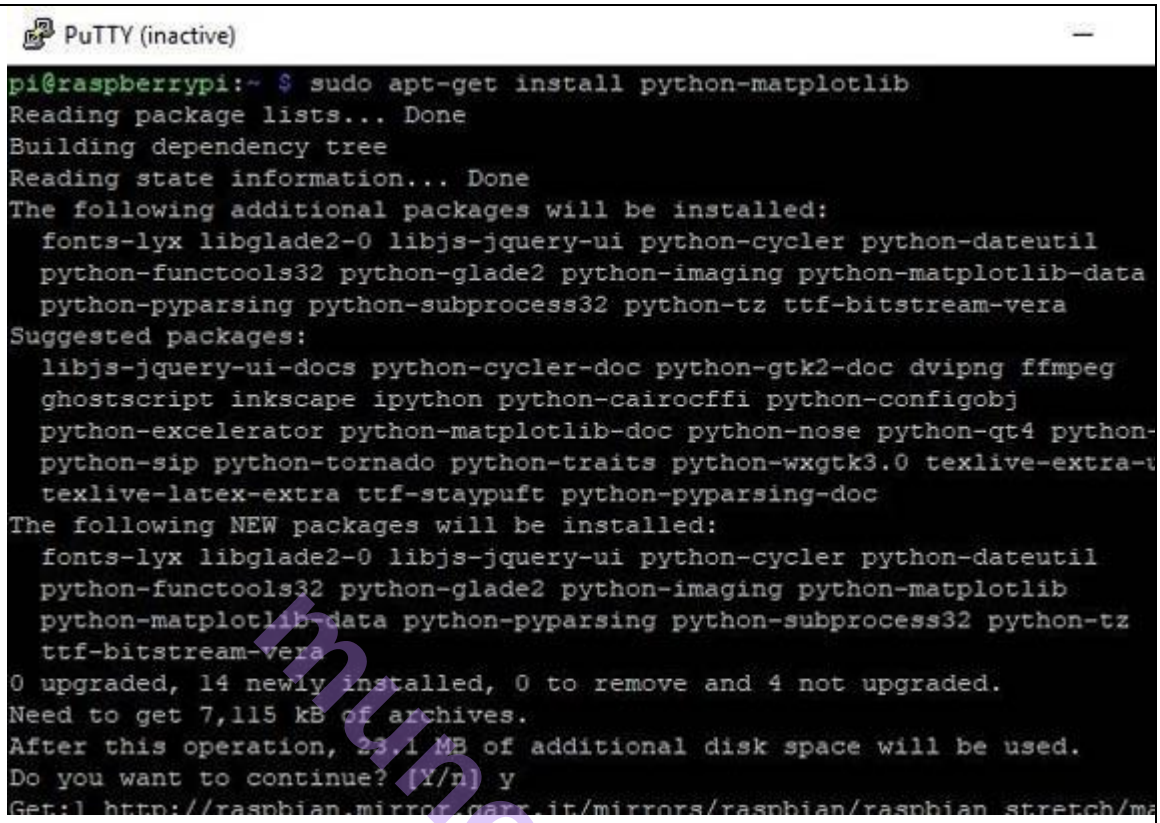
If an error occurs, check to ensure the ADC is well connected to the PI and I2C communication is enabled on the Pi.

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-pip
```

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.

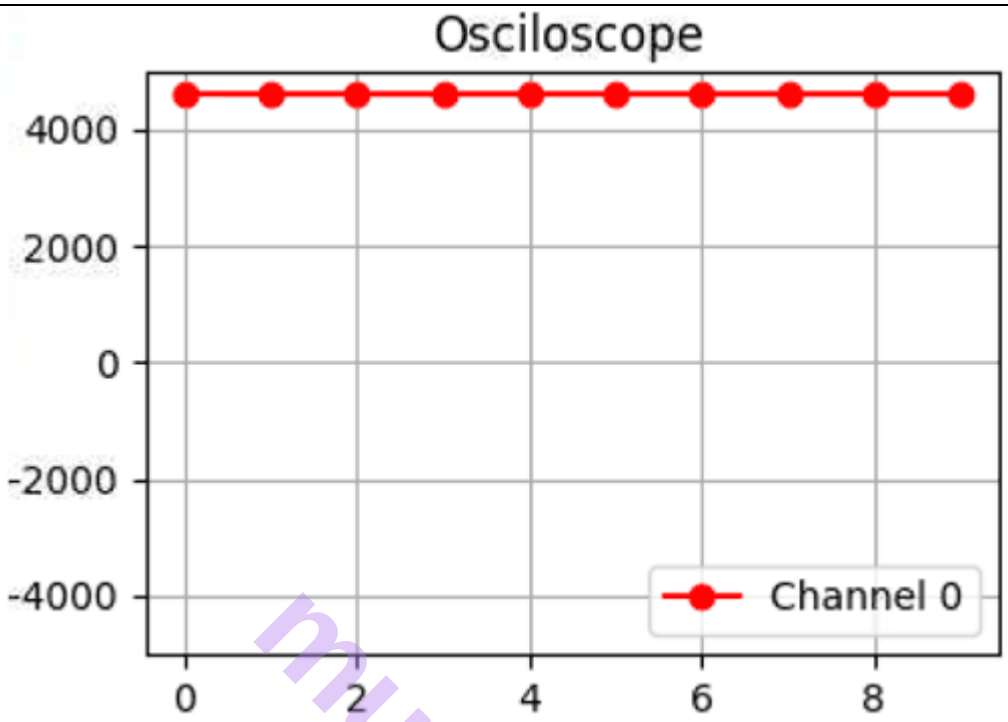
After a few minutes, you should see the ADC data being printed on the terminal. Occasionally you may get a warning from *matplotlib* (as shown in the image below) which should be suppressed but it doesn't affect the data being displayed or the plot in anyway. To suppress the warning however, the following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

```
pi@raspberrypi:~ $ sudo nano scope.py
pi@raspberrypi:~ $ sudo python scope.py
Reading ADS1x15 channel 0
Channel 0: 4618
/usr/lib/python2.7/dist-packages/matplotlib/backend
plotlibDeprecationWarning: Using default event loop
cific to this GUI is implemented
  warnings.warn(str, mplDeprecation)
Channel 0: 4615
Channel 0: 4616
Channel 0: 4615
Channel 0: 4614
Channel 0: 4613
Channel 0: 4614
```



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 WhatsApp.**

Step 1: Installation

Update the packages with

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

Update firmware

```
sudo rpi-update
```

Prepare the system with the necessary components to Yowsup

```
sudo apt-get install python-dateutil
sudo apt-get install python-setuptools
sudo apt-get install python-dev
sudo apt-get install libevent-dev
sudo apt-get install ncurses-dev
```

Download the library with the command

```
git clone git://github.com/tgalal/yowsup.git
```

navigate to the folder

cd yowsup

and install the library with the command

sudo python setup.py install

Step 2: Registration

After installing the library we have to register the device to use WhatsApp. Yowsup comes with a cross platform command-line frontend called yowsup-cli. It provides you with the options of registration, and provides a few demos such as a command line client.

WhatsApp registration involves 2 steps. First you need to request a registration code. And then you resume the registration with code you got.

Request a code with command

python yowsup-cli registration --requestcode sms --phone 39xxxxxxxxx --cc 39 --mcc 222 --mnc 10

Replace with your data ,

cc is your country code in this example 39 is for Italy,

mcc is Mobile Country Code

mnc is Mobile Network Code

You should receive on your phone a sms message with a code like xxx-xxx

Send a message to request registration with this command, (replace xxx-xxx with code you received)

python yowsup-cli registration --register xxx-xxx --phone 39xxxxxxxxx --cc 39

If all goes well, we should get a message like this

status: ok

kind: free

pw: xxxxxxxxxxxxxxxxxx=

price: € 0,89

price_expiration: 1416553637

currency: EUR

cost: 0.89

expiration: 1445241022

login: 39xxxxxxxxxx

type: existing

Warning

WhatsApp requires the registration of a number, and with that number you can use WhatsApp on only one device at a time, so it is preferable to use a new number.

WhatsApp can be used on one device at a time and if you will make many attempts to register the number, it could be banned. We recommend you using Telegram.

Step 3: Utilization

Create a file to save your credentials

sudo nano /home/pi/yowsup/config

with this content

Actual config starts below

cc=39 #if not specified it will be autodetected

phone=39xxxxxxxxx

password=xxxxxxxxxxxxxxx=

Ok, we're ready for the test, Yowsup has a demo application in /home/pi/yowsup/yowsup/demos

Navigate to yowsup folder

	cd /home/pi/yowsup
	Start yowsup-cli demos with the command
	yowsup-cli demos --yowsup --config config
	You can see Yowsup prompt If type "/help" you can see all available commands First use the '/L' command for login; to send a message type
	/message send 39xxxxxxxxxx "This is a message sent from Raspberry Pi"
	replace xxx with the recipient number If you respond with a message it will be displayed on Raspberry.
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

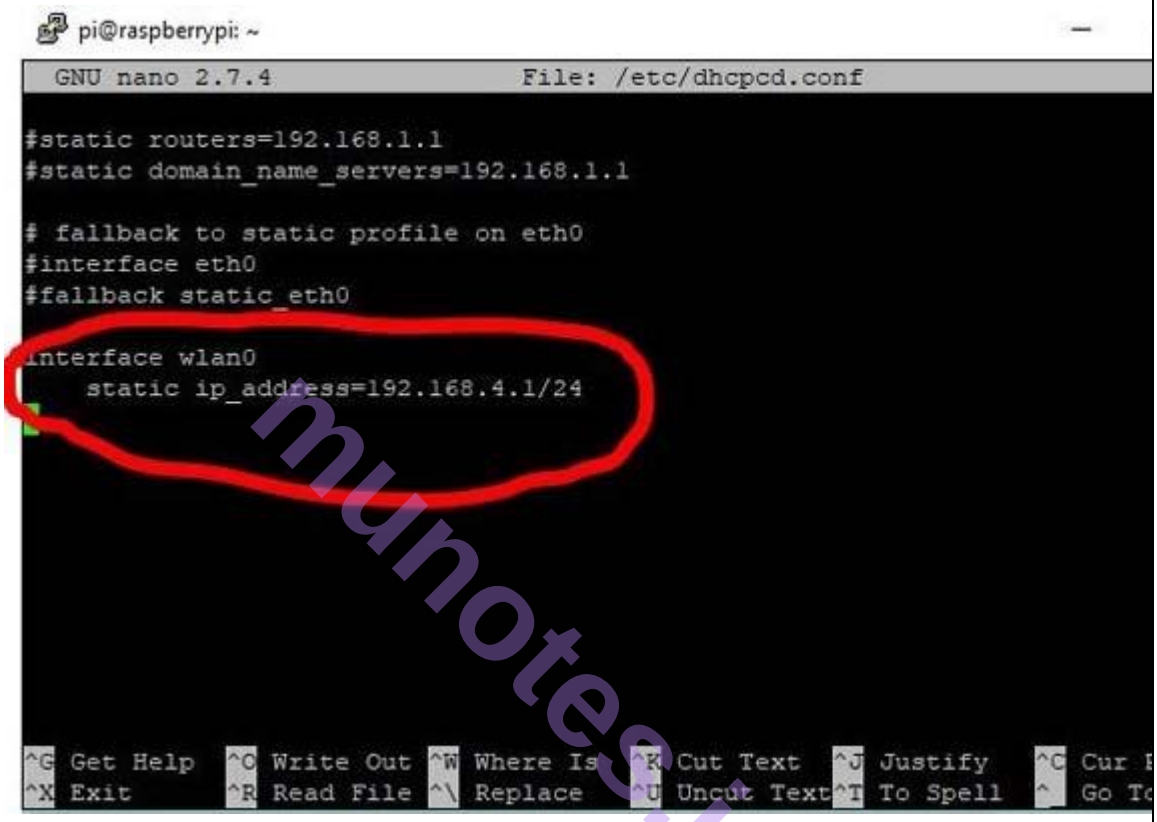
Confirm the *wlan* port on which the wireless device being used is connected. For my Pi, the wireless is on wlan0. **Setting up the Raspberry Pi to act as a server** requires us to assign a static IP address to the wireless port. This can be done by editing the *dhcpcd* config file. To edit the configuration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

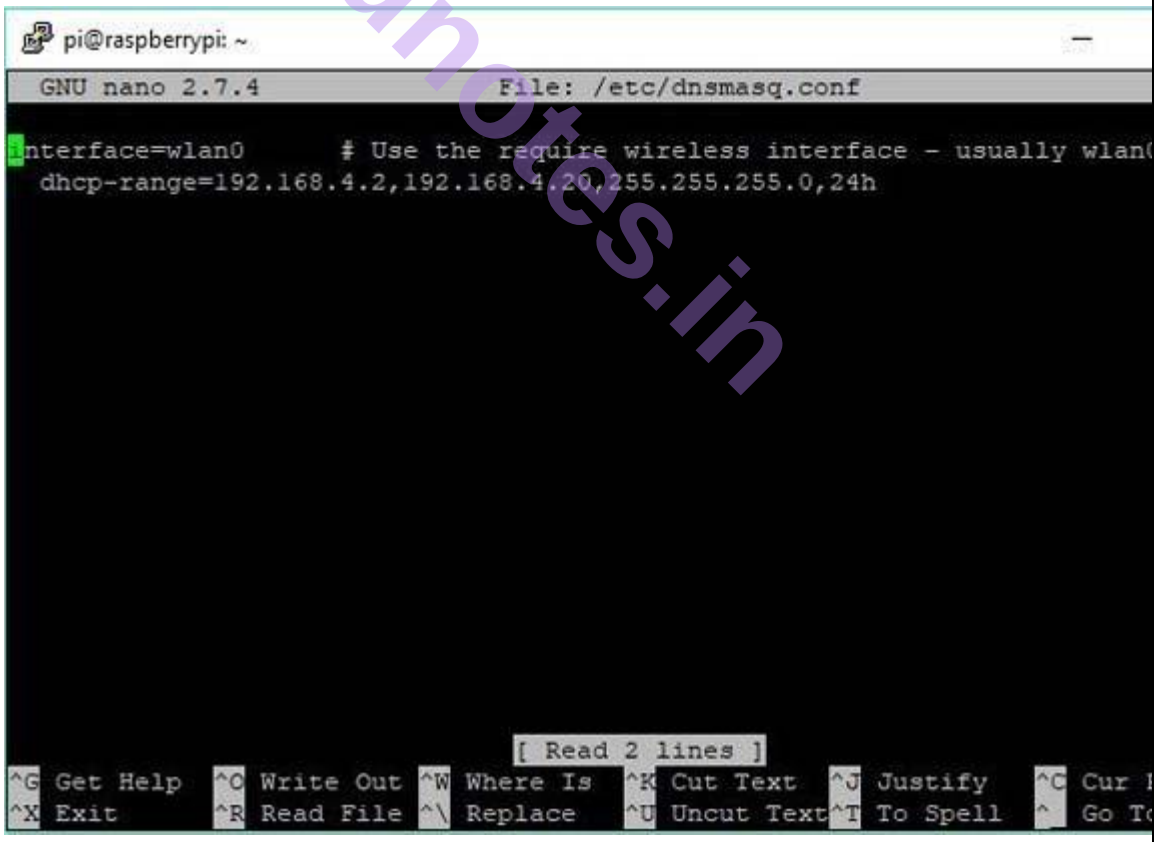
```
sudo nano /etc/dnsmasq.conf
```

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.

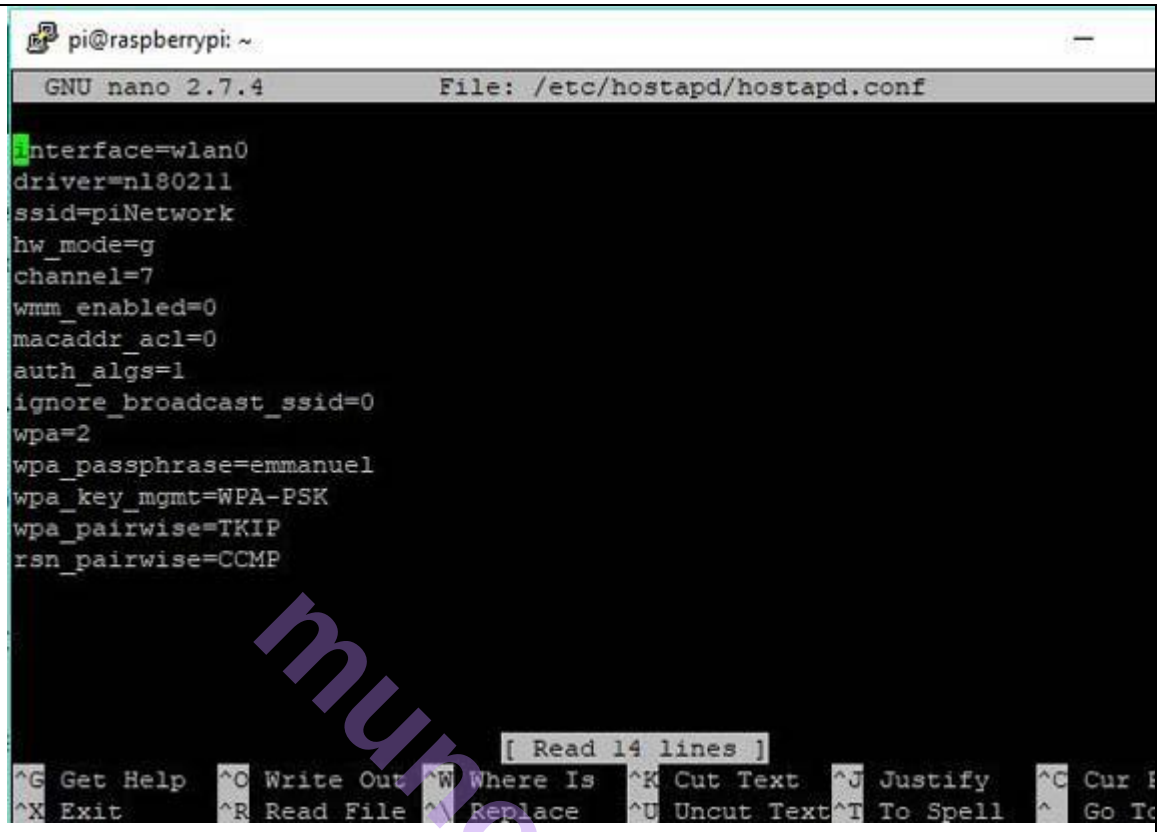
The next set of steps will help us configure the access point host software, setup the ssid, select the encrytpion etc.

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.



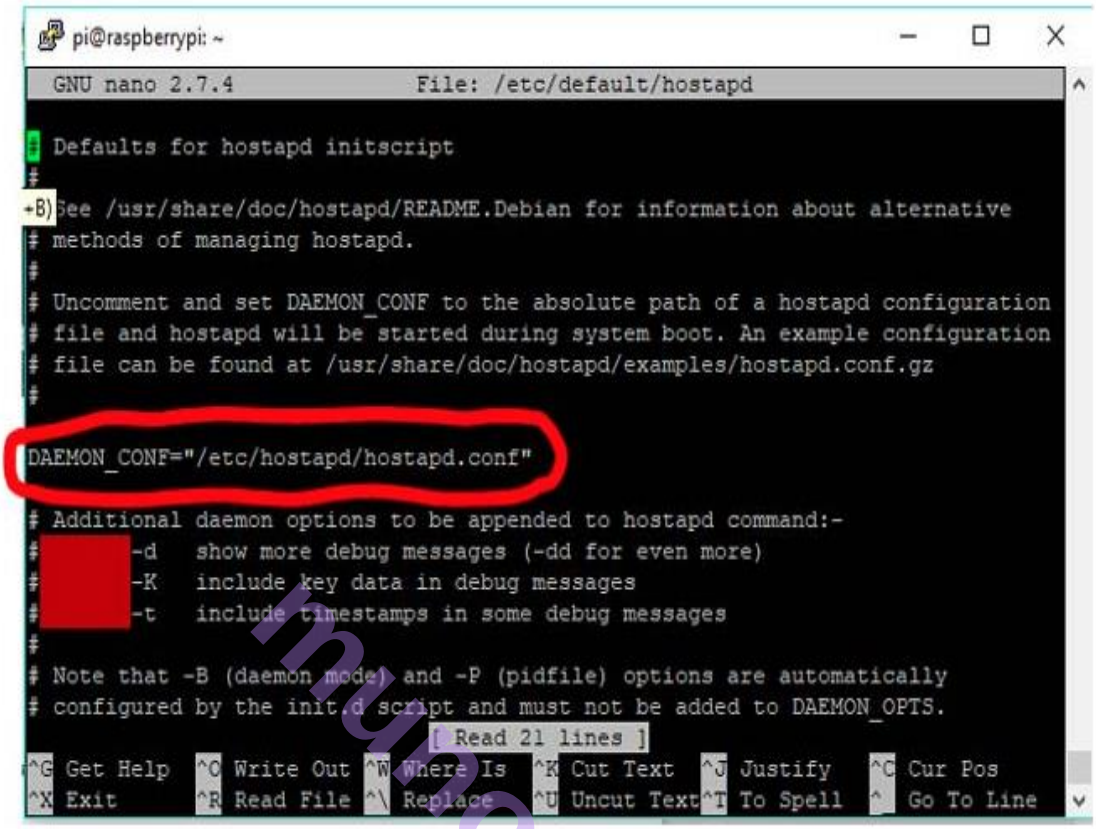
Feel free to change the ssid and password to suit your needs and desire.

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

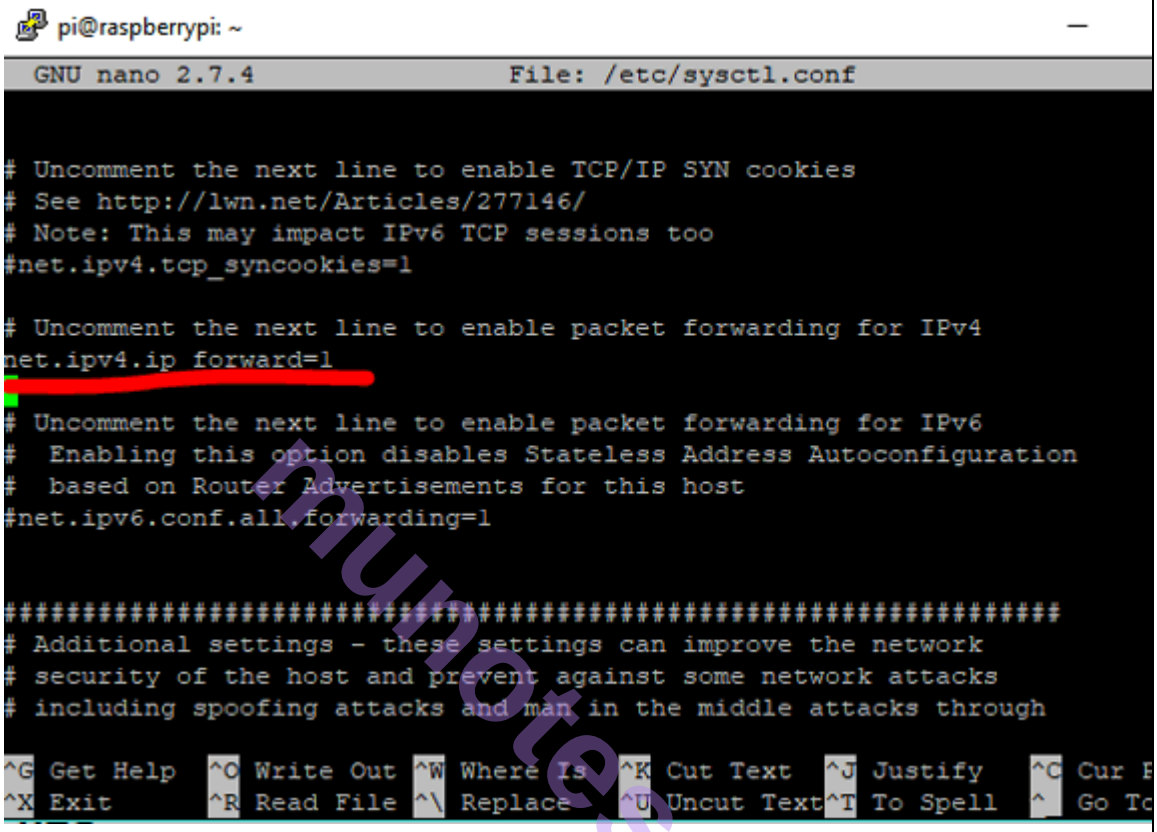
Step 8: Routing and masquerade for outbound traffic

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
```

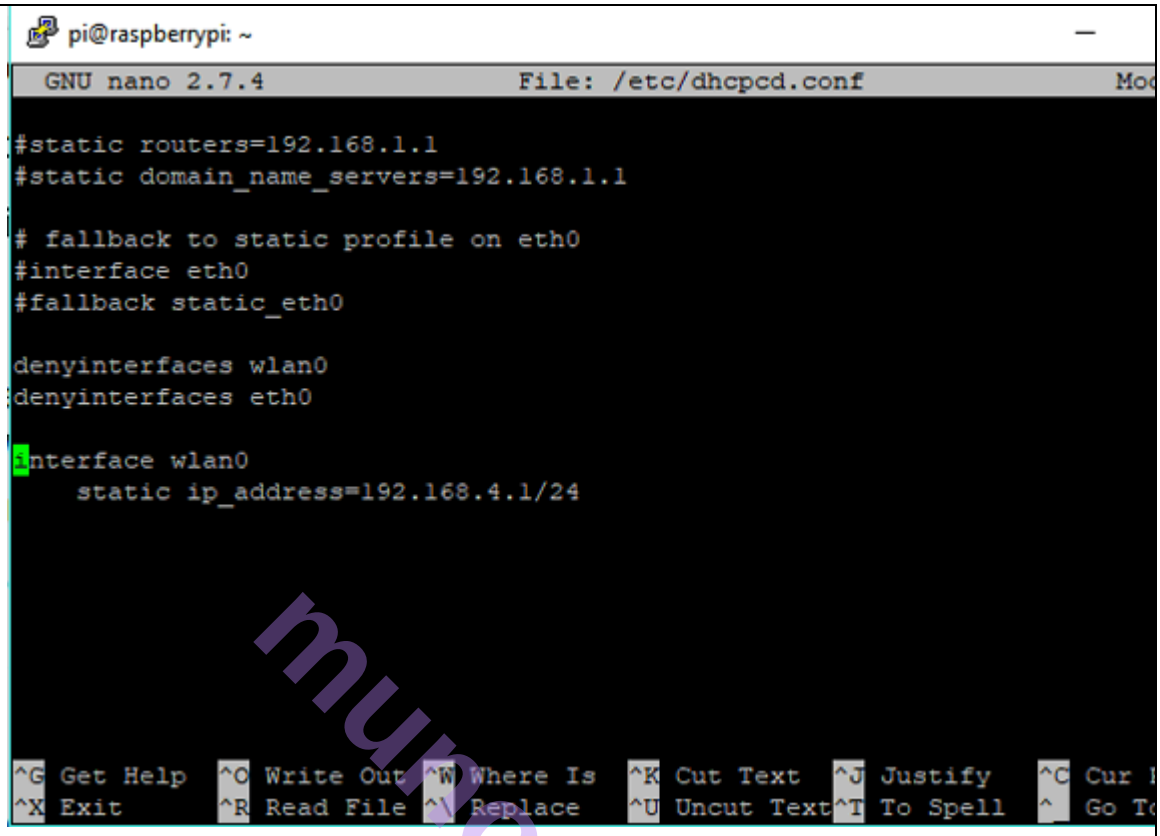
Next, we stop hostapd so as to configure the software.

```
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	**Fingerprint Sensor interfacing with Raspberry Pi**
	Finger Print Sensor, which we used to see in Sci-Fi moives few years back, is now become very common to verify the identity of a person for various purposes. In present time we can see fingerprint-based systems everywhere in our daily life like for attendance

in offices, employee verification in banks, for cash withdrawal or deposits in ATMs, for identity verification in government offices etc. We have already interfaced it with Arduino, today we are going to **interface FingerPrint Sensor with Raspberry Pi**. Using this Raspberry Pi FingerPrint System, we can enroll new finger prints in the system and can delete the already fed finger prints. Complete working of the system has been shown in the **Video** given at the end of article.

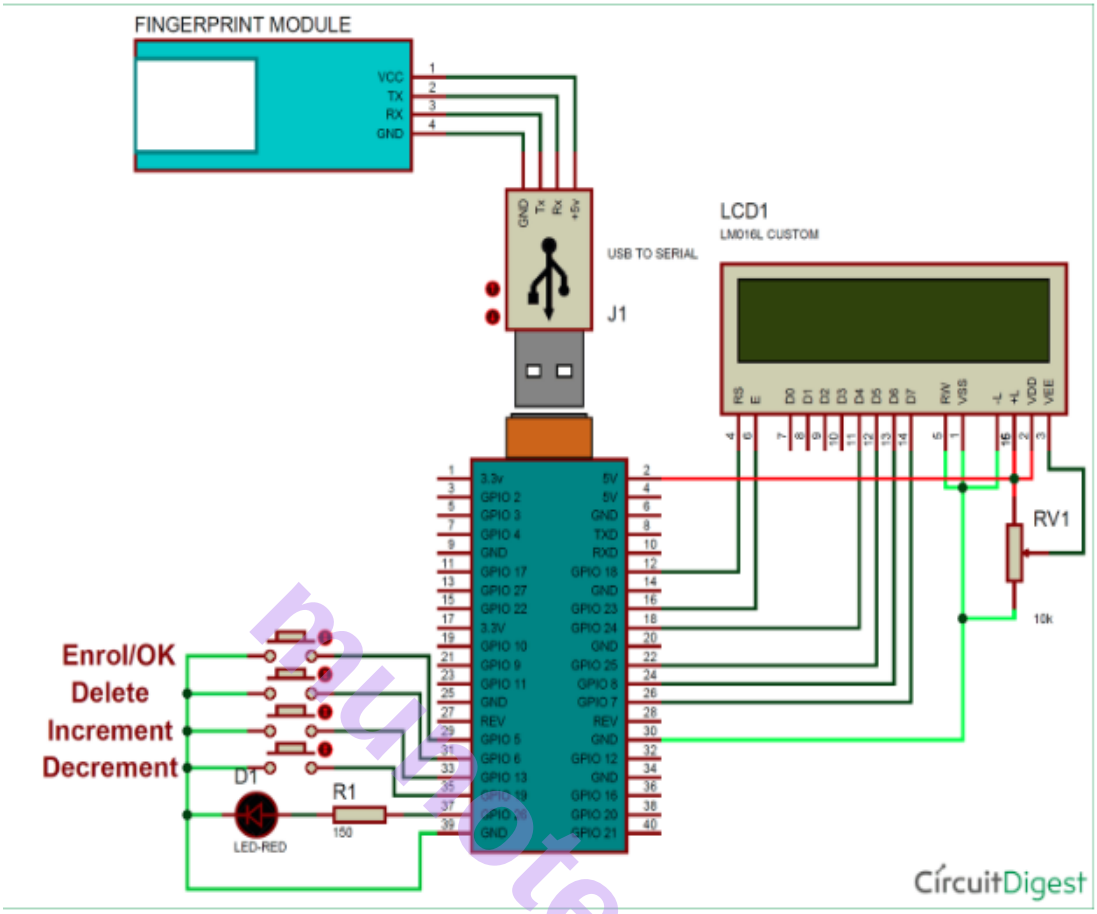
Required Components:

1. Raspberry Pi
2. USB to Serial converter
3. Fingerprint Module
4. Push buttons
5. 16x2 LCD
6. 10k pot
7. Bread Board or PCB (ordered from JLCPCB)
8. Jumper wires
9. LED (optional)
10. Resistor 150 ohm -1 k ohm (optional)

Circuit Diagram and Explanation:

In this **Raspberry Pi Finger Print sensor interfacing project**, we have used a **4 push buttons**: one for enrolling the new finger pring, one for deleting the already fed finger prints and rest two for increment/decrement the position of already fed Finger prints. A **LED** is used for indication that fingerprint sensor is ready to take finger for matching. Here we have used a fingerprint module which works on UART. So here we have interfaced this fingerprint module with Raspberry Pi using a **USB to Serial converter**.

So, first of all, we need to make the all the required connection as shown in Circuit Diagram below. Connections are simple, we have just connected fingerprint module to Raspberry Pi USB port by using USB to Serial converter. A 16x2 LCD is used for displaying all messages. A 10k pot is also used with LCD for controlling the contrast of the same. 16x2 LCD pins RS, EN, d4, d5, d6, and d7 are connected with GPIO Pin 18, 23, 24, 25, 8 and 7 of Raspberry Pi respectively. Four push buttons are connected to GPIO Pin 5, 6, 13 and 19 of Raspberry Pi. LED is also connected at pin 26 of RPI.



Installing Library for Finger Print Sensor:

After making all the connections we need to power up Raspberry Pi and get it ready with terminal open. Now we need to **install fingerprint library for Raspberry Pi** in python language by following the below steps.

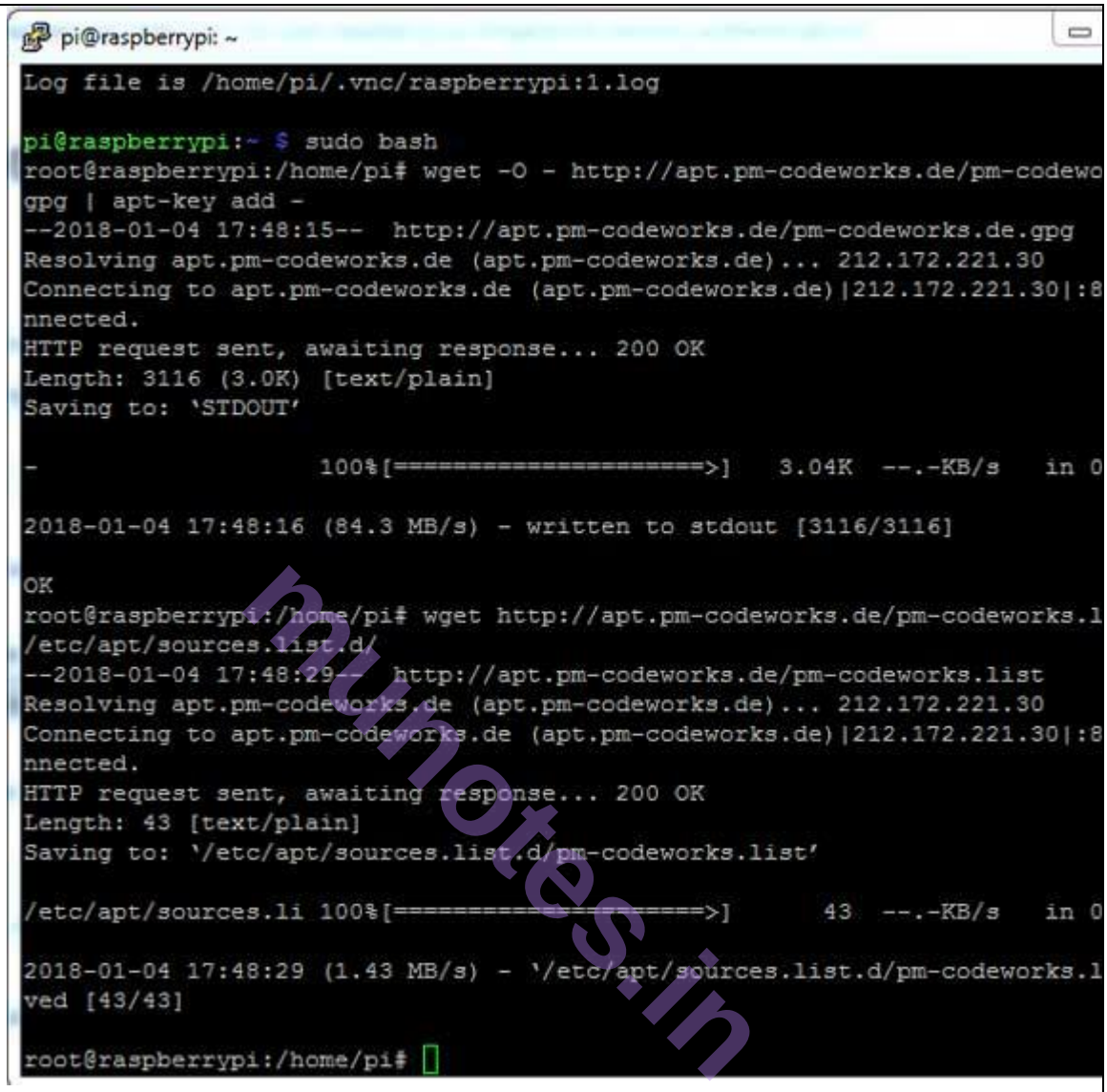
Step 1: To install this library, root privileges are required. So first we enter in *root* by given command:

```
sudo bash
```

Step 2: Then **download some required packages** by using given commands:

```
wget –O – http://apt.pm-codeworks.de/pm-codeworks.de.gpg | apt-key add –

wget http://apt.pm-codeworks.de/pm-codeworks.list -P /etc/apt/sources.list.d/
```



Step 3: After this, we need to **update the Raspberry pi and install the downloaded finger print sensor library**:

sudo apt-get update

sudo apt-get install python-fingerprint –yes

```
root@raspberrypi:/home/pi# apt-get install python-fingerprint --yes
Reading package lists... Done
Building dependency tree
Reading state information... Done
python-fingerprint is already the newest version.
The following packages were automatically installed and are no longer req
  libasn1-8-heimdal libgssapi3-heimdal libhcrypto4-heimdal
  libheimbase1-heimdal libheimntlm0-heimdal libhx509-5-heimdal
  libkrb5-26-heimdal libroken18-heimdal libwind0-heimdal
Use 'apt-get autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 294 not upgraded.
root@raspberrypi:/home/pi# exit
exit
pi@raspberrypi:~ $
```

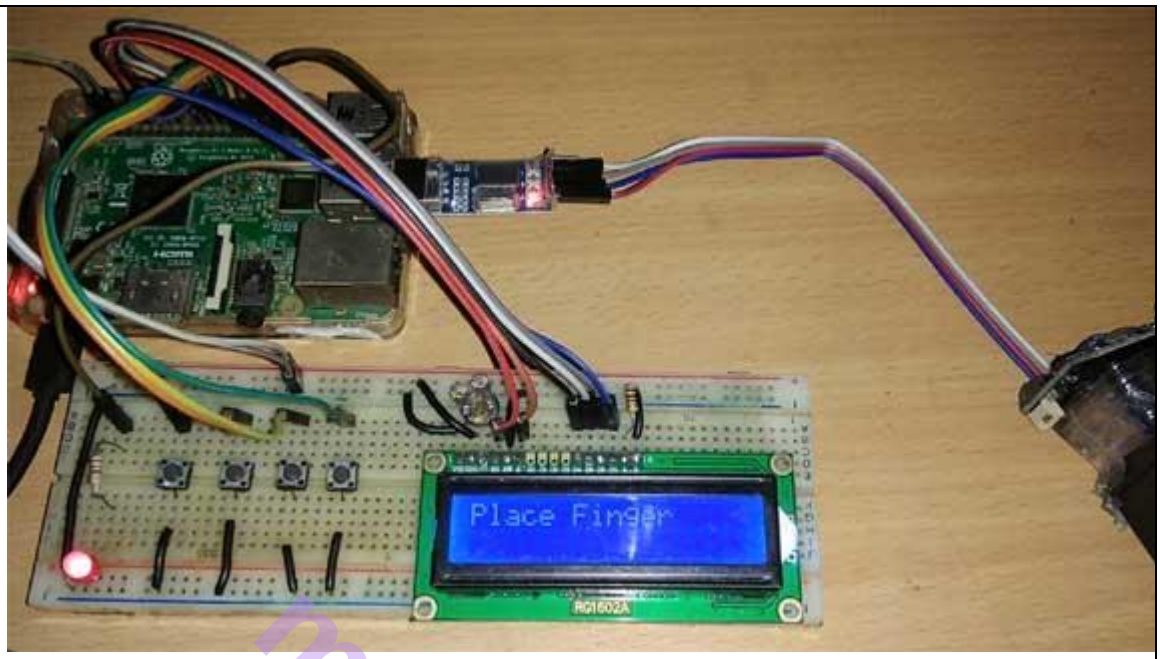
Step 4: After installing library now we need to **check USB port** on which your finger print sensor is connected, by using given the command:

```
ls /dev/ttyUSB*
```

Now replace the USB port no., with the USB port you got over the screen and replace it in the python code. **Complete Python code** is given at the end of this project.

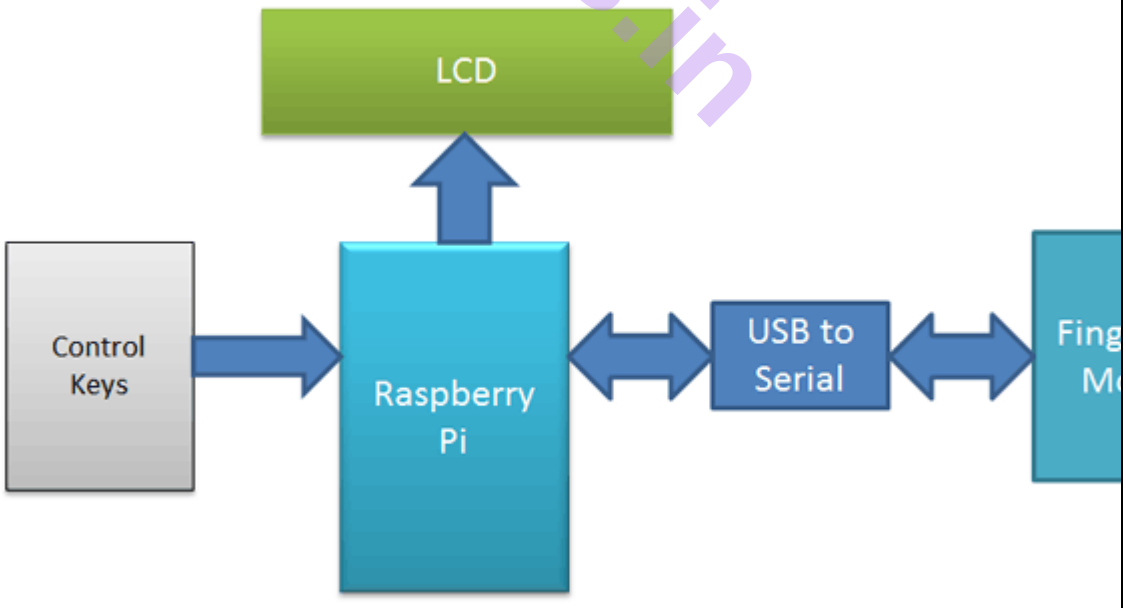
Operation of Fingerprint Sensor with Raspberry Pi:

Operation of this project is simple, just run the python code and there will be some intro messages over LCD and then user will be asked to ***Place Finger*** on Finger Print Sensor. Now by putting a finger over fingerprint module, we can check whether our finger prints are already stored or not. If your fingerprint is stored then LCD will show the message with the storing position of fingerprint like ***'Fount at Pos:2'*** otherwise it will show ***'No Match Found'.***



Now **to enroll a finger Print,** user needs to press enroll button and follow the instructions messages on LCD screen.

If the user wants **to delete any of fingerprints** then the user needs to press *delete button.* After which, LCD will ask for the position of the fingerprint which is to be deleted. Now by using another two push button for increment and decrement, user can select the position of saved Finger Print and press enroll button (at this time **enroll button behave as Ok button**) to delete that fingerprint. For more understanding have a look at the **video given at the end** of the project.



Python Programming:

Python for **interfacing Finger Print Sensor with RPi** is easy with using fingerprint library functions. But if the user wants to interface it himself, then it will be little bit difficult for the first time. In finger print sensor datasheets, everything is given that is required for interfacing the same module. A GitHub code is available to test your Raspberry pi with Finger Print sensor.

Here we have used the library so we just need to call library function. In code, first we need to import libraries like fingerprint, GPIO and time, then we need to **define pins for LCD, LED and push buttons.**

```
import time
from pyfingerprint.pyfingerprint import PyFingerprint
import RPi.GPIO as gpio


RS =18
EN =23
D4 =24
D5 =25
D6 =8
D7 =7


enrol=5
delet=6
inc=13
dec=19
led=26


HIGH=1
LOW=0
```


After this, we need to **initialize and give direction to the selected pins**

```
gpio.setwarnings(False)
gpio.setmode(gpio.BCM)
gpio.setup(RS, gpio.OUT)
gpio.setup(EN, gpio.OUT)
gpio.setup(D4, gpio.OUT)
gpio.setup(D5, gpio.OUT)
gpio.setup(D6, gpio.OUT)
gpio.setup(D7, gpio.OUT)

gpio.setup(enrol, gpio.IN, pull_up_down=gpio.PUD_UP)
gpio.setup(delet, gpio.IN, pull_up_down=gpio.PUD_UP)
gpio.setup(inc, gpio.IN, pull_up_down=gpio.PUD_UP)
gpio.setup(dec, gpio.IN, pull_up_down=gpio.PUD_UP)
gpio.setup(led, gpio.OUT)
```

Now we have **initialized fingerprint Sensor**

```
try:
    f = PyFingerprint('/dev/ttyUSB0', 57600, 0xFFFFFFFF, 0x00000000)
    if ( f.verifyPassword() == False ):
        raise ValueError('The given fingerprint sensor password is wrong!')
except Exception as e:
    print('Exception message: ' + str(e))
    exit(1)
```

We have written some function to initialize and drive the LCD, check the complete code below in code section:

def begin(), def lcdcmd(ch), def lcdwrite(ch), def lcdprint(Str), def setCursor(x,y)

After writing all LCD driver functions, we have placed functions for fingerprint enrolling, searching and deleting.

def enrollFinger() function is used for enrol or save the new finger prints. **def searchFinger()** function is used to searthc the already stored finger prints **def deleteFinger()** functinos is used to deoted the already saved finger print by pressing the correspontind push button.

All above function's Code is given the in **python code** given below.

After this, finally, we need to **initialize system** by in *while 1* loop by asking to *Place Finger* on finger print sensor and then system will check whether this finger print it valid or not and display the results accordingly.

```
begin()
lcdcmd(0x01)
lcdprint("FingerPrint ")
lcdcmd(0xc0)
lcdprint("Interfacing ")
time.sleep(3)
lcdcmd(0x01)
lcdprint("Circuit Digest")
lcdcmd(0xc0)
lcdprint("Welcomes You  ")
time.sleep(3)
flag=0
lcdclear()


while 1:
    gpio.output(led, HIGH)
    lcdcmd(1)
    lcdprint("Place Finger")
    if gpio.input(enrol) == 0:
```

```
        gpio.output(led, LOW)
        enrollFinger()
    elif gpio.input(delet) == 0:
        gpio.output(led, LOW)
        while gpio.input(delet) == 0:
            time.sleep(0.1)
        deleteFinger()
    else:
        searchFinger()
```

Complete Python Code is given below.

Code:

```
import time
from pyfingerprint.pyfingerprint import PyFingerprint
import RPi.GPIO as gpio

RS =18
EN =23
D4 =24
D5 =25
D6 =8
D7 =7

enrol=5
delet=6
inc=13
dec=19
led=26

HIGH=1
LOW=0

gpio.setwarnings(False)
gpio.setmode(gpio.BCM)
gpio.setup(RS, gpio.OUT)
gpio.setup(EN, gpio.OUT)
gpio.setup(D4, gpio.OUT)
gpio.setup(D5, gpio.OUT)
gpio.setup(D6, gpio.OUT)
gpio.setup(D7, gpio.OUT)
```

```
gpio.setup(enrol, gpio.IN, pull_up_down=gpio.PUD_UP)
gpio.setup(delet, gpio.IN, pull_up_down=gpio.PUD_UP)
gpio.setup(inc, gpio.IN, pull_up_down=gpio.PUD_UP)
gpio.setup(dec, gpio.IN, pull_up_down=gpio.PUD_UP)
gpio.setup(led, gpio.OUT)

try:
    f = PyFingerprint('/dev/ttyUSB0', 57600, 0xFFFFFFFF, 0x00000000)

    if ( f.verifyPassword() == False ):
        raise ValueError('The given fingerprint sensor password is wrong!')

except Exception as e:
    print('Exception message: ' + str(e))
    exit(1)

def begin():
  lcdcmd(0x33)
  lcdcmd(0x32)
  lcdcmd(0x06)
  lcdcmd(0x0C)
  lcdcmd(0x28)
  lcdcmd(0x01)
  time.sleep(0.0005)

def lcdcmd(ch):
  gpio.output(RS, 0)
  gpio.output(D4, 0)
  gpio.output(D5, 0)
  gpio.output(D6, 0)
  gpio.output(D7, 0)
  if ch&0x10==0x10:
    gpio.output(D4, 1)
  if ch&0x20==0x20:
    gpio.output(D5, 1)
  if ch&0x40==0x40:
    gpio.output(D6, 1)
  if ch&0x80==0x80:
    gpio.output(D7, 1)
  gpio.output(EN, 1)
  time.sleep(0.005)
  gpio.output(EN, 0)
  # Low bits
  gpio.output(D4, 0)
  gpio.output(D5, 0)
  gpio.output(D6, 0)
  gpio.output(D7, 0)
  if ch&0x01==0x01:
```

```
    gpio.output(D4, 1)
  if ch&0x02==0x02:
    gpio.output(D5, 1)
  if ch&0x04==0x04:
    gpio.output(D6, 1)
  if ch&0x08==0x08:
    gpio.output(D7, 1)
  gpio.output(EN, 1)
  time.sleep(0.005)
  gpio.output(EN, 0)

def lcdwrite(ch):
  gpio.output(RS, 1)
  gpio.output(D4, 0)
  gpio.output(D5, 0)
  gpio.output(D6, 0)
  gpio.output(D7, 0)
  if ch&0x10==0x10:
    gpio.output(D4, 1)
  if ch&0x20==0x20:
    gpio.output(D5, 1)
  if ch&0x40==0x40:
    gpio.output(D6, 1)
  if ch&0x80==0x80:
    gpio.output(D7, 1)
  gpio.output(EN, 1)
  time.sleep(0.005)
  gpio.output(EN, 0)
  # Low bits
  gpio.output(D4, 0)
  gpio.output(D5, 0)
  gpio.output(D6, 0)
  gpio.output(D7, 0)
  if ch&0x01==0x01:
    gpio.output(D4, 1)
  if ch&0x02==0x02:
    gpio.output(D5, 1)
  if ch&0x04==0x04:
    gpio.output(D6, 1)
  if ch&0x08==0x08:
    gpio.output(D7, 1)
  gpio.output(EN, 1)
  time.sleep(0.005)
  gpio.output(EN, 0)
def lcdclear():
```

```
  lcdcmd(0x01)

def lcdprint(Str):
  l=0;
  l=len(Str)
  for i in range(l):
    lcdwrite(ord(Str[i]))

def setCursor(x,y):
    if y == 0:
        n=128+x
    elif y == 1:
        n=192+x
    lcdcmd(n)
def enrollFinger():
    lcdcmd(1)
    lcdprint("Enrolling Finger")
    time.sleep(2)
    print('Waiting for finger...')
    lcdcmd(1)
    lcdprint("Place Finger")
    while ( f.readImage() == False ):
        pass
    f.convertImage(0x01)
    result = f.searchTemplate()
    positionNumber = result[0]
    if ( positionNumber >= 0 ):
        print('Template already exists at position #' + str(positionNumber))
        lcdcmd(1)
        lcdprint("Finger ALready")
        lcdcmd(192)
        lcdprint("   Exists     ")
        time.sleep(2)
        return
    print('Remove finger...')
    lcdcmd(1)
    lcdprint("Remove Finger")
    time.sleep(2)
    print('Waiting for same finger again...')
    lcdcmd(1)
    lcdprint("Place Finger")
    lcdcmd(192)
    lcdprint("   Again    ")
    while ( f.readImage() == False ):
        pass
```

```
    f.convertImage(0x02)
    if ( f.compareCharacteristics() == 0 ):
        print "Fingers do not match"
        lcdcmd(1)
        lcdprint("Finger Did not")
        lcdcmd(192)
        lcdprint("   Mactched   ")
        time.sleep(2)
        return
    f.createTemplate()
    positionNumber = f.storeTemplate()
    print('Finger enrolled successfully!')
    lcdcmd(1)
    lcdprint("Stored at Pos:")
    lcdprint(str(positionNumber))
    lcdcmd(192)
    lcdprint("successfully")
    print('New template position #' + str(positionNumber))
    time.sleep(2)

def searchFinger():
    try:
        print('Waiting for finger...')
        while( f.readImage() == False ):
            #pass
            time.sleep(.5)
            return
        f.convertImage(0x01)
        result = f.searchTemplate()
        positionNumber = result[0]
        accuracyScore = result[1]
        if positionNumber == -1 :
            print('No match found!')
            lcdcmd(1)
            lcdprint("No Match Found")
            time.sleep(2)
            return
        else:
            print('Found template at position #' + str(positionNumber))
            lcdcmd(1)
            lcdprint("Found at Pos:")
            lcdprint(str(positionNumber))
            time.sleep(2)

    except Exception as e:
        print('Operation failed!')
        print('Exception message: ' + str(e))
```

```
        exit(1)

def deleteFinger():
    positionNumber = 0
    count=0
    lcdcmd(1)
    lcdprint("Delete Finger")
    lcdcmd(192)
    lcdprint("Position: ")
    lcdcmd(0xca)
    lcdprint(str(count))
    while gpio.input(enrol) == True:   # here enrol key means ok
        if gpio.input(inc) == False:
            count=count+1
            if count>1000:
                count=1000
            lcdcmd(0xca)
            lcdprint(str(count))
            time.sleep(0.2)
        elif gpio.input(dec) == False:
            count=count-1
            if count<0:
                count=0
            lcdcmd(0xca)
            lcdprint(str(count))
            time.sleep(0.2)
    positionNumber=count
    if f.deleteTemplate(positionNumber) == True :
        print('Template deleted!')
        lcdcmd(1)
        lcdprint("Finger Deleted");
        time.sleep(2)

begin()
lcdcmd(0x01)
lcdprint("FingerPrint ")
lcdcmd(0xc0)
lcdprint("Interfacing ")
time.sleep(3)
lcdcmd(0x01)
lcdprint("Circuit Digest")
lcdcmd(0xc0)
lcdprint("Welcomes You  ")
time.sleep(3)
flag=0
lcdclear()
```

```
while 1:
    gpio.output(led, HIGH)
    lcdcmd(1)
    lcdprint("Place Finger")
    if gpio.input(enrol) == 0:
        gpio.output(led, LOW)
        enrollFinger()
    elif gpio.input(delet) == 0:
        gpio.output(led, LOW)
        while gpio.input(delet) == 0:
            time.sleep(0.1)
        deleteFinger()
    else:
        searchFinger()
```

7 **Raspberry Pi GPS Module Interfacing.**

Required Components:

1. Raspberry Pi 3
2. Neo 6m v2 GPS Module
3. 16 x 2 LCD
4. Power source for the Raspberry Pi
5. LAN cable to connect the pi to your PC in headless mode
6. Breadboard and Jumper cables
7. Resistor / potentiometer to the LCD
8. Memory card 8 or 16Gb running Raspbian Jessie

GPS Module and Its Working:

GPS stands for Global Positioning System and used to detect the Latitude and Longitude of any location on the Earth, with exact UTC time (Universal Time Coordinated). GPS module is the main component in our vehicle tracking system project. This device receives the coordinates from the satellite for each and every second, with time and date.

GPS module sends the data related to tracking position in real time, and it sends so many data in NMEA format (see the screenshot below). NMEA format consist several sentences, in which we only need one sentence. This sentence starts from **$GPGGA** and contains the coordinates, time and other useful information. This **GPGGA** is referred to **Global Positioning System Fix Data**. Know more about Reading GPS data and its strings here.

We can extract coordinate from $GPGGA string by counting the commas in the string. Suppose you find $GPGGA string and stores it in an array, then Latitude can be found

after two commas and Longitude can be found after four commas. Now these latitude and longitude can be put in other arrays.

Identifier	Description
$GPGGA	Global Positioning system fix da
HHMMSS.SSS	Time in hour minute se milliseconds format.
Latitude	Latitude (Coordinate)
N	Direction N=North, S=South
Longitude	Longitude(Coordinate)
E	Direction E= East, W=West
FQ	Fix Quality Data
NOS	No. of Satellites being Used
HPD	Horizontal Dilution of Precision
Altitude	Altitude from sea level
M	Meter
Height	Height
Checksum	Checksum Data

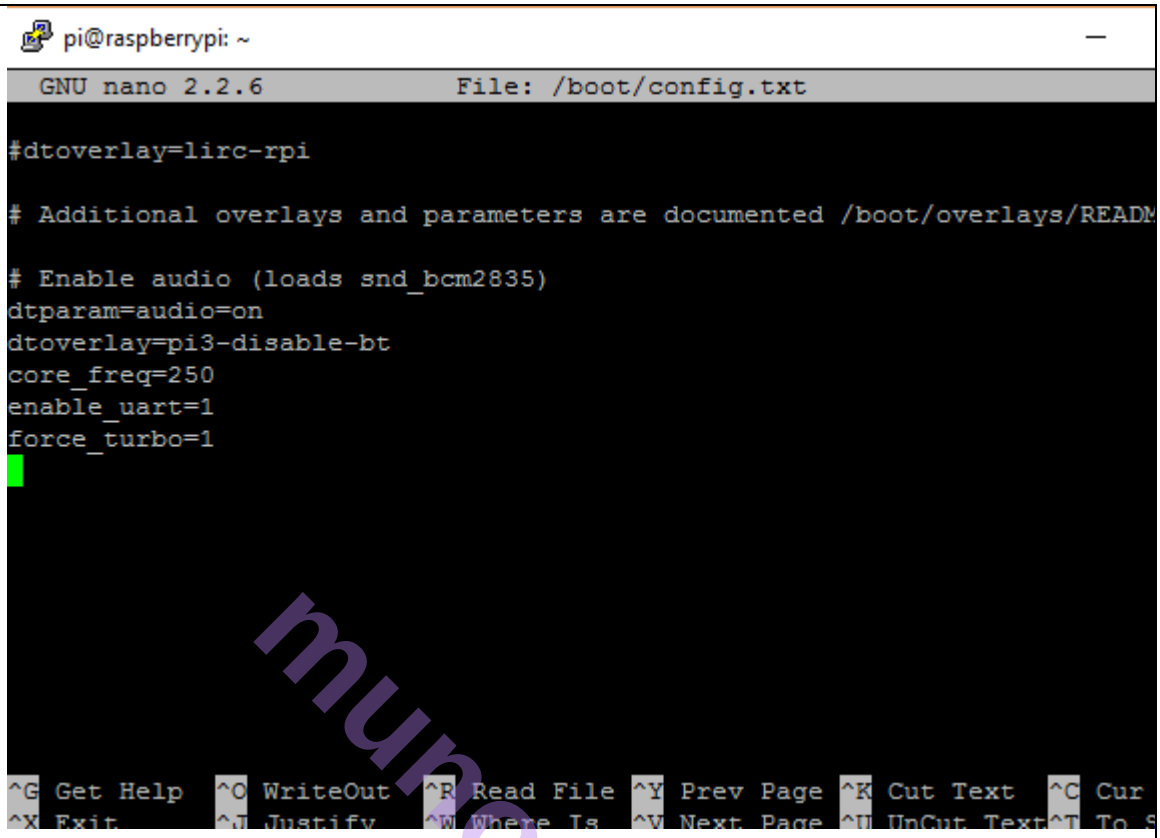
Preparing the Raspberry Pi to communicate with GPS:

Step 1: Updating the Raspberry Pi:

The first thing I like I like to do before starting every project is updating the raspberry pi. So lets do the usual and run the commands below;

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

	then reboot the system with;
	sudo reboot
	Step 2: Setting up the UART in Raspberry Pi: The first thing we will do under this is to edit the */boot/config.txt* file. To do this, run the commands below:
	sudo nano /boot/config.txt
	at the bottom of the config.txt file, add the following lines
	dtparam=spi=on dtoverlay=pi3-disable-bt core_freq=250 enable_uart=1 force_turbo=1
	ctrl+x to exit and press y and enter to save.



Ensure there are no typos or errors by double checking as an error with this might prevent your pi from booting.

What are the reasons for these commands, ***force_turbo*** enables UART to use the maximum core frequency which we are setting in this case to be 250. The reason for this is to ensure consistency and integrity of the serial data been received. Its important to note at this point that using ***force_turbo=1*** will void the warranty of your raspberry pi, but asides that, its pretty safe.

The ***dtoverlay=pi3-disable-bt*** disconnects the bluetooth from the *ttyAMA0*, this is to allow us access to use the full UART power available via *ttyAMAO* instead of the mini UART ttyS0.

Second step under this UART setup section is to edit the *boot/cmdline.txt*

I will suggest you make a copy of the cmdline.txt and save first before editing so you can revert back to it later if needed. This can be done using;

```
sudo cp boot/cmdline.txt boot/cmdline_backup.txt

sudo nano /boot.cmdline.txt
```

Replace the content with;

```
dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait quiet splash plymouth.ignore-serial-consoles
```

Save and exit.

With this done then we will need to reboot the system again to effect changes (*sudo reboot*).

Step3: Disabling the Raspberry Pi Serial Getty Service

The next step is to disable the Pi's serial the *getty service*, the command will prevent it from starting again at reboot:

```
sudo systemctl stop serial-getty@ttyS0.service
sudo systemctl disable serial-getty@ttyS0.service
```

The following commands can be used to enable it again if needed

```
sudo systemctl enable serial-getty@ttyS0.service
sudo systemctl start serial-getty@ttyS0.service
```

Reboot the system.

Step 4: Activating ttyAMAO:

We have disabled the ttyS0, next thing is for us to enable the *ttyAMAO*.

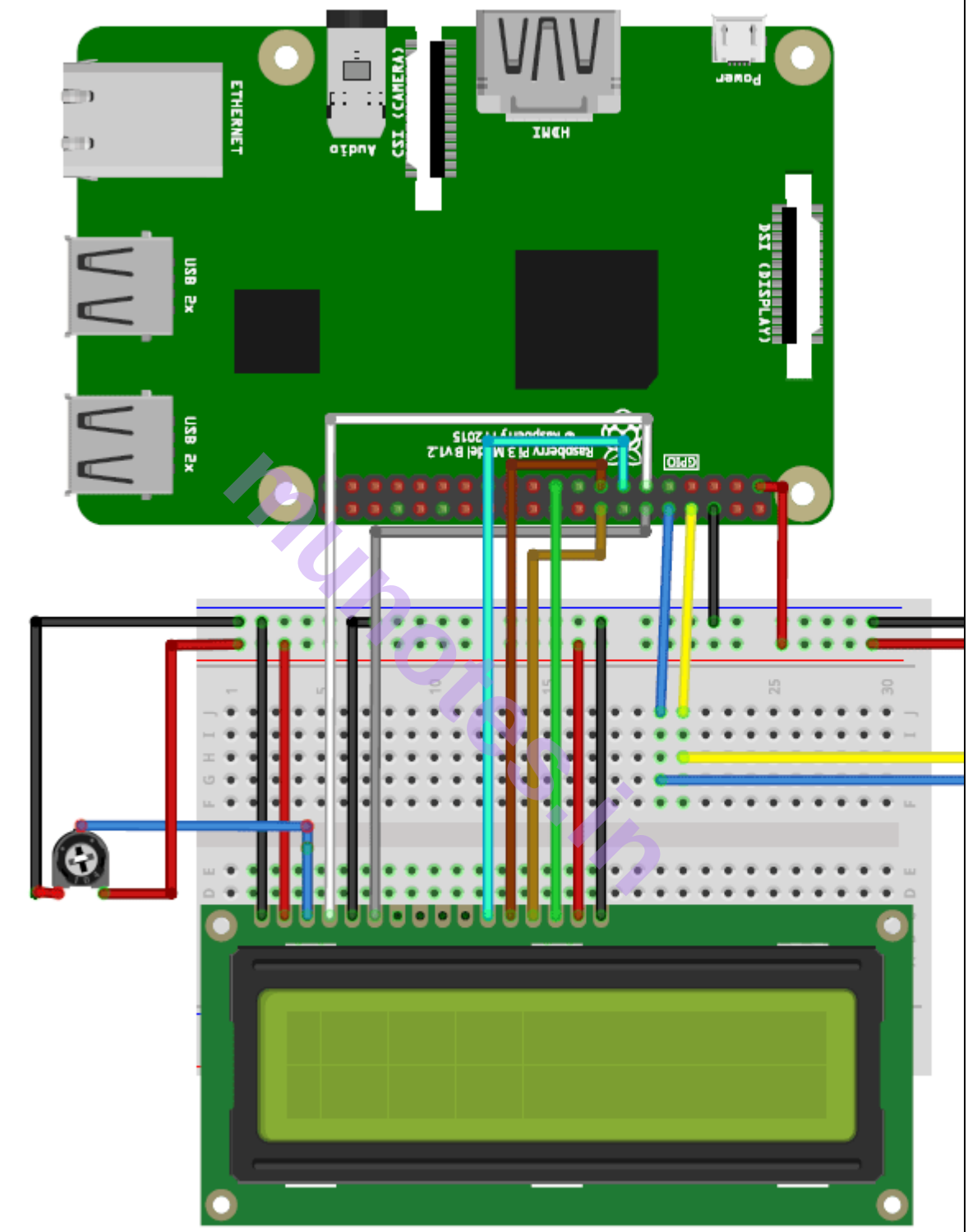
```
sudo systemctl enable serial-getty@ttyAMA0.service
```

Step5: Install Minicom and pynmea2:

We will be minicom to connect to the GPS module and make sense of the data. It is also one of the tools that we will use to test is our GPS module is working fine. An alternative to minicom is the daemon software GPSD.

```
sudo apt-get install minicom
```

	To easily parse the received data, we will make use of the ***pynmea2 library***. It can be installed using;
	sudo pip install pynmea2
	Step 6: Installing the LCD Library: For this tutorial we will be using the AdaFruit library. The library was made for AdaFruit screens but also works for display boards using HD44780. If your display is based on this then it should work without issues. I feel its better to clone the library and just install directly. To clone run;
	git clone https://github.com/adafruit/Adafruit_Python_CharLCD.git
	change into the cloned directory and install it
	cd ./Adafruit_Python_CharLCD sudo python setup.py install
	Connections for Raspberry Pi GPS module Interfacing: Connect the GPS Module and LCD to the Raspberry Pi as shown in the Circuit Diagram below.


ETHERNET
Audio
CSI (CAMERA)
HDMI
Power
USB 2x
USB 2x
DSI (DISPLAY)
GPIO

Testing before Python Script:

Its important to test the GPS module connection before proceeding to the python script, We will employ minicom for this. Run the command:

```
sudo minicom -D/dev/ttyAMA0 -b9600
```

where 9600 represents the baud rate at which the GPS module communicates. This may be used for once we are sure of data communication between the GPS and the RPI, its time to write our python script.

The test can also be done using cat

```
sudo cat /dev/ttyAMA0
```

```
pi@raspberrypi:~ $ clear
pi@raspberrypi:~ $ sudo cat /dev/ttyAMA0

$GPGLL,0649.0846,N,00327.1015,E,172206.000,A,A*5B

$GPGSA,A,3,09,06,19,23,28,03,22,30,01,17,,,2.0,0.9,1.8*3B

$GPGSV,3,1,12,01,17,147,17,02,09,322,,03,23,085,17,06,50,320,31*7A

$GPGSV,3,2,12,07,74,060,,09,30,011,30,17,28,218,24,19,30,244,20*7A

$GPGSV,3,3,12,22,11,103,15,23,13,035,28,28,17,176,18,30,66,194,28*73

$GPRMC,172206.000,A,0649.0846,N,00327.1015,E,0.00,102.80,021017,,,A*62

$GPVTG,102.80,T,,M,0.00,N,0.00,K,A*36

$GPZDA,172206.000,02,10,2017,00,00*51

$GPTXT,01,01,01,ANTENNA OK*35

$GPGGA,172207.000,0649.0846,N,00327.1016,E,1,10,0.9,57.6,M,0.0,M,,*50

$GPGLL,0649.0846,N,00327.1016,E,172207.000,A,A*59
```


In Window, you can see **NMEA sentences** which we have discussed earlier.

Code:

```
import time
import serial
import string
import pynmea2
import RPi GPIO as gpio
```

```
#to add the LCD library
import Adafruit_CharLCD as LCD

gpio.setmode(gpio.BCM)

#declaring LCD pins
lcd_rs = 17
lcd_en = 18
lcd_d4 = 27
lcd_d5 = 22
lcd_d6 = 23
lcd_d7 = 10

lcd_backlight = 2

lcd_columns = 16 #Lcd column
lcd_rows = 2 #number of LCD rows

lcd = LCD.Adafruit_CharLCD(lcd = LCD.Adafruit_CharLCD(lcd_rs, lcd_en, lcd_d4,
lcd_d5, lcd_d6, lcd_d7, lcd_columns, lcd_rows, lcd_backlight)

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()
   except:
print("loading")
#wait for the serial port to churn out data

   if data[0:6] == '$GPGGA': # the long and lat data are always contained in the GPGGA
string of the NMEA data

      msg = pynmea2.parse(data)

#parse the latitude and print
      latval = msg.lat
concatlat = "lat:" + str(latval)
      print concatlat
lcd.set_cursor(0,0)
lcd.message(concatlat)
```

```
#parse the longitude and print
longval = msg.lon
concatlong = "long:"+ str(longval)
print concatlong
lcd.set_cursor(0,1)
lcd.message(concatlong)

   time.sleep(0.5)#wait a little before picking the next data.
```

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

Required Components:

For this project, the requirements will fall under two categories, Hardware and Software:

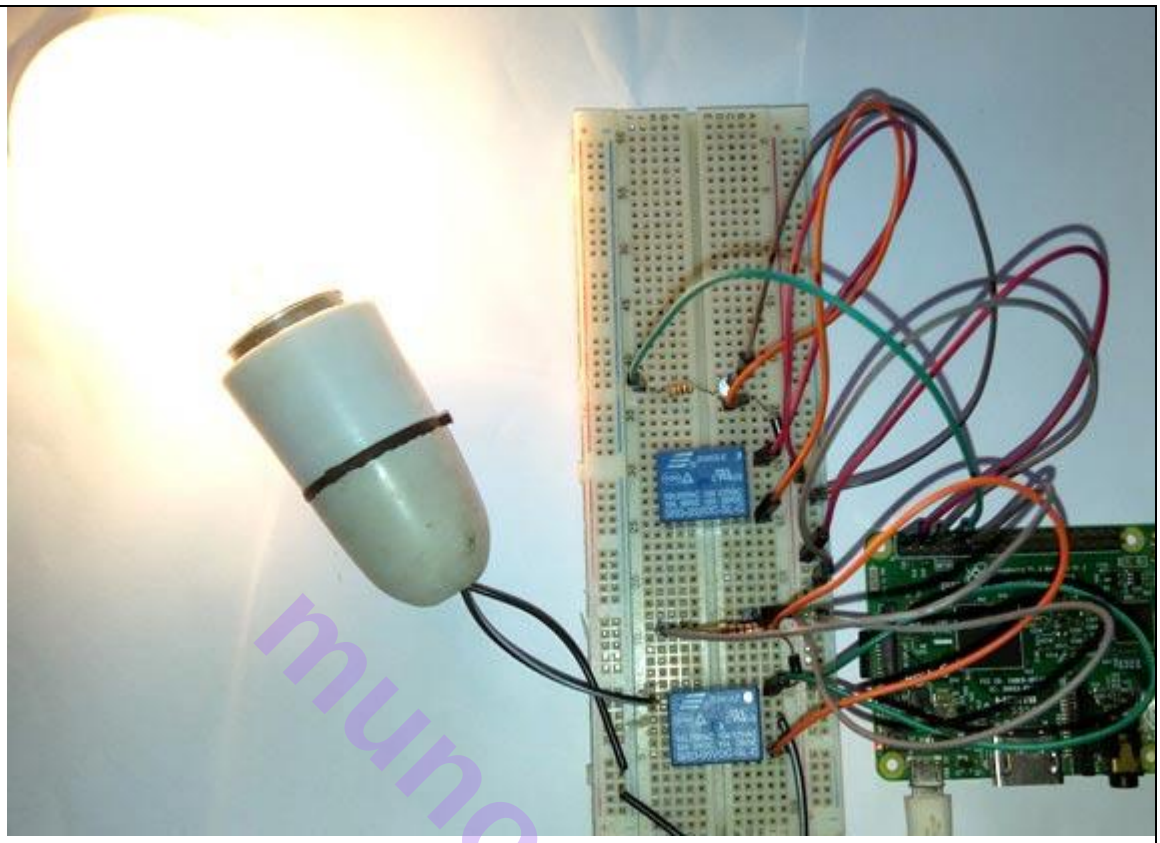
I. Hardware Requirements:

1. Raspberry Pi 3 (Any other Version will be nice)
2. Memory card 8 or 16GB running Raspbian Jessie
3. 5v Relays
4. 2n222 transistors
5. Diodes
6. Jumper Wires
7. Connection Blocks
8. LEDs to test.
9. AC lamp to Test
10. Breadboard and jumper cables
11. 220 or 100 ohms resistor

II. Software Requirements:

Asides the Raspbian Jessie operating system running on the raspberry pi, we will also be using the **WebIOPi** frame work, **notepad++** running on your PC and **filezila** to copy files from the PC to the raspberry pi, especially the web app files.

Also you **dont need to code in Python** for this Home Automation Project, WebIOPi will do all the work.



Preparing the Raspberry Pi:

To **update the raspberry Pi** below commands and then reboot the RPi;

```
sudo apt-get update
sudo apt-get upgrade
sudo reboot
```

With this done, the next thing is for us to **install the webIOPi framework**.
Make sure you are in home directory using;

```
cd ~
```

Use wget to get the file from their sourceforge page;

```
wget http://sourceforge.net/projects/webiopi/files/WebIOPi-0.7.1.tar.gz
```

When download is done, extract the file and go into the directory;

```
tar xvzf WebIOPi-0.7.1.tar.gz
cd WebIOPi-0.7.1/
```

At this point before running the setup, we need to **install a patch as this version of the WebIOPi**does not work with the raspberry pi 3 which I am using and I couldn't find a version of the WebIOPi that works expressly with the Pi 3.

Below commands are used to install patch while still in the WebIOPi directory, run;

```
wget https://raw.githubusercontent.com/doublebind/raspi/master/webiopi-pi2bplus.patch
patch -p1 -i webiopi-pi2bplus.patch
```

Then we can run the setup installation for the WebIOPi using;

```
sudo ./setup.sh
```

Keep saying yes if asked to install any dependencies during setup installation. When done, reboot your pi;

```
sudo reboot
```

Test WebIOPi Installation:

Before jumping in to schematics and codes, With the Raspberry Pi back on, we will need to test our WebIOPi installation to be sure everything works fine as desired.

Run the command;

```
sudo webiopi -d -c /etc/webiopi/config
```

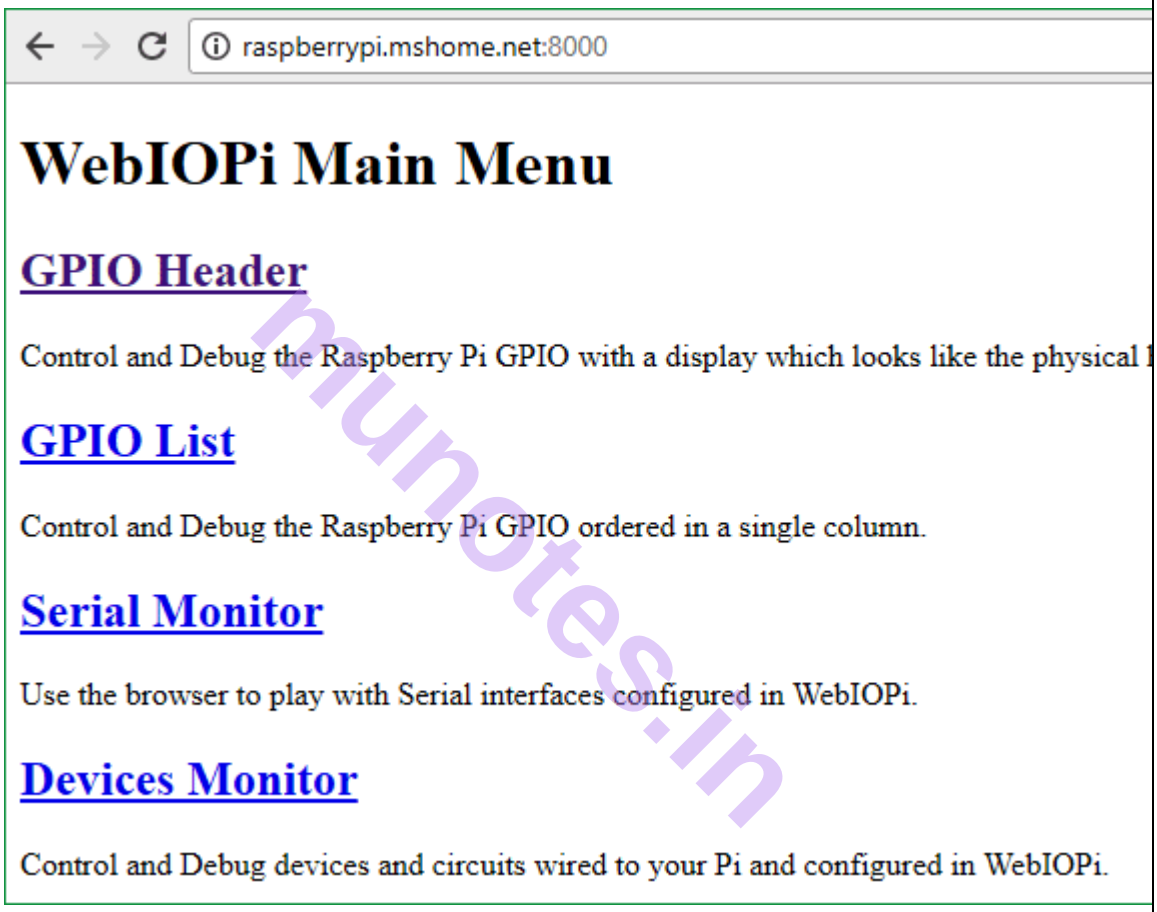
After issuing the command above on the pi, point the web browser of your computer connected to the raspberry pi to **http://raspberrypi.mshome.net:8000 or http;//thepi'sIPaddress:8000**. The system will prompt you for username and password.

Username is *webiopi*

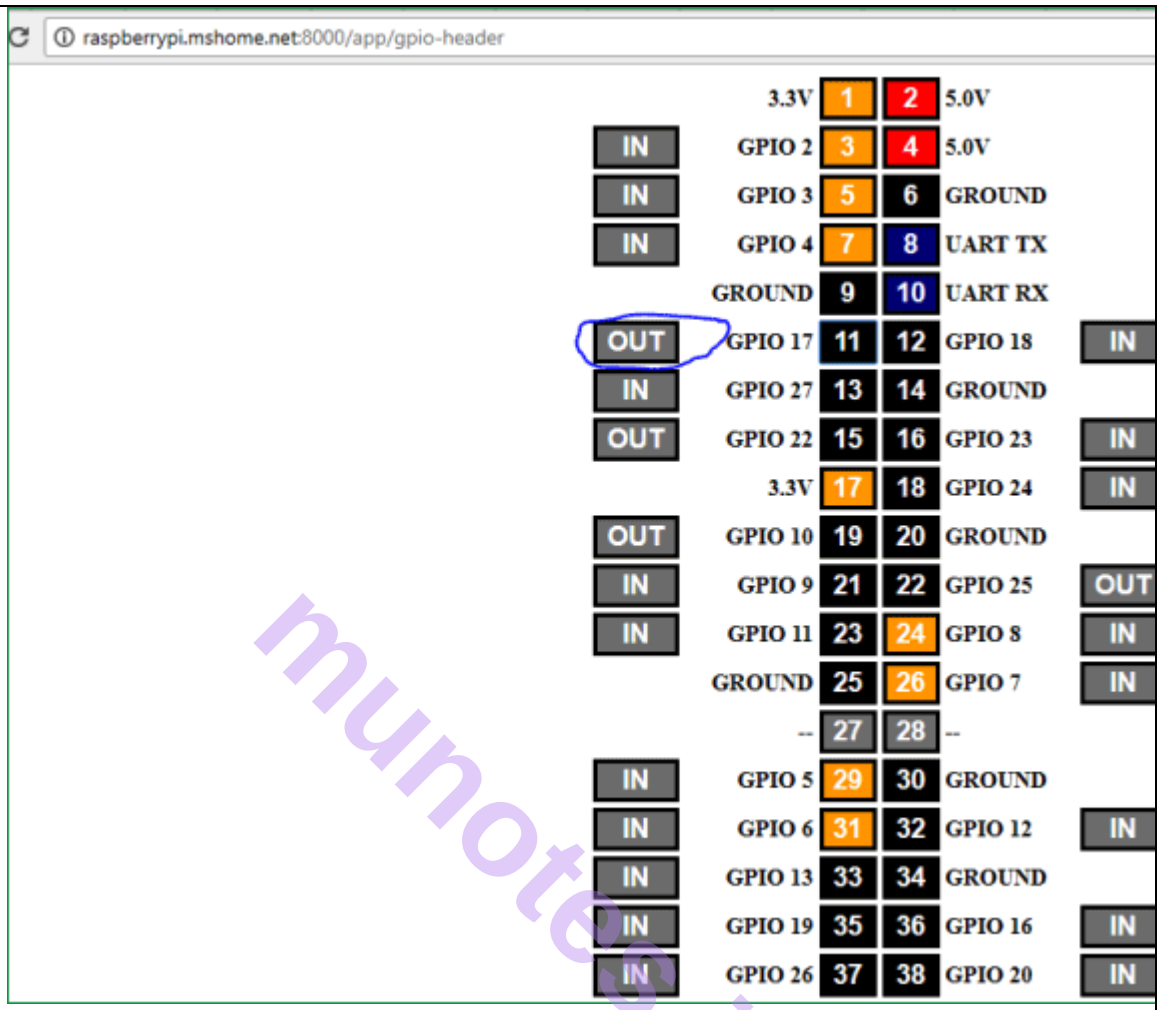
Password is *raspberry*

This login can be removed later if desired but even your home automation system deserves some extra level of security to prevent just anyone with the IP controlling appliances and IOT devices in your home.

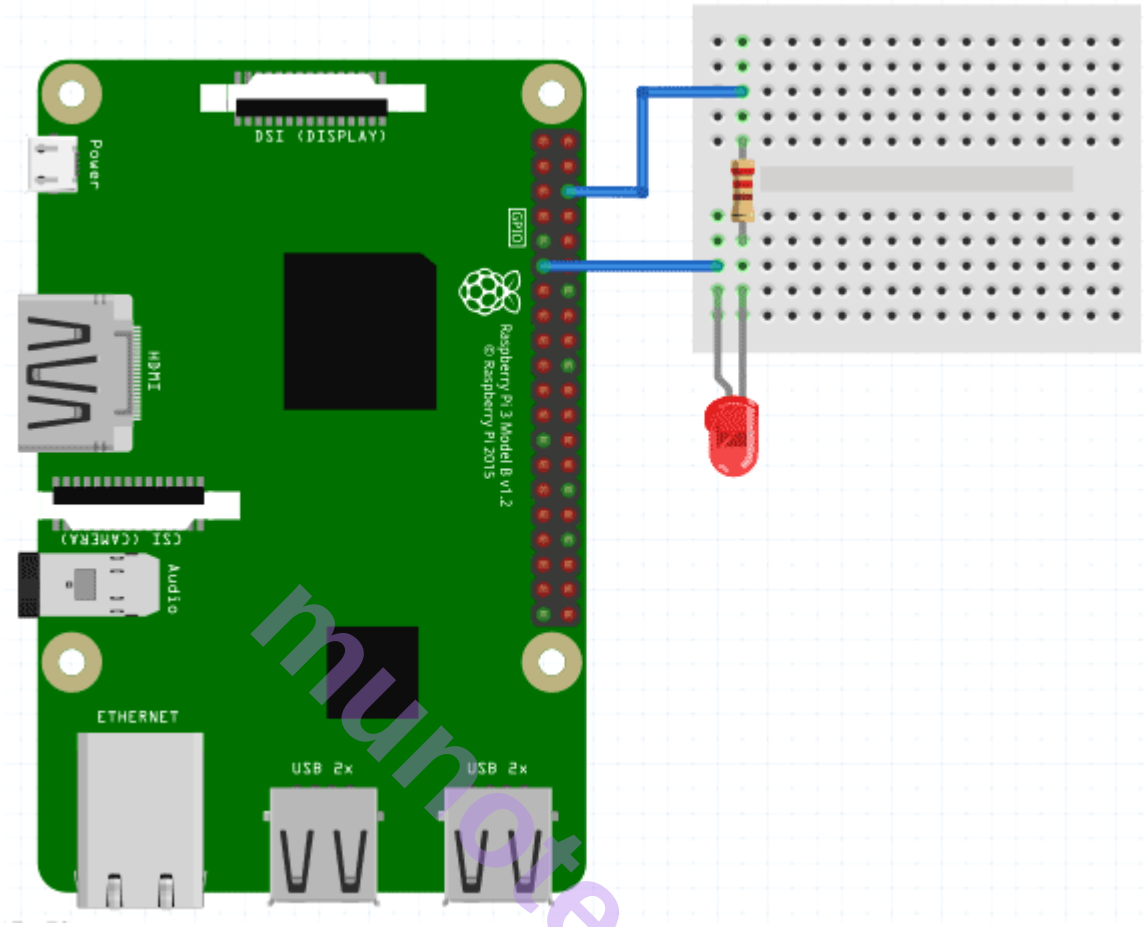
After the login, look around, and then **click on the GPIO header link**.



For this test, we will be connecting an LED to GPIO 17, so go on and set GPIO 17 as an output.



With this done, connect the led to your raspberry pi as shown in the schematics below.



After the connection, go back to the webpage and **click the pin 11 button to turn on or off the LED**. This way we can control the Raspberry Pi GPIO using *WebIOPi.*

After the test, if everything worked as described, then we can go back to the terminal and stop the program with CTRL + C. If you have any issue with this setup, then hit me up via the comment section.

Building the Web Application for Raspberry Pi Home Automation:

Here we will be editing the default configuration of the WebIOPi service and add our own code to be run when called. The first thing we will do is get ***filezilla*** or anyother FTP/SCP copy software installed on our PC. You will agree with me that coding on the pi via the terminal is quite stressful, so filezilla or any other SCP software will come in handy at this stage. Before we start writing the html, css and java script codes for this **IoT Home automation Web application** and moving them to the Raspberry Pi, lets create the project folder where all our web files will be.

Make sure you are in home directory using, then create the folder, go into the created folder and create html folder in the directory:

cd ~

mkdir webapp

cd webapp

mkdir html

Create a folder for scripts, CSS and images inside the html folder

mkdir html/css

mkdir html/img

mkdir html/scripts

```
pi@raspberrypi:~/projects $ cd ~
pi@raspberrypi:~ $ mkdir webapp
pi@raspberrypi:~ $ cd webapp
pi@raspberrypi:~/webapp $ mkdir html
pi@raspberrypi:~/webapp $ mkdir html/css
pi@raspberrypi:~/webapp $ mkdir html/img
pi@raspberrypi:~/webapp $ mkdir html/scripts
pi@raspberrypi:~/webapp $ cd html
pi@raspberrypi:~/webapp/html $ ls
css  img  scripts
pi@raspberrypi:~/webapp/html $
```


With our files created lets move to writing the codes on our PC and from then move to the Pi via filezilla.

The JavaScript Code:

The first piece of code we will write is that of the javascript. Its a simple script to communicate with the WebIOPi service.

Its important to note that for this project, our web app will consist of four buttons, which means we plan to control just four GPIO pins, although we will be controlling just two relays in our demonstration. Check the **full Video** at the end.

```
webiopi().ready(function() {
        webiopi().setFunction(17,"out");
        webiopi().setFunction(18,"out");
```

```
webiopi().setFunction(22,"out");
webiopi().setFunction(23,"out");
        var content, button;
content = $("#content");
        button = webiopi().createGPIOButton(17," Relay 1");
content.append(button);
        button = webiopi().createGPIOButton(18,"Relay 2");
content.append(button);
        button = webiopi().createGPIOButton(22,"Relay 3");
content.append(button);
        button = webiopi().createGPIOButton(23,"Relay 4");
content.append(button);
    });
```

The code above runs when the WebIOPi is ready.

Below we have explained the JavaScript code:

webiopi().ready(function(): This just instructs our system to create this function and run it when the webiopi is ready.

webiopi().setFunction(23,"out"); This helps us tell the WebIOPi service to set GPIO23 as output. We Have four buttons here, you could have more of it if you are implementing more buttons.

var content, button; This line tells our system to create a variable named content and make the variable a button.

content = $("#content"); The content variable is still going to be used across our html and css. So when we refer to #content, the WebIOPi framework creates everything associated with it.

button = webiopi().createGPIOButton(17,"Relay 1"); WebIOPi can create different kinds of buttons. The piece of code above helps us to tell the WebIOPi service to create a GPIO button that controls the GPIO pin in this case 17 labeled “Relay 1”. Same goes for the other ones.

content.append(button); Append this code to any other code for the button created either in the html file or elsewhere. More buttons can be created and will all have the same properties of this button. This is especially useful when writing the CSS or Script.

The CSS Code:

The first part of the script represent the stylesheet for the body of the web app and its shown below;

```
body {
    background-color:#ffffff;
    background-image:url('/img/smart.png');
    background-repeat:no-repeat;
    background-position:center;
    background-size:cover;
    font: bold 18px/25px Arial, sans-serif;
    color:LightGray;
  }
```

I want to believe the code above is self-explanatory, we set the background color as #ffffff which is white, then we add a background image located at that folder location (Remember our earlier folder setup?), we ensure the image doesn't repeat by setting the background-repeat property to no-repeat, then instruct the CSS to centralize the background. We then move to set the background size, font and color.

With the body done, we written the css for **buttons** to look pretty.

```
button {
    display: block;
    position: relative;
    margin: 10px;
    padding: 0 10px;
    text-align: center;
    text-decoration: none;
    width: 130px;
    height: 40px;
    font: bold 18px/25px Arial, sans-serif;  color: black;
```

```
text-shadow: 1px 1px 1px rgba(255,255,255, .22);
-webkit-border-radius: 30px;
-moz-border-radius: 30px;
border-radius: 30px;
```

To keep this brief, every other thing in the code was also done to make it look good. You can change them up see what happens, I think its called learning via trial and error but one good thing about CSS is things being expressed in plain English which means they are pretty easy to understand. The other part of the button block has few extras for text shadow on the button and button shadow. It also has a slight transition effect which helps it look nice and realistic when the button is pressed. These are defined separately for webkit, firefox, opera etc just to ensure the web page performs optimally across all platforms.

The next block of code is for the **WebIOPi service** to tell it that this is an input to the WebIOPi service.

```
input[type="range"] {
        display: block;
        width: 160px;
        height: 45px;
}
```

The last thing we want to do is give some sort of indication **when button has been pressed**. So you can sort of look at the screen and the color of the buttons let you know the current state. To do this, the line of code below was implemented for every single button.

```
#gpio17.LOW {
        background-color: Gray;
        color: Black;
}
#gpio17.HIGH {
```

```
                background-color: Red;
                color: LightGray;
        }
```

The lines of codes above simply changes the color of the button based on its current state. When the button is off(LOW) the buttons background color becomes gray to show its inactive and when its on(HIGH) the background color of the button becomes RED.

CSS in the bag, lets save as smarthome.css, then move it via filezilla(or anyother SCP software you want to use) to the styles folder on our raspberry pi and fix the final piece, the html code. Remember to download full CSS from here.

HTML Code:

The html code pulls everything together, javascript and the style sheet.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta name="mobile-web-app-capable" content="yes">
    <meta name="viewport" content = "height = device-height, width = device-width, user-scalable = no" />
    <title>Smart Home</title>
    <script type="text/javascript" src="/webiopi.js"></script>
    <script type="text/javascript" src="/scripts/smarthome.js"></script>
    <link rel="stylesheet" type="text/css" href="/styles/smarthome.css">
    <link rel="shortcut icon" sizes="196x196" href="/img/smart.png" />
</head>
<body>
        </br>
        </br>
```

```
<div id="content" align="center"></div>
</br>
</br>
</br>
<p align="center">Push button; receive bacon</p>
</br>
</br>
</body>
</html>
```

Within the **head tag** exist some very important features.

```
<meta name="mobile-web-app-capable" content="yes">
```

The line of code above enables the web app to be saved to a mobile desktop is using chrome or mobile safari. This can be done via the chrome menu. This gives the app an easy launch feel from the mobile desktop or home.

The next line of code below gives some sort of responsiveness to the web app. It enables it occupy the screen of any device on which its launched.

```
<meta name="viewport" content = "height = device-height, width = device-width, user-scalable = no" />
```

The next line of code declares the title shown on the title bar of the web page.

```
<title>Smart Home</title>
```

The next four line of codes each perform the function of linking the html code to several resources it needs to work as desired.

```
<script type="text/javascript" src="/webiopi.js"></script>
<script type="text/javascript" src="/scripts/smarthome.js"></script>
```

```
<link rel="stylesheet" type="text/css" href="/styles/smarthome.css">
<link rel="shortcut icon" sizes="196x196" href="/img/smart.png" />
```

The **first line** above calls the main WebIOPi framework JavaScript which is hard-coded in the server root. This needs to be called every time the WebIOPi is to be used.

The **second line** points the html page to our jQuery script, **the third** points it in the direction of our style sheet. Lastly the last line helps set up an icon to be used on the mobile desktop in case we decide to use it as a web app or as a favicon for the webpage.

The body section of the html code just contains break tags to ensure the buttons aligned properly with the line below telling our html code to display what is written in the JavaScript file. The *id=”content”* should remind you of the content declaration for our button earlier under the JavaScript code.

```
<div id="content" align="center"></div>
```

WebIOPi Server Edits for Home Automation:

At this stage, we are ready to start creating our schematics and test our web app but before then we need to **edit the config file of the webiopi** service so its pointed to use data from our html folder instead of the config files that came with it.

To edit the configuration run the following with root permission;

```
sudo nano /etc/webiopi/config
```

Look for the http section of the config file, check under the section where you have something like, *#Use doc-root to change default HTML and resource files location*

Comment out anything under it using # then if your folder is setup like mine, point your doc-root to the path of your project file

```
doc-root = /home/pi/webapp/html
```

Save and exit. You can also change the port from 8000, in case you have another server running on the pi using that port. If not save and quit, as we move on.

Its Important to note that you can change the password of the WebIOPi service using the command;

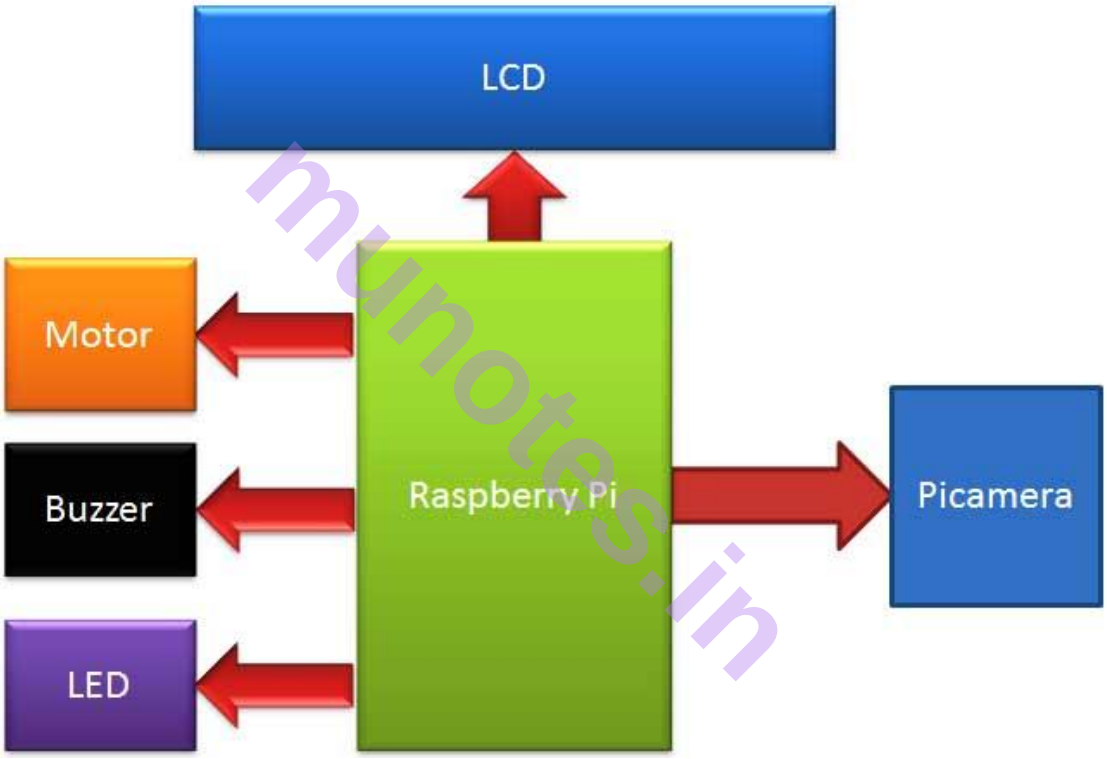
```
sudo webiopi-passwd
```

	It will prompt you for a new username and password. This can also be removed totally but security is important right? Lastly run the WebIOPi service by issuing below command: sudo /etc/init.d/webiopi start The server status can be checked using; sudo /etc/init.d/webiopi status It can be stopped from running using; sudo /etc/init.d/webiopi stop To setup WebIOPi to run at boot, use; sudo update-rc.d webiopi defaults If you want to reverse and stop it from running at boot, use; sudo update-rc.d webiopi remove
9	**Visitor Monitoring with Raspberry Pi and Pi Camera.**
	Components Required: 1. Raspberry Pi 2. Pi camera 3. 16x2 LCD 4. DC Motor 5. IC L293D 6. Buzzer 7. LED 8. Bread Board 9. Resistor (1k,10k) 10. Capacitor (100nF) 11. Push Button 12. Connecting wires 13. 10k Pot

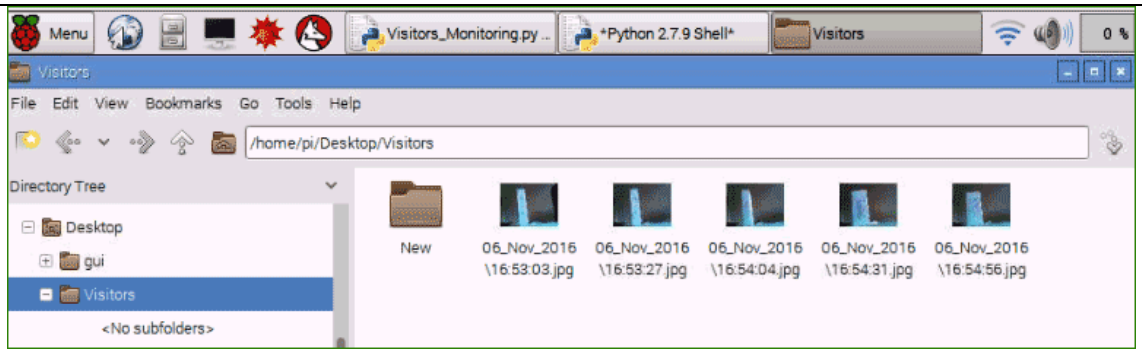
14. Power supply

Working Explanation:

Working of this **Raspberry Pi Monitoring System** is simple. In this, a **Pi camera** is used to capture the images of visitors, when a push button is pressed or triggered. A **DC motor is used as a gate**. Whenever anyone wants to enter in the place then he/she needs to push the button. After pushing the button, Raspberry Pi sends command to Pi Camera to click the picture and save it. After it, the gate is opened for a while and then gets closed again. The buzzer is used to generate sound when button pressed and LED is used for indicating that Raspberry Pi is ready to accept Push Button press, means when LED is ON, system is ready for operation.

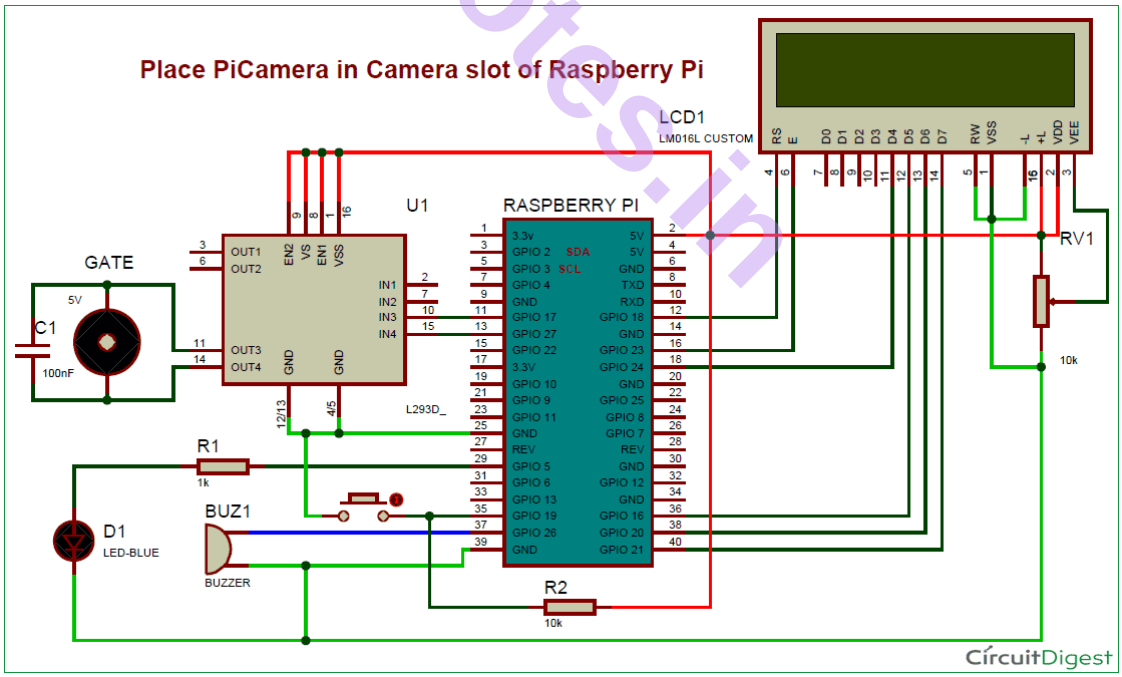


Here the pictures of visitors are saved in Raspberry Pi with the name which itself contains the time and date of entry. Means there is no need to save date and time separately at some other place as we have assigned the time and date as the name of the captured picture, see the image below. We have here taken the image of a box as visitor,



Circuit Explanation:

Circuit of this **Raspberry Pi Visitor Surveillance System** is very simple. Here a **Liquid Crystal Display**(LCD) is used for displaying Time/Date of visitor entry and some other messages. LCD is connected to Raspberry Pi in 4-bit mode. Pins of LCD namely RS, EN, D4, D5, D6, and D7 are connected to Raspberry Pi GPIO pin number 18, 23, 24, 16, 20 and 21. **Pi camera module** is connected at camera slot of the Raspberry Pi. A buzzer is connected to GPIO pin 26 of Raspberry Pi for indication purpose. LED is connected to GPIO pin 5 through a 1k resistor and **a push button** is connected to GPIO pin 19 with respect to ground, to trigger the camera and open the Gate. **DC motor (as Gate)** is connected with Raspberry Pi GPIO pin 17 and 27 through **Motor Driver IC (L293D).** Rest of connections are shown in circuit diagram.

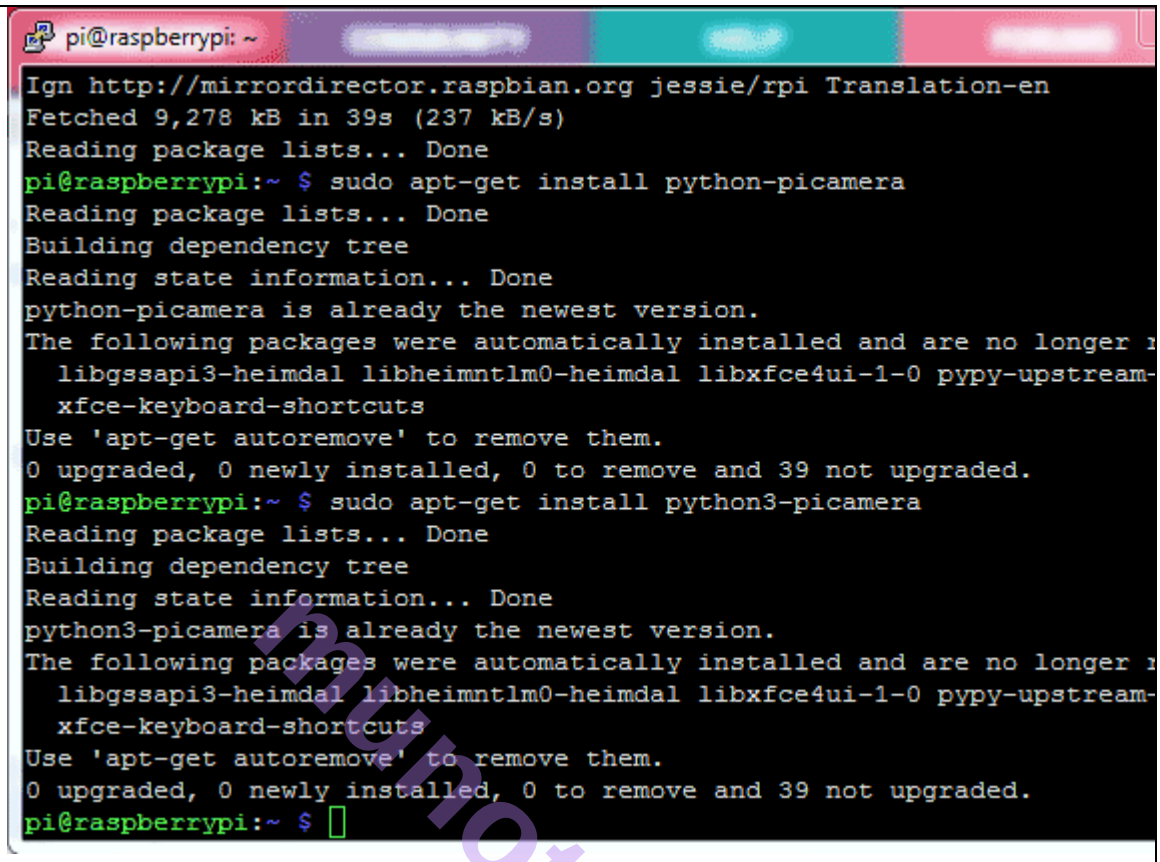


To connect the Pi Camera, insert the Ribbon cable of Pi Camera into camera slot, slightly pull up the tabs of the connector at RPi board and insert the Ribbon cable into the slot, then gently push down the tabs again to fix the ribbon cable.

Raspberry Pi Configuration and Programming Explanation:

After successfully installing Raspbian OS on Raspberry Pi, we need to **install Pi camera library files**for run this project in Raspberry pi. To do this we need to follow given commands:

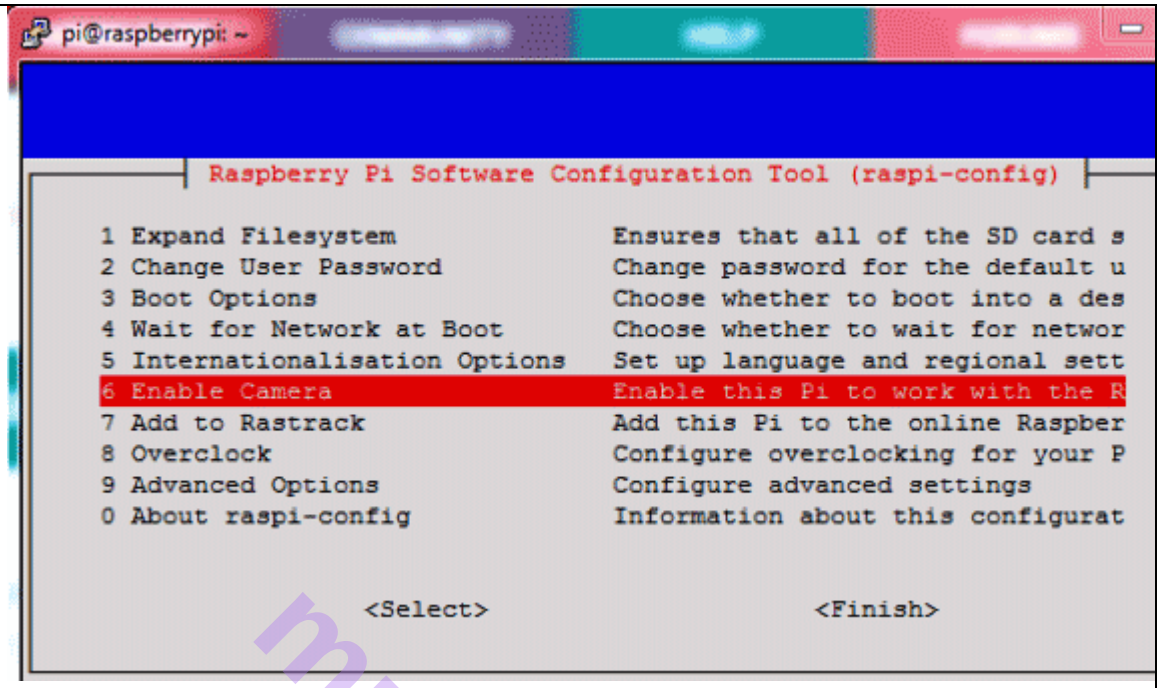
```
$ sudo apt-get install python-picamera
$ sudo apt-get install python3-picamera
```



After it, user needs to **enable Raspberry Pi Camera** by using Raspberry Pi Software Configuration Tool (raspi-config):

```
$ sudo raspi-config
```

Then select **Enable camera** and Enable it.



Then user needs to **reboot Raspberry Pi**, by issuing *sudo reboot*, so that new setting can take. Now your Pi camera is ready to use.

```
$ sudo reboot
```

The Python Program of this project plays a very important role to perform all the operations. First of all, we include required libraries, initialize variables and define pins for LCD, LED, Motor and other components.

```
import RPi.GPIO as gpio
import picamera
import time

m11=17
m12=27
led=5
buz=26
```

```
button=19


RS =18
... ....
... .....
```

Function *def capture_image()* is created to **capture the image of visitor** with time and date.

```
def capture_image():
   lcdcmd(0x01)
   lcdprint("Please Wait..");
   data= time.strftime("%d_%b_%Y\%H:%M:%S")
   camera.start_preview()
   time.sleep(5)
   print data
   camera.capture('/home/pi/Desktop/Visitors/%s.jpg'%data)
   camera.stop_preview()
   lcdcmd(0x01)
   lcdprint("Image Captured")
   lcdcmd(0xc0)
   lcdprint(" Successfully ")
   time.sleep(2)
```

Function *def gate()* is written for **driving the DC motor** which is used as a Gate here.

```
def gate():
        lcdcmd(0x01)
        lcdprint("    Welcome  ")
        gpio.output(m11, 1)
```

```
        gpio.output(m12, 0)
        time.sleep(1.5)
        gpio.output(m11, 0)
        gpio.output(m12, 0)
        time.sleep(3)
        gpio.output(m11, 0)
        gpio.output(m12, 1)
        time.sleep(1.5)
        gpio.output(m11, 0)
        gpio.output(m12, 0)
        lcdcmd(0x01);
        lcdprint("  Thank You  ")
        time.sleep(2)
```

Some functions are defined for LCD like *def begin()* function is used to initialize LCD, *def lcdcmd(ch)* function is used for sending command to LCD, *def lcdwrite(ch)* function is used for sending data to LCD and *def lcdprint(Str)* function is used to send data string to LCD. You can check all these functions in Code given afterwards.

Then we have initialized the LCD and Pi Camera, and **continuously read the Push button** using *while*loop. Whenever the push button is pressed, to open the gate for entry, image of the visitor is captured and saved at the Raspberry pi with date & time and gate gets opened.

```
while 1:
    d= time.strftime("%d %b %Y")
    t= time.strftime("%H:%M:%S")
    lcdcmd(0x80)
    lcdprint("Time: %s"%t)
    lcdcmd(0xc0)
    lcdprint("Date:%s"%d)
```

```
    gpio.output(led, 1)
    if gpio.input(button)==0:
        gpio.output(buz, 1)
        gpio.output(led, 0)
        time.sleep(0.5)
        gpio.output(buz, 0)
        capture_image()
        gate()
    time.sleep(0.5)
```

This **Camera Monitoring System** has lot of scope to upgrade, like a software can be built in Computer Vision or in OpenCV to match the captured picture of visitor with the already stored images and only authorized the visitor if some match has been found, this will only open the gate for authorised people.

Code:

```
import RPi.GPIO as gpio
import picamera
import time

m11=17
m12=27
led=5
buz=26

button=19

RS =18
EN =23
D4 =24
D5 =16
D6 =20
D7 =21

HIGH=1
LOW=0

gpio.setwarnings(False)
gpio.setmode(gpio.BCM)
gpio.setup(RS, gpio.OUT)
gpio.setup(EN, gpio.OUT)
gpio.setup(D4, gpio.OUT)
```

```
gpio.setup(D5, gpio.OUT)
gpio.setup(D6, gpio.OUT)
gpio.setup(D7, gpio.OUT)
gpio.setup(led, gpio.OUT)
gpio.setup(buz, gpio.OUT)
gpio.setup(m11, gpio.OUT)
gpio.setup(m12, gpio.OUT)
gpio.setup(button, gpio.IN)
gpio.output(led , 0)
gpio.output(buz , 0)
gpio.output(m11 , 0)
gpio.output(m12 , 0)
data=""

def capture_image():
   lcdcmd(0x01)
   lcdprint("Please Wait..");
   data= time.strftime("%d_%b_%Y\%H:%M:%S")
   camera.start_preview()
   time.sleep(5)
   print data
   camera.capture('/home/pi/Desktop/Visitors/%s.jpg'%data)
   camera.stop_preview()
   lcdcmd(0x01)
   lcdprint("Image Captured")
   lcdcmd(0xc0)
   lcdprint(" Successfully ")
   time.sleep(2)

def gate():
        lcdcmd(0x01)
        lcdprint("    Welcome  ")
        gpio.output(m11, 1)
        gpio.output(m12, 0)
        time.sleep(1.5)
        gpio.output(m11, 0)
        gpio.output(m12, 0)
        time.sleep(3)
        gpio.output(m11, 0)
        gpio.output(m12, 1)
        time.sleep(1.5)
        gpio.output(m11, 0)
        gpio.output(m12, 0)
        lcdcmd(0x01);
        lcdprint("  Thank You  ")
        time.sleep(2)
```

```
def begin():
  lcdcmd(0x33)
  lcdcmd(0x32)
  lcdcmd(0x06)
  lcdcmd(0x0C)
  lcdcmd(0x28)
  lcdcmd(0x01)
  time.sleep(0.0005)

def lcdcmd(ch):
  gpio.output(RS, 0)
  gpio.output(D4, 0)
  gpio.output(D5, 0)
  gpio.output(D6, 0)
  gpio.output(D7, 0)
  if ch&0x10==0x10:
    gpio.output(D4, 1)
  if ch&0x20==0x20:
    gpio.output(D5, 1)
  if ch&0x40==0x40:
    gpio.output(D6, 1)
  if ch&0x80==0x80:
    gpio.output(D7, 1)
  gpio.output(EN, 1)
  time.sleep(0.005)
  gpio.output(EN, 0)
  # Low bits
  gpio.output(D4, 0)
  gpio.output(D5, 0)
  gpio.output(D6, 0)
  gpio.output(D7, 0)
  if ch&0x01==0x01:
    gpio.output(D4, 1)
  if ch&0x02==0x02:
    gpio.output(D5, 1)
  if ch&0x04==0x04:
    gpio.output(D6, 1)
  if ch&0x08==0x08:
    gpio.output(D7, 1)
  gpio.output(EN, 1)
  time.sleep(0.005)
  gpio.output(EN, 0)

def lcdwrite(ch):
  gpio.output(RS, 1)
```

```
  gpio.output(D4, 0)
  gpio.output(D5, 0)
  gpio.output(D6, 0)
  gpio.output(D7, 0)
  if ch&0x10==0x10:
    gpio.output(D4, 1)
  if ch&0x20==0x20:
    gpio.output(D5, 1)
  if ch&0x40==0x40:
    gpio.output(D6, 1)
  if ch&0x80==0x80:
    gpio.output(D7, 1)
  gpio.output(EN, 1)
  time.sleep(0.005)
  gpio.output(EN, 0)

  # Low bits
  gpio.output(D4, 0)
  gpio.output(D5, 0)
  gpio.output(D6, 0)
  gpio.output(D7, 0)
  if ch&0x01==0x01:
    gpio.output(D4, 1)
  if ch&0x02==0x02:
    gpio.output(D5, 1)
  if ch&0x04==0x04:
    gpio.output(D6, 1)
  if ch&0x08==0x08:
    gpio.output(D7, 1)
  gpio.output(EN, 1)
  time.sleep(0.005)
  gpio.output(EN, 0)

def lcdprint(Str):
  l=0;
  l=len(Str)
  for i in range(l):
    lcdwrite(ord(Str[i]))

begin()
lcdcmd(0x01)
lcdprint("Visitor Monitoring")
lcdcmd(0xc0)
lcdprint("    Using RPI     ")
time.sleep(3)
lcdcmd(0x01)
lcdprint("Circuit Digest")
```

```
lcdcmd(0xc0)
lcdprint("Saddam Khan")
time.sleep(3)
lcdcmd(0x01)
camera = picamera.PiCamera()
camera.rotation=180
camera.awb_mode= 'auto'
camera.brightness=55
lcdcmd(0x01)
lcdprint(" Please Press ")
lcdcmd(0xc0)
lcdprint("    Button      ")
time.sleep(2)
while 1:
    d= time.strftime("%d %b %Y")
    t= time.strftime("%H:%M:%S")
    lcdcmd(0x80)
    lcdprint("Time: %s"%t)
    lcdcmd(0xc0)
    lcdprint("Date:%s"%d)
    gpio.output(led, 1)
    if gpio.input(button)==0:
       gpio.output(buz, 1)
       gpio.output(led, 0)
       time.sleep(0.5)
       gpio.output(buz, 0)
       capture_image()
       gate()
    time.sleep(0.5)
```


10 **Interfacing Raspberry Pi with RFID.**

Setting up Raspberry Pi for Serial Communication

Before proceeding with the Interface of Raspberry Pi and RFID Reader Module, there are a few things you need to do in your Raspberry Pi in order to enable the Serial Communication in Raspberry Pi.

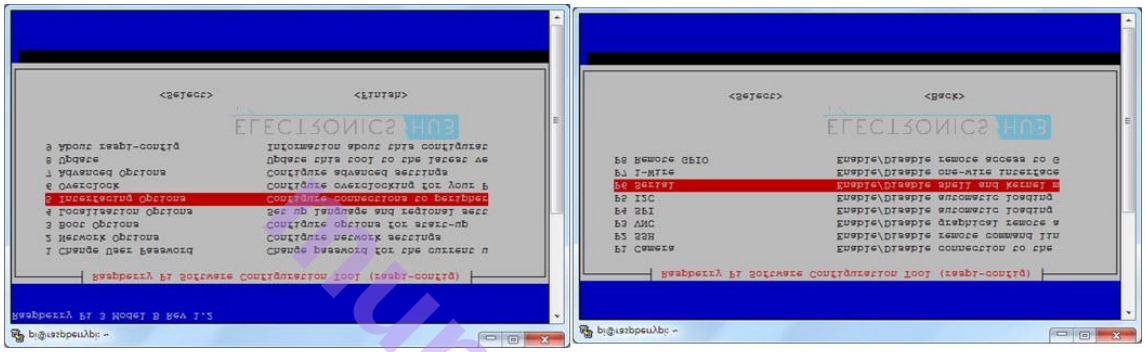
In Raspberry Pi, the Serial Port can be used or configured in two ways: Access Console and Serial Interface. By default, the Serial Port of the Raspberry Pi is configured to access the Linux Console i.e. as Console I/O pins.

But, we want to change this to act as a Serial Communication Port so that we can connected external peripherals, like RFID Reader in this project, to communicate through serial communication.

In order to do this, first login to your Raspberry Pi using SSH (Putty). Enter the Raspberry Pi Configuration Tool by entering the following command.

```
sudo raspi-config
```

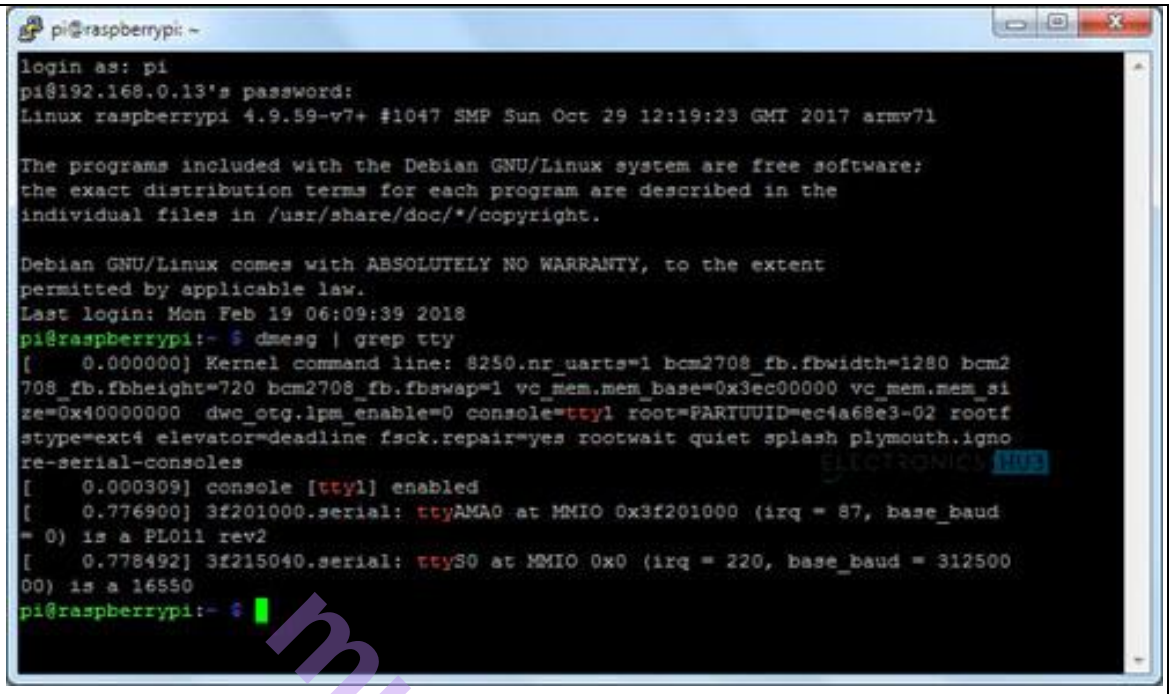
In the “Interfacing Options”, select the “Serial” option.



Now, it asks whether you would like to access the login shell over serial communication. Select “No” option. Then, it asks do you want to enable the Serial Port Hardware. Select “Yes” option.

Finish the process and Reboot the Raspberry Pi. After the Raspberry Pi is powered up, once agin login using Putty and in order to check whether the Serial Port is Enabled or not, enter the following command.

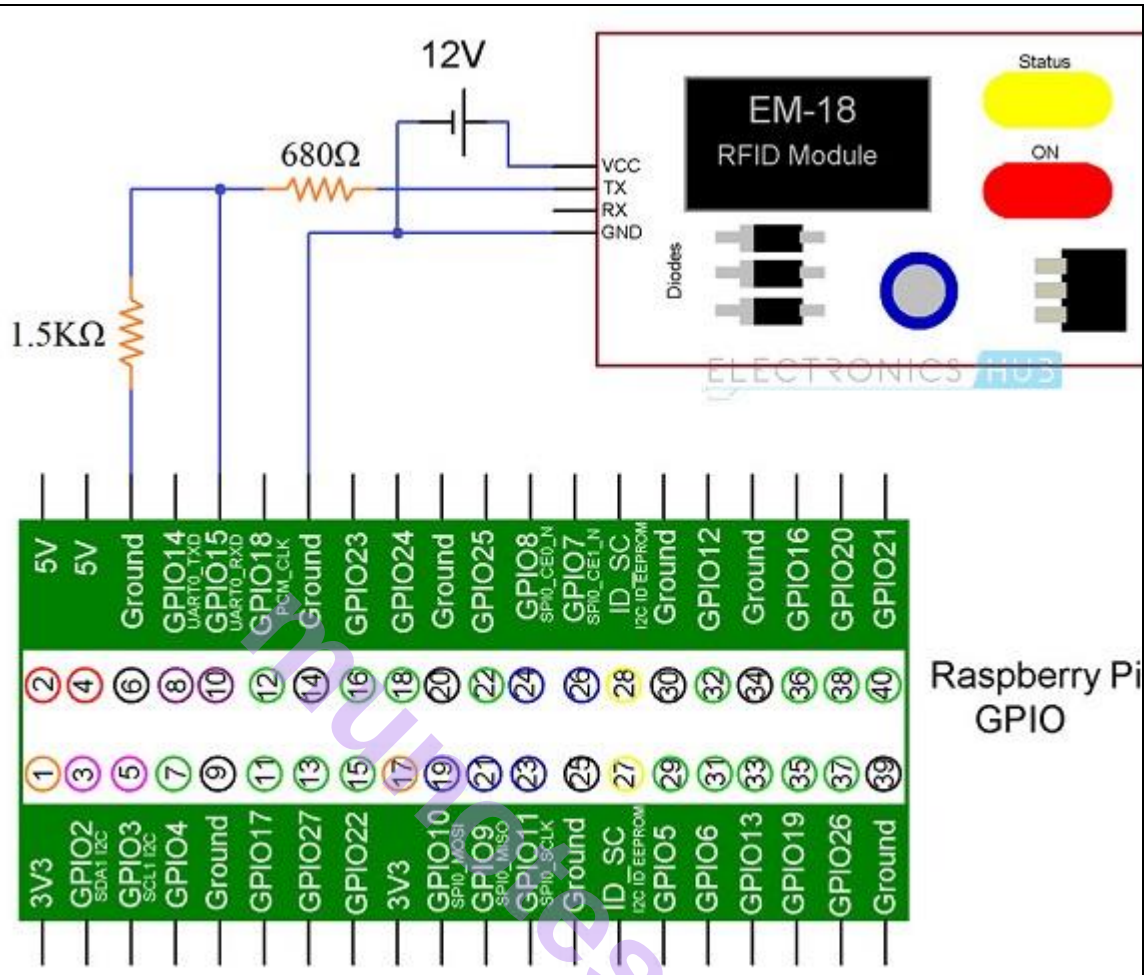
```
dmesg | grep tty
```



At the bottom, you can see, "ttyS0" is configured as Serial. Now, you can proceed with Interfacing RFID Reader Module with Raspberry Pi to communicate over Serial.

Circuit Diagram of Raspberry Pi RFID Reader Interface

The following image shows the connections between the Raspberry Pi and the EM-18 RFID Reader.



Components Required

- Raspberry Pi 3 Model B
- EM-18 RFID Reader Module
- RS232 – to – USB Adapter (as my RFID Reader has only RS232 Output)
- Few RFID Cards or RFID Tags
- Power Supply for RFID Reader (my RFID Reader has a 5V Regulator)
- 5V Supply for Raspberry Pi and RS232-USB Adapter
- Connecting Wires
- 680Ω Resistor (1/4 Watt)
- 1.5KΩ Resistor (1/4 Watt)

Circuit Design

On Raspberry Pi, the GPIO14 and GPIO14 i.e. Physical Pins 8 and 10 are the UART TX and RX Pins respectively. As we have already enabled the Serial Port of the Raspberry Pi, you can connect these pins to the external peripherals.

It is now a good time to note that Raspberry Pi works on 3.3V Logic. Hence, the RX Pin of the Raspberry Pin must only be given with 3.3V Logic. In order to do that, we need to level convert the TX line of the RFID Reader to 3.3V using a simple Voltage Divider Network consisting to two resistors.

I have used 680Ω and 1.5KΩ resistors. The output of the voltage divider is connected to the UART RXD pin of the Raspberry Pi i.e. GPIO15. Make a common ground connection between the Raspberry Pi and the RFID Reader Module.

Code

A simple Python Script is written to read the values from the RFID Card, compare it with the predefined values (I have already collected the data of all the RFID Cards beforehand) and display specific information.

```
import time
import serial



data = serial.Serial(
                port='/dev/ttyS0',
                baudrate = 9600,
                parity=serial.PARITY_NONE,
                stopbits=serial.STOPBITS_ONE,
                bytesize=serial.EIGHTBITS
                )
                #timeout=1 # must use when using data.readline()
                #)
print " "
```

```
try:
  while 1:
      #x=data.readline()#print the whole data at once
      #x=data.read()#print single data at once

      print "Place the card"
      x=data.read(12)#print upto 10 data at once and the
                  #remaining on the second line

      if x=="13004A29E191":
         print "Card No - ",x
         print "Welcome Bala"
         print " "

      elif x=="13006F8C7282":
         print "Card No - ",x
         print "Welcome Teja"
         print " "
      else:
         print "Wrong Card....."
         print " "

      #print x
```

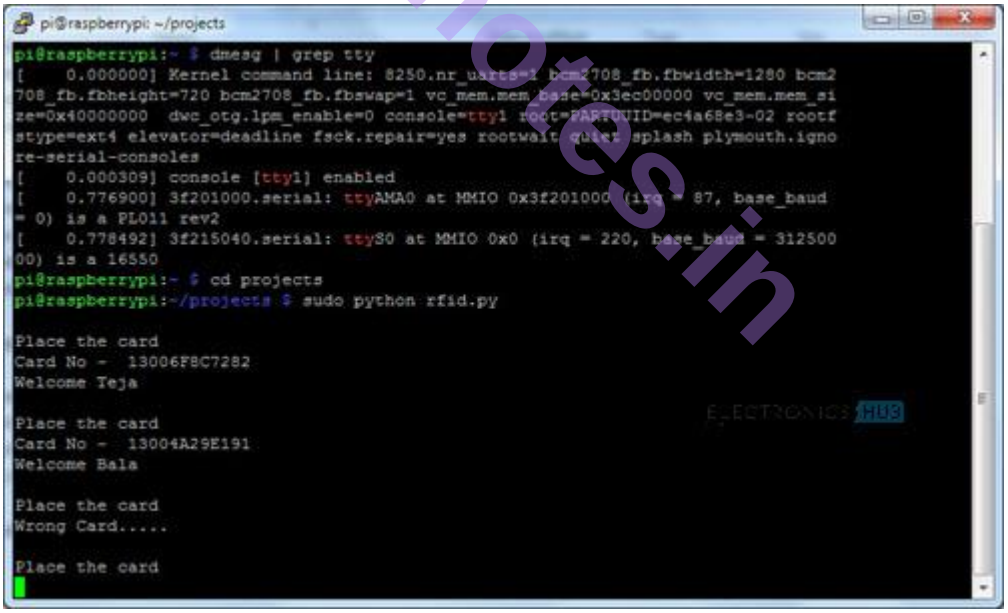
```
except KeyboardInterrupt:
    data.close()
```

Working

The Working of the Raspberry Pi RFID Reader Module Interface is very simple. After enabling the Serial Port on the Raspberry Pi, we must assign the rest of the parameters associated with UART Communication i.e. Baud Rate, Parity, Stop Bits and the Size of the Data. All these values are set in the Python code.

After this, you will get a message as “Place the Card”. When you place your RFID Card on the RFID Reader, the RFID Reader Module it will read the data from the Card and sends the data to the Raspberry Pi over Serial Communication.

This data is further analyzed by the Raspberry Pi and appropriate messages are displayed in the screen.



Applications

Interfacing RFID Reader with Raspberry Pi can be very useful as you can implement a wide range of applications like:

1. Access Control

	2. Authentication 3. e-Ticket 4. e-Payment 5. e-Toll 6. Attendance
11	**Building Google Assistant with Raspberry Pi.**
	Equipment List Recommended: 1. Raspberry Pi 2 or 3 2. Micro SD Card 3. USB Microphone 4. Speakers 5. Ethernet Network Connection or Wifi dongle (The Pi 3 has WiFi inbuilt) Optional: Raspberry Pi Case **Testing your Audio for Google Assistant** **1.** Before we get into all the hard work of setting up our Google Assistant and setting up the required API .we will first test to ensure our audio is working. At this stage, you must have your USB microphone and speakers attached to your Raspberry Pi. Once you are sure both are connected to the Raspberry Pi, we can test to make sure that the speakers are working correctly by running the following command. speaker-test -t wav You should hear sound from your speakers. This sound will be a person speaking. If you do not hear anything coming from your speaker's double check they are plugged in correctly and are turned up. **2.** Now, let's test our microphone by making a recording, to do this we will run the following command on your Raspberry Pi. This command will make a short 5-second recording. arecord --format=S16_LE --duration=5 --rate=16000 --file-type=raw out.raw If you receive an error when running this command make sure that you have your microphone plugged in, this command will only succeed if it can successfully listen to your microphone. **3.** With our recording done we can now run the following command to read in our raw output file and play it back to our speakers. Doing this will allow you to test the playback volume and also listen to the recording volume. Doing this is a crucial task as you don't want your Raspberry Pi picking up every little noise but you also don't want it being able to barely hear you when you say "**Ok Google**". aplay --format=S16_LE --rate=16000 out.raw **4.** If you find the playback volume or recording volume is either too high or too low, then you can run the following command to launch the mixer. This command will allow you to tweak the volumes for certain devices. alsamixer

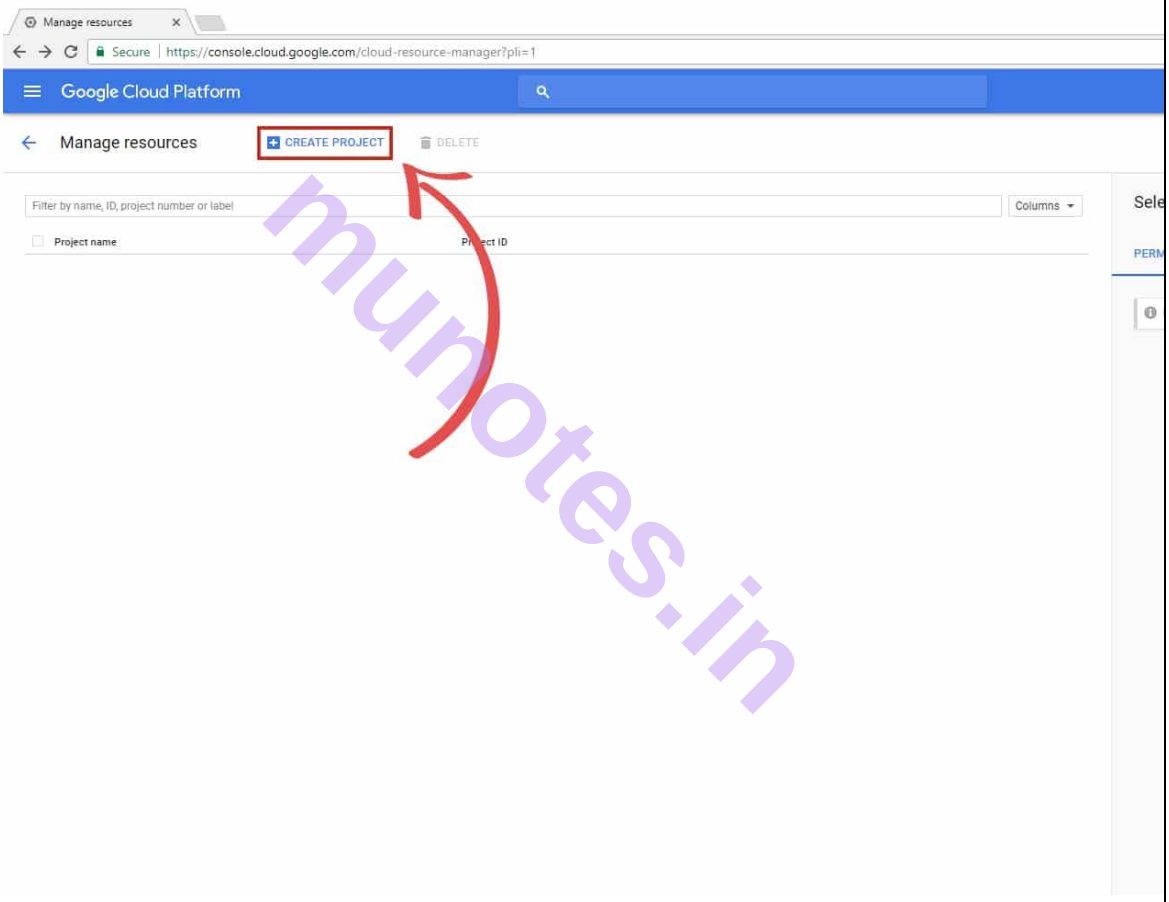
Once you have confirmed that your microphone and speakers are working correctly, you can move onto setting up your very own Raspberry Pi Google Assistant.

Registering for the Google API

1. Before we get started with setting up the Google Assistant code on the Raspberry Pi itself, we must first register and setup oAuth access to Google's Assistant API. To do this, you will need to have a Google account already.

Once you have your Google account ready, go to the following web address.
https://console.cloud.google.com/project

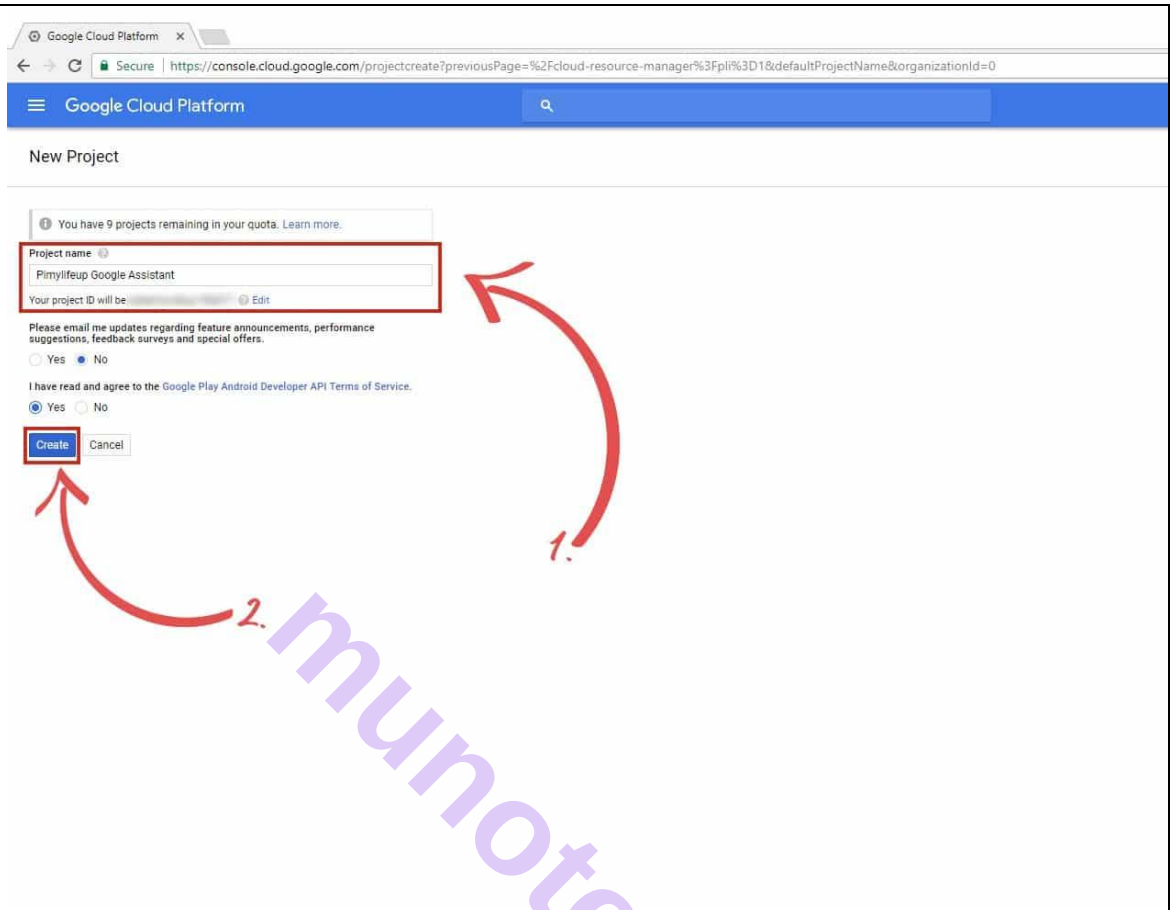
2. Once you have logged into your account, you will be greeted with the following screen. On here you will want to click the "Create Project" link as shown in our screenshot.



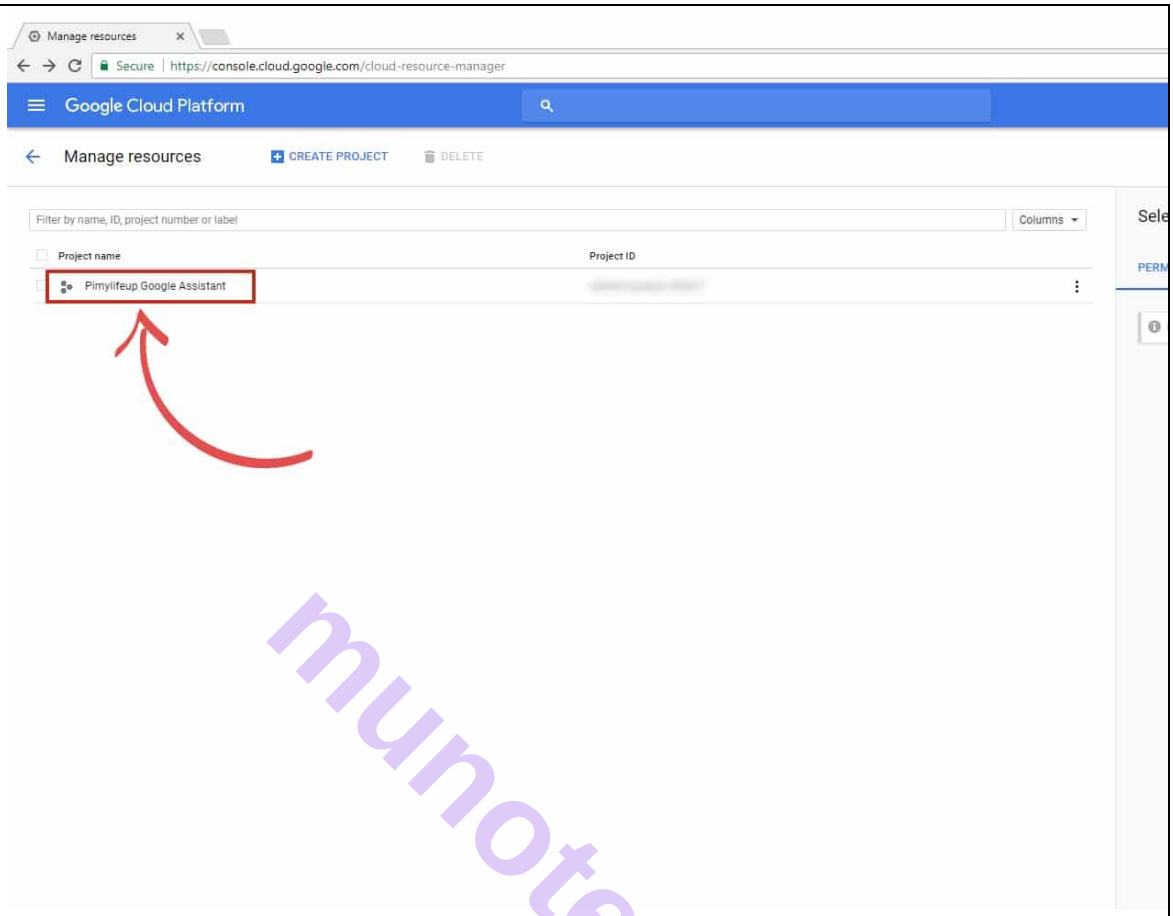
3. On this next screen, you will be asked to enter a **project name** (**1.**) as well as selecting the two radial boxes. For the project name just set something relevant to what you plan on doing. For instance, we set ours to "**Pimylifeup Google Assistant**"

For the two radial boxes, we selected '**No**' to wanting email updates and '**Yes**' to agree to their terms of service.

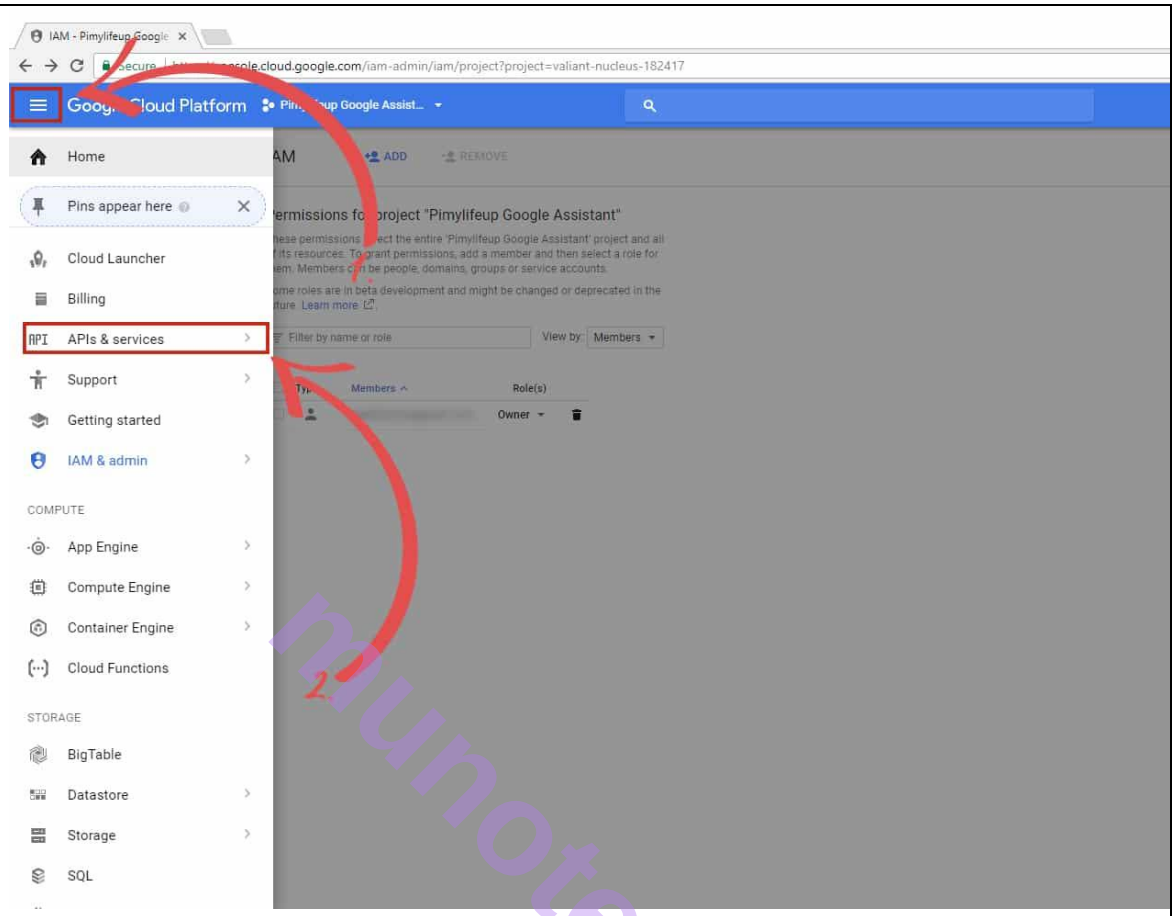
Finally, you will need to press the "**Create**" button (**2.**).



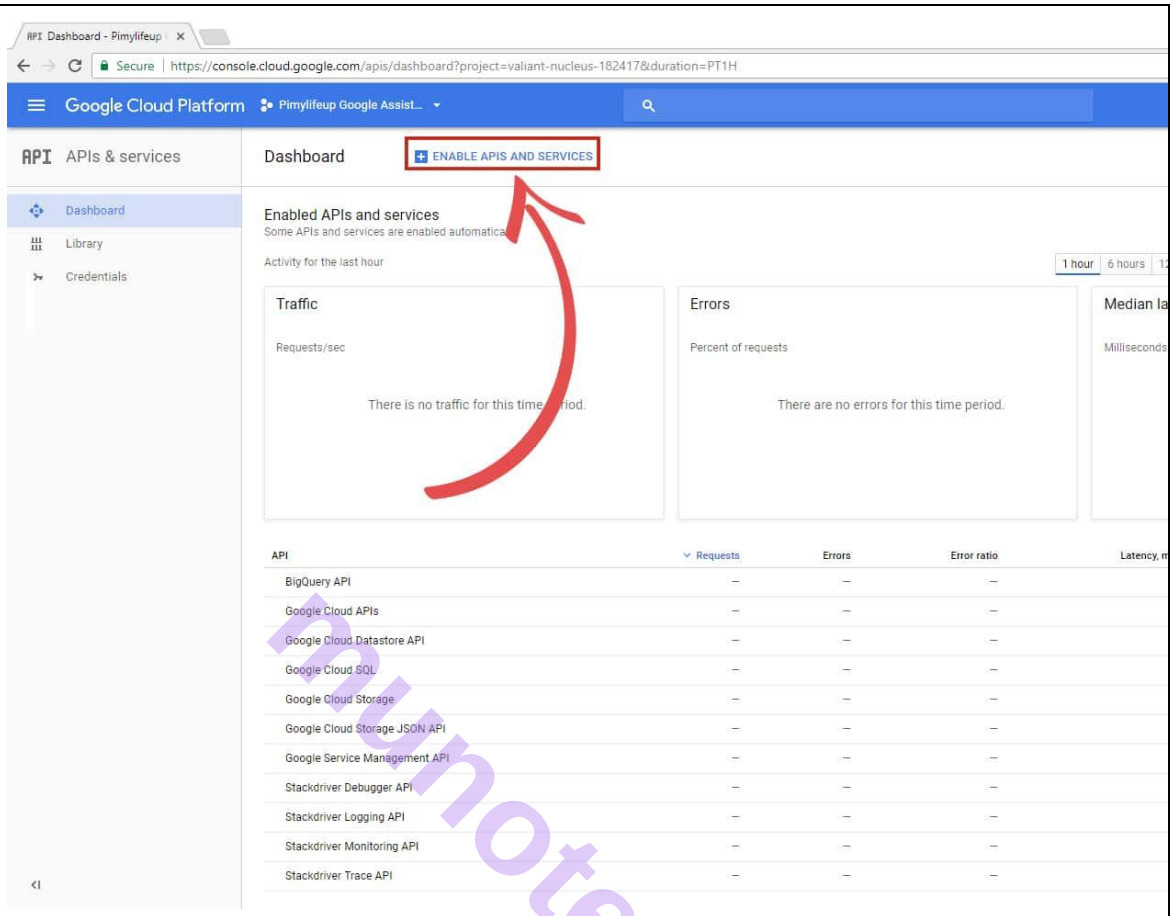
4. The project creation process can take some time, so be patient. You should receive a notification in the top right-hand corner of the screen when it's complete. If it doesn't automatically appear after some time, try refreshing the page.
Once it has appeared, click the project name to select it as shown below.



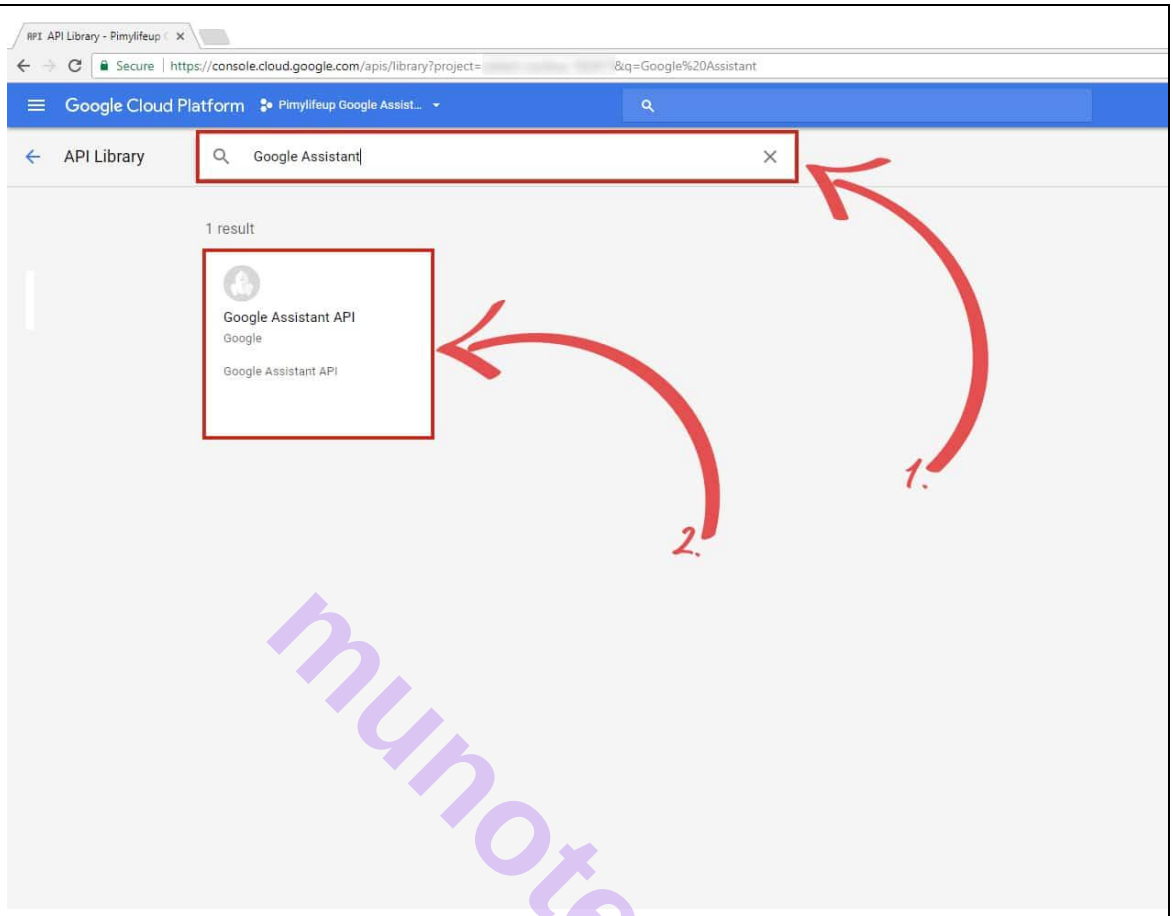
5. Now on this screen click the **hamburger icon** (**1.**) in the top right-hand corner to bring out the side menu. Then on the side menu, you will want to select "**API's and Service**" (**2.**). This screen is where we will create all the authentication details that we need and also where we will enable the Google Assistant API.



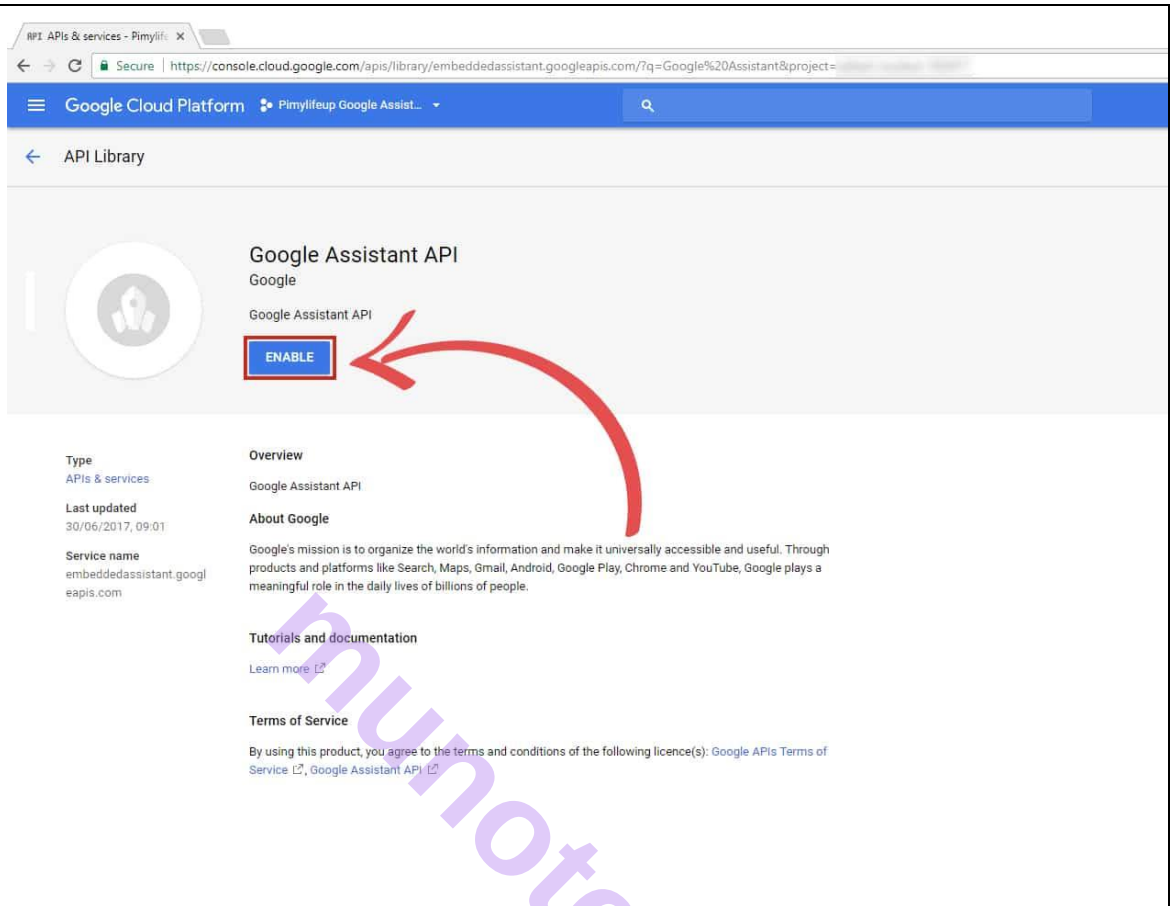
6. Now to get to the more interesting part of enabling the correct API, we need to click the "**Enable APIS and SERVICES**" link like as shown below



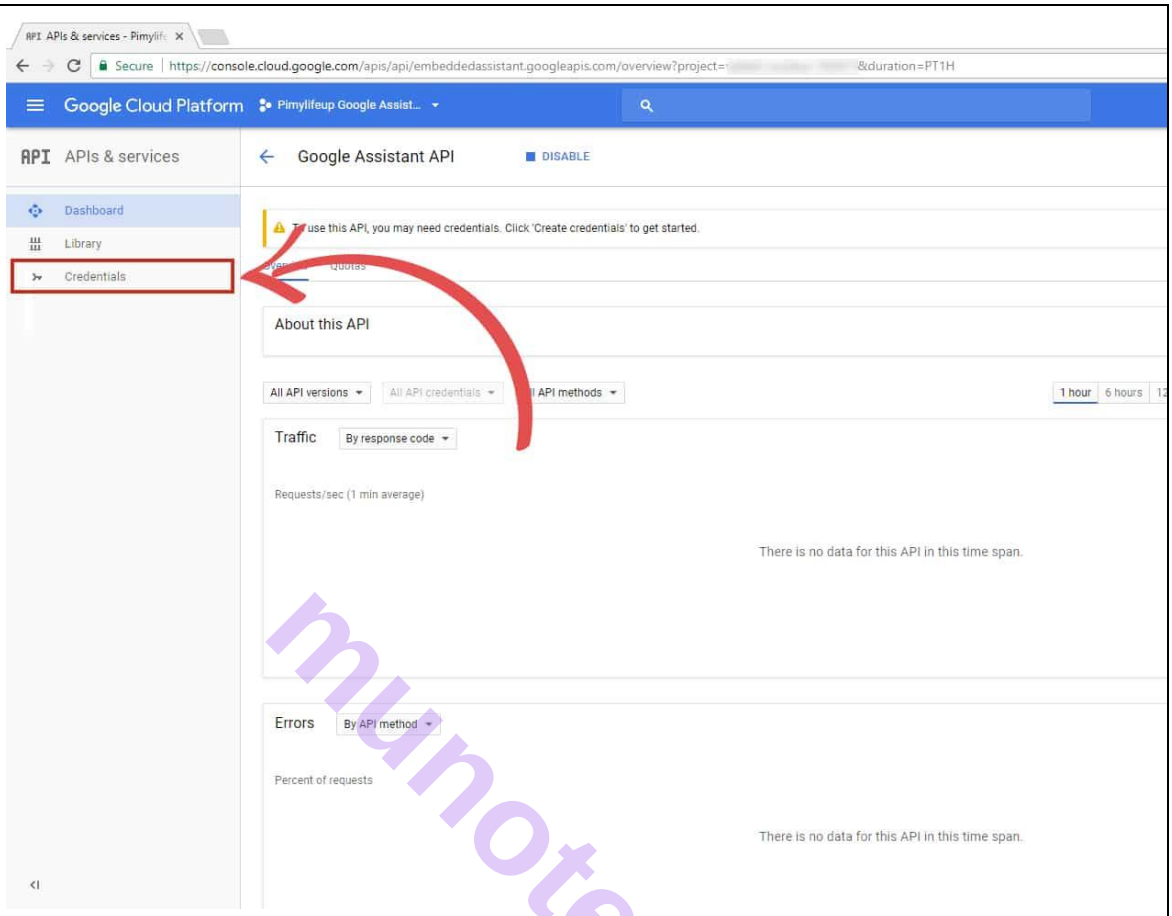
7. Finally, let's search up the API that we are after. To do this type in "**Google Assistant**" in the search box. (1.) Once the results have shown up, you need to click the box that has "**Google Assistant API**" (**2.**) written on it, as this will take us to the screen to activate it.



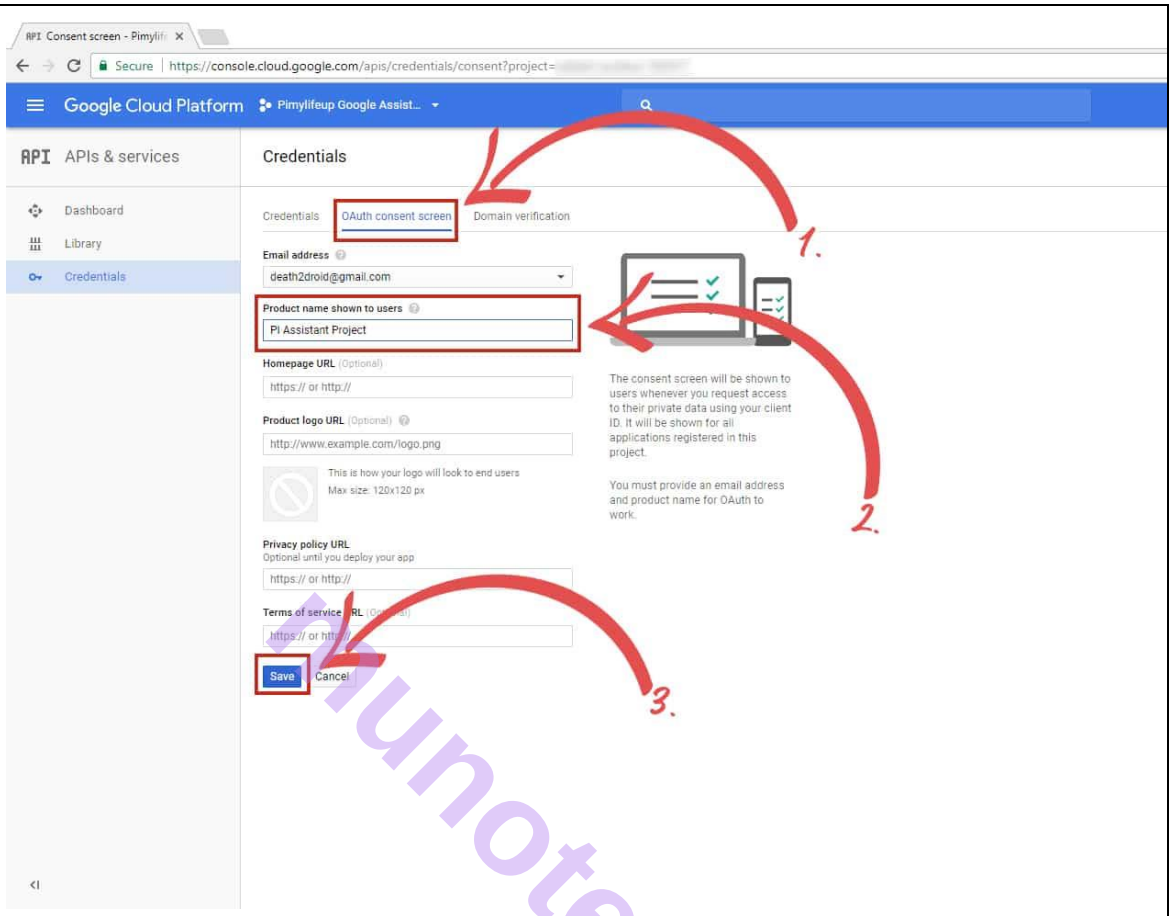
8. On this screen, you need to click the "**Enable**" button as shown below.



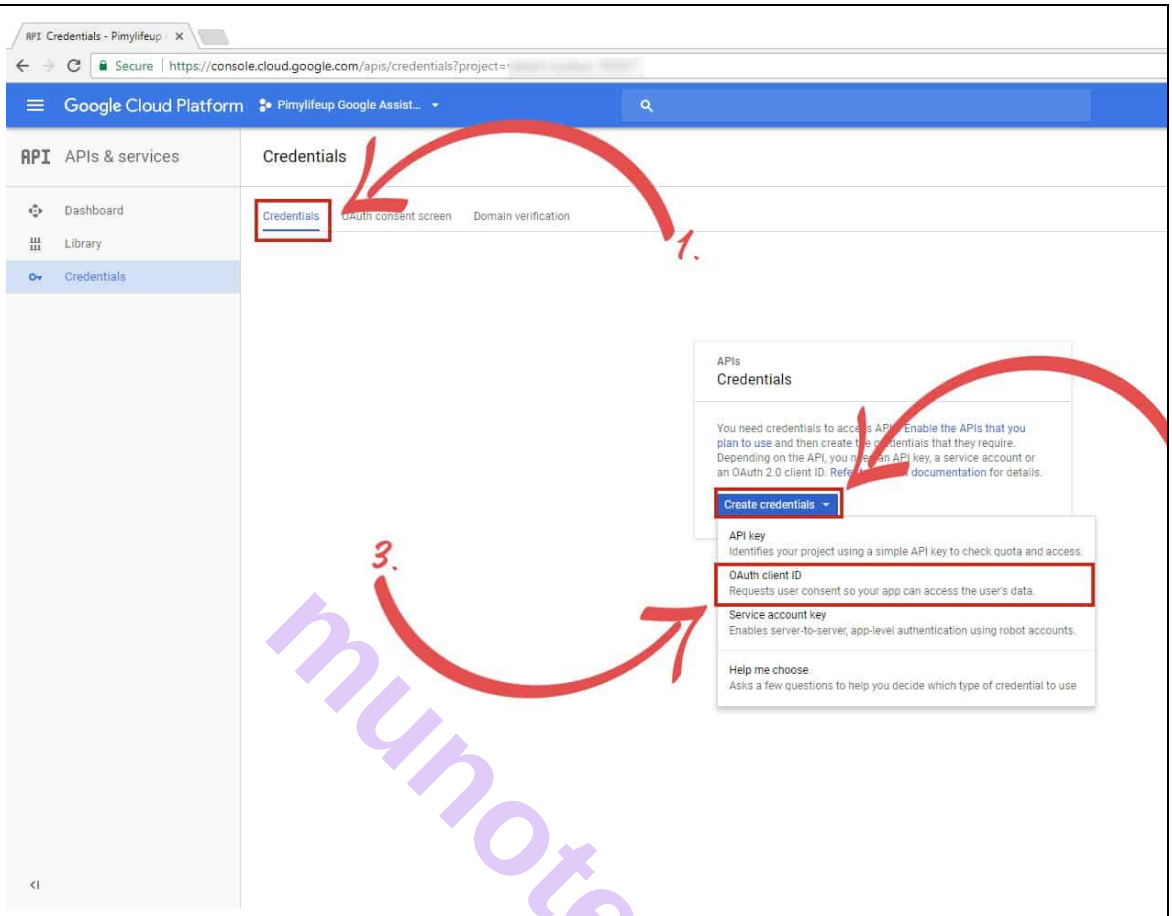
9. You should now be taken back to the "**APIs & services**" page, here you want to click "**Credentials**" in the sidebar.



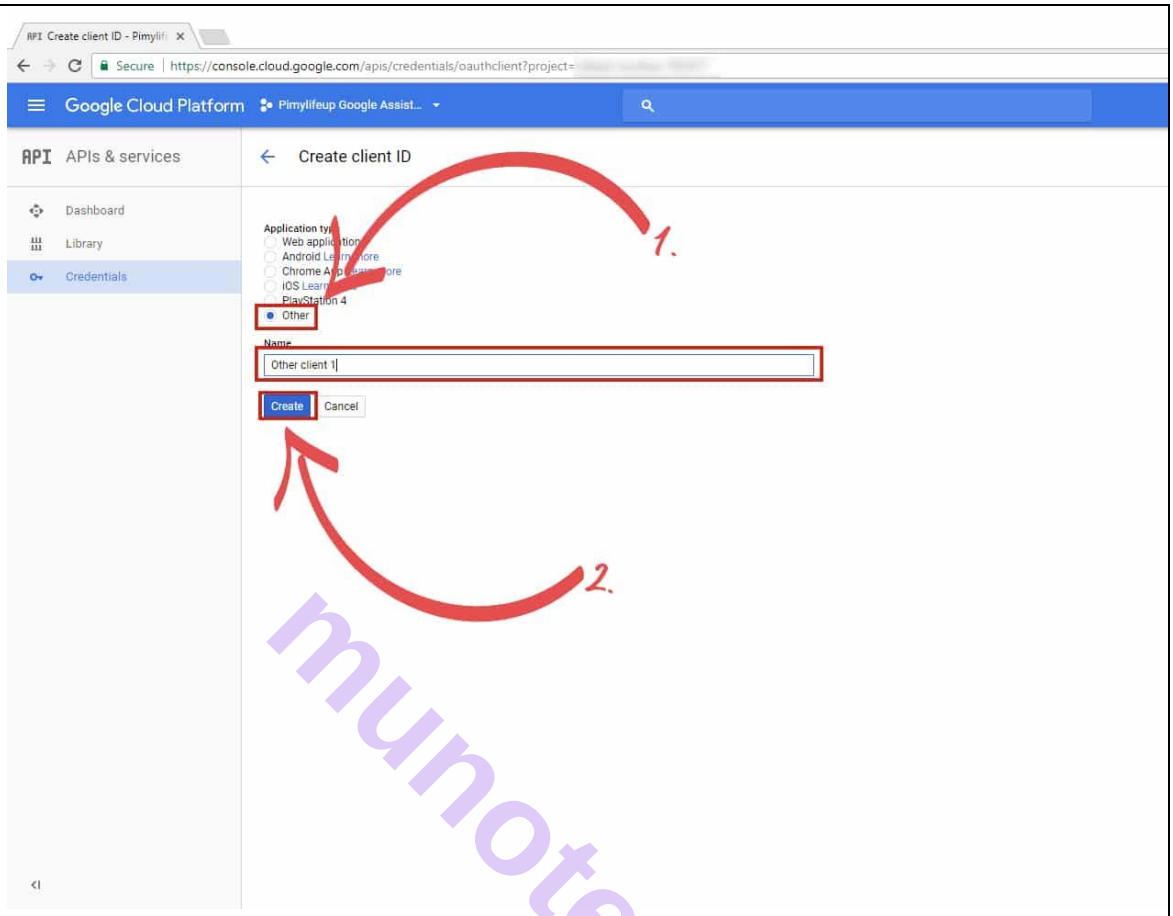
10. Now that we are on the "**Credentials**" screen we need to switch to the "**OAuth consent screen**" tab (**1.**)
Afterward, you will need to enter a name in the "**Project name shown to users**" field (**2.**), for our tutorial we named this "**Pi Assistant Project**". If you use your name, make sure you don't use the word Google in it as it will automatically refuse you.
Once you have entered the Product name of your choice, you can click the "**Save**" button (**3.**)



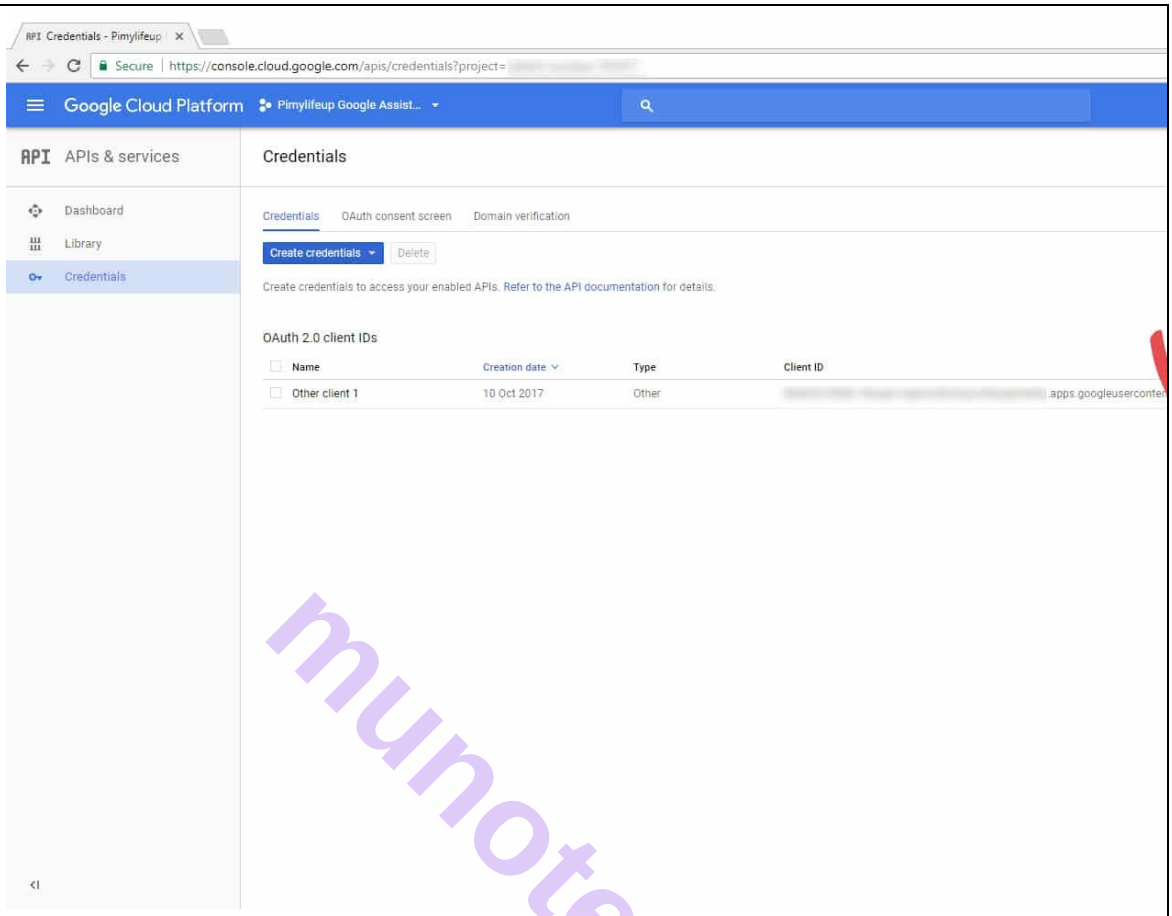
11. With the OAuth consent screen now setup we can create some credentials for it. To do this make sure you go back to the "**Credentials**" tab (**1.**).
On this screen click the "**Create credentials**" button (**2.**) which will open up a drop-down menu.
In this drop-down menu click "**OAuth client ID**" (**3.**).



12. Now while creating your client ID, set the "**Application type**" to "**Other**" (**1.**). You can also decide to change the "**Name**" for the client id, in our case we just left it set to the default name.
Finally, press the "**Create**" Button (**2.**)



13. Now we need to download the credentials file for our newly created oAuth credential. To do this click the download button as shown in the screenshot below. Keep this somewhere safe, as we will the text inside the file to the Raspberry Pi. (Of course, unless you downloaded it directly to your Pi)



14. Finally, we need to go to the URL displayed below, on here you will need to activate the following activity controls to ensure that the Google Assistant API works correctly.

- Web & App Activity
- Location History
- Device Information
- Voice & Audio Activity

https://myaccount.google.com/activitycontrols

Downloading and setting up Google Assistant

1. Now that we have setup your Google account with the Google Assistant API there are a few things we need to do. Before we begin, we should update the Raspberry Pi's operating system to ensure that we are running the latest available software.
To do this run the following two commands on the Raspberry Pi.
sudo apt-get update

sudo apt-get upgrade

2. Once the Raspberry Pi has finished updating, we can then proceed with setting up everything we need for running the Google Assistant API.

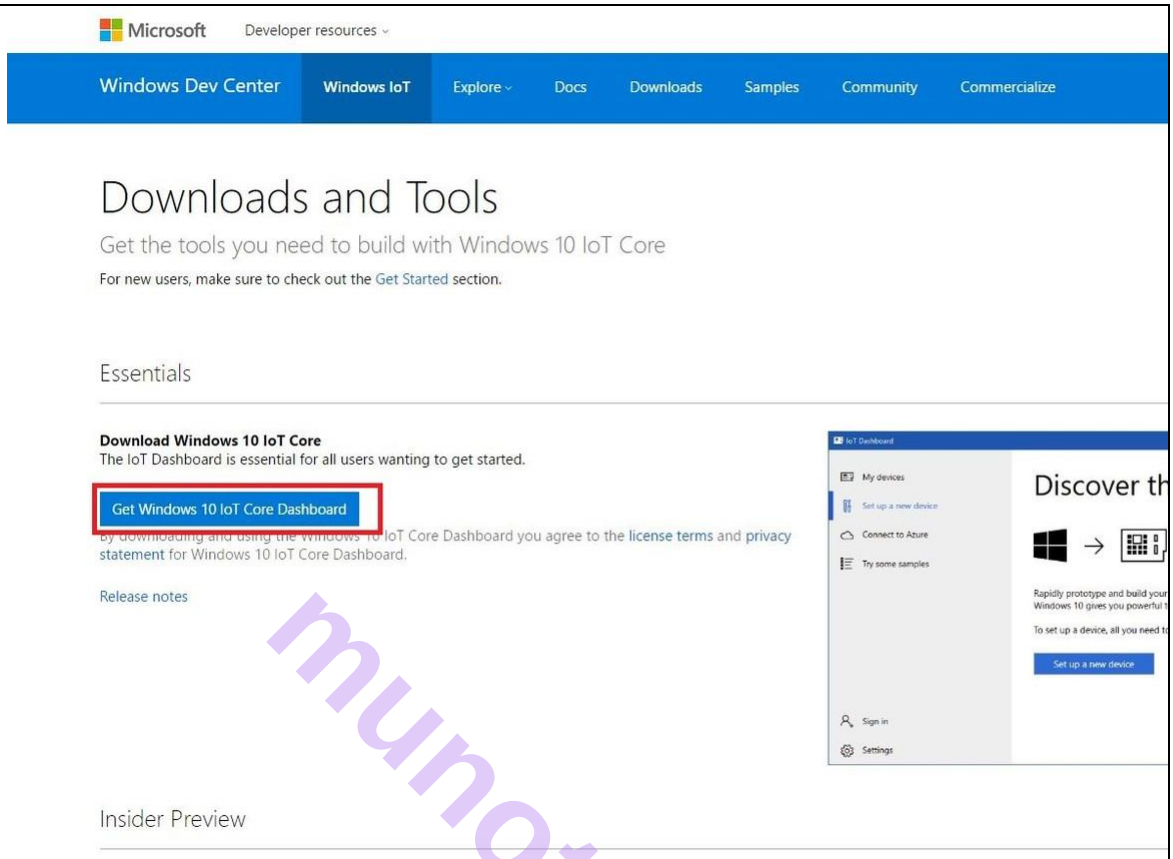
On your Raspberry Pi, we will be creating a file where we will store the credentials we downloaded earlier on our computer.
To do this run the following two commands to create a folder and begin writing a file to store the credentials in.
mkdir ~/googleassistant

nano ~/googleassistant/credentials.json
3. Within this file, we need to copy the contents of the credentials file that we downloaded to your computer. You can open the **.json** file in any text editor and press **CTRL + A** then **CTRL + C** to copy the contents.
Now in your SSH window, **right click** and click **"Paste"**.
4. Once you have copied the contents of your credentials over to our nano session, we can then save the file by pressing **Ctrl + X** then **Y** and then finally hitting **Enter**.
5. Now with the credentials file now saved safely to our Raspberry Pi we will start installing some of the dependencies we rely on.
Run the following command to install Python3 and the Python 3 Virtual Environment to our Raspberry Pi.
sudo apt-get install python3-dev python3-venv
6. We can now enable python3 as our virtual environment variable by running the following command on our Raspberry Pi.
python3 -m venv env
7. With that now enabled we can go ahead and ensure that we have installed the latest versions of pip and the **setuptools**. To do this, we will run the following command on the Raspberry Pi.
env/bin/python -m pip install --upgrade pip setuptools --upgrade
8. To get into this new Python environment that we have set up we should run the following command in terminal.
source env/bin/activate
9. Now that we have all the packages we need to install the Google Assistant Library, to do this we will run the following command to utilize pip to install the latest version of the Python package.
python -m pip install --upgrade google-assistant-library

Getting the Google Assistant Running

1. Now that we have set up all the prerequisites to running the Google Assistant software on our Raspberry Pi we can finally get up to running it.
To do this, we must first install the Google authorization tool to our Raspberry Pi. This package will allow us to authenticate our device and give ourselves the rights to be able to make Google Assistant queries for your Google Account.
Run the following command on the Raspberry Pi to install the Python authorization tool.
python -m pip install --upgrade google-auth-oauthlib[tool]
2. With the Google Authentication library now installed, we need to run it. To do this, we will be running the following command on our Raspberry Pi.
This command will generate a URL you will need to go to in your web browser so be prepared.
google-oauthlib-tool --client-secrets ~/googleassistant/credentials.json --scope https://www.googleapis.com/auth/assistant-sdk-prototype --save --headless

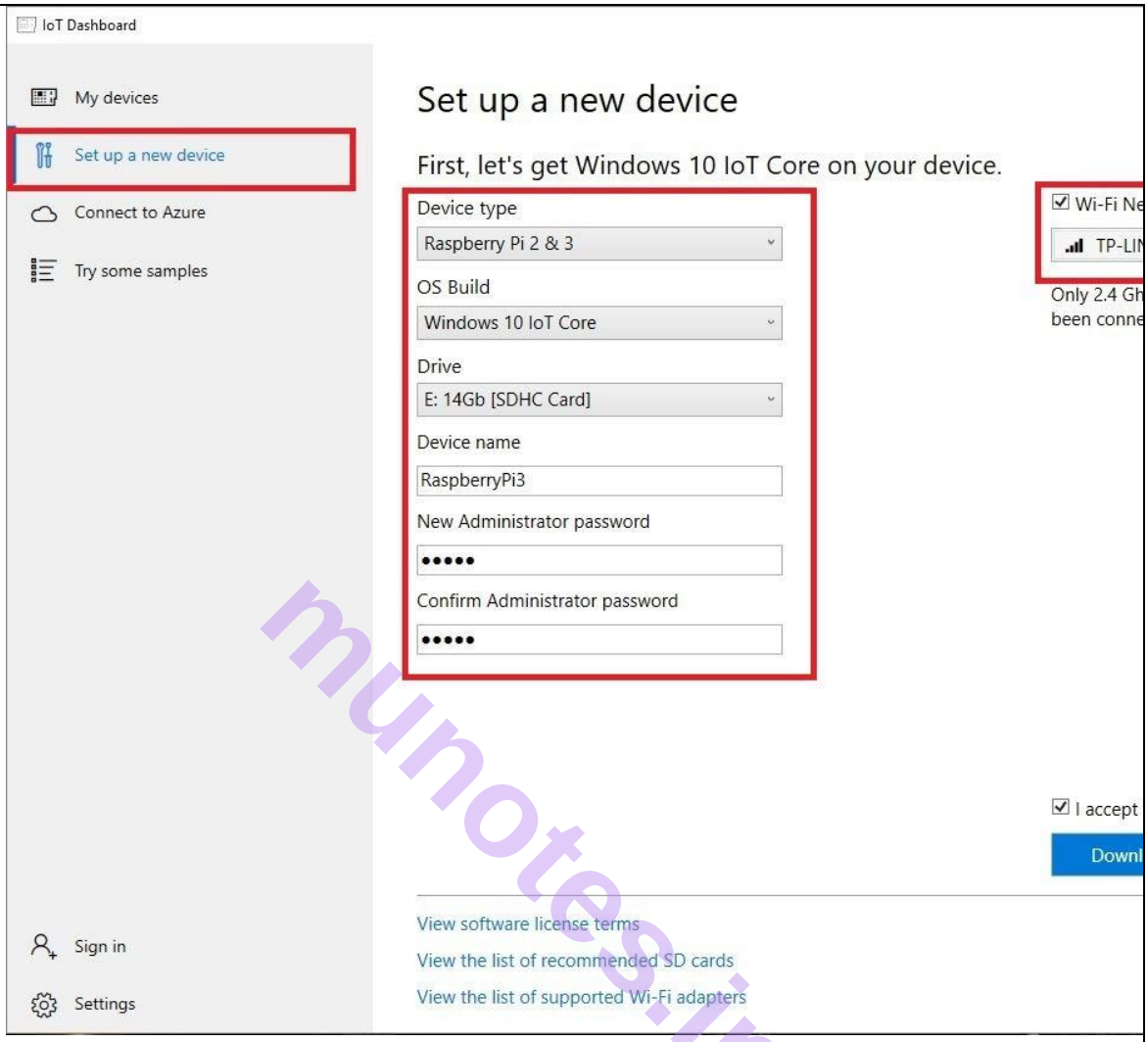
	3. You will now be presented with the text "**Please visit this URL to authorize this application:**" followed by a very long URL. Make sure you copy this URL entirely to your web browser to open it. **4.** On this screen login to your Google account, if you have multiple accounts make sure you select the one you set up your API key with. Afterward, you should be presented with a screen with the text "Please copy this code, switch to your application and paste it there" followed by a long authentication code. Copy the authentication code and paste it back into your terminal session and press enter. If the authentication was accepted you should see the following line appear on your command line: "**credentials saved: /home/pi/.config/google-oauthlib-tool/credentials.json**" **5.** Finally, we have finished setting up everything we need to ruin the Google Assistant sample on our Raspberry Pi. All that is left to do now is to run it. We can run the software by running the following command on your Raspberry Pi. google-assistant-demo **6.** Say "**Ok Google**" or "**Hey Google**", followed by your query. The Google Assistant should give you a response. If the Assistant software doesn't respond to your voice, then make sure that you have correctly setup your microphone and speakers by checking all cables.
12	**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.

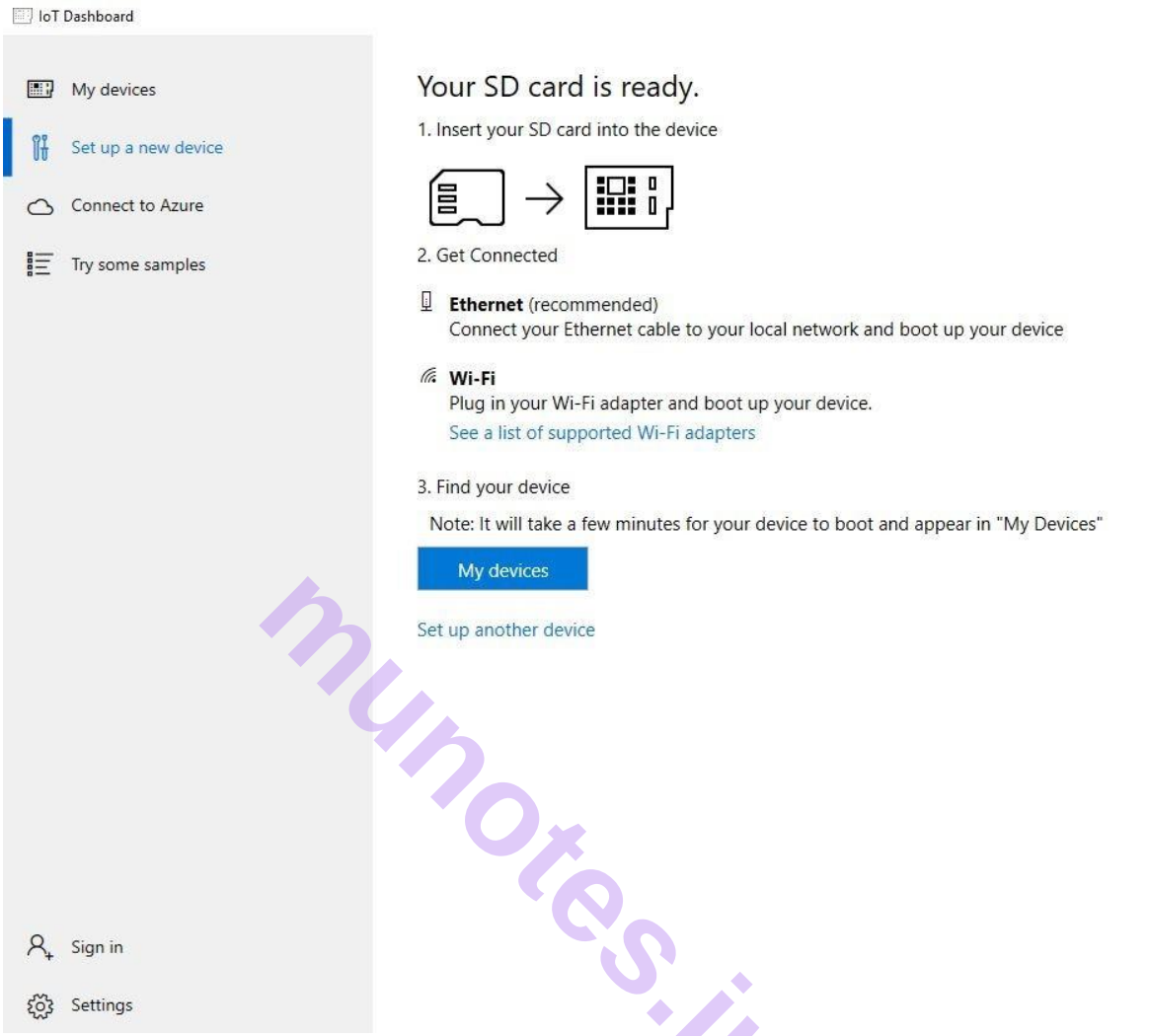
Select the options as shown in the image below. Make sure you select the correct drive for your microSD card and give your device a name and admin password.



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.