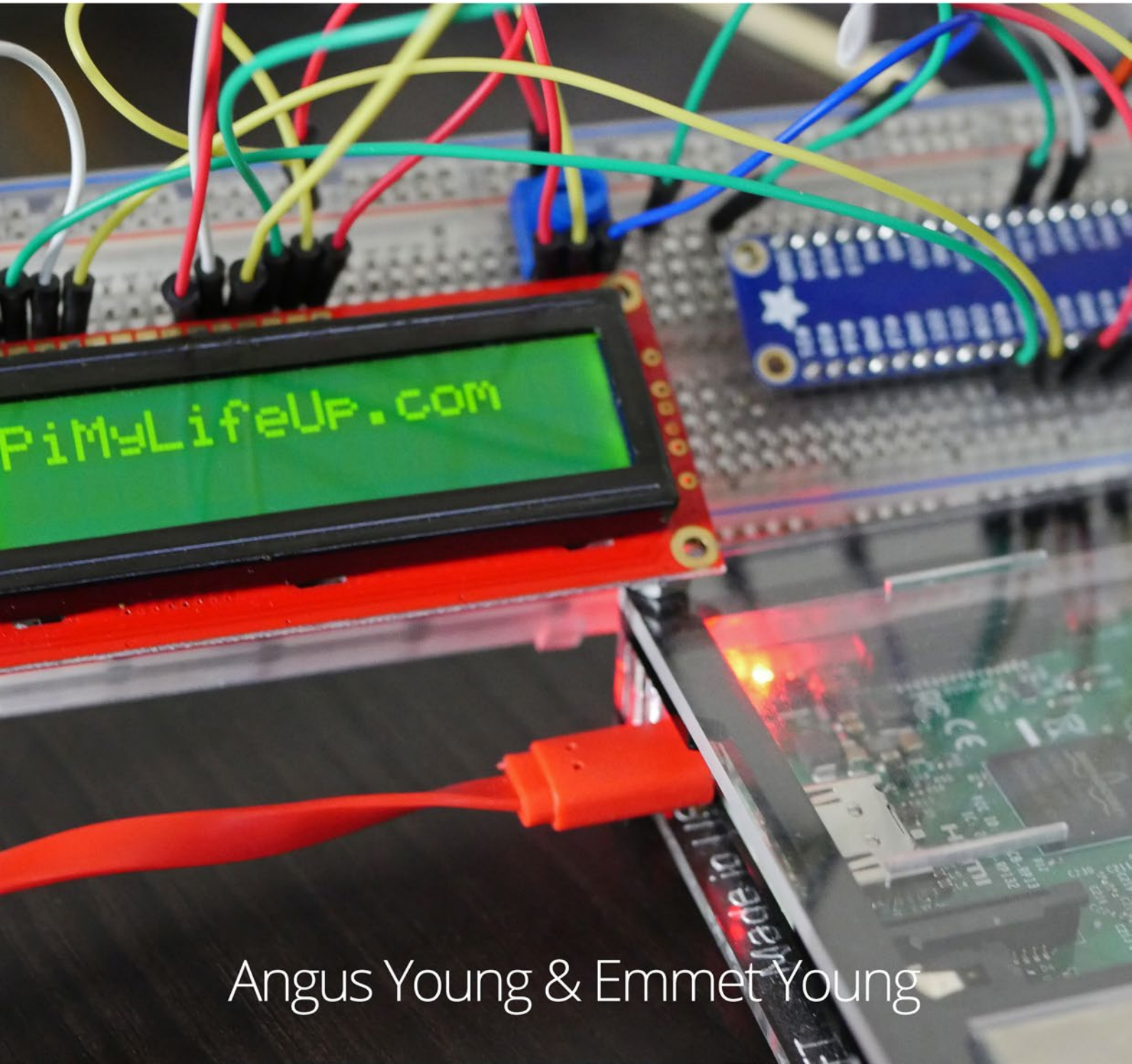
Pi My Life Up's **Circuitry Projects** For the Raspberry Pi



Angus Young & Emmet Young

PiMyLifeUp

For permission contact Pi My Life Up at:

support@pimylifeup.com

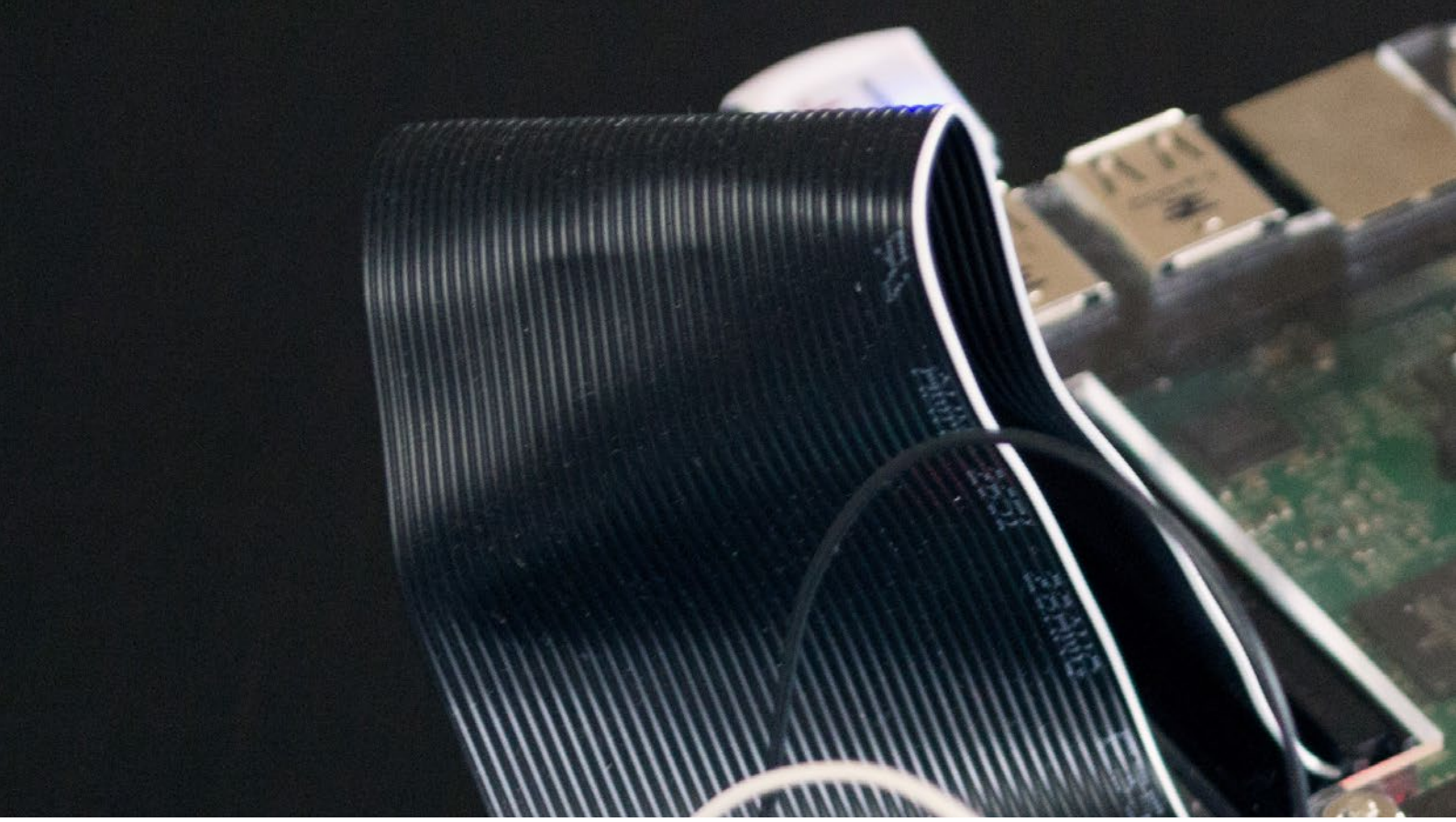
The tutorials in this book are for general information purpose only.

While we strive to keep our information up to date, we do not make any warranties about the completeness, reliability, and accuracy of this information.

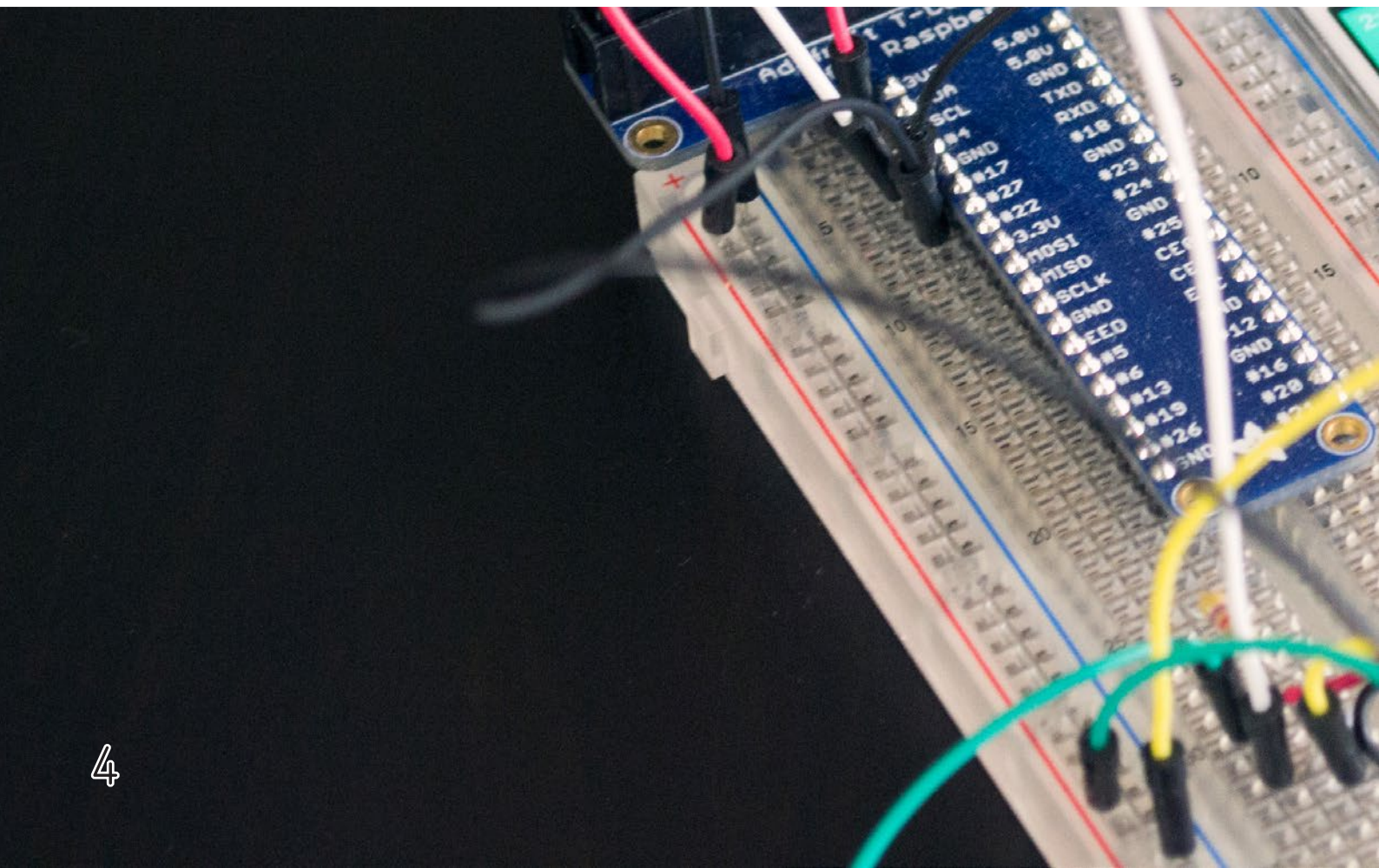
Any action you take upon the information on within this book is strictly at your own risk.

We will not be liable for any losses and damages in connection with the use of the information within this book

Contents



IoT & Sensor Projects




My Room
RAM
27%
Storage
CPU
4%
RPI
41.16
Processes
0
00000702ffa
23.38
Digital Output

Cayenne IoT Builder

Project Description

In this tutorial, I will be looking at how to setup Raspberry Pi cayenne. This is a simple process and will give you access to a powerful IoT software package. If you're a fan of sensors, collecting data and the overall concept of IoT then this is for you.

There are quite a few features and things you're able to do with this software but to keep things relatively simple and straightforward I will just touch on the basics in this tutorial.

One thing that you will probably like especially if you're a beginner is that you don't need to do any coding to get a project up and running.

All you need to do is connect the sensors up correctly and add the sensors/devices in cayenne. In Cayenne, itself you can create triggers, events, monitoring widgets and much more.

Equipment

Required

- Raspberry Pi
- SD Card (8 GB+ Recommended)
- Network Connection

Optional

- Raspberry Pi Case
- USB Keyboard
- USB Mouse

Equipment used in sample circuit

- DS 18B20 Temperature Sensor
- 4v7k Resistor
- GPIO breakout kit
- Breadboard
- Breadboard Wire

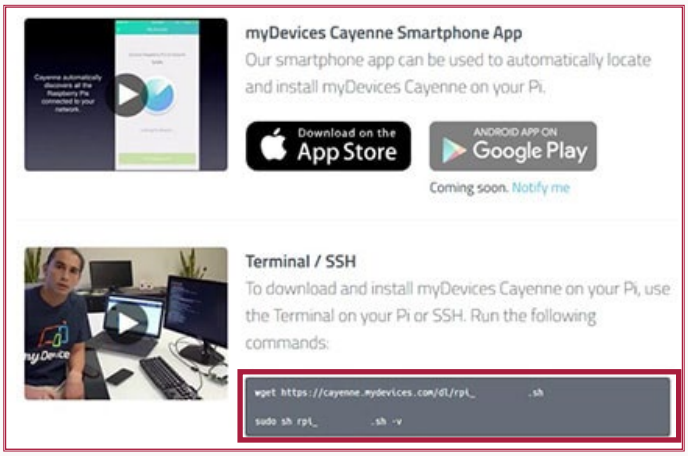
Installing Raspberry Pi Cayenne

The process of installing Cayenne onto the Raspberry Pi is pretty simple and shouldn't take you too long to get it up and running.

1. First, head over to myDevices Cayenne through this link, https://pimylifeup.com/out/cayenne and sign up for a free account.

2. Once you are signed up, you will need to register/connect the Pi up to the account you just created.

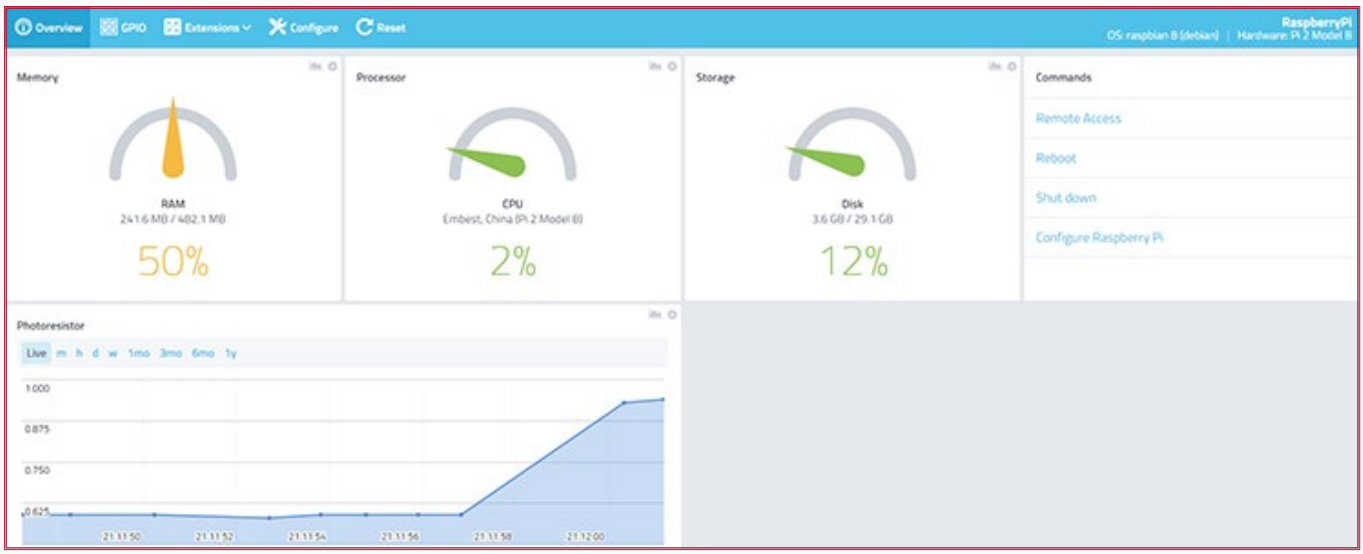
To do this just copy the two command lines shown after you sign up. Enter these into the terminal for your Pi. (These files are unique for every new install)



Alternatively, you can download the app, and it can automatically locate & install Cayenne onto your Pi.(Keep in mind SSH needs to be enabled)

3. It will take a few minutes to install onto your Pi depending on how fast your internet connection is. The web browser or app should automatically update with information on the installation process.

4. Once installed the dashboard on your myDevices Cayenne page will display and should look a bit like below.



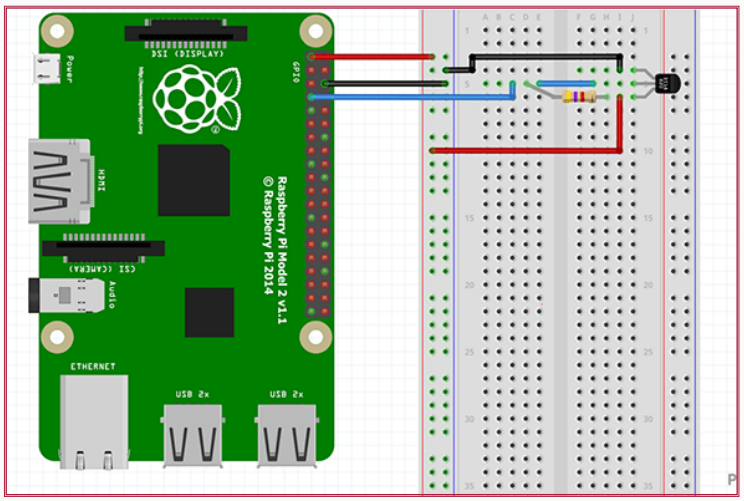
Setting up your first sensors on Cayenne

In this example, I am going to set up a temperature sensor. The sensor that I use is called DS18B20 and is the same sensor we use in a tutorial in the advanced section of this book. The process of setting this up without Cayenne was a little more in-depth but using Cayenne makes it super simple.

For starters, all you need to do is set up the circuit and have it connected to the Pi.

If you need the full tutorial on how to do this then be sure to check out the **Raspberry Pi DS18B20** tutorial in the advanced section of this book, or if you know what you're doing just check out the circuit diagram to the right.

I also added an LED that is connected to **pin #17** with a 100-ohm resistor to the ground rail.



Now when I set this up the sensor was automatically detected and added to my dashboard. Which is a cool feature for the software, however, if it didn't add automatically then you will need to add it manually. To add it manually, do the following.

1. Go to **add new** in the upper left corner of the dashboard.
2. Select **device** from the dropdown box.
3. Find the device, in this case it is a **DS18B20 temperature sensor**.
4. Add all the details of the device. In this case, you will need the slave address for the sensor. To get the slave address enter the following into the Pi's terminal.

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

5. The slave address will look like this **28-000007602ffa**. Just copy & paste this into the slave field within Cayenne dashboard.
6. Once entered select **"add sensor"**.
7. The sensor should now be displayed on the dashboard.
8. If you need to customize your sensor press the cog and it will come up with some options.
9. You can also see stats/graphs. For example, the temperature sensor can plot data in real time and will keep historical data too.

Setting up your first sensors on Cayenne - Continued

If you want also to add an LED that you can turn on & off via the dashboard follow the next few instructions.

1. Now let's add one more device. Except this one will be an LED.
2. So go back to add new device.
3. Now search for digital output and select it.
4. For this device select your Pi, **widget type** is **button**, **icon** can be whatever you want, and then select **integrated GPIO**. Finally, **channel** is the pin/channel that our LED is connected to. For this example, it is **pin #17**. (This is the GPIO numbering of the pins).
5. Now press the add sensor button.
6. You can now turn the GPIO pin **high** & **low** from the dashboard and use it as a **trigger**.
7. We are now ready to set up our first **trigger**.

You should now have two devices on your dashboard that should look something like this.

Note: If you are new to setting up sensors and using the GPIO pins then it might be worthwhile checking out my guide for the GPIO pins on the Raspberry Pi.

Setting up your first trigger

Triggers in Cayenne are a way of having your Pi react to a particular change on the Raspberry Pi Pi itself or a change in a sensor attached to it.

This trigger could be something as simple as a temperature exceeding a specific value or even just your Pi going offline.

As you could imagine this can be quite powerful in creating smart devices that react to the surroundings. For example, If the room gets too cold, then turn a heater on.

The process of adding a trigger is super simple, and I will go through the basics of setting up a couple.

1. Go to **add new** in the upper left corner of the dashboard.

2. Select **trigger** from the down box.

3. First name your trigger, we will call ours "**Too hot**".

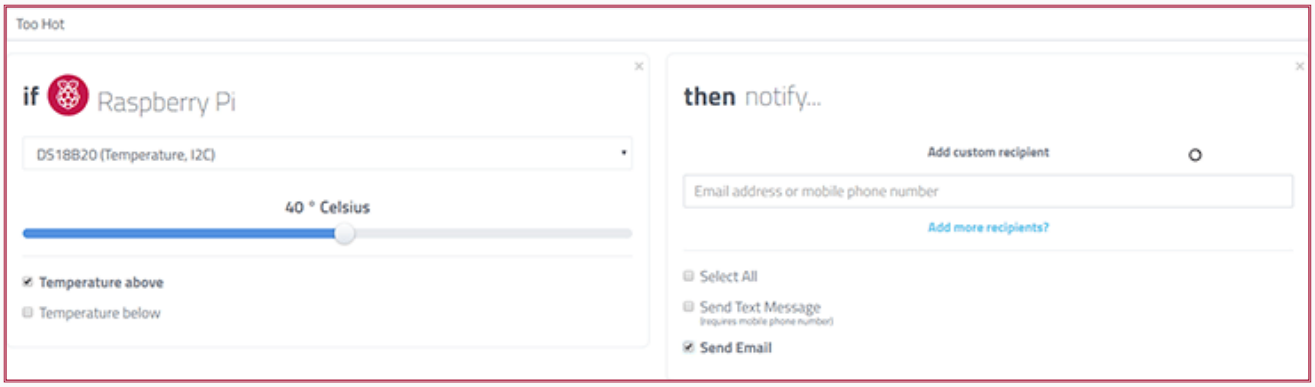
4. Now drag and drop your Raspberry Pi from the far-left corner into the **if box**.

5. Underneath this select, the **Temperature sensor** and have the checkbox next to "**Temperature above**" selected.

If device options aren't displayed then just refresh the page

6. Now in the **then box** select notification and add an email address or your phone number for a text message (You can add both).

Make sure you tick the checkboxes as well.



7. Now click "**Save trigger**"

8. It should now be saved and will start to send you alerts whenever the temperature sensor is over 40 degrees Celsius.

9. You can also drag the Raspberry Pi into the then box and have it do many things including controlling output devices.

For example, in our circuit, we have an LED that will be switched on when the temperature from the sensor exceeds 40 degrees Celsius.

Setting up your first trigger

10. To make the LED trigger click on new trigger located in the upper part of the page.

Name this trigger "**Activate LED**".

11. Now drag the Pi into the **if box** and then select the temperature sensor again with 40 degrees above Celsius.

12. Now drag the Raspberry Pi into the **then box**. Select our digital output and have the **on checkbox** ticked.

13. Now click **"Save trigger"**

14. Now whenever our temperature sensor connected to the Pi reports a temperature that is above 40 degrees Celsius, it will send an email and turn on the LED.

You will also need to add another trigger to turn the LED off when it drops back below 40 degrees, but we will leave that for you to work out and move on to events.

Cayenne Events

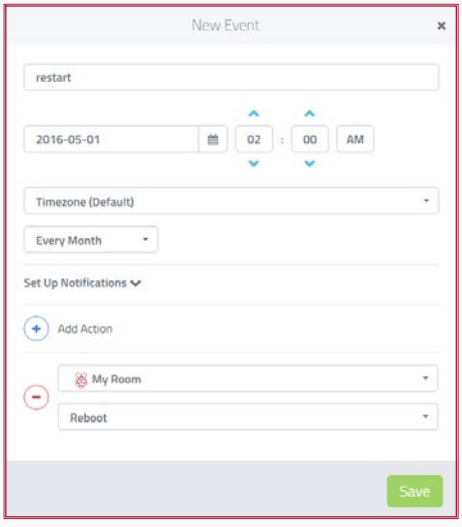
Events in Raspberry Pi Cayenne is somewhat like triggers, but they are time dependent rather than relying on a change in a sensor or the device itself. Setting up an event is easy, I will just quickly go through how to setup your Pi to restart once a month.

1. Go to **Add new** in the upper left corner of the dashboard.

2. Select **Event** from the down box.

3. You should now see a screen with a **calendar**, and a popup called **New event**.

4. Enter the details of your event. For example, ours is called **restart** and will be triggered on the first of every month at 2 am. To the right is an example of the screen.



5. Once done, click on save.

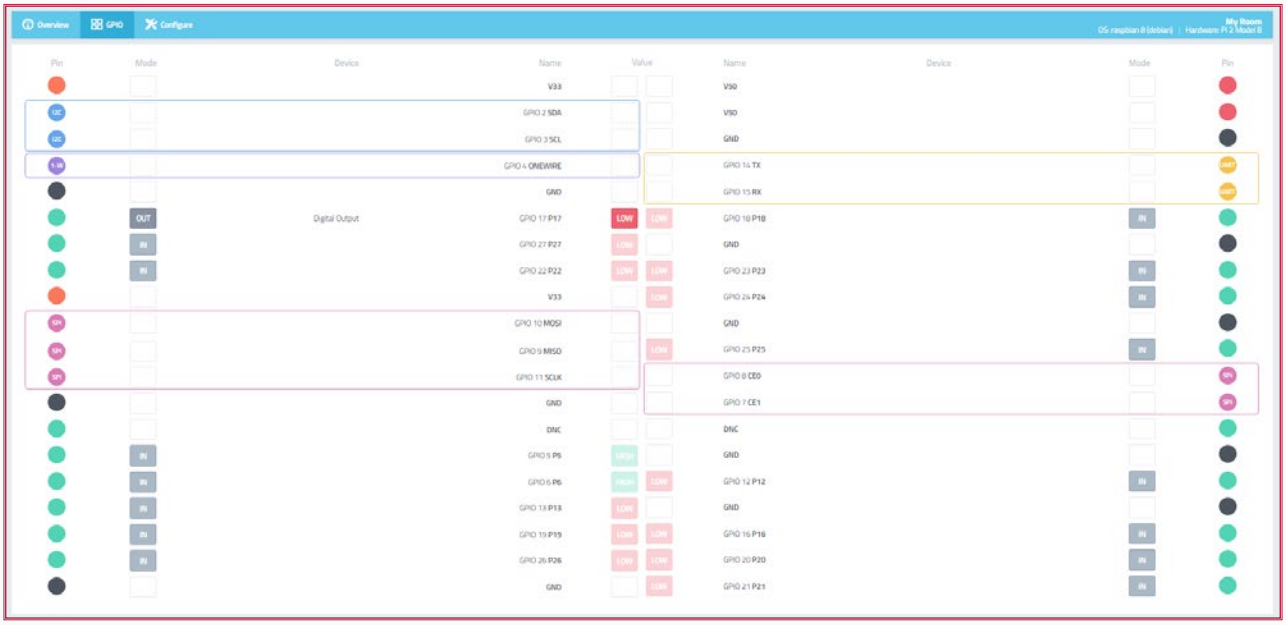
6. You should now be able to see your event on the calendar. Just click on it if you wish to edit it.

As you could imagine events can be powerful so it will be worth considering these more. An excellent example of using events will be if you need something to run or turn on such as lights needing to be turned on at a specific time.

Cayenne GPIO Pannel

The GPIO panel within Raspberry Pi Cayenne allows you to control and alter the pins on the Pi.

For example, you can turn a pin from being input to an output and vice versa. You can also turn output pins both low & high.



As you can see it also makes for a great reference sheet if you need to refer back and see which pins are the ones you need.

You can also look at the devices that are currently assigned to specific pins. You are also able to see the current status of a pin. (e.g., Input or output & low or high)

Cayenne Remote Desktop

We have touched on connecting remotely to the Pi twice before. The first time was via secure shell and the second was setting up VNC viewer.

If you install Cayenne, you can also remote desktop to your Raspberry Pi through either the web browser or via the mobile app. You can do this by doing the following.

1. On the dashboard find the widget that says **"Commands".**

2. Within this widget click on **Remote access**.

3. It will now connect to the Raspberry Pi, and a new window will be opened. If a new window doesn't open your browser most likely blocked it.

Simply allow **cayenne.mydevices** to open new tabs.

4. Once done you can control your Pi like as if you were there with it.

5. One of the pros with using Cayenne as a remote desktop tool is that you can access it anywhere in the world quite easily rather than needing to set up a VPN or open ports on your network.

Cayenne Remote Desktop

Cayenne is still relatively new and has so much more coming in the future. It is already jam-packed full of features.

We have read of future integrations with the Pi camera and other devices. The upcoming integrations would open up Cayenne to creating some cool projects.

We hope you have been able to get myDevices Cayenne installed onto your Raspberry Pi without any issues.

Weather Station
Using the Sense HAT

Project Description

In this tutorial, we will be shows you how to set up a very basic Raspberry Pi weather station by utilizing the Sense HAT. The Sense HAT is a fantastic piece of equipment that comes with a vast abundance of sensors all in a single package.

This weather station tutorial will show you how to setup the Sense HAT software and how to retrieve the data from its three primary sensors, those being the temperature, humidity and pressure sensors. We will also briefly touch on how to write text to the LED Matrix as we will use this as a way of displaying your sensor data.

We will also go into setting up your Raspberry Pi weather station so that the data is stored on Initial States Data analytics service. This service will allow us to stream the data directly to there web interface and view all the data in a pretty graph.

This tutorial will make use of a significant amount of Python code.

Equipment

Required
- Raspberry Pi
- SD Card (8 GB+ Recommended)
- Sense HAT
- Network Connection

Optional
- Raspberry Pi Case
- USB Keyboard
- USB Mouse

Getting started with the Sense HAT

Before you get started with this tutorial, make sure that you have correctly placed the Sense HAT over the GPIO pins on the Raspberry Pi. It's an incredibly easy installation and shouldn't require any extra fiddling once put on correctly.

1. Before we get started, we need to run the following commands to ensure that the Raspberry Pi is running the latest software.

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

2. With the Raspberry Pi up to date, we now need to install the **sensehat** software package, this provides all the libraries we need to interact with the Sense HAT.

```
sudo apt-get install sense-hat
sudo reboot
```

3. Now we have the software installed will write a quick script to ensure the Sense HAT is working correctly.

We can start writing this script with the following command.

```
sudo nano ~/sensehat_test.py
```

4. Now write the following lines into this file, we will explain what each section of code does as we go.

```
from sense_hat import SenseHat
```

This line imports the SenseHat module from the sense_hat library. This library allows us to interact with the sense hat itself through python.

```
sense = SenseHat()
```

This line creates a link to the **SenseHat** library and initializes itself so we can start making calls to it.

```
sense.show_message("Hello World")
```

This line writes a message to the Sense HAT. You should see "Hello World" scroll across the RGB lights.

Press **Ctrl + X** then **Y** then press **Enter** to save the file.

5. With the file now saved we can run it with the following command:

```
sudo python ~/sensehat_test.py
```

The text "**Hello World**" should now scroll across the RGB Leds on top of the Sense Hat. If it doesn't, it is likely that Sense HAT has not been properly pressed down on top of the GPIO pins.

If it is still not working, try restarting the Raspberry Pi by running the following command.

```
sudo reboot
```

Setting up your Sense HAT as a weather station

1. Now that we have tested that the Sense HAT is working correctly we can now get writing our python weather script.

We will start with a basic script that continually reads and displays all the data from the Sense HAT's sensors.

To begin writing our new python script run the following command in terminal.

```
sudo nano ~/weather_script.py
```

2. Now enter the following lines of code into the script, we will explain each block of code as we go.

```
#!/usr/bin/python
from sense_hat import SenseHat
import time
import sys
```

First, need to import all the libraries that we plan to utilize in our script. In our case, we will be using the following libraries.

sense_hat Library
This library is what we utilize to interact with the Sense Hat itself, without this we wouldn't be able to read any of the sensor data or interact with the LED matrix.

time Library
This library allows us to do a large variety of different time stuff, but for our simple script, we will be just using its sleep functionality. This function will enable us to suspend the current thread for a small period.

sys Library
This library allows us access to some variables and functions that are managed by the Python interpreter. In the case of our script, we will be using this to terminate the script if we ever need to do so.

```
sense = SenseHat()
sense.clear()
```

The first line creates a link to the SenseHat library and initializes itself so we can start making calls to it.

The second line tells the library to clear the LED Matrix, by default this means switching off all the leds. It's always a good idea to do this when dealing with the Sense HAT as it ensures nothing is being displayed already.

```
try:
    while True:
```

In this code block set up our first try statement. We need to do this so we can break out of our while loop by pressing **Ctrl + C**. make sure you keep the indentation for the **while True**.

The indentation is needed because Python is sensitive to indentation. The next few lines of code will need a three-tab indentation.

```
temp = sense.get_temperature()
```

Getting the temperature is straightforward thanks to the Senese Hat Library, all we have to do is make a call to the library for it to retrieve the temperature from the sensor.

The output that this will give us will be in Celsius, it also provides a larger decimal number, but we will deal with that on our next line of code.

We will explain how to convert the temperature to Fahrenheit if you would prefer to deal with that instead of Celsius.

Celsius

```
temp = round(temp, 1)
```

Fahrenheit

```
temp = 1.8 * round(temp, 1)  + 32
```

Here we give you two choices for this line of code. The Celsius code just utilizes the value we got from the sensor and rounds it to the nearest decimal place.

The Fahrenheit code is virtually the same, the only difference being that we convert the value from Celsius to Fahrenheit.

Celsius

```
print("Temperature C",temp)
```

Fahrenheit

```
print("Temperature F",temp)
```

This bit of code is extremely basic and just prints the temperature to the terminal.

```
humidity = sense.get_humidity()
humidity = round(humidity, 1)
print("Humidity :",humidity)

pressure = sense.get_pressure()
pressure = round(pressure, 1)
print("Pressure:",pressure)
```

Both the humidity and pressure sensors can be read just like the temperature sensor. Luckily for us, the Sense Hat library makes this incredibly simple.

Their values also come back with as a large decimal number, so we will again round them then display the values to the terminal.

There isn't much extra to say about these two code blocks as they operate just like the temperature code.

```
            time.sleep(1)
```

This line is a simple call to the time library that puts the script to sleep for approximately 1 second. This sleep codex is to reduce the rate at which the data is read and output.

You can speed up the read rate by decreasing this number or deleting the line.

You can also slow it down further by increasing the number.
The number should approximately be the number of seconds you want it to wait between reads.

```
    except KeyboardInterrupt:
        pass
```

This code makes the try wait for an **KeyboardInterrupt** exception. With it triggered we ignore the exception to cause the script to leave the while loop cleanly. We do this by utilizing **pass**.

3. With all the code entered into our python file, you should end up with something that looks like below, of course, this will differ if you used the Fahrenheit conversion code and not just straight Celsius.

```
#!/usr/bin/python
from sense_hat import SenseHat
import time
import sys

sense = SenseHat()
sense.clear()

try:
    while True:
        temp = sense.get_temperature()
        temp = round(temp, 1)
        print("Temperature C",temp)

        humidity = sense.get_humidity()
        humidity = round(humidity, 1)
        print("Humidity :",humidity)

        pressure = sense.get_pressure()
        pressure = round(pressure, 1)
        print("Pressure:",pressure)

        time.sleep(1)
except KeyboardInterrupt:
    pass
```

4. Once your code looks something like the one displayed above, and you are sure you have correctly indented your code you can quit and save.

To do this press **Ctrl+X** and then press **Y** and then press **Enter**.

Setting up your Sense HAT as a weather station - Continued

4. We can now run our new python script by running the following command in terminal.

```
sudo python ~/weather_script.py
```

5. You should now start to start to see text similar to the following appear in your terminal if everything is working as it should be.

```
('Temperature C', 30.0)
('Humidity :', 39.8)
('Pressure:', 1025.7)
```

Once you are happy with the data that is being displayed, you can stop the script by pressing the following keys, **Ctrl + C**.

This will terminate the script from running. Of course, you probably don't want to have to be looking at your Raspberry Pi's terminal to be able to get the current data from it.

Instead, we are going to show you two other methods for displaying data, these is just to display the data to the LED matrix.

The other method is to utilize a piece of software called InitialState that will allow us to graph the data, and if you decide to pay for the software, you can also store the data over a period.

Improving your weather station - Utilizing the LED Matrix

1. Changing our weather python script to start displaying its data to the LED matrix is relatively easy. It will involve us concatenating all our prints together into a single string, then issuing the **show_message** command to the Sense HAT library.

Before we get started let's begin editing our weather script by running the following command.

```
sudo nano ~/weather_script.py
```

2. Now that we are in our weather script we can begin making some changes, above the following line we need to add this long line of code. This line of code handles everything in one single line.

Above

```
time.sleep(1)
```

Add

```
sense.show_message("Temperature C" + str(temp) + "Humidity:" + str(humidity) + "Pressure:" + str(pressure), scroll_speed=(0.08), back_colour= [0,0,200])
```

Make sure you have typed out this line of code all onto a single line. This code will make the temperature, humidity, and pressure scroll across the Sense HAT's LED matrix.

We will explain a bit of what we are doing here. We firstly concatenate the temperature onto the end of "Temperature C", however since the temp variable is a number we need to wrap it in **str()** to convert it to a string.

We also decrease the scroll speed to 0.08. We do that with the following part of the code **scroll_speed=(0.08)** you can increase or reduce this number to either speed up or decrease the rate the text scrolls.

The last bit of code we have on this line is back_colour= [0,0,200], this part of the code sets the background color of the text to blue.

You can also change the color of the text itself. We can do this with our example of setting the color of the text to a pinky color by adding the following: **, text_color=[200,0,200]** after **back_color[0,0,200]**.

3. With that line added there is one last thing we will want to add, that being a call to **sense.clear()**. At the bottom of your script after the exception handling and pass, add the following line of code.

```
sense.clear()
```

This code will ensure that the LED matrix is completely cleared if we kill the script for any reason.

This line stops the annoyance of having to deal with a partially turned on LED matrix and has the added benefit of saving a tiny amount of power, but there is no point on leaving something switched on when you are not utilizing it.

It is also good coding practice to make sure you clean-up everything when you stop a script from running. It helps stop things like memory leaks from occurring.

Improving your weather station - LED Matrix - Continued

4. With those changes made your code should look similar to what is displayed below. Remember that the **sense.show_message** code should be contained within a single line.

```
#!/usr/bin/python
from sense_hat import SenseHat
import time
import sys

sense = SenseHat()
sense.clear()

try:
    while True:
        temp = sense.get_temperature()
        temp = round(temp, 1)
        print("Temperature C",temp)

        humidity = sense.get_humidity()
        humidity = round(humidity, 1)
        print("Humidity :",humidity)

        pressure = sense.get_pressure()
        pressure = round(pressure, 1)
        print("Pressure:",pressure)

        sense.show_message("Temperature C" + str(temp) + "Humidity:" + str(humidity)
+ "Pressure:" + str(pressure), scroll_speed=(0.08), back_colour= [0,0,200])

        time.sleep(1)
except KeyboardInterrupt:
    pass

sense.clear()
```

Once your code looks something like the one displayed above, and you are certain you have correctly indented your code you can quit and save, press **Ctrl+X** and then press **Y** and then press **Enter**.

5. Now we can test our modified script to make sure that it functions correctly. Run the following command in terminal to run the script.

```
sudo python ~/weather_script.py
```

The text should now begin to scroll across the Sense HAT's LED matrix. Of course, as you will quickly notice, this isn't the easiest way of viewing all the sensors data.

The reason for this is mainly due to the small nature of the LED matrix making the text more difficult to read and only being able to display a single letter at one time.

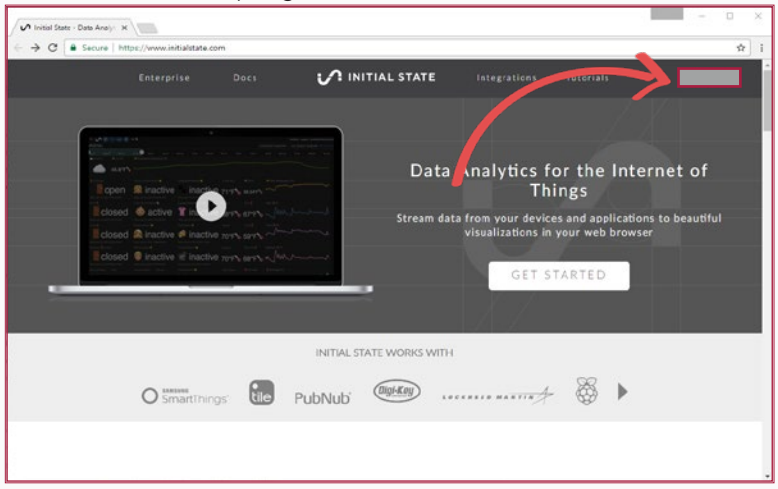
A more helpful way of displaying your data is by utilizing a piece of software such as Initial State. We will go into setting up Initial state and getting our script to send data to it on the next page.

Improving your weather station - Initial State

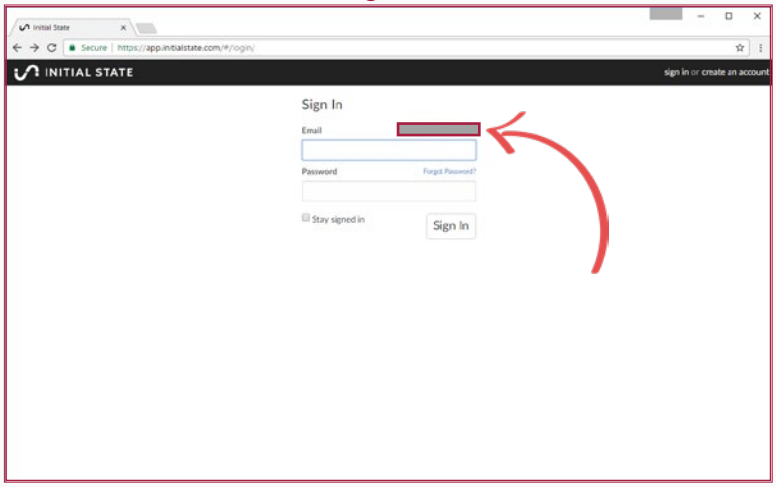
1. For those who don't know, Initial State is a website designed to act as a sort of data storage and data analytics site for the Internet of Things devices like the Raspberry Pi.

Before we get started implementing everything on your Raspberry Pi, we will first have to sign up for a free account on https://www.initialstate.com/

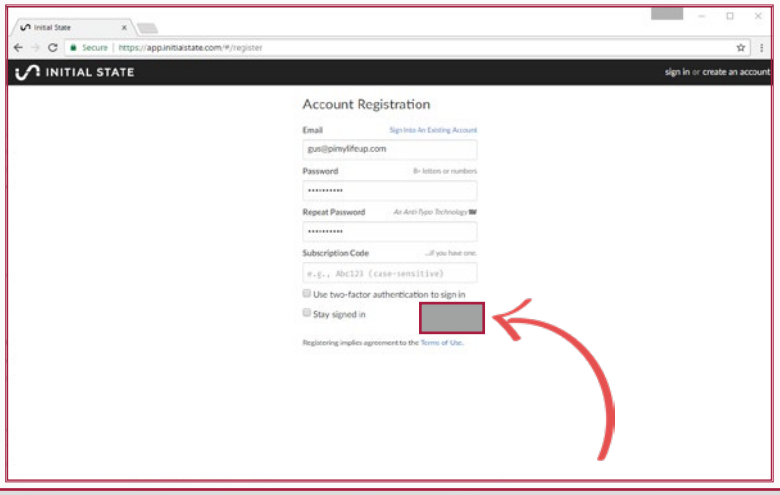
Once on the site, go to "**SIGN IN**", in the top right-hand corner of the screen.



2. Now on the sign in screen, we need click on "**Register A New Account**".

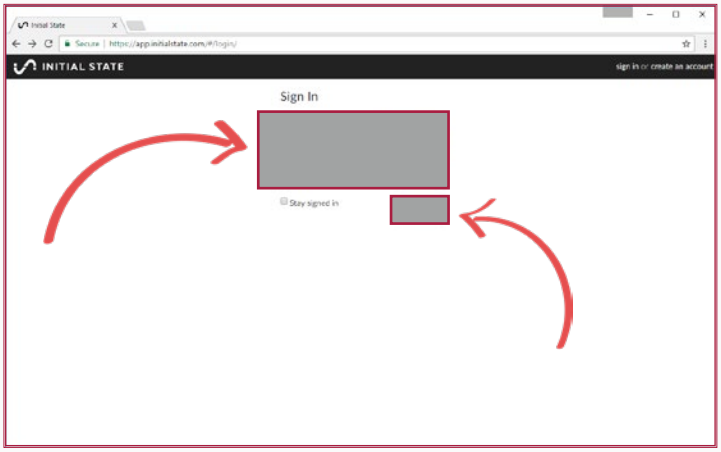


3. On here sign up your new user, make sure you set a strong password that you can remember as you will need this to get your API key and view your data.

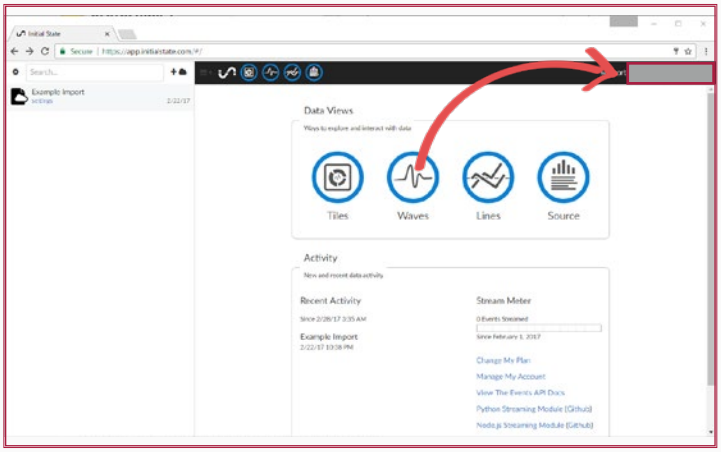


Improving your weather station - Initial State - Continued

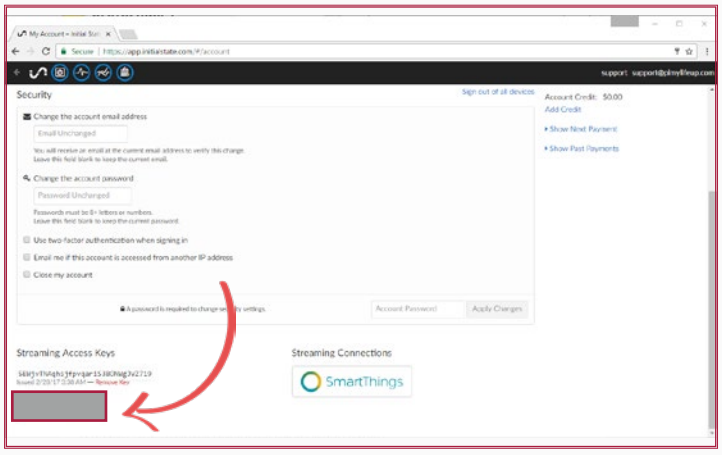
4. Now with your new user, go back to the sign in page and sign into your new Initial State user, we will need to grab the API key from here before we get started with our script.



5. You should now be greeted by an empty dashboard screen like below, to get to the screen where you can generate an API key you will need to click on your email in the top right-hand corner.



6. Now that we are on our account screen we need to scroll all the way to the bottom and look for "**Streaming Access Keys**", underneath here you should see a "**Create a New Key**" Button, press it. Once the key is generated write it down somewhere safe, as you will need this for our script.



Improving your weather station - Initial State - Continued

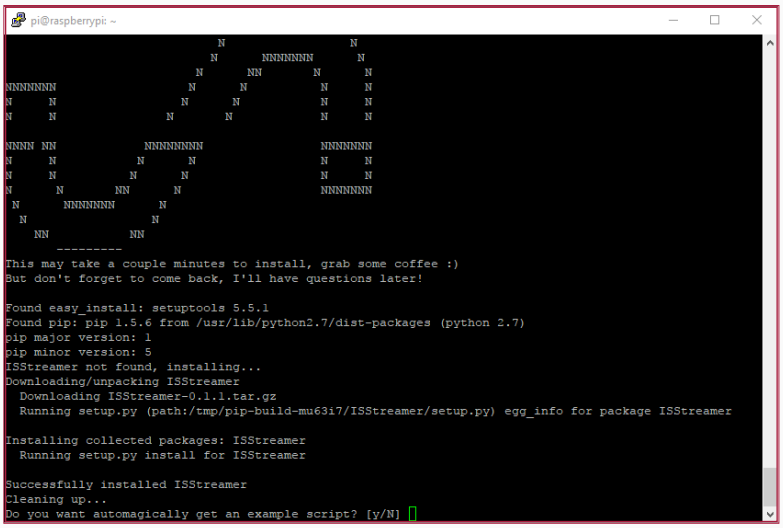
7. We can now get on with installing the Initial State python streamer, to do this we are going to grab there install script directly from their website.

While it is not ordinarily safe to run a script through CURL, Initial State is a trusted site. However, if you want to check the script yourself, you can view it by going to the following URL: https://get.initialstate.com/python

If you are okay with running the script then enter this command into the terminal:

```
curl -sSL https://get.initialstate.com/python -o - | sudo bash
```

8. Eventually, you will be greeted with the following prompt, press **N** to skip having to download the example code. We will not need it for our tutorial.



9. Now we have the streamer installed. We can begin editing our previous weather script. Run the following command to start editing the file:

```
sudo nano ~/weather_script.py
```

10. Now we will have first to make a couple of code additions to this file. Add the following lines to the file.

Below

```
import sys
```

Add

```
from ISStreamer.Streamer import Streamer
```

This line just imports the Initial State streamer package. This package will allow us to make a connection to their website and stream our data to it.

Below

```
sense = SenseHat()
```

Add

```
logger = Streamer(bucket_name="Sense Hat Sensor Data", access_key="YOUR_KEY_HERE")
```

This line of code creates the Streamer and initializes a connection with Initial Senses servers, make sure you replace **YOUR_KEY_HERE** with the key you got in **step 6**.

Improving your weather station - Initial State - Continued

Replace

```
print(
```

With

```
logger.log(
```

On this step, you need to replace all occurrences of **print(** with **logger.log(**, this will make the script log all your sensor data to Initial State's website. Luckily for us, the log method of the Streamer object follows the same parameters as print. So, we can just swap out the functions.

Your code should now look like something similar to what is displayed below.

```
#!/usr/bin/python
from sense_hat import SenseHat
import time
import sys
from ISStreamer.Streamer import Streamer

sense = SenseHat()
logger = Streamer(bucket_name="Sense Hat Sensor Data", access_key="YOUR_KEY_
HERE")
sense.clear()

try:
    while True:
        temp = sense.get_temperature()
        temp = round(temp, 1)
        logger.log("Temperature C",temp)

        humidity = sense.get_humidity()
        humidity = round(humidity, 1)
        logger.log("Humidity :",humidity)

        pressure = sense.get_pressure()
        pressure = round(pressure, 1)
        logger.log("Pressure:",pressure)

        time.sleep(1)
except KeyboardInterrupt:
    pass
```

Once your code looks something like the one displayed above and you are certain you have correctly indented your code you can quit and save, press **Ctrl+X** and then press **Y** and the press **enter**.

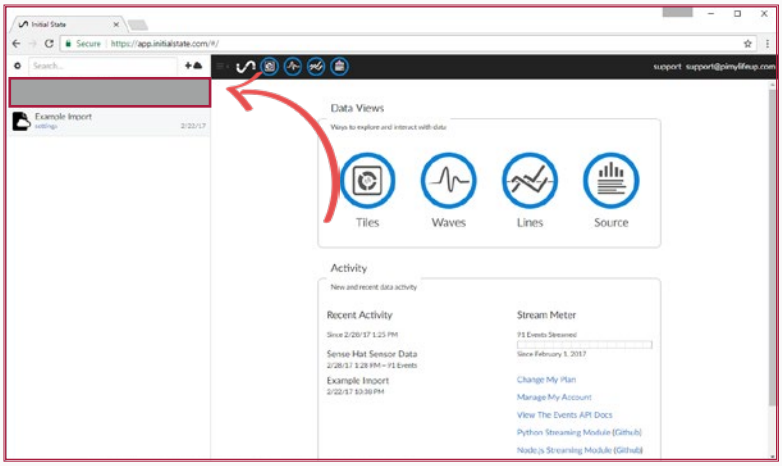
11. Now we have made those changes we can finally run our updated script by running the following command.

```
sudo python ~/weather_script.py
```

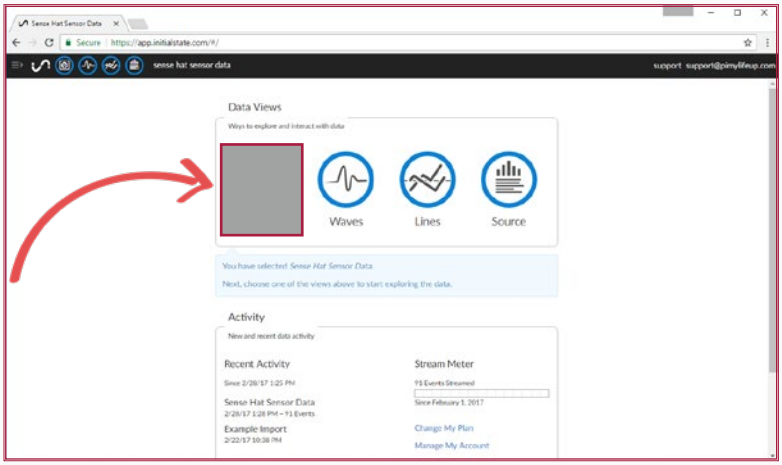
The script will immediately start to send your data to the website.

Improving your weather station - Initial State - Continued

12. Now going back to your Initial State dashboard, you should now see your new data set on the left-hand side, click it.

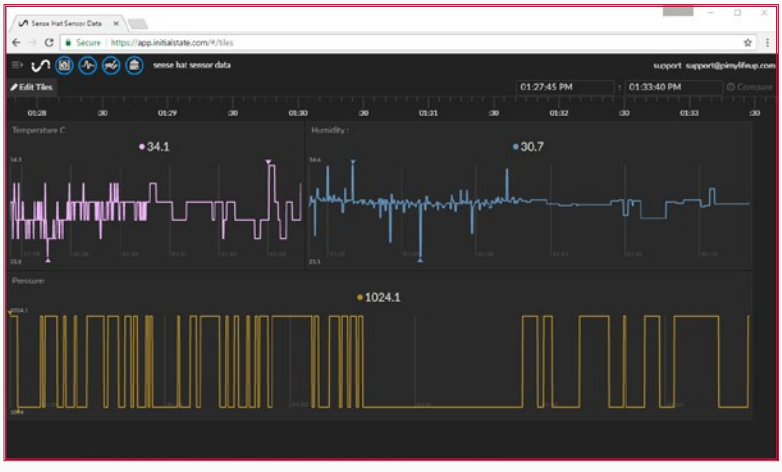


13. With the dataset selected we can now choose how we want to view the data, for this tutorial we will just select "Tiles" as this is the most versatile option.



14. You should now be greeted with a display like below. This screen will continually update itself with the new values as they are sent from your Raspberry Pi.

If you want to change the graph that is being displayed for any of the values, you can right click on them and click on "**Configure Tile**".



Improving your weather station - Start script at startup

1. Before we set up our script, we first need to install an additional package we may have to rely on. This package is dos2unix. This package converts DOS-style line endings into something that is Unix friendly.

Run the following command in terminal to install the package.

```
sudo apt-get install dos2unix
```

2. To set up our python script as a service, so it starts on boot, we will need to write up a short script. The advantage of this script is that we will be able to stop it and force a restart easily.

To begin let's start writing our new script by running the following command:

```
sudo nano /etc/init.d/weatherstation
```

Now write the following into the file, this is quite a bit of code.

```
#!/bin/bash
### BEGIN INIT INFO
# Provides:          weatherstation
# Required-Start:
# Required-Stop:
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Start/stops the weatherstation
# Description:       Start/stops the weatherstation
### END INIT INFO

DIR=/home/pi
DAEMON=$DIR/weather_script.py
DAEMON_NAME=weatherstation

DAEMON_USER=root

PIDFILE=/var/run/$DAEMON_NAME.pid

. /lib/lsb/init-functions

do_start () {
    log_daemon_msg "Starting system $DAEMON_NAME daemon"
    start-stop-daemon --start --background --pidfile $PIDFILE --make-pidfile --user $DAEMON_USER --chuid $DAEMON_USER --startas $DAEMON
    log_end_msg $?
}
do_stop () {
    log_daemon_msg "Stopping system $DAEMON_NAME daemon"
    start-stop-daemon --stop --pidfile $PIDFILE --retry 10
    log_end_msg $?
}

case "$1" in
    start|stop)
        do_${1}
        ;;

    restart|reload|force-reload)
        do_stop
        do_start
        ;;

    status)
        status_of_proc "$DAEMON_NAME" "$DAEMON" && exit 0 || exit $?
        ;;
    *)

        echo "Usage: /etc/init.d/$DAEMON_NAME {start|stop|restart|status}"
        exit 1
        ;;
esac
exit 0
```

Improving your weather station - Start script at startup - Continued

3. Now we have written all that code into the file. We can now save and exit it by pressing **Ctrl + X** then pressing **Y** and then **Enter**.

4. Now we have the file saved there is a few things we will need to do to make sure this will correctly work.

First things first we will run dos2unix on our newly created file. This will ensure the line endings are correct.

```
sudo dos2unix /etc/init.d/weatherstation
```

5. With that done, we need to now change the permissions for our python script. Otherwise our init.d bash script will fail to work. Type the following into the terminal to change its permissions.

```
sudo chmod 755 /home/pi/weather_script.py
```

6. We also need to modify the permissions of our **weatherstation** bash script. We just need to give it execution rights. We do that by entering the following

```
sudo chmod +x /etc/init.d/weatherstation
```

7. Now finally, we just need to create a symbolic link between our bash script and the rc.d folders. We can do that by running the following command in terminal.

```
sudo update-rc.d weatherstation defaults
```

8. Everything should now be setup, and we can now interact with our new bash file like any other service.

To start up our python script, we can just run the following command.

```
sudo service weatherstation start
```

9. Now everything should now be correctly set up. The weatherstation service should now automatically start on boot. We can also stop the script, reload it, and check the status of the script like most services.

Below is a list of commands you can call to interact with the **weatherstation** service.

```
sudo service weatherstation start
```

This command starts up the service which keeps track of your **weather_script.py** file.

```
sudo service weatherstation stop
```

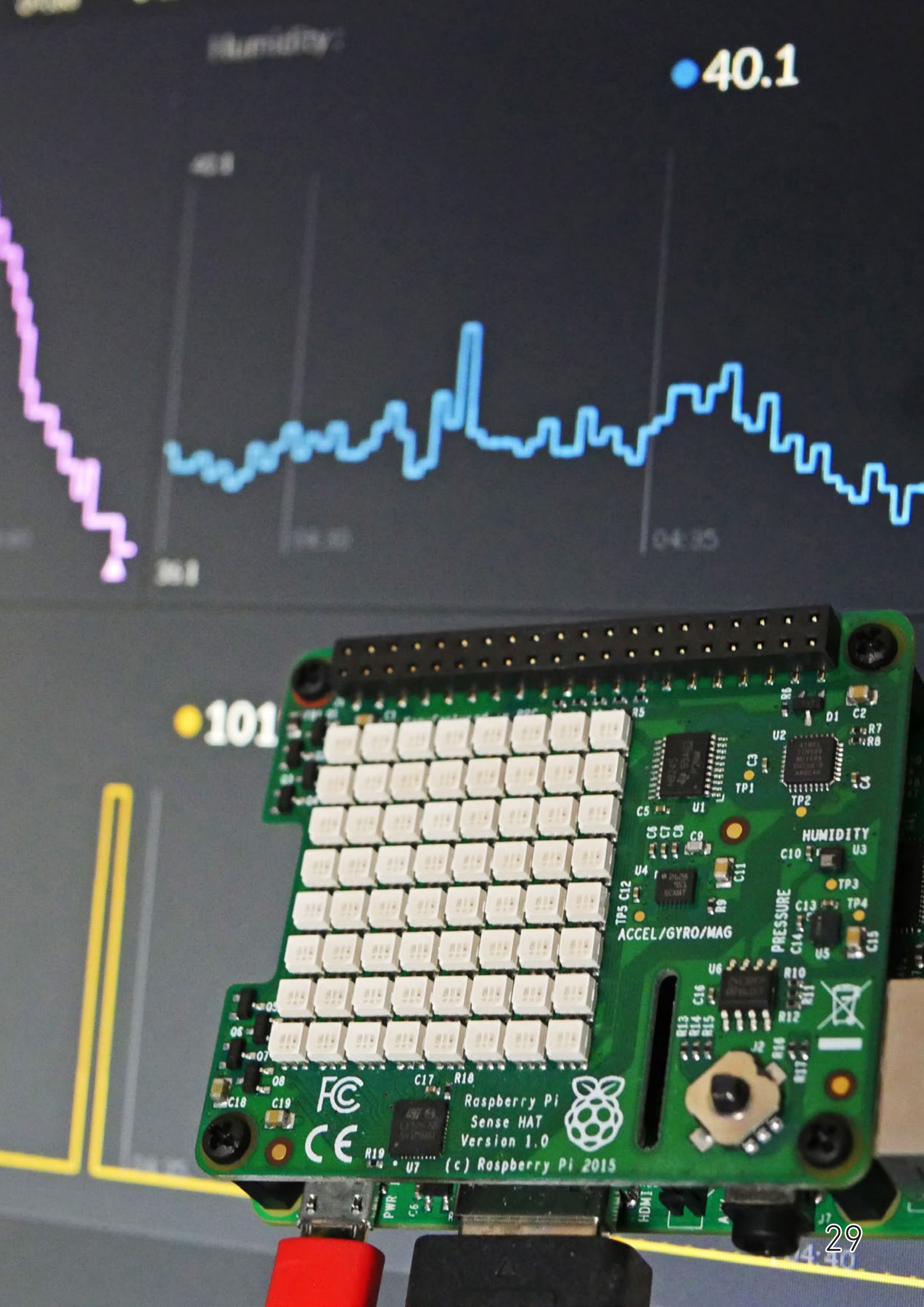
This command stops the **weatherstation** service and kills the process that is running our **weather_script.py**.

```
sudo service weatherstation reload
```

This command reloads **weatherstation** service by killing the process and reloading it.

```
sudo service weatherstation status
```

This command retrieves the status of the **weatherstation** service and our **weather_script.py** script.


40.1
101
HUMIDITY
PRESSURE
ACCEL/GYRO/MAG
Raspberry Pi
Sense HAT
Version 1.0
(c) Raspberry Pi 2015

Sense HAT Email Notifier

Project Description

In this tutorial, we will be setting up a Raspberry Pi Sense HAT Email notifier. This tutorial will make use of the LED Matrix to display a visual notification of the status of your Email. This project is good to use alongside our Sense HAT Weather Station Tutorial if you plan to utilize your Sense HAT inside your house.

This project will utilize an IMAP client to make a remote connection to your email server, make sure the email provider you use fully supports IMAP otherwise this tutorial will not work.

We will write our script to handle this purely in Python. This script allows the project to be quite versatile if you ever decide to increase the projects in functionality in the future.

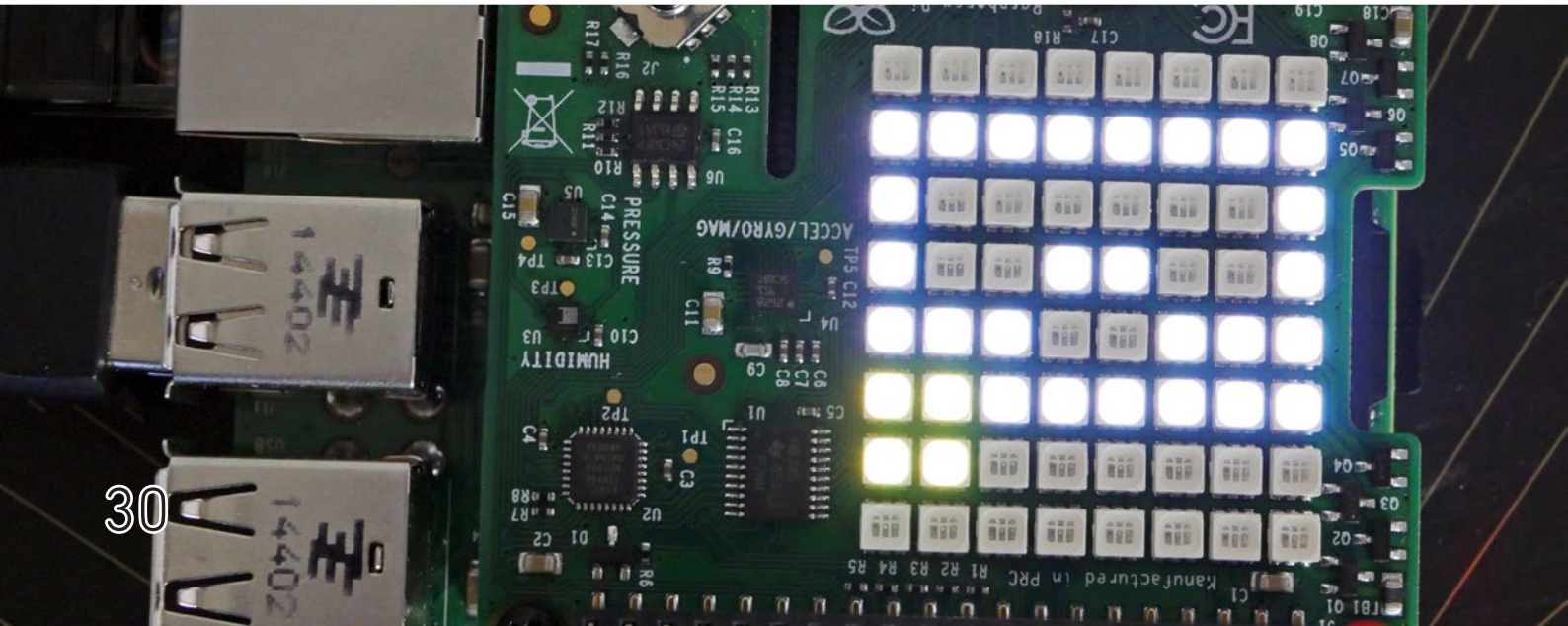
Equipment

Required

- Raspberry Pi
- SD Card (8 GB+ Recommended)
- Sense HAT
- Network Connection

Optional

- Raspberry Pi Case
- USB Keyboard
- USB Mouse



Getting started with the Sense HAT

Before you get started with this tutorial, make sure that you have correctly placed the Sense HAT over the GPIO pins on the Raspberry Pi. It's an incredibly easy installation and shouldn't require any extra fiddling once put on correctly.

1. Before we get started, we need to run the following commands to ensure that the Raspberry Pi is running the latest software.

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

2. With the Raspberry Pi up to date, we now need to install the sensehat software package. This package provides all the libraries we need to interact with the Sense HAT.

```
sudo apt-get install sense-hat
sudo reboot
```

3. Now we have the software installed will write a quick script to ensure the Sense HAT is working correctly.

We can start writing this script with the following command.

```
sudo nano ~/sensehat_test.py
```

4. Now write the following lines into this file, we will explain what each section of code does as we go.

```
from sense_hat import SenseHat
```

This line imports the SenseHat module from the sense_hat library. This library allows us to interact with the sense hat itself through python.

```
sense = SenseHat()
```

This line creates a link to the SenseHat library and initializes itself so we can start making calls to it.

```
sense.show_message("Hello World")
```

This line writes a message to the Sense HAT. You should see "Hello World" scroll across the RGB lights.

Press **Ctrl + X** then **Y** then press **Enter** to save the file.

5. With the file now saved we can run it with the following command:

```
sudo python ~/sensehat_test.py
```

The text "**Hello World**" should now scroll across the RGB Leds on top of the Sense Hat. If it doesn't, it is likely that Sense HAT has not been properly pressed down on top of the GPIO pins.

If it is still not working, try restarting the Raspberry Pi by running the following command.

```
sudo reboot
```

Writing our Email Notification Script

For our python script we will be utilizing the imapclient python package to handle all connections to the mail server, but first, we must remove the default version of PIP.

1. The version of PIP provided with the Raspbian operating system seems to run into issues with installing the **imapclient**, so we will first remove it and install a new version by running the following commands:

```
sudo apt-get remove python-pip
sudo wget https://bootstrap.pypa.io/get-pip.py
sudo python get-pip.py
sudo apt-get install python-pip
```

2. Now we can successfully install the **imapclient** package, run the following two commands in terminal.

```
sudo apt-get install build-essential libssl-dev libffi-dev python-dev
sudo pip install imapclient
```

3. Since we are utilizing a few images for this tutorial, we will start off by making a directory to contain everything.

This folder will make it much neater and easier to deal with then just dumping everything in the home directory.

Let's create this directory and go into it with the following commands:

```
sudo mkdir ~/emailnotifier
cd ~/emailnotifier
```

4. Now let's download and extract the images that we will be using in this tutorial. We will use these images to display a visual notification of your emails.

You can edit these if you want, taking note each image must be 8x8 pixels as each pixel represents an LED on the 8x8 LED Matrix.

Run the following command to download and extract the images into our **emailnotifier** directory.

```
curl -L https://pimylifeup.com/out/sensehatemailimages | sudo tar xvz
```

5. Finally, we can get writing the script itself, create and begin editing the file by running the following command.

```
sudo nano emailnotifier.py
```

Writing our Email Notification Script - Continued

6. Now write the following lines into this file, we will explain what each section of code does as we go.

```
#!/usr/bin/python
from sense_hat import SenseHat
from imapclient import IMAPClient
import time
import sys
```

First, need to import all the libraries that we plan to utilize in our script. In our case, we will be using the following libraries.

sense_hat Library

This library is what we will utilize to interact with the Sense Hat itself, without this we wouldn't be able to read any of the sensor data or interact with the 8x8 LED matrix.

In addition to providing the sensor data the library also provides access to the joystick that is located on the Sense HAT. However, we won't be using that functionality in this tutorial.

imapclient Library

This library is what we utilize to interact with our webmail servers, and it provides the functionality to connect to a web service over the IMAP protocol.

time Library

This library allows us to do a large variety of different time stuff, but for our simple script, we will be just using its sleep functionality.

This function allows us to suspend the current thread for a small period, within this script we use this to control the frequency at which we poll the email server.

sys Library

This library provides us access to some variables and functioned that are managed by the interpreter itself.

In the case of our script, we will be using the functionality provided by this library to terminate the script if we ever need to do so.

Writing our Email Notification Script - Continued

```
HOSTNAME = 'imap.gmail.com'
USERNAME = 'username'
PASSWORD = 'password'
MAILBOX = 'Inbox'
NEWMAIL_OFFSET = 0
MAIL_CHECK_FREQ = 20
```

Here we define the variables that we plan to utilize within our python script. These are crucial to the way the whole program works and helps make it easy to tweak the script.

Make sure you change each of these settings to match your relevant user and email service.

HOSTNAME - This is the variable that will store the address to the IMAP server that we will be connecting to, if you are using **GMAIL** then **imap.google.com** should work if you are using **Outlook (Hotmail)** then use **imap-mail.outlook.com**.

If you are unsure if your service provides the IMAP protocol to connect to, then try googling your service provider and IMAP.

USERNAME - Set this to your email/username for the email provider you intend on using, as an example if I were utilizing Gmail, I would use an email such as **pimylifeup@gmail.com** as the username.

PASSWORD - Set this to the password for the user you specified for the **USERNAME** variable.

MAILBOX - This is the mailbox that you want to check for new emails. This is useful if you want to check a specific folder and not your whole Inbox.

Leave this set to **Inbox** if you don't want to scan a particular folder.

NEWMAIL_OFFSET - You set this variable to the number of emails you want the script to consider it as empty.

For example, if you have 200 unread emails but only want the emails after that considered as new, then set the value to 200.

MAIL_CHECK_FREQ - Set this to how frequently you want to check for new emails, this value is in seconds.

The value of 20 is probably the best value as it's not to often but ensures that most new emails will be picked up quickly.

```
sense = SenseHat()
sense.load_image("hello.png")
```

The first line here creates a link to the SenseHat library and initializes itself so we can start making calls to it.

The second makes a call to the SenseHat library to load in our image named "**Hello.png**" and displays it to the 8x8 LED Matrix.

Writing our Email Notification Script - Continued

```
try:
    server = IMAPClient(HOSTNAME, use_uid=True, ssl=True)
    server.login(USERNAME, PASSWORD)
except:
    connected = False
    sense.load_image("error.png")
else:
    connected = True
    sense.load_image("done.png")
    select_info = server.select_folder(MAILBOX)
```

This block of code does several things, the most important part however if the "**try:**" segment of the code, as this section is what starts the connection to the email servers.

This will create a new IMAPClient Object with our email server's hostname, will we enforce using unique ID's and enforce using SSL for connections.

After that, we make the login connection to the server with our provided username and password. Thanks to this being encased in a try statement, if the connection fails we handle it in the "**except:**" clause.

In this clause, we set our **connected** variable to **False** and load in our error image.

Otherwise, if there is no error, we set the **connected** variable to **True** and load in our done image to the LED matrix. We then also get the **imapclient** to select our specified **MAILBOX** folder.

```
try:
    while connected:
        folder_status = server.folder_status(MAILBOX, 'UNSEEN')
        newmails = int(folder_status['UNSEEN'])
        if newmails > NEWMAIL_OFFSET:
            newmails -= NEWMAIL_OFFSET
            sense.show_message(str(newmails))
            if newmails == 1:
                sense.load_image("mail.png")
            elif newmails > 1 and newmails < 15 :
                sense.load_image("mailFew.png")
            elif newmails > 14:
                sense.load_image("mailLot.png")
        else:
            sense.load_image("nomail.png")
        time.sleep(MAIL_CHECK_FREQ)
except KeyboardInterrupt:
    pass
sense.clear()
```

Our next and final block of code, runs within a try, while loop. The try statement is there to handle breaking out of the while loop if **Ctrl + C** is ever pressed.

The while loop is designed so we can continually run our mail checking. It will only run however if a successful connection to the mail server has been established.

Writing our Email Notification Script - Continued

We then make a call to check the status of the mailbox. This call will return us the amount of unseen/unread emails for that specific mailbox.

We then check to make sure the number of new emails is greater than our offset. We can change new-mails to equal only the amount of new emails after the offset.

Once we have done that, we now make a call to the sense hat library to display the number of new emails to the Raspberry Pi Sense HAT's LED screen.

Next, we check if there is only one new email we display the mailbox with a green box in the top hand corner.

If there is more than one email but less than fourteen we then display an email with an orange box in the top hand corner.

If there are more than fifteen emails, we then display an email with a red box in the top hand corner.

Finally, for that set of if statements we have an "**else**" statement, this handles the case of there being no new emails.

Afterward, we put the script to sleep for the number of seconds defined in our variable, **MAIL_CHECK_FREQ**

Finally, we have our exception. This exception handles **KeyboardInterrupts**, so we cleanly break out of the while loop and run our final bit of code, that being our call to sense.clear() to clear out the LED Matrix.

You should now have some idea on how the code operates, if you want to see the final version of the script, look over onto the next page.

It's a good idea to check it out either way as it is a rather long script.

Writing our Email Notification Script - Continued

```
#!/usr/bin/python
from sense_hat import SenseHat
from imapclient import IMAPClient
import time
import sys

HOSTNAME = 'imap.gmail.com'
USERNAME = 'username'
PASSWORD = 'password'
MAILBOX = 'Inbox'
NEWMAIL_OFFSET = 0
MAIL_CHECK_FREQ = 20

sense = SenseHat()
sense.load_image("hello.png")

try:
    server = IMAPClient(HOSTNAME, use_uid=True, ssl=True)
    server.login(USERNAME, PASSWORD)
except:
    connected = False
    sense.load_image("error.png")
else:
    connected = True
    sense.load_image("done.png")
    select_info = server.select_folder(MAILBOX)

try:
    while connected:
        folder_status = server.folder_status(MAILBOX, 'UNSEEN')
        newmails = int(folder_status['UNSEEN'])
        if newmails > NEWMAIL_OFFSET:
            newmails -= NEWMAIL_OFFSET
            sense.show_message(str(newmails))
            if newmails == 1:
                sense.load_image("mail.png")
            elif newmails > 1 and newmails < 15 :
                sense.load_image("mailFew.png")
            elif newmails > 14:
                sense.load_image("mailLot.png")
        else:
            sense.load_image("nomail.png")
        time.sleep(MAIL_CHECK_FREQ)
except KeyboardInterrupt:
    pass
sense.clear()
```

7. Now we have written all that code into the file. We can now save and exit it by pressing **Ctrl + X** then pressing **Y** and then **Enter**.

Writing our Email Notification Script - Continued

8. With our script finally written, we can now run it. We do this by just typing the following script into the terminal, don't worry about it taking control of the terminal for now.

We will be writing another script to handle running the email notifier as a service.

```
sudo python emailnotifier.py
```

9. You should immediately start to see images displayed on your Raspberry Pi's LED Matrix, while it is loading it should show three different colored boxes.

Once it successfully starts to read your emails, it should display the number of emails that are currently unread (after the threshold amount) and display a visual indicator afterward.

The visual indicator should give you an idea of how many unread emails there are without needing to read the text scrolling across.

If there is a green box then there is only one unread email, a yellow box indicates there is more than one email unread but less than 14 emails. And finally, a red box indicates there are more than 14 emails that are unread.

This task should continually loop and update every 20 seconds by default, unless you have changed the default **MAIL_CHECK_FREQ** value.

If after the original three blocks nothing is displayed and the script terminates, this indicates that the script has failed to make a successful connection to your email server, double check the values you have set for **HOSTNAME**, **USERNAME**, and **PASSWORD**.

If you are running behind two-factor authentication for your mailing service, you may need to generate an application password for that specific service.

10. Now you should have a fully working Raspberry Pi Email Notifier utilizing the Sense HAT. However, if you plan to utilize this daily basis, you won't want to have to run the **emailnotifier.py** python script all the time.

To improve this, we will be writing a bash script so we can utilize our new python script as a service, this will allow us to interact with our script as a service.

This service allows us to start, stop, restart, and check the status of the script. It will also allow us to have the script automatically started on bootup.

If you are interested in setting this up, then you should follow the next couple of pages as they will go into detail how to handle it.

Improving your email notifier - Start script at startup

1. Before we get to set up our script, we first need to install an additional package we may have to rely on. This package is dos2unix. This package converts DOS-style line endings into something that is Unix friendly.

Run the following command in terminal to install the package.

```
sudo apt-get install dos2unix
```

2. To set up our python script as a service, so it starts on boot, we will need to write up a short script. The advantage of this script is that we will be able to stop it and force a restart easily.

To begin let's start writing our new script by running the following command:

```
sudo nano /etc/init.d/emailnotifier
```

Now write the following into the file, this is quite a bit of code.

```
#!/bin/bash
### BEGIN INIT INFO
# Provides:          emailnotifier
# Required-Start:
# Required-Stop:
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Start/stops the emailnotifier
# Description:       Start/stops the emailnotifier
### END INIT INFO

DIR=/home/pi
DAEMON=$DIR/emailnotifier/emailnotifier.py
DAEMON_NAME=emailnotifier

DAEMON_USER=root

PIDFILE=/var/run/$DAEMON_NAME.pid

. /lib/lsb/init-functions

do_start () {
    log_daemon_msg "Starting system $DAEMON_NAME daemon"
    start-stop-daemon --start --background --pidfile $PIDFILE --make-pidfile --user $DAEMON_USER --chuid $DAEMON_USER --startas $DAEMON
    log_end_msg $?
}
do_stop () {
    log_daemon_msg "Stopping system $DAEMON_NAME daemon"
    start-stop-daemon --stop --pidfile $PIDFILE --retry 10
    log_end_msg $?
}

case "$1" in
    start|stop)
        do_${1}
        ;;

    restart|reload|force-reload)
        do_stop
        do_start
        ;;

    status)
        status_of_proc "$DAEMON_NAME" "$DAEMON" && exit 0 || exit $?
        ;;
    *)

        echo "Usage: /etc/init.d/$DAEMON_NAME {start|stop|restart|status}"
        exit 1
        ;;
esac
exit 0
```

Improving your email notifier - Start script at startup - Continued

3. Now we have written all that code into the file. We can now save and exit it by pressing **Ctrl + X** then pressing **Y** and then **Enter**.

4. Now we have the file saved there is a few things we will need to do to make sure this will correctly work.

First things first we will run dos2unix on our newly created file. This will ensure the line endings are correct.

```
sudo dos2unix /etc/init.d/emailnotifier
```

5. With that done, we need to now change the permissions for our python script. Otherwise our init.d bash script will fail to work. Type the following into the terminal to change its permissions.

```
sudo chmod 755 /home/pi/emailnotifier/emailnotifier.py
```

6. We also need to modify the permissions of our email notifier bash script. We just need to give it execution rights. We do that by entering the following

```
sudo chmod +x /etc/init.d/emailnotifier
```

7. Now finally, we just need to create a symbolic link between our bash script, and the rc.d folders. We can do that by running the following command in terminal.

```
sudo update-rc.d emailnotifier defaults
```

8. Everything should now be set up, and we can now interact with our new bash file like any other service.

To start up our python script we can just run the following command.

```
sudo service emailnotifier start
```

9. Now everything should now be correctly set up. The email notifier service should now automatically start on boot. We can also stop the script, reload it, and check the status of the script like most services.

Below is a list of commands you can call to interact with the **emailnotifier** service.

```
sudo service emailnotifier start
```

This command starts up the service which keeps track of your **emailnotifier.py** file.

```
sudo service emailnotifier stop
```

This command stops the **emailnotifier** service and kills the process that is currently running our **emailnotifier.py**.

```
sudo service emailnotifier reload
```

This command reloads **emailnotifier** service by killing the process and reloading it.

```
sudo service emailnotifier status
```

This command retrieves the status of the **emailnotifier** service and our **emailnotifier.py** script.

Raspberry Pi
Sense HAT
Version 1.0
(c) Raspberry Pi 2015
ACCEL/GYRO/MAG
PRESSURE
HUMIDITY
R19
C17
R18
R17
R16
J2
R13
R14
R15
C16
R12
R11
R10
U6
C15
U5
C14
C13
TP4
TP3
U3
C10
TP5
C12
R9
C11
U4
C9
C6
C7
C8
C5
U1
TP2
C4
TP1
C3
R7
R8
U2
D1
C2
R6
R1 R2 R3 R4 R5
nufactured in PRC

Sense HAT
Digital Clock

Project Description

In this tutorial we will be showing you how you can set up your Raspberry Pi Sense HAT as a digital clock by utilizing its LED matrix.

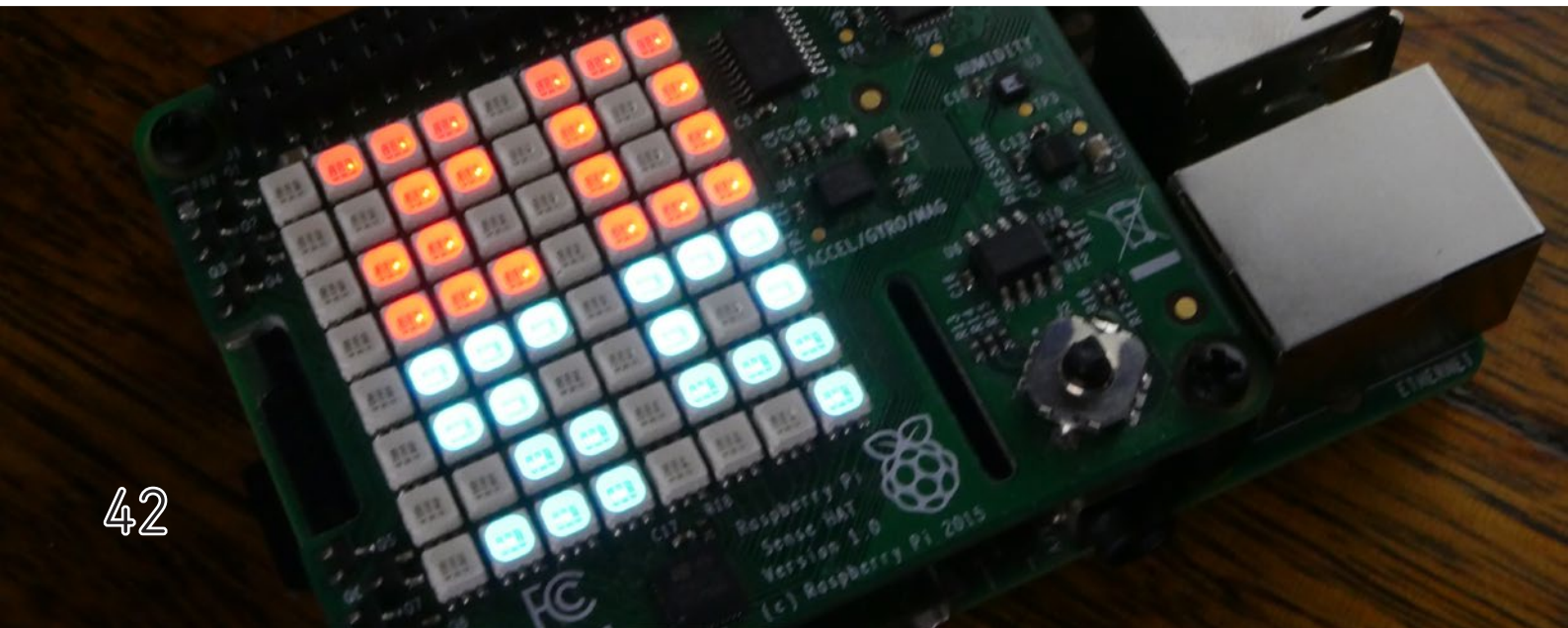
Throughout the tutorial we will be showing you how to write a script on your Raspberry Pi that can grab the current system time and then interpret that into something that we can display on the LED matrix.

This tutorial will teach you how you can utilize arrays and numerical values to print values to the Raspberry Pi's Sense HAT's LED matrix, row by row.

Equipment

Required
- Raspberry Pi
- SD Card (8 GB+ Recommended)
- Sense HAT
- Network Connection

Optional
- Raspberry Pi Case
- USB Keyboard
- USB Mouse



Getting started with the Sense HAT

Now before we start making use of our SenseHAT as a digital clock we must first set it up correctly and test to make sure everything is working.

1. Begin by running the following commands on your Raspberry Pi to update the operating system to the latest version.

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

2. Now that we have updated our Raspberry Pi's operating system we must install the sense-hat package. This package includes all the libraries that we need to interact with the device.

```
sudo apt-get install sense-hat
sudo reboot
```

3. With the sense-hat libaries now download to our Raspberry Pi we will quickly write a script to ensure we can actually talk with the device.

We can start writing this script with the following command on the Raspberry Pi.

```
sudo nano ~/sensehat_test.py
```

4. Now within this script we need to write the following lines of code. We will explain what each section of code does as we go.

```
from sense_hat import SenseHat
```

This line imports the SenseHat module from the sense_hat library. This allows us to interact with the SenseHAT through Python itself.

```
sense = SenseHat()
```

The line instantiates a copy of the SenseHat class to our sense variable. We will be using the sense variable to interact with the SenseHat and call its functions.

```
sense.show_message("Hello World")
```

This line simply pushes a text message to the LED matrix on the SenseHAT. This particular call will make the text "Hello World" appear across the device.

Press **Ctrl + X** then **Y** then press **Enter** to save the file.

5. Now that we have saved the file, run it by using the following command within the terminal.

```
sudo python ~/sensehat_test.py
```

You should now see the text "**Hello World**" scroll across the Led Matrix on the Sense Hat.

If nothing appears on your Sense HAT, it is likely that Sense HAT has not been properly pressed down on top of the GPIO pins, or the device is faulty.

If it is still not working, try restarting the Raspberry Pi by running the following command on it.

```
sudo reboot
```

Turning your SenseHAT into a digital clock

1. Now that we have installed everything that we need for our script and checked that our SenseHat is working we can proceed to actually writing the script.

Before we get to far ahead though, lets make a folder to keep our new script in. We will name this folder digitialclock and keep it in our Pi users home directory.

Run the following command in terminal to create this folder.

```
mkdir ~/digitalclock
```

2. With the folder now created we will change into the directory and begin writing our script by running the following two commands.

```
cd ~/digitalclock
nano digitalclock.py
```

3. Now write the following lines of code to the file. We will explain each section of the code so you get a good understanding of how everything works.

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

from sense_hat import SenseHat
import time

sense = SenseHat()
```

The very first line tells the operating system how to interpret the file, in our case we are using it to tell the operating system to use Python to run the file.

The next line imports the SenseHat module from the sense_hat library, this is the library that we installed during the first section of this tutorial.

We utilize the SenseHat module within this script to talk with the SenseHat device and display our time to the 8x8 LED matrix by switching on specific LEDs on the matrix.

On the next line we import another library, this time we import the time library. We utilize the time library to retrieve the current time directly from the operating system itself.

By getting these values we can choose from our arrays to decide what pixels need to be switched on or off.

Next we create a copy of the SenseHat object, we will be utilizing this object throughout the script to control the SenseHat.

We really only need this to be able to pass our array to the device that tells it which pixels need to be switched on or off.

You will see in the next couple of sections of code how we utilize these libraries.

```
number = [
[[0,1,1,1], # Zero
[0,1,0,1],
[0,1,0,1],
[0,1,1,1]],
[[0,0,1,0], # One
[0,1,1,0],
[0,0,1,0],
[0,1,1,1]],
[[0,1,1,1], # Two
[0,0,1,1],
[0,1,1,0],
[0,1,1,1]],
[[0,1,1,1], # Three
[0,0,1,1],
[0,0,1,1],
[0,1,1,1]],
[[0,1,0,1], # Four
[0,1,1,1],
[0,0,0,1],
[0,0,0,1]],
[[0,1,1,1], # Five
[0,1,1,0],
[0,0,1,1],
[0,1,1,1]],
[[0,1,0,0], # Six
[0,1,1,1],
[0,1,0,1],
[0,1,1,1]],
[[0,1,1,1], # Seven
[0,0,0,1],
[0,0,1,0],
[0,1,0,0]],
[[0,1,1,1], # Eight
[0,1,1,1],
[0,1,1,1],
[0,1,1,1]],
[[0,1,1,1], # Nine
[0,1,0,1],
[0,1,1,1],
[0,0,0,1]]
]
```

This very large array is what we will be using to grab the numbers for our clock. Each number for the clock is split into its own list, then is spit again into 4 individual sections.

We split the array into these lists to make it easier to output to the LED Matrix. You will see shortly he we utilize our large array to push these numbers to the device.

```
noNumber = [0,0,0,0]
```

We also have a small array/list that just has 4 zero values. We will use this list to fill in spots in the LED matrix where we want no number to be displayed.

```
hourColor = [255,0,0] # Red
minuteColor = [0,255,255] # Cyan
empty = [0,0,0] # Black/Off
```

These three variables are used to declare the RGB values of the LED's for the hour, minute and empty slots.

If you want to change the colour for the hour for instance, you just need to change the 3 values of the array to the relevant RGB value.

```
clockImage = []

hour = time.localtime().tm_hour
minute = time.localtime().tm_min
```

In this section of code we first create an empty array/list. We need this empty as we will be pushing all our numbers from our large array to it.

Next we retrieve the current hour and minute and store it in our variables. We will be utilising these to decide what number needs ot be displayed.

```
for index in range(0, 4):
    if (hour >= 10):
        clockImage.extend(number[int(hour/10)][index])
    else:
        clockImage.extend(noNumber)
    clockImage.extend(number[int(hour%10)][index])

for index in range(0, 4):
    clockImage.extend(number[int(minute/10)][index])
    clockImage.extend(number[int(minute%10)][index])
```

Here is where we do most of our magic. We run two different for loops, the first loop is for the top hour numbers, the second loop handles the minutes that are displayed on the bottom of the LED matrix.

For each loop we use Pythons **.extend** function for lists, using this we insert a row of 4 elements from each number in a single call, this saves us having to run two concurrent loops.

To get the correct number from our array we make use of some simple maths. In a 2 digit number, we use a division by 10 to retrieve the first number. For instance 12, we would retrieve 1 from a division by 10.

To get the second digit we make use of modulus which returns us the remainder from a division, so in the case of our previous number (12) we will get the return of 2.

```
    for index in range(0, 64):
        if (clockImage[index]):
            if index < 32:
                clockImage[index] = hourColor
            else:
                clockImage[index] = minuteColor
        else:
            clockImage[index] = empty
```

Now our final loop goes through the entire clockImage array, this loop is simply designed to swap out every number with our RGB values.

We achieve this by just checking whether there is a number 0 or 1 in that index. If its 0 we just output our empty variables value to that number.

Otherwise we check to see if we are dealing with the first 32 pixels (the top half of the LED matrix), if we are in the top height we grab the RGB values from our hourColour variable, otherwise we utilize the RGB values from the minuteColour variable.

```
    sense.set_rotation(90) # Optional
    sense.low_light = True # Optional
    sense.set_pixels(clockImage)
```

Now here are our final 3 lines of code.

The first of these lines sets the rotation of the LED matrix, you can change this based on which direction you want your digital clock to be displayed.

The second line sets the low light mode on the SenseHAT on, this basically dims down the LED's so they aren't nearly as bright. Comment out or remove this line if you have your SenseHAT in an area thats in constant bright light.

Finally the last line outputs our **clockImage** array to the SenseHAT, this is the line that finally displays our time on the SenseHAT's led matrix.

Over the next couple of pages you can compare your code to see if you have made any mistakes. If you are sure what you have is correct you can skip on to **step 4** of this tutorial.

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

from sense_hat import SenseHat
import time

sense = SenseHat()

number = [
[[0,1,1,1], # Zero
[0,1,0,1],
[0,1,0,1],
[0,1,1,1]],
[[0,0,1,0], # One
[0,1,1,0],
[0,0,1,0],
[0,1,1,1]],
[[0,1,1,1], # Two
[0,0,1,1],
[0,1,1,0],
[0,1,1,1]],
[[0,1,1,1], # Three
[0,0,1,1],
[0,0,1,1],
[0,1,1,1]],
[[0,1,0,1], # Four
[0,1,1,1],
[0,0,0,1],
[0,0,0,1]],
[[0,1,1,1], # Five
[0,1,1,0],
[0,0,1,1],
[0,1,1,1]],
[[0,1,0,0], # Six
[0,1,1,1],
[0,1,0,1],
[0,1,1,1]],
[[0,1,1,1], # Seven
[0,0,0,1],
[0,0,1,0],
[0,1,0,0]],
[[0,1,1,1], # Eight
[0,1,1,1],
[0,1,1,1],
[0,1,1,1]],
[[0,1,1,1], # Nine
[0,1,0,1],
[0,1,1,1],
[0,0,0,1]]
]
```

```
noNumber = [0,0,0,0]

hourColor = [255,0,0] # Red
minuteColor = [0,255,255] # Cyan
empty = [0,0,0] # Black/Off

clockImage = []

hour = time.localtime().tm_hour
minute = time.localtime().tm_min

for index in range(0, 4):
    if (hour >= 10):
        clockImage.extend(number[int(hour/10)][index])
    else:
        clockImage.extend(noNumber)
    clockImage.extend(number[int(hour%10)][index])

for index in range(0, 4):
    clockImage.extend(number[int(minute/10)][index])
    clockImage.extend(number[int(minute%10)][index])

for index in range(0, 64):
    if (clockImage[index]):
        if index < 32:
            clockImage[index] = hourColor
        else:
            clockImage[index] = minuteColor
    else:
        clockImage[index] = empty

sense.set_rotation(90) # Optional
sense.low_light = True # Optional
sense.set_pixels(clockImage)
```

4. Once you are happy with all the code, you can save the file by pressing **CTRL + X** then **Y** and finally **ENTER**.

5. Now we can go ahead and test run the code, just run the following command on your Raspberry Pi.

```
python ~/digitalclock/digitalclock.py
```

If everything is running correctly you should now see the current time appear on your device.

Of course having a clock that never actually updates is kind of a useless, in the next section we will show you how to make use of crontab to run the script every minute.

Automating your SenseHAT digital clock

1. The first thing we must do before we start adding our script to the crontab is make it executable.

We can do this easily by using the following command on the Raspberry Pi, this will use chmod to add exectuion rights to the script.

```
chmod +x ~/digitalclock/digitalclock.py
```

2. Now that we have given our digital clock script excecution privlidges we can now go ahead and begin editing the crontab.

Run the following command on your Raspberry Pi to edit the crontab.

```
sudo crontab -e
```

If you are asked what editor you want to utilize to edit the crontab file, we reccomend that you choose Nano as its one of the most straightforward editors to use.

3. To this file add the following line. This line will basically tell the operating system that it needs to run this file every minute

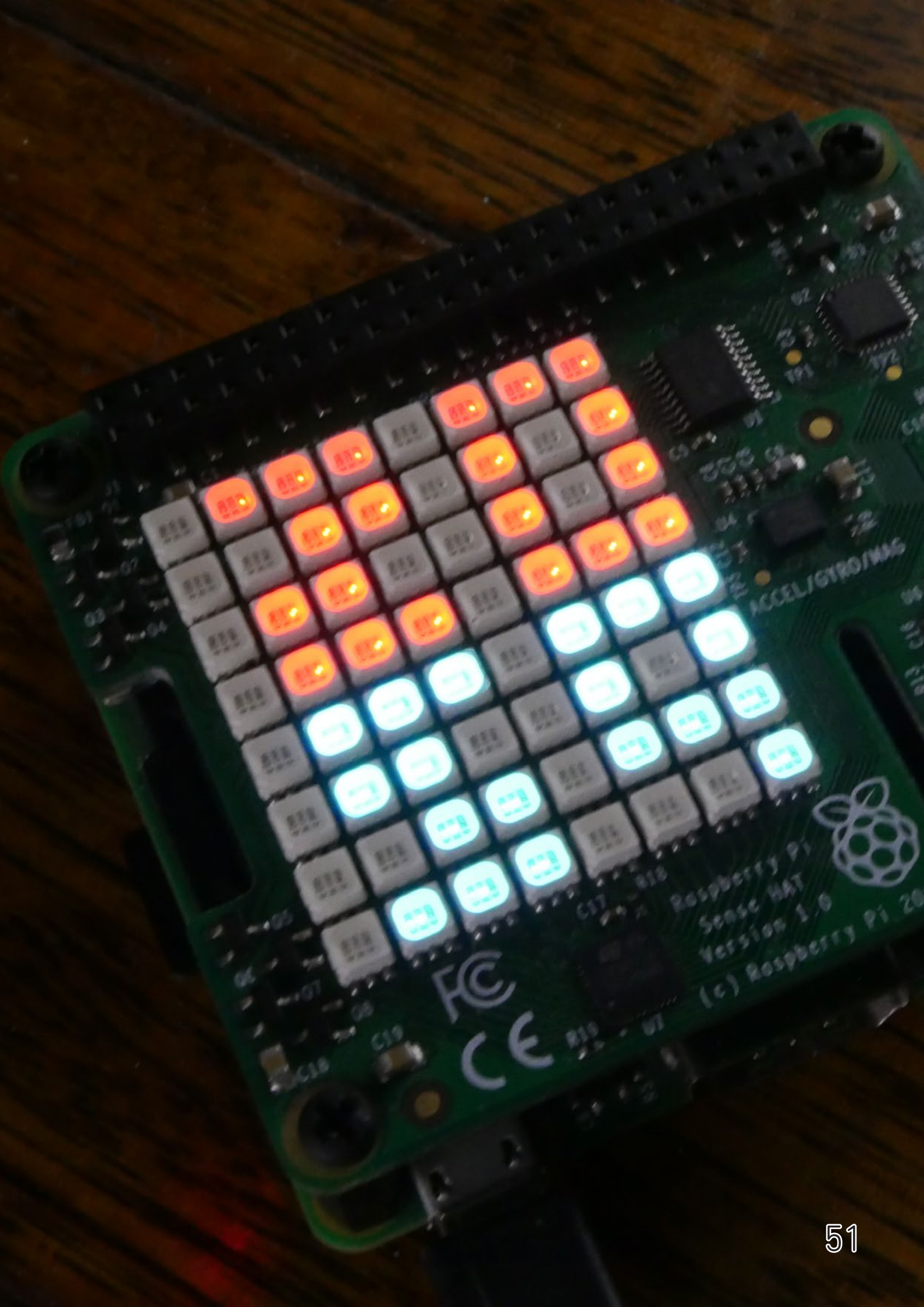
If you want to adjust how often the digital clock updates we recommend using a service such as Crontab Guru, you can find there website at https://crontab.guru/.

```
* * * * * /home/pi/digitalclock/digitalclock.py
```

4. With that change made to the file, you can save it by pressing **CTRL + X** then **Y** and finally **ENTER**.

The system will then automatically load in the new crontab and begin processing it.

5. You should now see the time on your SenseHAT update every minute.


ACCEL/GYRO/MAG
Raspberry Pi
Sense HAT
Version 1.0
(c) Raspberry Pi

Setting up a Light Sensor Circuit

Project Description

In this Raspberry Pi light sensor tutorial, I show you how to correctly connect the light dependent resistor (LDR) sensor up to the GPIO pins. We will also show you how it can be used in a simple python script, so you're able to gather and use the data from the sensor.

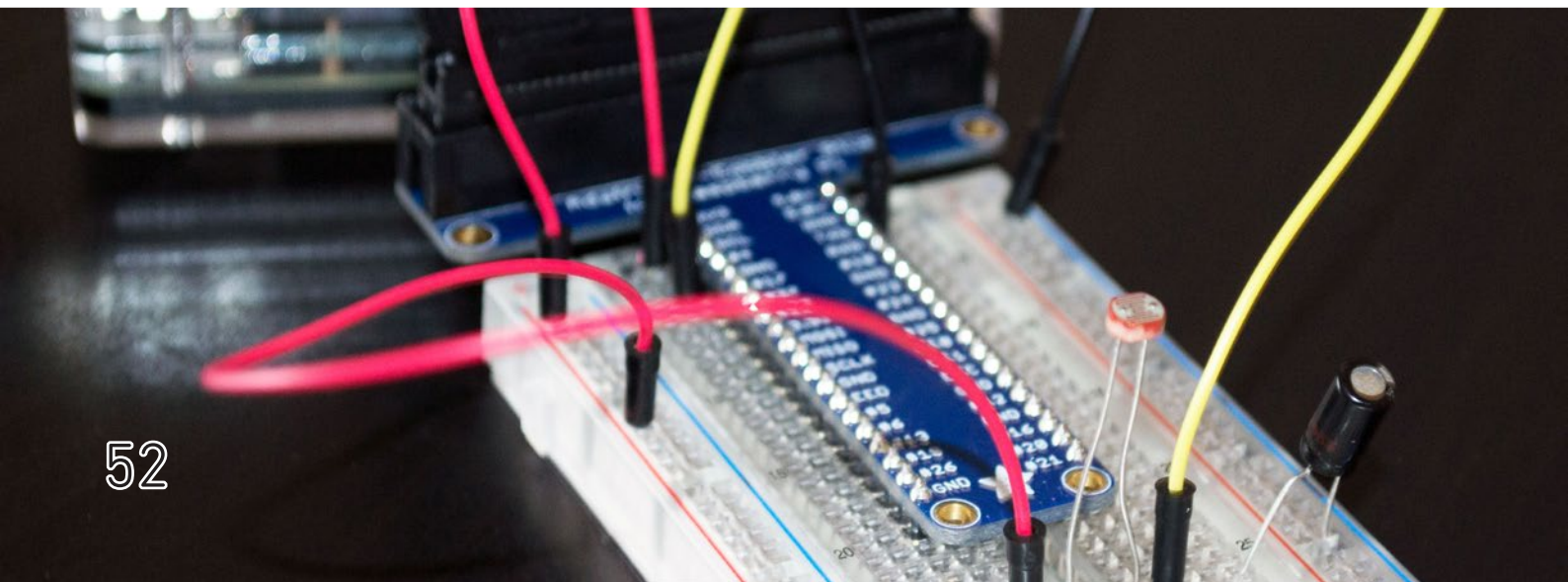
I explain a bit further down each of the parts that I will be using in this circuit. But be sure to read up on any of the components if you are unsure of what their purpose is after I explained it.

It is important to note we are just using a simple photocell sensor while these are perfect for some tasks they might not be as accurate as you would like.

Equipment

Required
- Raspberry Pi
- SD Card (8 GB+ Recommended)
- Network Connection
- Light Sensor (LDR Sensor)
- 1 uF Capacitor

Optional
- Raspberry Pi Case
- GPIO Breakout Kit
- Breadboard (Reccomended)
- Breadboard Wire



The Raspberry Pi Light Sensor Circuit

The **light dependent resistor** which is also referred to as an **LDR sensor** is the most important piece of equipment in our circuit. Without it, we cant detect whether it is dark or light.

In the light, this sensor will have a resistance of only a few hundred ohms while in the dark it can have a resistance of several mega ohms.

The capacitor in our circuit is there to measure the resistance of the LDR. A capacitor acts like a battery charging up while receiving power and then discharging when no longer receiving power.

Since we are using this in series with the LDR, we can work out how much resistance the LDR is giving out thus whether it is light or dark.

To get the light sensor circuit built correctly follow the steps below or check out the circuit diagram right underneath the steps.

In the following steps, I am referring to the **physical numbers** of the **pins** (**Logical order**).

1. First connect **pin #1 (3v3)** to the **positive rail** on the breadboard.

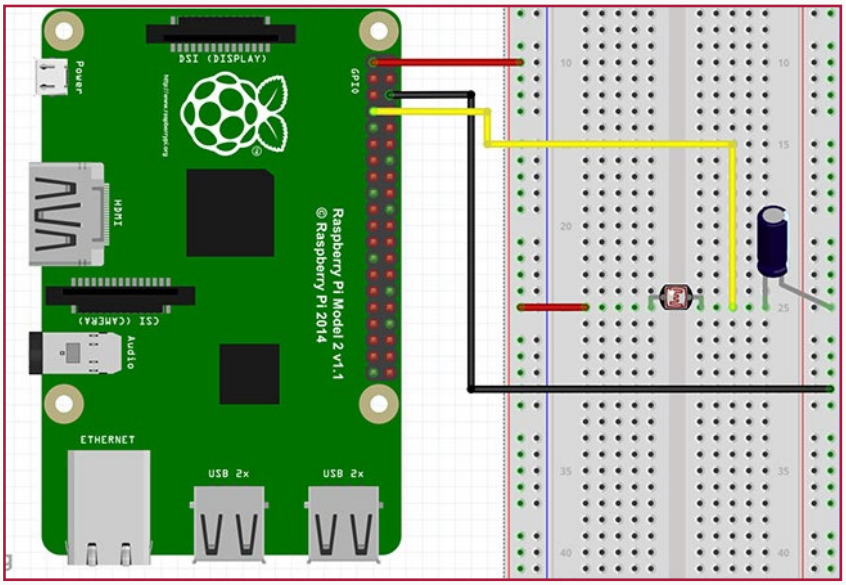
2. Next connect **pin #6 (ground)** to the **ground rail** on the breadboard.

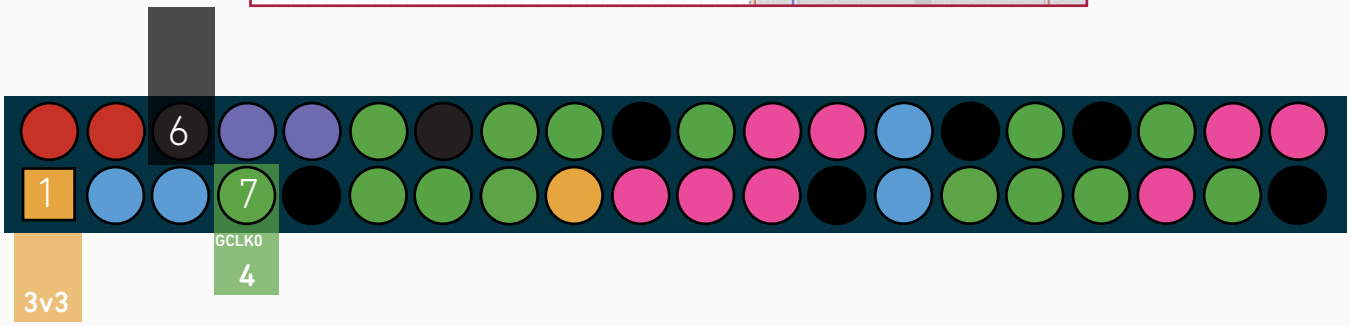
3. Now place the **LDR sensor** on the board and have a wire go from one end to the **positive rail**.

4. On the other side of the **LDR sensor** place a wire leading **back to the Raspberry Pi**. Hook this to **pin #7**.

5. Finally, place the capacitor from the wire to the negative rail on the breadboard. Make sure you have the **negative pin** of the **capacitor** in the **negative rail**.

If you have any trouble with the circuit refer to the diagrams below.





The Code

The code for this project is simple and will tell us roughly whether it is light, shady, or completely dark.

The most significant problem we face with this circuit is the fact that the Raspberry Pi doesn't have any analog pins. All the pins require a digital input (1 or 0), so we can't accurately measure the variance in resistance on our input.

This wasn't a problem in the motion sensor tutorial since the output from it was either high or low (1 or 0).

To utilize the LDR sensor, we will measure the time it takes for the capacitor to charge and send the pin high. Using a capacitor is an easy but inaccurate way of telling whether it is light or dark.

To begin our script, we need first to import the **RPi.GPIO** package. This package provides the functionality we will need so that we can communicate with the GPIO pins and control them.

We also need to import the **time** package, so we're able to put the script to sleep for when we need to.

```
#!/usr/local/bin/python

import RPi.GPIO as GPIO
import time
```

We then need to set the **GPIO mode** to **GPIO.BOARD** this means all the numbering we use in this script will refer to the **physical numbering of the pins**. **GPIO.BOARD** is a little easier to deal with then **GPIO.BCM**, especially if you want to use your code for the original versions of the Raspberry Pi. as well

Since we only have the **one input and output pin** we just need to define **one variable** to keep track of it. Set this variable to the **number of the pin** you have acting as the input and output pin. If you have been following along with this tutorial, then that pin will be **pin 7**.

```
GPIO.setmode(GPIO.BOARD)

#define the pin that goes to the circuit
pin_to_circuit = 7
```

The Code - Continued

Next, we define our function called **rc_time**. This function is designed to require one parameter that we define as **pin_to_circuit**. This variable will have the **pin number** that wants to interact with passed into it.

Within this function, we initialize a variable called **count**. This variable will give an idea of how long it takes for the **GPIO input** to start returning **GPIO.HIGH**. We return this value once the function finishes running.

We then set the pin passed into the function to act as an **output** and then set the output to the pin **low**, we then have the script sleep for 10ms before proceeding.

After this, we then need to set the pin to become an **input** so we can start receiving values from the capacitor. We will need to track the values from the **input** within our while loop.

We stay in this loop until the pin goes to **high**, this is when the **capacitor charges** to about **3/4**.

Once the **input** goes **high**, we leave the loop return the **count** value to the **main function**. You can use this value to turn on and off an LED, activate something else or just log the data and keep statistics on any variance in light.

For our example script, we will just continually call our **rc_time** function in an infinite while loop and print out the values received back from our function.

```
def rc_time (pin_to_circuit):
    count = 0

    #Output on the pin for
    GPIO.setup(pin_to_circuit, GPIO.OUT)
    GPIO.output(pin_to_circuit, GPIO.LOW)
    time.sleep(0.1)

    #Change the pin back to input
    GPIO.setup(pin_to_circuit, GPIO.IN)

    #Count until the pin goes high
    while (GPIO.input(pin_to_circuit) == GPIO.LOW):
        count += 1

    return count

#Catch when script is interrupted, cleanup correctly
try:
    # Main loop
    while True:
        print rc_time(pin_to_circuit)
except KeyboardInterrupt:
    pass
finally:
    GPIO.cleanup()
```

Deploying & Running the Code on your Raspberry Pi

This step is incredibly easy, but we will quickly go through the steps so you can have it up and running on your Pi as quickly and smoothly as possible.

While all the software packages should already be installed in some cases, it may not be. If you want to learn more about the GPIO pins and how to update/install the software, then be sure to check out the tutorial in our "**getting started**" section of the book.

Now you have two choices for setting up the script, one of those options is to clone the code for the Light Sensor directly from our GIT page.

You can clone the script by running the following two simple lines:

```
git clone https://github.com/pimylifeup/Light_Sensor.git
cd ./Light_Sensor
```

Alternatively, you can also write out the code we discussed over the last two pages into the python script yourself. We can start creating this file by running the following command:

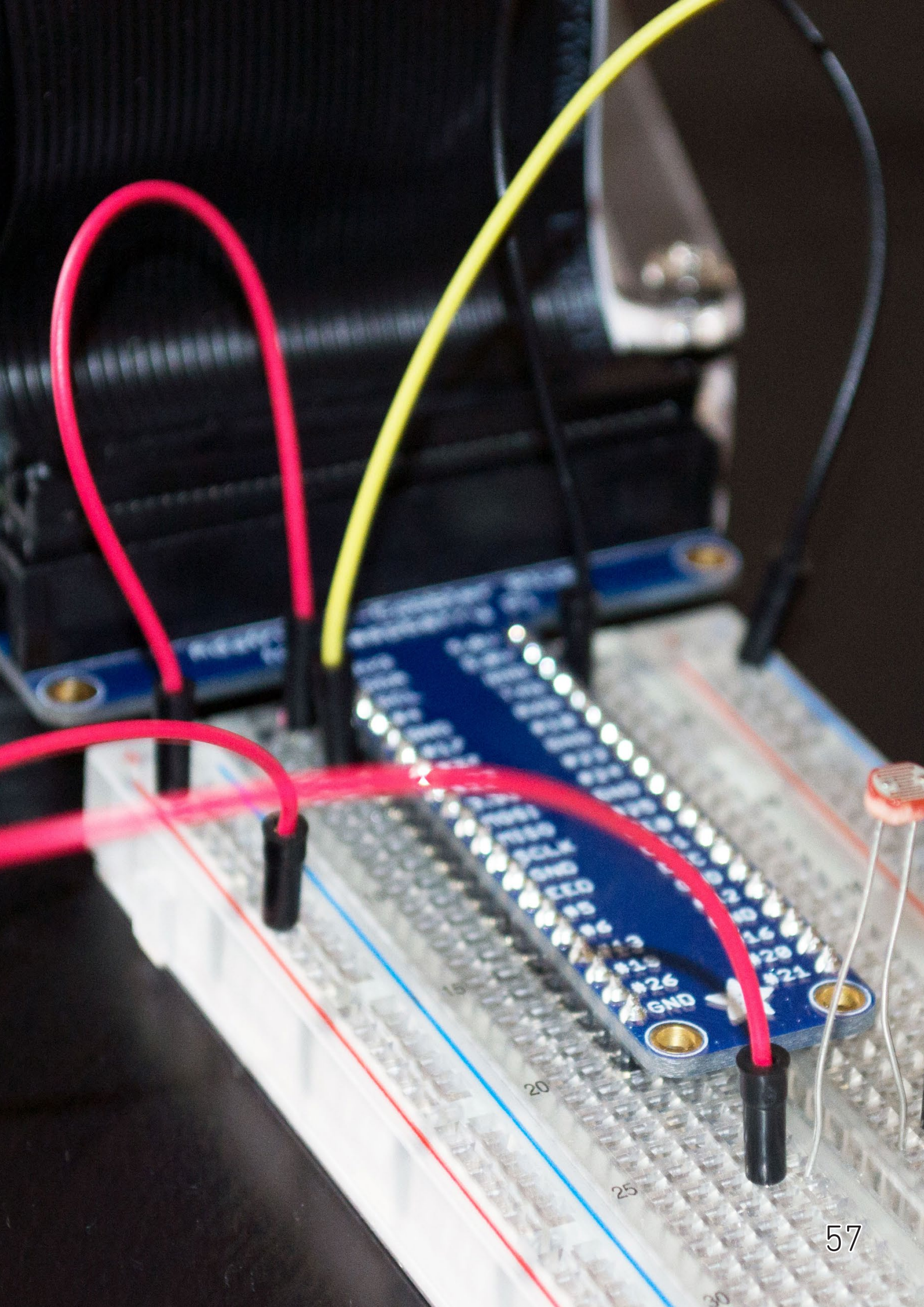
```
sudo nano light_sensor.py
```

Once you are sure you have written the code in its entirety into the folder, don't forget to press **Ctrl + X** then **Y** and press **Enter**.

Finally, we can run the code you either cloned or wrote out yourself by running the following command, this tells the python application to interpret our file:

```
sudo python light_sensor.py
```

I hope you now have the script working and you're receiving data that correctly reflects the changes in light on the sensor.


GND
20
25

Setting up a Temperature Sensor

Project Description

In this tutorial, I will be going through the steps on how to set up your very own Raspberry Pi tempera-ture sen-sor. Like most of the sensor tutorials they are straightforward consisting of a basic circuit and some code.

I will be making use of a DS18B20 waterproof sensor, this probe can provide temperatures over a one wire in-terface. Even better this is waterproof making it perfect if you need to use the sensor in a wet environment.

The sensor that I am using in this tutorial is a waterproof version of the DS18B20 sensor. It simply looks like a very long cord with a thick part on the end.

If you have just a plain version of the sensor without the wiring and waterproofing, then it should look a lot like a transistor.

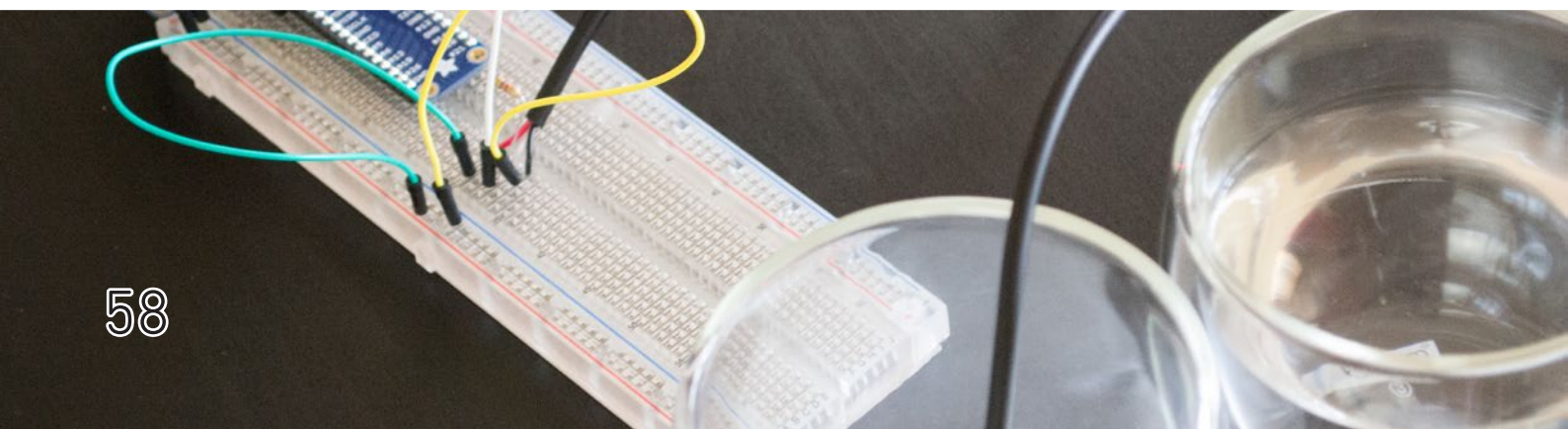
Equipment

Required

- Raspberry Pi
- SD Card (8 GB+ Recommended)
- DS18B20 Temperature Sensor
- 4.7k Resistor
- Breadboard
- Breadboard Wire

Optional

- Raspberry Pi Case
- USB Keyboard
- USB Mouse
- Network Connection
- GPIO Breakout Kit



Building the Raspberry Pi DS18B20 Circuit

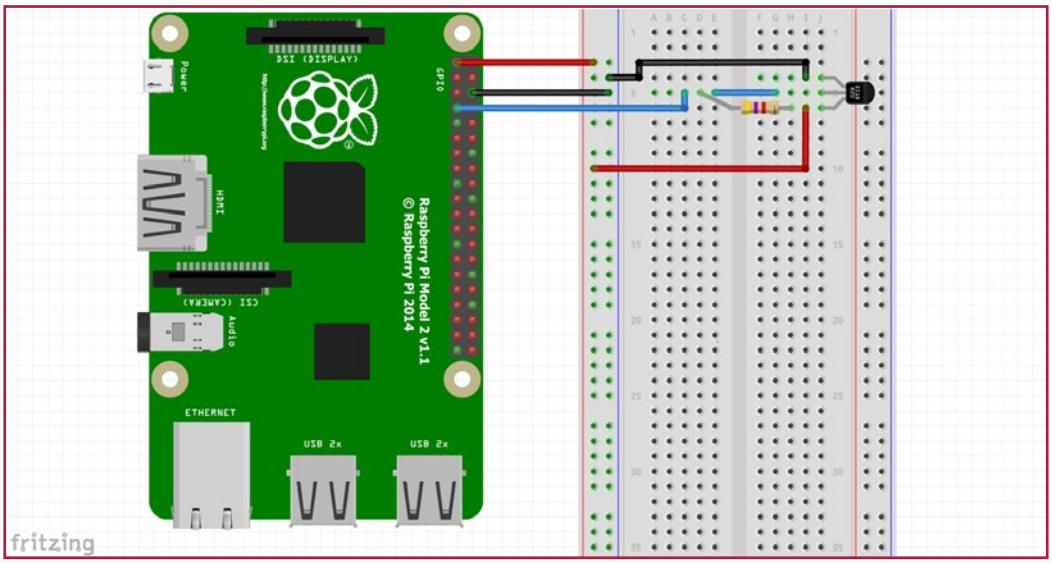
This sensor is pretty accurate with results from it being within **0.05°C** of the actual temperature. It can handle temperatures of **up to 125°C** (**260°F**), but it's recommended you **keep it below 100°C** (**210°F**).

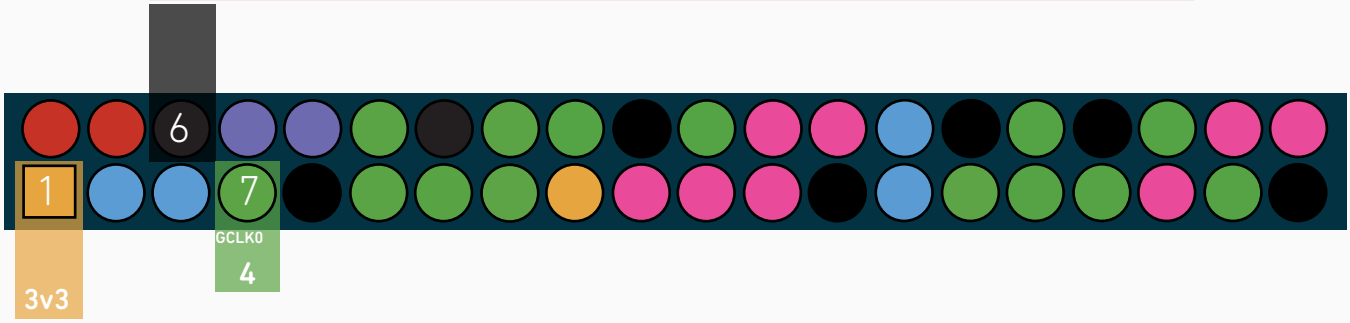
Keep in mind not all temperature sensors are the same something like the **TMP36** will not replace the **DS18B20** in this tutorial. The **TMP36 is an analog device** making it slightly harder to integrate with the Raspberry Pi.

Putting this circuit together is super simple, I will quickly go through some basic instructions below. If you're having trouble following them, then please refer to the diagram below.

1. First, connect the **3v3 pin** (**Pin 1**) from the Raspberry Pi up to the **positive rail** and a **ground pin** (**Pin 6**) to the **ground rail** on the **breadboard**.
2. Now place the **DS18B20 sensor** onto the **breadboard**.
3. Place a **4.7k resistor** between the **positive lead** (**Pin 3 on DS18B20**) and the **output lead** (**Pin 2 on DS18B20**) of the sensor.
4. Place a wire from the **positive lead** (**Pin 3 on DS18B20**) to the **positive rail** (**Pin 1 on GPIO**).
5. Place a wire from the **output lead** (**Pin 2 on DS18B20**) back to **pin 7 on the GPIO**.
6. Place a wire from the **ground lead** (**Pin 1 on DS18B20**) to the **ground rail** (**Pin 6 on GPIO**).

Once done your circuit should look something like the diagram we have shown below. Keep in mind some versions of the temperature sensor may have more wires than just 3.





The Temperature Sensor Setup

The code for setting up the temperature sensor is a little more complicated than the circuit itself. This added complexity is just because of the way we need to handle the data that comes from the sensor.

Before we make the Python script we first need to set up the Raspberry Pi so it can read data from the sensor, to do this we need to enable OneWire support, luckily, we can quickly do this through the **raspi-config** tool.

1. To enable support for the 1-Wire interface, we need to first start up the **raspi-config** tool. We do this by running the following command:

```
sudo raspi-config
```

2. Once open, go to **5. Interfacing Options** then select **P7. 1-Wire** then select **Yes** to enabling it. Once you have done this, you will want to reboot the Raspberry Pi.

```
sudo reboot
```

3. Once the Raspberry Pi has booted back up, we need to run modprobe so we can load the correct modules.

```
sudo modprobe w1-gpio
sudo modprobe w1-therm
```

4. Now change into the devices directory and use **ls** to see the folder/files in this directory.

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

5. Now run the following command, change the numbering after **cd** to what has appeared in your directory by using the **ls** command before. (If you have multiple sensors there will be more than one directory)

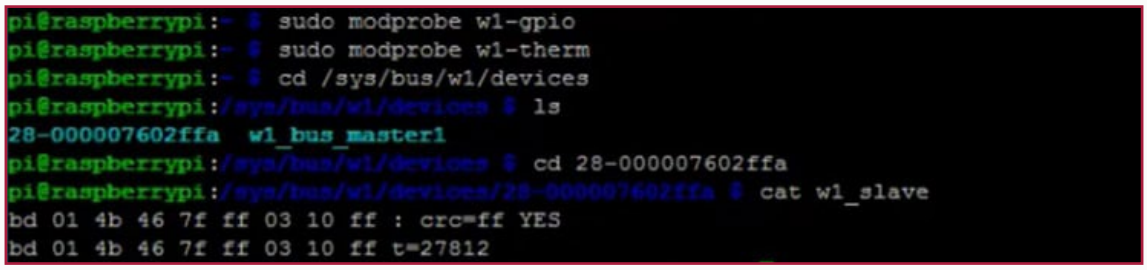
```
cd 28-000007602ffa
```

6. Now run the following command.

```
cat w1_slave
```

7. This command should output data but as you will notice it is not very user-friendly.
The **first line** should have a **YES** or **NO** at the **end of it**. **If it is YES**, then a **second line** with the **temperature** should appear.

This would look similar to something like **t=12323**. You will need to do a bit of math to make this a useable temperature that we can understand easily. For example, **Celsius** you just divide by **1000**.



Coding for the Temperature Sensor

Now it's time to move onto the python script.
I'll briefly explain the code below if you want to learn more about it. If you want to download it, you can just download it to your Raspberry Pi with the following command.

```
git clone https://github.com/pimylifeup/temperature_sensor.git
```

To begin the python script, we **import 3 packages**, **OS**, **Glob** and **time**.

Next, we run the **modprobe** commands these are the same as what we used before.

We then declare **3 different variables** that will point to the **location of our sensor data**. You shouldn't have to change any of these.

```
import os
import glob
import time

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

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

In the **read_temp_raw** function we open up the file that contains our **temperature output**, we determine these folders by making use of the **glob.glob** function.

We read all the lines from this file and then return it so the code that has called this function can use it, this file contains all the raw temperature information.

In this case, the **read_temp** function calls this function and interprets the data.

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

Coding for the Temperature Sensor - Continued

```
def read_temp():
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0
        temp_f = temp_c * 9.0 / 5.0 + 32.0
        return temp_c, temp_f
```

In the **read_temp** function, we process the data from the **read_temp_raw** function. We first make sure that the **first line contains YES**. By checking for '**YES**', we know there will be a line with a temperature in it.

If there is a temperature we then find the line with **t= by using lines[1].find('t=')**. **Lines[1]** means we're looking at the **2nd element in the array**, in this case, the **2nd line**.

Once we have the line we get all the numbers that are after the **t=**, this is done with this part of the code **lines[1][equals_pos+2:]**. **Equals_pos** is the start position of the **temperature (t)**, and we **add 2** to the **position**, so we only get the **actual temperature numbers**.

We then convert the number into both a **Celsius** and **Fahrenheit temperature**. We return both to the code that called this function, that being the **print function** located in the **while loop**.

```
while True:
        print(read_temp())
        time.sleep(1)
```

The **while loop** is always **true** so it will **run forever** until the program is interrupted by an error or the user canceling the script.

It just calls the **read_temp** within the **print function**. The **print function** allows us to see the **output on our screen**.

The script is then put to **sleep** for **1 second** every time it has read the sensor.

Once you have either downloaded or finished writing up the code and you also have set up the circuit correctly we can now call the Python script. To call the python script simply run the following command.

```
sudo python thermometer_sensor.py
```

You should now have an output of temperatures in both Fahrenheit and Celsius. You can alter this just to display your preferred temperature scale.

That's everything you need to know for getting your Raspberry Pi **DS18B20 temperature sensor** up and running.

If you're looking to extend this one step further, then be sure to check out my tips on the following page.

Possible Further Work

There are quite a few more things you're able to do with this Raspberry Pi temperature sensor. I will quickly mention a couple of ideas some that I will probably be doing sometime in the future.

You can have the Python script connect to a database such as MYSQL and store the data that way. If you add a timestamp to the data, you will be able to look back on data in the future and compare any changes.

This next tip would work great with the one above. You can use the data stored in the MYSQL database to make nice pretty graphs to show to temperature over the course of a day, month or even year. You could make it plot a graph in real time too.

Motion Sensor Circuit

Project Description

In this tutorial, we're building a Raspberry Pi motion sensor that makes use of a PIR sensor (Passive Infrared Sensor). A PIR sensor is most commonly seen in security systems to detect movement before sending the alarm off. They detect motion whenever there is a change of infrared temperature in their field of view.

Also, to make this project a little more interesting, we will also be using a piezo speaker whenever motion is detected. The piezo buzzer is a simple speaker that outputs a sound whenever a current is run through it. In this circuit, the buzzer will give a loud beep whenever the motion detector circuit is triggered.

Both these devices will need to be hooked up to the GPIO pins for the Raspberry Pi to control them so we will go into how to wire them up to the Raspberry Pi, and what code to write to make use of them.

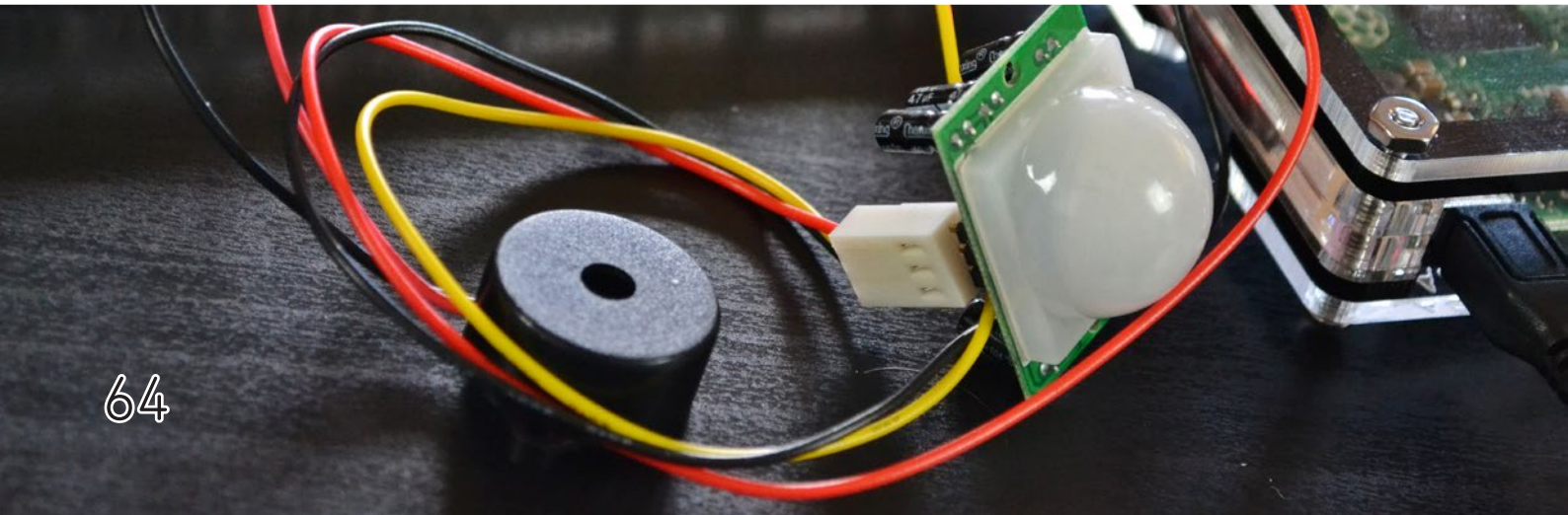
Equipment

Required

- Raspberry Pi
- SD Card (8 GB+ Recommended)
- PIR Sensor
- Piezo Speaker
- 1x 100 OHM speaker
- Breadboard
- Breadboard Wire / Jumper Cables

Optional

- Raspberry Pi Case
- External Hard drive or USB Drive
- Network Connection
- GPIO Breakout Kit



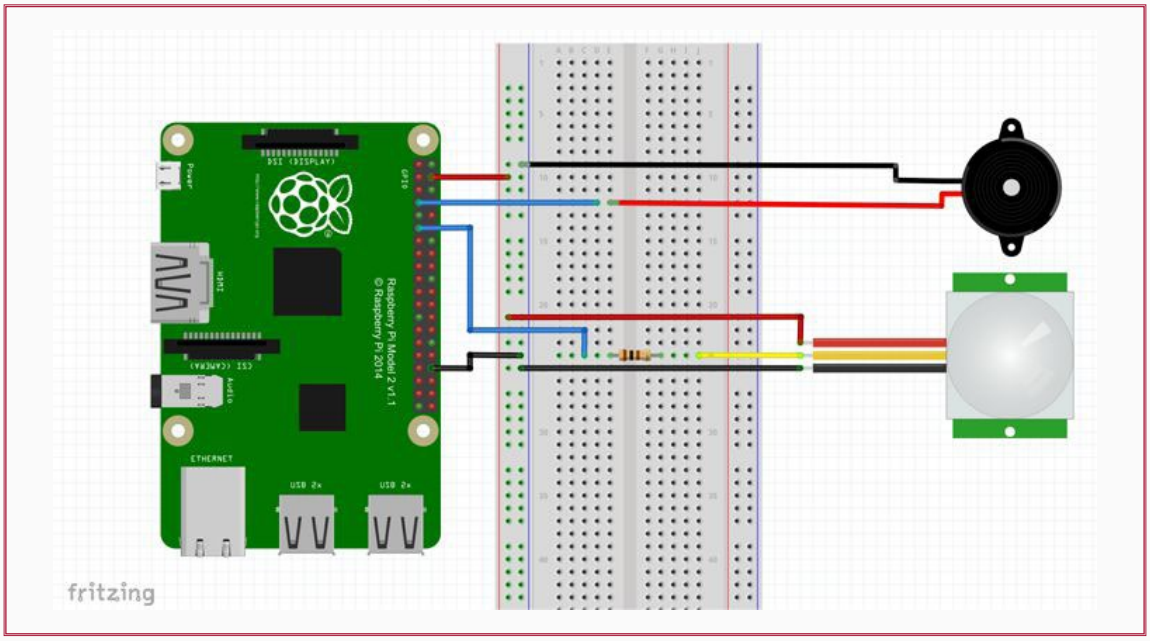
Raspberry Pi PIR Sensor Hardware Setup

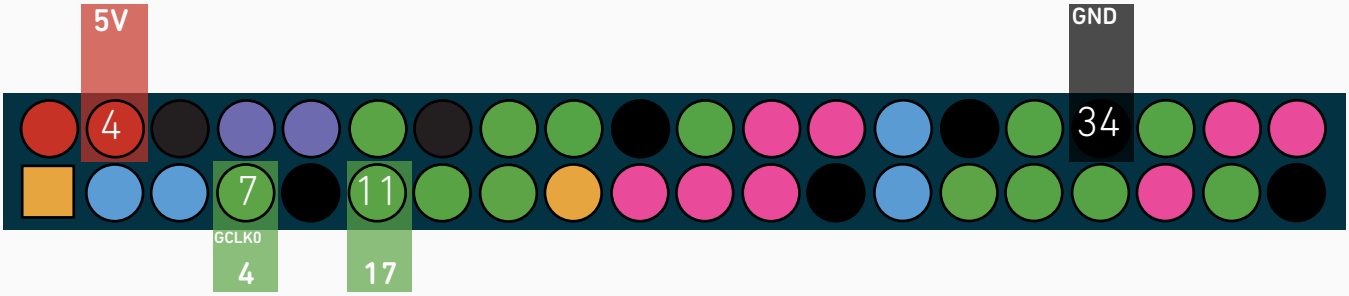
We will start off by putting together a simple circuit that makes use of a PIR sensor and a piezo speaker. This circuit will be designed to hook into the Raspberry Pi's GPIO pins so it will have complete control over the circuit.

One important thing to note is that most PIR sensors have some adjustable screws on them that will allow you to adjust both the time and sensitivity of them. The time will allow you to set a delay before it goes off (Send a high signal). (About 2-4 seconds). The sensitivity is how much movement needs to occur before it goes off.

A breadboard isn't necessarily required for this project but, I would highly recommend using one. They make prototyping and building circuitry a lot easier.

Instead of using a breadboard you can instead directly wire these up to the Raspberry Pi, but it does add a bit of extra complexity to the wiring.





To construct the circuit just do the following.
Don't forget you can use the diagram above to give yourself an idea of what you need to do.

1. Run a **ground pin (Pin 6)** to the **ground/negative rail** on the **breadboard**.
2. Run a **5v pin (Pin 4)** to the **positive rail** on the **breadboard**.
3. Connect the **piezo buzzer** to **pin 7** (**Red wire**) and the **negative rail** (**Black wire**).
4. Run a wire from **pin 11** to the **breadboard**. Place a **100-ohm resistor** at the end of the wire. Then **connect this** to the **yellow wire** of the **PIR sensor**.
5. Now for the PIR sensor run the **red wire** to the **positive rail** and the **black wire** to the **ground rail** on the **breadboard**.

Raspberry Pi Motion Sensor Software Setup

To bring our Raspberry Pi Motion sensor circuit to life, we will need to do a little bit of programming.

Firstly, on your Pi enter the following command to create a python script and load it in the nano text editor.

```
sudo nano motion_sensor.py
```

I will briefly explain what each part of the code does.

Firstly, we import the GPIO & time Python packages as we will need these to be able to interact with the GPIO pins and pause our script.

We set 3 variables, the first 2 are references to our pins, and thus I have named appropriately. The current state variable is where we will store our sensor state. If this is 0, then it is off or, 1 means it has been activated.

We also set our GPIO mode to reference the physical numbering of the pins rather than the actual numbering. We do this as GPIO mode is easier to understand/remember as all the pins are numbered in order.

We also set up our GPIO pins to be either outputs or inputs. For example, we want to detect motion so our PIR sensor will be input. On the other hand, our piezo buzzer is going to need to act as an output.

```
import RPi.GPIO as GPIO
import time

pir_sensor = 11
piezo = 7

GPIO.setmode(GPIO.BOARD)
GPIO.setup(piezo,GPIO.OUT);
GPIO.setup(pir_sensor, GPIO.IN)

current_state = 0
```

Motion Sensor Software Setup

In this next part, we have an infinite while loop. This means it will never exit because it is always true. (You can still cancel the script by press **Ctrl+C** in the terminal).

We begin by putting the script to sleep for 0.1 seconds. After this we get current state of the sensor and if it is 1 (e.g. detected motion) then we run the code inside the if statement. If it isn't 1, then we continue to loop constantly checking the sensor.

The code in the if statement sends the piezo buzzer to high that should emit a noise. The script will do this for a second then turn the buzzer off. After this it will wait for another 5 seconds before exiting the if statement and then checking the Raspberry Pi PIR sensor again.

We have also nested out code within a try, except, finally block. We have added this because we will need to use the keyboard to stop the script. It also very important that we run GPIO.cleanup() to ensure our script cleans up nicely. The try, except, finally code allows us to do this.

```
try:
        while True:
                time.sleep(0.1)
                current_state = GPIO.input(pir_sensor)
                if current_state == 1:
                        print("GPIO pin %s is %s" % (pir_sensor, current_state))
                        GPIO.output(piezo,True)
                        time.sleep(1)
                        GPIO.output(piezo,False)
                        time.sleep(5)
except KeyboardInterrupt:
        pass
finally
        GPIO.cleanup()
```

Once you have finished working in the script it's now time to turn it on and test it out. To do this enter the following command:

```
sudo python motion_sensor.py
```

If you move in front of the Raspberry PI PIR sensor, then it should turn the piezo buzzer on and emit a noise.

If it doesn't then is likely you have hooked up wires to the wrong pins or there is an error in the code. If it is a code error you will most likely see an error in the terminal of the Raspberry Pi.

This tutorial is pretty basic and only the beginning of the many applications you can use the PIR sensor. You can get it trigger all sorts of things from something as simple as a counter (counts as people/cars/things go past it), set off a Raspberry Pi camera, activate a different script and lots more.

Distance Sensor Circuit

Project Description

In this tutorial, we will be utilizing the HC-SR04 Ultrasonic with our Raspberry Pi. This guide will go through showing you how to wire up the sensor with the Raspberry Pi as well as exploring how we can utilize the sensor also to read distance.

We will be showing you how to wire the HC-SR04 sensor up to the Raspberry Pi, including how to wire a voltage divider as the circuit requires one to drop the 5v output from the sensor to 3.3v for the Raspberry Pi.

Including showing how to wire up the sensor to the Raspberry Pi, we also explore writing a Python script that will utilize the HC-SR04 Ultrasonic sensor to calculate distance. We achieve this by measuring the time it takes the ultrasonic pulse being sent out to it being received back by the sensor.

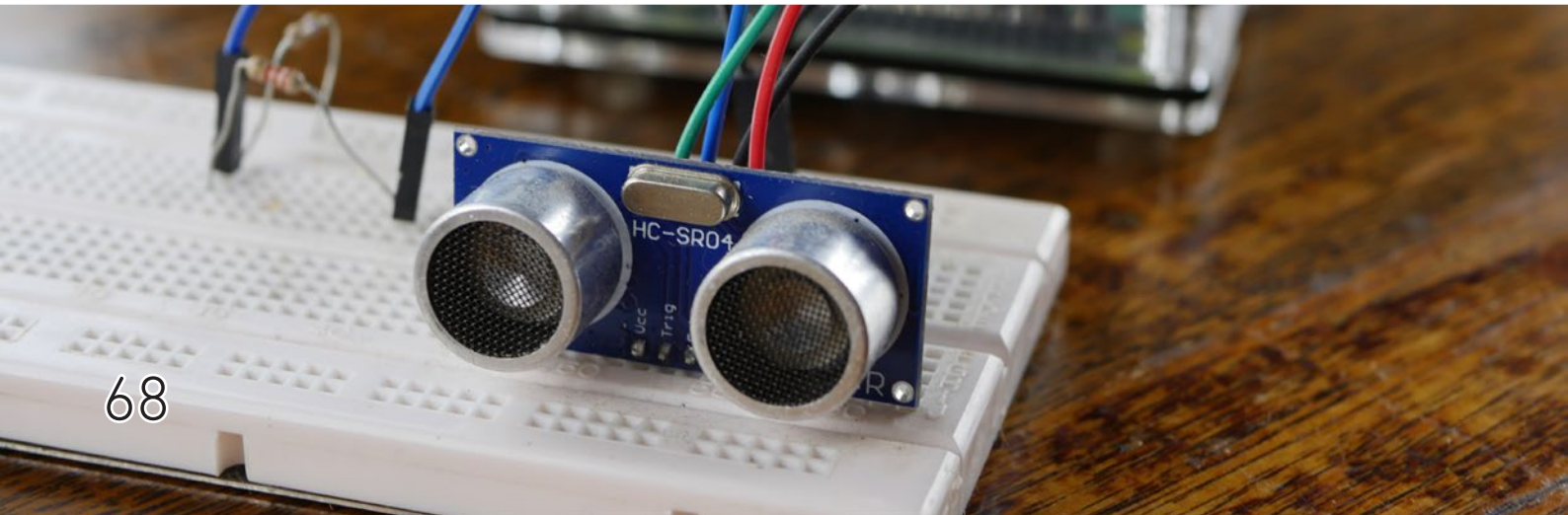
Equipment

Required

- Raspberry Pi
- SD Card (8 GB+ Recommended)
- HC-SR04 Ultrasonic Sensor
- 1x 1k OHM Resistor
- 1x 2k OHM Resistor
- Breadboard
- Breadboard Wire / Jumper Cables

Optional

- Raspberry Pi Case
- External Hard drive or USB Drive
- Network Connection
- GPIO Breakout Kit



Raspberry Pi Distance Sensor Hardware Setup

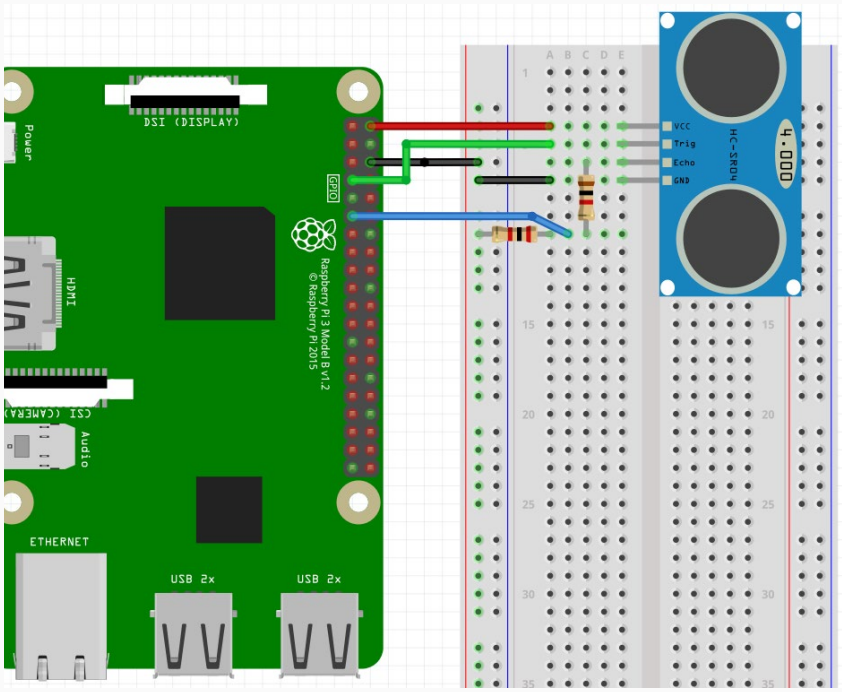
In this section, we will be showing you how to wire your HC-SR04 Distance Sensor to your Raspberry Pi. Wiring your sensor is a relatively simple process as most pins of the distance sensor map directly to a pin on the Raspberry Pi.

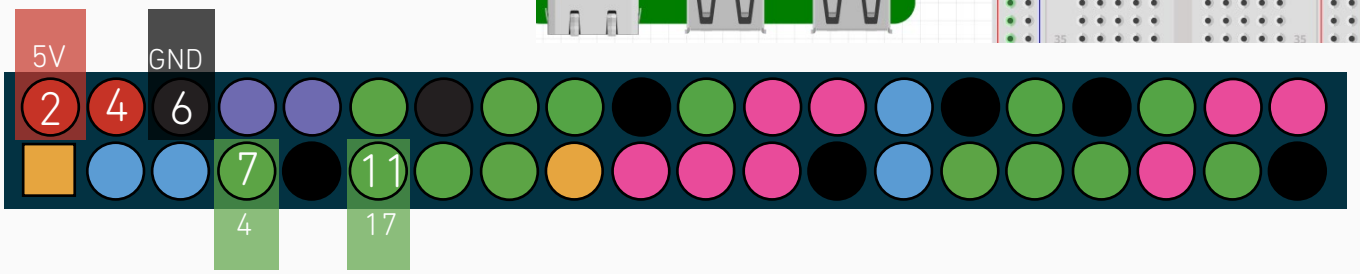
The only slightly complicated part of wiring the device to the Raspberry Pi is the voltage divider that we set up for the echo pin.

The reason for adding a voltage divider is to drop the voltage going to the GPIO pins down to **3.3v** from **5v**. In this guide, we will be utilizing a **1k Ω** resistor and a **2k Ω** resistor to achieve this. If you want to understand how we calculate the values of a voltage divider, you can check out the guide featured at the end of this book.

Follow the guides below to see how to wire your HC-SR04 distance sensor to your Raspberry Pi.

- **VCC** Connects to **Pin 2** (**5v**)
- **Trig** Connects to **Pin 7** (**GPIO 4**)
- **Echo** Connects to **R1** (**1k Ω**)
- **R2** (**2k Ω**) Connects from **R1** to **Ground**
- Wire from **R1** and **R2** connects to **Pin 11**
- **GND** connects to **Pin 6** (**Ground**)





To construct the circuit just do the following.
Don't forget you can use the diagram above to give yourself an idea of what you need to do.

1. Run a wire from the **ground pin (Pin 6)** to the **ground/negative rail** on the **breadboard**.

2. Run a wire from **5v pin (Pin 2)** to the **VCC** pin on the HC-SR04 distance sensor.
3. Run a wire from **pin 7** to the **TRIG** pin on the distance sensor.
4. Run a **1k Ω** resistor from the **ECHO** pin on the distance sensor to a spot on the breadboard.
5. Run a **2k Ω** resistor from the **1k Ω** resistor to the **ground/negative rail** on the Raspberry Pi.

6. Run a wire from between the **1k Ω** resistor and the **2k Ω** resistor to **pin 11** on the Raspberry Pi.

Utilizing your Raspberry Pi Distance Sensor

1. To utilize our Raspberry Pi Distance Sensor we luckily only have to program up a python script. Since we only use GPIO pins to interact with the distance sensor, there is no need to mess around with the raspi-config tool.

Before we begin writing our script, lets first make a folder to keep it. Run the following two commands on your Raspberry Pi to create the folder and change directory to it.

```
mkdir ~/distance_sensor/
cd ~/distance_sensor
```

2. Now that we are in our new directory, let's begin writing our distance sensor python script by running the following command on the Raspberry Pi.

```
nano distance_sensor.py
```

3. Within this file, write the following lines of code. We will explain the important parts of each block of code as we write it. Remember when writing in python code that it is whitespace sensitive, so make sure that you retain the tabs that we have put in.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

try:
```

The first line tells the shell what we should be utilizing to run our python file.

We then proceed to import the two libraries that we need to write our script and interact with the distance sensor from our Raspberry Pi. The two libraries that we import are the following:

Rpi.GPIO - This is the package that allows us to control and interact with the GPIO pins, without this package we wouldn't be able to talk with our distance sensor.

time - This package allows us to control time-related functions with the script. We mainly just use this to put the script to sleep and also time how long it takes to receive data back from the sensor.

Finally, we begin our **try:** statement, we cover the vast majority of the code within this to ensure that we clear the GPIO pins on exit.

```
    GPIO.setmode(GPIO.BOARD)

    PIN_TRIGGER = 7
    PIN_ECHO = 11

    GPIO.setup(PIN_TRIGGER, GPIO.OUT)
    GPIO.setup(PIN_ECHO, GPIO.IN)
```

On the first line here we set our GPIO mode to GPIO.BOARD, this means we are using the physical pin numbers and not the BCM numbers. We use this to ensure compatibility with our script as the BCM Numbers are not guaranteed to stay the same.

Next two lines we create two variables to store the physical pins that we have the trigger and echo wired to. We define them here to make it easy to change them without having to sift through the code continually.

Next we setup both our pins to either expect an output or an input. Our trigger pin is obviously an output pin as we need to ping this pin to start the sensor. The Echo pin is where we expect our data from, so we set this to receive input.

```
        GPIO.output(PIN_TRIGGER, GPIO.LOW)

        print "Waiting for sensor to settle"

        time.sleep(2)
```

The first line we set our **PIN_TRIGGER** GPIO Pin, so it doesn't send out anything by setting it to "Low". We do this to let our **HC-SR04** sensor settle.

We then proceed to sleep the script for 2 seconds to ensure we give the distance sensor enough time to settle and don't immediately start triggering it.

```
        print "Calculating distance"

        GPIO.output(PIN_TRIGGER, GPIO.HIGH)

        time.sleep(0.00001)

        GPIO.output(PIN_TRIGGER, GPIO.LOW)
```

Finally, we get to the piece of code which gets the **HC-SR04 distance sensor** to trigger. To do this, we need to first set our **PIN_TRIGGER** to high.

We then need to sleep the script for 1 nanosecond, the reason for this is that the HC-SR04 distance sensor requires a pulse of 1 nanosecond to trigger it.

Immediately after the sleep has completed, we set **PIN_TRIGGER** low again.

```
        while GPIO.input(PIN_ECHO)==0:
            pulse_start_time = time.time()
        while GPIO.input(PIN_ECHO)==1:
            pulse_end_time = time.time()

        pulse_duration = pulse_end_time - pulse_start_time

        distance = round(pulse_duration * 17150, 2)

        print "Distance:",distance,"cm"
```

First, we run a while loop to continually check if **PIN_ECHO** is **low** (**0**) if it is we continually set the **pulse_start_time** to the current time until it becomes **high** (**1**)

Once **PIN_ECHO** reads **high**, we set **pulse_end_time** to the current time. We do this until **PIN_ECHO** is set to **low** again.

These two loops allow us to calculate the time that it took for the ultrasonic pulse to bee bounced back. Once we have both times, we just minus the two saved times to work out the duration.

With the pulse duration calculated we can work out the distance it traveled since we know the rough speed of ultrasonic sound is **34300 cm/s**.

Since the duration of the pulse is the time it took for the ultrasonic sound to hit an object and bounce back, we will just use half the speed to calculate the distance it traveled before returning.

So finally our distance is equal to the duration of the pulse, multiplied by **17150 cm/s**.

```
    finally:
        GPIO.cleanup()
```

These two final lines of code are quite crucial as it ensures an end to our try: statement and ensures that we run **GPIO.cleanup()** when the script is terminated in any way. Failing to do so will throw warnings when rerunning the script or any other script that makes use of the GPIO pins.

4. Once you have finished typing in all the code, you should end up with something that looks like what we have below.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

try:
    GPIO.setmode(GPIO.BOARD)

    PIN_TRIGGER = 7
    PIN_ECHO = 11

    GPIO.setup(PIN_TRIGGER, GPIO.OUT)
    GPIO.setup(PIN_ECHO, GPIO.IN)

    GPIO.output(PIN_TRIGGER, GPIO.LOW)

    print "Waiting for sensor to settle"

    time.sleep(2)

    print "Calculating distance"

    GPIO.output(PIN_TRIGGER, GPIO.HIGH)

    time.sleep(0.00001)

    GPIO.output(PIN_TRIGGER, GPIO.LOW)

    while GPIO.input(PIN_ECHO)==0:
        pulse_start_time = time.time()
    while GPIO.input(PIN_ECHO)==1:
        pulse_end_time = time.time()

    pulse_duration = pulse_end_time - pulse_start_time

    distance = round(pulse_duration * 17150, 2)

    print "Distance:",distance,"cm"

finally:
    GPIO.cleanup()
```

You can save the file by pressing **Ctrl + X** then **Y** and pressing **Enter**.

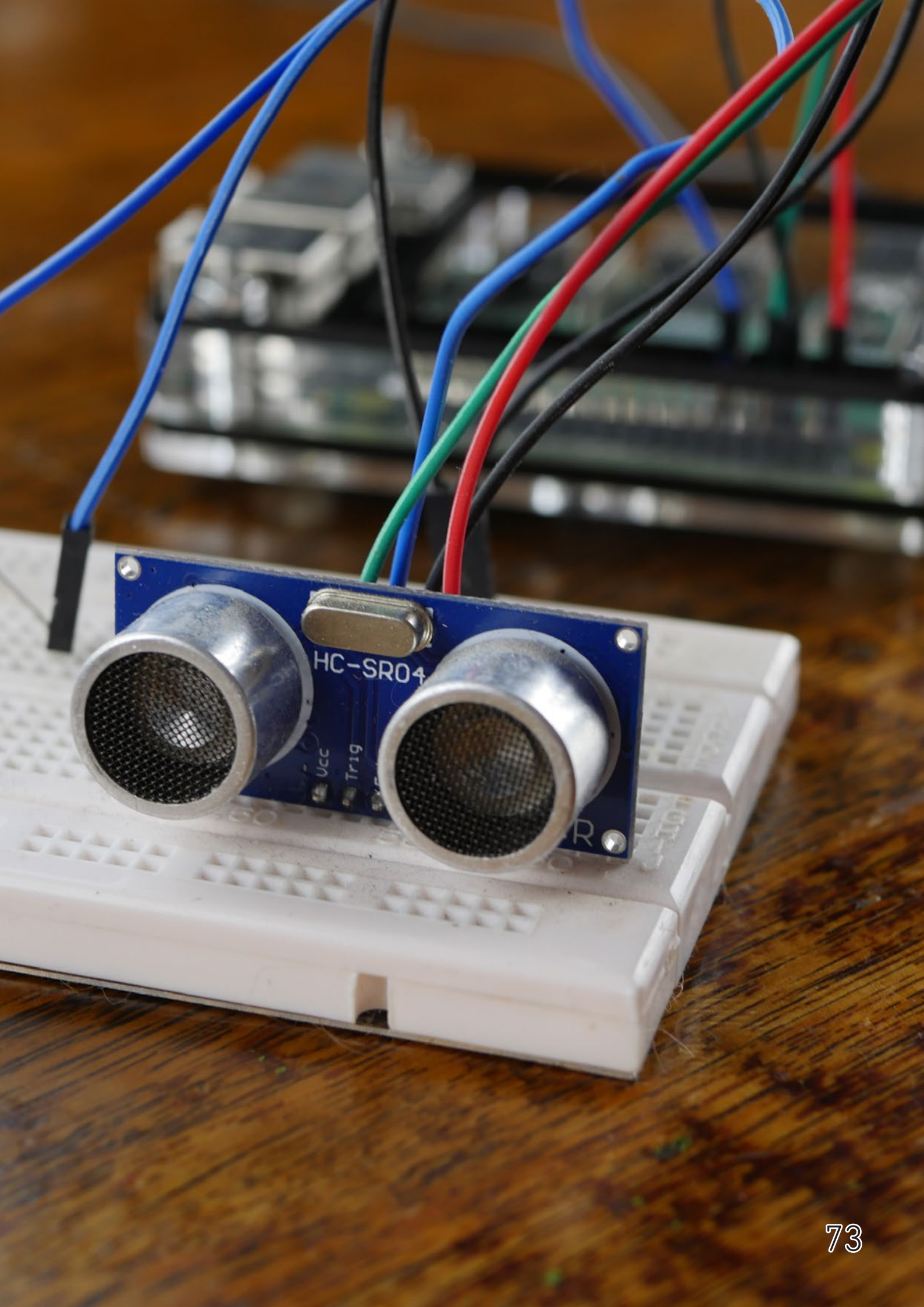
5. Now that you have successfully written the script we can run it by running the following simple command on your Raspberry Pi.

```
python ~/distance_sensor/distance_sensor.py
```

6. If everything is working correctly, you should get something like what is shown below in your terminal. Of course, the distance will be different to what we got.

```
Waiting for sensor to settle
Calculating distance
Distance: 29.69 cm
```

Hopefully, by now you will have successfully set up your Raspberry Pi **HC-SR04** distance sensor and also written a python script that interacts with the sensor and successfully calculates the distance.


HC-SR04
Vcc
Trig

Pressure Pad Sensor

Project Description

In this tutorial we will be walking you through the process of connecting a pressure pad to your Raspberry Pi and utilizing its values within Python.

There are many uses for a pressure pad such as detecting when someone sits down or detecting when pressure is placed on a specific part of an object. It is a pretty great sensor to make use of in IoT projects.

In this tutorial we only go through the steps of hooking the pad, so you can detect if enough pressure has been applied or not.

We will also go through methods to receive variable values by using a capacitor or an analog to digital converter (ADC).

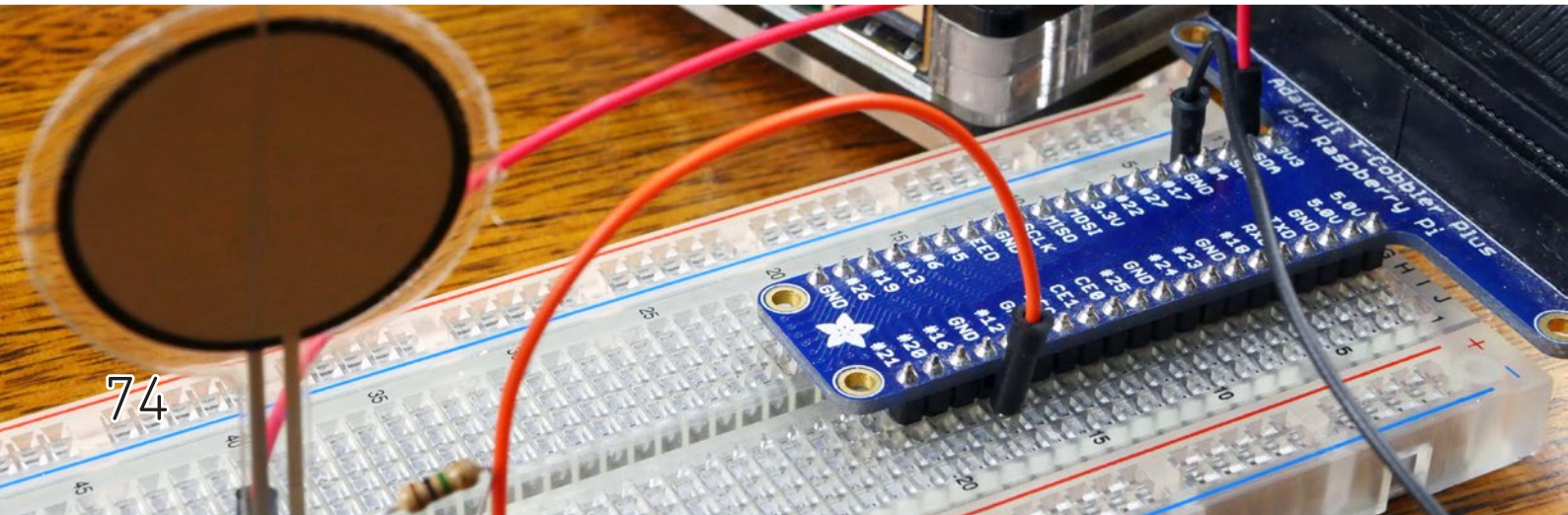
Equipment

Required

- Raspberry Pi
- SD Card (8 GB+ Recommended)
- Beadboard
- Breadboard wire
- Pressure Pad
- 1m Resistor

Optional

- MCP3008
- 1uf Capacitor
- GPIO Breakout Kit



Raspberry Pi Pressure Pad Hardware Explanation

The circuit for this pressure pad is surprisingly simple. First off I will touch on the parts that make up the circuits and why we're making use of them below.

Keep in mind that you won't need all the parts, it just depends on what you plan on doing.

Pressure Pad Sensor

The pressure pad is the most important part of our circuit as it won't work without it in use. I made use of a FlexiForce pressure pad sensor, but you're able to use cheaper alternatives.

The FlexiForce sensor resistance will vary between near infinite when you're applying little to no pressure, to under 25k ohms for when you're applying a lot of pressure to it.

It's very much like a light dependent sensor, but instead of light it uses direct pressure.

Resistor

The resistor is required in all our circuits except for the capacitor circuit. This resistor will act as a voltage divider and divides the 3v3 between the pressure pad and the resistor. When pressure is applied it will provide enough voltage to send our pin to high.

Below is the equation you can workout what the voltage out to our GPIO pin will be. For example, if our pressure pad is max resistance, it will be (1,000,000 / (1,000,000 + 10,000,000)) * 3.3 = 0.3 volts. This voltage isn't enough to send the pin to high.

On the flip side, if we're applying a high amount of pressure so the pressure pad resistance is only 50,000 then our equation will be (1,000,000 / (1,000,000 + 50,000)) * 3.3 = 3.14 volts. This voltage is enough to send the pin to high.

$$V_{out} = V_{in} \times \frac{R_2}{R_1 + R_2}$$

Since the Raspberry Pi doesn't have analog pins it will only ever be high or low. We can workaround this by either using a capacitor or analog to digital converter, see below.

Capacitor

A capacitor will allow us to get variable results without the use of an analog to digital converter. The results aren't as accurate but still incredibly handy as you can see varying amount of pressure being applied to the pressure pad.

The capacitor will act as a battery charging up while receiving power and discharging when it's full or when it stops receiving a charge. By using this in series with our pressure pad we can work out the resistance of the pressure pad. Higher the resistance the less pressure is being applied.

Analog to Digital Converter

An ADC allows us to connect analog sensors and devices to the Raspberry Pi. It involves quite a bit more wiring but will allow us to get a variable value rather than a simple on or off.

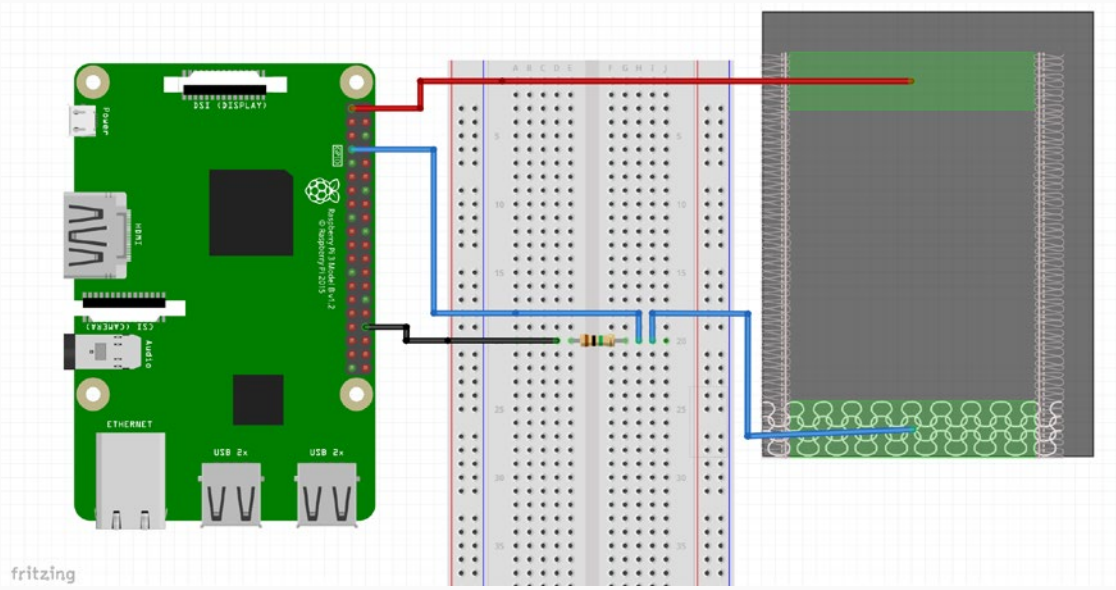
A variable value allows us to do more such as checking how much pressure is being applied and performing a different function depending on the result. Great if you need to keep an eye on pressure on specific areas.

Circuit Diagram using a Resistor

The steps to setting up this circuit are straightforward and shouldn't take you long at all. This circuit will give you a simple on or off result from the pressure pad.

- Connect one end of the pressure pad to a **3v3 pin** on your Raspberry Pi.
- Place a resistor onto the breadboard and have one end go to GND and the other to **pin 7**.
- Lastly, have the resistor end that goes to **pin 7** also go to the other end of the pressure pad.

The diagram below shows you how the circuit should be wired up.

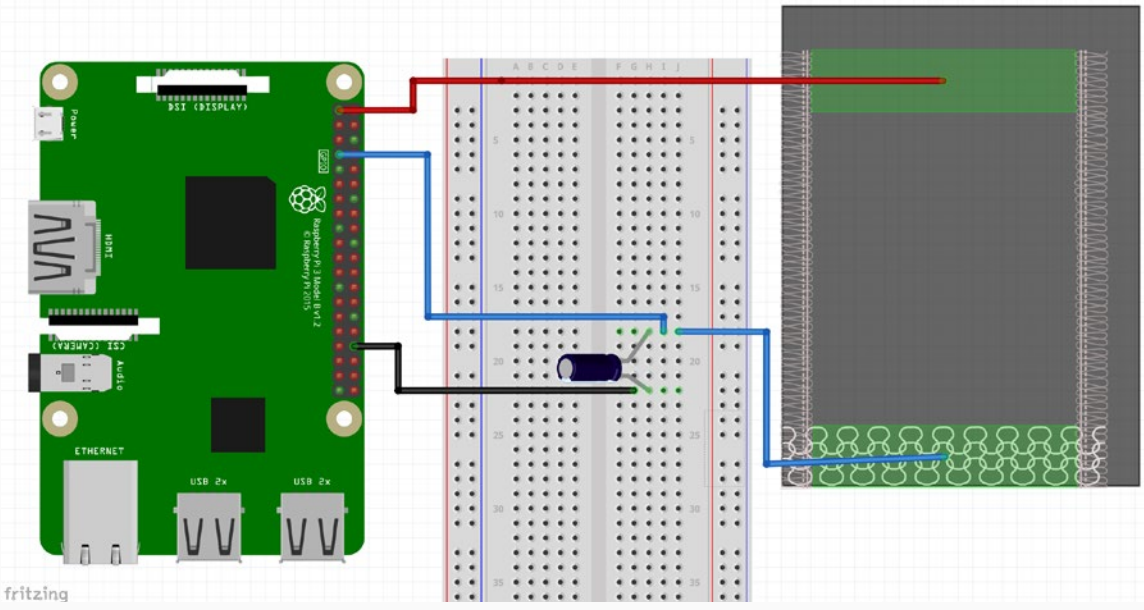


Circuit Diagram using a Capacitor

We can use a capacitor to measure the resistance of the pressure pad. This circuit is ideal if you need to know roughly how much pressure is being applied to the sensor.

- Connect one end of the pressure pad to a **3v3 pin** on your Raspberry Pi.
- Place a capacitor onto the breadboard and have the **negative end** go to **GND** and the other to **pin 7** and the pressure pad.

The diagram below shows you how the circuit should be wired up.



Circuit Diagram using a Analog to Digital Converter

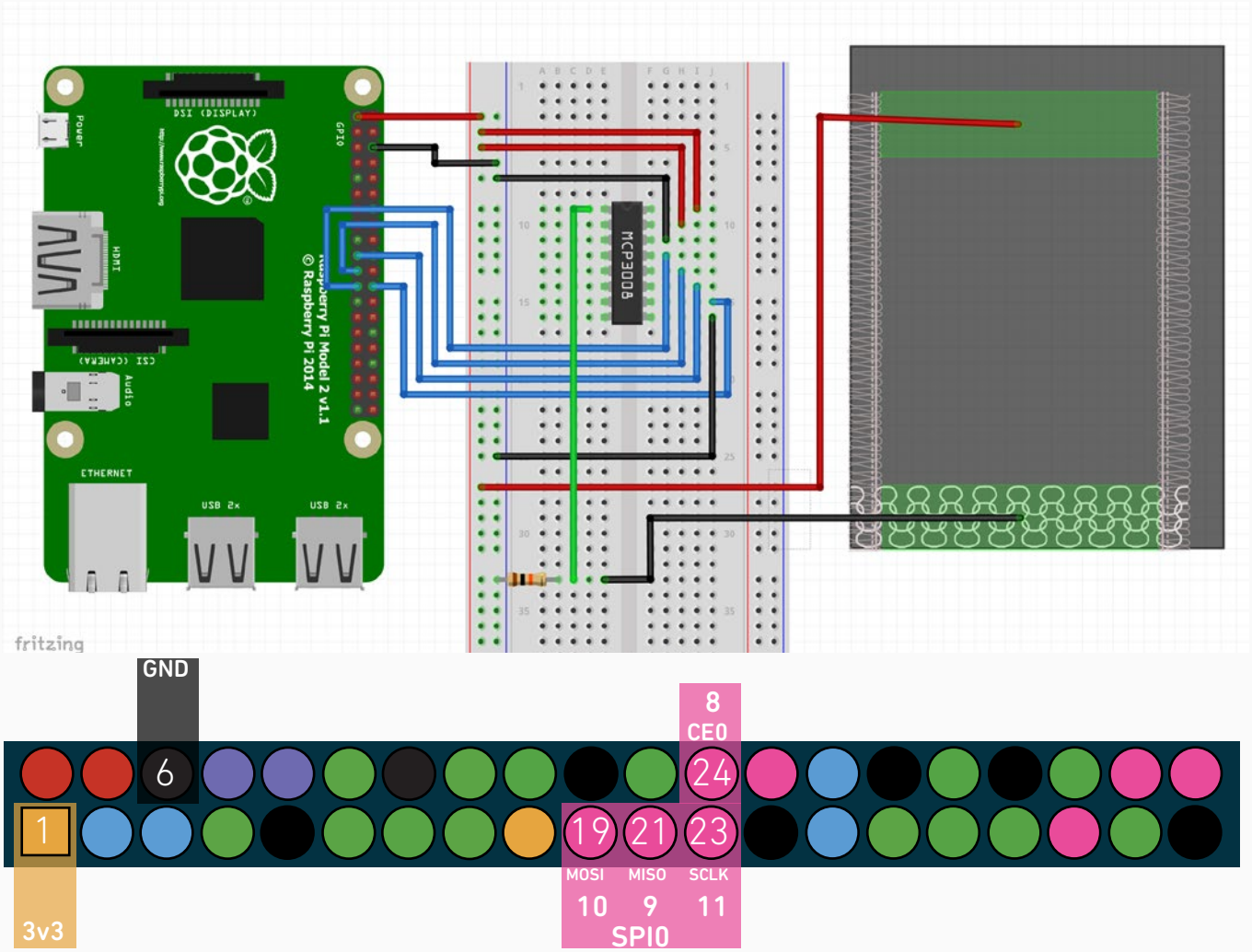
The last circuit we're looking at makes use of an analog to digital converter (ADC).

This ADC will allow you to get semi-accurate values from the pressure pad rather then just whether it is HIGH or LOW.

We also make use of the resistor and use it as a voltage divider to provide the analog to digital converter with our readings.

- VDD (**Pin 16**) wire this to **3.3V**
- VREF (**Pin 15**) wire this to **3.3V**
- AGND (**Pin 14**) wire this to **GROUND**
- CLK (**Pin 13**) wire this to **GPIO11** (**Pin 23**/SCLK)
- DOUT (**Pin 12**) wire this to **GPIO9** (**Pin 21**/MISO)
- DIN (**Pin 11**) wire this to **GPIO10** (**Pin 19**/MOSI)
- CS (**Pin 10**) wire this to **GPIO8** (**Pin 24**/CE0)
- DGND (**Pin 9**) wire this to **GROUND**
- Wire the pressure pad to **3.3V**
- Wire a resistor from **GROUND** to **CH0**
- Also wire **CH0** to the pressure pad

The diagram below shows you how the circuit should be wired up.



Coding the Pressure Pad

Each circuit setup requires its own unique code. The simplest is using the resistor while the most complicated is to use an analog to digital converter.

Code using a Resistor

Much like the circuit, the code for the resistor is also easy since we're only looking for when the pin goes to high.
In the code, we import our required packages, GPIO for interaction with the GPIO pins and time for pausing the script.

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

Now we first set up the **GPIO** so that it runs in **BCM mode** and setup **GPIO pin 4** to act as an input.

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(4,GPIO.IN)
```

Next, we enter an infinite loop and read the GPIO pin. If it is high and the previous reading was low, then we print a message. We then pause the script slightly before repeating.

```
while True:
   #take a reading
   input = GPIO.input(4)
   #if the last reading was low and this one high, alert us
   if ((not prev_input) and input):
      print("Under Pressure")
   #update previous input
   prev_input = input
   #slight pause
   time.sleep(0.10)
```

You can get the full code for retrieving a value from the resistor circuit below.

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(4,GPIO.IN)

prev_input = 0
try:
   while True:
      #take a reading
      input = GPIO.input(4)
      #if the last reading was low and this one high, alert us
      if ((not prev_input) and input):
         print("Under Pressure")
      #update previous input
      prev_input = input
      #slight pause
      time.sleep(0.10)
except KeyboardInterrupt:
   pass
finally:
   GPIO.cleanup()
```

Code using a Resistor

Much like resistor code this isn't overly complex and is based on our LDR code that we made use of in a previous tutorial.
Firstly, we must insert all the packages we require for this to work correctly, time and GPIO.

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

The rest of our code is pretty simple, we have a function called **rc_time**, and this takes one parameter. In this function, we wait until the **pin goes to high** and then return the count where "**count**" is the **time it took to go high**.

Once we have returned the count we have the pin to **act as an output** and **set it to low**, we then **wait 10ms** before switching it back to an **input**. It will then count until the pin goes high.

You can use the count value to determine how much pressure somebody is applying to the pressure pad. If the value is really high **such as 50k+**, then you're applying little to no pressure. **Lower than 10k** means there is quite a bit of pressure on the pad.

The full code can be found below.

```
#!/usr/local/bin/python

import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

#define the pin that goes to the circuit
pin_to_circuit = 4

def rc_time (pin_to_circuit):
    count = 0

    #Output on the pin for
    GPIO.setup(pin_to_circuit, GPIO.OUT)
    GPIO.output(pin_to_circuit, GPIO.LOW)
    time.sleep(0.1)

    #Change the pin back to input
    GPIO.setup(pin_to_circuit, GPIO.IN)

    #Count until the pin goes high
    while (GPIO.input(pin_to_circuit) == GPIO.LOW):
        count += 1

    return count

#Catch when script is interrupted, cleanup correctly
try:
    while True:
        print(rc_time(pin_to_circuit))
except KeyboardInterrupt:
    pass
finally:
    GPIO.cleanup()
```

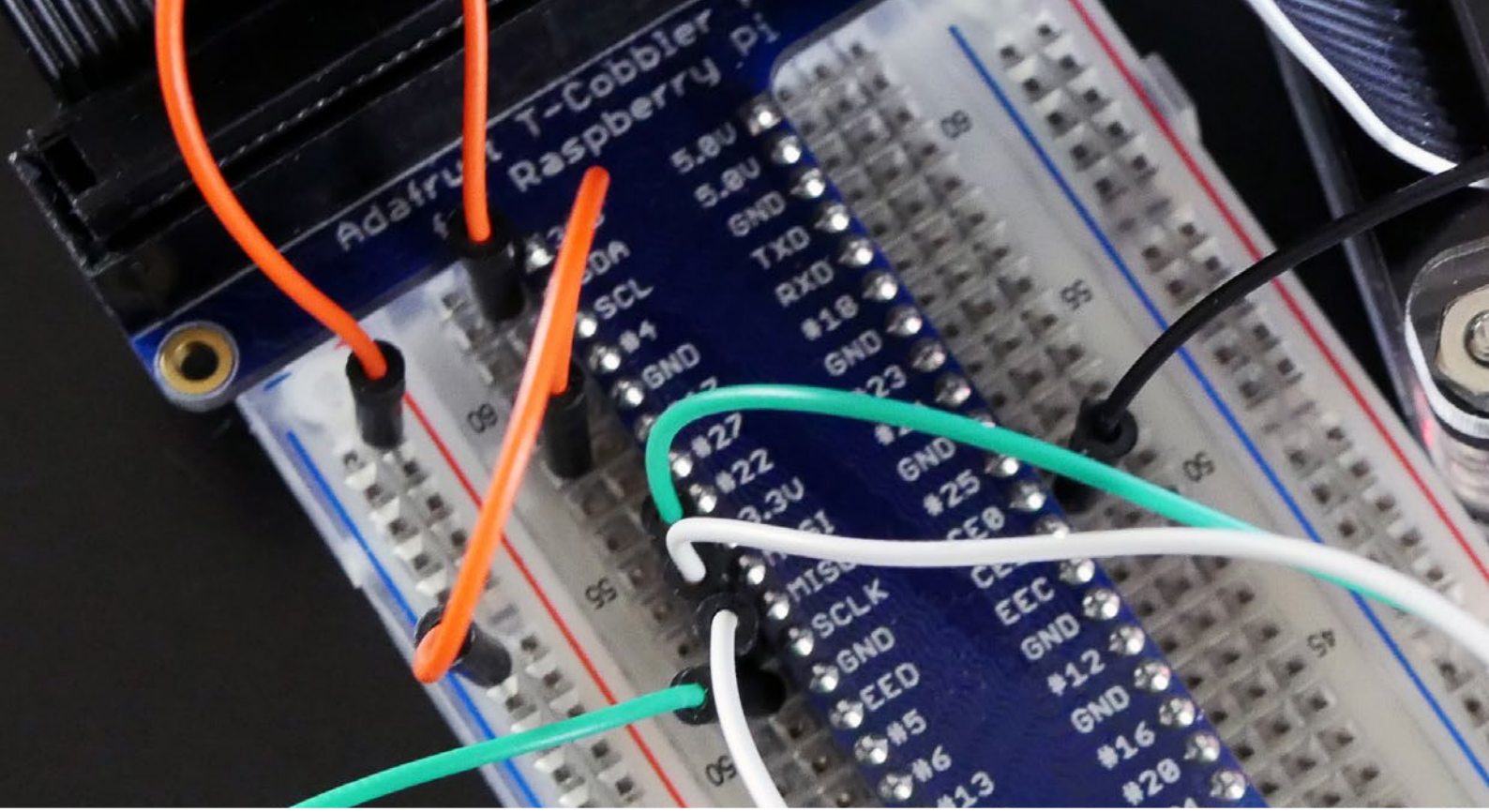
Code using a Analog to Digital Converter (ADC)

The analog to digital converter code is pretty straightforward, the tricky part is receiving the value from the MCP3008 chip.

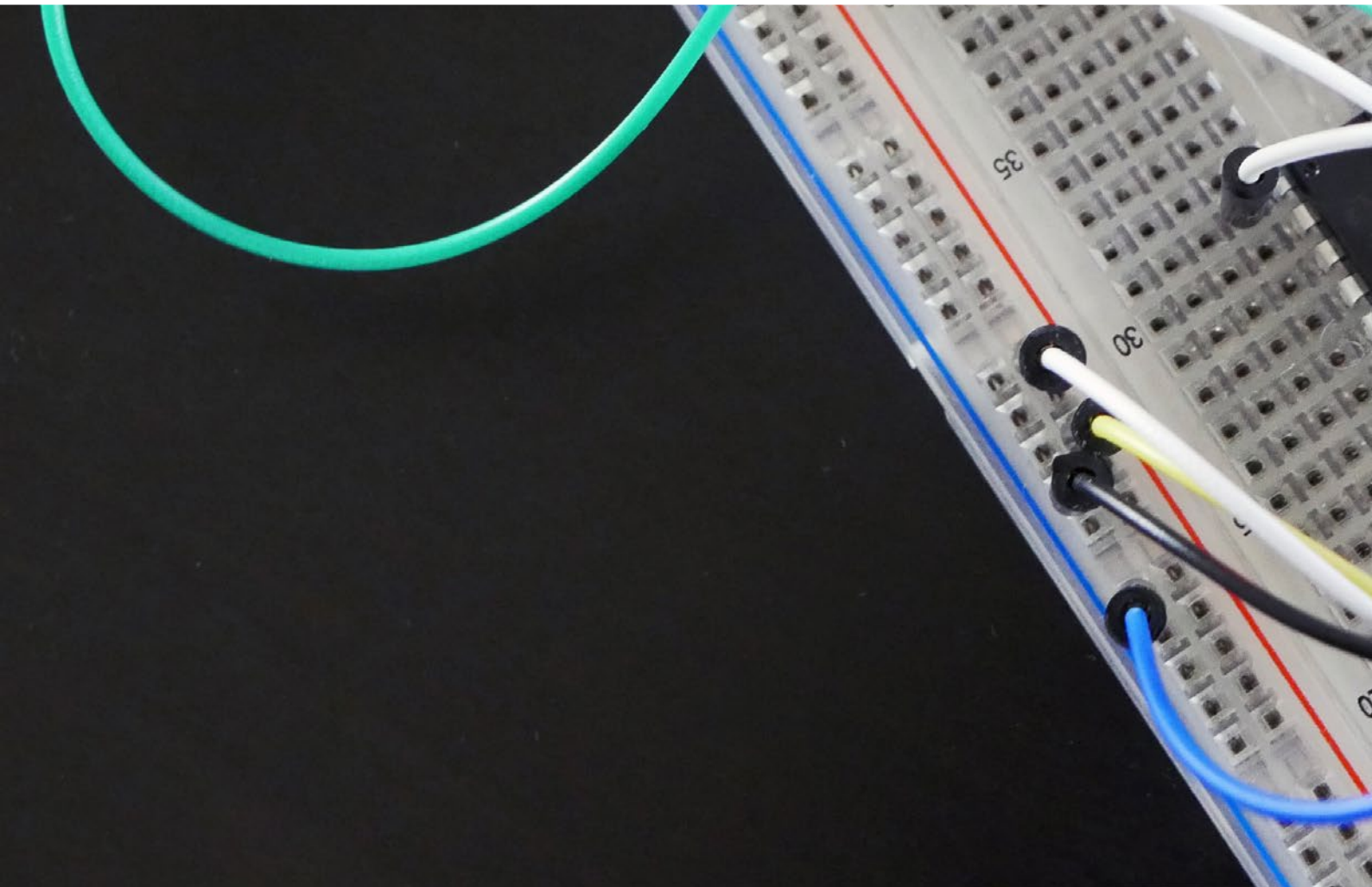
If you want to know more about this code, check out our analog to digital converter tutorial as that goes to the in's and outs of how an Analog to Digital converter works.

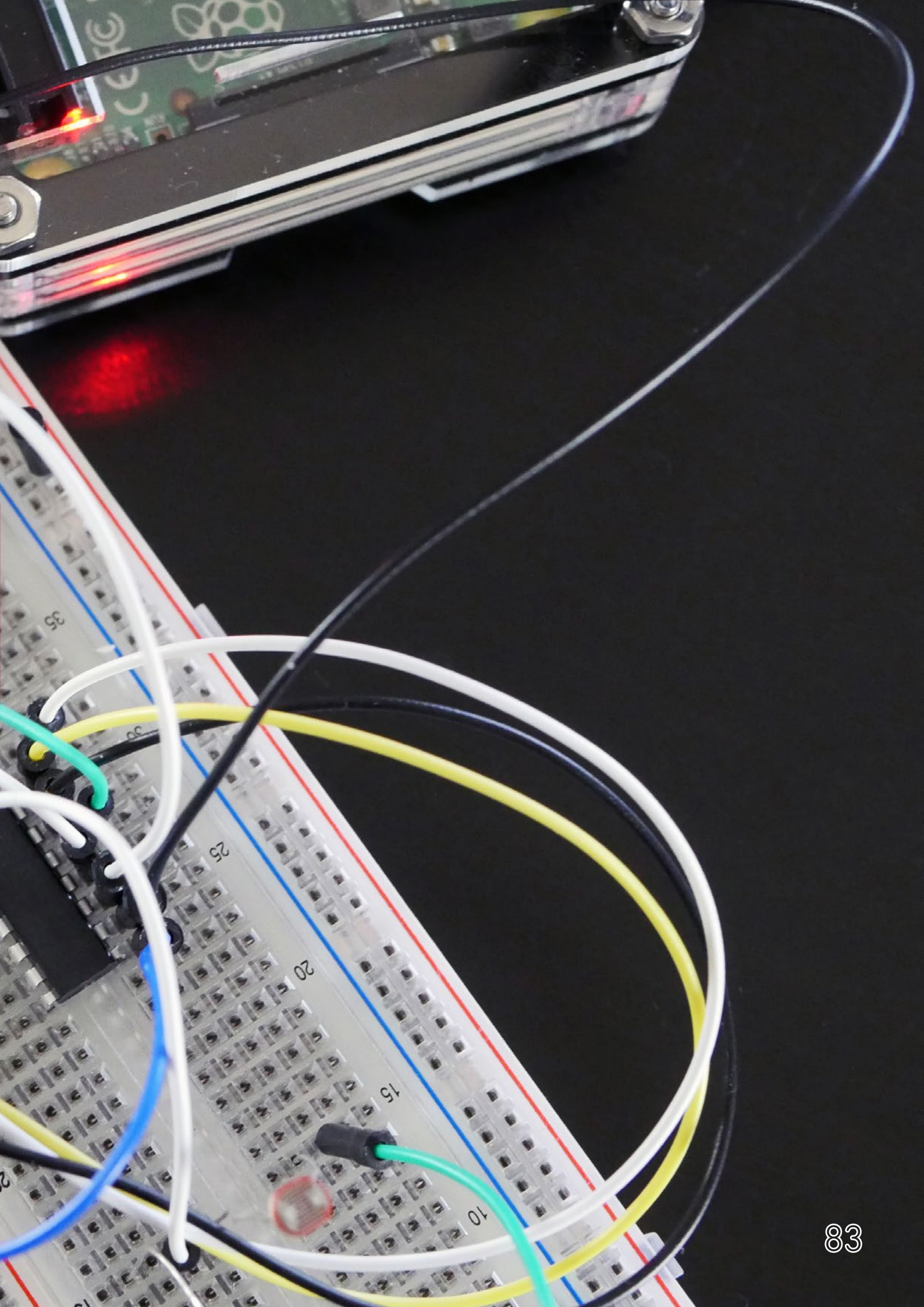
The full code can be found below.

```
#!/usr/bin/python

import spidev
import time

#Define Variables
delay = 0.5
pad_channel = 0

#Create SPI
spi = spidev.SpiDev()
spi.open(0, 0)
#spi.max_speed_hz=1000000

def readadc(adcnum):
    # read SPI data from the MCP3008, 8 channels in total
    if adcnum > 7 or adcnum < 0:
        return -1
    r = spi.xfer2([1, 8 + adcnum << 4, 0])
    data = ((r[1] & 3) << 8) + r[2]
    return data

try:
    while True:
        pad_value = readadc(pad_channel)
        print("----------------------------------------")
        print("Pressure Pad Value: %d" % pad_value)
        time.sleep(delay)
except KeyboardInterrupt:
    pass
```



Circuitry Projects





Utilizing an Analogue to Digital Convertor

Project Description

In this tutorial, I go through the steps of setting up a Raspberry Pi ADC (Analog to digital converter). As you may already know the Raspberry Pi doesn't have any GPIO pins that are analog. The lack of an analog pin makes connecting analog sensors a little more complex.

There are several solutions to the lack of Analog pins like the one I did in the Raspberry Pi LDR tutorial which involved using a capacitor to measure the resistance of the **LDR** (Light Dependent Resistor). A better solution to this would be to use what is known as an analog-digital converter (**MCP3008**).

I also go into setting up the **MCP3008** with **MyDevices Cayenne**, this is a much easier process than writing the code from scratch, but sometimes half the fun is in learning how to code.

If you are interested in making use of Cayenne, don't forget to sign up at https://pimylifeup.com/out/cayenne-adc.

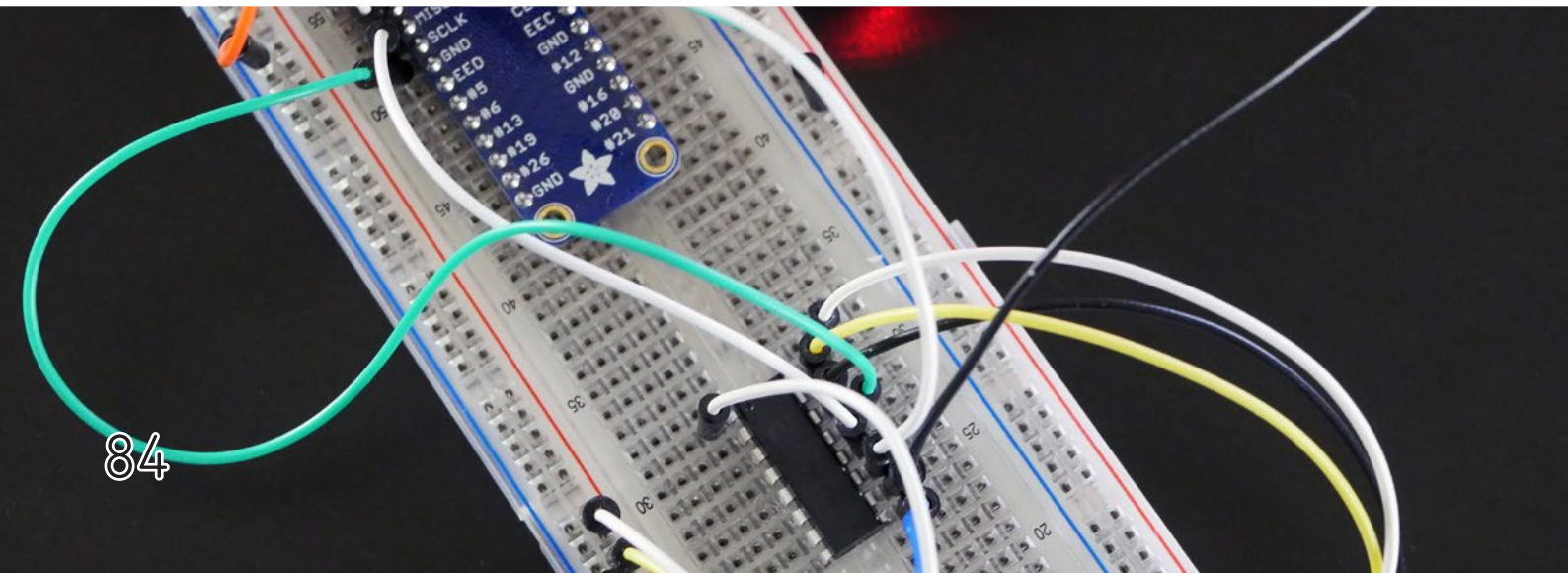
Equipment

Required

- Raspberry Pi
- SD Card (8 GB+ Recommended)
- Network Connection
- Light Sensor (LDR Sensor)
- 10k Resistor
- MCP3008 or similar

Optional

- Raspberry Pi Case
- GPIO Breakout Kit
- Breadboard (Reccomended)
- Breadboard Wire (Reccomended)



The Raspberry Pi ADC Circuit

The circuit for connecting the **MCP3008** to the Pi looks quite involved, but it's all about just connecting the wires up correctly.

The **MCP3008** is the chip that we will be using in our tutorial. There is a lot of technical information on this chip, but I will just touch on the bare basics.

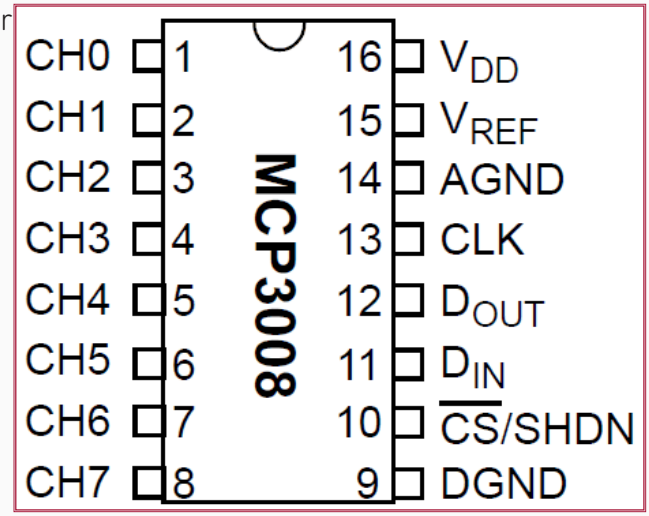
For anyone who is new to microchips, you will find that one end of the chip will have a notch in it. This notch represents the end where **pin 1** is.

Pin 1 is usually the **first pin** on the **left-hand side** with the **notch facing away from you** (see diagram below). Pin one also sometimes has a little dot above it.

This chip makes use of the **SPI** (Serial Peripheral interface bus) which means it will only **require 4 pins** and is relatively easy to communicate to thanks to the **SPIDev library for python**.

The **CH0**->**CH7** pins (**Pins 1-8**) are the analog inputs for the **MCP3008**.

- **DGND (Pin 9)** is the digital ground pin for the chip.
- **CS (Pin 10)** is the chip select.
- **DIN (Pin 11)** is the data in from the Raspberry Pi itself.
- **DOUT (pin 12)** is the data out pin.
- **CLK (Pin 13)** is the clock pin.
- **AGND (Pin 14)** is the analog ground.
- **VREF (Pin 15)** is the analog reference voltage.
- **VDD (Pin 16)** is the positive power pin for the chip.



Putting the circuit together is straightforward. The circuit will look a bit messy because there will be quite a few wires but you can clean this up later if you want to make it more of a permanent project.

In this example, we have included an **LDR** to show how you can get the value on the Raspberry Pi using the **analog to digital converter (ADC)**.

First, connect a **pin 1** to the **positive rail** on the **breadboard** and a **pin 6** to the **ground rail** on the **breadboard**. Also, place the **MCP3008 chip** into the **middle** of the **breadboard**. You can see a diagram of how we set up this circuit on the next page.

- **VDD (Pin 16)** wire this to 3.3V **(Pin 1**, or **positive rail** on the **breadboard)**
- **VREF (Pin 15)** wire this to 3.3V **(Pin 1**, or **positive rail** on the **breadboard)**
- **AGND (Pin 14)** wire this to ground **(Pin 6**, or the **ground rail** on the **breadboard)**
- **CLK (Pin 13)** wire this to GPIO11 **(Pin 23 /** SCLK)
- **DOUT (Pin 12)** wire this to GPIO9 **(Pin 21 /** MISO)
- **DIN (Pin 11)** wire this to GPIO10 **(Pin 19 /** MOSI)
- **CS (Pin 10)** wire this to GPIO8 **(Pin 24 /** CE0**)**
- **DGND (Pin 9)** wire this to GROUND **(Pin 6**, or the **ground rail** on the **breadboard)**

The Raspberry Pi ADC Circuit - Continued

Now once you have done that you will probably want to connect the **LDR** up to the **MCP3008**, this is pretty easy to do.

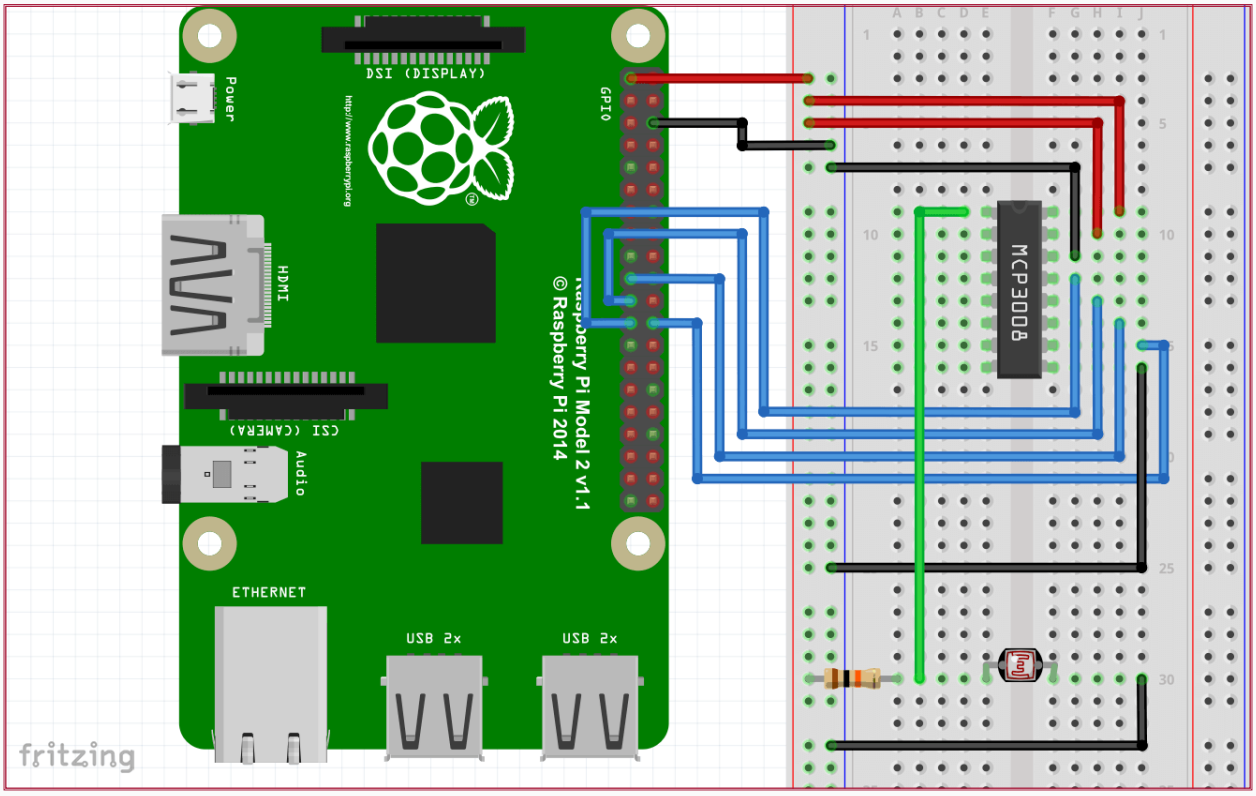
First, place the **LDR** onto the **breadboard**. Next, have the **10k resistor** go to one end of the **LDR** and the other end to the **positive rail**.

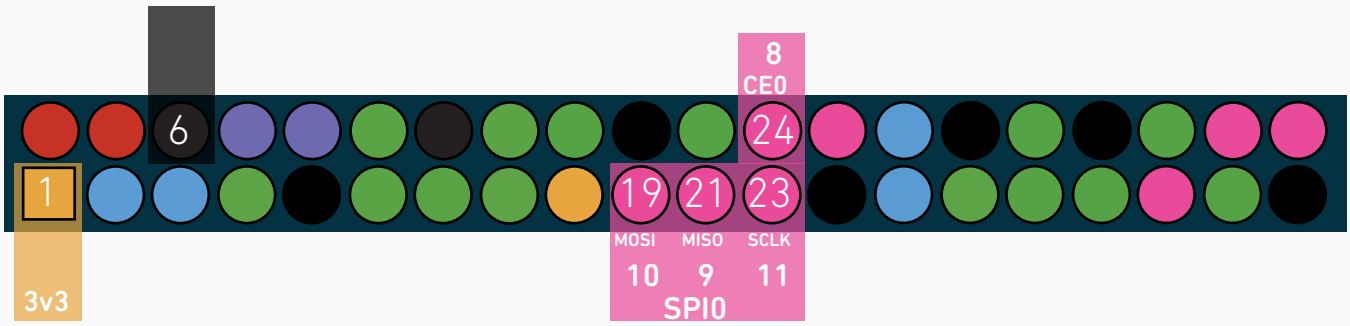
Have a wire go from **in between the LDR** and the **positive wire** to **pin 1 (CH0)** of the **MCP3008**.

Lastly, have a wire go from the **other side of the LDR** to the **ground rail** on the **breadboard**.

If you need further help on making the circuit, you can find a detailed diagram below.

If you are still having trouble, then double check the connections while also making sure all the pins are connected correctly.





The Code

The code for this is simple but is also only handling just one input, the LDR. The code may be a little more complicated the more devices you add.

All the required packages for this tutorial should be installed by default on Raspbian.

If you find it a bit overwhelming than an easier option would be to use a tool such as Raspberry Pi cayenne that I mention below.

If you come across any issues, then check out my guide on the Raspberry GPIO pins in the "**Getting Started**" section of the book.

It goes through how to set them up and install the necessary software.

To quickly download the code straight onto your Pi simply run the following line in the terminal.

```
git clone https://github.com/pimylifeup/Pi-ADC-Example-Code
```

You will need to then move into the folder that contains our script by running the following line.

```
cd ./Pi-ADC-Example-Code
```

You can now launch it by running the following line.

```
sudo python adc-code.py
```

You can stop the script by pressing **ctrl+c**.

I will now just briefly explain the code. At the start of the code, we define the path of where Python is installed.

After this, we import two packages that we will need.
Time allows us to put the script to sleep (Add a delay).
Spidev will enable us to communicate with **SPI devices**, in this case, the **MCP3008**.

```
#!/usr/bin/python

import spidev
import time
```

Next, we define the variables and create our **SPIdev object**.

We create two variables, the first being a delay in milliseconds. Alter this if you want there to be a greater amount of time between updates.

Next is the channel our LDR is located on, this is channel 0/pin 1.

Finally, we create the **SPIdev object**. This object allows us to get the values we need from the device connected the **SPI** enabled pins.

```
#Define Variables
delay = 0.5
ldr_channel = 0

#Create SPI
spi = spidev.SpiDev()
spi.open(0, 0)
```

The Code - Continued

Next, we have the **readadc** function in which we do a couple of things.

Firstly, we check the **parameter value** to make sure it's between **0** & **7**, if not then we **return -1**.

If it is between those numbers, we then proceed to get the value from the channel specified and then return the value.

```
def readadc(adcnum):
    # read SPI data from the MCP3008, 8 channels in total
    if adcnum > 7 or adcnum < 0:
        return -1
    r = spi.xfer2([1, 8 + adcnum << 4, 0])
    data = ((r[1] & 3) << 8) + r[2]
    return data
```

Lastly, we have a while loop that continues until you force quit it by pressing **Ctrl + C**.

The first thing we do is call the **readadc** function with the **ldr_channel** variable as the parameter.

Once we get the value back, we simply print it.

The **%d** refers to a decimal value; we then have our variable after the print statement.

You can print multiple values per line by doing something like this. print '**%s %d' % (name, number)**

After we have printed the data, we delay the script for whatever the delay time is set at. By default, it is set for half a second.

```
while True:
    ldr_value = readadc(ldr_channel)
    print "----------------------------------------"
    print "LDR Value: %d" % (ldr_value)
    time.sleep(delay)
```

Using the ADC with Raspberry Pi Cayenne

We will now quickly go through the steps to set up the **MCP3008** with **Cayenne**. I will also go through how to setup the **LDR**.

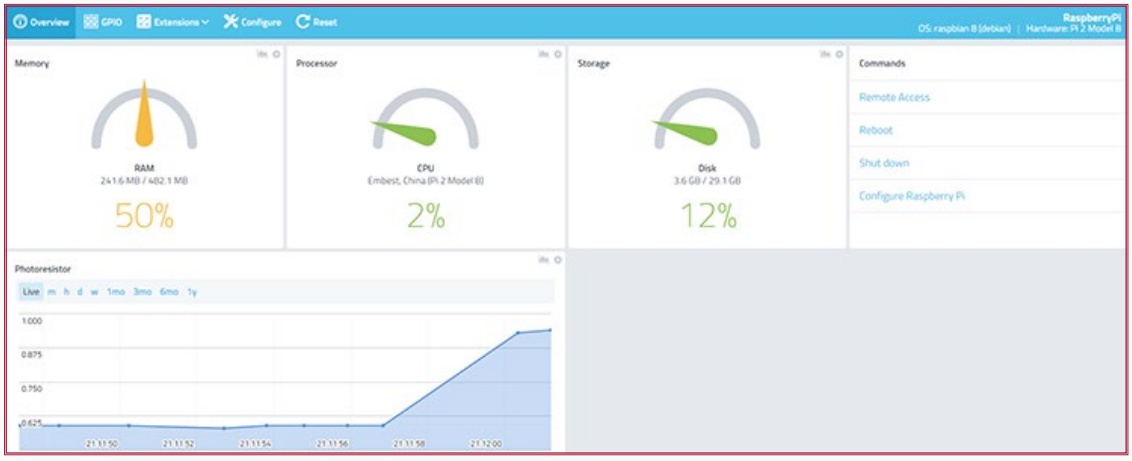
If you haven't already gone through our getting started with Cayenne guide then make sure you do as the tutorial goes through the steps to getting it installed and most of the basics you need to know for creating your smart applications.

If you just want to jump straight into **Cayenne** you can go straight to their signup page here: https://pimylifeup.com/out/cayenne-adc

1. To begin first go to **Add new** and then select **Device** from the dropdown box.
2. Then either search or select "**Analog converter**".
3. In this list select the **MCP3008**
4. Now select the **Raspberry Pi** and then **SPI Chip-Select 0**
5. This will now bring up a screen that shows you all the values of the channels.

Now to add the **LDR**, the process is pretty much the same, but I will go through it anyway.

1. Go back up to **Add new** and then **Select device**.
2. Find or select the **Photoresistor (Luminosity Sensor)**.
3. For this one select your **Raspberry Pi**, your preferred widget (I went **Graph**), then select the **MCP3008** for the analog converter and finally **Channel 0** for the channel.
4. This should now add without any issue. It will now produce a graph from the values it gets from the analog converter.



How to setup a 16x12 LCD Display

Project Description

In this tutorial, we go through the steps on how to set up a Raspberry Pi LCD 16×2 display. This display is a cool way to display some information from the Pi without needing any expensive or complicated display setup.

A 16×2 display unlike a touchscreen or a standard LCD screen is best used to display short messages or information.

You will find it extremely handy when you only need to display some essential data but don't need anything too large and expensive.

This tutorial will go through the basics of getting the screen setup and is incredibly handy for anyone who is completely new to the circuitry.

Equipment

Required

- Raspberry Pi
- SD Card (8 GB+ Recommended)
- 16×2 LCD with header pins
- 10k Potentiometer
- Breadboard
- Breadboard Wire

Optional

- Raspberry Pi Case
- GPIO Breakout Kit
- Network Connection
- USB Keyboard
- USB Mouse

The Raspberry Pi LCD 16×2 Circuit

It may look like that there is quite a bit that makes up this circuit, but it just involves connecting the wires up correctly to and from the display.

The **potentiometer** that's in the circuit is essential for **controlling the screen brightness**. If you do not have one, then you can try swapping this out for a **resistor**.

If you do use a a standard resistor, then try using anything between **5k** and **10k ohms**. You may need to try out a few different values before getting the perfect resistance.

A typical **16×2 LCD** has **16 pins**, but not all of them need to be used. In this circuit, we will only need to use **4 of the data bus lines** since we are just going to use it in **4-bit mode**.

You will find that most **16 connector displays** will be using an **HD44780 controller**. This controller makes the display versatile and can be used across a wide range of devices.

The **typical pin layout** of the **LCD board** can be found below.

PIN NO	Symbol	Fuction
1	VSS	GND
2	VDD	+5V
3	V0	Contrast adjustment
4	RS	H/L Register select signal
5	R/W	H/L Read/Write signal
6	E	H/L Enable signal
7	DB0	H/L Data bus line
8	DB1	H/L Data bus line
9	DB2	H/L Data bus line
10	DB3	H/L Data bus line
11	DB4	H/L Data bus line
12	DB5	H/L Data bus line
13	DB6	H/L Data bus line
14	DB7	H/L Data bus line
15	A	+4.2V for LED
16	K	Power supply for BKL(0V)

Assembling the 16×2 LCD

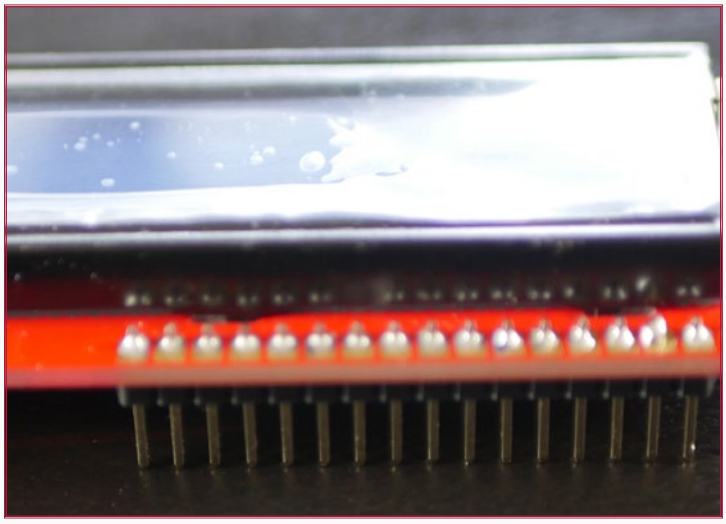
You will find that most 16×2 displays do not come with the header pins pre-soldered. Of course this means you will need to solder some header pins on before you can use it. It's tough getting a good connection to the screen without them. Soldering the pins is a straightforward task and should only take a few minutes for anyone who has soldered before.

1. First, snap the header pins, so you have 1 line of 16.

2. Place the header pins up through the holes in the displays circuit board. The short side of the header pins should stick up. One tip is to use a breadboard to hold the pins in place.

3. Now using a hot soldering iron and some solder, slowly solder each of the pins.

4. It's now ready for use.



Connecting Everything Up

Connecting the 16×2 LCD to the Raspberry Pi is straightforward. There will be quite a few wires to connect, but there isn't anything overly complex.

There is one thing that you should be aware of before you jump in and start assembling the circuit. Since we do not want 5v feeding back into the Pi (Pi's GPIO pins are rated 3v3) we will need to make the read/write pin of the LCD go to ground.

In the steps below the physical/logical numbering of the pins are in the brackets otherwise it's the GPIO numbering.

1. Place a wire from 5v (**Pin 2**) to the positive rail on the breadboard.

2. Place a wire from the ground (**pin 6**) to the ground rail on the breadboard.

3. Place the 16×2 display on the breadboard

.

4. Place the potentiometer onto the breadboard.

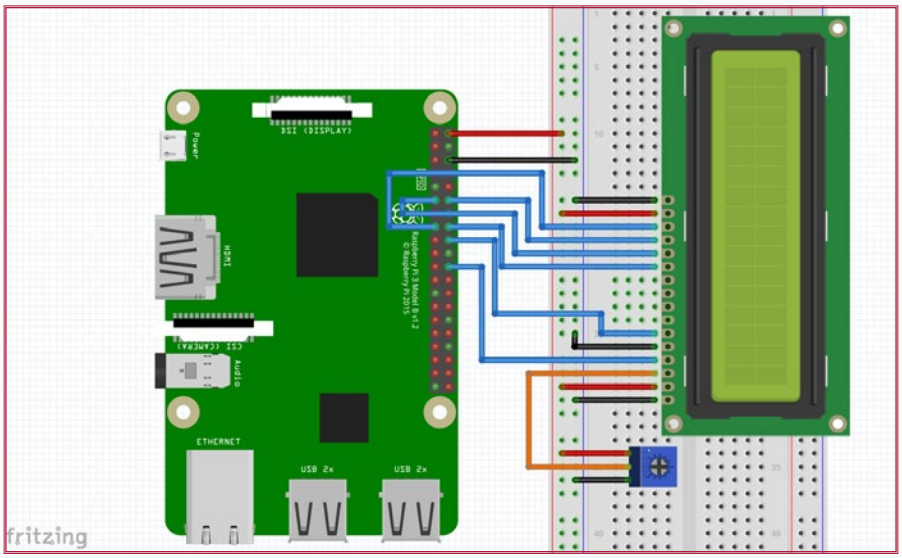
5. Connect the positive and ground pins to the relevant rails on the breadboard.

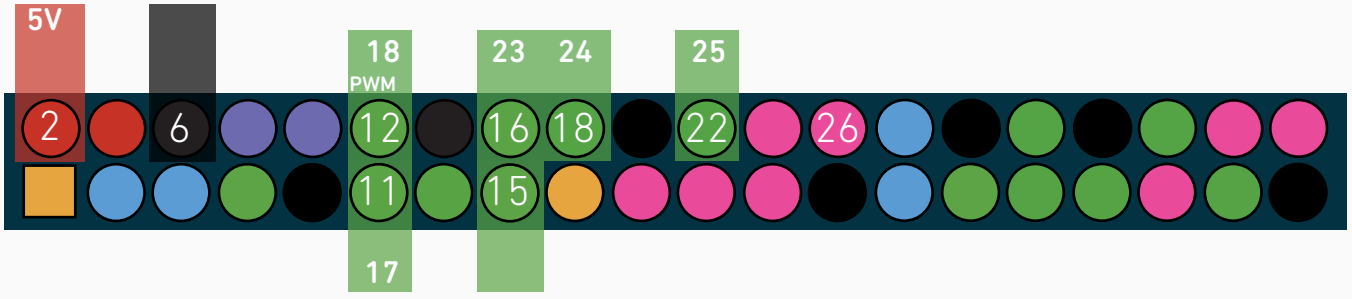
Connecting Everything Up

Starting from pin 1 of the LCD do the following or simply refer to the circuit diagram below. Pin 1 of the screen is the pin closest to two edges of the board.

1. **Pin 1** (**Ground**) goes to the **ground rail**.
2. **Pin 2** (**VCC/5v**) goes to the **positive rail**.
3. **Pin 3** (**V0**) goes to the **middle wire** of the **potentiometer**.
4. **Pin 4** (**RS**) goes to GPIO25 (**Pin 22**)
5. **Pin 5** (**RW**) goes to the **ground rail**.
6. **Pin 6** (**EN**) goes to GPIO24 (**Pin 18**)
7. **Pin 11** (**D4**) goes to GPIO23 (**Pin 16**)
8. **Pin 12** (**D5**) goes to GPIO17 (**Pin 11**)
9. **Pin 13** (**D6**) goes to GPIO18 (**Pin 12**)
10. **Pin 14** (**D7**) goes to GPIO22 (**Pin 15**)
11. **Pin 15** (**LED +**) goes to the **positive rail**.
12. **Pin 16** (**LED -**) goes to the **ground rail**.

That's all you need to do; the screen should now be able to turn on and communicate with the Raspberry Pi without any issues. If you are having trouble, then refer to the circuit diagrams below.





Code to communicate with the 16×2 Display

On the latest version of Raspbian, all the required packages are pre-installed for communicating with GPIO devices. You should also find that python is already installed.

Required Library

In this example, I am going to install and use the library from **Adafruit**. It's designed for **Adafruit LCD boards** but will also work with other brands as well. If your display board uses an **HD44780 controller**, then it should work with no issues at all.

First clone the required GIT directory to the Raspberry Pi by running the following command.

```
git clone https://github.com/adafruit/Adafruit_Python_CharLCD.git
```

Next change into the directory we just cloned and run the setup file.

```
cd ./Adafruit_Python_CharLCD
sudo python setup.py install
```

Once it's done installing you can now call the Adafruit library in any python script on the Raspberry Pi. To include it just add the following line at the top of the python script. You can then initialize the board and perform actions with it.

```
import Adafruit_CharLCD as LCD
```

Communicating with the Display

Communicating with the Raspberry Pi LCD 16×2 display is very easy thanks to the library provided by Adafruit. It makes it simple to write Python scripts to setup and alter the display.

In the folder that we just downloaded there are a few examples of how to use the LCD library. It's important that before you run any of these examples that you update the pin variables at the top of the file. If you followed my circuit, then the values below are the correct ones.

```
lcd_rs      = 25
lcd_en      = 24
lcd_d4      = 23
lcd_d5      = 17
lcd_d6      = 18
lcd_d7      = 22
lcd_backlight = 4
lcd_columns = 16
lcd_rows = 2
```

If you want to check out one of the examples simply open up the file by entering the following.

```
cd ~/Adafruit_Python_CharLCD/examples/
sudo nano char_lcd.py
```

In here update the pin configuration values to the ones listed above. Once done simply exit by pressing **CTRL+X** then **Y**.

Now to run this code just enter python followed by the file name into the terminal (including the extension).

```
python char_lcd.py
```

Functions & Python Code

I will go through some of the important methods that you will need to know about interacting with the screen using Python.

To initialize the pins, you will need to call the following class. Make sure all the variables that are being passed as parameters are defined before calling the class.

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

Once that's done you can then change the display to however you need it. I will quickly go through some of the methods that are available to you using the Adafruit library.

home() – This method will move the cursor back to home which is the first column on the first line.

clear() – This method clears the LCD so that it's completely blank.

set_cursor(col, row) – This method will move the cursor to a specific position. You specify the position by passing the column and row numbers as parameters. **Eg. set_cursor(1,4)**

enable_display(enable) – This **enables** or **disables** the display. Set it to **true** to enable it.

show_cursor(show) – This either **shows** or **hides** the cursor. Set it to **true** if you want the cursor to be displayed.

blink(blink) – Turns **on** or **off** a blinking cursor. Again, set this to **true** if you want the cursor to be blinking.

move_left() or move_right() – Moves the cursor either **left** or **right** by **one** position.

set_right_to_left() or set_left_to_right() – Sets the cursor **direction** either **left to right** or **right to left**.

autoscroll(autoscroll) – If **autoscroll** is set to **true** then the text will **right justify** from the cursor. If set to **false** it will **left justify** the text.

message(text) – Simply **writes text** to the display. You can also include **newlines (\n)** in your message.

There are a few more methods available but it's unlikely that you will need to use them.

If you want to find all the methods that are available then open up the **Ardafruit_CharLCD.py** file located within the **Adafruit_CharLCD** folder, this can be found in the **Adafruit_Python_CharLCD** folder.

```
sudo nano ~/Adafruit_Python_CharLCD/Adafruit_CharLCD/Ardafruit_CharLCD.py
```

Functions & Python Code - Continued

Below is a straightforward script that I have put together that allows the user to input text that is then displayed on the screen.

```
#!/usr/bin/python
# Example using a character LCD connected to a Raspberry Pi
import time
import Adafruit_CharLCD as LCD

# Raspberry Pi pin setup
lcd_rs = 25
lcd_en = 24
lcd_d4 = 23
lcd_d5 = 17
lcd_d6 = 18
lcd_d7 = 22
lcd_backlight = 2

# Define LCD column and row size for 16x2 LCD.
lcd_columns = 16
lcd_rows = 2

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

lcd.message('Hello\nworld!')
# Wait 5 seconds

time.sleep(5.0)
lcd.clear()
text = raw_input("Type Something to be displayed: ")
lcd.message(text)

# Wait 5 seconds
time.sleep(5.0)
lcd.clear()
lcd.message('Goodbye\nWorld!')

time.sleep(5.0)
lcd.clear()
```

If the display isn't showing anything when your python script is running, then it is likely the pins defined in your script are wrong. Double check the values and double check the connections on the breadboard.

Further Work

This tutorial just covers the basics of setting up the **16×2 LCD** correctly with the Raspberry Pi. There is a lot that you can do with this cool display. For example, you can have a script launch on startup that can display specific values such as IP address, time, temperatures and so much more.

There is also a vast range of sensors that you should try incorporating with this display. Something like the **DS18B20 temperature sensor** would work great with the display. Just update the display every few seconds with the new temperature.

Setting up a RFID RC522 Reader

Project Description

In this tutorial, I will be walking you through the steps on how to setup and wire the RFID RC522 with your Raspberry Pi. This circuit is fun to play around with and opens you up to quite a wide variety of different projects from using it as an attendance system to using it to open a lock.

The RFID RC522 is a very low-cost RFID reader and writer that is based on the MFRC522 microcontroller. This microcontroller provides its data through the SPI protocol and works by creating a 13.56MHz electromagnetic field that it uses to communicate with the RFID tags.

Make sure that the tags you purchase for your RFID RC522 operate on the 13.56MHz frequency otherwise we will fail to read them.

We will be showing you how to wire up your RFID RC522 to your Raspberry Pi in this tutorial, alongside showing you how to write Python scripts to interact with your RFID RC522 to be able to read and write to your RFID Tags

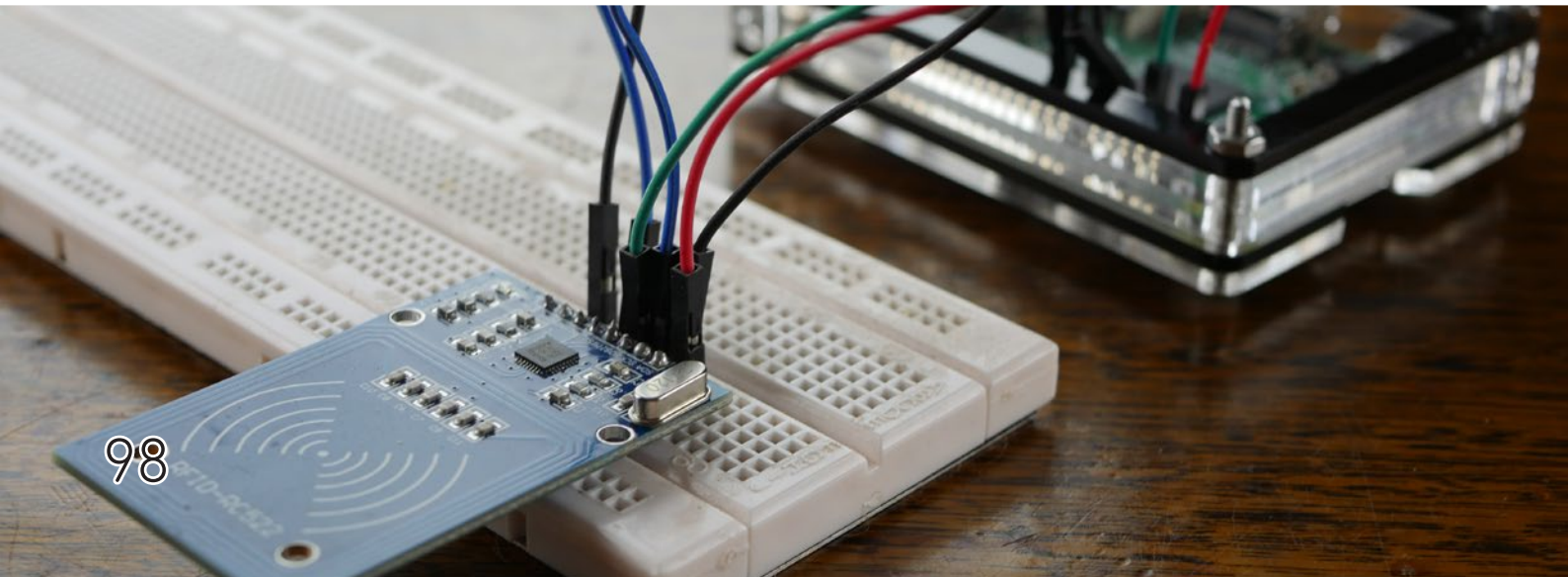
Equipment

Required

- Raspberry Pi
- SD Card (8 GB+ Recommended)
- RC522 RFID Reader
- Breadboard Wire

Optional

- Raspberry Pi Case
- GPIO Breakout Kit
- Network Connection
- USB Keyboard
- USB Mouse



Assembling the RFID RC522

One thing you will notice when purchasing an RFID RC522 Reader is that 90% of them don't come with the header pins already soldered in. Meaning you will have to do it yourself, luckily soldering header pins is a rather simple task, even for beginners.

1. First off if the header pins you received with your RC522 isn't the correct size snap them down, so you only have a single row of 8 pins.

2. Place the header pins up through the holes of your RC522 circuit. One handy trick is to put the long side of the header pins into a breadboard and then putting the circuit over the top of the header pins. The breadboard will hold the pins tightly making it easier to solder them to the RFID RC522 circuit.

3. Now using a hot soldering iron and some solder, slowly solder each of the pins. Remember it is best to heat the joint slightly before applying solder to it, this will ensure that the solder will adhere more to the joint and reduce the chances of creating a cold joint.

4. With the header pins now soldered to your RFID RC522 circuit, it is now ready to use, and you can continue with the tutorial.

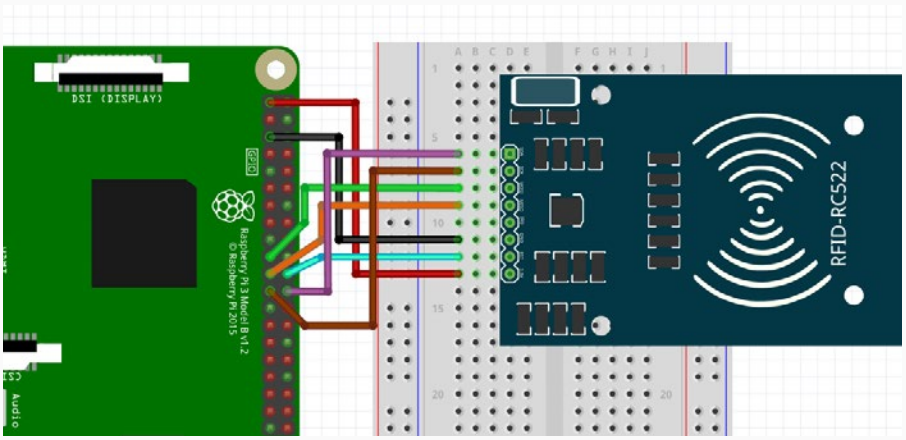
Wiring the RFID RC522 to your Raspberry Pi

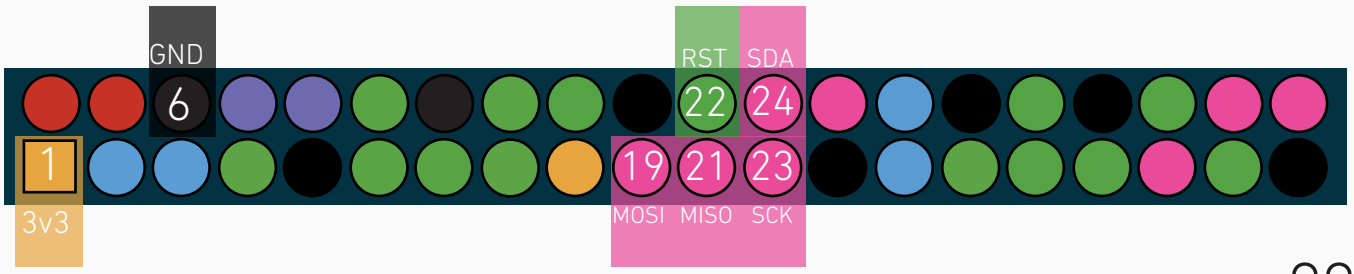
On your RFID RC522 you will notice that there are 8 possible connections on it, these being **SDA** (**Serial Data Signal**), **SCK** (**Serial Clock**), **MOSI** (**Master Out Slave In**), **MISO** (**Master In Slave Out**), **IRQ** (**Interrupt Request**), **GND** (**Ground Power**), **RST** (**Reset-Circuit**) and **3.3v** (**3.3v Power In**). We will need to wire all of these but the **IRQ** to our Raspberry Pi's GPIO pins.

You can either wire these directly to the GPIO Pins or like we did in this tutorial, plug the RFID RC522 into our Breadboard then wire from there to our Raspberry Pi's GPIO Pins.

Wiring your RFID RC522 to your Raspberry Pi is relatively simple, with it requiring you to connect just 7 of the GPIO Pins directly to the RFID RC522. Follow our table below, and check out our GPIO guide to see the positions of the GPIO pins that you need to connect your RC522 to.

- **SDA** connects to **Pin 24**.
- **SCK** connects to **Pin 23**.
- **MOSI** connects to **Pin 19**.
- **MISO** connects to **Pin 21**.
- **GND** connects to **Pin 6**.
- **RST** connects to **Pin 22**.
- **3.3v** connects to **Pin 1**.





Setting up Raspbian for the RFID RC522

Before we begin the process of utilizing the RFID RC522 on our Raspberry Pi, we will first have to make changes to its configuration.

By default, the Raspberry Pi has the SPI Interface disabled, but luckily we can easily change that.

1. Let's begin by first opening the **raspi-config** tool, we can do this by opening terminal and running the following command:

```
sudo rapsi-config
```

2. This tool will load up a screen showing a variety of different options. If you want an in-depth look at the tool, check out our "**Getting Started with the Raspberry Pi**" book.

On here use the **arrow keys** to select "**5 Interfacing Options**". Once you have this option selected, press **Enter**.

3. Now on this next screen, you want to use your **arrow keys** again to select "**P4 SPI**", again press **Enter** to select the option once it is highlighted.

4. You will now be asked if you want to enable the SPI Interface, select **Yes** with your **arrow keys** and press **Enter** to proceed. You will need to wait a little bit while the **raspi-config** tool does its thing in enabling SPI.

5. Once the SPI interface has been successfully enabled by the raspi-config tool you should see the following text appear on the screen, "**The SPI interface is enabled**".

Before the SPI Interface is fully enabled, we will first have to restart the Raspberry Pi. To do this first get back to the terminal by pressing **Enter** and then **ESC**.

Type the following command into the terminal on your Raspberry Pi to restart your Raspberry Pi.

```
sudo reboot
```

6. Once your Raspberry Pi has finished rebooting, we can now check to make sure that it has in fact been enabled. The easiest way to do this is to run the following command to see if **spi_bcm2835** is listed.

```
lsmod | grep spi
```

If you see **spi_bcm2835**, then you can proceed on with this tutorial and skip on to the next section. If for some reason it had not appeared when you entered the previous command, try following the next 3 steps.

7. If for some reason the SPI module has no activated, we can edit the boot configuration file manually by running the following command on our Raspberry Pi.

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

8. Within the configuration file, use **Ctrl +W** to find "**dtparam=spi=on**".

If you have found it, check to see if there is a **#** in front of it, if there is remove it. If you can't find the line at all, just add "**dtparam=spi=on**" to the bottom of the file.

Once you have made the changes, you can press **Ctrl + X** then pressing **Y** and then **Enter** to save the changes.

You can now proceed to **Step 5** again, rebooting your Raspberry Pi then checking to see if the module has been enabled.

Getting Python ready for the RFID RC522

Now that we have wired up our RFID RC522 circuit to the Raspberry Pi we can now power it on and begin the process of programming simple scripts in Python to interact with the chip.

The scripts that we will be showing you how to write will show you how to read data from the RFID chips and how to write to them. These will give you the basic idea of how data is dealt with and will be the basis of further RFID RC522 tutorials.

1. Before we start programming, we first need to update our Raspberry Pi to ensure it's running the latest version of all the software. Run the following two commands on your Raspberry Pi to update it.

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

2. Now the final thing we need before we can proceed is to install the **python2.7-dev** package, just run the following command on your Raspberry Pi to install it.

```
sudo apt-get install python2.7-dev
```

3. To begin, we must first clone the Python Library SPI Py and install it to our Raspberry Pi. This library helps handle interactions with the SPI and is a key component to this tutorial as we need it for the Raspberry Pi to interact with the RFID RC522.

Run the following two commands on your Raspberry Pi to clone the source code.

```
cd ~
git clone https://github.com/lthiery/SPI-Py.git
```

4. With the SPI Py Python Library now cloned to our Raspberry Pi, we need to install it. Installing the library is incredibly simple as all we need to do is change into its directory and run a simple python command on our Raspberry Pi.

```
cd ~/SPI-Py
sudo python setup.py install
```

5. Now that we have installed SPI-Py we can now clone our RFID RC522 Python code from the PiMyLifeUp Github. There are two files included in this repository:

- **MFRC522.py** which is an implementation of the RFID RC522 circuit.
- **SimpleMFRC522.py** that takes the **MFRC522.py** file and greatly simplifies it.

To clone this repository, you can type the following two commands into your Raspberry Pi.

```
cd ~
git clone https://github.com/pimylifeup/MFRC522-python.git
```

6. With the repository now saved to our Raspberry Pi, we can begin programming for our RFID RC522. To start off with we will be showing you how to write data to your RFID cards by using the RC522. Just go to our next section to begin programming our first Python script.

Writing with the RFID RC522

For our first Python script, we will be showing you how to write data from the RC522 to your RFID tags. Thanks to the SimpleMFRC522 script this will be relatively simple.

1. Begin by changing directory into our newly cloned folder, and begin writing our **Write.py** python script.

```
cd ~/MFRC522-python
sudo nano Write.py
```

2. Within this file, write the following lines of code. This code will ask you for text to input and then write that text to the RFID Tag.

```
#!/user/bin/env python

import RPi.GPIO as GPIO
import SimpleMFRC522
```

The first line of this segment of code helps tell the terminal how to interpret the file. It lets it know that it should use Python when executing it and not something else such as Bash.

Our first import, **RPi.GPIO** has all the functions needed to interact with the GPIO Pins. We need this to make sure they are cleared when the script finishes running.

The second import, imports our **SimpleMFRC522** library, this is what we will use to talk with the RFID RC522, it dramatically simplifies dealing with the chip compared to the base MFRC522 library.

```
reader = SimpleMFRC522.SimpleMFRC522()
```

This line creates a copy of the SimpleMFRC522 as an object, runs its setup function then stores it all in our reader variable.

```
try:
        text = raw_input('New data:')
        print("Now place your tag to write")
        reader.write(text)
        print("Written")
```

Our next block of code we keep within a **try** statement, this is so we can catch any exceptions and clean up properly. Make sure that you retain the 'tabs' after **try:** as Python is whitespace sensitive, and it is how it differs between blocks of code.

The second line here reads in an input from the command line. We use **raw_input** in Python 2.7 to read in all input and store it in our text variable.

With the third line, we utilize **print()** to notify the user that they can now place their RFID tag down onto the reader for writing.

Afterward, on our fourth line of code we use our reader object to write the values we stored in the **text** variable to the RFID tag, this will tell the RFID RC522 Circuit to write the text values to a certain sector.

Finallym on the 5th line of code, we use **print()** again to notify the user that we have successfully written to the RFID tag.

```
finally:
    GPIO.cleanup()
```

Our final two lines of code handle exiting of the script. Finally, always occurs after the try statement, meaning no matter what we run the **GPIO.cleanup()** function. Doign this is crucial as failing to cleanup can prevent other scripts from working correctly.

Writing with the RFID RC522

3. Once you have finished writing your script, it should look something like below.

```
#!/user/bin/env python

import RPi.GPIO as GPIO
import SimpleMFRC522

reader = SimpleMFRC522.SimpleMFRC522()

try:
        text = raw_input('New data:')
        print("Now place your tag to write")
        reader.write(text)
        print("Written")
finally:
        GPIO.cleanup()
```

Once you are happy that the code looks correct, you can save the file by pressing **Ctrl + X** then pressing **Y** and then finally hitting **Enter**.

4. Now that we have written our script, we will want to finally test it out. Before testing out the script make sure that you have an RFID tag handy. Once ready, type the following command into your Raspberry Pi's terminal.

```
sudo python Write.py
```

5. You will be asked to write in the new data, in our case we are going to just type in **pimylifeup** as its short and simple. Press **Enter** when you are happy with what you have written.

6. With that done, just place your RFID Tag on top of your RFID RC522 circuit. As soon as it detects it, it will immediately write the new data to the tag. You should see "**Written**" appear in your command line if it was successful.

You can look at our example output below to see what a successful run looks like.

```
pi@raspberrypi:~/MFRC522-python $ sudo python Write.py
New data:pimylifeup
Now place your tag to write
Written
```

7. You have now successfully written your Write.py script. We can now proceed to show you how to read data from the RFID RC522 in the next segment of this tutorial.

Reading with the RFID RC522

Now that we have written our script to write to RFID tags using our RC522 we can now write a script that will read this data back off the RFID Tag.

1. Let's start off by changing directory to make sure we are in the right place, and then we can run **nano** to begin writing our **Read.py** script.

```
cd ~/MFRC522-python
sudo nano Read.py
```

2. Within this file, write the following lines of code. This script will sit and wait till you put your RFID tag on the RFID RC522 reader, it will then output the data it reads off the tag.

```
#!/user/bin/env python

import RPi.GPIO as GPIO
import SimpleMFRC522
```

The first line of code tells the operating system how to handle the file when a user executes it. Otherwise, it will try and just run it as a normal script file and not a python file.

The first import is, **RPi.GPIO**. This library contains all the functions to deal with the Raspberry Pi's GPIO pins. We mainly import this to ensure that we clean up when the script finishes executing.

The second import is, **SimpleMFRC522**. This script contains a few helper functions to make it an awful lot easier to deal with writing and to read from the RFID RC522, without it our simple scripts would become quite long.

```
reader = SimpleMFRC522.SimpleMFRC522()
```

This line is quite important as it calls SimpleMFRC522's creation function and then stores that into our reader variable as an object so we can interact with it later

```
try:
    id, text = reader.read()
    print(id)
    print(text)
```

This next block of code is contained within a **try** statement, we use this so we can catch any exceptions that might occur and deal with them nicely. You need to ensure that you use the '**tabs**' as shown after **try:** as Python is whitespace sensitive.

The second line in this block of code makes a call to our reader object, in this case it basically tells the circuit to begin reading any RFID tag that is placed on top of the RC522 reader.

With the third and fourth lines we utilize **print()** to print out the information that we received from reading the RFID Chip, this includes the ID associated with the RFID tag and the text that was stored on the tag.

```
finally:
    GPIO.cleanup()
```

The two last lines of code handle the termination of the script. The **finally** statement always triggers after the try statement even if we get an exception.

This statement ensures that no matter what we run the **GPIO.cleanup()** function.

It is quite crucial as failing to clean up the GPIO can prevent other scripts from working correctly.

Reading with the RFID RC522

3. Now that you have finished writing your Read.py script for your RFID RC522 it should look something like what is shown below:

```
#!/user/bin/env python

import RPi.GPIO as GPIO
import SimpleMFRC522

reader = SimpleMFRC522.SimpleMFRC522()

try:
        id, text = reader.read()
        print(id)
        print(text)
finally:
        GPIO.cleanup()
```

Once you are sure you have entered the code correctly, then you can save the file by pressing **Ctrl + X** then pressing **Y** and then finally hitting **Enter**.

4. Now that we have finally finished our Read.py script we need to test it out. Before we test out the script, grab one of the RFID tags that you want to read. Once that you are ready, type the following command into your Raspberry Pi's terminal.

```
sudo python Read.py
```

5. With the script now running, all you need to do is place your RFID Tag on top of your RFID RC522 circuit. As soon as the Python script detects the RFID tag being placed on top, it will immediately read the data and print it back out to you.

An example of what a successful output would look like is displayed below.

```
pi@raspberrypi:~/MFRC522-python $ sudo python Read.py
827843705425
pimylifeup
```

7. If you successfully receive data back from your Read.py script with the text that you pushed to the card using your **Write.py** script, then you have successfully setup your Raspberry Pi to connect with your RFID RC522 Circuit.

We will be going into more depth with these scripts and the RFID RC522 in later tutorials, exploring how to set up an attendance system among other things.

Read and Write From A Serial Port

Project Description

In this tutorial, we will be showing you how to read and write data through the serial GPIO connections that are made available to you on your Raspberry Pi.

We will be showing you how to do these serial writes by using an RS232 to TTL Adapter to create a loop back to the Raspberry Pi. The same concept will work with any serial device.

You will be learning what GPIO pins you need to utilize to be able to wire up to your serial device to the Raspberry Pi. We will also be showing the steps you must go through to allow the Raspberry Pi to read and write through the TX and RX GPIO pins.

We will also be teaching you how you would read and write data through the serial ports using the Python programming language. This tutorial should give you a good idea how you would deal with normal serial devices and not just a loopback to the Raspberry Pi.

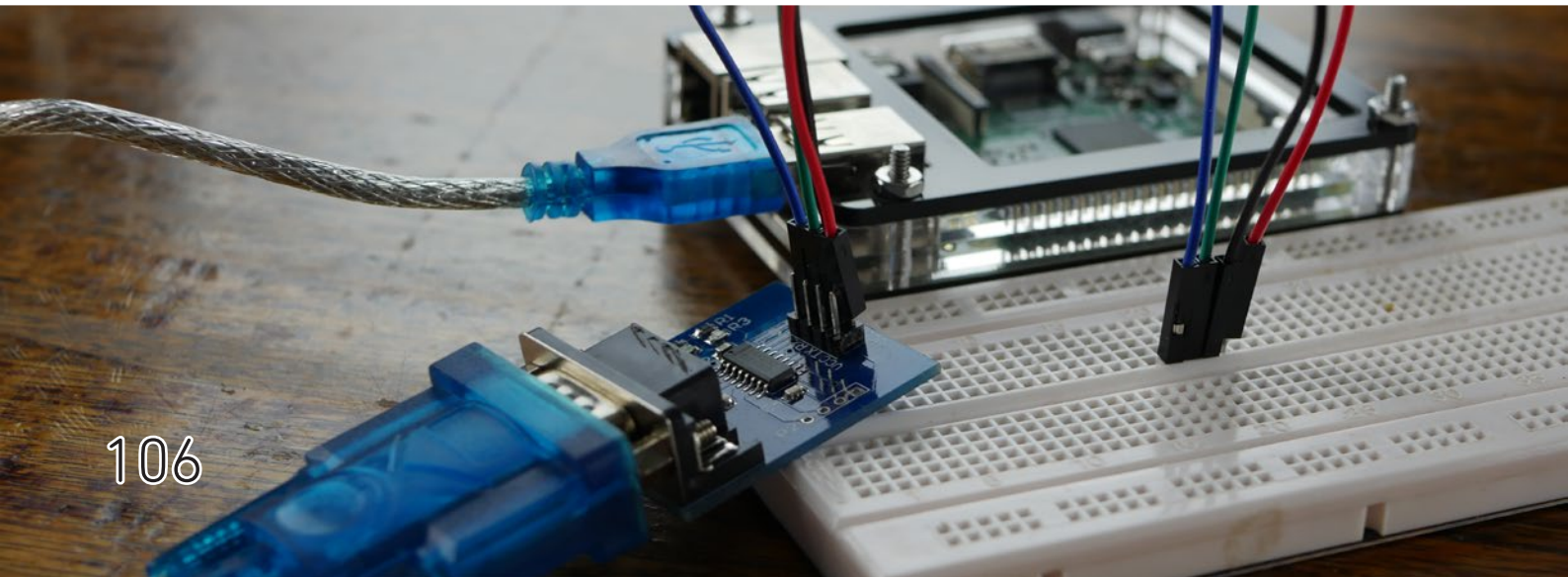
Equipment

Required

- Raspberry Pi
- SD Card (8 GB+ Recommended)
- RS232 to TTL Adapter
- RS232 to USB adapter
- Breadboard wire
- Breadboard

Optional

- Raspberry Pi Case
- GPIO Breakout Kit
- Network Connection
- USB Keyboard
- USB Mouse



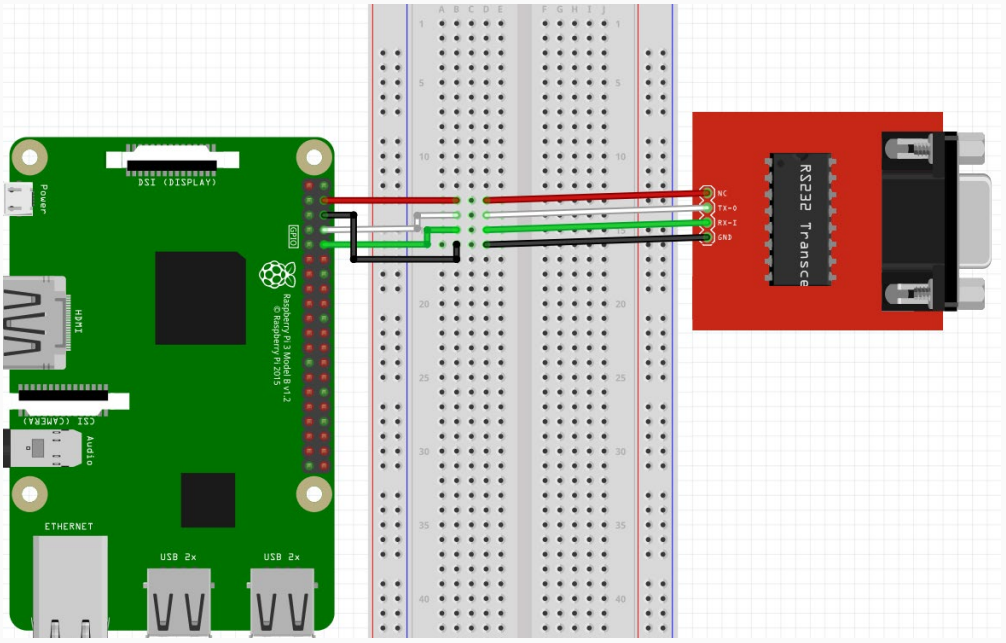
Wiring your Raspberry Pi for Serial

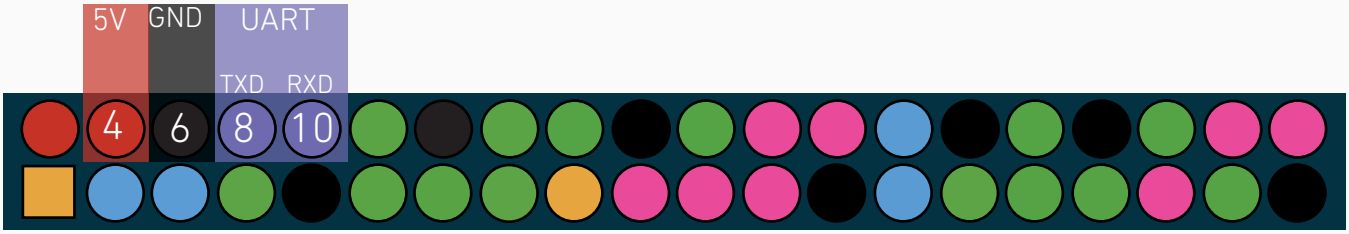
On your **RS232 to TTL adapter** you should find at least four connections, some circuits do come with more connections but the only four you need is: **VCC** (**IC Power-supply pin**), **TX** (**Transmitted Data**), **RX** (**Received Data**) and **GND** (**Ground power-supply pin**)

You can connect the wires directly to the GPIO Pins or use the breadboard as a middleman as we did in this tutorial. We mainly did this as we didn't have any female to female breadboard wire available to us.

Wiring your RS232 to TTL adapter to your Raspberry Pi is a simple process, with it requiring only 4 of the GPIO connecting to be wired to the serial connector, even better all 4 GPIO pins needed are in a row, so it is easy to follow. Make use of our table and guide below to connect your serial connector to your Raspberry Pi

- **VCC** connects to **Pin 4**.
- **TX** connects to **Pin 8**.
- **RX** connects to **Pin 10**.
- **GND** connects to **Pin 6**.





Setting up the Raspberry Pi for Serial Read and Write

Before we begin writing any code that interacts with the serial in and serial out we must first deal with one issue. That is by default the Raspberry Pi's serial port is configured to be used for console input and output.

While this is fantastic for trying to fix issues that arise during boot or logging into the Pi if both network and video are unavailable it's not good when you need to talk another device.

Over the next few steps, we will show you how to disable this behavior

1. Let's begin by this tutorial by first ensuring our Raspberry Pi is up to date by running the following two commands on the Raspberry Pi.

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

2. Now that our Raspberry Pi is up to date we can make use of the raspi-config tool. This tool will allow us to easily disable the serial input/output interface that is enabled by default

```
sudo raspi-config
```

3. This command will load up the Raspberry Pi configuration screen. This tool allows us to make quite a numerous amount of different changes to the Raspberry Pi's configuration, for now, though we are only after one particular option.

Use the **arrow keys** to go down and select "**5 Interfacing Options**".

Once this option has been selected, you can press **Enter**.

4. With the next screen, you will want to use the **arrow keys** again to select "**P6 Serial**", press **Enter** once highlighted to select this option.

5. You will now be prompted as to whether you want the **login shell to be accessible over serial**, select **No** with your **arrow keys** and press **Enter** to proceed.

6. Immediately after you will be asked if you want to make use of the **Serial Port Hardware**, make sure that you select **Yes** with your **arrow keys** and press **Enter** to proceed.

7. Once the Raspberry Pi has made the changes, you should see the following text appear on your screen.

"**The serial login shell is disabled**
The serial interface is enabled".

Before these changes entirely take effect, we must first restart the Raspberry Pi. To do this first get back to the terminal by pressing **Enter** and then **ESC**.

Type the following command into the terminal on your Raspberry Pi to restart your Raspberry Pi.

```
sudo reboot
```

Setting up the Raspberry Pi for Serial Read and Write

8. Let's now check to make sure that everything has been changed correctly by running the following command on your Raspberry Pi.

```
dmesg | grep tty
```

Here you want to make sure the following message is **not displayed in the output**, if it is **not** there then you can skip onto the next section.

Otherwise, start over from step 2. These messages indicate that Serial Login is still enabled for that interface.

Raspberry Pi 3 & Raspberry Pi Zero W

```
[ttyS0] enabled
```

Raspberry Pi 2 and earlier & Raspberry Pi Zero

```
[ttyAMA0] enabled
```

Utilizing Serial Read and Write on your Raspberry Pi

In this segment of the tutorial, you will need to have your USB-Serial adapter plugged into the RS232 adapter. You then want the USB end of the USB-Serial adapter to be plugged into your Raspberry Pi's USB Ports.

Of course, in a practical application, you would be connecting your serial connection to an actual device such as a modem, a printer or even some RFID readers. However, for this tutorial, we will be just showing you how it all works, and how you can read data that's coming over the serial lines.

1. Once you have connected your USB-Serial adapter up and it is plugged into the Raspberry Pi as well, we can run the following command in terminal.

```
dmesg | grep tty
```

2. In the output from this command, you want to take note of any additional lines that appear that also specify the USB its attached to.

For example, my convertor was attached to **ttyUSB0** like we have shown in our output here. Make a note of what your USB device was attached to as you will need this to complete the tutorial.

```
[ 2429.234287] usb 1-1.2: ch341-uart converter now attached to ttyUSB0
```

3. Now that we know what our USB device is attached to we can proceed with programming our two scripts. One of these scripts will read the data through the **ttyUSB0** port. The other will write data through the **ttyS0/ttyAMA0** port.

To start off we will be writing our serial write script, go to the next section to learn how to do serial writes in Python.

Programming the Raspberry Pi for Serial Writing

1. To start off let's begin writing our **serial_write.py** script, this will write data over the serial port. Run the following two commands on your Raspberry Pi to begin writing the file.

```
mkdir ~/serial
cd ~/serial
nano serial_write.py
```

2. Within this file write the following lines of code:

```
#!/usr/bin/env python
import time
import serial
```

The first line of code is there to tell the operating system what it should try running the file with. Otherwise, it will likely attempt to run it as a normal bash script.

The first import is the **time** library. We use this library to temporarily sleep the script now and then for our test counter. You don't need this package to be able to do serial writes.

The second import is, **serial**. This library contains all the functionality to deal with serial connections, this allows reading and writing through the serial ports.

```
ser = serial.Serial(
    port=' /dev/ttyS0', #Replace ttyS0 with ttyAM0 for Pi1,Pi2,Pi0
    baudrate = 9600,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS,
    timeout=1
)
counter=0
```

This section of code primarily instantiates the serial class, setting it up with all the various bits of information that it needs to make the connection with.

- **port** - This defines the serial port that the object should try and do read and writes over. For Pi 3 and
  Pi Zero W this should be tty0. If you are using a Pi 2 and older, or the base Pi Zero, then you should be using ttyAM0.
- **baudrate** - This is the rate at which information is transferred over a communication channel.
- **parity** - Sets whether we should be doing parity checking, this is for ensuring accurate data transmission between nodes during communication.
- **stopbits** - This is the pattern of bits to expect which indicates the end of a character or the data transmission.
- **bytesize** - This is the number of data bits.
- **timeout** - This is the amount of time that serial commands should wait for before timing out.

```
while 1:
    ser.write('Write counter: %d \n'%(counter))
    time.sleep(1)
    counter += 1
```

This code is rather simple. It loops forever continually writing the text "**Write Counter: 1**" (where 1 is replaced with the current counter number) to the serial port. This code means that any script or device listening on the other side will continually receive that text.

On each loop, we use the time library to sleep the script for 1 second before increasing the counter. We do this to try and not spam the serial port.

Programming the Raspberry Pi for Serial Writing

3. Once you have finished writing our **serial_write.py** script it should look somewhat like what is displayed below

```
#!/usr/bin/env python
import time
import serial

ser = serial.Serial(
    port=' /dev/ttyS0', #Replace ttyS0 with ttyAM0 for Pi1,Pi2,Pi0
    baudrate = 9600,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS,
    timeout=1
)
counter=0

while 1:
    ser.write("Write counter: %d \n"%(counter))
    time.sleep(1)
    counter += 1
```

Once you are sure you have entered the code correctly, then you can save the file by pressing **Ctrl + X** then pressing **Y** and then finally hitting **Enter**.

4. Now that we have completed writing the **serial_write.py** script we can't test it just yet. First, we need to write our **serial_read.py** script. The reason for this is to tell if serial writes are being written through the serial, we need something actually to be receiving them.

In the next section, we will explore writing our **serial_read.py** script. Don't worry as it is like our serial write code.

Programming the Raspberry Pi for Serial Reading

1. To start off let's begin writing our **serial_read.py** script, this will write data over the serial port. Run the following two commands on your Raspberry Pi to start writing the file.

```
mkdir ~/serial
cd ~/serial
nano serial_read.py
```

2. Within this file write the following lines of code:

```
#!/usr/bin/env python
import time
import serial

ser = serial.Serial(
    port=' /dev/ttyUSB0',
    baudrate = 9600,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS,
    timeout=1
)
```

Since we have already gone through large amounts of this codem we won't bother going over it again. The only difference here is that for the port we are using our USB device. In our case this was **ttyUSB0**, remember to change this if you got a different result earlier on in this tutorial.

```
while 1:
    x=ser.readline()
    print x
```

This piece of code is straightforward. It utilizes a function from a serial object that we set up earlier in the code. This function reads a terminated line, meaning it reads until it hits a line that ends in "**\n**". Anything after will be rejected. Once it reads the value, it stores it into our **x** variable.

Finally, we print the value that we obtain user the **ser.readline()** function.

3. Once you have finished writing our **serial_read.py** script it should look somewhat like what is displayed below:

```
#!/usr/bin/env python
import time
import serial

ser = serial.Serial(
    port=' /dev/ttyUSB0',
    baudrate = 9600,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS,
    timeout=1
)

while 1:
    x=ser.readline()
    print x
```

Once you are sure you have entered the code correctly, then you can save the file by pressing **Ctrl + X** then pressing **Y** and then finally hitting **Enter**.

Testing our read and write python scripts

1. Now that we have written our serial read and serial write Python scripts we can now finally go onto testing them.

To do this, you will need to have two active terminal windows, meaning either opening the Terminal application open twice on your Raspberry Pi or just starting two separate SSH connections.

The reason for doing this is that you can see the serial write script and the serial read script work at the same time.

2. Now in one terminal window, you will want to type in the following two commands to startup our **serial_read.py** python script. This will immediately start reading in all data that is passed through its serial connection by our **serial_write.py** script.

```
cd ~/serial
sudo python serial_read.py
```

3. Now in our other terminal window, type in the following two commands to start up the **serial_write.py** python script. This command will start outputting data through the serial connection which we will soon receive using our other script.

```
cd ~/serial
sudo python serial_write.py
```

4. You should now notice that in your first terminal window, that the serial data that we are writing with our **serial_write.py** script is being displayed.

This means that we have successfully written our two scripts and that **serial_read.py** is successfully receiving the data from **serial_write.py**.

We hope that upon completing this tutorial that you now have an idea on how writing and reading data through the serial bus works. This should give you an idea on how you could potentially interact with other devices.

Adding a Real Time Clock to the Pi

Project Description

In this tutorial, we will be showing you how to add either the PCF8523, DSL1307 or DS3231 real time clock (RTC) modules to your Raspberry Pi.

We will be showing you how each of the individual real-time clock chips needs to be wired up to the Raspberry Pi to function correctly by providing the pin numbers and a helpful GPIO guide.

You will also be learning in this tutorial the changes you need to make to the Raspberry Pi's configuration as well as modify packages on Raspbian, so it will read the time from your real-time clock module and not utilize the fake time it relies on by default.

We will also be showing you how you set the time on your real-time clock module when required to do so.

Equipment

Required

- Raspberry Pi
- SD Card (8 GB+ Recommended)
- PCF8523, DSL1307 or DS3231 RTC Modules

Optional

- Raspberry Pi Case
- GPIO Breakout Kit
- Network Connection
- USB Keyboard
- USB Mouse

Wiring your RTC module to the Raspberry Pi

On your **RTC Module**, you should find at least four connections. Some RTC circuits may come with more, but we only need the following four for the Raspberry Pi:

- **VCC/5V/Vin** (**IC Power-supply pin**)
- **SDA** (**Serial Data Line**)
- **SCL** (**Serial Clock Line**)
- **GND** (**Ground power-supply pin**)

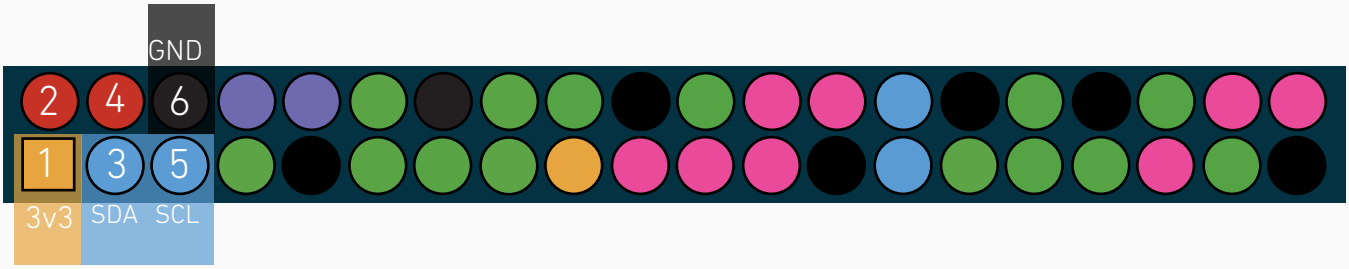
You can either connect these lines directly to your Raspberry Pi or connect it to a breadboard and then to the Raspberry Pi.

For this tutorial, we utilized the **Pi RTC PCF8523** from Adafruit which plugs in directly over the first 6 pins which significantly simplifies the process of setting up an RTC (Real Time Clock) module.

However, wiring up a normal **PCF8523**, **DSL1307** and a **DS3231** isn't a complicated process, following our guide below you should have everything connected to in no time.

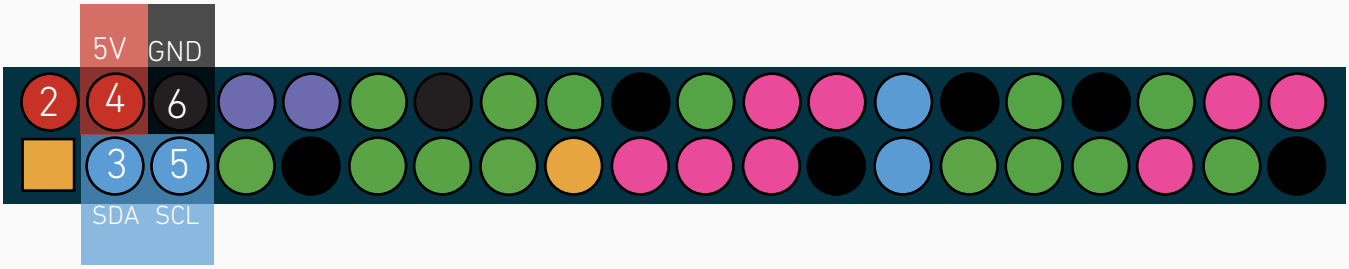
DS3231 & PCF8523

- **Vin** connects to **Pin 1**.
- **SDA** connects to **Pin 3**.
- **SCL** connects to **Pin 5**.
- **GND** connects to **Pin 6**.



DS1307

- **Vin** connects to **Pin 4**.
- **SDA** connects to **Pin 3**.
- **SCL** connects to **Pin 5**.
- **GND** connects to **Pin 6**.



Configuring the Raspberry Pi for I2C

Before we begin setting up and utilizing our RTC on the Raspberry Pi, we first have to make use of the **raspi-config** tool to configure our Raspberry Pi for use with I2C.

1. Let's begin this tutorial by ensuring our Raspberry Pi is entirely up to date; this ensures that we will be utilizing all the latest software available.

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

2. With the Raspberry Pi now entirely up to date we can now run its configuration tool to begin the process of switching on I2C. Run the following command to launch the configuration tool.

```
sudo raspi-config
```

3. This command will bring up the configuration tool; this tool is an easy way to make a variety of changes to your Raspberry Pi's configuration. Today, however, we will only by exploring how to enable the I2C interface.

Use the **arrow keys** to go down and select "**5 Interfacing Options**". Once this option has been selected, you can press **Enter**.

4. On the next screen, you will want to use the **arrow keys** to select "**P5 I2C**", press **Enter** once highlighted to choose this option.

5. You will now be asked if you want to enable the "**ARM I2C Interface**", select **Yes** with your **arrow keys** and press **Enter** to proceed.

6. Once the raspi-config tool makes the needed changes, the following text should appear on the screen.
"The ARM I2C interface is enabled".

However, before I2C is genuinely enabled, we must first restart the Raspberry Pi. To do this first get back to the terminal by pressing **Enter** and then **ESC**.

Type the following command into the terminal on your Raspberry Pi to restart it.

```
sudo reboot
```

7. Once the Raspberry Pi has finished restarting we need to install an additional two packages, these packages will help us tell whether we have setup I2C successfully and that it is working as intended.

Run the following command on your Raspberry Pi to install **python-smbus** and **i2c-tools**:

```
sudo apt-get install python-smbus i2c-tools
```

8. With those tools now installed run the following command on your Raspberry Pi to detect that you have correctly wired up your RTC device.

```
sudo i2cdetect -y 1
```

If you have successfully wired up your RTC circuit, you should see the **ID #68** appear. This id is the address of the **DS1307**, **DS3231** and the **PCF85231** RTC Chips.

Once we have the Kernel driver up and running the tool will start to display **UU** instead, this is an indication that it is working as intended.

Setting up the Raspberry Pi RTC Time

With I2C successfully setup we can now begin the process of configuring the Raspberry Pi to use our RTC Chip for its time.

1. To do this, we will first have to modify the Raspberry Pi's boot configuration file so that the correct Kernel driver for our RTC circuit will be successfully loaded in.

Run the following command on your Raspberry PI to begin editing the **/boot/config.txt** file.

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

2. Within this file, you will want to add one of the following lines to the bottom, make sure you use the correct one for the RTC Chip you are using. In our case, we are using a **PCF8523**.

DS1307

```
dtoverlay=i2c-rtc,ds1307
```

PCF8523

```
dtoverlay=i2c-rtc,pcf8523
```

DS3231

```
dtoverlay=i2c-rtc,ds3231
```

Once you have added the correct line for your device to the bottom of the file you can save and quit out of it by pressing **Ctrl + X**, then **Y** and then **Enter**.

3. With that change made we need to restart the Raspberry Pi, so it loads in the latest configuration changes.

Run the following command on your Raspberry Pi to restart it.

```
sudo reboot
```

4. Once your Raspberry Pi has finished restarting we can now run the following command, this is so we can make sure that the kernel drivers for the RTC Chip are loaded in.

```
sudo i2cdetect -y 1
```

You should see a wall of text appear, if **UU** appears instead of **68** then we have successfully loaded in the Kernel driver for our RTC circuit.

5. Now that we have successfully got the kernel driver activated for the RTC Chip and know it's communicating with the Raspberry Pi, we need to know remove the "**fake hwclock**" package.

This package acts as a placeholder for the real hardware clock when you don't have one.

Type the following two commands into the terminal on your Raspberry Pi to remove the **fake-hwclock** package.

We also remove hwclock from any startup scripts as we will no longer need this.

```
sudo apt-get -y remove fake-hwclock
sudo update-rc.d -f fake-hwclock remove
```

Setting up the Raspberry Pi RTC Time

6. Now that we have disabled the **fake-hwclock** package we can proceed with getting the original hardware clock script that is included in Raspbian up and running again by commenting out a section of code.

Run the following command to begin editing the original RTC script.

```
sudo nano /lib/udev/hwclock-set
```

7. Find and comment out the following three lines by placing **#** in front of it as we have done below.

Find

```
if [ -e /run/systemd/system ] ; then
    exit 0
fi
```

Replace With

```
#if [ -e /run/systemd/system ] ; then
#    exit 0
#fi
```

Once you have made the change, save the file by pressing **Ctrl + X** then **Y** then **Enter**.

Syncing time from the Pi to the RTC module

Now that we have our RTC module all hooked up and everything configured correctly we need to synchronize the time with our RTC Module. The reason for this is that the time provided by a fresh RTC module will be incorrect.

1. You can read the time directly from the RTC module by running the following command if you try it now you will notice it is currently way off our current real time.

```
sudo hwclock -D -r
```

2. Now before we go ahead and sync the correct time from our Raspberry Pi to our RTC module, we need to run the following command to make sure the time on the Raspberry Pi is in fact correct.

If the time is not right, make sure that you are connected a Wi-Fi or Ethernet connection.

```
date
```

3. If the time displayed by the date command is correct, we can go ahead and run the following command on your Raspberry Pi. This command will write the time from the Raspberry Pi to the RTC Module.

```
sudo hwclock -w
```

4. Now if you read the time directly from the RTC module again you will notice that it been corrected to the same time as what your Raspberry Pi was set at. You should never have to run the previous command again if you keep a battery in your RTC module.

```
sudo hwclock -r
```

You should hopefully now have a fully operational RTC module that is actively keeping your Raspberry Pi's time correct even when it loses power or loses an internet connection.

Basic Guides To Electronics

Basic Guide to Voltage Dividers

Introduction

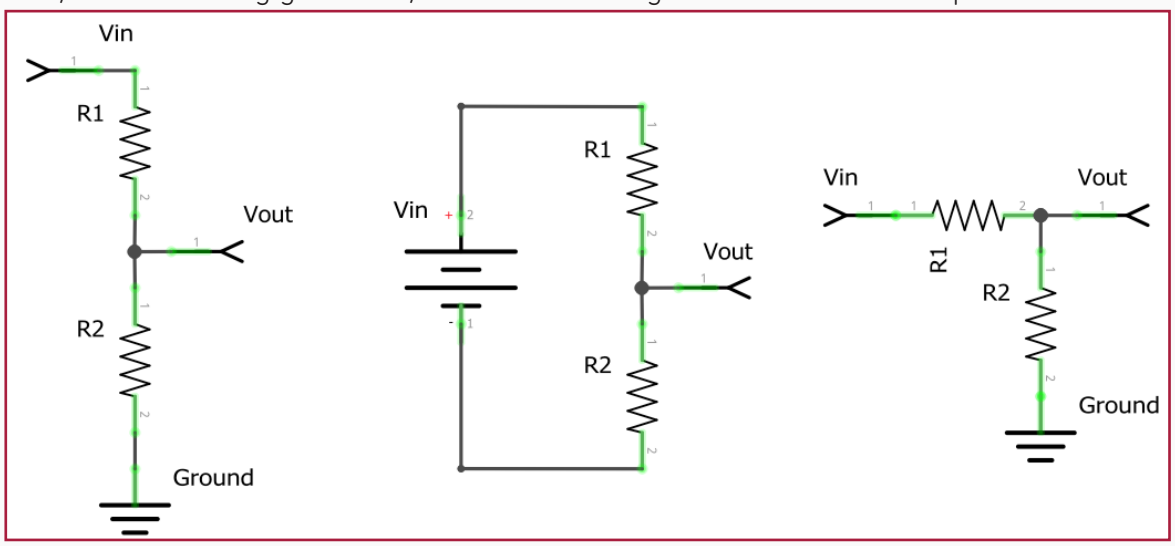
In this guide, we will be exploring a very crucial and fundamental element in electronic circuits, and that is voltage dividers. A voltage divider is a somewhat basic passive circuit that plays a very crucial role, in basic terms, a voltage divider transforms a large voltage down into a smaller one.

A basic voltage divider circuit consists of two resistors wired in series that produces an output voltage that is just a fraction of its input voltage.

The input voltage is applied across the two resistors with your desired output voltage coming from the connection between the two resistors. The second resistor is typically connected to ground.

Basic Voltage Divider Circuits

Below are some examples of how you could see a voltage divider circuit setup or drawn. Of course, there are numerous other ways to set up the circuit but, they should all be the same circuit with the two resistors, with one being grounded, and a line coming from in between the pair of resistors.



As you can see in a basic setup of the voltage divider circuit the resistor that is closet to the input voltage (V_{in}) will usually be referred to as R_1 with the resistor that is closest to the ground connection being referred to as R_2.

The voltage drop that is caused by the input voltage going through the resistor pair (R_1 and R_2) is referred to as V_{out}. The resulting voltage is what we call our divided voltage, the voltage through this is a fraction of the original input voltage (V_{in}).

We use R_1, R_2, V_{in}, and V_{out} to name elements of the circuit as they are crucial to understanding the values you will need for the voltage divider equation.

The Voltage Divider Equation

The voltage divider equation assumes that you know three of the values utilized in the circuits above. The values you will need to know to utilize the equation are the following. Both the resistor values (R_1 and R_2) and also the input voltage (V_{in}). Utilizing these 3 values in the equation below will allow us to calculate the output voltage of a voltage divider circuit.

$$V_{out} = V_{in} \times \frac{R_2}{R_1 + R_2}$$

Over the page we will walk you through some examples of using the voltage divider equation to give you a good idea on how to utilize it.

Basic Guide to Voltage Dividers

The Voltage Divider Equation Examples

For our first example of utilizing the Voltage Divider equation we are going to utilize the following values:

- Vin as 5v,
- R1 as a 220Ω resistor,
- and R2 as a 680Ω resistor.

Now if we plug these values into our voltage diver equation, we should end up with something like what we have displayed below.

$$V_{out} = 5 \times \frac{680}{220 + 680}$$

To begin with, we will first plus the R1 and the R2 resistor values together, so in our example, this will end up being **220 + 680**, which is equal to **900**. We will replace the value in our equation, so you end up with the following.

$$V_{out} = 5 \times \frac{680}{900}$$

Now that we have done the simple addition we can finally calculate the division part of the voltage divider equation. Just divide your **R2** value with your calculated **R1 + R2** value, in our example, this will be **680** divided by **900**.

Using a calculator this will give us **0.7555555555555556**, but we round this to 2 decimal places, meaning the number will become **0.76**.

Replace the division part of your equation with the value you have gotten, just like that we have done below.

$$V_{out} = 5 \times 0.76$$

Finally, we can just multiply the **Vin** with our calculated resistor division amount. In our case simply multiply **5** by **0.76**. The result of this multiplication will give you your **Vout** amount.

$$V_{out} = 3.8$$

The Voltage Divider Equation Simplifications

There are a few simplifications we can make about the voltage divider equation however for this guide we will just walk you through the one below. These can make it easier to evaluate a voltage divider circuit quickly.

$$\text{if } R1 = R2 \text{ then } V_{out} = \frac{V_{in}}{2}$$

This simplification says that if the value of the R1 resistor and the R2 resistor are the same, then the voltage out is equal to half the voltage in.

Basic Guide to Voltage Dividers

Voltage Divider Applications

Voltage dividers have many applications in electronic circuitry and are a core component of very many electronic circuits. Below we will show you some of the few applications of the voltage divider circuit.

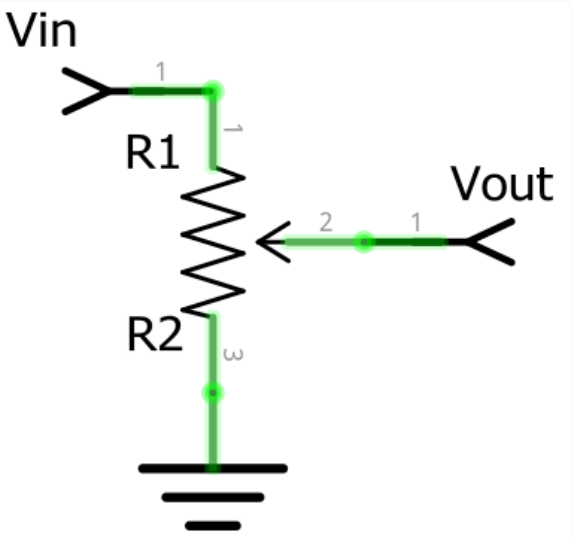
Potentiometers

A potentiometer is one of the most well-used pieces of electronic and is featured as a core component in a large array of different electronics.

Some examples of the sort of devices that a potentiometer is used in are the following: measure the position on a joystick, create a reference voltage, control the audio level in speakers, among many other things.

A potentiometer is a variable resistor that acts just like an adjustable voltage divider. Inside the pot is a single resistor that is separated by a wiper, this wiper is what you move that adjusts the ratio between the two halves of the resistor.

Outside of the pot, you will find three pins, the pins on either side are the connection between each end of the resistor, you could consider these as being R_1 and R_2. The pin in the middle is what is connected to the wiper. This theoretically is like the V_{out} in a voltage divider circuit.



To wire a potentiometer so it acts like an adjustable voltage divider you will need to connect one side to your input voltage (V_{in}) and the other side of your ground. With both of the two outside pins wired correctly, the middle pin will then act as your voltage dividers output (V_{out}).

Turning the pot in one direction will make the voltage go towards zero, setting it to the other side will make the voltage approach the input voltage. Turning the pot to the middle position will effectively mean the output voltage will be half the input voltage.

Basic Guide to Voltage Dividers

Level Shifting

Some of the more complicated sensors that utilize interfaces such as UART, SPI or I2C to transmit their readings will only use a light, many of these also run using a low voltage to conserve power.

Luckily for us, the Raspberry Pi provides a few 3.3v power pins so we can power these circuits without needing to step down the voltage like you have to do with some Arduinos.

While the Raspberry Pi can easily handle both 5v and 3.3v voltage outputs, it's GPIO pins aren't designed to handle 5v input through its GPIO Pins.

This becomes an issue as the Raspberry Pi only expects 3.3v in through its GPIO pins meaning we need to step down the amount of voltage. Luckily we can achieve this by utilizing a voltage divider.

Utilizing a voltage divider within a circuit will allow us to step down the voltage from 5v down to 3.3v for the input pin. Having a higher voltage go through the Raspberry Pi can damage the device

For example, in our distance sensor tutorial, we utilize the HC-SR04 ultrasonic sensor. This sensor utilizes a 5v power input, meaning we need to step down the output on the Echo pin from 5v to 3.3v before it reaches the GPIO Pins.

We can calculate the resistors we need by choosing an initial resistor value. Resistors between **1kΩ-10kΩ** work best for dropping the voltage from **5v** to **3.3v**. For our example, we will utilize a **1kΩ** resistor. To work out the second resistor that we need to use we will use yet another re-arranged version of the voltage divider equation.

To calculate the **R_2** value that we need to use we need to know the **Vin**, the **Vout** and the **first resistor** we plan on using. With those 3 values handy we can utilize the following equation.

$$R2 = \frac{Vout * R1}{Vin - Vout}$$

Filling that equation in to calculate what resistor we need to drop the voltage from **5v** to **3.3v** with a **1kΩ** resistor you should end up with the following.

$$R2 = \frac{3.3 * 1000}{5 - 3.3}$$

First, you should calculate both halves of the division, if you **multiply** the **Vout** (**3.3**) by the **R1** (**1000**) value you should get 3300. Now we also need to do the bottom half, subtracting the Vout from the Vin, in this example that is 5 - 3.3 which is equal to 1.7.

$$R2 = \frac{3300}{1.7}$$

Finally, divide both values to get your resistance value, in our example, this is **3300 divided** by **1.7**. Putting this into a calculator we get a big long number, but we will simplify this to the closest 2 decimals.

$$R2 = 1941.18$$

With this value, we can deduce that a **2kΩ** resistor should be more than enough to lower the **5v** voltage down to **3.3v**.

Reading Resistive Sensors

One thing you may notice is that many sensors in the real world are just simple resistive devices designed to react to certain elements.

For example, an LDR (Light Dependant Resistor) Sensor like that we use in earlier in our light sensor tutorial works by producing a resistance that is proportional to the amount of like that is touching it.

There are also many other sensors that are effectively just fancy resistors such as thermistors, flex sensors, and force-sensitive resistors.

Sadly though unlike voltage (Coupled with an analog-to-digital converter), resistance is not as easy to measure for microcontrollers such as the Raspberry Pi.

However we can make it easier by rejigging the circuit to be a voltage divider, this is simple as adding a resistor so the circuit is much more like the circuits we showed earlier in this guide. That way we can utilize the voltage provided to us from the voltage divider to calculate the current light level.

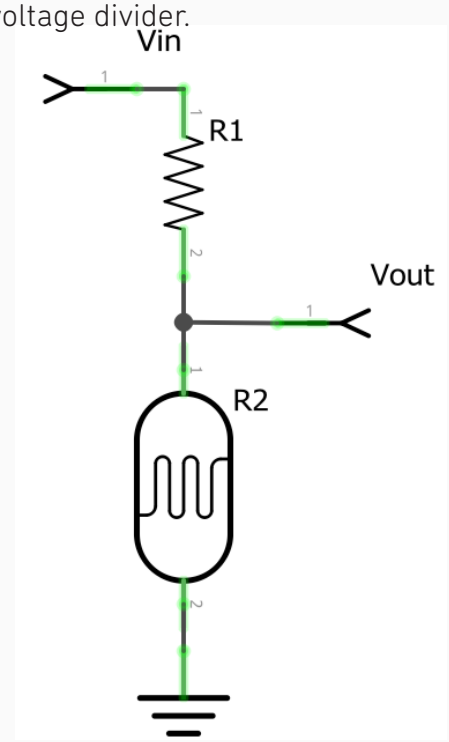
Adding a resistor of a value you know, such as a 1k ohm resistor you can then work out the resistance of the LDR at different light levels by re-arranging the equation used earlier. All we need to know, is the **Vin**, the **Vout**, and the **R1** resistor value.

$$R2 = \frac{Vout * R1}{Vin - Vout}$$

Using the above equation you can quickly calculate the resistance of the LDR at its darkest level of light and its brightest.

This would give us an idea of its highest and lowest resistance. When you have both of these resistance values, you can work out a resistor value that sits in between, this will give you the largest resolution for calculating the current light through an analog-to-digital converter.

For example, a photocell's resistance can vary between 1kΩ in the light and approximately 10kΩ in the dark. So by utilizing a resistor that has a value somewhere in the middle, such as a 5.1kΩ resistor, we can get the widest range out of our voltage divider.



References

Linux Cheat Sheet

File System

ls	Directory Listing
tree	Indented file/directory listing
cd	Change directory
pwd	Display current directory
mkdir newDir	Creates a directory
rmdir oldDir	Removes a directory
cp file newfilename	Copies a file to a new location
mv file.txt ./newDir	Moves a file to a new location
touch file	Sets the last modified timestamp
cat	List contents of a file
head file	Outputs first 10 lines of a file
tail file	Outputs last 10 lines of a file
chmod 777 file	CHMOD is used to alter the permissions of a file or files.
chmod u=rw file	You can use symbols or numbers depending on what you prefer
chown pi:root file	CHOWN will change the user and/or the group that owns a file.
unzip archive.zip	Unzip will extract the files and directories from a compressed zip archive.
tar -cvzf **tar** -xvzf	Compress and extract the contents of an archive in the tar format. -c to compress & -x to extract

Users Management

id	Get the id of a user or group
who	List users that are logged in
last	List of users recently logged in
groupadd name	Adds a new group
useradd name	Adds a new user
userdel name	Deletes a user & all files related to that user
deluser name	Deletes user with options to remove certain data
usermod	Modify a user account
passwd	Change your password or a password of another account

Process Management

ps	Show snapshot of current processes
top	Show real time processes
kill pid	Kill process with id pid
pkill name	Kill process with name
killall name	Kill processes with name

Networking

ping host	Pings a host
hostname	Hostname of the system
ifconfig	Shows network configuration details for the interfaces on current system
ssh	Connect to another computer using SSH
scp	Transfer files over SSH
wget {URL}	Downloads a file directly to the device
netstat -tupl	Active connections to/from device

Shortcuts

!!	Repeats last command
ctrl+z	Stops command, resume using **fg** for foreground or **bg** for background
ctrl+c	Halts the current command
ctrl+w	Deletes 1 word on the current line
ctrl+u	Deletes entire line
ctrl+r	Search previous commands
exit	log out of current session

General Commands

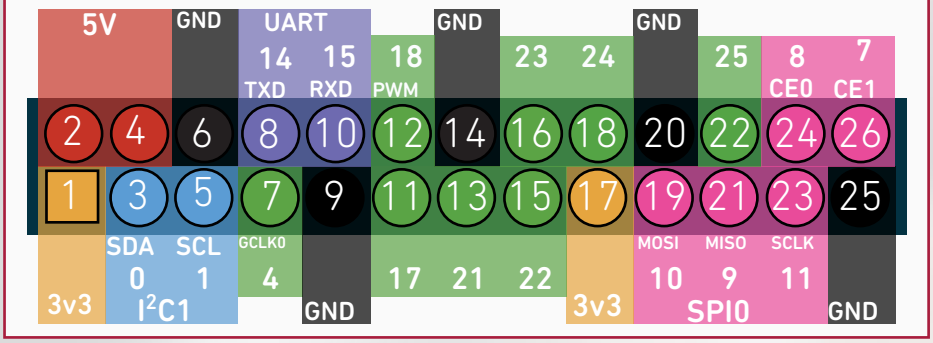
Man ls	Brings up the manual page for ls
ls **|** head	Pipes	allows you to direct output from one command into another
date	Shows system date
whoami	Shows your username
uptime	Shows uptime
uname -a	Show system & kernel

Searching

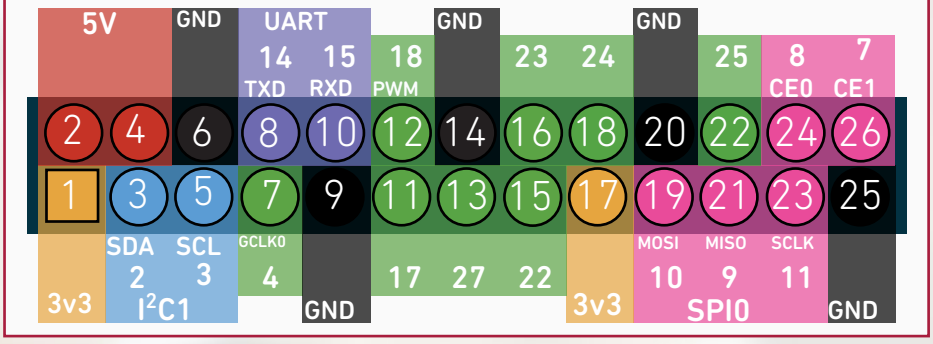
grep "search" *.txt	Search for a pattern inside files
find . -name 'help'	Will search for directories & files that match a pattern
whereis ls	Displays documentation, binaries & source files of a command

Raspberry Pi GPIO Pins

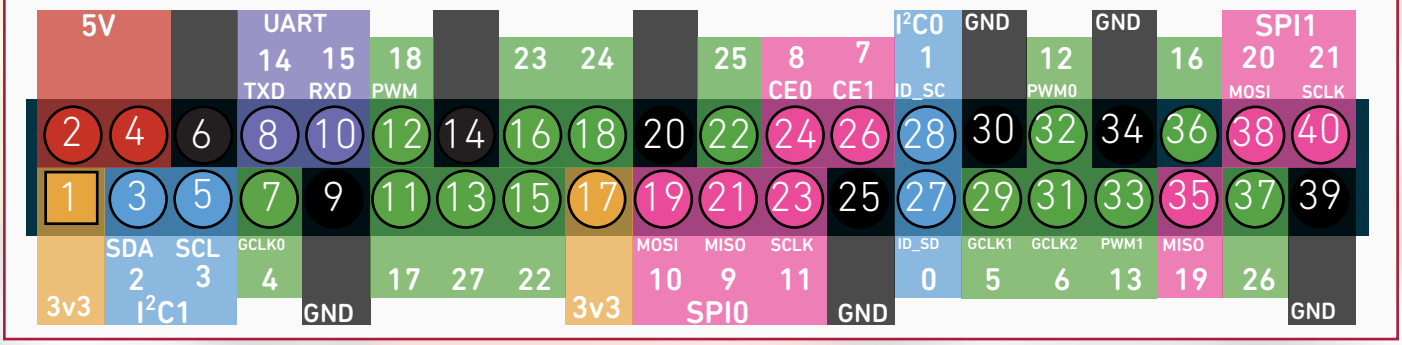
Raspberry Pi (Revision 1)



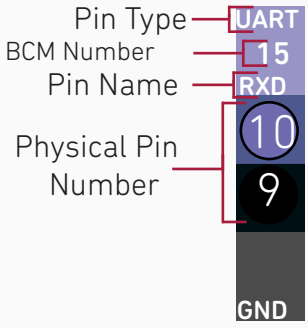
Raspberry Pi (Revision 2)



Raspberry Pi B+, 2, 3 & Zero

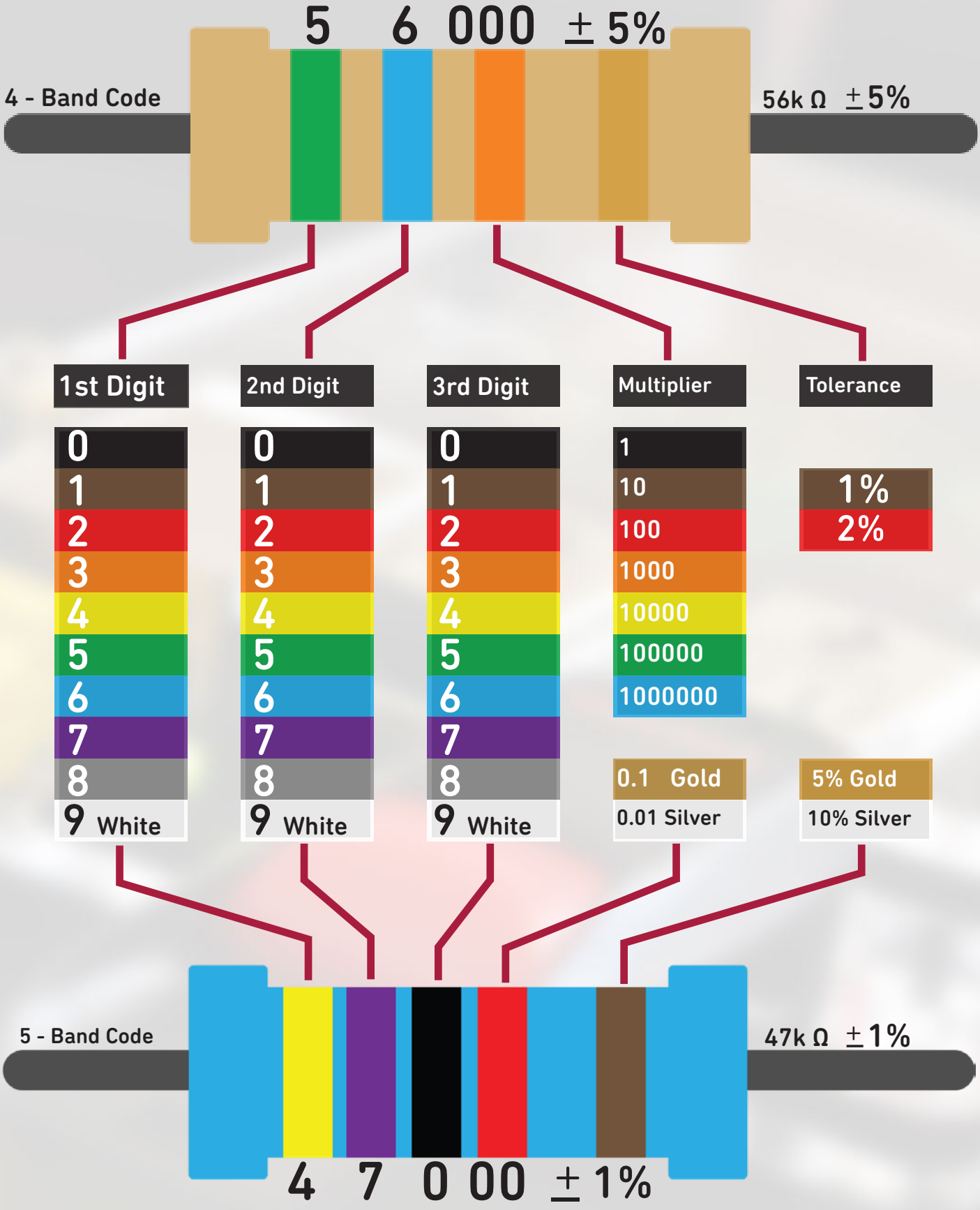


Key



- GPIO Pins
- Ground
- Serial Peripheral Interface (SPI)
- Universal asynchronous receiver/transmitter (UART)
- Inter-Intergrated Circuit (I^2C)
- 5v Power Pins
- 3v3 Power Pins

Resistor Colour Guide



Voltage Divider Equation

Calculating the Voltage Out

$$V_{out} = V_{in} \times \frac{R_2}{R_1 + R_2}$$

Calculating the R2 Value

$$R_2 = \frac{V_{out} * R_1}{V_{in} - V_{out}}$$