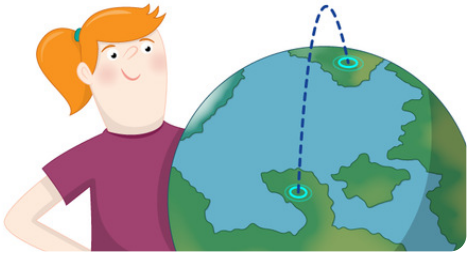
Fetching the weather

Access the Weather Station database and fetch the weather



Step 1 What you will make

This resource teaches you how to access the Raspberry Pi Weather Station database using a RESTful API, how to use the haversine formula to calculate which weather station is closest to you, and how to fetch the latest weather data from that station.

What you will learn

By creating a script to fetch data from a weather database, you will learn:

- How to access a RESTful API in Python
- How to convert JSON data into dictionaries
- How to pretty-print data
- How to calculate distances between two points on the Earth's surface

This resource covers elements from the following strands of the Raspberry Pi Digital Making Curriculum (https://www.raspberrypi.org/curriculum/):

- Apply abstraction and decomposition to solve more complex problems (https://www.raspberrypi.org/curriculum/programming/developer)

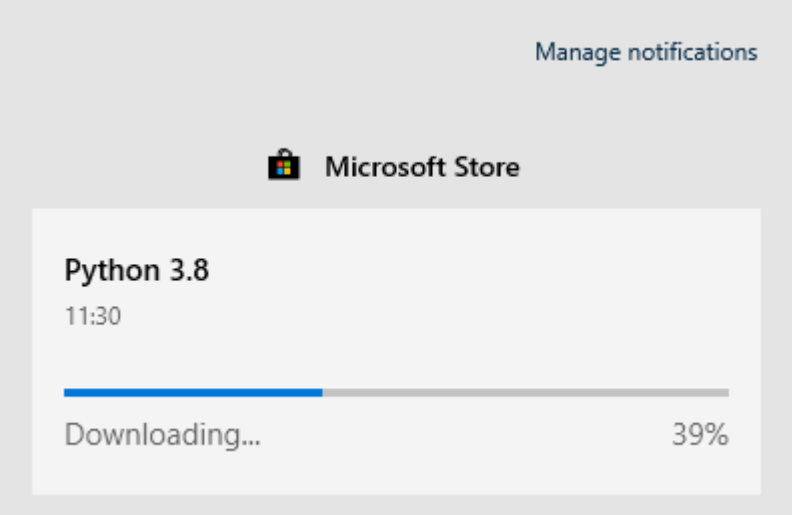
Step 2 What you will need

- A computer (PC/Mac/Linux/Raspberry Pi)
- An internet connection
  Software
- Python 3
- requests (http://docs.python-requests.org) Python module

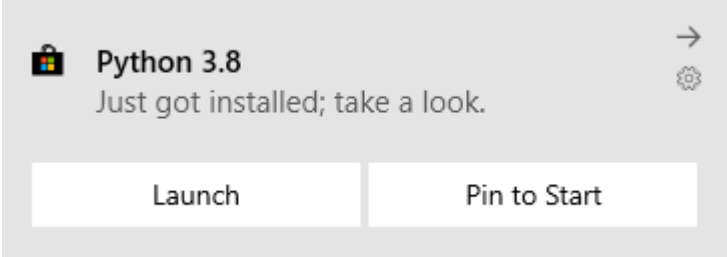
How to install Python 3

If Python 3 or IDLE isn't installed on your computer, follow the installation instructions below for your operating system:
Microsoft Windows
macOS
Raspberry Pi OS and Linux
Microsoft Windows
It is recommended that you install Python via the Microsoft Store. If this is not possible, you can also use a Python installer from www.python.org (https://www.python.org).
Microsoft Store (recommended)
Open the Python 3.8 application in the Microsoft Store (ms-windows-store://pdp/?ProductId=9MSSZTT1N39L).
Click the Get button to download and install Python 3.8.

Python 3.8 will be downloaded and installed. Progress will be shown in the notification bar.



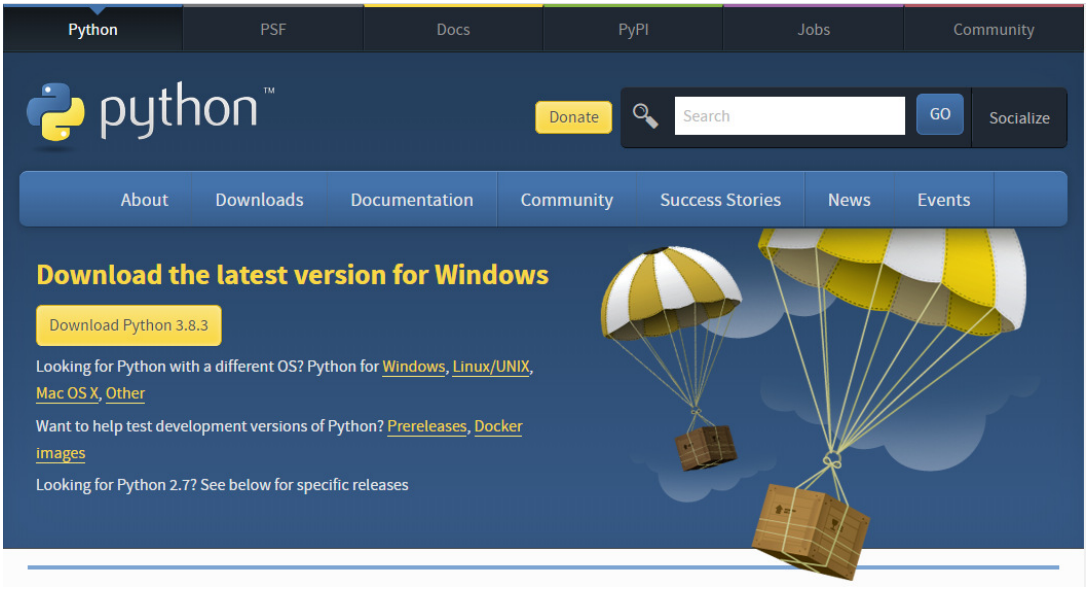
When the installation process is complete, a notification will appear.



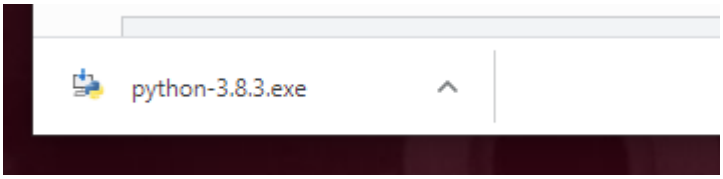
Python installer

Open your web browser and navigate to www.python.org/downloads (https://www.python.org/downloads).

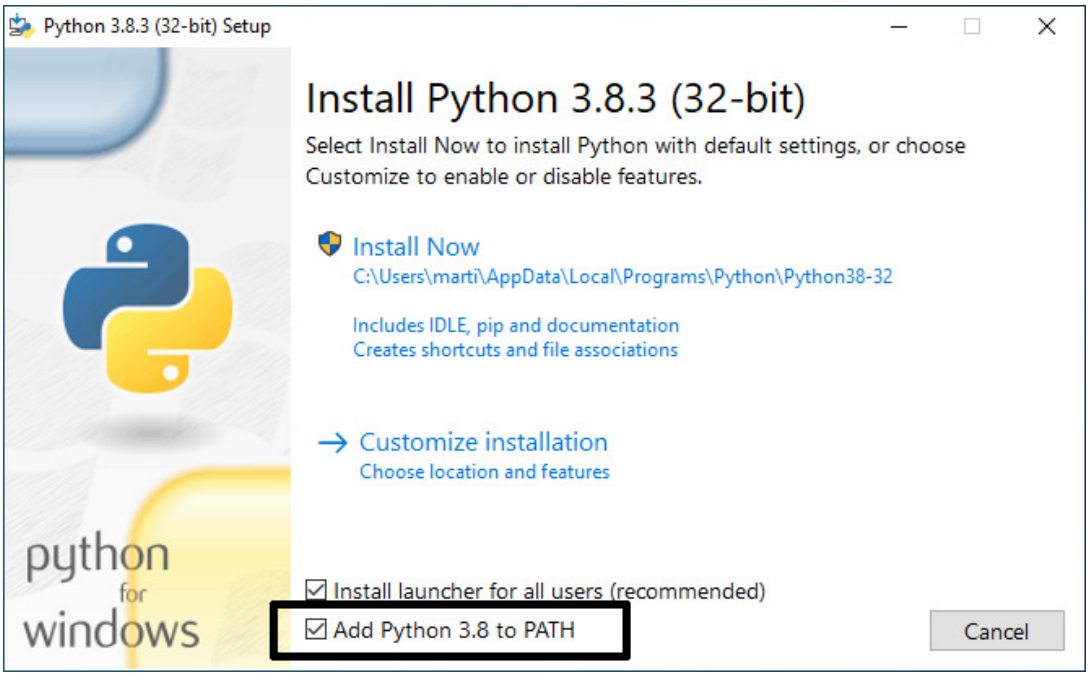
On this web page, you will see a button to install the latest version of Python 3. Click the button, and a download will start automatically.



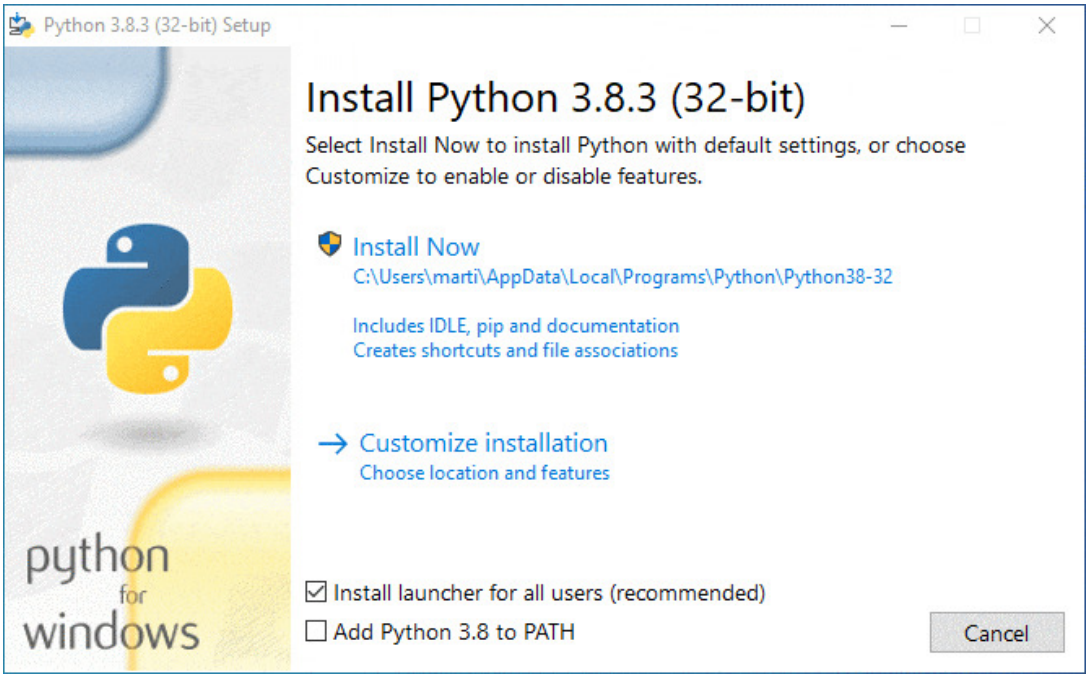
Click on the `.exe` file to run it. (It will have been saved in your `Downloads` folder, or wherever your computer saves downloaded files by default.)

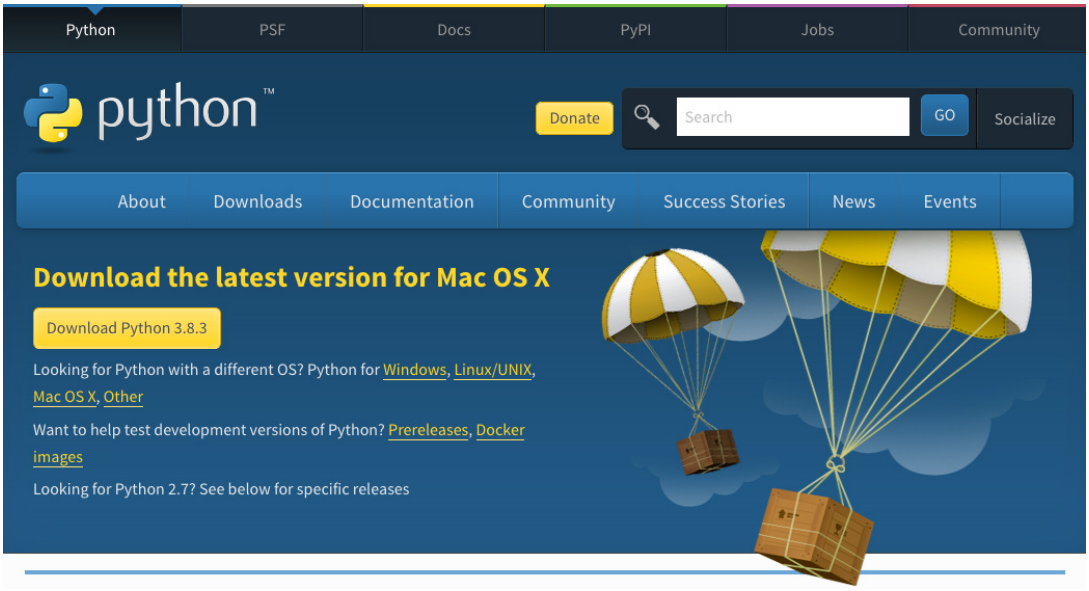


In the dialogue box that opens, it is important that you first tick the box next to Add Python 3 to PATH.



Click on Install Now and follow the installation guide The setup process will take a little time




Python
PSF
Docs
PyPI
Jobs
Community
python™
Donate
Search
GO
Socialize
About
Downloads
Documentation
Community
Success Stories
News
Events
Download the latest version for Mac OS X
Download Python 3.8.3
Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other
Want to help test development versions of Python? Prereleases, Docker images
Looking for Python 2.7? See below for specific releases


python-3.8.3-....pkg

`apt`

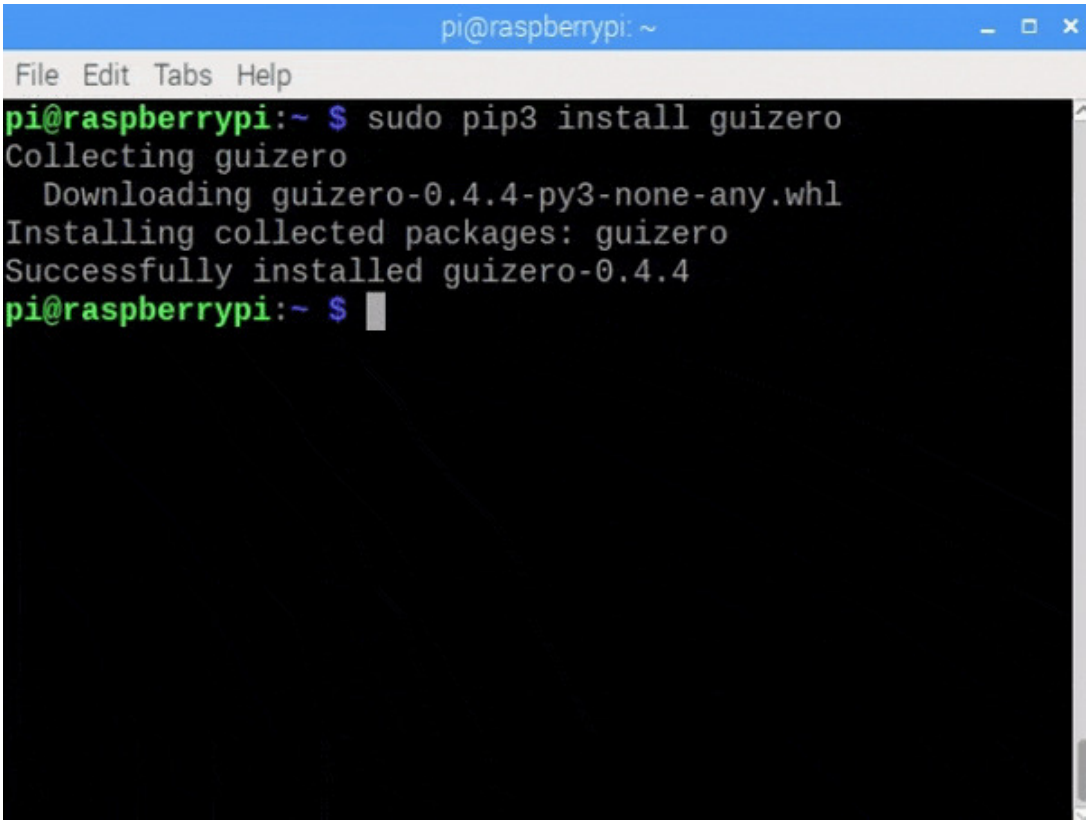
```
sudo apt update
sudo apt install python3 idle3
```

You can use `pip` to install the `requests` module.

Installing Python modules with pip

Installing Python modules with pip
`pip` or `pip3` is a command line tool for installing Python 3 modules.
Modules can be downloaded as packages from the Python Package Index (https://pypi.python.org/pypi) and installed on your computer automatically.
To install a module, use the `pip3 install name_of_module` command, replacing `name_of_module` with the module you wish to install.
Follow the instructions below for your operating system.
Raspberry Pi
Open a terminal window by clicking Menu > Accessories > Terminal.
Enter this command to install a module:

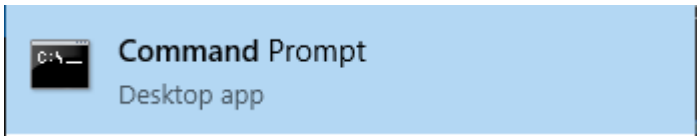
```
sudo pip3 install name_of_module
```



If you experience problems, have a look at our guide *Using pip on Raspberry Pi* (https://projects.raspberrypi.org/en/projects/using-pip-on-raspberry-pi).

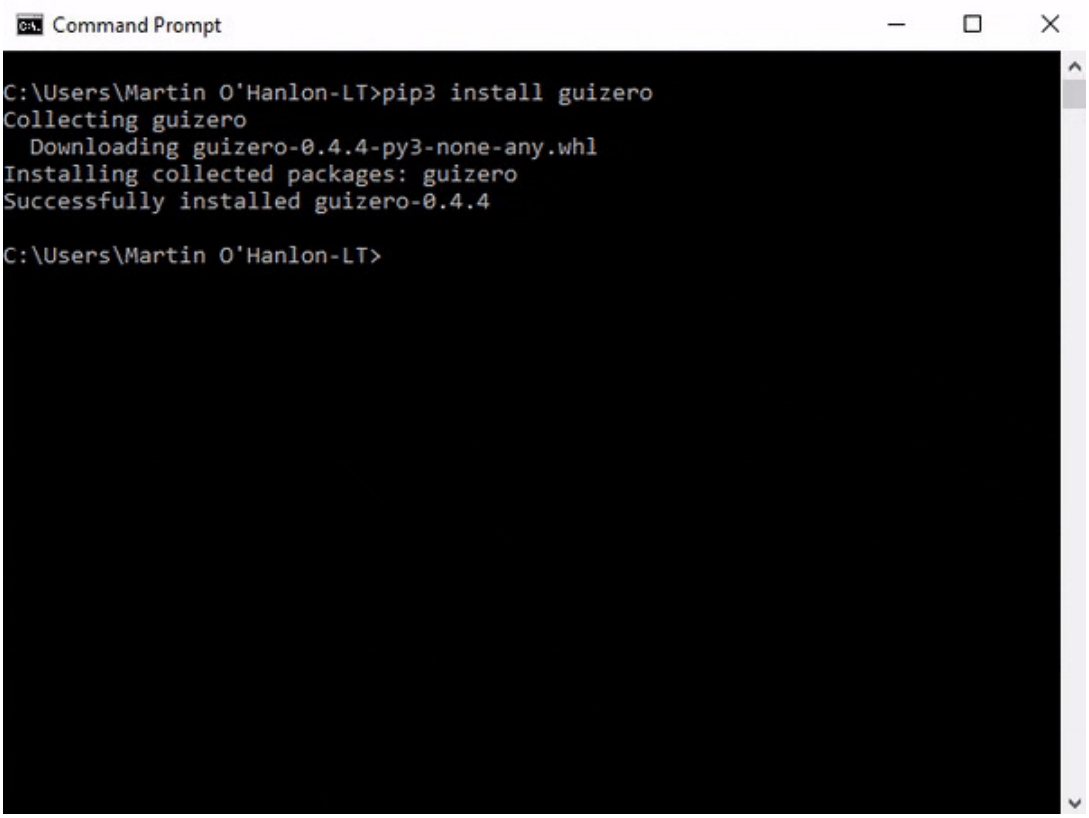
Windows

Open a command prompt by clicking Start > Windows System > Command Prompt, or by typing ‘command’ into the start menu’s search bar.



Enter this command to install a module:

```
pip3 install name_of_module
```



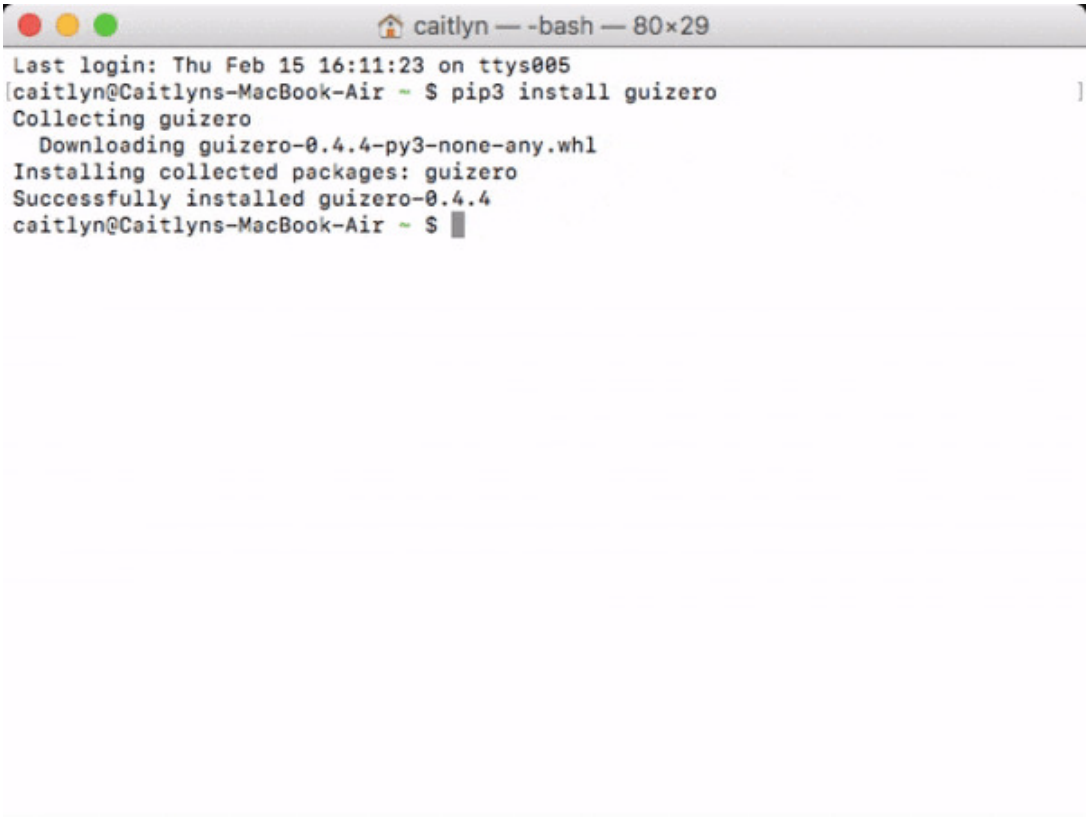
If you experience problems, have a look at our guide *Using pip on Windows* (https://projects.raspberrypi.org/en/projects/using-pip-on-windows).

macOS

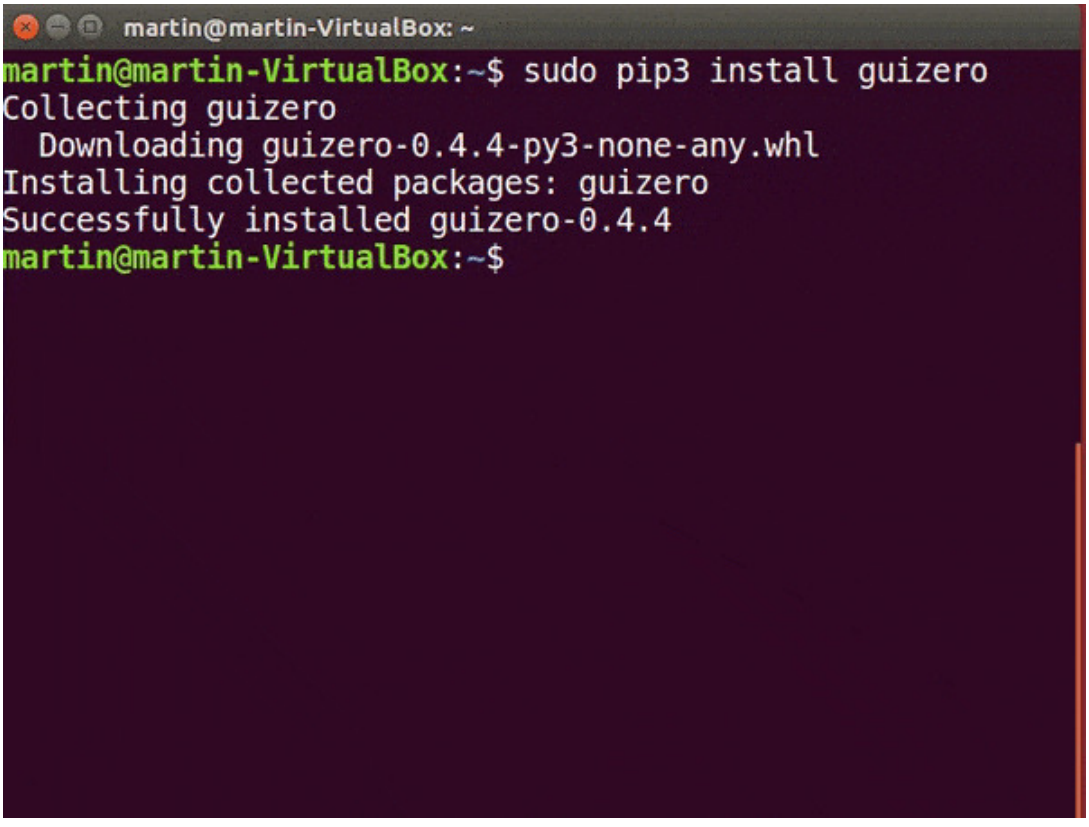
Open a terminal window by clicking Applications > Utilities > Terminal, or by typing ‘terminal’ into the desktop’s search bar.

Enter this command to install a module:

```
pip3 install name_of_module
```




```
sudo pip3 install name_of_module
```



```
Could not find a version that satisfies the requirement <package-name (from
versions: )>
```

```
No matching distribution found for <package-name>
```

`pillow` `PIL`

`pip3`

```
 pip3 list
```

```
pip3 install --upgrade name_of_module
```

```
pip3 uninstall name_of_module
```

One thousand Weather Stations were sent out to schools all over the world at the beginning of 2016, ready to be assembled and begin collecting global weather data.

Each Weather Station comes equipped with the sensors shown in the table below:

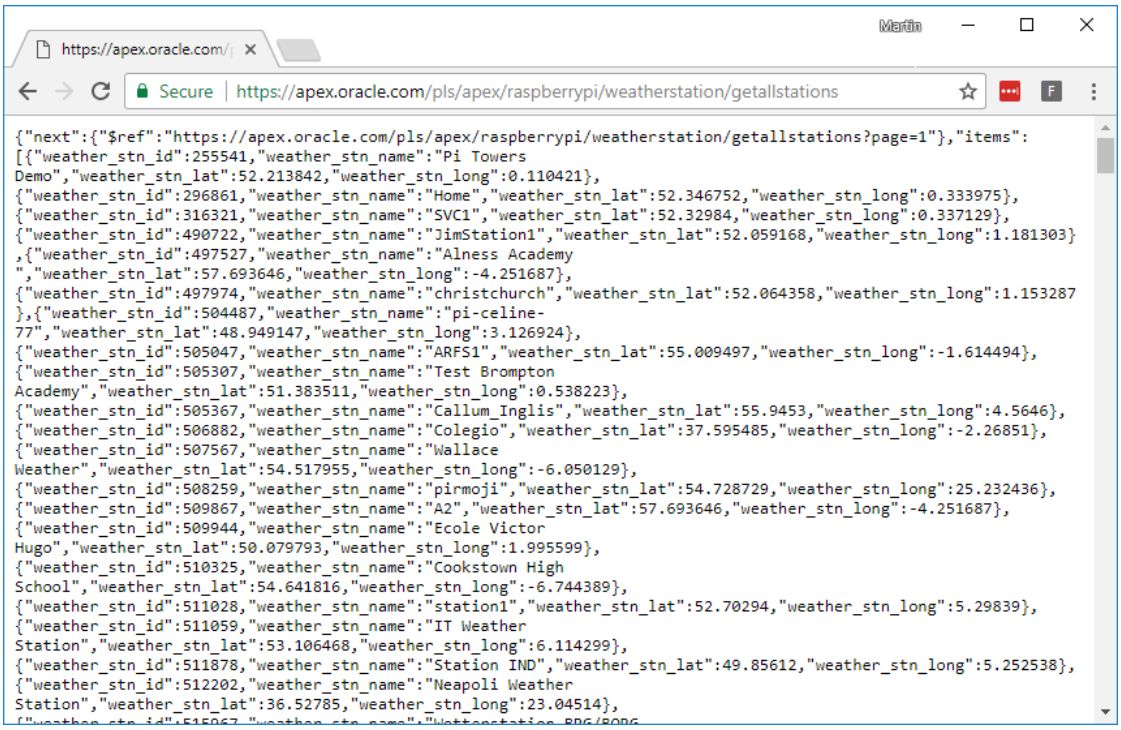
Sensor Name	Purpose
Rain gauge	Measures the volume of rain falling in millimetres
Anemometer	Measures the wind speed in kilometres per hour
Weathervane	Measures the wind direction in degrees
Soil temperature probe	Measures the soil temperature in degrees Celsius
Temperature sensor	Measures the air temperature in degrees Celsius

Sensor Name	Purpose
Humidity sensor	Measures the relative humidity of the air as a percentage
Pressure sensor	Measures the atmospheric pressure in Pascals
Air quality sensor	Measures the air quality as a relative percentage

The Weather Stations continually monitor the weather and then send their data to an Oracle database, where it is stored and can be accessed.
In this resource you're going to learn how to find a Weather Station you're interested in, and then get the latest weather updates from that station.

You can get a list of all the Weather Stations that are currently online, using a simple URL. This is because the database that all the Weather Stations upload data to has a RESTful API. This is a method by which you can write code that uses simple HTTP requests (just like a browser) to fetch the data.
Copy and paste the following URL into a web browser:

```
https://apex.oracle.com/pls/apex/raspberrypi/weatherstation/getallstations
```



You should see a web page filled with data. This is a little difficult to read, though. Luckily, we can grab this data with a little Python code and then present it in a format that's easier to read.

- Open `IDLE Python 3` to open the Python shell
- Open a new file by clicking `File` > `New File`.
- Save your program as `fetch_stations.py`.

- The first thing you'll need is a few Python modules. `requests` is not in the standard library, but you can install it using the instructions from the What you will need section (What you will need).

```
from requests import get
import json
from pprint import pprint
```

The `requests` module allows you to fetch web pages from the World Wide Web. The `json` module allows you to easily read JSON data (which is a way of organising data into dictionaries). The `pprint` module is short for pretty-print, and just makes presenting text a little clearer.

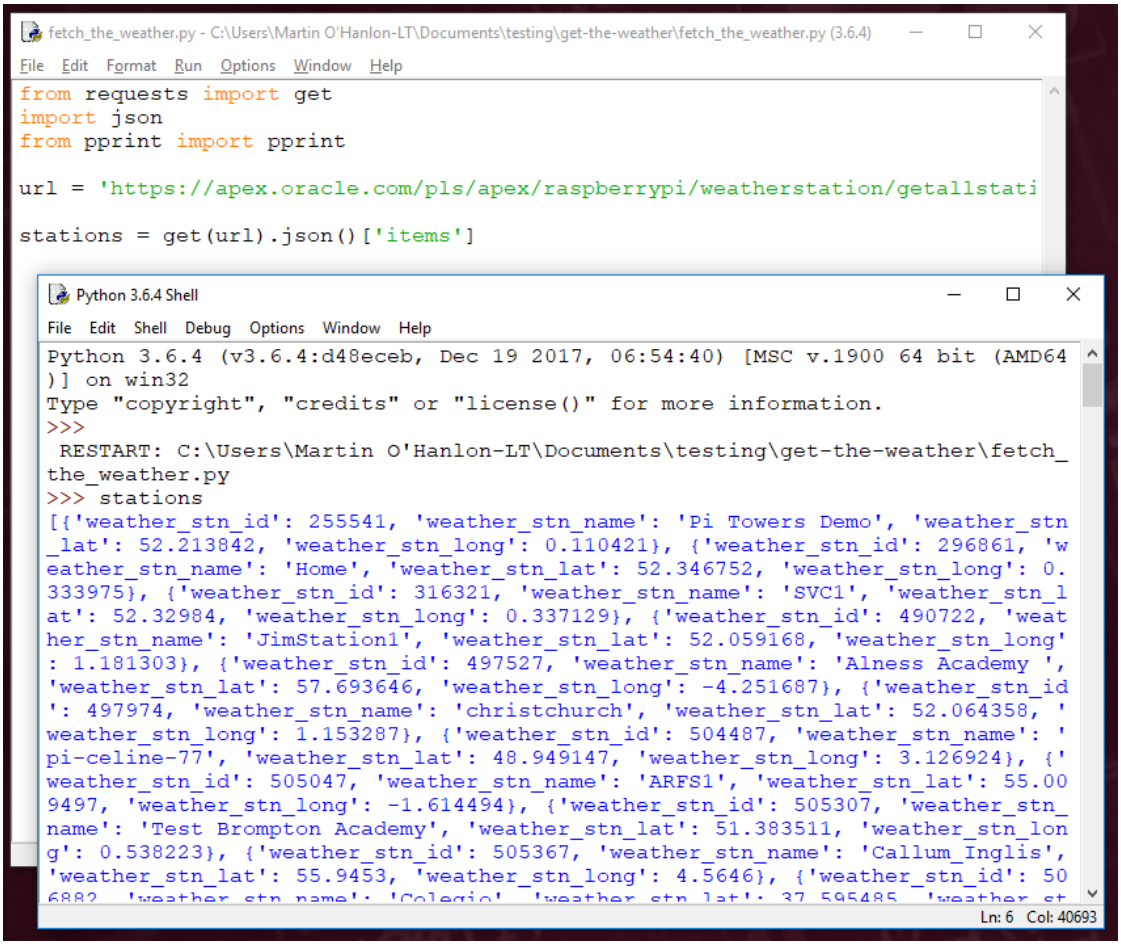
- The next thing to do is to save that URL you used earlier as a variable:

```
url =
'https://apex.oracle.com/pls/apex/raspberrypi/weatherstation/getallstations'
```

- Using `get` from the `requests` module you can now fetch the data, and translate it into Python dictionaries using the `json` module:

```
stations = get(url).json()['items']
```

- Save and run your code.
- Type `stations` into the Python shell to have a look at the data.



- It still looks pretty ugly. Try typing `pprint(stations)` and see what happens. You should see a huge list of Weather Stations dictionaries. Each dictionary should look something like this:

```
{'weather_stn_id': 1648902,
 'weather_stn_lat': 52.197834,
 'weather_stn_long': 0.125366,
 'weather_stn_name': 'ACRG_ROOF'}]
```

What you're seeing is:

- `weather_stn_id` - the unique ID of the station (`weather_stn_id`)
- its location in the world (you will learn about this later in the project)
- `weather_stn_lat` - the latitude
- `weather_stn_long` - the longitude
- `weather_stn_name` - the name of the Weather Station.

For the next part, you're going to need to pick a Weather Station to fetch the weather from.

- Scroll up and down the list and pick a `weather_stn_id` that you'd like to have a look at.

Now that you have a Weather Station to look at, you can learn how to fetch the last weather recording from that station.
This is again handled using the RESTful API of the Weather Station database. This time, the URL you need is made up of two parts. The first tells the database that you're requesting the latest measurements:

```
https://apex.oracle.com/pls/apex/raspberrypi/weatherstation/getlatestmeasurements/
```

You need to add the ID of the Weather Station you wish to access to the end of this. For example:

```
https://apex.oracle.com/pls/apex/raspberrypi/weatherstation/getlatestmeasurements/
```

- Enter the URL of the webstation you would like to see the data from into your web browser.
- Create a new Python file again, by clicking on `File` > `New File`.
- Save your program as `fetch_weather.py`.
- Once again, you'll need the `requests` and `json` modules, as well as `pprint`:

```
from requests import get
import json
from pprint import pprint
```

- Now you can define a new `url` variable, but using the Weather Station ID you've chosen:

```
url =
'https://apex.oracle.com/pls/apex/raspberrypi/weatherstation/getlatestmeasurements
```

- To get the latest measurements you need one line of code, but we'll add a second line to pretty-print it straight away:

```
weather = get(url).json()['items']
pprint(weather)
```

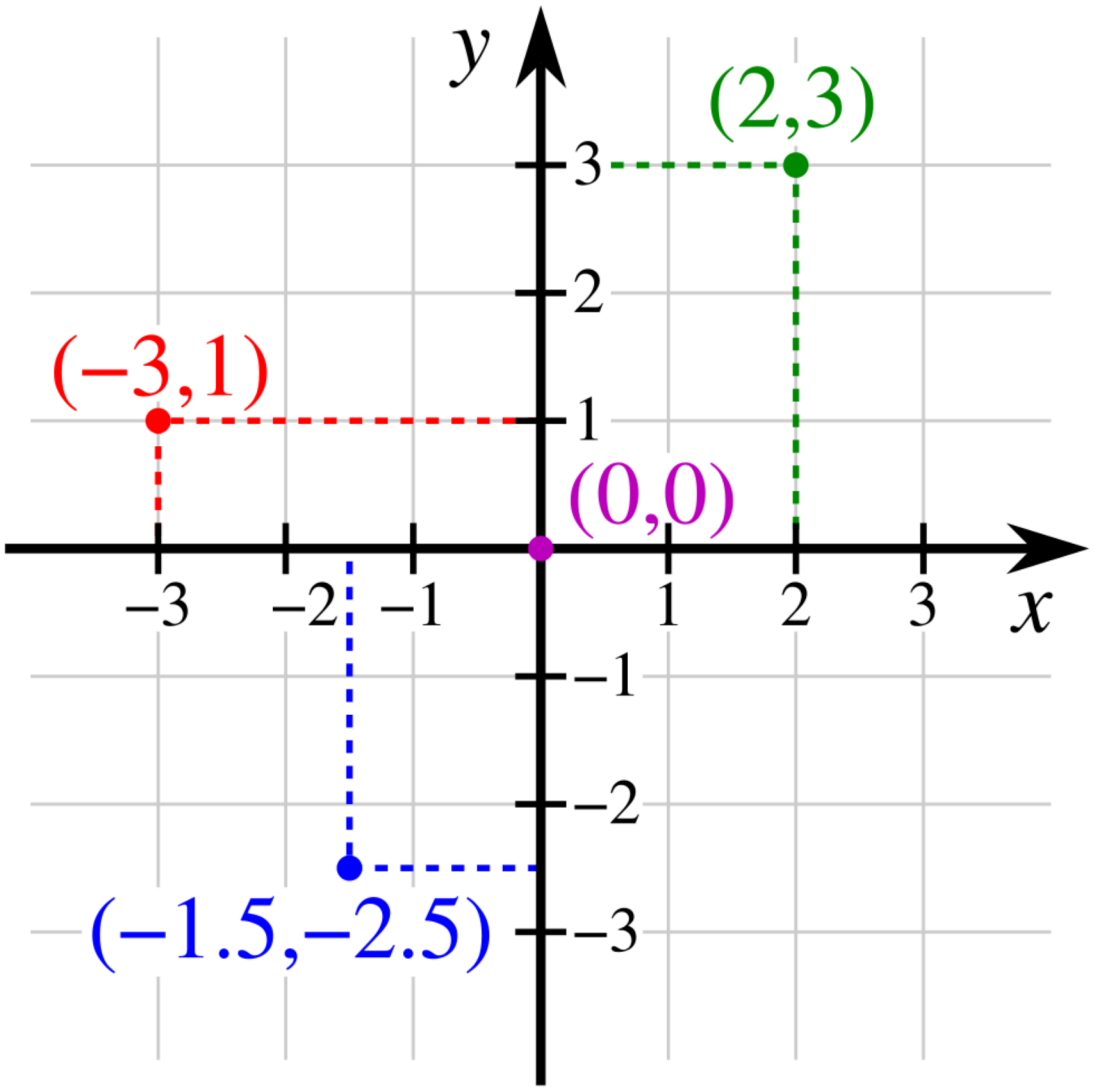
- Run your program
  You should see data from the weather station you picked appear in the shell:


```
[{'air_pressure': 1008.81,
'air_quality': 74.9,
'ambient_temp': 23.58,
'created_by': 'ACRG_ROOF',
'created_on': '2016-11-16T12:00:01Z',
'ground_temp': 18.69,
'humidity': 33.41,
'id': 1669238,
'rainfall': 0,
'reading_timestamp': '2016-11-16T12:00:01Z',
'updated_by': 'ACRG_ROOF',
'updated_on': '2016-11-16T12:05:02.437Z',
'weather_stn_id': 1648902,
'wind_direction': 315,
'wind_gust_speed': 0,
'wind_speed': 0}]
```


- If you don't see any data, it might be because the Weather Station is offline. Just try another Weather Station Id.

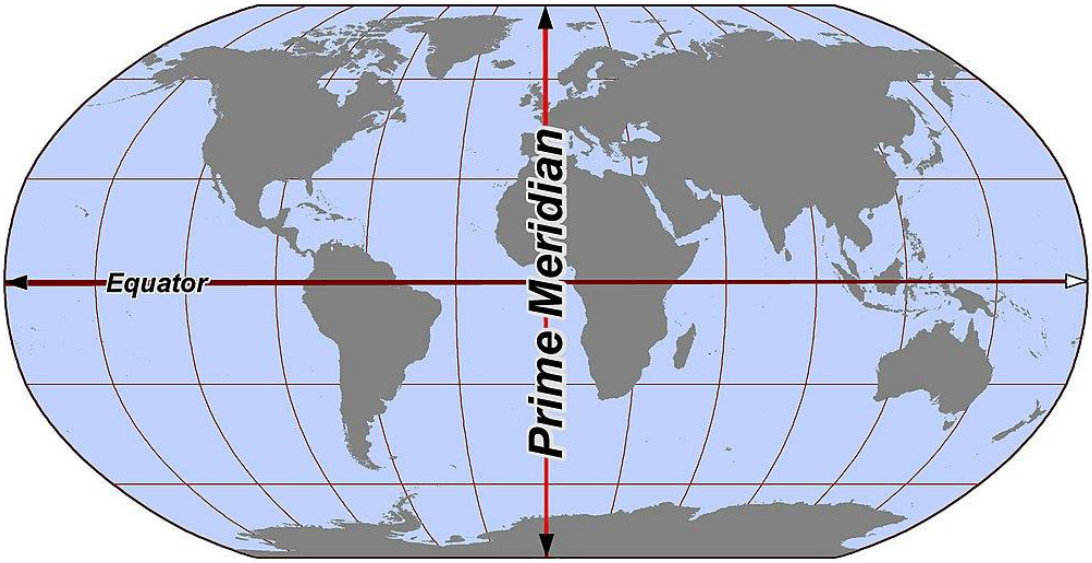
It would be nice if you could pick a Weather Station that's close to you to fetch the data from. You can do this, because the database stores the longitude and latitude of all the Weather Stations around the world. Let's have a look at what we mean by longitude and latitude.

If you wanted to pinpoint a place on a 2D object like a piece of paper, you could use x and y coordinates. The x coordinate would place the point's horizontal position, and the y coordinate would place the vertical position. You can see an example of this below.

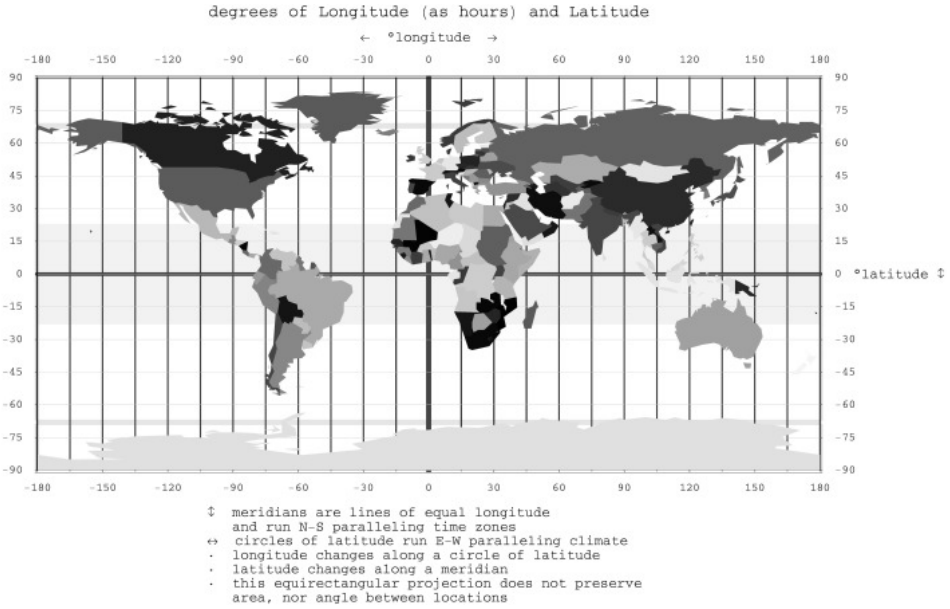


Things aren't so simple when you're trying to pinpoint a location on a sphere, like the Earth. The vertical and horizontal positions wrap around the sphere, for a start. Also, travelling 5 units of distance along the equator would be a completely different distance to walking 5 units of distance near one of the poles. For this reason we use longitude and latitude when locating items on the Earth's surface.

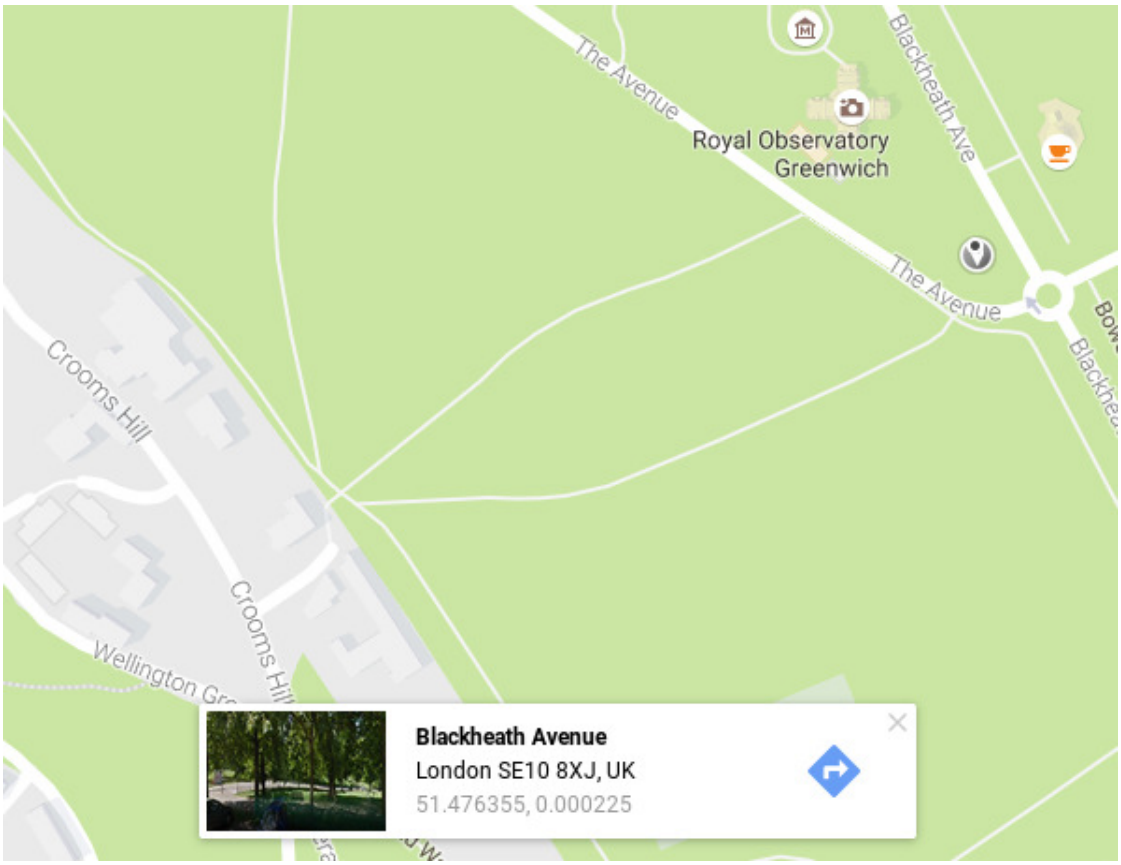
You can draw two imaginary circles around the Earth. The first is called the equator, which you're probably familiar with. The second is called the prime meridian, which passes through both the North and South Poles and also through Greenwich in London.



The centre of these two circles is at the centre of the Earth. Imagine you were standing in the centre of the Earth; you would be able to pinpoint any location on the surface by talking about how many degrees you needed to turn within each of these circles. Longitude tells you how many degrees you need to turn east or west from the prime meridian. Latitude tells you how many degrees you need to turn north or south from the equator.



- To find your longitude and latitude open Google Maps (https://www.google.co.uk/maps/).
- Click on your location on the map and your longitude and latitude will be revealed at the bottom of the screen.



The first number is your latitude and the second is your longitude.

- Make a note of the values you get, as you'll need them later.

The next part is a little technical. You need to be able to find the distance between two points on the Earth, given their longitudes and latitudes. This will allow you to find the closest Weather Station to you.

If you're not particularly interested in how this works, then rather than write the code, you can download the file you need from here (https://raw.githubusercontent.com/raspberrypilearning/fetching-the-weather/master/code/haversine.py). Just make sure it's saved as `haversine.py` and stored in the same directory as the rest of your code.

As discussed earlier, we use longitude and latitude to work out the exact position of places on the Earth. Finding distances between these points is quite tricky, as the distance is over the surface of a sphere, and therefore not in a straight line. To do this calculation, you need a clever bit of maths called the haversine formula (https://en.wikipedia.org/wiki/Haversine_formula).

Without getting too technical, the haversine formula can provide the distance between two points on a sphere using longitudes and latitudes.

- Create a new program by clicking on `File` > `New File` in the Python script.
- Click on `File` > `Save As` and call your file `haversine.py`.
- To begin with, you're going to need a few functions from the `maths` library. Start off your file by importing the following:

```
from math import radians, cos, sin, asin, sqrt
```

- Now you can define a new function, which we'll call `haversine`. It's going to take 4 arguments, which will be the longitude and latitude of the two points whose distance we need to find.

```
def haversine(lon1, lat1, lon2, lat2):
```

- Most mathematical formulae require us to work in radians (http://www.bbc.co.uk/bitesize/higher/maths/trigonometry/radian_and_equations/revision/1/) rather than degrees when dealing with angles, so the first thing to do is to convert each of the latitudes and longitudes passed into the function as arguments into radians.

```
def haversine(lon1, lat1, lon2, lat2):
    #convert degrees to radians
    lon1 = radians(lon1)
    lat1 = radians(lat1)
    lon2 = radians(lon2)
    lat2 = radians(lat2)
```

- Now we want to find the difference between the two longitudes and latitudes, so add this into your function:

```
    dlon = lon2 - lon1
    dlat = lat2 - lat1
```

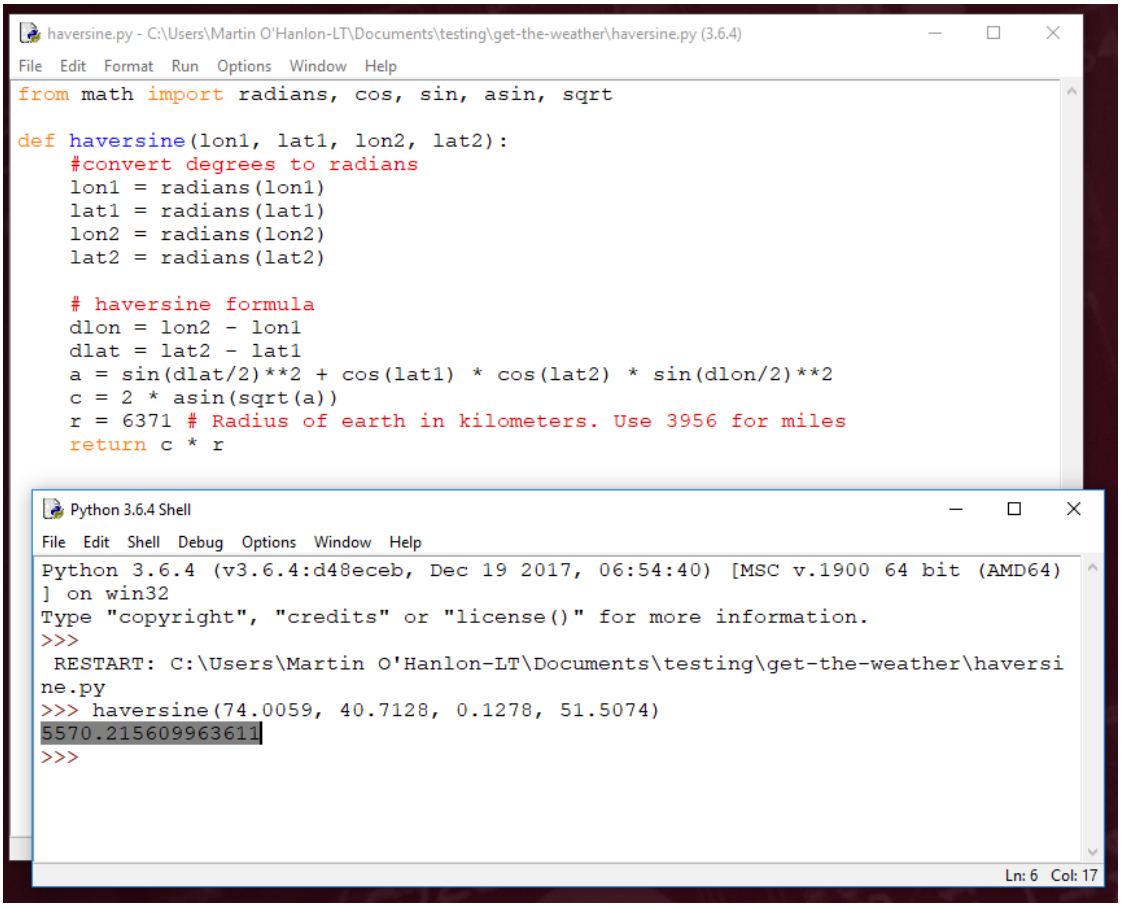
- Now comes the tricky bit. If you want to know more about the haversine formula then you can have a read of the Wikipedia article linked above. Otherwise, you can just take it at face value that the following lines of code will calculate the distance between the two points:

```
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    distance = 2 * asin(sqrt(a)) * 6371 #6371 is the radius of the Earth
    return distance
```

The number `6371` in the code above is the radius of the Earth

- Save your file and run your program to test it.
- In the Python shell type the following:

```
haversine(74.0059, 40.7128, 0.1278, 51.5074)
```



You should get an answer of 5570. This is the distance from London to New York. You can check the answer online if you like, although the values will be slightly different as the Earth is not an exact sphere. It's good enough for our purposes, though.

- Try a few more longitudes and latitudes from Google Maps.

Step 8 Fetch your local weather

At the start of this project, you fetched all the Weather Stations that are currently registered. The data came in as a huge list of dictionaries. By iterating through this list, you can pick out the longitude and latitude of the Weather Stations, and then run it through your haversine function to find the closest one.

s

- Create a new Python file (`File` > `New File`)
- Save the file as `fetch_local_weather.py` and make sure you save it in the same directory as your `haversine.py` file.
- Start by importing the `requests`, `json`, and `pprint` modules that you used before, but you can now also import your haversine function:

```
from requests import get
import json
from pprint import pprint
from haversine import haversine
```

- Previously you used two URLs to get the Weather Stations and the latest weather. You can declare these variables straight away:

```
stations = 'https://apex.oracle.com/pls/apex/raspberrypi/weatherstation/getallstations'
weather = 'https://apex.oracle.com/pls/apex/raspberrypi/weatherstation/getlatestmeasurements
```

The second URL isn't complete, as you need to add the weather station ID to the end. You're going to do that in code.

- Now add in variables for your current longitude and latitude, that you found using Google Maps:

```
my_lat = 52.194504
my_lon = 0.134708
```

- To finish off this section, you can fetch the list of all stations, just like you did in Worksheet One:

```
all_stations = get(stations).json()['items']
```

For this to work, you're going to need to run the longitude and latitude of all the stations through the haversine function. The trick will be finding the smallest distance to your current longitude and latitude, and saving this as a variable.

- Start by defining a new function, and setting a variable within it for the smallest distance. The longest possible distance between two points on the Earth's surface is 20036km, so this would be a good place to start the variable:

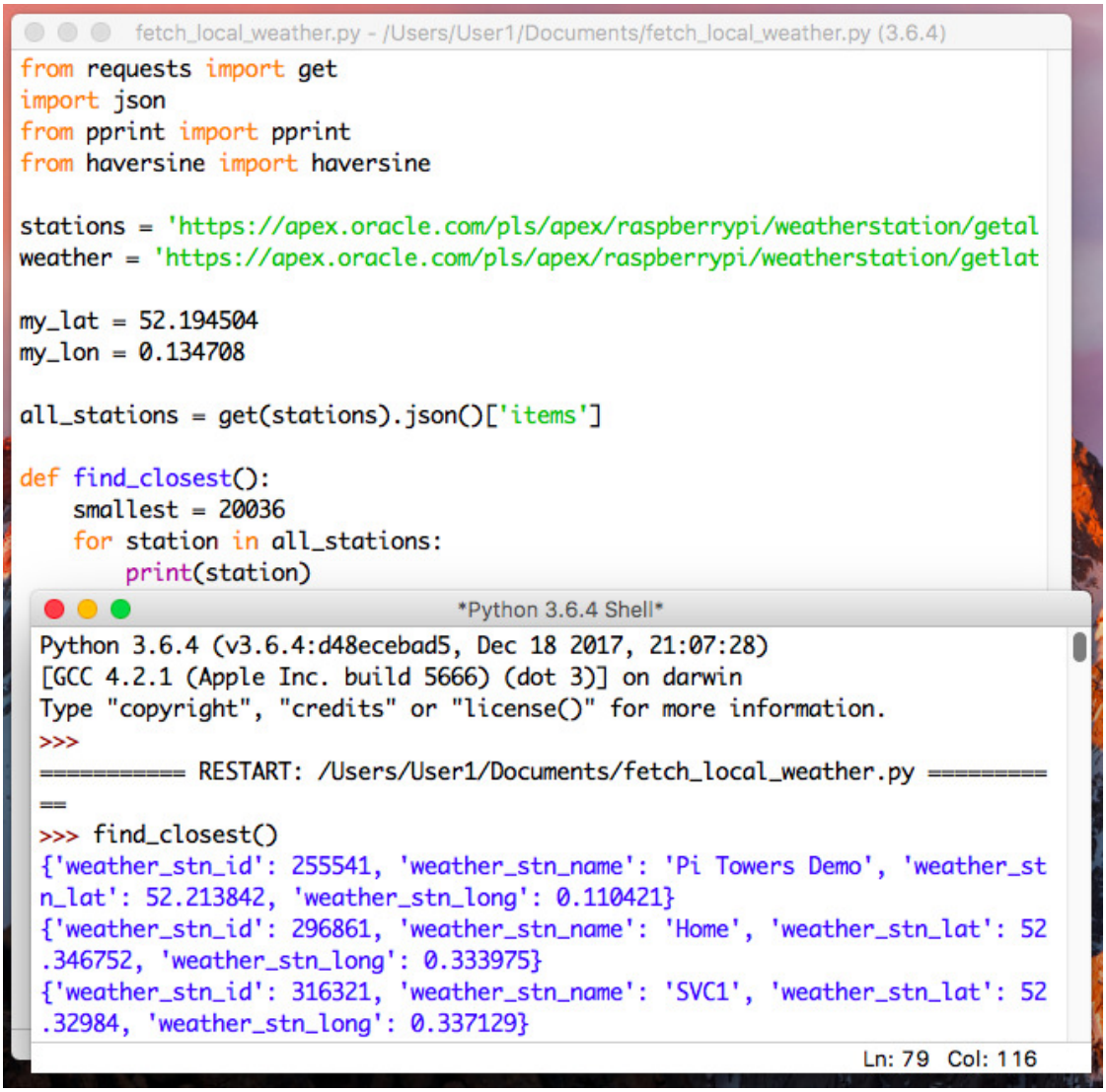
```
def find_closest():
    smallest = 20036
```

- Now you can use a `for` loop to iterate through all the stations. Let's start by printing the data for each:

```
    for station in all_stations:
        print(station)
```

- Run your program and type the following into the shell to get the list of stations:

```
find_closest()
```



You should see a large list of dictionaries, with each dictionary looking something like this:

```
{'weather_stn_name': 'ACRG_ROOF', 'weather_stn_lat': 52.197834,
'weather_stn_id': 1648902, 'weather_stn_long': 0.125366}
```

The data we're interested in is the `'weather_stn_lat'` and `'weather_stn_long'`. These are the values we want to use in the haversine function.

- Go back to your program; you can now get those values in the `find_closest` function. Remove the `print(station)` line and then add the following:

```
        station_lon = station['weather_stn_long']
        station_lat = station['weather_stn_lat']
```

- Now that you have all the data, it can be run through the haversine function to find the station's distance to you:

```
            distance = haversine(my_lon, my_lat, station_lon, station_lat)
            print(distance)
```

- Run the code again and type `find_closest()` in the shell again. That's a *long* list of distances.
- Next, you need to find the smallest one and then save that station's ID. If the distance is smaller than the `smallest` variable it can be saved, and then next time around the loop it can be checked again.

```
            if distance < smallest:
                smallest = distance
                closest_station = station['weather_stn_id']
        return closest_station
```

Your `find_closest` function should now look like this:

```
def find_closest():
    smallest = 20036
    for station in all_stations:
        station_lon = station['weather_stn_long']
        station_lat = station['weather_stn_lat']
        distance = haversine(my_lon, my_lat, station_lon, station_lat)
        if distance < smallest:
            smallest = distance
            closest_station = station['weather_stn_id']
    return closest_station
```

Be really careful with the indents otherwise the function wont work properly.

Now that you can get the closest Weather Station to you, you can get the data from that weather station just like you did previously.

- Start by calling your newly created function and saving the weather station ID:

```
closest_stn = find_closest()
```

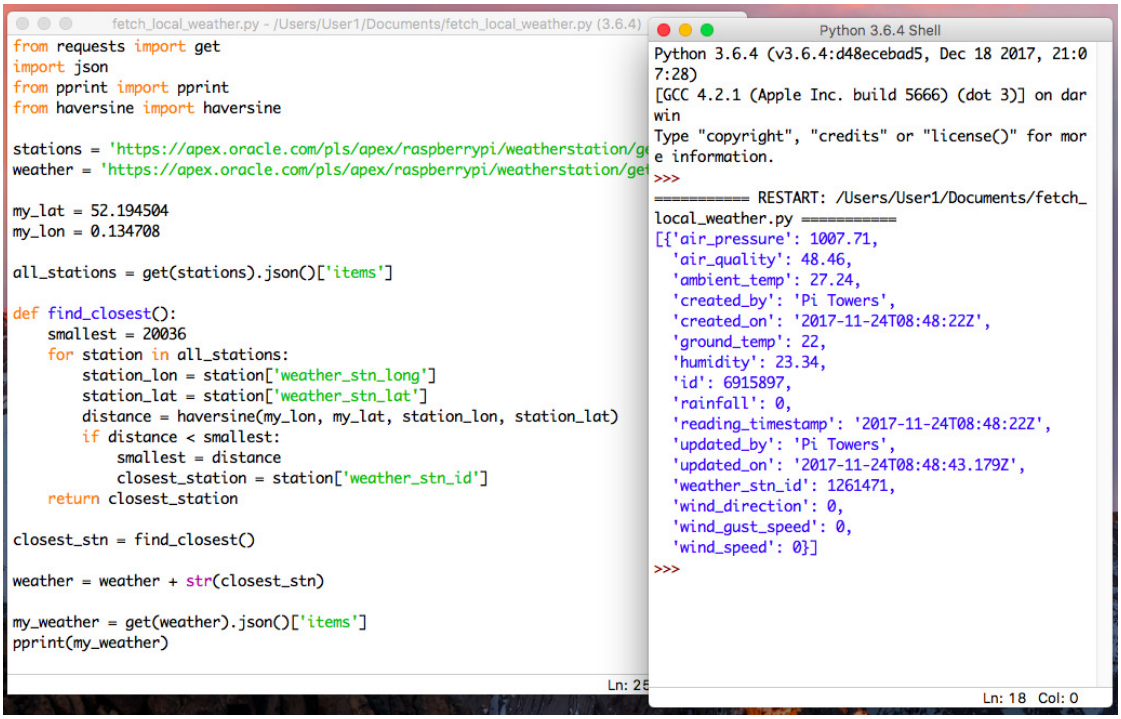
- Now this can be added to the end of the `weather` variable that stores the URL. It's an integer at the moment though, so it needs to be changed to a string:

```
weather = weather + str(closest_stn)
```

- Finally, you can use `requests` to get the data and then pretty-print it:

```
my_weather = get(weather).json()['items']
pprint(my_weather)
```

- Run your code and you should see the weather data for the station nearest you, printed out in the shell.


fetch_local_weather.py - /Users/User1/Documents/fetch_local_weather.py (3.6.4)
from requests import get
import json
from pprint import pprint
from haversine import haversine
stations = 'https://apex.oracle.com/pls/apex/raspberrypi/weatherstation/g
weather = 'https://apex.oracle.com/pls/apex/raspberrypi/weatherstation/ge
my_lat = 52.194504
my_lon = 0.134708
all_stations = get(stations).json()['items']
def find_closest():
smallest = 20036
for station in all_stations:
station_lon = station['weather_stn_long']
station_lat = station['weather_stn_lat']
distance = haversine(my_lon, my_lat, station_lon, station_lat)
if distance < smallest:
smallest = distance
closest_station = station['weather_stn_id']
return closest_station
closest_stn = find_closest()
weather = weather + str(closest_stn)
my_weather = get(weather).json()['items']
pprint(my_weather)
Ln: 25
Python 3.6.4 Shell
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: /Users/User1/Documents/fetch_local_weather.py ==========
[{'air_pressure': 1007.71,
'air_quality': 48.46,
'ambient_temp': 27.24,
'created_by': 'Pi Towers',
'created_on': '2017-11-24T08:48:22Z',
'ground_temp': 22,
'humidity': 23.34,
'id': 6915897,
'rainfall': 0,
'reading_timestamp': '2017-11-24T08:48:22Z',
'updated_by': 'Pi Towers',
'updated_on': '2017-11-24T08:48:43.179Z',
'weather_stn_id': 1261471,
'wind_direction': 0,
'wind_gust_speed': 0,
'wind_speed': 0}]
>>>
Ln: 18 Col: 0

Step 11 What next?

- You could have a look at some weather data from other locations in the world. Use the web to find some longitudes and latitudes of other places, and then fetch weather from their nearest stations.
- How about importing the data from several Weather Stations into some spreadsheet software and drawing some graphs? Or maybe you'd like to try and use Python to draw some graphs for you?
- Create a simple interface using the Python function `input` which asks for your longitude and latitude and finds the weather near you.