

Table of Contents

I.	INTRODUCTION	2
II.	PRE-REQUISITES	2
III.	INSTALLATIONS	3
A.	IDE	3
B.	BREW	3
C.	PIPVENV	3
D.	PYTHON	3
E.	SOURCE FROM TAR	4
F.	SOURCE FROM GIT (IF TAR PROVIDED – CAN SKIP).....	4
IV.	CODE LAYOUT	5
A.	SOURCE CODE.....	5
B.	UNIT TESTS	6
V.	RUNNING THE APPLICATION	7
VI.	TASKS	8
A.	TASK 1 : DISPLAY WIND DIRECTION	8
B.	TASK 2 : DISPLAY WIND CARDINAL DIRECTION	11
C.	TASK 3 : UNIT TESTS	13
D.	TASK 4 : DISPLAY WEATHER FROM LONGITUDE AND LATITUDE.....	16
VII.	DOCUMENTATION REFERENCES.....	19

I. Introduction

This workshop will guide you through the process of creating a simple **python3** web-app, utilizing the **Flask** web framework. The app will retrieve weather information for a given location or longitude/latitude and displays max/min/current temperature, wind speed information and a brief weather status along with a .png icon for display.

OpenWeatherMap is one of the most popular and common weather APIs, we will be using this in conjunction with **Flask** (a popular python-based web-framework) to create our python3 application.

This workshop will guide you through the following steps:

- Installing an IDE for python3 development
- Installing brew and pipenv dependencies
- Cloning the project repository
- Initialising pipenv with python dependencies
- Setting up the config file with API key for **OpenWeatherMap**
- Running Flask and the initial webapp
- Extending functionality to include wind speed per location
- Unit-testing our config validation method
- EXTENSION: Expanding the app to search by longitude/latitude

II. Pre-Requisites

- OpenWeatherMap valid API key
- Macbook or linux-based machine with a terminal emulator

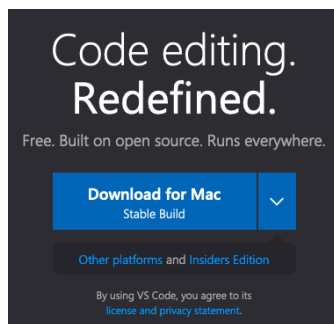
III. Installations

A. IDE

Firstly, we will require an **IDE** (integrated development environment) to speed up the development process and provide useful helpers when writing Python3 code. **VSCode** is recommended for its simplicity and lightweight installation, this can be installed from here:

<https://code.visualstudio.com/>

Follow the download link and unzip the .zip archive, copy the application to your “Applications” folder.



B. Brew

Brew is an extremely useful and versatile package manager tool for MacOS (like apt for Ubuntu or yum for Fedora) and is necessary to install the next dependency.

Run the following command to install Brew:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
```

C. Pipenv

When developing an application in Python, we often rely on packages (<https://pypi.org/>) to speed up the development of our code. For this particular project, we require the package **pipenv** to allow the creation of a virtual environment containing all dependencies.

Pipenv can be installed as follows:

```
brew install pipenv
```

D. Python

POTENTIAL SECTION HERE ON INSTALLING PYTHON3, VERIFY IF MACOS CATALINA SHIPS WITH IT IN /usr/bin/python3, if not we can install with brew update && brew upgrade python

E. Source from Tar

You will be provided with a .tar.gz containing all the workshop content. Untar this to your local filesystem.

F. Source from Git (If tar provided – can skip)

We will need to clone the workshop source code from a git repository, try run “**git**” from a terminal. A prompt to install “xcode developer tools” will appear and install git. If no prompt appears – great! You already have git installed on your machine.

IV. Code Layout

A. Source code

The code is organized into a typical project structure as follows:




1. **Pipfile & Pipfile.lock:** this file contains all the dependencies for the current project, the lock file ensures versions aren't automatically updated.
2. **README.md:** most projects contain a README file with basic run instructions and useful information.
3. **instance/** : this is a folder automatically generated by **pipenv**.
4. **tests/** : all unit tests for the code are within this directory.
5. **weather/** : all main application code is within this directory.
6. **weather/config.yaml** : This file contains the following configuration parameters:
 - a. **API key:** this is required to interact with **OpenWeatherMap** API
Unit in which we want to measure temperature, currently configured as '**celsius**' but can also be '**kelvin**' or '**fahrenheit**'
7. **weather/loader:** This module loads and validates the **config.yaml** file ensuring both **api_key** and a valid value for **temperature** are set.
8. **weather/weather_api** : This module creates a wrapper around **pyowm**.
Pyowm is an open source module we are using as a dependency within this project, it allows for simple interaction and integration with **OpenWeatherMap** API methods.
Further details about this package can be found here <https://github.com/csparpa/pyowm>

9. **weather/templates:** This folder contains html templates that are used by flask to render webpages that can directly interact with python variables (input a location, output some information).

WeatherApp

Submit

Current weather in London

Status: Clouds 

Temperature	Average	Max	temp_min
	15.89	17.22	15.0

B. Unit tests

The unit test folder is where currently there is a unittest file `test_validate_config.py` to verify the config loader functionality.

The unit tests are currently blank and will need to be implemented as part of your task.

Running the unit tests:

Tests can be run using the following command:

```
pipenv run python -m unittest discover -s tests -v
```

```
(src) → src git:(master) X
(src) → src git:(master) X
(src) → src git:(master) X pipenv run python -m unittest discover -s tests -v
test_missing_fields_validation (test_validate_config.TestValidateConfig) ... ok
test_standard_validation (test_validate_config.TestValidateConfig) ... ok
test_wrong_temperature_units_validation (test_validate_config.TestValidateConfig) ... ok
# sds-notifications
-----
Ran 3 tests in 0.000s
OK
# skygirls
# j. Morewood
(src) → src git:(master) X
```

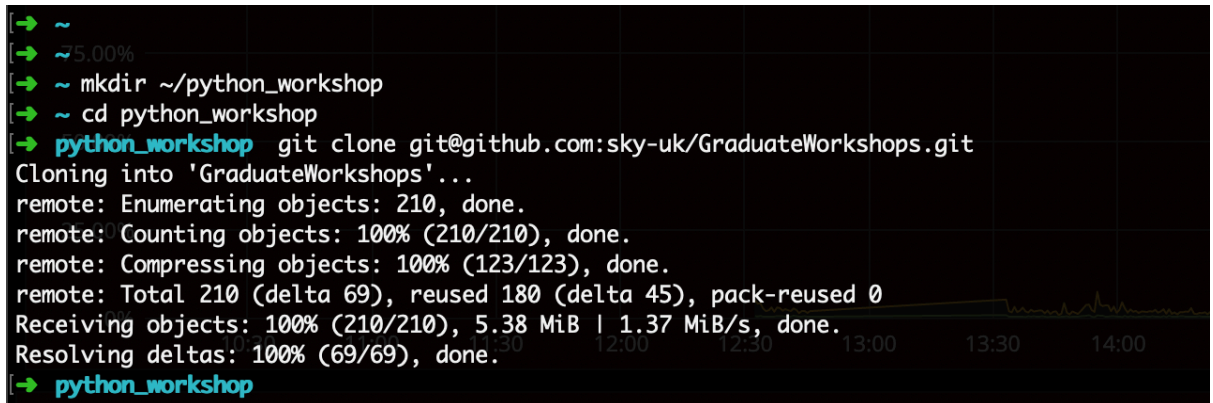
You can target a specific test by referencing the function name within the “**TestValidateConfig**” class.

```
(src) → src git:(master) X
(src) → src git:(master) X
(src) → src git:(master) X pipenv run python -m unittest tests.test_validate_config.TestValidateConfig.test_missing_fields_validation -v
test_missing_fields_validation (tests.test_validate_config.TestValidateConfig) ... ok
# sds-notifications
-----
Ran 1 test in 0.000s
OK
# skygirls
# j. Morewood
(src) → src git:(master) X
```

Note: in case you get any errors related to **'ModuleNotFoundError'** then it is worth trying to run **'pipenv sync'** to ensure the dependencies are locked to the correct version.

V. Running the Application

1. Within the terminal, make a new directory for the project in your home folder called **"python_workshop"** and navigate to that directory:



```
→ ~  
→ ~5.00%  
→ ~ mkdir ~/python_workshop  
→ ~ cd python_workshop  
→ python_workshop git clone git@github.com:sky-uk/GraduateWorkshops.git  
Cloning into 'GraduateWorkshops'...  
remote: Enumerating objects: 210, done.  
remote: Counting objects: 100% (210/210), done.  
remote: Compressing objects: 100% (123/123), done.  
remote: Total 210 (delta 69), reused 180 (delta 45), pack-reused 0  
Receiving objects: 100% (210/210), 5.38 MiB | 1.37 MiB/s, done.  
Resolving deltas: 100% (69/69), done.  
→ python_workshop
```

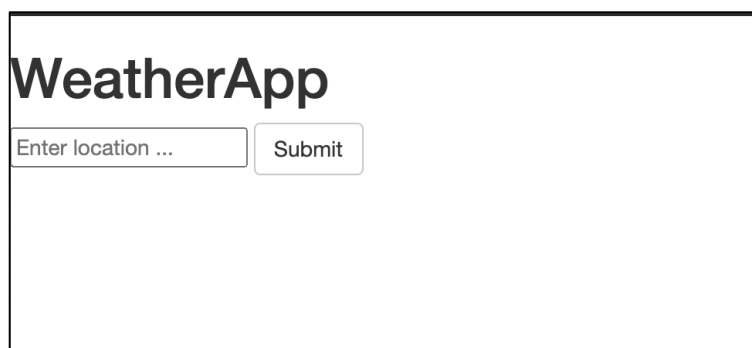
2. Place the API key for **OpenWeatherMap** you received as part of this workshop into **"src/weather/config.yaml"** under **"api_key"**.
3. Now run the following commands

./run.sh

NOTE: This will execute the following commands, and has been included as a helper script:

```
`FLASK_APP=weather/main.py pipenv run python -m flask run`
```

4. Success! You have a development webserver now running locally serving your python web application! Click on the URL <http://127.0.0.1:5000/> and it should take you to the web app as follows:



The screenshot shows a web browser displaying a page titled "WeatherApp". Below the title, there is a text input field with the placeholder text "Enter location ..." and a "Submit" button to its right. The rest of the page is currently blank.

5. Now enter a location such as 'London', and click submit. The app will then extract a variable from this request form, and send via an argument to **get_current_weather_at_location()**.

This will in turn retrieve a dictionary object which is, rendered via Flask onto this webpage. (this can be observed in "**templates/index.html**", using jinja syntax to access elements of a dictionary) -

<https://jinja.palletsprojects.com/en/2.11.x/templates/>

WeatherApp

Current weather in London

Status: Clouds ☁

Temperature	Average	Max	temp_min
	15.89	17.22	15.0

6. If you enter an incorrect location say 'xxx' and submit then an error page will be displayed. This is handled by checking for any errors returned by our weather_api in the form of a dictionary. "**templates/500.html**" renders any errors passed as arguments.

WeatherApp

An unexpected error has occurred

Sorry for the inconvenience!

VI. Tasks

A. Task 1 : Display wind direction

1. Get wind info

We query the **OpenWeatherMap API** for weather information using the **weather_at_place** method within **pyowm**. An example of the structured API response that we receive is as follows:


```

{"coord": { "lon": 139,"lat": 35},
  "weather": [
    {
      "id": 800,
      "main": "Clear",
      "description": "clear sky",
      "icon": "01n"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 281.52,
    "feels_like": 278.99,
    "temp_min": 280.15,
    "temp_max": 283.71,
    "pressure": 1016,
    "humidity": 93
  },
  "wind": {
    "speed": 0.47,
    "deg": 107.538
  },
  "clouds": {
    "all": 2
  },
  "dt": 1560350192,
  "sys": {
    "type": 3,
    "id": 2019346,
    "message": 0.0065,
    "country": "JP",
    "sunrise": 1560281377,
    "sunset": 1560333478
  },
  "timezone": 32400,
  "id": 1851632,
  "name": "Shuzenji",
  "cod": 200
}

```

This response contains a dictionary with wind info which contains the two keys “**speed**” and “**deg**”.

In **api.py**, under the **weather_api** folder, we have a function called **get_current_weather_at_location()**. Here we retrieve a **weather_location** object.

Within the “**weather_api/api.py**” method, note the **get_status()** and **get_temperature()** functions. These are helper methods within **pyowm** we can use to retrieve useful pythonic objects from the **OpenWeatherMap** API.

Using the docs below as a module reference, implement an existing method within **pyowm** to retrieve wind speed in degrees.

<https://buildmedia.readthedocs.org/media/pdf/pyowm/latest/pyowm.pdf>

*HINT: This is a method within **weather_at_place()**.*

*Note: **temperature_unit** is defined in the **config.yaml** file, it is recommended to define and use **wind_unit** here as the **unit** argument to **get_wind()**.*

*Using **config.yaml** to store these options allows our application to be highly configurable, and we desire a single source of truth for all options.*

2. Display wind information on the webpage

Once we have the wind info it should be appended to the dictionary **weather_dict** in the **get_current_weather_at_location()** function. This can be done in the same way as temperature.

weather_dict then needs to be referenced in **templates/index.html**. Reference the wind speed and degree direction in **index.html** as follows.

Note: Ensure the keys in your dictionary match the value you are querying via the below Jinja syntax.

```
<h4>Wind speed: {{ weather['speed'] }}</h4>
<h4>Wind degree direction: {{ weather['deg'] }}</h4>
```

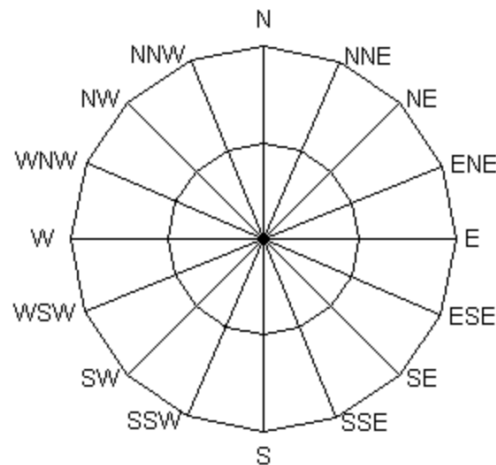
B. Task 2 : Display wind cardinal direction

1. Wind degrees to text utility function

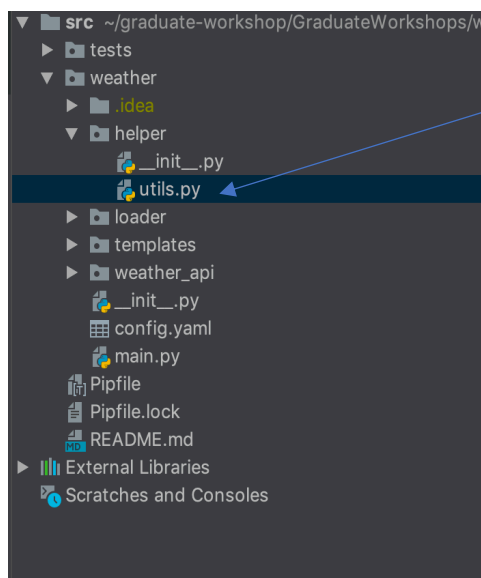
The following table can be used to map the wind degree direction to wind cardinal direction. In this task we will only consider N, NE, E, SE, S, SW, W, NW directions.

Table 3: Wind Direction and Degrees

Cardinal Direction	Degree Direction
N	348.75 - 11.25
NNE	11.25 - 33.75
NE	33.75 - 56.25
ENE	56.25 - 78.75
E	78.75 - 101.25
ESE	101.25 - 123.75
SE	123.75 - 146.25
SSE	146.25 - 168.75
S	168.75 - 191.25
SSW	191.25 - 213.75
SW	213.75 - 236.25
WSW	236.25 - 258.75
W	258.75 - 281.25
WNW	281.25 - 303.75
NW	303.75 - 326.25
NNW	326.25 - 348.75



To convert wind degrees to a cardinal direction we will first need to create a helper function. Under the weather folder create a folder titled **"helper"**, then create a file titled **utils.py**.



In this file we need to add the function **wind_deg_to_text()** as given below.

```
# Accepts wind direction in degrees
def wind_deg_to_text(degree):
    sectors = ['Northerly', 'North Easterly', 'Easterly', 'South Easterly',
               'Southerly', 'South Westerly', 'Westerly',
               'North Westerly']

    degree += 22.5

    if degree < 0:
        degree = 360 - abs(degree) % 360
    else:
        degree = degree % 360

    which_sector = int(degree // 45)
    return sectors[which_sector]
```

This function will calculate the wind degree direction using the table illustrated above, and map to the corresponding cardinal direction string within the “sectors” dictionary.

2. Referencing the utility function

Now we are successfully retrieving the wind speed in degrees from **OpenWeatherMap**, it would be useful to utilise the helper function we created above to map this to a friendly cardinal direction.

The above created utility function should be referenced within the **get_current_weather_at_location()** function in “**weather_api/api.py**”.

Pass wind speed in degrees to the utility function **wind_deg_to_text()** and return the cardinal direction as a string.

3. Display wind cardinal direction

Once cardinal direction is returned as a string, store it within the **weather_dict** dictionary in **get_current_weather_at_location()**.

“**templates/index.html**” will need to be modified to display this newly created dictionary key/value. Add a html line to display the wind direction referencing it from this dictionary.

Example:

```
<h4>Wind cardinal direction: {{ weather['wind_direction'] }}</h4>
```

*NOTE: This must be passed as an argument to flask’s **render_template** function to be used within jinja templating.*

C. Task 3 : Unit Tests

1. To implement test_validate_config.py

This file is mainly to test validate_config functionality. Currently the functions in the file are not implemented and the template is as follows

```
class TestValidateConfig(unittest.TestCase):
    """Unit tests to ensure validate_config works correctly"""

    # TODO fill in unit tests

    def test_standard_validation(self):
        raise NotImplementedError

    def test_missing_fields_validation(self):
        raise NotImplementedError

    def test_wrong_temperature_units_validation(self):
        raise NotImplementedError
```

As you can see in the file template, the aim of a unit test is to cover the normal working scenario, the failure scenarios and any edge cases. If you run the unit test now then they will error

```
(src) ~ src git:(master) x pipenv run python3 -m unittest discover -s tests -v
Courtesy Notice: Pipenv found itself running within a virtual environment, so it will automatically use that environment, instead of creating its own for any project. You can set PIPENV_IGNORE_VIRTUAL
ENV=1 to force pipenv to ignore that environment and create its own instead. You can set PIPENV_VERBOSITY=-1 to suppress this warning.
test_missing_fields_validation (test_validate_config.TestValidateConfig) ... ERROR
test_standard_validation (test_validate_config.TestValidateConfig) ... ERROR
test_wrong_temperature_units_validation (test_validate_config.TestValidateConfig) ... ERROR

=====
ERROR: test_missing_fields_validation (test_validate_config.TestValidateConfig)
-----
Traceback (most recent call last):
  File "/Users/NTE02/graduate-workshop/GraduateWorkshops/workshops/01-python/src/tests/test_validate_config.py", line 13, in test_missing_fields_validation
    raise NotImplementedError
NotImplementedError

=====
ERROR: test_standard_validation (test_validate_config.TestValidateConfig)
-----
Traceback (most recent call last):
  File "/Users/NTE02/graduate-workshop/GraduateWorkshops/workshops/01-python/src/tests/test_validate_config.py", line 10, in test_standard_validation
    raise NotImplementedError
NotImplementedError

=====
ERROR: test_wrong_temperature_units_validation (test_validate_config.TestValidateConfig)
-----
Traceback (most recent call last):
  File "/Users/NTE02/graduate-workshop/GraduateWorkshops/workshops/01-python/src/tests/test_validate_config.py", line 16, in test_wrong_temperature_units_validation
    raise NotImplementedError
```

Each of these unit tests must be implemented. If we take an example of the first case then we need to call validate_config with a correct set of values and then test the behaviour. We also need to make sure that the return value is what we expect, in the example below we are using assertTrue to achieve this. The various unittest assertions can be found here -

<https://docs.python.org/3/library/unittest.html>

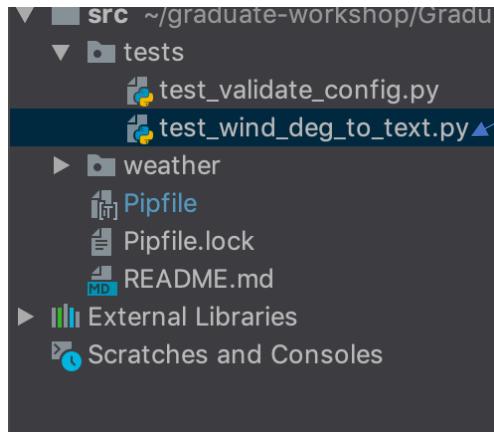
```
def test_standard_validation(self):
    config = {"temperature_unit": "celsius", 'api_key': "xxx"}
    self.assertTrue(validate_config(config))
```

2. For Wind degrees to text added in Task 2

In Task2 we created the function **wind_deg_to_text()** within “**helper/utils.py**”.

Unit tests are now required to ensure that that the function behaves correctly when given different input parameters..

In the project we already have a tests folder, you can create a new file in there as follows:



In this file we can use the following template to for our unit tests:

```
import unittest
from weather.helper.utils import wind_deg_to_text

class TestWindDegToText(unittest.TestCase):

if __name__ == '__main__':
    unittest.main()
```

Within the **TestWindDegToText** class we now need to add some positive and negative test cases.

1. Positive test case: If the function **wind_deg_to_text()** receives an input value between 0 and 360 then it will be a positive test case. It can be something as follows:

```
def test_standard_wind_deg(self):
    wind_direction = wind_deg_to_text(60)
    self.assertTrue(wind_direction == 'North Easterly')
```

In the above scenario we have given a normal wind degree and are checking if the return value is as we expect – the cardinal direction is mapped correctly.

2. Negative test case: If the function **wind_deg_to_text()** receives an input value which is negative. e.g: -45 or something greater than 360.

Write a test case for this scenario.

3. Edge cases: If the function **wind_deg_to_text()** receives an input value which is 0 or 360.

Write a test case for this scenario.

HINT: When considering wind direction in degrees, examine 0 and 360. Refer to the cardinal direction table if required.

Once all the test cases are in place you should be able to run the tests.

Refer to section “**B. Unit Tests:** Running the unit tests” for instructions on how to run these tests.

D. Task 4 : Display weather from Longitude and Latitude

1. Create Longitude and Latitude input parameters

Currently, only location is accepted as a valid input to our weather API functions. We now need to extend this input form to accommodate for Latitude and Longitude as input values.

For this we need to modify the “**templates/index.html**”. In here you will see we already have an `<input>` tag for location. This needs to be extended for Longitude and Latitude:

```
<form method='POST' action='/>
  <input type="text" name="location" placeholder="Enter location ..." style="max-
width: 10000px;" autofocus required/>
  <input type="text" name="longitude" id="longitude" placeholder="Enter longitude
..." style="max-width: 10000px;" autofocus required/>
  <input type="text" name="latitude" id="latitude" placeholder="Enter latitude
..." style="max-width: 10000px;" autofocus required/>
  <!-- -->
  <button type="submit" class="btn btn-default">Submit</button>
</form>
```

The default page when loaded should be displayed as follows:



The screenshot shows a web form titled "WeatherApp". It contains three text input fields with placeholders "Enter location ...", "Enter longitude ...", and "Enter latitude ...". To the right of these fields is a "Submit" button. The form is enclosed in a simple rectangular border.

2. Read Longitude and Latitude parameters

In the “**weather/main.py**” we need to read the parameters exposed by our input form. We have a section where the location form is handled.

You can add a similar section under it to read Longitude and Latitude:

```
try:
    latitude = int(request.form['latitude'])
    longitude = int(request.form['longitude'])
except:
    raise Exception("Unable to get longitude or latitude in request, something has
gone badly wrong!")
    exit(1)
```


3. Query weather API based on Longitude and Latitude

In the “**weather_api/api.py**” create a helper function **get_current_weather_at_coordinates()** that will help us get weather based on co-ordinates.

```
def get_current_weather_at_coordinates(self, longitude, latitude, unit)
    """
    Gets current weather status
    returns a dict of form
    {'status': 'Clouds', 'icon_url': 'http://openweathermap.org/img/w/04d.png', 'temp':
    14.32, 'temp_max': 15.0,
    'temp_min': 12.78, 'temp_kf': None}
    """
```

You can use the function **weather_at_coords()** within **pyowm** to achieve this. Ensure you handle errors in an efficient way – refer to the other methods for examples.

Ensure you retrieve **status**, **temperature** and **icon_url** from this object, and assemble a dictionary.

```
def weather_at_coords(self, lat, lon):
    """
    Queries the OWM Weather API for the currently observed weather at the
    specified geographic (eg: 51.503614, -0.107331).

    :param lat: the location's latitude, must be between -90.0 and 90.0
    :type lat: int/float
    :param lon: the location's longitude, must be between -180.0 and 180.0
    :type lon: int/float
    :returns: an *Observation* instance or ``None`` if no weather data is
        available
    :raises: *ParseResponseException* when OWM Weather API responses' data
        cannot be parsed or *APICallException* when OWM Weather API can not be
        reached
    """
```

NOTE: ensure to handle the **pyowm** client using **self** – examples of this can be seen within **get_current_weather_at_location()**.

4. Display weather information at coordinates

Within “**templates/index.html**”, the above dictionary returned by **get_current_weather_at_coordinates()** must be displayed in a tabular format.

In “**weather/main.py**” we need to utilize **get_current_weather_at_coordinates()** and create a new dictionary variable we are able to pass to **index.html**.

Create the **coord_weather** dictionary as follows, we must also pass this as an argument to our **render_template** function. This ensures the variables can be accessed in html rendering.

```
coord_weather = weather_api_wrapper.get_current_weather_at_coordinates(longitude,
latitude, options['temperature_unit'])

return render_template("index.html", weather=weather, coord_weather=coord_weather,
location=location, longitude=longitude, latitude=latitude)
```

Within “**templates/index.html**” you can use the following block to display the co-ordinate based weather info:

```
{% if coord_weather %}
    <h2>Current weather at Longitude {{ longitude }} and Latitude {{ latitude
}}</h2>
    <h3>Status: {{ coord_weather['status'] }} </h3>
    <table class="table table-striped table-bordered table-condensed">
        <tbody>
            <tr>
                <td>Temperature</td>
                <td><b>Average<b></td>
                <td><b>Max<b></td>
                <td><b>temp_min<b></td>
            </tr>
            <tr>
                <td></td>
                <td>{{ coord_weather['temp'] }}</td>
                <td>{{ coord_weather['temp_max'] }}</td>
                <td>{{ coord_weather['temp_min'] }}</td>
            </tr>
            <tr>
            </tr>
        </tbody>
    </table>
{% endif %}
```

Have a go at running the application, you should now be able to query weather information based on co-ordinates and location names!

VII. Documentation references

Flask - <https://flask.palletsprojects.com/en/1.1.x/>

Jinja - <https://jinja.palletsprojects.com/en/2.11.x/>

Pyowm module reference -

<https://buildmedia.readthedocs.org/media/pdf/pyowm/latest/pyowm.pdf>

OpenWeatherMap API reference - <https://openweathermap.org/current>