Course Online Openweathermap

Contents

1. INTRODUCTION TO PYOWM

1.1 INTRODUCTION

In this course we will be using the OpenWeatherMap API to get weather data. Let's dive right in!

OpenWeatherMap provides several ways to specify the location that you want weather data for. The following methods are all valid ways to specify the location:

City name

City ID

Lat/Lon Coordinates

Zip Code

We'll explore each of these methods in the next steps.

```
import os
API_KEY = os.environ['API_KEY']
```

1.2 CITY NAME

Open city_name.py in the editor. We will be using this file throughout this step. Don't worry about understanding every line of the code snippets in this lesson. We'll dive deeper into what exactly this code is doing in the next lesson.

When requesting the weather data, one way to specify what location you want the weather for is by specifying the city name. You can do this by calling the weather_at_place() method of a WeatherManager instance. Line 5 in city_name.py illustrates the use of this function:

obs = mgr.weather_at_place('San Francisco')

The weather_at_place() method supports specifying the country code as well. This can be useful to make sure you get the data for the city you want, since there could be a city with the same name in a different country. An example of specifying the country code for the United States is:

obs = mgr.weather_at_place('San Francisco, US')

Click the Run Code button to see what the temperature is in San Francisco!

```
import pyowm
```

```
from api_key import API_KEY

owm = pyowm.OWM(API_KEY)
mgr = owm.weather_manager()
obs = mgr.weather_at_place('San Francisco')
weather = obs.weather
temperature = weather.temperature(unit='fahrenheit')['temp']

print(f'The temperature is {temperature} degrees Fahrenheit.')
```

1.3 CITY ID

Open city_id.py in the editor. We will be using this file throughout this step.

Another way to specify the location is using the city ID. This is similar to using the city name, except that the method weather_at_id() is used instead of weather_at_place(), and it is passed the city ID instead of the city name:

obs = mgr.weather_at_id(5391959)

The list of city IDs can be downloaded as city.list.json.gz from here http://bulk.openweathermap.org/sample/.

```
import pyowm
from api_key import API_KEY

owm = pyowm.OWM(API_KEY)
mgr = owm.weather_manager()
obs = mgr.weather_at_id(5391959)
weather = obs.weather
temperature = weather.temperature(unit='fahrenheit')['temp']

print(f'The temperature is {temperature} degrees Fahrenheit.')
```

1.4 LAT/LON COORDINATES

Open lat_lon.py in the editor. We will be using this file throughout this step.

Latitude and longitude coordinates can also be used to specify the desired location. To do this, call the weather_at_coords() method and pass the latitude and longitude as arguments as seen on line 5:

obs = mgr.weather_at_coords(lat=37.774929, lon=-122.419418)

Here are the latitude and longitude coordinates for several cities. Use this chart to complete the instruction.

City	Latitude	Longitude
New York City, US	40.730610	-73.935242
Paris, France	48.853409	2.3488
London, UK	51.509865	-0.118092

```
import pyowm
from api_key import API_KEY

owm = pyowm.OWM(API_KEY)
mgr = owm.weather_manager()
obs = mgr.weather_at_coords(lat=37.774929, lon=-122.419418)
weather = obs.weather
temperature = weather.temperature(unit='fahrenheit')['temp']

print(f'The temperature is {temperature} degrees Fahrenheit.')
```

1.5 ZIP CODE

Open zip_code.py in the editor. We will be using this file throughout this step.

One final way to specify the location is by using the zip code and country. To do this, call the method weather_at_zip_code() and pass the zip code and country code as strings as seen on line 5:

obs = mgr.weather_at_zip_code('94016', 'US')

```
import pyowm
from api_key import API_KEY

owm = pyowm.OWM(API_KEY)
mgr = owm.weather_manager()
obs = mgr.weather_at_zip_code('94016', 'US')
weather = obs.weather
temperature = weather.temperature(unit='fahrenheit')['temp']

print(f'The temperature is {temperature} degrees Fahrenheit.')
```

Summary

All the supported cities along with their ID, country, and lat/lon coordinates can be found in the file named city.list.json.gz which can be downloaded at this URL: http://bulk.openweathermap.org/sample/.

2. GET CURRENT WEATHER DATA

OVERVIEW

The current weather data can be retrieved from OpenWeatherMap using the Observation module in PyOWM (Python OpenWeatherMap). We've already worked with code that gets the current weather. Now we'll explore that code to see how it works.

weather.py contains some sample code requesting the current weather for San Francisco, California. Let break that code down line-by-line to see what it's doing.

```
import pyowm
from api_key import API_KEY
owm = pyowm.OWM(API_KEY)
```

In the first three lines, we import the PyOWM library, which is called `pyowm`. Next we import the API key from the *api_key.py* file. Most APIs require you to provide some kind of authentication to use their service. For this course we have provided you with an API key to use. We then use this API key to authenticate with OpenWeatherMap by calling `pyowm.OWM()`.

```
mgr = owm.weather_manager()
```

To get the weather we first create a `WeatherManager` instance

```
obs = mgr.weather_at_place('San Francisco, US')
```

We explored this line pretty throughly in the last lesson. This method returns a PyOWM Observation instance.

```
weather = obs.weather
```

The `Observation` instance contains a Weather object. This line gets that `Weather` object.

```
temperature = weather.temperature(unit='fahrenheit')['temp']
print(f'The temperature in San Francisco, California is {temperature} degrees Fahrenheit.')
```

There are multiple properties of a `Weather` object. It contains the temperature, rain, snow, wind, sunrise, sunset, visibility, and more! This line gets the temperature in Fahrenheit. Finally, the last line prints this temperature. Go ahead and click **Run Code** to see the temperature.

Sweet! Now we know how to get the temperature! In the rest of this lesson we'll learn some more about getting the temperature as well as how to access the additional properties in a `Weather` object.

2.1 GETTING THE TEMPERATURE

Open temperature.py in the editor. We will be using this file throughout this step.

In the previous step we took a look at how to get the temperature in Fahrenheit. In this step we'll take a look at other formats we can get the temperature in.

The temperature method returns a Python dictionary, an example of which is below:

```
{'temp': 290.42, 'temp_max': 294.26, 'temp_min': 287.04, 'feels_like': 285.12, 'temp_kf': None}
```

- temp: The current temperature.
- temp_max: The temperature for a large city may vary throughout the city. This is the calculated temperature at the hottest location of the city.
- temp_min: This is the calculated temperature at the coldest location of the city.
- feels_like: This value is provided if available.
- temp_kf: Value used internally by OpenWeatherMap

The default units for the temperature are Kelvin. Fahrenheit and Celsius are also supported by passing a parameter to the function.

The temperature units can be specified by including the desired unit type as a parameter. The following parameter values are supported:

- 'kelvin'
- 'celsius'
- 'fahrenheit'

For example, to get the temperature dictionary in Fahrenheit, you could write weather.temperature('fahrenheit').You may also specify the parameter name, like this weather.temperature(unit='fahrenheit')

The 'temp' key contains the average temperature in the city, while 'temp_min' and 'temp_max' contain data for the coldest and warmest areas in the city respectively. Most of the time the temperature you are looking for will be the value for key 'temp'. Retrieve this value and print it for the following two instructions.

2.2 WEATHER INFO

Open *weather_info.py* in the editor. We will be using this file throughout this step.

PyOWM gives you access to other weather information in addition to the temperature. Some of the information you can get is:

- wind - wind()

- clouds - clouds
- humidity - humidity

The values in the clouds and humidity properties are of type int. The values represent the percentage of cloud cover and the humidity percentage respectively. For example, to print the humidity after obtaining a Weather object, you could write:

```
humidity = weather.humidity
print(f'The current humidity is {humidity}%')
```

The data returned by the wind() method is in a dictionary. To get the data from the dictionary, you have to know what keys are in the dictionary. Here's an example of values that could be returned:

```
# wind - direction (degrees), and speed  (meters/second)
{'deg': 270, 'speed': 2.1}
```

wind() accepts an optional unit parameter. It defaults to 'meters_sec', but also supports 'miles_hour'.

```
import pyowm
from api_key import API_KEY

owm = pyowm.OWM(API_KEY)
mgr = owm.weather_manager()
obs = mgr.weather_at_place('Los Angeles, US')
weather = obs.weather

# Write your code here
temperature = weather.temperature(unit='fahrenheit')['temp']
print(f'The temperature in San Francisco, California is {temperature} degrees Fah
renheit.')
```

2.3 GET THE WEATHER STATUS

Open *status.py* in the editor. We will be using this file throughout this step.

Sometimes you may not care about exact numbers and you just want to get a short description of the weather. PyOWM has just the properties you need!

- status - This property contains a short one-word description of the weather.
- detailed_status - This property contains a bit more detail with a one or two word description.

Both properties are available on a Weather object. For example, see the following snippet:

```
weather = obs.weather
print(weather.status)
```

```
import pyowm
from api_key import API_KEY

owm = pyowm.OWM(API_KEY)
mgr = owm.weather_manager()
obs = mgr.weather_at_place('Seattle, US')
weather = obs.weather

# Write your code here
weather = obs.weather
print(weather.status)
```

2.4 SUNRISE AND SUNSET TIMES

Open *sun.py* in the editor. We will be using this file throughout this step.

Knowing what time the sun rises and sets is great information to have when planning your day! Let's see how we can get these times using PyOWM.

- sunrise_time() - Returns the GMT time of sunrise
- sunset_time() - Returns the GMT time of sunset

For example, checkout the following snippet:

```
print(weather.sunrise_time())
```

You'll notice that the sunrise time looks a little cryptic. That's because the value returned is in unix time. PyOWM allows you to add a parameter to specify ISO time as well, which is more easily readable. For example, checkout the snippet below:

```
print(weather.sunrise_time(timeformat='iso'))
```

Bonus: Timezones

Keep in mind that the times are for the GMT timezone, so they may still look a little strange. Boston is in the EST timezone, so converting the returned time from GMT to EST will make it look more reasonable. To do this, we can use the following code snippet:

```
# Get the sunset time in the GMT timezone in the unix time format
sunset_unix = weather.sunset_time()

# Create `pytz` timezone objects
eastern = pytz.timezone('US/Eastern')
gmt = pytz.timezone('GMT')

# Create a UTC `datetime` object from the unix timestamp
sunset = datetime.utcfromtimestamp(sunset_unix)

# Make the `datetime` object timezone aware so Python will
# know how to convert it to a different timezone
sunset = gmt.localize(sunset)

# Convert time to Eastern time and print it
sunset_eastern = sunset.astimezone(eastern)
print(sunset_eastern)
```

Note that the above snippet uses the `pytz` module. `pytz` is a Python module that simplifies working with timezones. Also note that both UTC and GMT are used in the above snippet. UTC and GMT are different, but share the same time.

To learn more about working with timezones using `pytz`, checkout the docs.

```
import pyowm
import pytz
from datetime import datetime
from api_key import API_KEY

owm = pyowm.OWM(API_KEY)
mgr = owm.weather_manager()
obs = mgr.weather_at_place('Boston, US')
weather = obs.weather

# Write your code here
```

To see every method that PyOWM supports, read the docs on the Weather module.

3. WEATHER FORECASTS

Up to this point we have been getting information about the current weather. PyOWM also provides a way to retrieve weather forecasts.

The OpenWeatherMap free tier gives you access to 5 day forecasts. The forecasts contain the weather data in three-hour intervals.

The forecast can be retrieved similar to how we retrieved the weather. The methods for retrieving the forecast are:

- `forecast_at_place('Los Angeles, US', '3h')`
- `forecast_at_id(5391959, '3h')`
- `forecast_at_coords(lat=37.774929, lon=-122.419418, interval='3h')`

Forecasts are useful if you want to know what the weather conditions will be like throughout the day/week. For example, If you want to find out if it will be raining at noon tomorrow, the three-hour forecast will provide you the data you need!

3.1 THREE HOUR FORECAST

Open *three_hours_forecast.py* in the editor. We'll be using this file throughout this step.

In this step we will get the forecast using the `forecast_at_place()` method. We start as usual on line 6 by creating an `OWM` object:

```
owm = pyowm.OWM(API_KEY)
```

Next on line 9 we get a `WeatherManager` instance.

```
mgr = owm.weather_manager()
```

Then on line 10 we get the three-hours forecast. This will always retrieve a 5 day forecast:

```
forecaster = mgr.forecast_at_place('Los Angeles, US', '3h')
```

On line 11 we get the forecast:

```
forecast = forecaster.forecast
```

At this point you might be wondering what the difference is between the `forecaster` object and the `forecast` object. The `forecast_at_place()` method returns a PyOWM `Forecaster` object and the `forecast` property contains a `Forecast` object.

The Forecast class gives you access to the actual values in the forecast, such as the temperature, amount of rainfall, etc. while the Forecaster class provides a high level interface to the forecast so you can check whether or not there are specific weather conditions (such as rain or clouds) in the forecast. We'll explore some of the methods for the Forecaster class in later steps.

On line 12 we get a list of Weather objects:

```
weather_list = forecast.weathers
```

Finally on lines 15-17 we loop through each Weather object and print some information about the weather:

```
for weather in weather_list:
    temp = weather.get_temperature(unit='fahrenheit')['temp']
    print(weather.reference_time('iso'), f'Temperature: {temp}{degree_sign}F')
```

```
import pyowm
from api_key import API_KEY

degree_sign= u'\N{DEGREE SIGN}'

owm = pyowm.OWM(API_KEY)

# Three Hours Forecast
mgr = owm.weather_manager()
forecaster = mgr.forecast_at_place('Los Angeles, US', '3h')
forecast = forecaster.forecast
weather_list = forecast.weathers

print('Three hours forecast (Times are in GMT):')
for weather in weather_list:
    temp = weather.temperature(unit='fahrenheit')['temp']
    print(weather.reference_time('iso'), f'Temperature: {temp}{degree_sign}F')
```

3.2 GET WEATHER AT A SPECIFIC TIME

Open *weather_at_time.py* in the editor. We'll be using this file throughout this step.

In the previous step, after retrieving the forecast, we used the weathers property to get a list of Weather objects in the forecast. The Forecaster class provides another useful method called get_weather_at() which retrieves a Weather object that is closest to a specified time.

For example, lines 11-14 in *weather_at_time.py* show how to get the weather 12 hours from the current time.

PyOWM also includes a timestamps module to make it easier to get the weather at a specific time. Line 17 shows the use of timestamps to get the time tomorrow at noon GMT time:

```
time = timestamps.tomorrow(12,0)
```

Note: Keep in mind that all times are in GMT time when working with PyOWM. You may convert the times to another timezone using the method shown in the previous lesson.

```
import pyowm
from pyowm.utils import timestamps
from datetime import timedelta, datetime
from api_key import API_KEY

degree_sign = u'\N{DEGREE SIGN}'

owm = pyowm.OWM(API_KEY)
mgr = owm.weather_manager()
forecaster = mgr.forecast_at_place('Los Angeles, US', '3h')

time = datetime.now() + timedelta(days=0, hours=12)
weather = forecaster.get_weather_at(time)
temperature = weather.temperature(unit='fahrenheit')['temp']
print(f'The temperature at {time.strftime("%Y-%m-
%d %H:%M:%S")} is {temperature}{degree_sign}F')

time = timestamps.tomorrow(12,0)
weather = forecaster.get_weather_at(time)
temperature = weather.temperature(unit='fahrenheit')['temp']
print(f'The temperature at {time} is {temperature}{degree_sign}F')

# Write your code here
time = timestamps.tomorrow(12,0)
```

3.3 WILL HAVE

Open *will_have.py* in the editor. We'll be using this file throughout this step.

Python Open Weather Map provides methods to check for certain weather conditions in a forecast. The following methods are available:

- will_have_rain()
- will_have_clear()
- will_have_fog()
- will_have_clouds()
- will_have_snow()
- will_have_storm()
- will_have_tornado()
- will_have_hurricane()

For example, the following snippet checks if there is rain in this week's forecast for LA:

```
forecaster = mgr.forecast_at_place('Los Angeles, US', '3h')
print(forecaster.will_have_rain())
```

```
import pyowm
from api_key import API_KEY

owm = pyowm.OWM(API_KEY)
mgr = owm.weather_manager()
forecaster = mgr.forecast_at_place('Los Angeles, US', '3h')

# Write your code here
print(forecaster.will_have_rain())
```

3.4 WILL BE

Open *will_be.py* in the editor. We'll be using this file throughout this step.

PyOWM includes a set of methods on a Forecaster object to check if a particular weather condition exists at a specified time. The following methods are available:

- will_be_rainy_at()
- will_be_clear_at()
- will_be_foggy_at()
- will_be_cloudy_at()
- will_be_snowy_at()

- will_be_stormy_at()
- will_be_tornado_at()
- will_be_hurricane_at()

For example, the following snippet checks if it will be stormy at 12:00 GMT on August 15th, 2018 in LA:

```
forecaster = mgr.forecast_at_place('Los Angeles, US', '3h')
print(forecaster.will_be_stormy_at(datetime(2018, 8, 15, 12, 0)))
```

The date specified must be a date that is included in the five-day forecast. If the date is not in the forecast it will cause an error.

Use timestamps.tomorrow() as explained in the **Get Weather At a Specific Time** step to complete the following instructions.

```
import pyowm
from pyowm.utils import timestamps
from datetime import timedelta, datetime
from api_key import API_KEY

owm = pyowm.OWM(API_KEY)
mgr = owm.weather_manager()
forecaster = mgr.forecast_at_place('Los Angeles, US', '3h')

# Write your code here
print(forecaster.will_be_stormy_at(datetime(2018, 8, 15, 12, 0)))
```

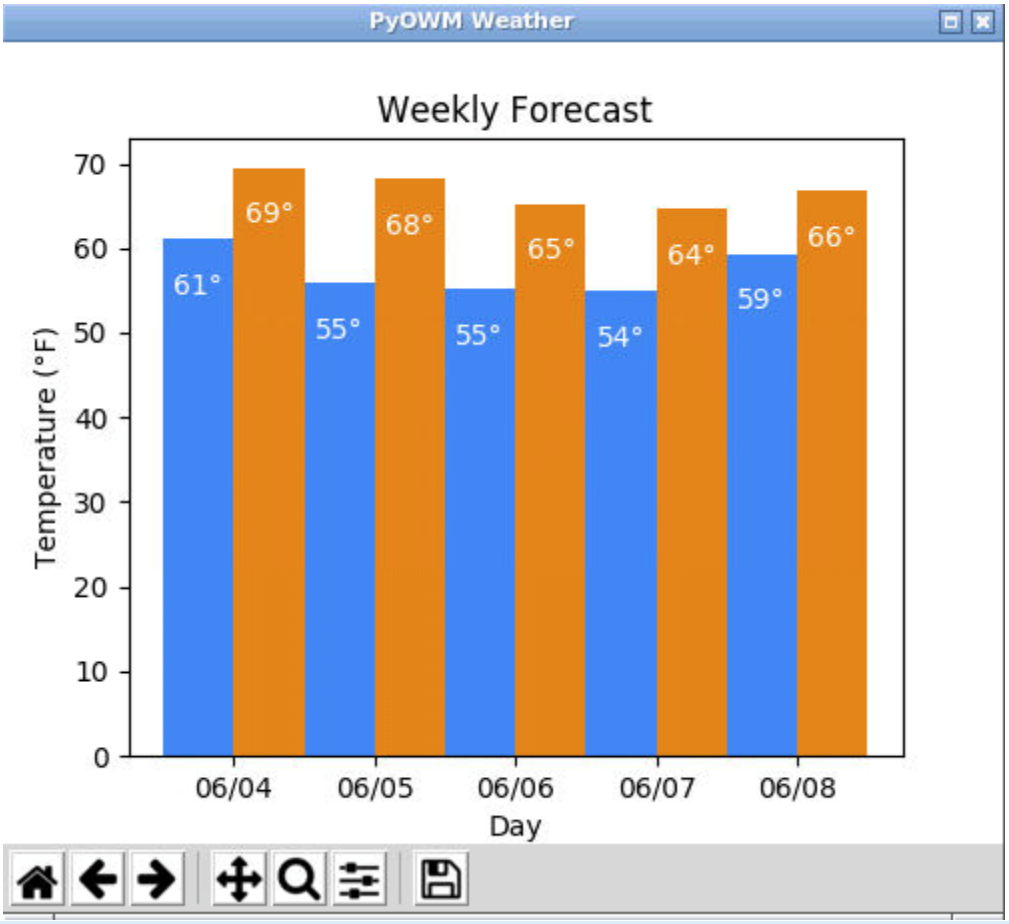
3.5 WEATHER FORECAST SUMMARY

In this lesson we focused on learning about the Forecaster class and the Forecast class. We explored some, but not all of the methods for each class. To view all the methods for each class, check out the documentation below.

- Forecaster class - View documentation
- Forecast class - View documentation

4. WEATHER GUI

For the final project in this course we will be building a GUI to display the temperatures for the next five days. If you've never built a GUI with Python, don't worry! We'll be going through step by step how to build it. Well be using Matplotlib to plot the weather data. Matplotlib uses Tkinter behind the scenes to display the interactive GUI. The final program will look like this:



You can click-and-drag the sides of the panes to resize them. It is recommended that you resize the pane that displays the GUI to make it bigger.

4.1 CREATE AN EMPTY PLOT

We will start creating the GUI by first creating an empty plot. Scroll to the bottom of *main.py* and uncomment the line that calls init_plot().

Go to the init_plot() function in *main.py*. In it, remove pass and add the following line:

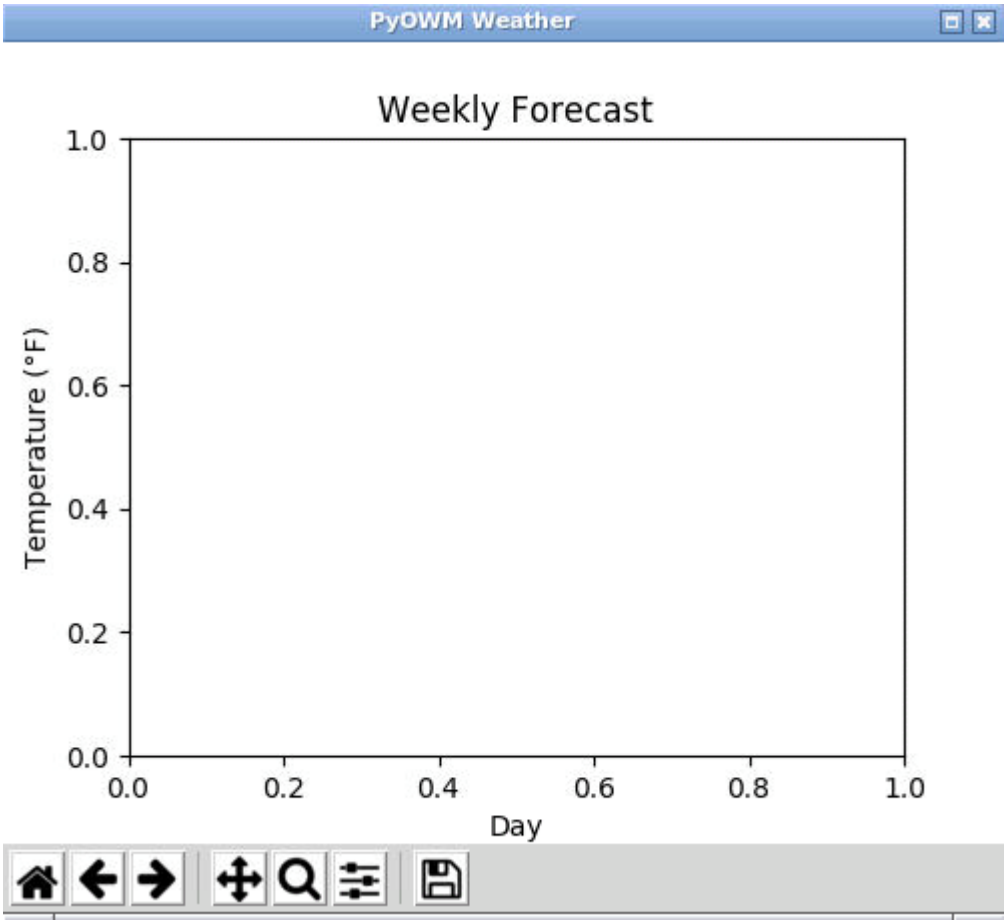
```
plt.figure('PyOWM Weather', figsize=(5,4))
```

This line creates an empty window, where we will create our plot. The name of the figure, which will appear in the titlebar, is **PyOWM Weather**. The figsize argument specifies the width and height of the plot in inches.

Next we'll add labels and a title to the plot. Add the following lines to the init_plot() function.

```
plt.xlabel('Day')
plt.ylabel(f'Temperature ({degree_sign}F)' )
plt.title('Weekly Forecast')
```

At this point your GUI should look like the following image. If it does, nice work! Head on to the next step! If not, see if you can spot whats wrong before continuing.




```
from matplotlib import pyplot as plt
from matplotlib import dates
from pyowm_helper import get_temperature

degree_sign= u'\N{DEGREE SIGN}'
```

```
def init_plot():
    plt.figure('PyOWM Weather', figsize=(5,4))
    plt.xlabel('Day')
    plt.ylabel(f'Temperature ({degree_sign}F)' )
    plt.title('Weekly Forecast')



if __name__ == '__main__':
    #init_plot()
    #days, temp_min, temp_max = get_temperature()
    #bar_min, bar_max = plot_temperatures(days, temp_min, temp_max)
    #label_xaxis(days)
    #write_temperatures_on_bar_chart(bar_min, bar_max)

    plt.show()
```

4.2 GET THE TEMPERATURE FORECAST

Next open *pyowm_helper.py*. Here we will define get_temperature(). This function will use PyOWM to get the weather forecast over the next five days. Start by removing pass from the get_temperature() function.

We will start by initializing some empty arrays. Type the following as the first four lines in the get_temperature() function:

```
days = []
dates = []
temp_min = []
temp_max = []
```

Next we will get the weather forecast for the next five days. Feel free to choose any city you like. For example, getting the forecast for New York would look like this:

```
forecaster = mgr.forecast_at_place('New York, US', '3h')
```

Remember, forecast_at_place() returns a Forecaster object, so on the next line get the forecast object by typing:

```
forecast = forecaster.forecast
```

Next we're going to loop through each weather object and store the high and low temperature for each day. This is slightly more complex than it sounds. Remember the three_hours_forecast() doesn't directly tell us the hotest and coldest temperatures for each day. Therefore we will need to calculate these values inside the for loop.

Type the code below as the next lines in the get_temperature() function, then we'll take a look at what exactly each line is doing.

```
for weather in forecast:
    day = datetime.utcfromtimestamp(weather.reference_time())
    date = day.date()
    if date not in dates:
        dates.append(date)
        temp_min.append(None)
        temp_max.append(None)
        days.append(date)
    temperature = weather.temperature('fahrenheit')['temp']
    if not temp_min[-1] or temperature < temp_min[-1]:
        temp_min[-1] = temperature
    if not temp_max[-1] or temperature > temp_max[-1]:
        temp_max[-1] = temperature
```

Alright, lets break this code down a bit more. First we're getting the time of the weather forecast and converting that to a Datetime object. Next we store just the date in the date object, getting rid of the hours, minutes, and seconds. Then we check if we've seen that date before.

If we haven't seen the date before, we record that we've seen that date and append new values to the temperature and days lists. We need to make a list of days so that later we can show on the plot what days the high and low temperatures are for. Also, appending None to the temperature lists tells us that we just started reading temperatures for a new day.

Next we get the temperature using the same method we used in the last lesson. Then we check if the last temperature in the temp_min list is None (meaning this is the first temperature we've seen for that day) or if the temperature is less than it. If so, we've found the lowest temperature we've seen so far for that day, so we update the last element of the list with the new lower temperature. We then do the same thing to record the highest tempearture.

When we're finished, the days list holds a Datetime object for each day in the forecast, the temp_min list holds the lowest temperature for each day, and the temp_max list holds the highest tempearture for each day. Now we'll finish up by returning these lists.

```
return(days, temp_min, temp_max)
```

Verify

Before we move on, lets make sure we defined this method correctly. Right before the return statement, add a print statement to print the lists so you can see their contents.

```
print(days, temp_min, temp_max)
```

Now click **Run Code**. The temp_max and temp_min arrays should print the high and low temperatures. The days array will print the days as Datetime objects. Remember though, get_reference_time() returns the times in GMT time. In the next section we will fix this so the dates are in the correct timezone for the city you chose.

```
import os
import pyowm
from datetime import datetime
from timezone_conversion import gmt_to_eastern

API_KEY = os.environ['API_KEY']

owm = pyowm.OWM(API_KEY)
mgr = owm.weather_manager()

def get_temperature():
    days = []
    dates = []
    temp_min = []
    temp_max = []
    forecaster = mgr.forecast_at_place('New York, US', '3h')
    forecast = forecaster.forecast

    for weather in forecast:
    day = datetime.utcfromtimestamp(weather.reference_time())
    date = day.date()
    if date not in dates:
        dates.append(date)
        temp_min.append(None)
        temp_max.append(None)
        days.append(date)
    temperature = weather.temperature('fahrenheit')['temp']
    if not temp_min[-1] or temperature < temp_min[-1]:
```

```
        temp_min[-1] = temperature
    if not temp_max[-1] or temperature > temp_max[-1]:
        temp_max[-1] = temperature
    print(days, temp_min, temp_max)
    return(days, temp_min, temp_max)


if __name__ == '__main__':
    pass
```

4.3 TIMEZONE CONVERSION

Open *timezone_conversion.py*. Here there is a function named gmt_to_eastern. This step will show you how to convert the time from GMT time to Eastern time. If the city you chose is in a different timezone such as Pacific, feel free to rename this function to gmt_to_pacific.

This step uses the same method for timezone conversion shown previously in this course, so it should look familiar. Here each line will be explained in a bit more detail than it was explained earlier in the course.

Remove pass from the first line of the function and add the following to create an object representing the timezone you chose.

```
eastern = pytz.timezone('US/Eastern')
```

If you are unsure what string to use for the timezone you chose, click the **Run Code** button. This will run the code at the end of the file, which prints the strings for each supported timezone.

Next create a GMT timezone object.

```
gmt = pytz.timezone('GMT')
```

Next we'll create a datetime object from the unix time parameter.

```
date = datetime.utcfromtimestamp(unix_gmt)
```

Now we need to use the astimezone method provided by the datetime module to convert the date to our chosen timezone. The only problem is that right now the date object is not associated with any timezone. The astimezone method needs to know the current timezone before it can convert the date to a different timezone. pytz contains a localize method that we can use to associate the date with the GMT timezone.

```
date = gmt.localize(date)
```

Finally we can convert the date to our chosen timezone and return that date.

```
eastern_time = date.astimezone(eastern)
return eastern_time
```

Update *pyowm_helper.py*

Now let's update the code to use the timezone conversion function. Head back on over to *pyowm_helper.py* and replace the line:

```
day = datetime.utcfromtimestamp(weather.get_reference_time())
```

with

```
day = gmt_to_eastern(weather.reference_time())
```

If you chose a different timezone and renamed the function, use that function name here. Also, change `gmt_to_eastern` to the name of your function at the top of the file where it is imported.

Now with *pyowm_helper.py* still open, and the print statement we added in the last step still present, click **Run Code**. Ensure that the program runs without error and and that the printout looks correct (it should look similar to the printout in the last step). If the printout looks correct, remove the print statement `print(days, temp_min, temp_max)` and continue to the next step.

```
import pytz
from datetime import datetime

def gmt_to_eastern(unix_gmt):
    pass

if __name__ == '__main__':
    for timezone in pytz.all_timezones:
        print(timezone)
```

4.4 PLOT THE TEMPERATURES

Now that we have a function to retrieve the weather, we're ready for the fun part, plotting the temperature!

Open *main.py*. Scroll to the bottom and uncomment the following two lines:

```
days, temp_min, temp_max = get_temperature()
```

and

```
bar_min, bar_max = plot_temperatures(days, temp_min, temp_max)
```

Then scroll back up to the definition for the `plot_temperatures()` function and remove `pass` from the definition.

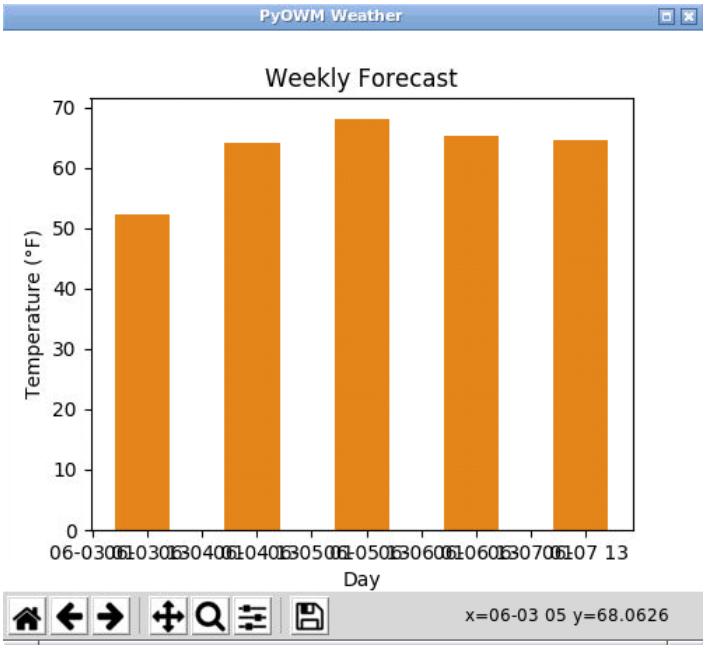
Add the following code in the function:

```
bar_min = plt.bar(days, temp_min, width=0.5, color='#4286f4')
bar_max = plt.bar(days, temp_max, width=0.5, color='#e58510')
return (bar_min, bar_max)
```

This creates two bar charts on the same plot, one for the low temperatures and one for the high temperatures for each day. The days are plotted on the x-axis and the temperatures are plotted on the y-axis. The width of the bars is 0.5 inches, and the color is provided as a hexadecimal value (blue for the low temperature and orange for the high temperature).

Go ahead and run the program to see the plot. When you do, you'll notice that it doesn't quite look right. For one, only the second bar plot for the high temperatures seems to be plotted. Also, the values on the x-axis are a mess and are completely unreadable. Let's fix this.

The reason it appears that only the second bar chart was plotted is that it was drawn over the first plot. We need to offset the two plots so that the bars don't cover each other.



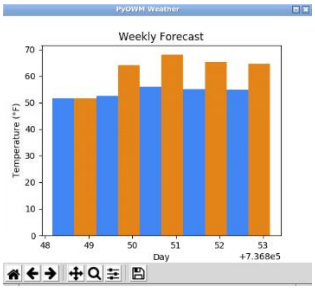
To offset the plots we will use the date2num() method from Matplotlib which is documented here. This method accepts a list of datetime objects and converts them to Matplotlib dates. This gives us numbers on the x-axis instead of datetime objects so that we can calculate offsets for each chart. Add the following line as the first line in the plot_temperatures() function.

```
days = dates.date2num(days)
```

The calculation is straight forward. Subtract **.25** from day in the low temperatures plot, and add **.25** to day in the high temperatures plot. The lines that create the plot should now look like this:

```
bar_min = plt.bar(days-.25, temp_min, width=0.5, color='#4286f4')
bar_max = plt.bar(days+.25, temp_max, width=0.5, color='#e58510')
```

Now run the program and you should see both the high and low temperatures plotted similar to the image below!



Notice that the x-axis labels still don't look right. That is because the values plotted are now Matplotlib dates instead of datetime objects. The fix is pretty straight forward though and we'll tackle that in the next step!

Optional

Depending on when you run the program, it may include today's date and show temperatures for six days. To remove today's temperature from the plot so that only the next five days are plotted, use the following code:

First add import time at the top of *pyowm_helper.py*. Then modify the beginning of the for loop in *pyowm_helper.py* to look as follows (if you chose a different name for your timezone coversion function, use that name here instead of gmt_to_eastern):

```
for weather in forecast:
    day = gmt_to_eastern(weather.reference_time())
    date = day.date()
    if (date == gmt_to_eastern(time.time()).date()):
        continue
```

4.5 LABEL THE X-AXIS

Uncomment the line at the bottom of *main.py* that calls label_xaxis(days). Then scroll up to the definition of the label_xaxis() function.

Start by removeing pass and adding the following line:

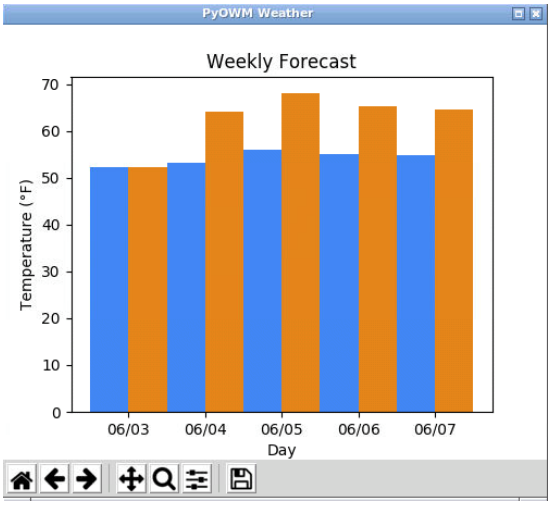
```
plt.xticks(days)
```

This takes the list of datetime objects that were passed in as a parameter and uses them for the x-axis labels. This is a step in the right direction, but the dates printed on the x-axis are still way too long. We will fix this by specifying a format to display the dates in. Add the following code to the label_xaxis() function:

```
axes = plt.gca()
xaxis_format = dates.DateFormatter('%m/%d')
axes.xaxis.set_major_formatter(xaxis_format)
```

This code starts by calling the gca() method to get the axes of the plot. Next the DateFormatter() method from Matplotlib is used to specify a format of the month and day separated by a slash. Finally that date format is set as the format for the x-axis.

Go ahead and run the code, the graph should look much better!



4.6 ADD THE TEMPERATURES AS LABELS ON THE BAR CHART

Congratulations on making it this far! At this point the plot is showing the forecast for the next five days. Now we're going to add one final feature to make it look even better!

Start by scrolling down to the bottom of *main.py* and uncommenting the line:

```
write_temperatures_on_bar_chart(bar_min, bar_max)
```

Then scroll up to the function definition and remove `pass`.

The final feature is to display the temperature at the top of each bar in the chart. To do this we will start by creating some variables. Add the following code inside `write_temperatures_on_bar_chart()`

```
axes = plt.gca()
y_axis_max = axes.get_ylim()[1]
label_offset = y_axis_max * .1
```

First we use the `gca()` to get the plot axes. Then we get the maximum value of the y-axis and multiple it by **.1**. Multiplying by **.1** gives us a small offset that we will use to write the label just below the top of each bar.

Next we will add the labels in a `for` loop:

```
# Write the temperatures on the chart
for bar_chart in [bar_min, bar_max]:
    for index, bar in enumerate(bar_chart):
        height = bar.get_height()
        xpos = bar.get_x() + bar.get_width()/2.0
        ypos = height - label_offset
```

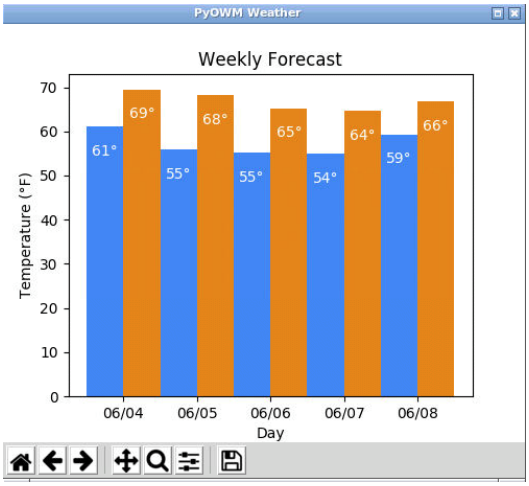
```
    label_text = str(int(height)) + degree_sign
    plt.text(xpos, ypos, label_text,
             horizontalalignment='center',
             verticalalignment='bottom',
             color='white')
```

The outer `for` loop loops through each bar chart. The inner `for` loop loops through each bar of each bar chart. For each bar, we get the bar height. Then we calculate the horizontal center of the bar and use that as the x-coordinate of the label. Next we calculate the y-coordinate of the label by subtracting the `label_offset` from the bar's height. This positions the label right below the top of the bar.

Next we set the label text equal to the height of the bar (the temperature) and add a degree symbol after it. Finally we create the label on the chart, centering it horizontally, setting the vertical alignment to **bottom**, and setting the text color to **white**.

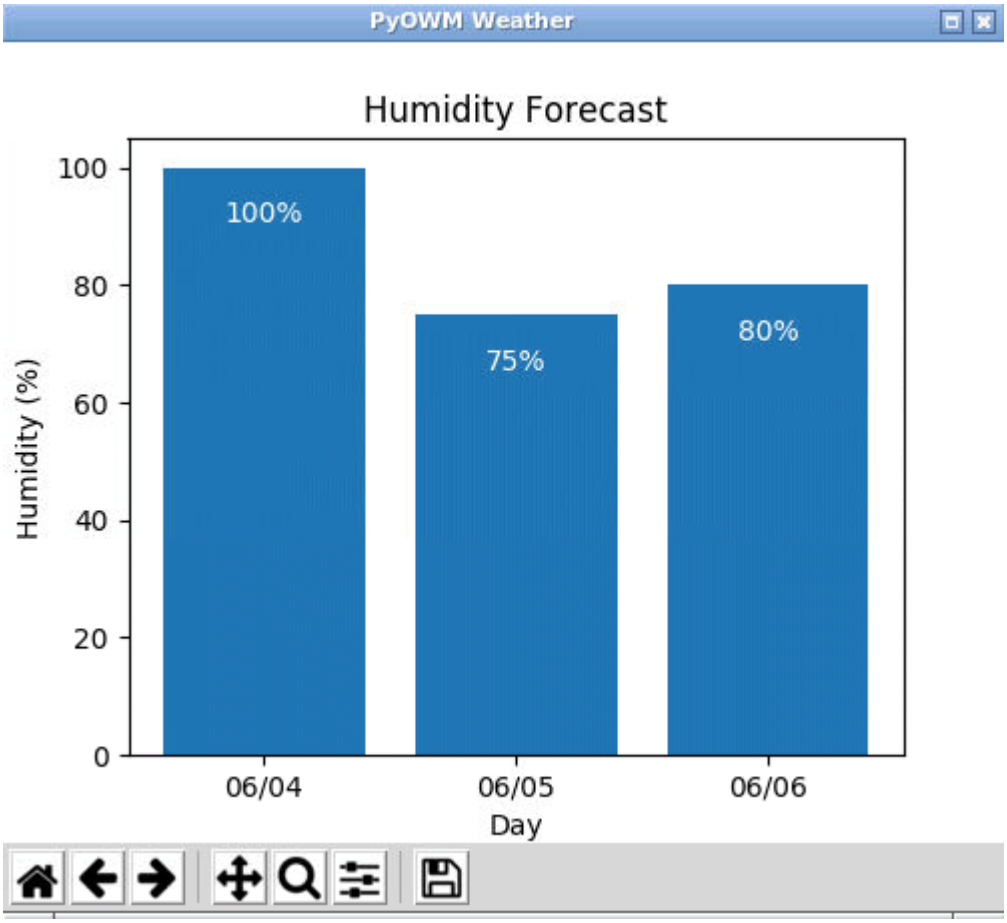
Note that the vertical alignment of **bottom** moves the text slightly higher on the chart, rather than lower.

If everything is working correctly, your plot should look similar to the image below!



5. WEATHER GUI CHALLENGE

Welcome to the final challenge for the PyOWM course. Your goal is to produce a weather GUI that shows the three-day humidity forecast for a city of your choice. The final GUI should look similar to the image below:



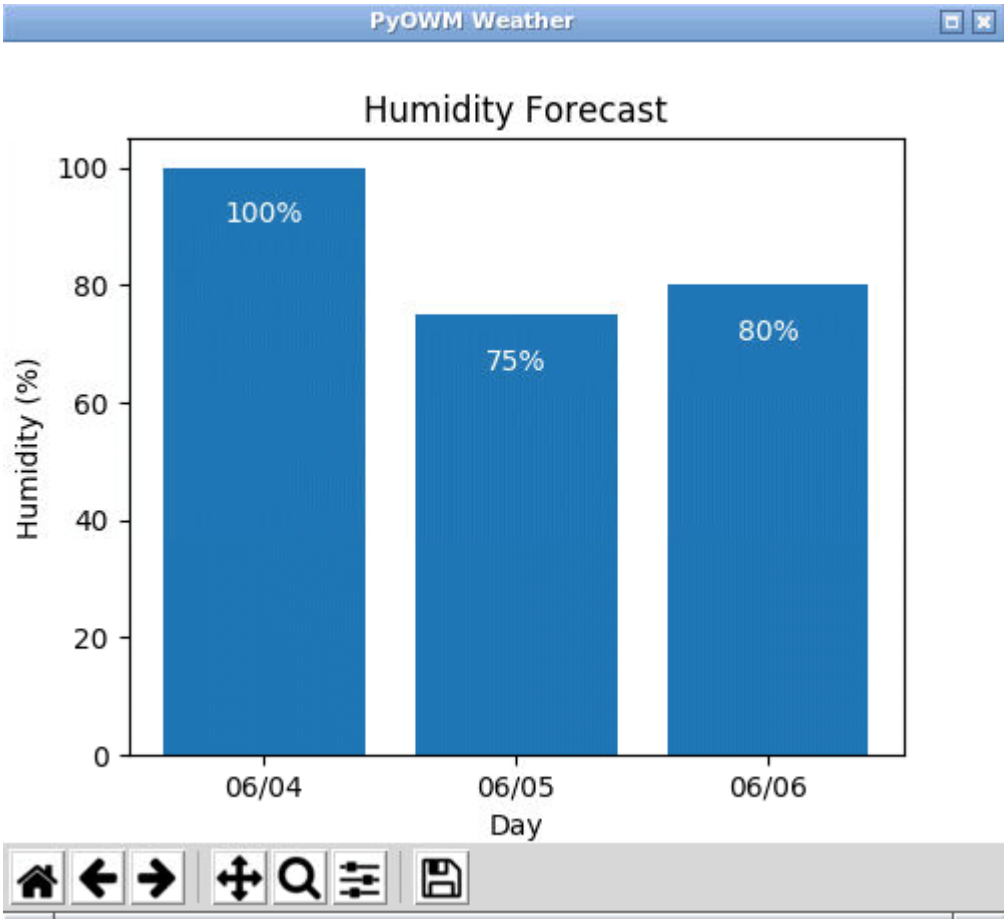
The starter code you have been given is the solution code for the previous project. Modify this code as necessary to produce the desired result.

The following steps contain hints to help guide you, but only use them if you need them.

5.1 HINT #1

Start by updating the values for the ylabel and title in the `init_plot()` method.

The final GUI is shown below for your reference.



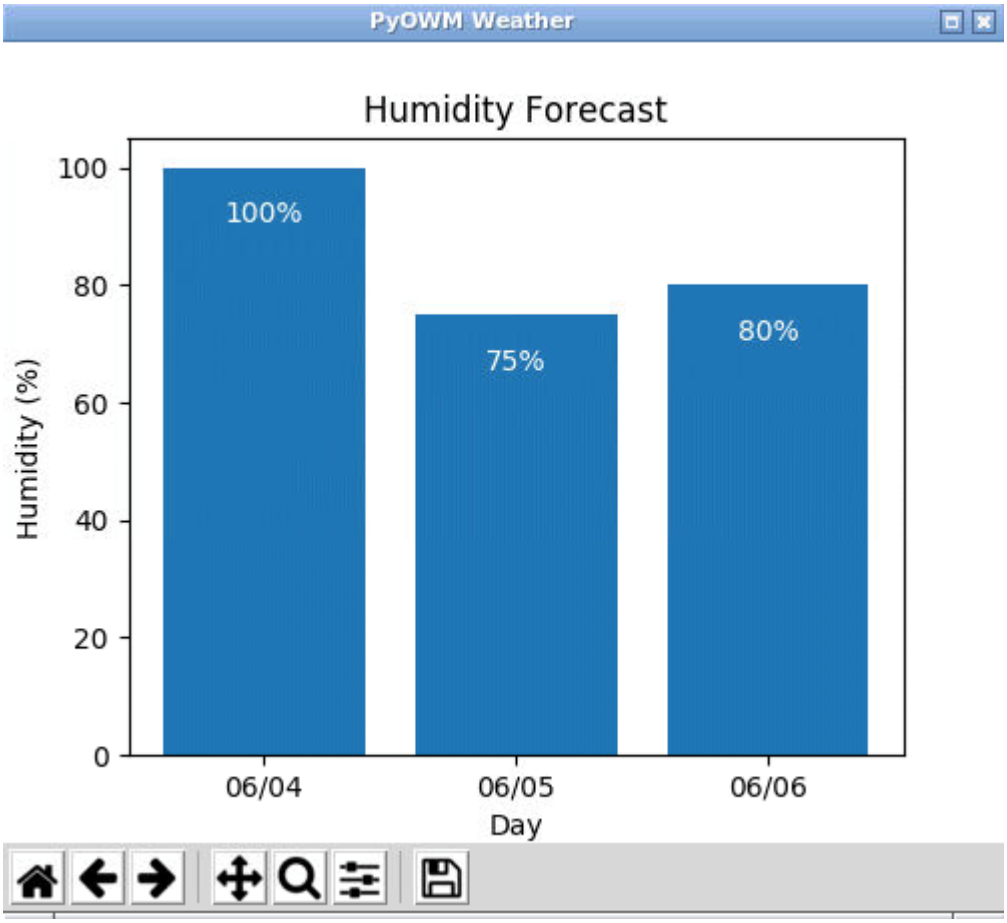
5.2 HINT #2

Change the code so that it only plots one bar chart. The following functions need to be changed:

- plot_temperatures()
- write_temperatures_on_bar_chart()
    - The outer for loop can be removed, since there will only be one bar chart.

It may be easiest to start by removing one of the temperature bar charts and leaving the other chart as is. Once the program runs with only one bart chart, then you can convert that bar chart to plot the humidity rather than the temperature.

The final GUI is shown below for your reference.

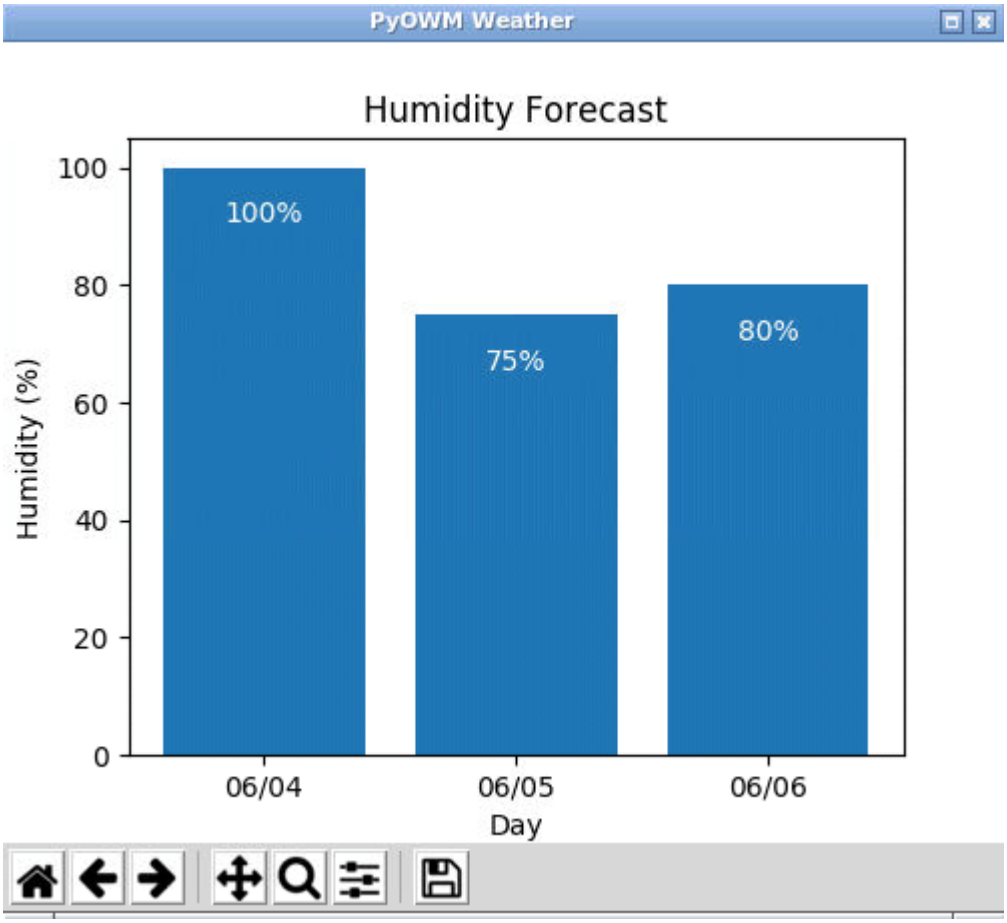


5.3 HINT #3

In *pyowm_helper.py*, check if `if len(days) >= 3` right inside the `if date not in dates:` statement, indicating the temperatures have been recorded for three full days. If so, return the three lists (and remove the return statement from the end of the function). Leave the rest of the logic that plots temperature data for now.

Style the chart how it is shown in the image (reproduced below). You can do this by removing the **.25** offset in `plot_temperatures()` as well as by removing the call to `date2num()`. In addition, the `width` and the `color` arguments can be removed from the `plt.bar()` function call.

Also replace the degree symbol with a percent symbol.



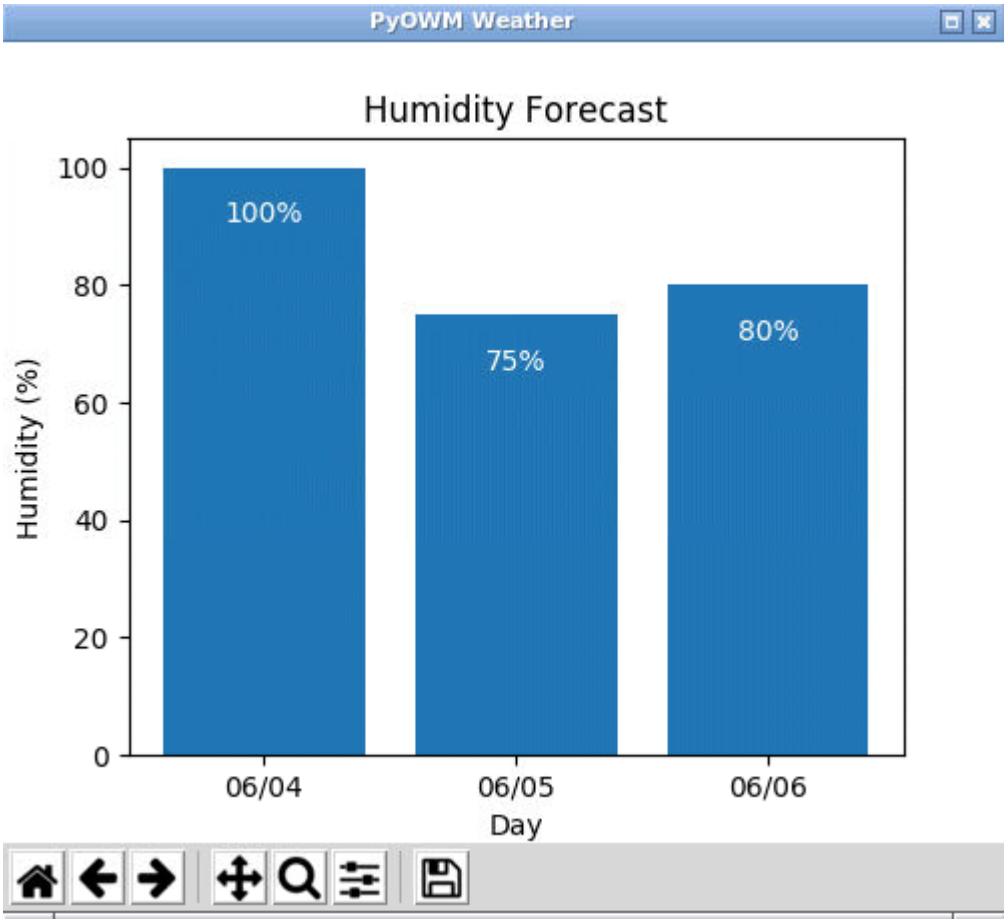
5.4 HINT #4

Change the get_temperature() function so that it retrieves the humidity instead of the temperature. This part is a little tricky and there are multiple ways to do it. The next and final hint has more guidance on this if you need it.

In get_temperature(), replace the two temperature lists with one list named humidity. The days list can be left as is. Rename this function to get_humidity() and update its name in all necessary files.

In *main.py*, the values returned by get_humidity() and the arguments to plot_temperatures() will need to be updated to account for one humidity list instead of two temperature lists.

The final GUI is shown below for your reference.



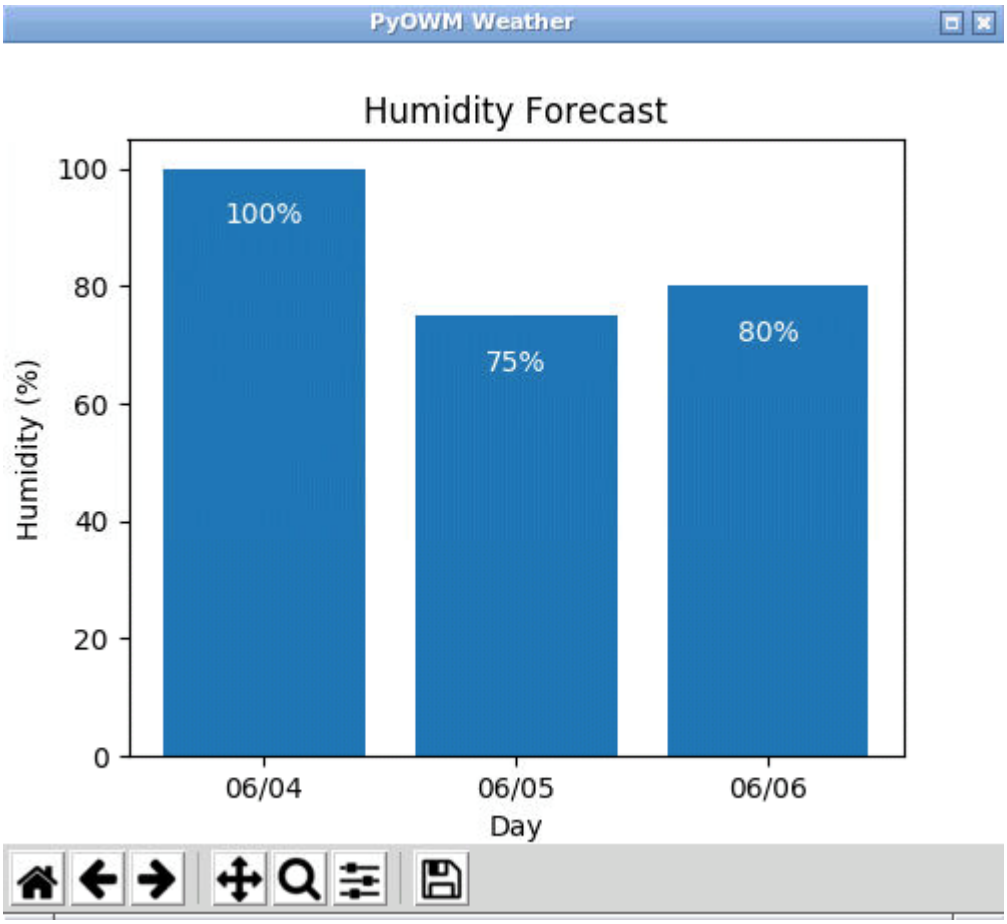
5.5 HINT #5

Here is one possible version of the `for` loop in the `get_humidity()` function. This stores the humidity around noon each day (Since we're converting the time to a different timezone, we may not have a weather sample from exactly noon. It could be from any of the three hours around noon). You could also take an average of the humidity each hour of the day and use that as the humidity.

```
for weather in forecast:
    day = gmt_to_eastern(weather.reference_time())
    date = day.date()
    if date not in dates:
        if len(days) >= 3:
            return(days, humidity)
        dates.append(date)
        humidity.append(weather.humidity)
        days.append(date)

    if day.hour in [11, 12, 13]:
        humidity[-1] = weather.humidity
```

The final GUI is shown below for your reference.



That's it!

Go back through the hints if needed to make sure you have updated everything. When you're done, feel free to share your project with others so they can see your awesome GUI!