Weather API

Sævar Ingi Sigurdarson
CSM0120 - Programming for Scientists

5th January 2021

Contents

1 Executive Summary

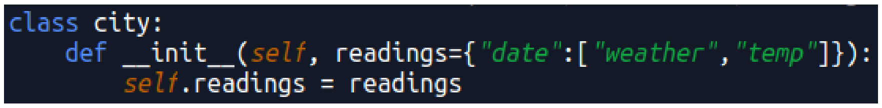
This is a report for a python program that gathers weather data from a web Application Programming Interface (API) and gives predictions at three hour intervals for the next five days. The program starts by initialising/defining the uniform resource locator (url) for the web-page, the API key that allows access to the data and the cities that the programs should gather data about. Once the program has the url, api key and the city names it will loop though for each city that has been asked for. The program requests a the url of the web-page with the api key and the city name as parameters, and if it gets a connection then it reads in the data in json format, which will then be converted to a python dictionary and then the desired data is extracted into python lists.

Once the data is in it will then be made into a comma-separated values (csv) file (like excel uses) for an easy read. And after that has happened for all the cities that were asked for, the cities are sorted into different categories depending of what the forecast for that city is. Following the specifications from the assignment brief, a city can be in three of four categories (depending on weather), raining, snowing, ice or other. The choice for where the city will be lies in the data for the next day. The program gets today's date and looks at the web data for the next day and sorts the cities depending on what the predictions were. After doing so for all the cities, the program prints out where it is likely to rain, snow, be icy and if the weather will be something else in the next day. lastly the program produces Extensible Markup Language (xml) file of the output for the day which gives the ability to read the suggestions without having to execute the program.

2 Technical Overview

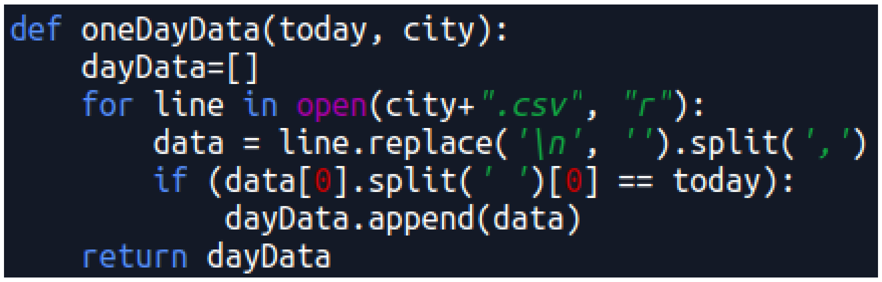
In this assignments a good variety of structures are used, and the choice for them is mainly based of what and how the material was taught during the course. The choice for how the cities are stored was made to be a list as that would be the easiest to alter to add a new city to it, it would just be adding it as a string. As for the categories of the weather that need to be printed out, a list was used there as well as they can be different sizes depending on the weather and day. The web url parameters was set up as a dictionary mainly because that is how it was taught, but also because the city parameter would change whenever the function is called. As for when the data is read, it comes in a json format and the easiest conversion to python is by changing it to a dictionary. Then the data is stored as lists for easy writing to the csv file, as that allowed all the data to match up completely. the main data structure used in this program is the list, as it is very versatile and changeable which suits the programs needs.

For how to handling of the data a choice was made to not to use a class for the cities but rather just read the data from the files that the program created. Figure1 shows the early attempt and thoughts at a class for the cities. However, the realisation that the data is all there in file form made the selection of just reading in the desired data more compelling to use. Figure2 shows how the data is read and selected depending on the day that is needed.



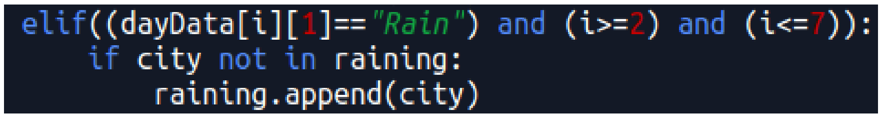
```
class city:
    def __init__(self, readings={"date":["weather","temp"]}):
        self.readings = readings
```

Figure 1: code snippet of what the idea of the class was supposed to look like



```
def oneDayData(today, city):
    dayData=[]
    for line in open(city+".csv", "r"):
        data = line.replace('\n', '').split(',')
        if (data[0].split(' ')[0] == today):
            dayData.append(data)
    return dayData
```

Figure 2: Function showing how the program reads in the data for one day



```
elif((dayData[i][1]=="Rain") and (i>=2) and (i<=7)):
    if city not in raining:
        raining.append(city)
```

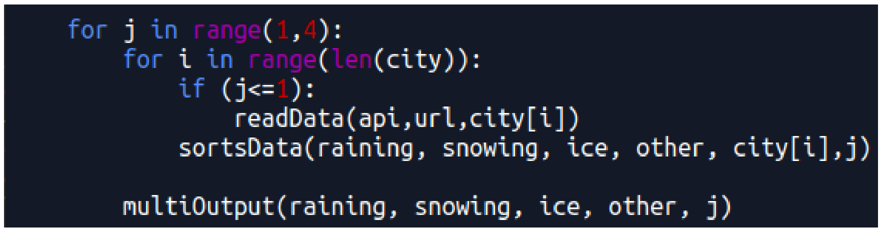
Figure 3: If statement to make sure rain is only measured between 06:00 and 20:59

As for the loops in the program the main loops used are for loops as they give the ability to reference the specific index for all lists that the program has. This was very useful as most data structures are lists. As for special cases, like the fact that a city has to have rain between 06:00:00 and 20:59:59 (the code for this is shown in Figure3). Here the time plays a part in if it will be classified as a rain city or not, which was handled with adding a few conditions to check if the time matched for the allowed time.

3 Software Testing

the code was tested in a manual way with continuous testing happening every time any changes were made. The way of testing the code could be referred to as a the one function, one city approach where the city list only had one city in it to see if that would work, and the addition of functions had to be able to run and work with the one city, until the whole program was done. After the program was able to handle one city and output everything correctly, the changes were made to see if the program was able to handle two, which was done by just adding a new city to the city list as the program was made around getting a list of cities.

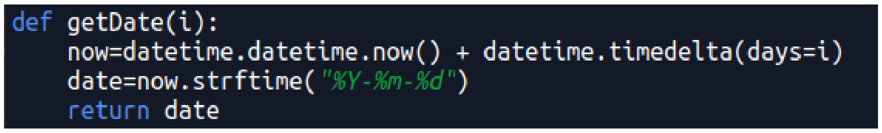
The next step was to add the multiple days which was neatly handled with a for loop that looped through a range(see Figure4) and passing the number into functions with the other data so that when the getDate function is called it knows how many days to add to get the correct day.(See Figure5). This was initially tested with just two cities, and later with the full list of cities.



```
    for j in range(1,4):
        for i in range(len(city)):
            if (j<=1):
                readData(api,url,city[i])
            sortsData(raining, snowing, ice, other, city[i],j)

        multiOutput(raining, snowing, ice, other, j)
```

Figure 4: Main body of the starting function



```
def getDate(i):
    now=datetime.datetime.now() + datetime.timedelta(days=i)
    date=now.strftime("%Y-%m-%d")
    return date
```

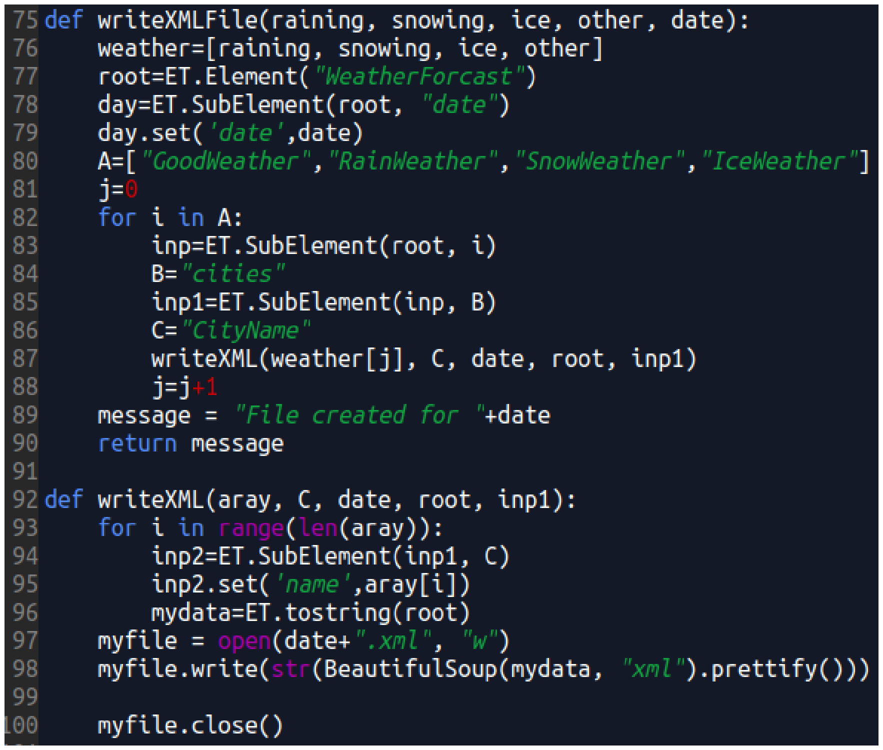
Figure 5: Function for getting the date and adding the appropriate amount

The assurance that the output is correct comes from having looked up the web page data in a browser and comparing it with the data in the csv files at the start and since that part of the code was never changed it is safe to say that the data gathered is correct. Also before creating the files all the data was printed in the console to see if the correct data was being gathered (as in the three aspects asked for in the brief).

4 Reflection and Future Work

For improvement on the code, the main change would be to change the inefficiency of the xml file writer, where it has for loop for the writing for each list of weather conditions after each other in one function. This was attempted to fix (See Figure6), but did not work so was not implemented. That would be the main improvement.

As for extending the program, the first change would be to make the list of cities available for change, where you can add and delete cities after personal needs, as well as add a function to print a list of the cities it will look for. After that, the next desirable function would be to have the program put onto a raspberry pi where the pi could run it at a specified interval to get up-to-date data of the weather and have that data uploaded to a web page for viewing when one is out and can not run the program.
The estimate time for the assignment was around 96 hours total, including report writting and commenting.



```python
def writeXMLFile(raining, snowing, ice, other, date):
    weather=[raining, snowing, ice, other]
    root=ET.Element("WeatherForcast")
    day=ET.SubElement(root, "date")
    day.set('date',date)
    A=["GoodWeather","RainWeather","SnowWeather","IceWeather"]
    j=0
    for i in A:
        inp=ET.SubElement(root, i)
        B="cities"
        inp1=ET.SubElement(inp, B)
        C="CityName"
        writeXML(weather[j], C, date, root, inp1)
        j=j+1
    message = "File created for "+date
    return message

def writeXML(aray, C, date, root, inp1):
    for i in range(len(aray)):
        inp2=ET.SubElement(inp1, C)
        inp2.set('name',aray[i])
        mydata=ET.tostring(root)
    myfile = open(date+".xml", "w")
    myfile.write(str(BeautifulSoup(mydata, "xml").prettify()))

    myfile.close()
```

Figure 6: Remake thoughts of the XML printing