Northeastern University
DS2001 – Programming with Data — Social Science Practicum
Fall 2019, Mattsson & McCabe

Practicum 9: Search

In this week's practicum we will be processing data from the web. When we send our search queries to the online server, it will return a dictionary of information.

1 Weather in Boston

Accessible weather predictions are an essential part of modern life. As it turns out, they are also an excellent example of working with nested data structures and classes. The online weather service we will be using is OpenWeather. It is available here: https://openweathermap.org/.

1.1 Download

Please download `pr09.zip` from the practicum website (or Blackboard), unzip it, and move the directory/folder to where you want it in your file system. Use Atom's `File > "Add Project Folder..."` to open this folder and view `weather.py` and `weather.json`.

1.2 JSON data

Dictionaries are more flexible that lists, so they can do more than embed the header in a simple table of data. Data can be stored on disk as a dictionary using the JSON format. Please familiarize yourself with this data format by poking around in `weather.json`. Note that it uses most of the types we have encountered in previous weeks, and their representations in the file are quite similar to the appropriate syntax in Python. So

```
"coord":{
      "lon":-71.06,
      "lat":42.36
   },
```

encodes a dictionary with two (string) keys, lat and lon, and their (float) values. Similarly, the presence of square brackets indicates a list.

1.3 The `json` module

You can load a dictionary from JSON file like this:

```
import json
content = open('filename.json', encoding='utf-8').read()
data = json.loads(content)
```

Please load the data from `weather.json` and confirm that you can navigate this data as a dictionary. Find:

- the name of the city

- the timestamp of the weather report
- the temperature
- the description of the weather

1.4 Human-readable weather

You'll notice that a few of these pieces of data are not what we would call "human readable". The time of the weather report is a UNIX integer timestamp and the temperature is in Kelvins. Please create two helper-functions and one print function that make the weather report human-readable.

`convert_time` This function should take a UNIX integer timestamp and return a human-readable string with the time. You can use the `time` module, like so:

```
import time
time_string = time.ctime(time_unix)
```

`convert_temp` This function should take a temperature in Kelvins and return a temperature in Fahrenheit. Hint: Wikipedia has the conversion formula.

`print_weather` This function should take a weather dictionary as its argument and print the weather in an understandable format. Your function should print (again) the name of the city, and the timestamp, temperature, and description of the weather.
HINT: Look at the `weather` key in `weather.json`. It's a list (of dictionaries)! Although in the input we're currently considering this list has only one element, let's make this function safe (and useful) for when there's multiple weather conditions (e.g, mist *and* rain) by using a list comprehension and `join`.

2 Weather data object

We know that we will always receive weather forecasts in this exact same dictionary format. We know that it will always be useful to use the helper functions we've written for this dictionary. And we know it will always be useful to print the contents of this dictionary using the specific function we've written.

Objects in Python let us assign functions to a particular data structure. You have already used several of these! For example, `append` and `pop` are a methods assigned to the `list` object. It would be useful to us to create a *data object* for a weather report.

2.1 Initialize a weather object

In your starter file (`weather.py`) you should see a statement that defines an object called `WeatherReport`. Object definition statements must include an `__init__` method that initializes an instance of that object. Methods assigned to an object always take `self` as an argument, and in this case we also want the initializing method to take a weather dictionary. Use the following syntax to fill in the `__init__` method:

```
def __init__(self, weather_dictionary):
    self.id = weather_dictionary["id"]
    self.name = weather_dictionary["name"]
    ...
    ...
```

Confirm that your object definition is working using the following syntax:

```
my_object_instance = MyObject(my_dictionary):
print(my_object_instance.property)
```

Our `WeatherReport` class is going to contain some, but not all, of the information contained in the JSON

2.2 Implement utility methods: `get_description`

Let's implement a method for our class to interact with various attributes.
The descriptions of the weather events are stored in the `weather` attribute, but this is a list of dictionaries that is somewhat awkward to work with. Our `get_description()` method will make these easier to access by returning a string containing all the descriptions lodges in this list of dictionaries.

2.3 Printing with methods: `summarize()`

Similar to our `print_weather` function from earlier, we can write a method that prints a nicely formatted string that we can print out. So, where we wrote

```
print_weather(our_weather_dict)
```

we can now write

```
our_weather_report.summarize()
```

2.4 Using the `__str__` method to print

What happens if we forget the method call and simply try to `print(our_weather_report)`? We will see something odd like

```
<__main__.WeatherReport object at 0x7fcfbc34b668>
```

What is this, and where did it come from? When Python tries to print an object, it attempts to get a string representation of that object by calling its `__str__` method; the default one simply prints out the module the object is located in, the name of the object, and its address in memory. This isn't very useful to us! Since we already have a function that creates a nice string representation of our `WeatherReport`, let's override the `__str__` method for our class so that we can just print it and see something useful. For this, we can copy a lot of the code we wrote in `summarize`, except here we want to build up a string to return instead of `print`ing.

3 Extra stuff!

3.1 APIs!

Various online services provide access to their data using the interfaces called APIs. The file you worked with in the previous sections was a saved version of a query to the OpenWeather API.

3.2 Get an API key from OpenWeather

Go to `https://home.openweathermap.org/users/sign_up` and request a free account. After you have registered go to `https://openweathermap.org/price` and press `Get API key and Start`. You should get an email with an API key.

3.3 Query the OpenWeather API

Your API key should work after a few minutes. Try by opening this link in your browser (remember to replace YOUR_APP_ID with the API key you got in your email):

https://api.openweathermap.org/data/2.5/weather?q=Boston&appid=YOUR_APP_ID
If there is a long delay in getting your API key working, you can try with mine (in your starter file).

3.4 Query the OpenWeather API using the `requests` module

You can also query APIs from within Python, and save the response as a variable. Use the following syntax:

```
import requests
url = 'https://api.openweathermap.org/data/2.5/weather?q=Boston&appid=YOUR_APP_ID'
content = requests.get(url).text
data = json.loads(content)
```

Play around with the different queries you can make to find the weather report for different cities and different times.

This handout was originally created by Carolina Mattsson and Stefan McCabe, Fall 2019. This exercise was originally created by Piotr Sapiezynski, Spring 2019.