CS2316 - HOMEWORK 05: WIKI WEBSCRAPER AND API ANALYST

Purpose

Often times in academic and corporate settings, engineers and professionals seek to derive meaningful information from data to meet a given objective. Before one can begin cleaning or analyzing data they first have to collect it. In some cases, data collection is as simple as downloading a CSV file, however this is infeasible and impractical for many applications. For example, an application that displays weather data should collect live data through an API instead of downloading a CSV file.

Skills
This assignment will give you practice scraping data from a website using the bs4 module, accessing data stored in API using the requests module, and transforming the raw data into the desired format.

Knowledge
The types of problems you will encounter working with data sets and attempting to get them into a consistent format is similar to that of industry professionals working with real-world data sets.

Important

1. Due Date: **03/11/2020** at **11:59 pm**
2. This homework is graded out of **100 points**. However, the autograder will only show up to **90 points** when you submit your code. We are saving some test cases for the final autograder that we will use. This is to encourage you to start developing ways to test features of your code when you aren't given test cases in an autograder. We also reserve the right to change the test cases later if we need to. Therefore, the grade reflected in Gradescope does not reflect your final grade on HW05.py.
3. This is an individual assignment. You may collaborate with other students in this class. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. Students may only collaborate with fellow students currently taking CS 2316, the TA's and the lecturer. You should not exchange code or write code for others. For individual assignments, each student must turn in a unique program. Your submission must not be substantially similar to another student's submission. Collaboration at a reasonable level will not result in substantially similar code.
4. For Help:
    - TA Helpdesk (Schedule posted on class website)
    - Email TA's or use Piazza Forums Notes
    - [Think Like a Computer Scientist]
    - [Requests Module Documentation]
    - [RegEx Module Documentation]
    - [BeautifulSoup Module Documentation]

5. Comment out or delete all your function calls. Only global variables, and comments are okay to be outside the scope of a function. When your code is run, all it should do is run without any errors.
6. Do not wait until the last minute to do this assignment in case you run into problems.
7. Read the entire specifications document before starting this assignment.
8. Having function calls or extraneous code outside the scope of functions will result in an automatic 0.
9. If your code cannot run because of an error, it is a 0%.

Introduction

The goal of this homework is to showcase your knowledge of Regular Expressions, web scraping, and Application Programming Interfaces (APIs). You should test the functions out first on your own computer, then when you have one or more of them working, upload the entire file (which must be named HW05.py) to GradeScope to see how well your code performs on the test cases we have created. You can submit the homework file as many times as you'd like before the deadline.

The file you should have downloaded from Canvas for this assignment is `wiki_tourism.html`, which contains the HTML code from this Wikipedia site. Make sure you have placed this file in the same directory as your HW05.py file. You may choose to open this HTML file using either a browser, such as Google Chrome, or a text editor. Notice that when you open it up in a browser, it does not look aesthetically pleasing. This is because the HTML code only contains the backbone structure of the website and needs to be rendered by CSS to look aesthetic. If you open it up in a text editor, you will be able to see the plain HTML code.

Part I: Introduction to the APIs

For this assignment, we will be using two APIs: *RESTCountries* and *OpenWeatherMap*. Documentation for both APIs can be found on their websites.

[RESTCountries]
[OpenWeatherMap]

Important: The *OpenWeatherMap* requires a key in order to use each request. Instructions on how you can acquire and use your personal key can be found here.

[OpenWeatherMap AppID]

CS2316 - HOMEWORK 05: WIKI WEBSCRAPER AND API ANALYST

Steps for obtaining a personal API Key:

1. Visit https://openweathermap.org/api and click “Sign Up” in the top right-hand corner, or simply visit https://home.openweathermap.org/users/sign_up
2. Follow the instructions listed on the sign-up page and create an account by filling out all of the information on the page (you do not have to select the system/product news checkboxes).
3. Log in with your account information and go to the “API Keys” tab or click here. It might take a few hours for your API Key to get activated.
4. Save and copy your API Key into your code for easy access. Replace your unique API Key within the denoted url formats.

Example:

```
https://api.openweathermap.org/data/2.5/weather?q={city name}&appid={your api key}
```

Tip for utilizing APIs:

Since APIs often return very long dictionaries, it is advisable to use Python’s built-in module called pprint in order to be able to better see the keys and values of your returned dictionary.

```
from pprint import pprint

import requests

web_data = requests.get(<url>)

data = web_data.json()

print(data) # prints out a hard to read dictionary

pprint(data) # prints out a beautiful dictionary <3
```

Part II: Google Chrome Inspect Tool

When dealing with web scraping, a handy tool to utilize is Google Chrome's Inspect Tool. This allows you to better visualize and read the source code for each html page.

There are 3 ways to access the Inspect Tool:

1. Click on the three vertical dots in the upper right hand cover → More tools → Developer Tools
2. Right-click on the page and choose "Inspect"
3. Ctrl + Shift + C (**Windows**) or Cmd + Opt + C (**Mac**)

You should now be able to see the page's source code including other various details. You can select the square with the cursor at the top left corner of the menu to bring up "Inspect Element" mode, which will highlight the specific HTML line when hovering over your page.

Part III: Functions for HW05.py

Note: The grading rubric at the end of the document for point distribution.

Application Programming Interface

For the following functions, you will need to request data from both the RESTCountries and OpenWeatherMap API. Consult the Requests Module Documentation if you run into difficulties. Hardcoding instead of using the requests module to request data from the specified APIs will not receive any points.

Note: Data contained in APIs can change over time. If your results do not match completely with the test cases, it does not mean you are necessarily incorrect.

Function name: **`country_statistics`**
Parameters: `country` (str)
Return Type: dict
Description: Write a function that takes in a `country` name as a string and returns a dictionary with the keys 'country', 'capital', 'population', 'coordinates', 'timezone', 'currency', and 'borders' with the corresponding values from the *RESTCountries* API. If the country is not a valid country listed in the API, you should return a string in the following format: `'{country} is not a valid listed country from the API.'`. You may hard code the keys of the output dictionary.
Note: Use the "Full Name" base url from RESTCountries.

Test Cases:

```
>>> country_statistics('China')
{
 'country': 'China',
 'capital': 'Beijing',
 'timezone': ['UTC+08:00'],
 'coordinates': [35.0, 105.0],
 'currency': [{'code': 'CNY', 'name': 'Chinese yuan', 'symbol':
              '¥'}],
 'population': 1377422166,
 'borders': ['AFG', 'BTN', 'MMR', 'HKG', 'IND', 'KAZ', 'PRK',
             'KGZ', 'LAO', 'MAC', 'MNG', 'PAK', 'RUS', 'TJK',
             'VNM']
}
```

*The output dictionary does not have to be in the specific order

```
>>> country_statistics('Atlantis')
Atlantis is not a valid listed country from the API.
```

Function name: **`bordering_countries`**
Parameters: `country` (str)
Return Type: list
Description: Write a function that takes in a `country` name as a string and returns a list of country names that border with the country parameter. If the country does not share any borders with other countries, return an empty list. If the `country` is not a valid country listed in the API, return the string in the following format: `'{country} is not a valid listed country from the API.'`.
Hint: You can use your **`country_statistics`** function.
Note: If a country shares borders, each bordering country is represented by 3-letter country codes. Make sure to use the "List of Codes" base url from RESTCountries.
Test Cases:

```
>>> bordering_countries('China')
['Afghanistan', 'Bhutan', 'Myanmar', 'Hong Kong', 'India',
'Kazakhstan', "Korea (Democratic People's Republic of)",
'Kyrgyzstan', "Lao People's Democratic Republic", 'Macao',
'Mongolia', 'Pakistan', 'Russian Federation', 'Tajikistan',
'Viet Nam']

>>> bordering_countries('Japan')
[]

>>> bordering_countries('Atlantis')
Atlantis is not a valid listed country from the API.
```

CS2316 - HOMEWORK 05: WIKI WEBSCRAPER AND API ANALYST

Function name: **`weather_watcher`**
Parameters: `country` (str), `max_humidity` (int)
Return Type: tuple
Description: Write a function that takes in a `country` name as a string and `max_humidity` as an int and returns a tuple with the following format:

- 0th index: boolean of `True` or `False`
- 1st index: str with the format:
    - `'The current forecast for {capital} is {description} with an average temperature of {average temperature} Kelvin and humidity of {humidity}%.'`

You should first find the capital of the given `country`. After finding the corresponding capital, you should then find the current humidity, average temperature, and weather description for that city. You can calculate the average temperature by finding the sum of the highest and lowest temperatures and dividing by 2. Then, round the average temperature to 2 decimal places. If the current humidity is less than or equal to the `max_humidity`, the boolean should be `True`. Otherwise, it should be `False`. If the country is not a valid country listed in the API, you should return the string in the following format: `'{country} is not a valid listed country from the API.'`.
Test Cases:
Your output is likely to be different from ours as the API reflects the real time data.

```
>>> weather_watcher('China', 55)
(True, 'The current forecast for Beijing is broken clouds with
an average temperature of 280.93 Kelvin and humidity of 21%.')

>>> weather_watcher('Japan', 50)
(False, 'The current forecast for Tokyo is few clouds with an
average temperature of 285.15 Kelvin and humidity of 54%.')

>>> weather_watcher('Atlantis', 65)
Atlantis is not a valid listed country from the API.
```

CS2316 - HOMEWORK 05: WIKI WEBSCRAPER AND API ANALYST

Web Scraping

For the following functions, you must scrape data from the provided `wiki_tourism.html` file. Keep in mind that you should look at the HTML code of the downloaded `wiki_tourism.html` file, **not** the HTML code of the live website. You may use Google Chrome's Inspect Tool to help locate the specific positions of elements from the webpage within the HTML code, but there may exist minor attribute value differences between the file and the live source code.

Consult the BeautifulSoup Documentation if you run into difficulties. Hardcoding instead of using bs4 to scrape the website will not receive any points.

Function name: **`most_visited_raw_list`**
Parameters: `file_name` (str)
Return Type: list
Description: Write a function that takes in the `file_name` of the HTML file you should parse and returns a list of lists representing the table on "***most visited destinations by international tourist arrivals - worldwide***" (first table in the HTML). The `file_name` parameter will not include the file extension. Each sublist represents a row in the table. Each string entry in the sublist should only contain the data itself. Entries should not contain trailing white space characters such as "\n", "\t", " ", or "\xa0" (a non-breaking space widely used by HTML). Do not include the up-arrow and down-arrow images for the entries corresponding to columns **Change (2017 to 2018) (%)** and **Change (2016 to 2017) (%)**. Instead, use "`Increase`" to denote an up-arrow or "`Decrease`" to denote a down-arrow. Make sure your output matches the test case output below. You may use the re module for this function.

Here is an example on the elements you should include:

Rank	Destination	International tourist arrivals (2018)[1]	International tourist arrivals (2017)[1]	Change (2017 to 2018) (%)	Change (2016 to 2017) (%)
1	France	89.4 million	86.9 million	▲2.9	▲5.1
2	Spain	82.8 million	81.9 million	▲1.1	▲8.7
3	United States	79.6 million	76.9 million	▲3.5	▲0.7

*do not include [1]

CS2316 - HOMEWORK 05: WIKI WEBSCRAPER AND API ANALYST

Test Case:

```
>>> most_visited_raw_list('wiki_tourism')
[
 ['Rank',
  'Destination',
  'International tourist arrivals (2018)',
  'International tourist arrivals (2017)',
  'Change (2017 to 2018) (%)',
  'Change (2016 to 2017) (%)'],
 ['1', 'France', '89.4 million', '86.9 million', 'Increase 2.9',
 'Increase 5.1'],
 ['2', 'Spain', '82.8 million', '81.9 million', 'Increase 1.1',
 'Increase 8.7'],
 ['3', 'United States', '79.6 million', '76.9 million',
 'Increase 3.5', 'Increase 0.7'],
 ['4', 'China', '62.9 million', '60.7 million', 'Increase 3.6',
 'Increase 2.5'],
 ['5', 'Italy', '62.1 million', '58.2 million', 'Increase 6.7',
 'Increase 11.2'],
 ['6', 'Turkey', '45.8 million', '37.6 million', 'Increase
 21.7', 'Increase 24.1'],
 ['7', 'Mexico', '41.4 million', '39.3 million', 'Increase 5.5',
 'Increase 12.0'],
 ['8', 'Germany', '38.9 million', '37.5 million', 'Increase
 3.8', 'Increase 5.2'],
 ['9', 'Thailand', '38.3 million', '35.5 million', 'Increase
 7.9', 'Increase 9.1'],
 ['10', 'United Kingdom', '36.3 million', '37.6 million',
 'Decrease 3.5', 'Increase 5.1']
]
```

Function name: **`most_visited_cleaned_list`**
Parameters: `raw_list` (list)
Return Type: list
Description: Write a function that takes in a list of lists returned from `most_visited_raw_list()` and transforms the data into the desired format described below:

column	type
Rank	int
Destination	str
International tourist arrivals (2018)	int

International tourist arrivals (2017)	int
Change (2017 to 2018) (%)	float
Change (2016 to 2017) (%)	float

Notice that you should convert entries for both column **International tourist arrivals (2018)** and column **International tourist arrivals (2017)** from "`xx.x million`" to integers. Also notice that for column **Change (2017 to 2018) (%)** and **Change (2016 to 2017) (%)**, you should use a positive float to denote "`Increase`" and a negative float to denote "`Decrease`".
Note: Make sure to match all the spaces within the headers correctly!
Test Case:

```
>>> raw = most_visited_raw_list('wiki_tourism')
>>> most_visited_cleaned_list(raw)
[
 ['Rank',
  'Destination',
  'International tourist arrivals (2018)',
  'International tourist arrivals (2017)',
  'Change (2017 to 2018) (%)',
  'Change (2016 to 2017) (%)'],
 [1, 'France', 89400000, 86900000, 2.9, 5.1],
 [2, 'Spain', 82800000, 81900000, 1.1, 8.7],
 [3, 'United States', 79600000, 76900000, 3.5, 0.7],
 [4, 'China', 62900000.0, 60700000, 3.6, 2.5],
 [5, 'Italy', 62100000, 58200000, 6.7, 11.2],
 [6, 'Turkey', 45800000, 37600000, 21.7, 24.1],
 [7, 'Mexico', 41400000, 39300000, 5.5, 12.0],
 [8, 'Germany', 38900000, 37500000, 3.8, 5.2],
 [9, 'Thailand', 38300000, 35500000, 7.9, 9.1],
 [10, 'United Kingdom', 36300000.0, 37600000.0, -3.5, 5.1]
]
```

Function name: **`receipt_list`**
Parameters: `file_name` (list)
Return Type: list
Description: Write a function that takes in the `file_name` of the HTML file you should parse and returns a list of lists representing the table from "***international tourism receipts - worldwide***" (seventh table in the HTML). The `file_name` parameter will not include the file extension. You should only include column **Rank**, column **Region**, and column **International tourism receipts (US$ billion)(2017)** . Each sublist represents a row in the table. Each string entry in the sublist should only contain the data itself. Entries should not contain trailing white space characters such as "\n", "\t", " ", or "\xa0" (a non-breaking space widely used by HTML). If there is a newline character in the

middle of a string entry, you need to remove that newline character as well. You should transform the data into the desired format described below:

column	type
Rank	int
Region	str
International tourism receipts (US$ billion)(2017)	float

Here is an example on the elements you should include:

Rank	Region	International tourism receipts (US$ billion) (2018)[1]	International tourism receipts (US$ billion) (2017)[3]	Change (2017 to 2018) (%)
1	United States	214.5	210.7	▲ 1.8%
2	Spain	73.8	68.1	▲ 8.4%
3	France	67.3	60.7	▲ 10.9%

*do not include [3]

Test Case:

```
>>> receipt_list('wiki_tourism')
[
 ['Rank',
  'Region',
  'International tourism receipts (US$ billion)(2017)'],
 [1, 'United States', 210.7],
 [2, 'Spain', 68.1],
 [3, 'France', 60.7],
 [4, 'Thailand', 56.9],
 [5, 'United Kingdom', 49.0],
 [6, 'Italy', 44.2],
 [7, 'Australia', 41.7],
 [8, 'Germany', 39.8],
 [9, 'Japan', 34.1],
 [10, 'China', 38.6]
]
```

Function name: **`expenditure_per_arrival`**
Parameters: `most_visited_list` (list), `expenditure_list` (list), `country` (str)
Return Type: int
Description: Write a function that takes in a list of lists returned from `most_visited_clean_list`, a list of lists returned from `receipt_list`, and a country name and returns the average expenditure of each international tourist arrival in that country in 2017. The average expenditure per arrival for the `country` should be calculated as {International tourism receipts in `country`} / {International tourist arrivals 2017 in `country`}. You should round the result to the nearest integer. For instance, the average expenditure per arrival for the United States is `210.7*1000000000/76900000 ≈ 2740`. Since `most_visited_list` and `expenditure_list` only include certain countries, the function should return `-1` if the `country` is not found in either of the lists.
Test Case:

```
>>> raw = most_visited_raw_list('wiki_tourism')
>>> cleaned = most_visited_cleaned_list(raw)
>>> expenditure_list = receipt_list('wiki_tourism')
>>> expenditure_per_arrival(cleaned, expenditure_list, 'Spain')
832
>>> expenditure_per_arrival(cleaned, expenditure_list, 'Canada')
-1
```

Part IV: Regex Practice

This assignment does not include practice on Regular Expressions. However, we strongly encourage you to get practice with using the re module in your own time.
Here are a few resources:

- Examples in Lecture Slides 19, 20, 21
- Handout *Requests, RegEx, BeautifulSoup*
- Recitation 8 slides and worksheet
- In-class Participation Exercise 3 on 02/27/2020
- Pythex: a Python regular expression editor: a great website to play around with regular expressions
- RegexOne: more regular expression practice

Hardcoding Clarification

You should not manually return the output from the specified APIs or the given HTML file in order to satisfy the test cases on GradeScope. We will be manually checking for hardcoding after the assignment is due and we will deduct points for hardcoding. How do you know if you are hard coding or not? A general rule of thumb is that there shouldn't be any actual data points in your code. Instead, your code should perform operations to retrieve those data points from either the APIs or the HTML file. If you are not sure, ask on Piazza.

Gradescope/Canvas Requirements

- **NO PRINT STATEMENTS**, as this will break the autograder
- **NO FUNCTION CALLS** outside of function definitions this will also break the autograder
- Make sure the file submitted is titled **HW05.py**
- Do not import any modules, packages, or libraries other than **re, requests, bs4, pprint**
- Only submit the file **HW05.py** file to GradeScope

Grading Rubric

```
country_statistics:                    10 pts
bordering_countries:                   15 pts
weather_watcher:                       15 pts
most_visited_raw_list:                 20 pts
most_visited_cleaned_list:             15 pts
receipt_list:                          15 pts
expenditure_per_arrival:               10 pts
_____________________________________________
Total                              100/100 pts
```

Note that the autograder on GradeScope does not include test cases for `expenditure_per_arrival`. Therefore, you are able to get up to **90 points** on GradeScope before the assignment is due. We are saving those test cases for later use. You are encouraged to write your own test cases for that function.