Homework 6: APIs, JSON, and Caching

In this assignment, you will get data using the OMDB API. You will also store the data in a cache file so you can save data instead of repeatedly requesting it from the API.

Click here to sign up for an API key. We strongly recommend reading the API documentation before getting started.

Strongly Recommended:

Choose an online JSON viewer. We recommend printing the API data/cache data and pasting it into the viewer to examine the structure of the data. Here are a few of the many available options for JSON viewers:

1. https://jsonformatter.org/
2. https://jsoneditoronline.org/

Tasks:

def load_json(filename):

- This function reads a cached JSON file (filename) and returns a dictionary with JSON data
    - If the request is unsuccessful, return an empty dictionary

def write_json(dict, filename):

- This function converts the cache dictionary into JSON format and writes the JSON to the cache file (filename) to save the search results

def get_movie_data(movie):

- This function takes in a movie name and makes a request to the OMDB API
    - If the request is successful, return a tuple with the data you got back from the API and the URL you requested
    - If a request is unsuccessful, return None

- **Hint:** Make sure you set the response type to JSON when you make the request

def cache_all_movies(movies, cache_file):

- This function goes through a list of movies (predefined for you in the main function) and gets IMBD data for each of them
    - Save data from successful requests to the cache
        - In your cache, the keys should be the URL you requested and the value should be a dictionary with the response data
        - If a request does not return any data, you should not add anything to the cache
        - If a movie is already in the cache, you should not add a duplicate entry
- This function returns a string that says the percentage of the films in movies that were successfully added to the cache
    - e.g. if you have 20 films in movies and can get the data for 19 of them, it should return “Cached data for 95% of movies”
    - If your code works correctly, this function should return “Cached data for 100% of movies” on the *first* call and “Cached data for 0% of movies” on all subsequent calls (because that data is already in the cache, there is no need to make an API request again)

Sample of a correctly formatted cache. Note that the full dictionary output is cut off since it’s quite lengthy:


```
{
  "http://www.omdbapi.com/?apikey=a&r=json&t=Mean
    +Girls": {
    "Title": "Mean Girls",
    "Year": "2004",
    "Rated": "PG-13",
    "Released": "30 Apr 2004",
    "Runtime": "97 min",
    "Genre": "Comedy",
    "Director": "Mark Waters",
    "Writer": "Rosalind Wiseman, Tina Fey",
    "Actors": "Lindsay Lohan, Jonathan Bennett, Rachel
      McAdams",
    "Plot": "Cady Heron is a hit with The Plastics,
      the A-list girl clique at her new school, until
      she makes the mistake of falling for Aaron
      Samuels, the ex-boyfriend of alpha Plastic
      Regina George.",
    "Language": "English, German, Vietnamese, Swahili"
      ,
    "Country": "United States, Canada",
    "Awards": "7 wins & 25 nominations",
    "Poster": "https://m.media-amazon.com/images/M
      /MV5BMjE1MDQ4MjI1OV5BMl5BanBnXkFtZTcwNzcwODAzMw@
      @._V1_SX300.jpg",
    "Ratings": [
      {
        "Source": "Internet Movie Database",
        "Value": "7.1/10"
```

def get_highest_box_office_by_year(year, cache_file):

- For a given year, this function gets the movie with the highest box office total
    - Gets data from the cache
        - Return the name and box office total for the most successful film as a tuple
        - You do not need to do anything to handle potential ties
        - **Hint:** The box office totals may not be formatted as a number (e.g. you may get values like "$1,000,000")
    - If there are no movies in our cache that were made in a given year, return "No films found"
- Example:
    - get_highest_box_office_per_year(2012) should return ('The Avengers', 623357910)

def get_genres_above_cutoff(cutoff, cache_file):

- This function finds the genres associated with at least "cutoff" number of movies in our dataset
    - For example, if cutoff = 5 we would get genres that are associated with at least 5 movies
    - **Hint:** Many movies are associated with multiple genres. Make sure to count each one.
- It returns a dictionary that maps genres to their counts.
- For example, say we have three movies and cutoff = 2
    - Movie A is labeled as 'Comedy' and 'Action'
    - Movie B is labeled as 'Drama' and 'Action'
    - Movie C is labeled as 'Comedy' and 'Romance'
    - We should return {'Comedy': 2, 'Action': 2}
- Example:
    - get_genres_above_cutoff(5, 'cache.json') should return {'Drama': 16, 'Romance': 6, 'Action': 6, 'Adventure': 12, 'Fantasy': 5, 'Comedy': 11}

Extra Credit (6 Points):

2 points: *read_api_key(filename):*

- It is best practice **to never copy and paste your API key into your code in plain text**. This is especially true in professional environments, where you may be working with APIs that require you to pay for each request.
- Save your API key to a .txt file called api_key.txt. Then, read that file and use its contents to reference your API key using ***read_api_key***.
    - This file should be in the same directory as your HW6.py file
    - <u>You will not receive points if you don't name your file correctly or if it is in the wrong directory</u>

- <u>You will not receive points if your API key is visible in your code in plaintext, even if you implement this function</u>
    - Having a variable that stores the *value* of your API key is okay, but there should not be a line like api_key = ##### where your key is visible in plain text

4 points: *get_rotten_tomatoes_rating(title, cache_file):*

- This function gets the Rotten Tomatoes rating for a given film
    - If there is no Rotten Tomatoes review, return “No review found”
    - Otherwise, return the rating given
- Example:
    - get_rotten_tomatoes_rating(‘Titanic’, ‘cache.json’) should return “88%”

Rubric:

- Do not modify any of the test cases we have provided. Deleting or modifying the test cases will result in points being deducted.
- load_json (5 points)
- write_json (5 points)
- get_movie_data(5 points)
- cache_all_movies *(15 points)*
- get_highest_box_office_by_year (15 points)
- get_genres_above_cutoff (15 points)