Homework 7: APIs, JSON, and Caching

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

We have provided the following in the starter code:

1. **read_cache(CACHE_FNAME)**: This function reads JSON from the cache file and returns a dictionary from the cache data. If the file doesn't exist, it returns an empty dictionary.
2. **main()** function
3. Test cases to test the functions you will write

Before you proceed to the tasks:

You will need an API key for this HW. You can generate your key here: http://www.omdbapi.com/apikey.aspx

Assign your API key to the variable API_KEY on line 13!

Strongly Recommended

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

1. https://jsonformatter.org/
2. https://jsonformatter-online.com/
3. https://jsonlint.com/

Tasks - You will write the following functions.

def write_cache(CACHE_FNAME, CACHE_DICT):

This function encodes the cache dictionary (***CACHE_DICT***) into JSON format and writes the JSON to the cache file (***CACHE_FNAME***) to save the search results.

def create_request_url(title):

This function prepares and returns the request URL for the API call. The documentation of the API parameters is at http://www.omdbapi.com/

You must provide the following parameters in the request URL, in addition to the title:

1. type: options are 'movie', 'series', 'episode'
2. plot: set to 'short'
3. r: set to 'json'

Example of a request URL for movie title The Dark Knight:
`http://www.omdbapi.com/?apikey=xxxxxxxxxx=The Dark Knight&type=movie&plot=short&r=json`

The API key has been blurred out since one shouldn't share API keys publicly

def get_data_with_caching(title, CACHE_FNAME):

This function uses the passed movie title to first generate a ***request_url*** (using the **create_request_url** function). It then checks if this URL is in the dictionary returned by the function **read_cache**. If the ***request_url*** exists as a key in the dictionary, it should print `"Using cache for <title>"` and return the results for that ***request_url***.

If the ***request_url*** does not exist in the dictionary, the function should print `"Fetching data for <title>"` and make a call to the OMDB API to get the movie data.

If data is found for the movie, it should add it to a dictionary (the key is the ***request_url***, and the value is the results) and write out the dictionary to a file using **write_cache**.

In certain cases, the OMDB API may return a response for the ***request_url*** but it may not contain any data for the movie: `{"Response":"False","Error":"Movie not found!"}`

DO NOT WRITE THIS DATA TO THE CACHE FILE!
Print "Movie Not Found" and return None

If there was an exception during the search (for reasons such as no network connection, etc), it should print out `"Exception"` and return None.

def top_ten_movies(key, CACHE_FNAME):

This function calls **read_cache()** to get the movie data stored in the cache file. It analyzes the dictionary returned by **read_cache()** and sorts the movies in the dictionary on the basis of the rating key (either imdb or metacritic). It returns a list of tuples. The first item in each tuple is the movie title, and the second item is the value of the rating key.

For example, if the *key* = *imdbRating*, the function will return top-ten movies ranked by their imdb rating. If the *key* = *Metacritic*, the function will return top-ten movies ranked by their metacritic score

Example Output

NOTE: Your example output ***may*** look different from this. It will depend on two things:

(1) ***whether you have commented out the unit tests*** - the test cases use a different list of movies and their results will also get stored in the cache file

(2) ***whether you delete the cache file*** - then you lose all the data stored in the cache file and you may see print statements which say "Fetching data from…"

```
Using cache for Inception
Using cache for Inception
_____________________________
Using cache for Parasite
Using cache for Parasite
_____________________________
Using cache for Ladybird
Using cache for Ladybird
_____________________________
Top 10 movies in the cache by imdb rating
[('The Dark Knight', 9.0), ('Inception', 8.8), ('Parasite', 8.6), ('Joker', 8.6), ('Spirited Away', 8.6), ('1917'
, 8.5), ('Memento', 8.4), ('The Seventh Seal', 8.2), ('Rashomon', 8.2), ('Raging Bull', 8.2)]
_____________________________
Top 10 movies in the cache by metacritic score
[('Rashomon', 98.0), ('Parasite', 96.0), ('Spirited Away', 96.0), ('Finding Nemo', 90.0), ('Raging Bull', 89.0),
('Lagaan: Once Upon a Time in India', 84.0), ('The Dark Knight', 84.0), ('Once Upon a Time... in Hollywood', 83.0
), ('Memento', 80.0), ('Gandhi', 79.0)]
_____________________________
Thriller movies in the cache:
['Memento', 'Parasite', 'The Dark Knight', 'Train to Busan', 'Joker', 'Inception']
_____________________________
Action movies in the cache:
['The Dark Knight', 'Train to Busan', 'Inception']
_____________________________
test_create_request_url (__main__.TestHomework7) ... ok
test_get_data_with_caching (__main__.TestHomework7) ... Using cache for The Dark Knight
Using cache for Rashomon
Using cache for The Seventh Seal
Using cache for Train to Busan
Using cache for Raging Bull
Fetching data for Random character
Movie not found!
Using cache for Once Upon a Time... in Hollywood
Using cache for Joker
Using cache for 1917
Using cache for Memento
Using cache for Spirited Away
Using cache for Finding Nemo
Using cache for Gandhi
Using cache for Lagaan
Fetching data for Random character
Movie not found!
ok
test_movie_list (__main__.TestHomework7) ... ok
test_top_ten_movie (__main__.TestHomework7) ... ok
test_write_cache (__main__.TestHomework7) ... ok

----------------------------------------------------------------------
Ran 5 tests in 0.330s
```

Grading Rubric

def test_write_cache - 5 points

- 5 points for writing the JSON data correctly to the cache file

def test_create_request_url - 5 points

- 2 points for including the API key in the request URL
- 2 points for including the movie title in the request URL
- 1 point for including the remaining 3 parameters - plot, type, r

def test_get_data_with_caching - 35 points

- 5 points for correctly getting existing data from the cache file
- 5 points for getting new data using the request_url from the API
- 5 points for checking if the data was found for the movie title provided
- 5 points for adding data to the cache dictionary and cache file only for movies that exist
- 5 points for writing out the changed cache dictionary to the cache file
- 5 points for returning the correct result & type
- 5 points for printing "Exception" if there was an exception and returning "None"

def test_top_ten_movies - 15 points

- 3 points for returning a list of tuples
- 3 points for sorting the items in the list on the basis of rating values
- 3 points for returning no more than 10 items
- 3 points for returning the right values in the tuple (movie title, rating value)
- 3 points for converting the ratings to float from string

Extra Credit - 6 points

def movie_list(genre, CACHE_FNAME):
The function calls **read_cache()** to get the movie data stored in the cache file. It analyzes the dictionary returned by **read_cache()** to identify all the movies that belong to the specified genre and returns a list of those movies.

def movie_list - 6 points

- 3 points for returning a list
- 3 points for the right movies in the list