Homework 6: APIs, JSON, and Caching

In this assignment, you will get data using the iTunes Search 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. **main()** function

2. test cases for all functions except sort price

This assignment does not require you to generate a personal API key, however, we strongly recommend you to read the [iTunes Search API documentation](https://affiliate.itunes.apple.com/resources/documentation/itunes-store-web-service-search-api/) first.

**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**

**def read\_cache(CACHE\_FNAME):**

This function reads a cached JSON file (CACHE\_FNAME) and returns a dictionary with JSON data or an empty dictionary if the cache does not exist. Hint: use a try except

**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. When you write cache into JSON format, you need to unpack the second item of the dictionary, which is the actual content of your item.

For example:

{'resultCount': 2, 'results': [{\*INFORMATION ABOUT EACH ITEM\*},{\*INFORMATION ABOUT EACH ITEM\*}]}

In the above case, the *resultCount* is 2 because we set the limit number to be 2. For this assignment, you will set the limit number to be 1.

**def create\_request\_url(term, number=1):**

This function prepares and returns the request url for the API call.  The documentation of the API parameters is at [iTunes Store API](https://affiliate.itunes.apple.com/resources/documentation/itunes-store-web-service-search-api/).

Specifically, for this task, we only focus on two parameters:

1. *term*: The URL-encoded text string you want to search for. For example: billie+eilish.

2. *limit*: The number of search results that you want the iTunes Store to return for your

specific searched keywords. For example: 20.The default is 50. For simplicity, we only need to return one single item (default value of number = 1).

Example of a request URL for 2 of Billie Eilish’s albums:

             "<https://itunes.apple.com/search?term=billie+eilish&limit=2>"

**def get\_data\_with\_caching(term, CACHE\_FNAME):**

This function uses the passed search term 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 <term>" 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 <term>" and make a call to the **Search API** to get the data for that specific term.

If data is found for the term, it should add it to the dictionary (the key is the ***request\_url***, and the value is part of the results) and write out the dictionary to a file using **write\_cache**.

If the ***request\_url*** is not generated correctly (e.g., for some reason the limit number is not 1) and thus returns zero or multiple items, **do not write this data into the cache file.** Instead, print “**Request not set correctly**” and return None.

If there was an exception during the search (for reasons such as no network connection, no results are returned), it should print out “Exception” and return None. Note that you may wish to use a try and except clause to achieve this output.

**def sort\_price(CACHE\_FNAME):**

This function finds the highest priced item in your cache and returns the name of the item and the price in a tuple in the following format (‘Album’, price).

Example Output

Text

Description automatically generated

Text

Description automatically generated

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 the same list of items 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…"

Grading Rubric

**def test\_read\_cache - 5 points**

* 5 points for reading the JSON data correctly from the cache file

**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**

● 3 points for including the term in the request URL

● 2 points for composing the correct request URL

**def test\_get\_data\_with\_caching - 30 points**

● 5 points for creating request urls and for 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 iTunes terms provided

● 5 points for adding data to the cache dictionary and cache file only for iTunes items that exist

● 5 points for returning the correct result & type

● 5 points for printing "Exception" if there was an exception (e.g., no item found) and returning "None"

**def sort\_price - 15 points**

* 5 points for reading cache file and using only *collectionName* and *collectionPrice* data
* 5 points for returning the right price and collection
* 5 points for constructing a test case for this function in the unittests (this is the only test you will need to write, the rest are provided)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Extra Credit - 6 points

**def itunes\_counts - 6 points**

The function calls read\_cache() to get the iTunes data from your cache. It analyzes the dictionary returned by read\_cache(). This function returns a dictionary with genres and the number of their occurrences in the cache sorted in ascending order.

Expected results will be in the following format

{'Electronic': 1, 'Rock': 2, 'Alternative': 3, 'Pop': 6}