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. **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

This assignment does not require you to generate a personal API key, however, we strongly recommend you to read the iTunes Search API documentation first.

(https://affiliate.itunes.apple.com/resources/documentation/itunes-store-web-service-search-api/)

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. 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
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.

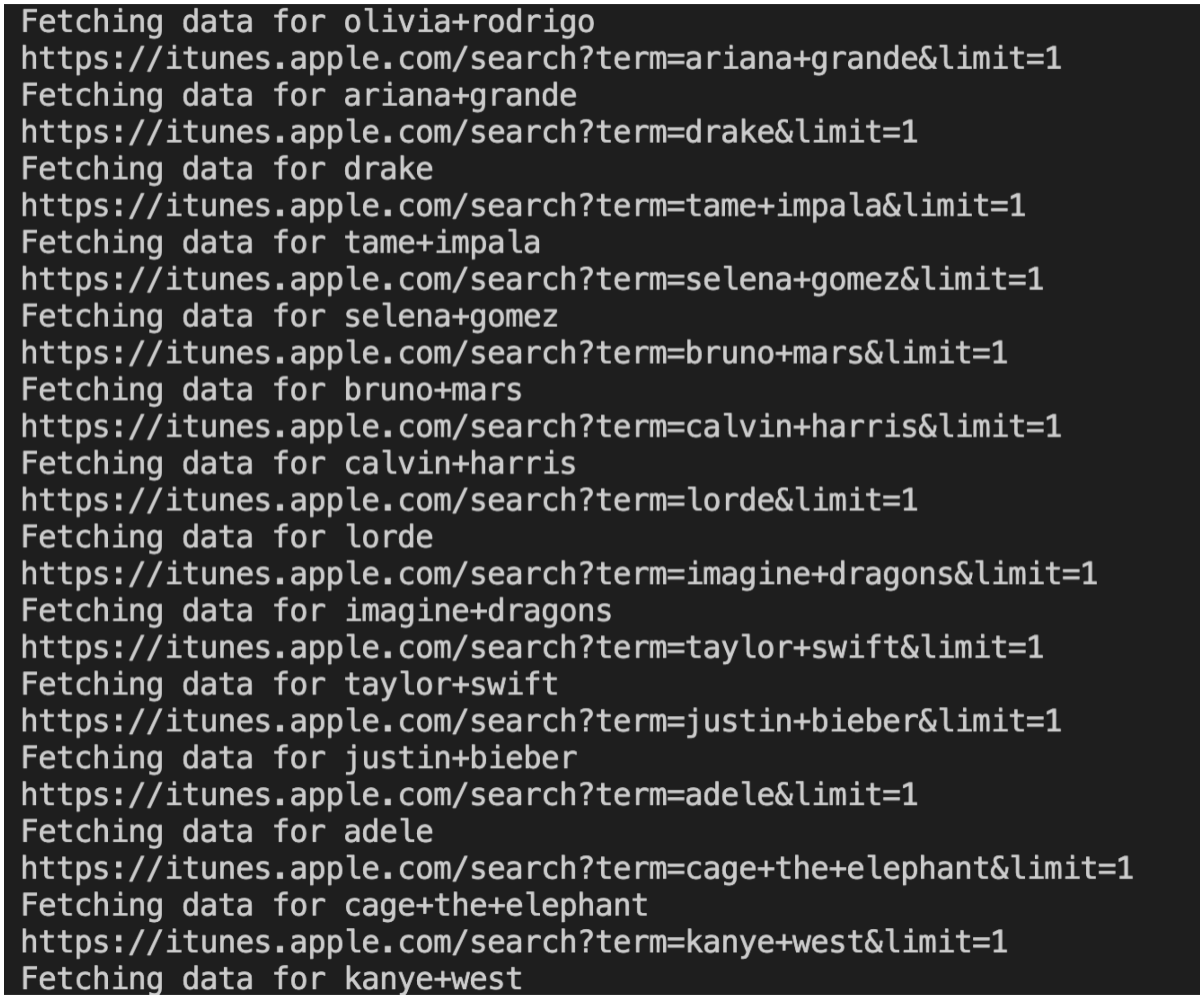
def sort_collectionid(CACHE_FNAME):

This function sorts a list of items on *collectionId* in ascending order and returns the collection price for the item with the smallest *collectionId*.

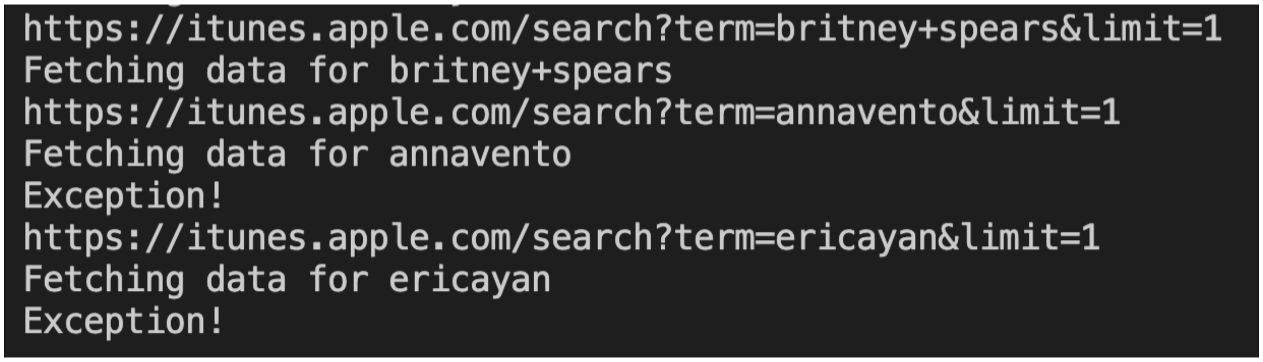
Example Output

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 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…"



```
Fetching data for olivia+rodrigo
https://itunes.apple.com/search?term=ariana+grande&limit=1
Fetching data for ariana+grande
https://itunes.apple.com/search?term=drake&limit=1
Fetching data for drake
https://itunes.apple.com/search?term=tame+impala&limit=1
Fetching data for tame+impala
https://itunes.apple.com/search?term=selena+gomez&limit=1
Fetching data for selena+gomez
https://itunes.apple.com/search?term=bruno+mars&limit=1
Fetching data for bruno+mars
https://itunes.apple.com/search?term=calvin+harris&limit=1
Fetching data for calvin+harris
https://itunes.apple.com/search?term=lorde&limit=1
Fetching data for lorde
https://itunes.apple.com/search?term=imagine+dragons&limit=1
Fetching data for imagine+dragons
https://itunes.apple.com/search?term=taylor+swift&limit=1
Fetching data for taylor+swift
https://itunes.apple.com/search?term=justin+bieber&limit=1
Fetching data for justin+bieber
https://itunes.apple.com/search?term=adele&limit=1
Fetching data for adele
https://itunes.apple.com/search?term=cage+the+elephant&limit=1
Fetching data for cage+the+elephant
https://itunes.apple.com/search?term=kanye+west&limit=1
Fetching data for kanye+west
```



```
https://itunes.apple.com/search?term=britney+spears&limit=1
Fetching data for britney+spears
https://itunes.apple.com/search?term=annavento&limit=1
Fetching data for annavento
Exception!
https://itunes.apple.com/search?term=ericayan&limit=1
Fetching data for ericayan
Exception!
```

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

- 3 points for including the term in the request URL
- 2 points for composing the correct request URL

def test_get_data_with_caching - 35 points

- 5 points for creating request urls
- 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 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_collectionid - 15 points

- 5 points for reading cache file and using only *collectionId* and *collectionPrice* data
- 5 points for sorting the items on the basis of *collectionId* in ascending order
- 5 points for returning the right value

Extra Credit - 6 points

def itunes_list():

This function does not take parameters

The function calls read_cache() to get the iTunes data from ***extra_credit.json***. It analyzes the dictionary returned by read_cache(). This function returns a tuple with two items: the first is a new dictionary with the *primaryGenreName* as the key and number of items with that genre as the value; the second is the genre name with most items.

def itunes_list- 6 points

- 4 points for returning the correct dictionary
- 2 points for returning the correct genre

Expected results should be:

({'Electronic': 1, 'Pop': 6, 'Hip-Hop/Rap': 1, 'Rock': 2, 'Alternative': 3, 'Country': 1, 'Drama': 1, 'Hip-Hop': 1, 'Biographies & Memoirs': 1, 'Dance': 1}, “Pop”)