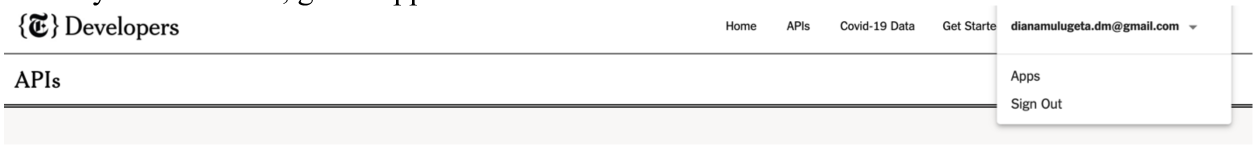
Homework 6: APIs, JSON, and Caching

In this assignment, you will get data using the New York Times Books 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 most_weeks

This assignment requires you to generate an API key. Follow the following instructions to generate one:

- Visit https://developer.nytimes.com/apis
- Create an account
- Under your username, go to Apps



- Create a new App
- Enable the Books API

Assign your API key to the variable API_KEY.

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_json(cache_filename):

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

def write_json(cache_filename, dict):

This function encodes the cache dictionary (dict) into JSON format and writes the JSON to the cache file (cache_filename) to save the search results.

def get_request_url(list):

This function prepares and returns the request url for the API call. The documentation of the API can be found at https://developer.nytimes.com/docs/books-product/1/overview. For this assignment, we will use the path **/lists/{date}/{list}.json**.

We will focus on one parameter:

- *list*: The URL-encoded text string of the name of the best sellers list you want to search for. For example: hardcover-fiction, paperback-nonfiction, e-book-fiction.

Besides the *list* parameter, we will set the *date* parameter to July 10th, 2016 (2016-07-10).

Example of a request URL for hardcover fiction:
https://api.nytimes.com/svc/books/v3/lists/2016-07-10/hardcover-fiction.json?api-key=API_KEY

def get_data_using_cache (list, cache_filename):

This function uses the passed list name to first generate a *request_url* (using the ***get_request_url*** function). It then checks if this URL is in the dictionary returned by the function ***read_json***. If the *request_url* exists as a key in the dictionary, it should print `"Using cache for <list>"` 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 <list>"` and make a call to the **New York Times Books API** to get the data for that specific list.

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

In certain cases, the New York Times Books API may return a response for the *request_url,* but it may not contain any data for the list:

```
{"status":"ERROR","copyright":"Copyright (c) 2022 The New York Times Company.
All Rights Reserved.","errors":["No list found for list name and\/or date
provided.","Not Found"],"results":[]}
```

Do not write this data to the cache file. Print "No list found for list name provided." and return None.

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

def most_weeks(cache_filename):

This function finds the book(s) that has been on the best seller list for the most weeks (hint: look at weeks_on_list) and returns a dictionary with the list name as the key and the title of the book as the value. If there is a tie, return a dictionary with the list name as the key and a list of the titles of the books.

Grading Rubric

def read_json **- 5 points**

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

def *write_json* - 5 points

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

def *get_request_url* - 5 points

- 1 point for including the list in the request URL
- 1 point for including the API key in the request URL
- 3 points for composing the correct request URL

def test_get_data_using_cache **- 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 list provided
- 5 points for adding data to the cache dictionary and cache file only for lists that exist
- 5 points for returning the correct result & type
- 5 points for printing "Exception" if there was an exception and returning "None"

def most_weeks **- 15 points**

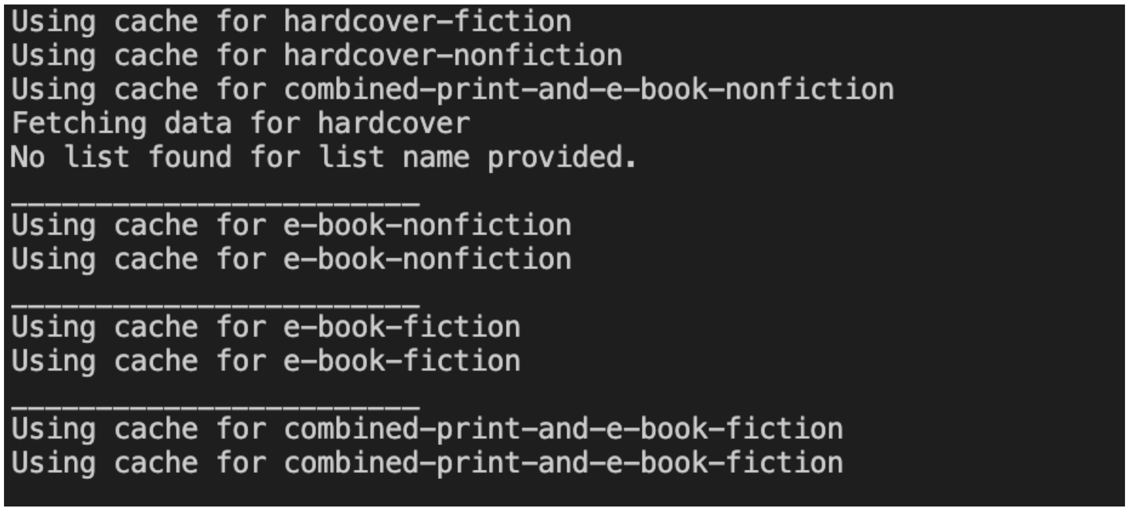
- 10 points for returning the correct book title
- 5 points for constructing a test case for this function in the unittests (this is the only test you will need to write, the rest are provided)

Extra Credit - 6 points

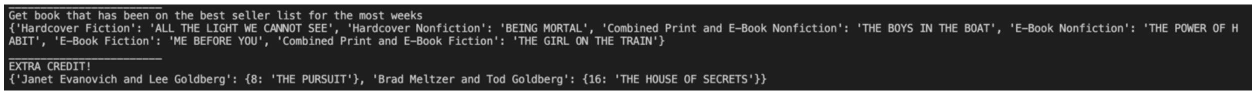
def get_best_seller_brothers(cache_filename):

This function looks for the two brothers that share the last name Goldberg in the bestseller list for Hardcover Fiction and returns a nested dictionary: {authors' names: {rank of the book: title of the book}, …}

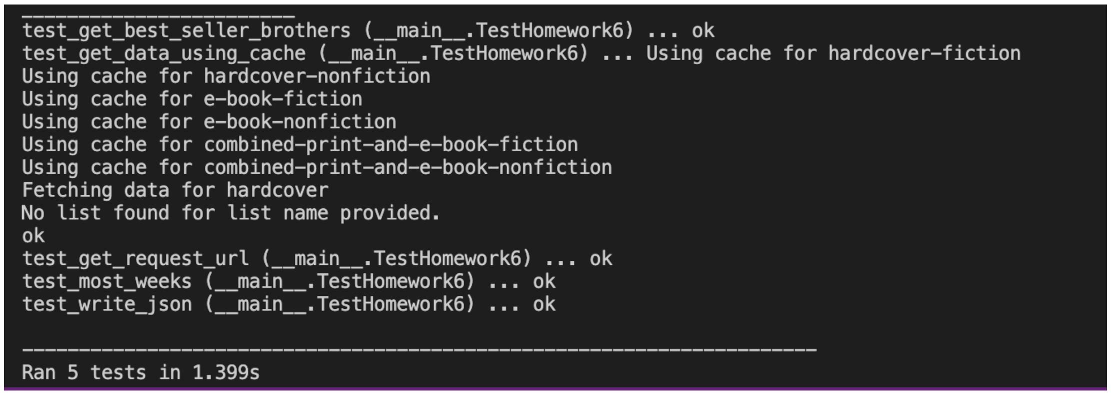
Example Output:



```
Using cache for hardcover-fiction
Using cache for hardcover-nonfiction
Using cache for combined-print-and-e-book-nonfiction
Fetching data for hardcover
No list found for list name provided.
________________________
Using cache for e-book-nonfiction
Using cache for e-book-nonfiction
________________________
Using cache for e-book-fiction
Using cache for e-book-fiction
________________________
Using cache for combined-print-and-e-book-fiction
Using cache for combined-print-and-e-book-fiction
```



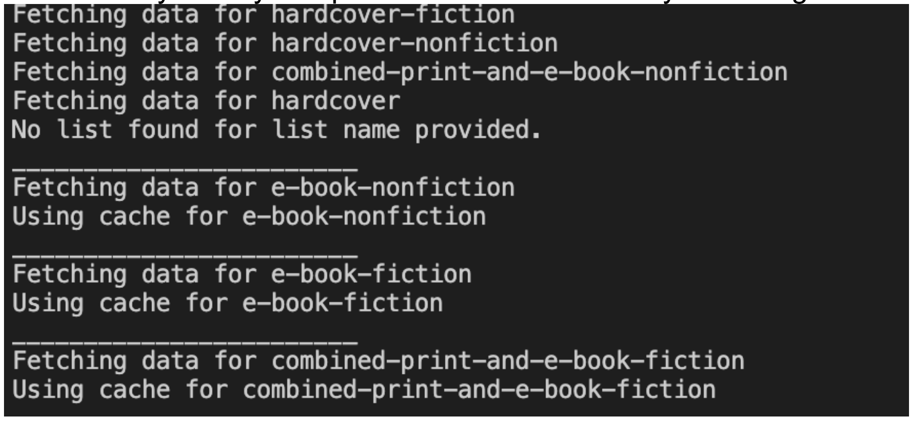
```
________________________
Get book that has been on the best seller list for the most weeks
{'Hardcover Fiction': 'ALL THE LIGHT WE CANNOT SEE', 'Hardcover Nonfiction': 'BEING MORTAL', 'Combined Print and E-Book Nonfiction': 'THE BOYS IN THE BOAT', 'E-Book Nonfiction': 'THE POWER OF H
ABIT', 'E-Book Fiction': 'ME BEFORE YOU', 'Combined Print and E-Book Fiction': 'THE GIRL ON THE TRAIN'}
________________________
EXTRA CREDIT!
{'Janet Evanovich and Lee Goldberg': {8: 'THE PURSUIT'}, 'Brad Meltzer and Tod Goldberg': {16: 'THE HOUSE OF SECRETS'}}
```



```
 ________________________
 test_get_best_seller_brothers (__main__.TestHomework6) ... ok
 test_get_data_using_cache (__main__.TestHomework6) ... Using cache for hardcover-fiction
 Using cache for hardcover-nonfiction
 Using cache for e-book-fiction
 Using cache for e-book-nonfiction
 Using cache for combined-print-and-e-book-fiction
 Using cache for combined-print-and-e-book-nonfiction
 Fetching data for hardcover
 No list found for list name provided.
 ok
 test_get_request_url (__main__.TestHomework6) ... ok
 test_most_weeks (__main__.TestHomework6) ... ok
 test_write_json (__main__.TestHomework6) ... ok

 ----------------------------------------------------------------------
 Ran 5 tests in 1.399s
```

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 hardcover-fiction
Fetching data for hardcover-nonfiction
Fetching data for combined-print-and-e-book-nonfiction
Fetching data for hardcover
No list found for list name provided.
________________________
Fetching data for e-book-nonfiction
Using cache for e-book-nonfiction
________________________
Fetching data for e-book-fiction
Using cache for e-book-fiction
________________________
Fetching data for combined-print-and-e-book-fiction
Using cache for combined-print-and-e-book-fiction
```