**Project Code**

<https://github.com/Quiet-ljb/SI507_fproj>

**Data sources**

1. **Yelp Fusion**

**Origin:** It’s a Web API and requires an API key. The url of this API is <https://api.yelp.com/v3/businesses/search> . The documentation of the API is <https://docs.developer.yelp.com/docs/fusion-intro>.

**Formats:** The format is json

**How to access:** First, I generate an API key according to the description in the documentation. Then I use requests package to access the API based on the format given by the documentation. I adjusted the parameter like “attributes”, “limit” and “sort\_by” to make sure that it returns the most popular restaurant for each city. Caching was used for this dataset.

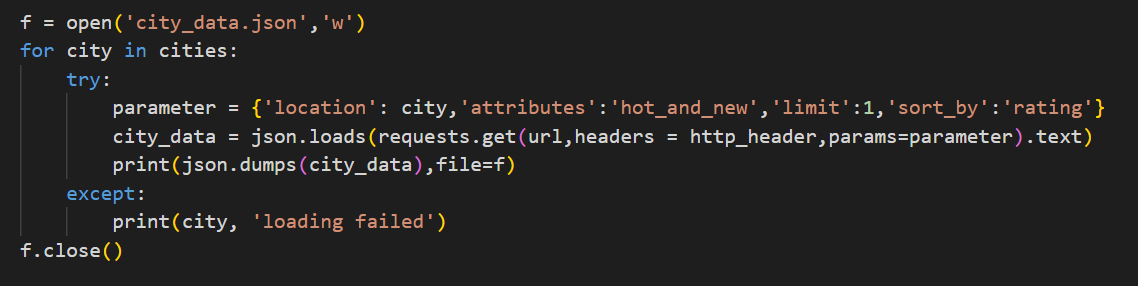
**Summary:** The records available and retrieved are both about 331, which is the number of cities in the U.S. The import data fields are:

**url:** the url of the restaurant

**display\_address:** the address of the restaurant

**display\_phone:** the phone number of the restaurant

**Evidence of caching:** Below shows my code of caching. The data are all stored in a json file.



1. **Web page**

**Origin:** The url of the web page is <https://en.wikipedia.org/wiki/List_of_United_States_cities_by_population>, and there is no documentation.

**Formats:** The format is html

**How to access:** I use beautiful soup to parse and scrape the data, which contains all cities in the U.S. Those cities are not directly stored in the cache since it will be used as the searching query for the first data source. (Actually, it is stored together with the first data source)

**Summary:** The records available and retrieved are both about 331, which is the number of cities in the U.S. Since it’s a web page, and the cities are stored in a table, the important data field is a tag “tr”, which means the table in html. And the first column stores the city name.

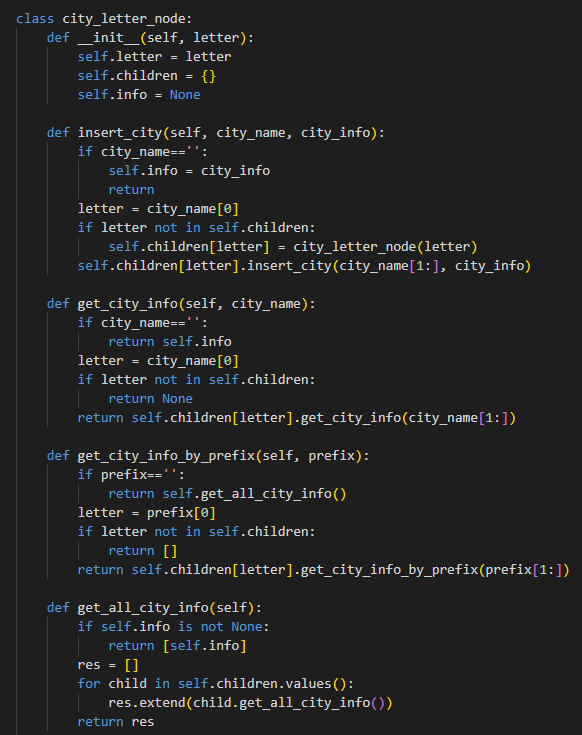
**Evidence:** See evidence of caching above. The first cache already contains all city information (In location field).

**Data structure**

All data was stored in a prefix tree, where all children of a node have common prefix stored in their parent node.

In this project, each node will store a letter and a dictionary. The letter represents part of a city’s name and the dictionary will contain the information of the city. The city name can be got from a node and all its ancestors. If a node and all its ancestors can’t represent a valid city name, the dictionary of this node will be empty.

Below shows the tree and its functions, which includes initialize, insert and search:



**Interaction**

I plan to create a web page which contains a searching box. It will prompt users to type in a city and return the most popular and highest rating restaurant in this city. Also, it will give the link, location and phone number of the restaurant. When the user clicks on the line, the web page will be redirected to the url link.

The technology I plan to use is Flask. It can create a web page as I want and get the information from the web page, so that the program can know the city that the user inputs. Also, it can show data stored in the cache on the web page. When I get the restaurant information, I can display it on the web page by using Flask. The Flask allows me to change the web page by writing python code.