David Nguyen SI 206 Winter 2017 Final Project

This is my final project for SI 206 at the University of Michigan School of Information

Option #2: API Mashup: Twitter & OMDB

This program searches through a list of movies and pulls data from the OMDB API. Also, the top actor is searched on Twitter through the Tweepy API and all of the data is stored into a database. There is some data manipulation also involved with this program and all of the results are outputted into a .txt file.

Getting Started

Ensure that a twitter_info.py file is contained within the same directory and run the file 206_final_project.py

Prerequisites

What things you need to run the program and how to install them

```
Modules to install include: requests, tweepy, sqlite3, regular expressions

Also, it is important to have a twitter_info.py file contining confidential
information for a Twitter account
```

Files Included

```
- 206_final_project.py (Contains the Python Program)
- 206_final_project_cached.json (Contains cached data to run program offline)
- 206_final_project_database.db (Contains the Databases)
- 206_final_project.txt (Text File)
```

Functions

These are the following functions that are contained in 206_final_project.py

```
def get_OMDB_WithCaching():

Input: baseURL and Parameters
Behavior: Checks if the search query is in the cached file or makes a request to
the OMDB API and then stores it within the cached file
Returns: JSON Object from OMDB API
```

```
def get_OMDB_data()
Input: movie_title (represents a movie as a string)
Behavior: Creates the URL for the API calling and calls get_OMDB_WithCaching
```

```
function
Returns: JSON Object from OMDB API
```

```
def get_twitter_handle
Input: search query
Behavior: Makes a call to the Tweepy API and caches it. Obtains the twitter handle
from the search query
Returns: Twitter Handle
```

```
def searching_twitter()
Input: search query
Behavior: Makes a call to the Tweepy API and caches it. Obtains the Twitter Search
results in a JSON Format
Returns: JSON Object from Tweepy API
```

```
def twitter_user_info()
Input: twitter_handle
Behavior: Makes a call to the Tweepy API and caches it. Obtains the Twitter User
Information results in a JSON Format
Returns: JSON Object from Tweepy API
```

```
def uploading_databases()
Input: none
Behavior: using all of the information made from the classes, this function
iterates through all of the moves in 'list_of_movies' and creates instances of them.
Those instances are created into tuples which are then appended to a list to be
used for database uploading later in the program.
Returns: none
```

```
def unique_handle()
Input: a list
Behavior: checks of the handle is within another list to prevent issues when
uploading as a PRIMARY KEY to the database
Returns: none
```

Classes

Below are the following classes that are contained in 206_final_project.py

```
class Movie(object):

Each Instance Represents a Single Movie
Required Constructor Input: Dictionary from OMDB API

Methods:

get_specific_rating():
Behavior: Takes all of the ratings and zips them into a dictionary where it can be
```

```
used to obtain specific ratings from particular sources
Returns: Dictionary

get_rotten_tomato_rating():
Behavior: takes the dictionary from get_specific_rating and extracts the rotten
tomato rating
Returns: Rotten Tomato Rating

get_top_actor():
Behavior: Splits the list of actors from constuctor.
Returns: Top Actor (first actor in list)

get_top_actor_twitter_handle() & get_director_twitter_handle()
Behavior: Takes in the name and searches twitter for their twitter handle
Reuturns: Twitter Handle

get_movie_id():
Returns: movie_id

__str__():
Returns: A string with some important information about the specific movie instance
```

```
class Tweet(object)

Each instance represents a single Twitter Search
Constructor takes in a dictionary object from the Tweepy API.
List comprehension is used to obtain instance variables that will be zipped
together later in the code for database uploading
```

```
class TwitterUser(object)

Each instance represents a single Twitter User and their coresponding information
Constructor takes in a dictionary object from the Tweepy API.
```

Databases

There is one database file in this program which is named 206_final_project_database.db

The database has three tables: Movies, Tweets, Users

```
Movies Table:
Each Row represents a single Movie

Each row contains a movie_id, title, genre, plot, top_actor, top_actor_twitter,
director, director_twitter, and rotten_tomato_rating
```

```
Tweets Table:
Each Row represents a single Tweet

Each row contains a tweet_id, screen_name of the person who tweeted it, user_id,
number of favortites, number of retweets, text of the tweet, the movie_id which
```

```
references the Movies table, and the search querty which references the top actor
in the Movie
```

```
Users Table:
Each Row represents a single Twitter User

Each row contains a user_id which can referneces the Tweets table, screen_name,
their number of followers, their number of favorites, and their twitter
description/bio
```

Data Manipulation

My code also does some data manipulation that is outputed onto a .txt file It's useful becuase it does some really cool things wiht database queries and also can present some interesting things :)

```
#1: List of all words in Tweets

There is a list of all the words of that are contained in the Tweets
```

```
#2: Most Common Word in Tweets

This takes in the all of the tweets that were obtained by the twitter searching and
find the most common character in ALL of the tweets and how many times it occurs
```

```
#3: Sorting:

All of the movies have a Rating from Rotten Tomatoes and the one with the highest
is returned and outputted in the text file
```

```
#4: Most Common Word in Plot:

All of the movies have a plot and all of the plots are parsed and the top two
common words are outputted with the number of times they occur. You may be
surpirsed to see the results :)
```

Why This Project?

I have had the great pleasure of working with various APIs throughout my time coding here as a UMSI student and I am really interested in how to manupulate data to do something really cool. As this will be my final coding class as a BSI student, I am glad to have made a final product!

SI 206 Specific Needs

Below are the Specific Lines where certain elements are to begin:

- List of Movies: 307

- Data Gathering Functions: 29 - 161 + 318 - 356
- Class Definitions: 166 - 304
- Creation of Database: 405 - 430
- Uploading of Database: 435 - 459
- Data Processing: 463 - 515
- Creating Output File: 519 - 575
- Testing Code: 583 - 670

Built With

- Sublime Text
- Python 3
- Many Hours of Hard Work

Authors

- **David Nguyen** - *Author* - GitHub

Acknowledgments

- Thank you to Jackie Cohen and her amazing teaching skills and dedication to her students
- Shoutout to the SI 206 Winter 2017 GSI for being so welcoming
- Thank You to the School of Information for creating such a great undergraduate program!
- The Mason Hall Computing Center for being open 24/7 for my late night coding needs