iTunes Proj - Interactive Command Line Search Tool

Objectives

- Design a program using object-oriented programming, including class inheritance
- Gain experience working with JSON data from a web API

Overview

In this project, you will build a simple command line tool for searching the iTunes store. You will build this application in a series of steps.

Knowledge Required

There are a number of topics you'll have to master to complete this assignment. Most of them are things you've seen before but may need to recall. Some of them are new this semester. And a couple are brand new, but easy to master with a pointer or two.

The following are topics we've covered in 507 this semester:

- Obtaining, validating, and processing user inputs
- Functions
- Classes and inheritance
- Web APIs, JSON, and caching

The following are covered in the textbook, and you have seen them to at least some extent in either 506 or 507.

- String manipulation (e.g., extracting substrings)
- json parsing using the json module
- Accessing Web APIs using requests
- Writing interactive command line programs (prompting for input, allowing user to quit)

The following haven't been covered in 506, but you'll learn them through this project:

- Named parameters with default values
- The webbrowser module (new, but super easy: you just call webbrowser.open(URL))
- Importing self-written modules and the meaning of __name__ == "__main__"
- Understanding the iTunes API (using documentation and "reverse engineering" of json output)

Before you get started

We have given you three files to start with: iTunesStarterCode.py, iTunesTestCode.py, and sample_json.json.

Here's how you will use them:

- iTunesStarterCode.py is where you will define your classes and write any functions you need to run your program. You will also write your interactive code for Part 4 in this file, in the designated code block.
- Sample_json.json provides sample output from the iTunes web API that you will need for Part 3.
- iTunesTestCode.py contains test cases for Part 1 and 2. We'll talk about running test cases in class and discussion sections. Essentially, this is a helper file that checks if your classes can instantiate objects that meet the requirements. You should run the test cases for Part 1 and 2 only when you're completely done with the respective parts. If you're done with Part 1 and would like to test it out, you can comment out the test cases for Part 2 and run. How to run the test cases:
  - Put iTunesStarterCode.py, sample_json.json, and iTunesTestCode.py in the same folder.
  - Go to your Terminal, cd into the directory that contains the three files, and run:

```
$ python iTunesTestCode.py
```

or if you have two versions of Python installed on your machine:

```
$ python3 iTunesTestCode.py
```

Part 1: Implement a class system

In this step, you will implement a set of classes to model a part of the iTunes data model.
Develop a base class (“Media”) and two subclasses (“Song” and “Movie”) that represent items in the iTunes store (there are other media types that you do not have to worry about).
Here are some details about each class you’ll need to implement:

Media:

- Instance variables: title (default value: “No Title”), author (default value: “No Author”), release_year (default value: “No Release Year”), url (default value: “No URL”)
- Methods:
  - __init__: takes title, author, release_year, and url as parameters. Use named parameters with defaults.
  - info(): returns “<title> by <author> (<release year>)”, filling in the appropriate instance variables. For example, “Bridget Jones's Diary (Unabridged) by Helen Fielding (2012)”
  - length(): returns 0

Song (subclass of Media):

- Additional instance variables: album (default value: “No Album”), genre (default value: “No Genre”), track_length (default value: 0)
- Methods:
  - __init__: takes title, author, release_year, url, album, genre, and track_length as parameters. Use named parameters with defaults. Call super( ) to initialize variables that belong to Media
  - info(): add “[<genre>]” to the end of the output from Media.info( ). For example “Hey Jude by The Beatles (1968) [Rock]”
  - length(): return track length in seconds (rounded to nearest second)

Movie (subclass of Media):

- Additional instance variables: rating (default value: “No Rating”), movie_length (default value: 0)
- Methods:
  - __init__: takes title, author, release year, url, rating, and movie_length as parameters. Use named parameters with defaults. Call super( ) to initialize variables that belong to Media.

- info(): add “[<rating>]” to the end of the output from Media.info( ). For example “Jaws by Steven Speilberg (1975) [PG]”
- length(): returns movie length in minutes (rounded to nearest minute)

Assessment

You will be assessed on:

- Whether you have correctly implemented the Media, Song, and Movie classes that have all aforementioned instance variables and methods
- Whether you have used inheritance and used super( ) correctly to avoid repeating code

Notes

- For this part, you will be creating objects “by hand,” i.e., by passing explicit information into the constructors. You will not be creating objects using JSON. That will be covered in the next part.
- You’ll see in Part 2 that song track length and movie length in the raw JSON is in milliseconds. Your track_length and movie_length instance variables should save those raw values in milliseconds, and the length() methods for the Song and Movie classes should return song track length and movie length in seconds and minutes respectively. That is, the length() methods will be responsible for some conversions from milliseconds to seconds or minutes.

Part 2: Create objects from JSON

Now add the ability to create objects using JSON. We have provided sample JSON for three media types that iTunes supports. You will show that you can correctly parse each of these JSON objects into properly constructed objects of the correct type.

To avoid repeating code, you should make sure that the Media class does as much of the parsing as possible, and that each subclass only parses information that is specific to that class.

Suggested approach: Add a named parameter ‘json’ to each constructor. This should have a default value of None. Depending on the value of json, either create the object from the json or from the explicit parameters used in Part 1. (When json is not None, create objects from json. Otherwise, create objects from explicit parameters).

Assessment

You will be assessed on

- Whether your classes can create objects from both json and explicit parameters.
- Whether your classes can parse json and instantiate correctly from json when json is provided to the constructor.
- Whether you have used super( ) correctly to avoid repeating code

Part 3: Create objects from iTunes API

Add the ability to fetch data from the iTunes API, and create lists of objects from the data retrieved. Since the data may change between calls, you will only need to show that the data returned from a set of pre-defined queries (of your choice) is processed by your program without errors, and that the number of objects created is within an expected range (e.g., either 0 or "more than zero but less than or equal to the number of results requested").

Assessment

You will be assessed on

- Whether you successfully fetch data from the iTunes API
- Whether you construct the correct type of object given the contents of a JSON object

Notes

The iTunes API limits you to "about 20 calls per minute." If you don't make more than 20 API calls in a minute, you should be fine. If this turns out to be a problem, you might consider caching results to use offline while debugging. We will work on caching next week, the week of February 4th. You may refer to the lecture notes and try to adapt and use here. Caching is not required for this assignment and not even particularly recommended unless you are very comfortable with it.

Part 4: Create an interactive search interface

For this last part, you will add the ability for users to enter their own queries and receive nicely formatted results. The output should be formatted as follows:

- The results should be grouped into Songs, Movies, and Other Media. In each category, the results should be printed using their respective info() method, one per line.
- Each result should be preceded by a number, starting at 1 and going up, printed at the beginning of the line.

When the program first runs, the user should be presented with two options: enter a search term, or enter exit to quit. After a query has been run, a third option becomes available: launch preview. To launch a preview, the user enters the number of the result they want to preview. Your program will then use the webbrowser module to open the trackViewURL embedded in the JSON item description, while also printing the URL to the screen.

Assessment

You will be assessed on:

- Whether your program returns expected results (we will compare output with a reference implementation we have built and see if the results are the same)
- Whether the results are grouped and formatted appropriately
    - Whether the preview functionality works
- Partial credit is possible if you are only able to get parts of this working

Part 4 Sample Output

```
Enter a search term, or "exit" to quit: Beatles

SONGS
1 Hey Jude by The Beatles (1968) [Rock]
2 Yesterday by The Beatles (1965) [Rock]
...

MOVIES
24 Help! by Richard Lester (1965) [G]
25 Yellow Submarine by George Dunning (1967) [G]
...

OTHER MEDIA
47 The Beatles by Hunter Davies (2009)
48 Dreaming the Beatles by Rob Sheffield (2017)
...

Enter a number for more info, or another search term, or exit: 2

Launching
https://itunes.apple.com/us/album/hey-jude/400835735?i=400835962&uo=4
in web browser…

Enter a number for more info, or another search term, or exit: exit

Bye!
```

Notes

- What's shown above is hypothetical output--not generated by actual code. Your results will almost certainly be different--this is just intended as an example of the format and interaction.
- The "..."s that show up above would NOT be part of your program output. We just didn't want to fill up a lot of space with lots of search results.
- There should be no repeated numbers--start with 1 and increment the number for each result.
- Your results should be grouped into SONGS, MOVIES, and OTHER MEDIA, and these categories should always appear in the same order
- Where it says "Launching XXX in web browser..." above you should actually launch the specified URL in a web browser.

- You will need to test if the user's input is a number (in which case you'd launch the browser) or a string (in which case you'd do a new search). The input '3' would launch the browser, since it can be converted to a number. The input '2 Live Crew', however, should be treated as a string. isnumeric() and int( ) are your friends here.
- You will have to think about what to do when there are no results in a category, or no results at all! Your program should give output that will inform the user of what has happened in such cases.

Rubrics

Req	Description	Part	Category	Point Value
1	Media class is defined with all required instance variables	1	Code	4
2	Media class's `info()` method returns well-formatted string that follows instructions	1	Code	5
3	Media class's `length()` method returns 0	1	Code	1
4	Song class inherits from Media class and is defined with all required additional instance variables	1	Code	5
5	Song class's `info()` method returns well-formatted string that follows the instructions	1	Code	5
6	Song class's `length()` method returns track length in seconds	1	Code	5
7	Movie class inherits from Media class and is defined with all required additional instance variables	1	Code	5

8	Movie class's `info()` method returns well-formatted string that follows the instructions	1	Code	5
9	Movie class's `length()` method returns movie length in minutes	1	Code	5
10	Uses inheritance to avoid repeating code	1	Code	5
11	Your Part 1 code passes Part 1 test cases	1	Behavior	5
12	Media class can successfully parse JSON and construct Media objects	2	Code	10
13	Song class can successfully parse JSON and construct Song objects	2	Code	10
14	Movie class can successfully parse JSON and construct Movie objects	2	Code	10
15	Uses inheritance - Media class should handle as much parsing as possible	2	Code	10
16	Media, Song, and Movie classes can instantiate objects from both explicit parameters and JSON	2	Code	5
17	Your Part 2 code passes Part 2 test cases	2	Behavior	5
18	Fetch data correctly from iTunes API based on queries. We will run a few sample queries and compare the data you retrieve with that of our reference implementation.	3	Code	20
19	Your code can construct the correct type of object given the contents of a JSON object from iTunes	3	Code	10

20	Objects created from the fetched data are sorted into three groups (i.e. lists) - other media, songs, and movies (this will help with Part 4)	3	Code	10
21	On program start, user is prompted with two options: enter a search term or type “exit” to quit. If user enters a search term, the program makes a request to iTunes API and fetches data according to user queries. If user enters “exit”, the program terminates.	4	Behavior	5
22	The data displayed is nicely formatted and is grouped into Songs, Movies, and Other Media. In each category, the results should be printed using their respective `info()` method, one per line (see sample output).	4	Behavior	10
23	Each result should be preceded by a number, starting at 1 and going up, printed at the beginning of the line (see sample output).	4	Behavior	5
24	After the first query is run, besides entering another search term or exiting, user should have a third option to launch preview by typing in the number preceding the result.	4	Behavior	5
25	If user enters a number within the range of number of results returned, your program can launch preview using webbrowser module.	4	Behavior	5
26	If user enters something else, it	4	Behavior	5

	should be treated as a search term, and another query would be run and results based on the search term would be displayed.			
27	Program keeps prompting user to either enter a search term, enter a number within range to launch preview, or enter “exit” to quit until user quits (i.e. it shouldn’t just run once)	4	Behavior	5
28	User should be able to quit the program at any time by entering “exit”	4	Behavior	5
29	Your program should be able to handle any queries and parse returned data gracefully without throwing errors (hint: each JSON object returned from iTunes may have slightly different structures and some keys could be missing. How do you handle KeyError?)	4	Behavior	5
30	Well-constructed code that follows our guidelines	ALL	Code	5
31	Program uses best practices for defining and using functions (hint: Parts 3 and 4 should be solved using at least a few functions)	ALL	Code	5
	Total			200