Objectives

- Design a program using object-oriented programming, including class inheritance
- Develop unit tests to verify a program works as specified
- 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:
- Classes and inheritance
- "Special" class methods like __str__( ), __repr__( ), and __len__( )
- Unit testing & test cases

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)
- Named parameters with default values
- 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 either 506 or 507, but we'll go over them briefly in class:
- 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: `proj1_w18.py`, `proj1_w18_test.py`, and `sample_json.txt` or sample_json.json. Here's how you will use them:

- `proj1_w8.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. There should be no test cases in this file, and there should be no output in the version of this file that you turn in that isn't related to Part 4.
- `proj1_w8_test.py` is where you will define your test cases for Parts 1-3. This is the file that you will *run* while working on Parts 1-3. Note that this file *imports* proj1_w8.py and can therefore instantiate classes and invoke functions from that file. When grading your project, the graders will run `proj1_w8_test.py` to grade Parts 1-3, and will run `proj1_w8.py` to grade Part 4.
- `Sample_json.txt` provides sample output from the iTunes web API that you will need for Part 3.

Part 1: Implement and test a class system (50 points)

In this step, you will implement a set of classes to model a part of the iTunes data model and you will write unit tests to show that the class implementation is correct.

Create 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, author, release year
- Methods: implement the following:
    - __init__: takes title, author, and release year as arguments. Use named arguments with defaults.
    - __str__: returns "<title> by <author> (<release year>)", filling in the appropriate instance variables. For example "Bridget Jones's Diary (Unabridged) by Helen Fielding (2012)."
    - __len__: returns 0

Song (subclass of Media):
- Additional instance variables: album, track length
- Methods:
    - __init__: takes title, author, release year, album, genre, and track length as arguments. Use named arguments with defaults. Call super( ) to initialize variables that belong to Media
    - __str__: add "[<genre>]" to the end of the output from Media.__str__( ). For example "Hey Jude by The Beatles (1968) [Rock]"
    - __len__: return track length in seconds

Movie (subclass of Media):
- Additional instance variables: rating, movie length
- Methods:

- __init__: takes title, author, release year, rating, and movie length as arguments. Use named arguments with defaults. Call super( ) to initialize variables that belong to Media.
- __str__: add “[<rating>]” to the end of the output from Media.__str__( ). For example “Jaws by Steven Speilberg (1975) [PG]”
- __len__: return movie length in minutes (rounded to nearest minute)

Test cases

Create test cases that show:

- All class constructors work as specified, and correctly populate instance variables
- __str__ and __len__ work properly for all three classes
- Classes do NOT have instance variables that are not relevant to them (e.g., Songs and Media do not have a ‘rating’ instance variable)

You must implement at least 3 test functions, and include at least 15 assertions (total, across all test functions). It’s fine to include more of either.

Assessment

You will be assessed on

- Whether your tests cover the criteria listed above
- Whether your tests pass
- Whether you have 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.

Part 2: Create objects from JSON (50 points)

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, so as not to break your Part 1 tests. Depending on the value of json, either create the object from the json or from the explicit parameters used in Part 1.

Test Cases

Create test cases that show:

- Objects of each class (Media, Song, Movie) are created correctly from the relevant JSON string. “Created correctly” means that instance variables are set to correct values and class methods (eg., __str__, __len__) behave correctly.

You must implement at least one test function (three would be reasonable too), with at least 15 assertions to test that all relevant instance variables are created as specified.

Assessment

You will be assessed on

- Whether your tests cover the criteria listed above
- Whether your tests pass
- Whether you have used super( ) correctly to avoid repeating code

Part 3: Create objects from iTunes API (50 points)

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 (either 0 or “more than zero but less than or equal to the number of results requested”).

Test Cases

Create test cases that show that your program responds within expected ranges to a several diverse queries, including common words (“baby,” “love”), less common words that are likely to produce specific matches (“moana,” “helter skelter”), nonsense queries (“&@#!$”), and a blank query.

Assessment

You will be assessed on

- whether your tests cover the criteria listed above
- whether your tests pass
- 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 any of your test cases (and you don’t need to make nearly that many), you should be fine. If this turns out to be a problem, you might consider caching results to use offline while debugging. If you took 506 previously you may have access to caching code that you can 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. We will work on caching later in the semester.

Part 4: Create an interactive search interface (50 points)

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 __str__ representation, 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.

Test Cases

You do not need to implement test cases for Part 4. We will test this part of your code manually.

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

```
$ python proj1_w18.py

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!
```

A few 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. int( ) and str( ) 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.