Searching and Summarizing IMDB Data

This homework assignment will assess your ability to use the text file processing techniques and data structures that we learned over the past few chapters. You will write a Python program that is capable of parsing a large data file into Python data structures that allow for efficient operations. This will allow you to put your newly acquired Python skills to use on a practical problem using real-world data.

[Due: Wednesday, November 1, 2017 at 11:59PM](#_4eoh90ydkifr)

[Background:](#_humrnmlcd46g)

[Deliverable](#_c42eoob34rba)

[Setup:](#_vzedddg0avev)

[build\_indexes](#_40yr7szdcdms)

[Testing with a Smaller File](#_va4h9l9bbu2n)

[search\_for\_title](#_guh799tps3e1)

[movies\_for\_actor](#_l1h0498c19ym)

[Sorting the data](#_i4ylynj7mbg1)

[Grading Rubric](#_qw2gncxfr0ip)

[Bonus Opportunity](#_wsl890ip6dkj)

# Background:

IMDb (Internet Movie Database) is the de facto source for information about films, TV, and video games. Nearly every has asked the question "who was in that one movie with so-and-so?" at one point, and IMDb is often the the best source for the answer. The full dataset contains more than 4 million titles and almost 8 million individuals. We'll be working with a subset of that that's still quite sizable. There are more than 1.5 million rows in this file (it's about 75MB), so your program will need to process quite a bit of information. Thus, it's going to be vital to choose the right data structures to represent the information that your program is going to need to work with.

# Deliverable

You should submit a single Python program file to CourseWeb for this assignment. Please name your file hw3\_<your\_pitt\_username>.py. Please read the instructions carefully to ensure that your program meets all requirements. If you are uncertain as to what the assignment's specifications are, please confirm with your instructor or a TA.

# Setup:

1. To get started, download and unzip the [hw3\_artifacts.zip](https://docs.google.com/uc?id=0B0ghZ7_iC3k5R0Q2NWJlcnpTWHM&export=download) file from Drive. You'll want to unzip its contents to a new folder. If you aren't sure how to unzip, do a Google search, ask me, or one of your TAs.
2. The included [hw3\_starter.py](http://tinyurl.com/jmypzwz?path=hw3/hw3_starter.py) file gives you a good starting point to work from where your task will be to implement the functions described below. You should rename this file to hw3\_<your\_pitt\_username>.py after downloading it.
   1. Don't make any modifications to the code that was already written for you. If you do, you will likely run into problems that will be difficult to debug.
3. After downloading and renaming your Python starter program, you should be able to run it. None of the functions do anything, so you will need to complete the implementations of the following functions outlined below.

# build\_indexes

When dealing with a large dataset, programmers often need to build data structures that organize, or index, the data according to the way that users will access it. Imagine how difficult it would be to find a word from a dictionary of English words if it wasn't organized alphabetically and you had to search through the entire contents each time? That's what we'll be doing here.

The main function calls build\_indexes when the program starts and passes in two empty dict objects. This is so we can process the data from the file and put it into Python's memory. Subsequent operations will then be much faster than if we were to try and read the file from start to finish each time the user wanted to do something.

## Testing with a Smaller File

You will probably want to download and test the function with the smaller, imdb\_data\_sample.tsv that's only 10,000 lines. This will allow you to make sure your algorithm is correct before moving to the full 75MB file, which takes substantially longer to process. You can do this by changing the name of the IMDB\_FILE variable at the top of your program.

The code you implement inside of build\_indexes should do the following.

1. Open up the imdb\_data.tsv file that you should have downloaded. You can use the module constant IMDB\_FILE.
2. Keep a count of the number of lines you process. You will need to return this value when the function completes.
3. Each line of the file contains 6 data fields separated by the tab (\t) character. These present the year the movie was released, its title, the name and gender of one of its actors or actresses, and the name of the character the actor/actress played. There will be multiple entries per movie since one movie will have many cast members.
   1. year, title, first\_name, last\_name, gender, character
   2. year is an integer and should be converted to the appropriate Python type.
4. You should build a dict object from the fields parsed out of each line of the file. Remember to strip any new line characters before you split the data into its fields. Your dict object's keys should get logical names that reflect the data they contain (ie. 'year' for the year value).
   1. You must combine the first\_name and last\_name data fields into a single data field called name inside of your dictionary. This will be simpler to use then having two separate name fields.
5. Each line of data contains a unique set of values. The titles\_index and actors\_index objects that we passed in need to be constructed so that they make it easy to lookup a list of actors/actresses that starred in a particular movie *title*, or a list of movie titles that a given actor/actress starred in.
6. For each *title* you encounter, you should create a key for it in the titles\_index dict. Its value should be a list of the dict objects you created for each row matching that title.
   1. NOTE: Your program should permit lookups to be case-insensitive. A user should be able to type 'Mary', 'MARY', or 'mary' and know how many children named 'Mary' were born each year. Only the keys should be case-insensitive, keep the names that you gather from the file in their original form.
7. Once you've processed all of the rows from the file, return the number of lines that were processed.
8. Test your program at this point. It will take a few seconds to run. If you've done things correctly, you should see that your program processed 1,858,689 rows in a few seconds, and is using a couple of MB of memory to store the data structures you populated.

# search\_for\_title

To implement the search\_for\_title function, you need to do the following:

1. Prompt the user to type a movie title.
2. Check to see if that title is found in the database you passed in via the titles\_index variable.
3. If the movie title is found, you'll want to print output that looks like the sample shown below for the movie Deadpool.
   1. Partial sample output for the title Deadpool is shown below:

Please type an option from the list above:

>>> 1

Type a movie title: deadpool

"Deadpool" was released in 2016

Aatash Amir played the character "Whisper Boyfriend"

Anthony J Sacco played the character "Thug #3"

Ben Wilkinson played the character "?"

Brad Archie played the character "Hospital Doctor"

Brianna Hildebrand played the character "Negasonic Teenage Warhead"

Bronwyne Sloley played the character "Fight Club Spectator"

...

1. If the title is not found, you should print a message like this:

Please type an option from the list above:

>>> 1

Type a movie title: Bad Movie

Sorry, "Bad Movie" could not be found

1. Test your program with menu option 1. Make sure it works as expected when trying some names you would expect to be found, and some that you would not. Make sure it works correctly regardless of the case used.

# movies\_for\_actor

To implement the movies\_for\_actor function, you need to do the following:

1. Prompt the user to type the name of an actor or actress.
2. Check to see if that name is found in the database you passed in via the actors\_index variable.
3. If the name is found, you'll want to print output that looks like the sample shown below for Tom Hanks.
   1. Sample output for Tom Hanks is shown below:

Please type an option from the list above:

>>> 2

Type an actor or actress name: Tom Hanks

Played "Eamon Bailey" in The Circle (2017)

Played "Alan" in A Hologram for the King (2016)

Played "Himself" in California Typewriter (2016)

Played "Waitstill Sharp" in Defying the Nazis: The Sharps' War (2016)

Played "Robert Langdon" in Inferno (2016)

Played "Chesley 'Sully' Sullenberger" in Sully (2016)

Played "Himself" in Tom Hanks: A League of His Own (2016)

Played "James B. Donovan" in Bridge of Spies (2015)

Played "Himself" in Everything Is Copy (2015)

Played "Mr. Macauley" in Ithaca (2015)

Played "?" in Good Morning Britain (2014)

Played "?" in Stop at Nothing: The Lance Armstrong Story (2014)

1. If the name is not found, you should print a message like this:

>>> 2

Type an actor or actress name: Jason Prodonovich

Sorry, "Jason Prodonovich" could not be found

1. Test your program with menu option 2. Make sure it works as expected when trying some famous actors/actresses you know.

# Sorting the data

In the starter code that you downloaded, you should notice the presence of two functions that were written for you, sort\_by\_name and sort\_by\_year. These are going to be used to make your program's output more sensible and easy for the user to interpret. The [sort()](https://docs.python.org/3.5/library/stdtypes.html#list.sort) method that we've been using for lists accepts some additional arguments. For lists that store complex data structures like dicts, we need to tell Python what to use to compare data members to each other.

In the search\_for\_title function, we want the output of actors to be in ascending order by name. Read the documentation for the [sort()](https://docs.python.org/3.5/library/stdtypes.html#list.sort) method and pass the name of the appropriate function to correct argument to sort the data by year before displaying it.

In the movies\_for\_actor function, we want the output of name occurrences to be in descending order from most recent year. This time, you'll want to choose the appropriate sort key and make sure the list is sorted in descending order.

Once you have implemented these changes to your search\_for\_title and movies\_for\_actor functions, you'll have a useful, working program that you can use to help analyze a large dataset.

# Grading Rubric

This assignment will be worth **50pts** (remember all homework assignments collectively make up 25% of your overall grade).

* 15 pts for correct logic and output from the build\_indexes() function.
* 15 pts for correct logic and output from the search\_for\_title() function.
* 15 pts for correct logic and output from the movies\_for\_actor() function.
* 5 pts for proper use of Python style conventions and correct output on all test inputs, etc.
  + Refer to your book, our lecture nodes, and the [official Python style guide](https://www.python.org/dev/peps/pep-0008/).

# Bonus Opportunity

There are a lot of interesting enhancements you can imagine for this program. You have the opportunity to earn up to 5 bonus points on this assignment if you implement a properly working enhancement that goes beyond the basic requirements for the program. Note that any enhancement you provide must not prevent the program from working as specified in these instructions. Here are some ideas, but feel free to think of others and run them by your instructor.

* Have the movies\_for\_actor function ask the user to view the full cast from one of the movies listed for a particular actor without having to go back to the main menu and do a full search.
* Have the search\_for\_title function ask the user for a gender restriction. If they say yes, apply the restriction and only show male or female cast members.
* Build an additional index that allows for an alternative type of search, perhaps movies for a certain year, or to be able to fill out part of the title.