CS 100 Programming I
Project 1

Revision Date: October 2, 2015

Preamble

You may develop your code on your anywhere, but you must ensure it runs correctly on a Linux distribution before submission.

Movie Database

All the world's a stage – William Shakespeare, in *As You Like It*

How much digital data will the world create and store this year? Some place that number over 2 zettabyes! That's 21 zeros and is roughly the amount of storage of 57.5 billion 32GB Apple Ipads. Furthermore, the amount of data created and stored doubles every two years. How do we store and access this information? Usually with storage software known as a *database*. Databases are implemented everywhere from web servers to automobiles, video games, smartphones, etc. The goal of this project is to implement a simple Python database and query system.

Tasks for your program:

- Display Movies
- Display longest movie
- Display shortest movie
- Display movies earlier than specified year
- Display movies later than specified year
- Display movies with specified rating
- Add Movie
- Delete Movie
- Save database

Program behavior

Your program should use a menu driven interface that allows the user to interact with one table in a database (see *Reading a database* for more information on a table). Once your program is launched, it should read in the file of movie information and then display the following menu:

```
Welcome to the Python Movie Database!
Main menu:
    1:  display all movies
    2:  display shortest movie
    3:  display longest movie
```

```
        4:  display older movies
        5:  display newer movies
        6:  display movies by rating
        7:  add movie
        8:  delete movie
        9:  save database
        0:  quit the program
    Enter option number:
```

The menu code (ideally a function call), should be placed in a loop. Only when the user enters a 0 should the loop be exited.

Option 1: Display all movies

Entering "1" will display the entire contents of the movie database. Based on the test *data* file, when you select the first option, your program should display something similar to:

```
 "The Matrix" "Machines enslave humans with virtual reality" 1999 136 R "Keanu Reeves, Laurence Fishburne" "Wachowski Brothers"
2 "Pulp Fiction" "Two thugs, boxer and crime boss meet their fates" 1994 154 R "John Travolta, Samuel L. Jackson" "Quentin Tarantino"
3 "Straight outta Compton" "Hip-hop will never be the same" 2015 147 R "O'Shea Jackson Jr., Corey Hawkins, Jason Mitchell" "F. Gary Gray"
4 "Blade Runner" "Futuristic detective hunts obsolete androids" 1982 122 R "Harrison Ford, Sean Young" "Ridley Scott"
5 "Alien" "A slimy creature boards a woman's space tanker" 1979 117 R "Sigourney Weaver, Tom Skerritt" "Ridley Scott"
6 "The Shawshank Redemption" "Innocent man sent to prison for life" 1994 142 R "Tim Robbins, Morgan Freeman" "Frank Darabont"
7 "The Wizard of Oz" "A tornado whisks Dorthy to a magical land" 1939 101 G "Judy Garland, Ray Bolger" "Victor Fleming"
8 "2001: A Space Odyssey" "Supercomputer HAL takes astronauts on ultimate trip" 1968 139 G "Keir Dullea, Gary lockwood" "Stanley Kubrick"
9 "Pitch Perfect" "A capella bellas" 2012 112 PG-13 "Anna Kendrick, Brittany Snow, Rebel Wilson" "Jason Moore"
10 "A Beautiful Mind" "True story of a math prodigy overcoming schizophrenia" 2011 136 PG-13 "Russell Crowe, Jennifer Connelly" "Ron Howard"
11 "The Martian" "A modern take on Robinson Crusoe (on Mars)" 2015 141 PG-13 "Matt Damon, Jessica Chastain, Kristen Wiig" "Ridley Scott"
```

Option 2: Display shortest movie

This option displays the shortest movie in the database. You will need to search your list of movies to find the shortest one. Note that your program will be tested on other databases.

Option 3: Display longest movie

Like Option 2, but displays the longest movie.

Option 4: Display older movies

When the user selects this option, the program queries the user for a year:

```
    Enter option number: 4
    Display movies older than what year? 1998
    ...
```

The program then displays movies that are older than the given year.

Option 5: Display newer movies

Like Option 4:

```
    Enter option number: 5
    Display movies newer than what year? 1998
    ...
```

but movies newer than the given year are displayed.

Option 6: Display movies by rating

When the user selects this option, the program queries the user for a rating:

```
Enter option number: 6
Display movies with what rating? R
...
```

The program then displays movies with the given rating.

Option 7: Add a movie

This is a CHALLENGE option. If you do not implement this option, then your system should display an appropriate message:

```
Enter option number: 7
Option 7 has not been implemented
```

If you implement this option, your program should prompt the user as follows:

```
Enter option number: 7
Adding movie...
    Title: New Movie
    Description: The best movie!
    Year: 2012
    Length in minutes: 145
    Rating: PG
    Main actors: Michael
    Director: SpongeBob
Movie added.
```

In the example, the user has entered the text `"New Movie"`, `"The best movie!"`, and so on. Selecting Option 1 again should display the new movie along with the others.

Option 8: Delete a movie

This is a CHALLENGE option. If you do not implement this option, then your system should display an appropriate message:

```
Enter option number: 8
Option 8 has not been implemented
```

If you implement this option, your program should prompt the user as follows:

```
Enter option number: 8
Delete movie with which ID? 10
Movie #10 has been deleted.
```

The deleted movie should not be seen when Option 1 is reselected.

Option 9: Save the database

This is a CHALLENGE option. If you do not implement this option, then your system should display an appropriate message:

```
Enter option number: 9
Option 9 has not been implemented
```

If you implement this option, your program should prompt the user as follows:

```
Enter option number: 9
Save database to what file? saved.db
Movie database has been saved.
```

Program organization

Create a *project1* directory off of your *cs100* directory and move into *project1*.

Name your main python file *movie.py*. Provide one more file named *database.py* The file *movie.py* should import the *database.py* file, define a *main* function, and then call it:

```
import sys
from database import *
...
def main():

    # read in the database
    db = readDatabase(sys.argv[1])

    option = -1
    while(option != 0):
        print("Welcome to the Python Movie Database!")
        #print user options...
        ...
        ...

        #read option
        option = eval(input("Enter option number: "))

        #process option
        #    process option by calling appropriate database function
        #        passing in the database
        #    do nothing for option 0
        ...

    print("Good bye!")

main()
```

Your *database.py* file should implement the following functions:

- `readDatabase(fileName)`, returns a database
- `readRecord(s)`, returns a record
- `getShortestMovie(db)`, returns a record
- `getLongestMovie(db)`, returns a record
- `getOlderMovies(db,year)`, returns a list of records
- `getNewerMovies(db,year)`, returns a list of records
- `getMoviesWithRating(db,rating)`, returns a list of records

In addition, you may choose to implement the following CHALLENGE functions:

- `addMovie(db)`, returns the modified database

- `deleteMovie(db)`, returns the modified database
- `saveDatabase(db)`, returns `None`

Of course, you may implement additional helper functions.

NOTE: In general, only your main function in *movie.py* (or helper functions in *main.py*) should print the results; the functions in *database.py* should only compute the values and return them. Of course, it is sometimes useful to temporarily add print statements to a function to figure out what is going on. That is OK as long as they are removed in the final version of your program.

Creating a menu driven interface

Here is a simple menu-driven interface after which you can model your version:

```
def main():
    option = -1
    while (option != 0):
        printMenu()
        option = getMenuChoice()
        processOption(option)
    print("Good-bye!")

def printMenu():
    print("for good, enter 1")
    print("for better, enter 2")
    print("for best, enter 3")
    print("to quit, enter 0)

def getMenuChoice():
    return eval(input("Enter option number: "))

def processOption(value):
    if (value == 1):
        print("yay")
    elif (value == 2):
        print("Hey!")
    elif (value == 3):
        print("HURRAH!!")
    elif (option != 0)
        print("option",value,"not understood")

main()
```

Reading a database

For this project, a record is a list of field values and a table is a list of records, and a database is a list of tables. In this case, there is but one table, so a table and a database are equivalent.

The format of each record in the table is as follows: Each movie record is composed of eight fields:

```
ID title description year length rating actors director
```

Each token in the movie record is delimited by whitespace. Data surrounded by quotations should be read as single string tokens. Thus, there are eight field values in the following record:

```
1 "The Matrix" "Machines enslave humans with virtual reality"
```

```
1999 136 R
"Keanu Reeves, Laurence Fishburne"
"Wachowski Brothers"
```

three field values on the first line, three one the second, and one each on the final two lines.

If you are creating movie records (one of the CHALLENGES), you must create the record in exactly this format. The ID of each movie must be unique since the delete movie option uses this field to determine which record to remove.

For details on reading a table/database of this sort, please read the texbook chapter More on Input.

Searching the database

Here is the correspondance between the task you need to implement and the pattern which implements that task:

task	*pattern*
get the longest movie	the extreme pattern
get the shortest movie	the extreme pattern
get older movies	the filtered search pattern
get newer movies	the filtered search pattern
get movies with rating	the filtered search pattern

You can read more about these patterns in the textbook chapter Looping.

Compliance Instructions

To make sure that you have implemented your program correctly, retrieve this file:

```
wget troll.cs.ua.edu/cs100/python/projects/movies.txt
```

You should then be able to run the following command:

```
python3 movie.py movies.txt
```

If your code fails with a runtime error while running this test, then you will receive a zero for this assignment.

Note that your answers do not have to be correct for your program to be graded, only that it not fail. Of course, correct answers will yield a much higher grade.

Challenges

Implement the *add*, *delete*, and *save* options. Try to get your program to keep the records in order by ID as you add and delete them. It's easy enough to append to the end of a file, but inserting records in the middle of a file is more interesting.

Implementation strategies

Start out by writing a program that can read and print out the database.

Once you can successfully read and print the database, implement a menu system that handles the 0 and 1 options.

With that working, implement each remaining option in turn, testing thoroughly before proceeding on to the next option.

Submission Instructions

Change to the *project1* directory containing your assignment. Do an *ls* command. You should see something like this:

```
movie.py   scanner.py   database.py   table0
```

Extra files are OK. Submit your program like this:

```
submit cs100 xxxxx project1
```

Remember to replace `xxxxx` with your instructor name.