CSE 231 **Fall 2022**

Computer Project #07

This assignment focuses on the design, implementation, and testing of a Python program that uses lists and tuples.

It is worth 55 points (5.5% of course grade) and must be completed no later than **11:59 PM on Tuesday, November 1, 2022).** After the due date, 1 point will be deducted for every 1 hour late. No more submissions will be accepted after Wednesday November 2, 2022.
If you submit by Monday, October 31, 2022, you will receive a 2-point bonus points. Note that you need to click on the submit button to be considered as submitted.

Assignment Overview

In this assignment, you will practice with lists of lists and tuples and write a program to answer the questions described below. **Do not use Dictionaries in this project.**

Assignment Background

The GroupLens Research Project at the University of Minnesota has gathered a dataset from the MovieLens website recording the ratings provided by different users on different movies on a scale of 1-5.

You will write a Python program that uses the MovieLens dataset to answer questions like the highest average rated movies in a year, specific genre, by a specific gender (male or female), or a specific profession (occupational group).

The data is provided in three files:
1) `reviews.txt`, This is a tab-separated list of

```
userID  movieID  rating  timestamp.
```

2) `users.txt`, Demographic information about the users; this is a bar-separated list of

```
userID | age | gender | occupation | zip code.
```

The `userIDs` are the ones used in the `reviews.txt` file.
3) `movies.txt`, Information about the items (movies); this is a bar-separated list of

```
movieID | movie title (release year) | release date | IMDb URL |
        unknown | Action | Adventure | Animation |
        Children's | Comedy | Crime | Documentary | Drama | Fantasy |
        Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi |
        Thriller | War | Western |
```

The last 19 fields are the genres, a 1 indicates the movie is of that genre, and a 0 indicates it is not; movies can be in several genres at once. The movieIDs are the ones used in the reviews.txt data file.

From these files, you will create these three corresponding list data structures:

- `L_users`, a **list of tuples** indexed by `userID`, where each user is a tuple`(age, gender, occupation)`

- `L_reviews`, a **list of lists** indexed by `userID`, where each user is a *list of tuples* and each tuple is`(movieID, rating)`
- `L_movies`, a **list of tuples** indexed by `movieID`, where each movie is a tuple `(movieName, releaseDate, list of genres)`

The index is the ID of whatever data you are looking at. If the `userID` of a movie in the movies file is 1, it should be at index ONE of `L_movies`, NOT index ZERO. The index IS the ID.

Assignment Specifications
You will develop a Python program that has the following functions

`open_file (s) → file pointer:`

a. This function prompts the user to input a file name to open and keeps prompting until a correct name is entered. The parameter, `s`, is a string to incorporate into your prompt so you are prompting the user for a particular type of file at one time `("user","review","movie").`
You need to include `encoding ="windows-1252"` as
`fp = open(filename,"r",encoding ="windows-1252")`
because the files were created on a Windows system with some special characters that cannot be read using the default encoding (which is usually `utf-8`). You can open all the provided files using the windows encoding.
b. Parameters: `string`
c. Returns: `a file pointer`
d. Display: prompts and error messages

`read_users (fp) → list of tuples:`

This function reads the `user.txt` file using file pointer fp and returns a list of tuples. The file has no header. The data is separated by vertical bars ('|'). Each line has five pieces of data; we use the first four, ignoring the last. The `reviewer_id` and `age` are ints; the `gender` and `occupation` are strings:
`reviewer_id | age | gender | occupation | X`
Important: the file is organized in order of `reviewer_id` with `reviewer_id == 1` first, so as you read the file sequentially and append to your master list they will be stored with the correct index ***IF*** you initialize your `master_list` to have an empty list to account for the non-existent `reviewer_id` of 0. (Think about the reviewer_id as the index for your data in the `master_list`). Each entry will be a tuple:
`(age, gender, occupation)`
with types
`(int, str, str)`
(Note that you really don't use `reviewer_id` directly, only implicitly.)
For example, Let's say the file has these 2 lines:

```
1|24|M|technician|85711
2|53|F|other|94043
```

Then your `master_list` will be:
`Master_list = [[],(24, ‘M’,  ‘technician’), (53, ‘F’, ‘other’)]`

a. Parameter: `file pointer`
b. Returns: `list of tuples`
c. Displays: nothing

`read_reviews (N,fp) → list of lists of tuples of ints:`

a. This function reads the `reviews.txt` file using the file pointer `fp`. The file has no header. `N`, an `int`, is the number of users. Knowing `N` allows us to initialize a list of `N+1` empty lists which makes programming of this function easier. Subsequent lines have four pieces of data separated by whitespace (tabs actually), each is an `int` – the last one we ignore:
```
userID    movieID   rating   X
```
You are to create a list of lists where each index is the `userID`; for this discussion let’s call that list_of_lists `L_reviews`. There is no `userID 0` so the first list, `L_reviews[0],` will be empty and that means that the list-of-lists `L_reviews` will be `N+1` items (lists) long. That is, the data for the reviewer with `reviewer_id` #43 will be at `L_reviews[43].` Each reviewer’s list will be a sorted list of tuples of their reviews: `(movieID, rating).` Each reviewer’s list of tuples will be sorted in ascending order by the `movieID`, i.e. the first item in the tuple (hint: that means that we can use the default behavior of sorting in Python).
Hint: Create a blank, empty list of the size you need first. Then grab the three pieces of information you need from the line you are looking at, then use `userID` to index your list and add the other information.
b. parameter: `a file pointer`
c. Returns: `list of sorted lists of tuples of ints`
d. Display: nothing

`read_movies (fp) → list of tuples:`

a. This function reads the `movies.txt` file using the parameter `fp`. The file has no header and items are separated by bars (‘|’). Each line has four pieces of data (5 if you count the empty space between date and url) followed by 19 genre indicators:
```
movieID | title | date || url | 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0
```
where the 19 genre options are indicated by a zero or one. The genre value indexes into the `GENRES` list constant provided. For example, a one in the last slot indicates that the movie is a ‘Western’, the last genre in the `GENRES` list. Note that the title field includes a year in parentheses; we consider that part of the title string.
The tuple is of the format:
`(title, date, [list of genres])`
For example, given this line from the file:
```
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-
exact?Toy%20Story%20(1995)|0|0|0|1|1|1|0|0|0|0|0|0|0|0|0|0|0
|0|0
```

The tuple will be
`('Toy Story (1995)', '01-Jan-1995', ['Animation', "Children's", 'Comedy'])`
Furthermore, since this movie has `movieID == 1` it will be at index 1 in the list that is returned. Hint: we handle `movieID` similar to how we handled `userID` and `reviewer_id.` Add a blank list at the 0 index of your list, like you do for `read_users().`

b. Parameter: `a file pointer`
c. Returns: `list of lists`
d. Display: nothing

`year_movies (year, L_movies) → list:`

a. This function filters the main movie list (returned by the `read_movies()` function) to find movies for a specific year and returns their ids `movieID` (ints) as a sorted list in ascending order. Few things to remember:
   i. the year in `L_movies` is in the date which is a string of the form "day-month-year". You will need to first extract the movie year, then compare it to the variable `year`.
   ii. The `movieID` is the index of the movie in `L_movies`.

b. Paremeters: `int, list of tuples`
c. Returns: `sorted list of ints`
d. Display: nothing

`genre_movies (genre, L_movies) → list:`

a. This function filters the main movie list to find movies for a specific genre and returns their ids as a list. Remember a movie can have multiple genres so if any of the genres is the specified genre the movie would be selected. This function is similar to `year_movies`() function.
b. Parameters: `string, list of tuples`
c. Return : `sorted list of ints`
d. Display: nothing

`gen_users (gender, L_users, L_reviews) → list of lists of tuples:`

a. This function filters the main reviews list to find reviews for a specific gender of users and returns them as a list of lists. That is, find the users in `L_users` of the specified gender then use the user's number (index) to index into the `L_reviews` to get their reviews. Return a list of those reviews.
b. Parameters: `str, list of tuples, list of list of tuples`
c. Return: `list of list of tuples`
d. Display: nothing

`occ_users (occupation, L_users, L_reviews)` → `list of lists of tuples`:

a. This function filters the main reviews list to find records for a specific occupational group of users and returns them as a list of lists of tuples. This function is the same as `gen_user` but filters by occupation instead of gender.
b. Parameters: `str, list of tuples, list of list of tuples`
c. Return: `list of list of tuples`
d. Display: nothing

`highest_rated_by_movie (L_in, L_reviews, N_movies)` → `list, float`

a. This function calculates the average rating for the reviews in `L_reviews` list of the movies in `L_in` list and returns a list of the highest average rated movies and the highest average. Round each average to 2 places:
   1. The `L_in` is in the form of what was returned by `year_movies` or `genre_movies`, i.e. a list of `movieID`s.
   2. `L_reviews` is a list of lists of tuples in the form `(movieID, rating)`. The form of what was returned by `read_reviews` function
   3. `N_movies` is the total number of movies in the file.

   For example:
```
N_movies = 10
L_reviews = [[],[(2,3),(3,3),(6,1),(9,5),(10,2)],
     [(3,1),(4,4),(5,3),(6,5),(7,3),(8,5)],
     [(1,2),(4,3),(7,3)], [(10,5)], [],
     [(2,2),(5,2),(8,3)], [], [],
     [(2,2),(10,4)], []]
L_in = [1,2,3,6,7]
```

   will return
```
[6, 7], 3.0
```

   Algorithm:
   For each movie in the list `L_in`, to find its average you need to sum the ratings and count the ratings for that movie so in this case you need to collect those two values for each movie (I have all movies in my list so I can keep track of which movies have which data) (Hint: initialize this data structure for all movies):
```
[[0, 0], [2, 1], [7, 3], [4, 2], [0, 0], [0, 0], [6, 2], [6, 2], [0, 0], [0, 0], [0, 0]]
```
   Then calculate the average for each movie (remember to round so you get 2.33):
```
[0, 2.0, 2.33, 2.0, 0, 0, 3.0, 3.0, 0, 0, 0]
```
   Then find the maximum average (Hint: `max()`)
```
3.0
```
   Finally, find which movies have that average
```
[6, 7]
```
   List comprehension is useful, but not necessary.

b. Parameters: `list of ints, list of lists of tuples, int`
c. Return: `list of floats, float`
d. Display: nothing

highest_rated_by_reviewer (L_in, N) → list:

a. This function calculates the average rating for movies by a specific group of users (`L_in`) and returns a list of the highest average rated movies and the highest average. Round each average to 2 places:
   1. `L_in` is a list of lists of tuples in the form `(movieID, rating)`. The form of what was returned by `gen_users` or `occ_users.`
   2. `N_movies` is the number of all movies.

   For example:
   ```
   N = 10
   L_in = [[(1,1),(0,0),(5,1),(7,1)],
   [(1,2),(3,4),(5,2),(2,0),(7,2),(9,2)],
   [(2,3),(4,3),(6,3),(8,3),(7,3),(3,3)],
   [(2,4),(4,4),(6,4),(8,4)]]
   ```
   Then proceed as with `highest_rated_by_movie`, but the difference is that `L_in` contains all the reviews you consider (as opposed to reviews for a specified set of movies). That is, collect total ratings and counts, find averages, find the max, and finally find those movies with that max average. Note that as with `highest_rated_by_movie` it is easiest to create a data structure for **all** movies to hold the totals and counts.

a. Parameters: list of list of tuples, int
b. Return: list of floats, float
c. Display: nothing

main():

This function would:

1- read the files (call the open_file function 3 times to open 3 different files. Remember that the open_file() function accepts a string as a parameter which is the type of the movie to open:
   - If we want to open a user file the argument should be `'users'`
   - If we want to open a review file the argument should be `'reviews'`
   - If we want to open a movies file the argument should be `'movies'`

2- create the main 3 lists of lists by calling (`read_users`, `read_reviews`, and `read_movies`).

3- Display the menu (MENU) and prompt to choose one of these five different options to select from:
```
1. Find the highest average rating item for a specific year
2. Find the highest average rating item for a specific genre
3. Find the highest average rating item by a specific gender
4. Find the highest average rating item by a specific occupational group
5. Quit
```

4- Depending on the option selected it would prompt the user for an appropriate option and display the result based on the selection. The function would keep prompting the user for option until the user quits:

a. For option 1, prompt for a year and re-prompt if the year is less than 1930 or greater than 1998 while checking that it is an integer. Call `year_movies` and then call `highest_rated_by_movie`. Then display the maximum average followed by the movies names (not `movieID`) that have that average (print one movie in a separate line). Remember that the name of the movie is in the list of tuples returned by `read_movies` function.

b. Option 2 is similar. Display all valid genres. Prompt for a genre and re-prompt if the genre is not a valid genre (check Error checking below). Call `genre_movies` and then call `highest_rated_by_movie`. Then display the maximum average followed by the movies names that have that average (print one movie in a separate line).

c. For options 3, prompt for a gender and re-prompt if the gender is not valid (check Error checking below). Call `gen_users` (note that the gender in the file is upper case) and then call `highest_rated_by_reviewer`. Then display the maximum average followed by the movies names that have that average (print one movie in a separate line).

d. Option 4 is similar. Display all valid occupations. Prompt for an occupation and re-prompt if the occupation is not valid. Call `occ_users` and then call `highest_rated_by_reviewer`. Then display the maximum average followed by the movie names that have that average (print one movie in a separate line).

e. **Error checking.** The year must be in the range noted above. The genre must be in the `GENRES` list constant provided, but it may be in any case (upper, lower, mixed). (Hint: genre are "capitalized" and there is a string method for that.) The occupation must be in the `OCCUPATIONS` list constant provided (note that your task is eased because the list is all lower case). The gender must be `'M'` or `'F'` in upper or lower case. If the input doesn't meet specifications, re-prompt for that input.

Assignment Notes and Hints

The data was collected from 1997-98 so there would be no movies after 1998. Be mindful when testing for a specific year. The movies in the data are as old as 1930s but nothing before that. If you code smartly there is a way to skip some user ratings when working on specific items.

1. The coding standard for CSE 231 is posted on the course website:

http://www.cse.msu.edu/~cse231/General/coding.standard.html

Items 1-9 of the Coding Standard will be enforced for this project.

2. **The `enumerate()` function is useful to iterate through these data structures to get both the tuple and its index because the index is the `userID` or `movieID`, depending on the data structure. However, `enumerate()` starts indexing at zero. You can index through a slice starting at 1 and also have the enumerate index starting at one using a second argument to enumerate:**

```
for i, tup in enumerate(lst[1:], 1):
```

3. The program will produce reasonable and readable output, with appropriate labels for all values displayed.

4. We provide a `proj07.py` program for you to start with.

5. Do not use Dictionaries for this project.

6. If you “hard code” answers, you will receive a grade of zero for the whole project. An example of hard coding is to simply print the approximate value of e rather than calculating it and then printing the calculated average.

Assignment Deliverable

The deliverable for this assignment is the following file:

`proj07.py` – the source code for your Python program

Be sure to use the specified file name and to submit it for grading before the project deadline.

Function Tests:
Look in the folder Function Tests if you want to test your function locally. Make sure that your code is in the same folder as the function tests in your computer.

I/O Test:
Look in the folder “I_O_Tests” for Test 1-6

Grading Rubric

Computer Project #07 Scoring Summary

General Requirements:

(3 pts) Coding Standard 1-9
(descriptive comments, function headers, mnemonic identifiers, format, etc...)

Implementation:

(3 pts) open_file function (No Coding Rooms tests)

(4 pts) read_users function

(5 pts) read_reviews function

(5 pts) read_movies function

(4 pts) year_movies function

(3 pts) genre_movies function

(2 pts) gen_users function

(2 pts) occ_users function

(5 pts) highest_rated_by_movie function

(5 pts) highest_rated_by_reviewer function

(3 pts) Test 1

(2 pts) Test 2

(2 pts) Test 3

(2 pts) Test 4

(2 pts) Test 5

(3 pts) Test 6

Note: hard coding an answer earns zero points for the whole project
-10 points for not using main()