

Week 2 - Assignment 1B

CHAPTER

IV

Warming up exercises

Exercise 6. The script `hello_from_place.py` (Content -> Module 1) can receive arguments when run from the terminal. Modify the code of that script to log every line printed by this script in a file called `visited_places.txt`. This file should be in a folder called `log`, in the first directory above the directory of the script! Use the `os` module to navigate through the different paths.
(Try to use a context manager to open the files.)

Remark. In the practice session the dunder variable `__file__` when accessed inside Spyder was in lower case, and because of that we could not access the folder of the script. That's an issue of Spyder. When running a program from the terminal, that variable accurately represents the path to the script.
Because of that, test your solution to the exercise above running the script directly from the terminal

Exercise 7. Create a function `filter_names()` that receive a list of strings `names` and a str `letter`, with length 1. The function should return a new list containing all the names of the list `name` that start with the character in `letter`. Use list comprehension.

In this chapter:

[Warming up exercises.](#)

[Mini-project 2.](#)

[Submission instructions.](#)

Mini-Project 2

Pair Programming

Task overview

This week you are going to work with a small movie dataset. Your task is to write a few predefined functions that read and manipulate the data.

About the data

Together with the other assignment files on Brightspace you can find a text file named **movie_dataset.txt** that contains information about 113 different musical movies. Every movie is on a new row and contains three attributes - **movie title, rating, duration**. The movie title is always in quotation marks, the rating is a float and the duration is an integer with the amount of minutes the movie has. For example, here are the first few movies in the database:

```
"Whiplash" 8.5 106
"Bohemian Rhapsody" 7.9 134
"CODA" 8.0 111
"La La Land" 8.0 128
"Black Snake Moan" 6.9 116
"Footloose" 6.6 107
...
```

movie_dataset.txt: some of the movie names in that file might have characters that can not be read in your computer. If you notice something like that, you can just remove the problematic line from the txt file, since the goal of the project is to implement the functions that handle the dataset.

Functions to implement

- **main()**. The function `main()` is already partially implemented. It first calls the function `read_data()`. Then asks for the user input to choose 1 out of 4 options (which call for one of the functions above with the appropriate arguments).

- 1: print all the movie titles in the dataset. Usage: (input the following string) '1'.
- 2: show the average duration of all movies with rating of your choice. Usage: enter '2' and rating separated by space, e.g. '2 8.8'.
- 3: find the rating of a specific movie. Usage: enter '3' and movie name separated by space, e.g. '3 Coco'. Note that the movie name can have more than 1 word.
- 4: find the best rated movie within a given duration range. Usage: enter '4', the minimal and maximal wanted duration and a flag which can be either 'True' or 'False', e.g. '4 60 120 True'. If you enter 'True' the shortest best movie will be presented, otherwise the longest best movie",

Before start working on your code, read the description of all the **functions** that need to be implemented. The explanation of the function `main` might be particularly relevant to understand the code already provided.

Modify the function `main` so that the user can **also** pass input directly from the terminal.

Example.

Running (in the terminal) the command:

```
python assignment1B.py 4 60 120 True
```

would have the same result as first running the script and giving the following input:

```
4 60 120 True
.
```

- **read_database(filename = 'movie_dataset.txt')** is a function that reads from the given text file 'movie_dataset.txt', located at the directory of the current script. The format of the entries in this text file is the following:

"title of the movie" rating duration

where each movie is on a new line.

The function `read_database()` should read the content of the file and return a dictionary with the information of each movie using the movie titles as keys (string), and a tuple of the respective rating (float) and duration (integer) as values.

For example, the first movie in the file could be added in a dictionary called `movies` as:
`movies['Whiplash'] = (8.5, 106)`

- Note that, in the dictionary, the movie title should **not** have quotations.
 Ex.: the movie *Whiplash* in the dictionary *movies* should be accessed using `movies['Whiplash']`, and not `movies['"Whiplash"]`.
- If two movie entries have the same title, the movie with the longer duration should be kept. Note that some movie titles might be several words long.
- If the filename is invalid the program should return `None` instead of a dictionary.
- **movie_title_list(movies : dict)** is a function that takes a dictionary as described above and returns a list of strings containing all the movie titles.
- **avg_duration(movies: dict, rating: float)** is a function with parameter `movies`, a dictionary with movies and their respective rating and duration, and parameter `rating`. This function computes and return the average duration of all movies with ratings **above** the given parameter rating.
 - If no movies have the above rating the function returns `None`.
 - If a movie has a duration of 0 minutes, that movie should not be included when computing the average duration.
- **bestRatedMovie(movies: dict, min_dur: int, max_dur: int, choose_shortest: bool)** is a function that returns the movie title with the highest rating with a duration between `min_dur` and `max_dur` (included). The optional parameter `choose_shorter` is a boolean that indicates what to do in case of a tie.
 - If `choose_shortest` is `True`, the shortest movie is returned in case of a tie in the ratings. Otherwise, the longest one is returned.
 - In case no movies in the dictionary fulfill the requirements the function should return `None`.

Tip: The function above are mostly independent from each other, but implementing them in the order they were given might be easier.

Important to keep in mind:

- Do not change the function definitions (names or parameter order).
- Comment your code and write the documentation of the functions (ex: the first function in the template).
- If a parameter is followed by an equal sign (such as the case of `data_list(filename = "movie_dataset.txt")`) that means that the value "movie_dataset.txt" is the default

value if no parameters were passed to the function.

- Your functions will be (mostly) automatically tested (similar to project Lovelace exercises). You can run yourself some of those tests before submitting your solutions. If you want, download the 'assignment1B_selftesting.py', import your solutions as a module and run the script to see if your functions pass the tests.
(the tests provided do not cover everything that is asked. Make sure your functions behave correctly in all the specified cases (ex: if no movies with specified rating is found when calling `bestRatedMovie()` the function returns `None`), so make sure to deal with those cases inside your functions.)

Submission instructions

Extra documents:

If you have worked together with another student, create a .txt file called `programming_partner.txt` containing the S-number of both of you.

Example. `programming_partner.txt`

```
1 Mini project 1:  
2 s012345  
3 s123456  
4
```

Assignment 1B is part of Assignment 1, which has its deadline after Week 2. Submit your final solutions in Brightspace.

Both students should submit their work individually on Brightspace.

General submission:

Create a zip file called `Assignment1` containing the files:

- `mini_proj_1.py`
- `mini_proj_2.py`
- `programming_partner.txt`

Submit your file to Assignment 1, in Brightspace.