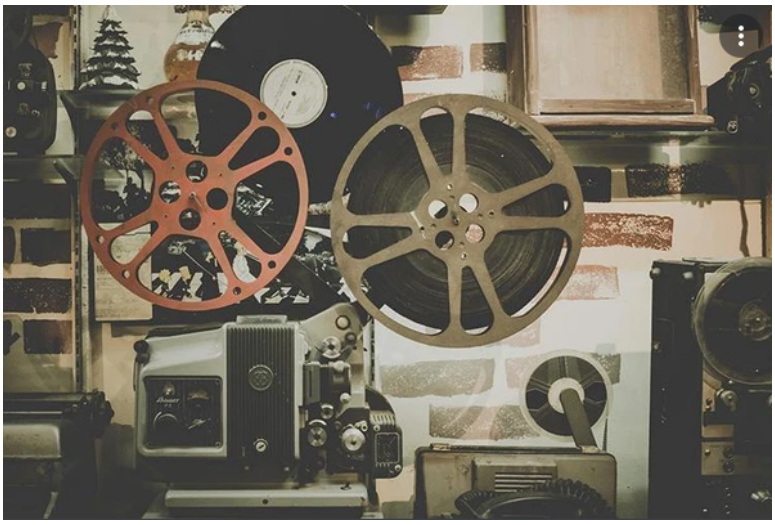
Computer Science AP/X CSCI-140/242
Movies Project 1

9/12/2021



Due Date: Wednesday, September 29, 2021 @ 11:59pm
Version Info:

- **9/12/2021:** Initial version.

1 Overview

In this project you will be working with datasets from the IMDB website. You will get experience reading in the data and using various structures and collections in Python to store, organize and efficiently query for different kinds of information relating to the movies and their ratings.

1.1 Individual Assignment

This project must be done individually. **Do not share your code with any other student in the class, and likewise do not accept code from any other student. Either case will result in automatic failure of this assignment and possibly further action**.

1.2 Getting Help

The GCCIS Tutoring Center, as well as your instructor's office hours, are both great resources to get individual help.

In addition you are encouraged to post any questions you have about this assignment to the MyCourses Project 1 Discussion Thread. These threads are frequently monitored by the instructors. The pinned post at the top will be used to address the most frequently asked questions. It is a good idea to monitor these discussions while you are working on this assignment.

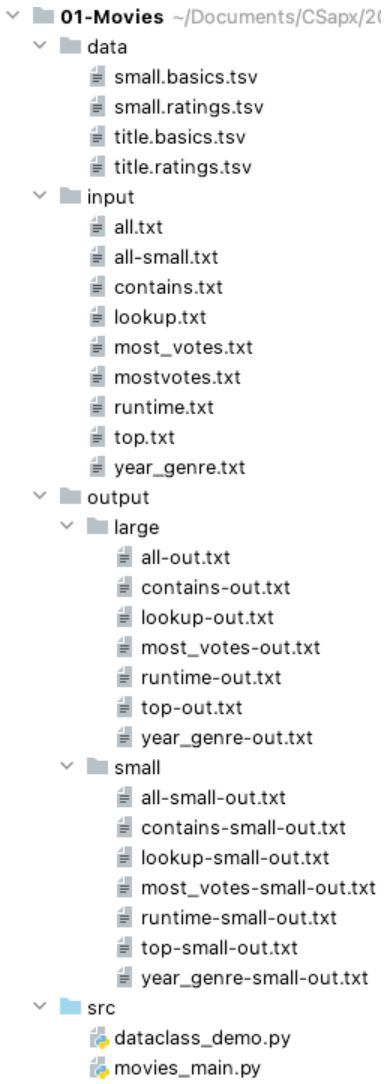
2 Getting Started

2.1 Git, GitHub and PyCharm

Most of the starter files for this project are being provided to you via GitHub. You will be using this resource for the remainder of the class to get starter code and version control the code you write.

To begin with you should work through the GitHub Setup instructions. When you get to the **Working on Your Assignment** section you should use the following GitHub classroom link for your section to create and clone your repository.

Once you have your project in PyCharm your project structure should look like the following (minus the large IMDB dataset files).



Because the IMDB dataset files are so large, you will need to download the dataset zip file separately from the course website. Once unzipped you can copy/move them into the `data` directory of your project.

2.2 Starter Files

There are four directories in the starter files.

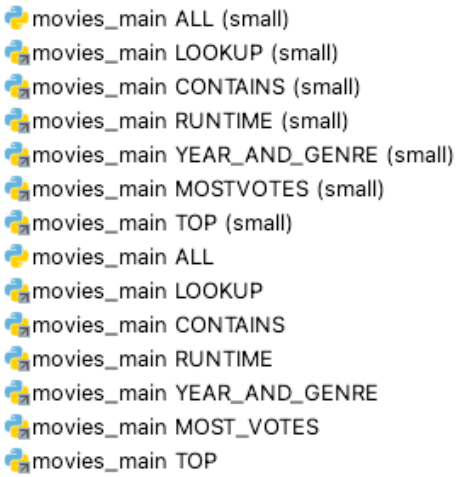
- **data**: The tab separated value (TSV) datasets for the movies and ratings. **You should not modify these files!**
  - `small.basics.tsv`: A small movie dataset
  - `small.ratings.tsv`: A small rating dataset
  - `title.basics.tsv`: The large IMDB movie dataset
  - `title.ratings.tsv`: The large IMDB rating dataset
- **input**: The files used as input for generating the dataset queries. **It is suggested you do not modify the provided inputs, but you are free and encouraged to create new ones.**
- **output**: The outputs of the main program when running with the inputs. The outputs are separated into a directory for the IMDB datasets (large) and the small datasets (small). **You should not modify these files!**
- **src**: The folder containing all Python source files you will implement, including the main program and a dataclass sorting demo program.

2.3 Project Setup Video

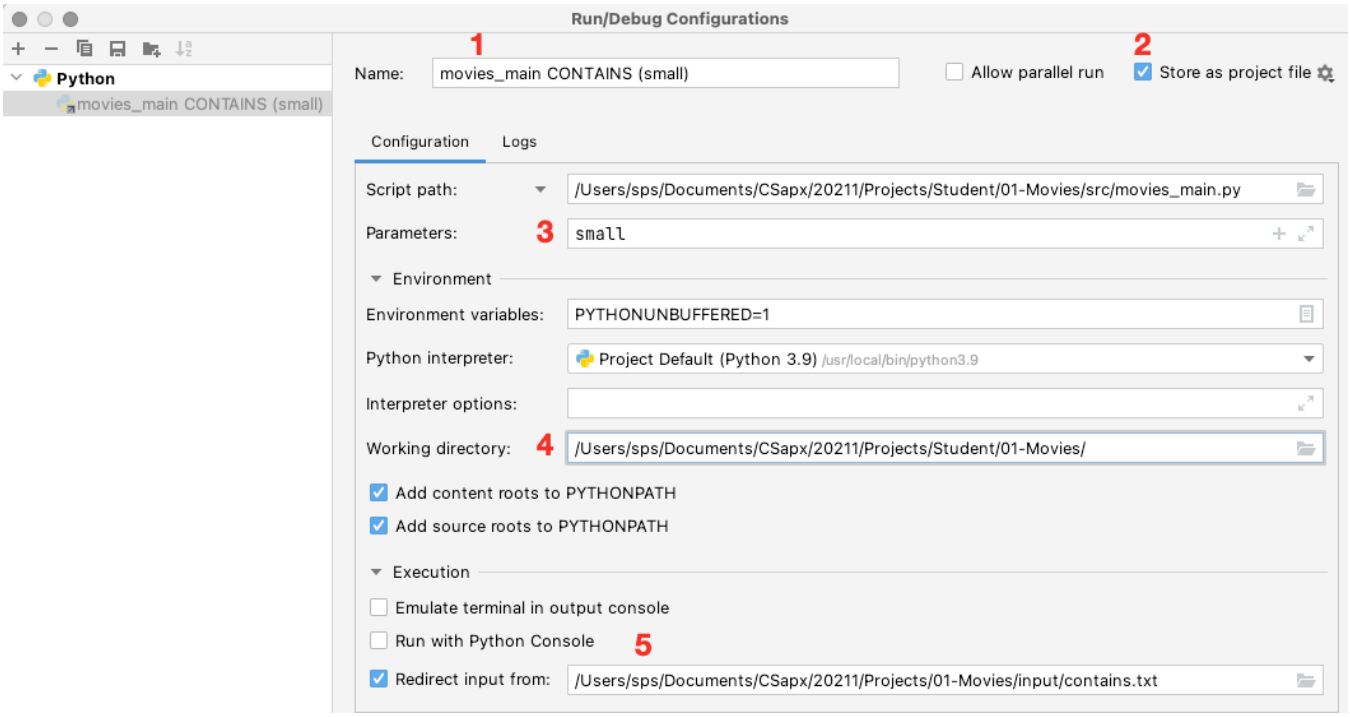
Here is a video that walks through getting set up for the project and working with Git.

2.4 Run Configurations

We are providing you with a lot of run configurations to make your development easier. There are run configurations with queries that work with the small and large datasets.

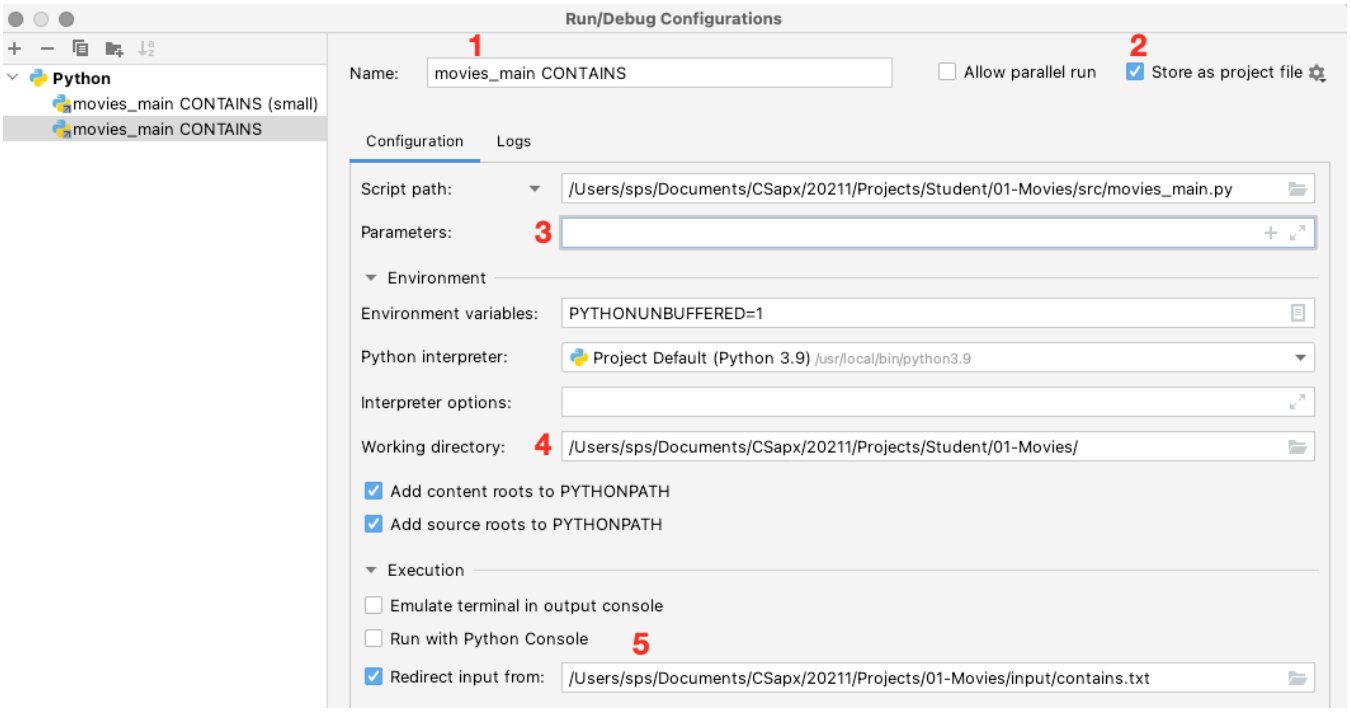


1. Take a look at the run configuration for the contains command for the small dataset by editing the configurations.
   1 The config is named with the query.
   2 The project file should already be saved for you.
   3 If a command line argument is present, it will use the small dataset file.



4 The working directory should be set to the root of your project.
5 Input is being redirected from the contains input file.

2. You'll notice the run configuration for the large dataset is very similar, except there are no command line arguments.



3 Requirements

3.1 Main Program

The main program is `movie_main.py` in the `src` folder uses the command line to determine whether to use the small or large datasets. If no command line arguments are present, the large datasets should be used. Otherwise the command line is assumed to contain a single argument `small`, and the small datasets should be used.

After the datasets are loaded, the queries detailed below can be entered into standard input. To speed this up you should use the supplied run configurations which automatically redirect the supplied input file queries to standard input.

The queries are guaranteed to always be correctly formatted. Read each query line and perform the corresponding query, until there are no queries left.

3.2 Datasets

There are two datasets that your program needs to work with, separated by small and large versions.

3.2.1 Movies Datasets

The movies datasets are **data/small.basics.tsv** and **data/title.basics.tsv** for the small and large TSV datasets. The first line of the file is the header which indicates the fields for each movie.

```
tconst titleType primaryTitle originalTitle isAdult startYear endYear runtimeMinutes genres
```

Each movie follows after the header, with one movie per line.

```
tt0033467 movie Citizen Kane Citizen Kane 0 1941 \N 119 Drama,Mystery
tt0052357 movie Vertigo Vertigo 0 1958 \N 128 Mystery,Romance,Thriller
tt0081505 movie The Shining The Shining 0 1980 \N 146 Drama,Horror
```

Since the delimiter between the fields is a tab character, you should use **"\t"** as the string to split a movie line on after reading it.

If a movie does not have a particular value for a field, the string **"\\N"** is used (the extra backslash is required to escape the expected backslash). In the previous example none of the movies have an end year. If a field has this value and it is to be displayed, the `None` or `0` should be used in its place.

3.2.2 Ratings Datasets

The ratings datasets are **data/small.ratings.tsv** and **data/title.ratings.tsv** for the small and large TSV datasets. The first line of the file is the header which indicates the fields for each rating.

```
tconst averageRating numVotes
```

Each rating follows after the header, with one rating per line.

```
tt0033467 8.3 404097
tt0052357 8.3 365021
tt0081505 8.4 899812
```

Note that the rating refers to the movie with the particular rating by using the **tconst** field.

3.2.3 Adult Movies

There are a lot of adult movies in the datasets that should be ignored when reading the movies in. To detect this, the `isAdult` field for the movie will be the string `"1"` to indicate it should be ignored.

Likewise, when reading the ratings after the movies, any rating that does not have a matching movie in the movie dictionary should be also be ignored.

Make sure when your program is done reading in the movies and ratings that you get the same counts as in the sample outputs (for the large datasets):

```
Total movies: 7965199
Total ratings: 1161579
```

3.3 Queries

There are a total of six unique kind of queries your program needs to support. They are listed here in the suggested order you should implement them.

1. **LOOKUP {tconst}**: Look up a movie and rating by its unique identifier.
2. **CONTAINS {titleType} {words}**: Find all movies of a certain type whose titles contain the sequence of words.
3. **YEAR_AND_GENRE {titleType} {year} {genre}**: Find all movies of a certain type from a particular year that match a genre.
4. **RUNTIME {titleType} {min-minutes} {max-minutes}**: Find all movies of a certain type that are within a range of runtimes.
5. **MOST_VOTES {titleType} {year} {num}**: Find the given number of movies of a certain type and given year with the most votes.
6. **TOP {titleType} {num} {start-year} {end-year}**: Find the number of movies of a certain type by a range of years that are the highest rated and have at least `1000` votes.

In general, once the datasets are loaded, all individual queries should be *very fast* and take no more than a few seconds to complete.

3.4 Program Output

In the **output** directory are example outputs of each query running with the small and large datasets (separated by directory).

3.4.1 File Reading Output

The first thing your program needs to do is read the movie and rating dataset files into an appropriate data structure with fast lookup. Regardless of the query, this section's output is always the same format:

```
    reading {movies-file} into dict...
    elapsed time (s): {###}

    reading {ratings-file} into dict...
    elapsed time (s): {###}

    Total movies: {##}
    Total ratings: {##}
```

Please note the timings you get are specific to your machine and are not required to match the solution output.

3.4.2 LOOKUP Output

When looking up a movie and rating, there are two possible outcomes. If the movie and rating are found the output should be:

```
processing: LOOKUP {tconst}
    MOVIE: Identifier: {tconst}, Title: {primaryTitle}, Type: {titleType}, Year: {startYear},
        Runtime: {runTime}, Genres: {genres}
    RATING: Identifier: {tconst}, Rating: {rating}, Votes: {numVotes}
elapsed time (s): {###}
```

If either the movie or rating is not found the output should be:

```
processing: LOOKUP tt0080684
    Movie not found!
    Rating not found!
elapsed time (s): {###}
```

3.4.3 CONTAINS Output

When looking up movies of a certain title type, whose primary title contains the sequence of words from the input, there are two possible outcomes. In the event one or more movies are found, the movies should be displayed alphabetically by primary title.

```
processing: CONTAINS {titleType} {words}
    Identifier: {tconst}, Title: {primaryTitle}, Type: {titleType}, Year: {startYear},
        Runtime: {runTime}, Genres: {genres}
    ...
elapsed time (s): {###}
```

If there are no movies that match the output should be:

```
processing: CONTAINS {titleType} {words}
    No match found!
elapsed time (s): {###}
```

3.4.4 YEAR_AND_GENRE Output

When looking up movies of a certain title type, whose start year and genre matches, there are two possible outcomes.

In the event one or more movies are found, the movies should be displayed alphabetically by primary title.

```
processing: YEAR_AND_GENRE {titleType} {startYear} {genre}
    Identifier: {tconst}, Title: {primaryTitle}, Type: {titleType}, Year: {startYear},
        Runtime: {runTime}, Genres: {genres}
    ...
elapsed time (s): {###}
```

If there are no movies that match the output should be:

```
processing: YEAR_AND_GENRE {titleType} {startYear} {genre}
    No match found!
elapsed time (s): {###}
```

3.4.5 RUNTIME Output

When looking up movies of a certain title type, whose runtime is between the start and end times (inclusive), there are two possible outcomes.

In the event one or more movies are found, the movies should be displayed first by descending runtime, and second ascending alphabetically by primary title.

```
processing: RUNTIME {titleType} {startTime} {endTime}
    Identifier: {tconst}, Title: {primaryTitle}, Type: {titleType}, Year: {startYear},
        Runtime: {runTime}, Genres: {genres}
    ...
elapsed time (s): {###}
```

If there are no movies that match the output should be:

```
processing: RUNTIME {titleType} {min-minutes} {max-minutes}
    No match found!
elapsed time (s): {###}
```

3.4.6 MOST_VOTES Output

When looking up the top number of movies of a certain title type, there are two possible outcomes.

In the event one or more movies are found, the movies should be displayed first by descending number of votes, and second ascending alphabetically by primary title.

```
processing: MOST_VOTES {titleType} {num}
    1. VOTES: {###}, MOVIE: Identifier: {tconst}, Title: {primaryTitle}, Type: {titleType},
        Year: {startYear}, Runtime: {runTime}, Genres: {genres}
    ...
elapsed time (s): {###}
```

If there are no movies that match the output should be:

```
processing: MOST_VOTES {titleType} {num}
    No match found!
elapsed time (s): {###}
```

3.4.7 TOP Output

When looking up the top number of movies of a certain title type for each year in a range of years (with at least 1000 votes), there are two possible outcomes.

In the event one or more movies are found, the movies should be displayed in increasing order by year. For each year, the top movies should display first by descending rating, second by descending number of votes, and third ascending alphabetically by primary title.

```
processing: TOP {titleType} {num} {startYear} {endYear}
    YEAR: {####}
    1. RATING: {###}, VOTES: {###}, MOVIE: Identifier: {tconst}, Title: {primaryTitle},
        Type: {titleType}, Year: {startYear}, Runtime: {runTime}, Genres: {genres}
    ...
    YEAR: {####}
    ...
elapsed time (s): {###}
```

If there are no movies that match the output should be:

```
processing: TOP {titleType} {num} {startYear} {endYear}
    No match found!
elapsed time (s): {###}
```

4 Design

4.1 Dataclasses and Dictionaries

You should design appropriate dataclasses to represent the information for a particular movie or rating, and dictionaries to store the information for fast lookup.

In order to provide fast access to the information, Python dictionaries are the ideal collection to use. For example the first query is to lookup a movie or rating by its unique **tconst** identifier. A dictionary whose key is the identifier, and whose value is the movie and/or rating object will allow for constant time access lookup, e.g.

```
# movies is a dict[str, Movie]
movie = movies['tt0033467']  # assuming: 'tt0033467' in movies
```

The incorrect way to lookup would be to loop over all the keys in the dictionary in order to find the object (because the lookup will now be in linear time), e.g.

```
for tconst in movies:
    if tconst == 'tt0033467':
        movie = movies['tt0033467']
```

4.2 Python Modules

You should consider breaking apart your implementation so it is not contained entirely in the main Python module. One suggestion is to separate things as follows:

- A module for each dataclass, e.g. Movie and Rating.
- A module to read in the dataset files.
- A module to handle the processing of the queries.

5 Implementation

5.1 Reading Datasets

The large datasets from IMDB are UTF-8 encoded and will cause problems if trying to `open` using the default encoding. To fix this always specify the encoding, e.g.:

```
with open('data/title.basics.tsv', encoding='utf-8') as f:
    ...
```

5.2 Timings

Every operation should be timed before and after it finishes. This includes reading each of the datasets, as well as executing each individual query. Python provides the built-in module timeit.

Here is a quick example showing how to find the elapsed time in seconds for a particular operation.

```
from timeit import default_timer as timer

start = timer()
# perform operation
end = timer() - start
print('elapsed time (s):', end - start)
```

5.3 Sorting Objects

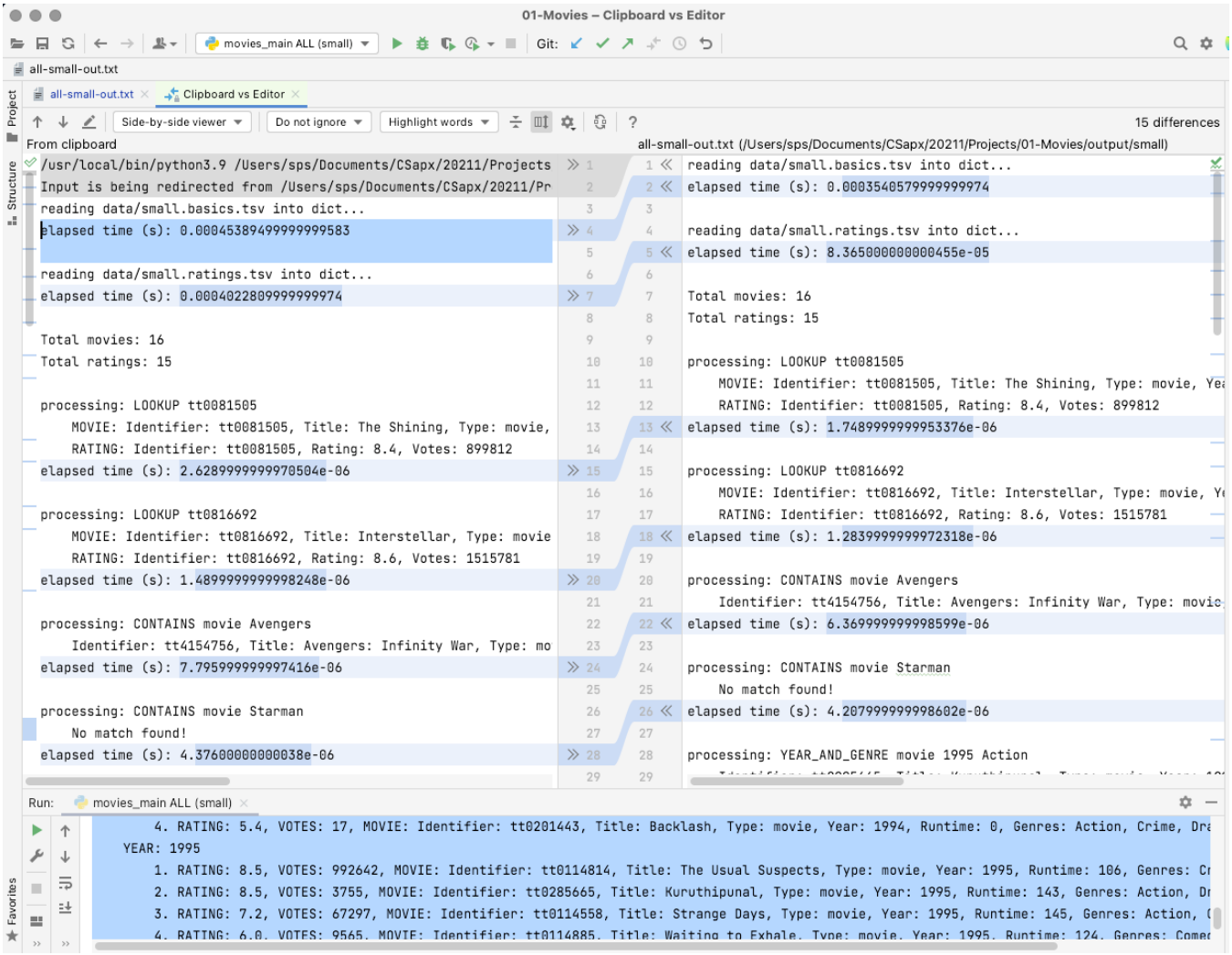
Many of the queries requiring sorting a list of objects. To assist you with understanding how to do this, a demo program, `dataclass_demo.py` is provided to you in the `src` directory. It shows how you can sort the list of dataclass objects by various dataclass fields - in both ascending and descending order. **The key to understand here is if you are sorting by multiple criteria then you apply the sorts in the reverse of the criteria order specified**.

Recall that Python's built in sort algorithm, Timsort, is a stable sort. This means when sorting two or more *equal*, they will appear in the sorted result in the same order they appear in the unsorted collection.

5.4 Comparing Output

The supplied outputs have a lot of text and can be very confusing to compare your output to. PyCharm provides a utility so you can compare a text file to the clipboard in a visual manner.

1. Open the solution output file in the PyCharm editor window.
2. Run your desired query with the output in the console window.
3. Select all the text in the console and copy it to the clipboard.
4. In the solution output editor window right click and select **Compare with Clipboard**.
5. A new editor window will open highlighting the differences.



The ultimate goal here would be to make sure the output matches exactly, minus the timings.

6 Submission

In a file explorer navigate to your project folder. Create a zip file of your `src` directory and name it `project1.zip`. **Please do not zip your entire project folder as it contains your `.git` directory and the large dataset files!**

Upload the zip file to the MyCourses Assignments dropbox by the project due date.

7 Grading

The grade breakdown for this assignment is as follows:

- **Functionality: 70%**
  - Reading Movies and Ratings: 5%
  - LOOKUP query: 5%
  - CONTAINS query: 5%
  - YEAR_GENRE query: 10%
  - RUNTIME query: 10%
  - MOST_VOTES query: 15%
  - TOP query: 20%
- **Design: 20%**
  - Your code supports code reuse and has ample reuse of common function/s.
  - You've also chosen to store your data in structures that are easier for a programmer to work with, e.g. dictionaries containing `namedtuple`'s, `dataclass`'s or `class`'s.
- **Code Documentation and Version Control: 10%**
  - Each Python module you write must have a file header docstring, and each method should have its own docstring.
  - You should have a *reasonable* number of commits that are pushed to GitHub. For this assignment a minimum of **six commits** would be ideal - one for each time you finish implementing a new query.