Student Name: **Weight: 25%**

Student ID:

Project: Classes

Type: Group Assignment

- Students should **ONLY USE** programming constructs covered in the course material.
- **Submissions using programming concepts that are not covered in the course material, using while True, using Boolean flag or any type of flag in loop condition when a clear loop condition exists, and if format that is not covered in the course will be penalized.**
    - **Penalty with no limit could be applied**
- **Late submissions will not be accepted**

Movie Library Management System Scenario

A movie rental store requires a Python application to manage their movie inventory and rental processes. The application should allow users to add, update, remove, search for, rent, and return a movie. The application also allows the user to list movies by genre, list the top rented movies, check the movies availability by genre, and display a summary of the movies' library.

Project Components

The project consists of two main components:

1. **Movie Class (movie.py):** Defines the Movie class with attributes and methods to manage individual movies.
2. **Movie Library Management Module (movie_library_management.py):** Implements the functionalities to load, save, search, rent, return, add, remove movies, etc.

Part 1: Movie Class

Define a Movie class with the following attributes and methods:

Attributes:

- id: Unique identifier for the movie (string).
- title: Movie title (string).
- director: Name of the director (string).

- genre: Integer representing a predefined genre (e.g., 0, 1, 2, etc.).
- available: Boolean indicating availability (True for available, False for rented).
- price: Price of renting the movie (float).
- fine_rate: Fine rate per day for late returns (float).
- rental_count: Number of times the movie has been rented (integer).

Note that movie's attributes MUST be hidden.

Methods:

Method	Description
Constructor	• Initialize movie id, title, director, genre, available, and price (default value is 0) attributes.
Getters	• Implement standard getters for each attribute.
get_genre_name()	• Implement an additional getter method that returns the name of the genre (as a string) according to the following table: **Genre** **Name** 0 Action 1 Comedy 2 Drama 3 Horror 4 Sci-Fi 5 Romance 6 Thriller 7 Animation 8 Documentary 9 Fantasy Hint: implementing this table as a class/static constant would allow your application module to use it too.
get_availability()	• Implement an additional getter method that returns a string based on the available attribute, I.e., If True Then 'Available' Else 'Rented'.
Setters	• Implement standard setter methods for all attributes except available and rental count attributes.
borrow_movie()	• When borrowing a movie, it sets its available attribute to False and increases the rental count attribute by 1.
return_movie()	• When returning a movie, it sets its available attribute to True.
__str__()	• Returns a formatted string representation of the movie. ID Title Director Genre Availability Price $ Rental Count

Part 2 – Movie Library Management Module

Create a **separate module** (e.g. **movie_library_management.py**) for your Movie Library Management application code which will be comprised of various functions, including a *main()* entry function to control overall processing. The functions listed below are required but you may use additional functions too.

Number	Function Name	Description
1	**load_movies(file_name)**	• Parameters: - file_name: The name of the file containing movie data. • Return Value: A list of Movie objects. • Description: Loads movies from a CSV file and returns them as a list of Movie objects.
2	**save_movies(file_name, movies)**	• Parameters: - file_name: The name of the file to save the movie data. - movies: A list of Movie objects. • Return Value: None. • Description: Saves the list of Movie objects to a CSV file.
3	**print_menu()**	• Parameters: None. • Return Value: The user's menu choice as a string. • Description: Displays the main menu and prompts the user for a valid choice.
4	**search_movies(movies, search_term)**	• Parameters: - movies: A list of Movie objects. - search_term: The term to search for in movie titles, directors, or genres. • Return Value: A list of matched Movie objects. • Description: Searches for movies that match the search term.
5	**rent_movie(movies, movie_id)**	• Parameters: - movies: A list of Movie objects. - movie_id: The ID of the movie to rent. • Return Value: A string indicating the result of the rental attempt.

		• Description: Rents a movie by its ID if it is available.
6	**return_movie(movies, movie_id)**	• Parameters: - movies: A list of Movie objects. - movie_id: The ID of the movie to return. • Return Value: A string indicating the result of the return attempt. • Description: Returns a rented movie by its ID.
7	**add_movie(movies)**	• Parameters: - movies: A list of Movie objects. • Return Value: None. • Description: Adds a new movie to the library after prompting the user for details.
8	**remove_movie(movies)**	• Parameters: - movies: A list of Movie objects. • Return Value: None. • Description: Removes a movie from the library by its ID. Displays a message confirming that the movie is removed, or the movie is not found.
9	**update_movie_details(movies)**	• Parameters: - movies: A list of Movie objects. • Return Value: None. • Description: Updates the details of a movie by its ID. Displays a message confirming that the movie is updated, or the movie is not found.
10	**list_movies_by_genre(movies)**	• Parameters: - movies: A list of Movie objects. • Return Value: None. • Description: Lists all movies of a specified genre. Displays a list of movies in the specified genre, no movies found, or invalid genre.
11	**check_availability_by_genre(movies)**	• Parameters: - movies: A list of Movie objects. • Return Value: None.

		• Description: Checks and displays the availability of movies in a specified genre. Displays a list of movies that are available in the specified genre or no available movies.
12	**display_library_summary(movies)**	• Parameters: - movies: A list of Movie objects. • Return Value: None. • Description: Displays a summary of the library, including the total number of movies, number of available movies, and number of rented movies.
13	**top_rented_movies(movies)**	• Parameters: - movies: A list of Movie objects. • Return Value: None. • Description: Displays the top 5 most rented movies based on their rental count. Hint: use the following lambda as a key to sort the movies: lambda movie: movie.get_rental_count()
14	**print_movies(movies)**	• Parameters: - movies: A list of Movie objects. • Return Value: None. • Description: Prints a list of movies in a formatted table.
15	**movie_index(movies, movie_id)**	• Parameters: - movies: A list of Movie objects. - movie_id: The ID of the movie to find. • Return Value: The index of the movie in the list, or None if not found. • Description: Helper function to find the index of a movie by its ID.
16	**main()**	• Parameters: None. • Return Value: None. • Description: The main function that runs the Movie Library Management System. It loads movies from a file, displays the menu, and

		handles user interactions.

CSV File (movies.csv)

The CSV file contains the initial movie data. Each row represents a movie with the following fields:

- **Movie ID**: A unique identifier for the movie.
- **Title**: The title of the movie.
- **Director**: The director of the movie.
- **Genre**: The genre of the movie (as an index from the GENRES list).
- **Available**: A boolean indicating whether the movie is available for rent.
- **Price**: The rental price of the movie.

Make sure that your application maintains the comma-separated format established in this file.

Test Plan

Please refer to the accompanying document, *Winter 2025 - Project Sample Run.pdf*.

Make sure your program output matches this sample run. When doing your final test run, be sure to begin with a fresh starting data file.

Equipment and Materials

For this assignment, you will need:

- Python IDE
- Git software
- GitHub repository

Instructions

This assignment consists of two parts, both to be completed outside of class time. See the course schedule and Brightspace for exact dates. The rubric is available in Brightspace.

Project Part 1: (No Submission)

1. Create the Movie class as outlined in the document (recommended filename: movie.*py*).
2. Test the movie class to verify its correct operation by running the supplied *test_movie.py* program.
3. Each student is encouraged to individually complete Part 1. At the very least, all group members MUST participate in Part 1.

Project Part 2: (Submission)

1. Have one member of your group set up a GitHub repository for Part 2 of this project. Make sure that this GitHub repository is **private**. Add group members and your instructor as collaborators.
2. Create a separate branch in GitHub for each group member, containing the part they will work on. The branch name should include the task name and the student name.
3. Develop the code for the Movie Library Management Module as specified in this document. As a team, you will also need to assess each member's Movie class and then finalize one version for your group's solution.
4. GitHub must be effectively used for group collaboration.
5. Ensure all group members publish their parts to their respective GitHub branches.
6. The final project code must be published to master/main on GitHub.
7. Check your solution against the detailed marking criteria available in Brightspace.
8. Create a project video.
   Each group will submit a **10-15 mins video** to Brightspace or their GitHub Repository (either as a URL or an uploaded video file). In the video:
   - EACH team member will introduce themselves – camera should be on. Please use laptop camera/screen share to create video (not a phone).

- Share the **group's final solution** (Python code) on screen and have EACH team member explain a **portion of the code** that they developed. A debugger may be used to step through sections of code (optional).
- Provide a demonstration of the application covering all the functionality CORRESPONDING TO THE PROVIDED SAMPLE RUNS.
- Ensure that the video is playable and has good audio quality.

9. **Submit the following files to Brightspace**:
    - A ZIP file for the whole project folder that includes:
        - The code of the program that you implemented (.py files).
        - A copy of your sample runs (.txt files) that correspond to the sample runs provided.
        - All data files updated by program after the test runs have been completed.
    - A link to your group's GitHub project repository.
    - Video presentation.
    - Peer assessment.

Peer Assessment

- **Expecting group members equally participate in completing the project tasks (e.g., coding, code style, documentation, testing, video, etc.).**
- Each member should assess the contribution/participation of all other group members.
- Each member should assign a mark out of 10 to other members.
- Ensure that the task description is clear and accurately reflects the actual participation.
- **Grades will be assigned based on the average achieved by each member.**
- Please use the following table to complete the peer assessment. Marks/tasks are just samples.

Member/ Reviewer	Member 1 Name	Member 2 Name	Member 3 Name	Completed Tasks
Member 1 Name	N/A	9	10	• Task 1 description • Task 2 description
Member 2 Name	8	N/A	9	• Task 3 description • Task 4 description
Member 3 Name	9	10	N/A	• Task 5 description • Task 6 description
Average	8.5	9.5	9.5	

A student can receive a deduction on their individual grade for the assignment if they do not meet the expected contribution/participation based on the feedback of other group members.

Grading

Item	Percentage
Working Code	**60**
Style	**10**
Documentation	**5**
Testing	**10**
Video	**5**
Version Control	**5**
Peer Assessment	**5**
Total	**100**

Marking Criteria

Rubric available in Brightspace.