**Member 1**: Hi, we’re Group [Your Group Name], and this is our Movie Library Management System for the Winter 2025 project. I’m [Member 1].  
**Member 2**: I’m [Member 2].  
**Member 3**: I’m [Member 3].  
**Member 4**: And I’m [Member 4]. Today, we’ll explain our code from movie.py and movie\_library\_management.py and demonstrate the system’s functionality.

**Member 1: Movie Class Overview (movie.py) (~1 minute 15 seconds)**

**[Screen share: Display movie.py code, focusing on class definition and attributes]**

Hi, I’m [Member 1], and I’ll start by explaining the Movie class in movie.py. This class is the core of our system, representing individual movies in the library.

The Movie class has private attributes like \_\_id, \_\_title, \_\_director, \_\_genre, \_\_available, \_\_price, \_\_fine\_rate, and \_\_rental\_count. These store details such as the movie’s unique ID, title, director, genre as an integer, availability status, rental price, fine rate for late returns, and how many times it’s been rented. We used a class constant, GENRES, to map genre integers to names like "Action" or "Sci-Fi."

The constructor initializes these attributes, with defaults for availability as True, price as 0.0, and fine rate as 1.0. We implemented getter methods, like get\_id() and get\_title(), to access attributes safely. Special getters, get\_genre\_name() and get\_availability(), convert the genre integer to a string and return "Available" or "Rented" based on the \_\_available flag. Setter methods allow updating attributes like title or price, but not availability or rental count, which are modified only through specific methods.

The borrow\_movie() method sets \_\_available to False and increments \_\_rental\_count, while return\_movie() sets \_\_available to True. The \_\_str\_\_ method formats movie details into a string for display, aligning with the project’s formatted output requirements.

This class provides a robust foundation for managing movie data, which the main module builds upon.

**Member 2: Core Management Functions (movie\_library\_management.py, Part 1) (~1 minute 15 seconds)**

**[Screen share: Display movie\_library\_management.py, focusing on load\_movies, save\_movies, and print\_menu functions]**

I’m [Member 2], and I’ll cover the core management functions in movie\_library\_management.py. This module handles the system’s operations, interacting with the Movie class and the movies.csv file.

The load\_movies function reads movie data from a CSV file, like movies.csv, which contains fields for ID, title, director, genre, availability, and price. It splits each line, creates Movie objects, and returns a list of movies. If the file doesn’t exist, it prints an error and starts with an empty list.

The save\_movies function writes the movie list back to the CSV file, ensuring data persistence. It formats each movie’s attributes, including availability as "True" or "False," into a comma-separated line.

The print\_menu function displays the main menu with 11 options, from searching movies to exiting the system. It uses a while loop for input validation, ensuring the user enters a number between 0 and 10. This menu drives the user interface, making the system interactive and user-friendly.

These functions set up the system’s data handling and user interaction, which other functions build on for specific tasks.

**Member 3: Movie Operations (movie\_library\_management.py, Part 2) (~1 minute 15 seconds)**

**[Screen share: Display movie\_library\_management.py, focusing on add\_movie, remove\_movie, rent\_movie, and return\_movie functions]**

I’m [Member 3], and I’ll explain the functions for managing movie operations in movie\_library\_management.py.

The add\_movie function prompts the user for a new movie’s ID, title, director, genre, and price. It validates inputs using while loops—ensuring the ID is unique and numeric, genre is between 0 and 9, and price is a valid float. A new Movie object is created and added to the movie list.

The remove\_movie function takes a movie ID, validates it, and uses the helper function movie\_index to find the movie. If found, it removes the movie and confirms the action; otherwise, it reports the movie wasn’t found.

The rent\_movie function allows renting a movie by ID. It checks if the movie exists and is available using movie\_index and get\_availability. If valid, it calls borrow\_movie to update the status and returns a success message. Similarly, return\_movie checks if the movie is rented, calls return\_movie to set it as available, and confirms the return.

These functions handle key user interactions for adding, removing, renting, and returning movies, ensuring robust library management.

**Member 4: Search, Reporting, and Demo (movie\_library\_management.py, Part 3) (~1 minute 15 seconds)**

**[Screen share: Display movie\_library\_management.py, focusing on search\_movies, top\_rented\_movies, and display\_library\_summary, then switch to a live demo]**

I’m [Member 4], and I’ll cover the search and reporting functions, followed by a brief demo.

The search\_movies function searches for movies by a term in the title, director, or genre name. It converts the search term to lowercase for case-insensitive matching and returns a list of matching Movie objects, which are displayed using print\_movies.

The top\_rented\_movies function sorts movies by rental\_count in descending order using a lambda function and displays the top five with non-zero rentals. It formats a table with ID, title, director, genre, and rental count.

The display\_library\_summary function counts total movies, available movies, and rented movies, providing a quick overview of the library’s status.

**[Switch to live demo]** Now, let’s demonstrate the system. [Run the program, load movies.csv, and show key features: search for “Nolan,” rent movie ID 1, return movie ID 11, add a movie, and display the summary.] As you can see, our system matches the sample run, handling all required functionalities efficiently.