CSE 1321L: Programming and Problem Solving I Lab

Assignment 7

OOP & Review

What students will learn

- Problem Solving.
- Basic Program Structure.
- Input and Output with the user.
- Write code that includes while/for loop and nested loops logic.
- Structure program to include methods and lists.

Content

- Overview
- Assignment7A: Movie Rental System
- Assignment7B: Student Performance Tracker

Overview:

For this assignment, you're going to practice making logic in your code. It will include loops, selection statements, functions and lists. In practical terms, this means you're going to expand on the concepts from previous assignments. Further, you will also implement one case study for OOP practices.

Final note: *Do not cheat*

If your temptation is to look online, don't. Come see us instead and ask questions – we are here to help. Remember, you are going to have to write codes in your future job interviews, so learn it now to secure a high-paying job later.

Assignment7A: Movie Rental System

In this question, we are going to write a program that can manage a movie rental store. Before the rise of modern digital platforms for streaming movies, such stores were popular for distributing the latest movies and games through physical copies, often in the form of DVDs.

Requirements:

1. Design a **Movie Rental System** using two classes: Movie and Customer.
2. The system should allow customers to rent and return movies according to specific rules.
3. Class Definitions (should be in a separate file named movie_class.py & customer_class.py):
    - Define both classes in a separate file and import them into your main program file using
        - from movie_class import Movie
        - from customer_class import Customer

Movie Class:

- **Attributes** (defined in __init__ method):
    - title (string): The title of the movie.
    - director (string): The name of the movie's director.
    - is_rented (boolean): Tracks whether the movie is currently rented.
- **Methods:**
    - rent(): Marks the movie as rented if it is not already.
        - Returns "You rented the movie" if successful, or "Movie already rented" if it is.
    - return_movie(): Marks the movie as returned if it was rented.
        - Returns "You returned the movie" if successful, or "Movie is not rented" if it wasn't.

Customer Class:

- **Attributes** (defined in __init__ method):
    - name (string): The name of the customer.
    - rented_movies (list): A list of Movie objects currently rented by the customer.
- **Methods:**
    - rent_movie(movie): Allows the customer to rent a movie according to these rules:
        - Customers can rent a maximum of 2 movies at a time. If they try to rent more, return "Rental limit reached".
        - If the customer already rented the movie, return "Movie already rented by this customer".
        - If the movie is rented by someone else, return "Movie already rented by someone else".
        - Otherwise, add the movie to rented_movies and return "Movie rented successfully".
    - return_movie(movie): Allows the customer to return a movie according to these rules:
        - If the customer didn't rent the movie, return "Movie not rented by this customer".
        - If they have, mark the movie as returned, remove it from rented_movies, and return "Movie returned successfully".

Testing the Code:

Define a movie and customer to test the renting and returning process in your main file.

Scenario based outputs are shown below. Once you implemented classes you can check your outputs by using the code below. You will write this code into main function and match the output,

```
# Define a Movie and Customer for testing
movie1 = Movie("Inception", "Christopher Nolan")
customer1 = Customer("Ethan")

# Scenario 1: Ethan rents the movie "Inception"
print(customer1.rent_movie(movie1))
# Output: "Movie rented successfully"

# Scenario 2: Ethan tries to rent "Inception" again
print(customer1.rent_movie(movie1))
# Output: "Movie already rented by this customer"

# Scenario 3: Ethan returns "Inception"
print(customer1.return_movie(movie1))
# Output: "Movie returned successfully"

# Scenario 4: Ethan tries to return "Inception" again
print(customer1.return_movie(movie1))
# Output: "Movie not rented by this customer"

# Scenario 5: Another customer, Olivia, tries to rent "Inception"
customer2 = Customer("Olivia")
print(customer2.rent_movie(movie1))
# Output: "Movie rented successfully"

# Scenario 6: Ethan tries to rent more than the limit of 2 movies
# Define more movies
movie2 = Movie("The Matrix", "Wachowskis")
movie3 = Movie("Interstellar", "Christopher Nolan")

# Ethan rents 2 movies
print(customer1.rent_movie(movie2))
# Output: "Movie rented successfully"
print(customer1.rent_movie(movie3))
# Output: "Movie rented successfully"

# Ethan tries to rent a third movie
movie4 = Movie("Avatar", "James Cameron")
print(customer1.rent_movie(movie4))
# Output: "Rental limit reached"
```

Assignment7B: Student Performance Tracker

Write a Python program to build a **Student Performance Tracker** that allows the user to enter information for multiple students and their scores in three subjects. For each student, the program should:

- Calculate the **total score** by summing the scores of all three subjects.
- Calculate the **average score** as:
  average = total_score / number_of_subjects
- Assign a **letter grade** based on the average score using the following scale:
  - A: 90–100
  - B: 80–89
  - C: 70–79
  - D: 60–69
  - F: Below 60
- Determine **pass/fail status**: Students with grades A to D pass, and F indicates failure.

You must ensure that:

- Scores must be between 0 and 100. If an invalid input is entered, ask the user to re-enter the score.
- The program should process multiple students using repetition structures (loops).
- Each student's data should be stored in a **dictionary** with keys: "name", "scores", "total", "average", "grade", and "status".
- All student dictionaries should be stored in a **list**.
- There are at least two functions in your program to calculate avg score of a student and letter grade.
  - calculate_average(scores: list) -> float
  - get_grade(avg: float) -> str

At the end of the program, display:

- A report of each student's information.
- The **class average score**.
- The **number of students who passed and failed**.
- The name of the **top-performing student** based on average score.

Example runs are shown below. The user input is shown in **red**.

Sample Output #1:

```
Enter number of students: 2

Enter details for student 1:
Enter student name: Alice
Enter score for Math: 87
Enter score for English: 92
Enter score for Science: 95
Total score: 274.0
Average score: 91.33
Grade: A
Status: Pass

Enter details for student 2:
Enter student name: Bob
Enter score for Math: 54
Enter score for English: 68
Enter score for Science: 23
Total score: 145.0
Average score: 48.33
Grade: F
Status: Fail

==============================
        Summary Report
==============================

Name: Alice
Scores: [87.0, 92.0, 95.0]
Total: 274.0
Average: 91.33
Grade: A
Status: Pass

Name: Bob
Scores: [54.0, 68.0, 23.0]
Total: 145.0
Average: 48.33
Grade: F
Status: Fail

Class Average: 69.83
Students Passed: 1
Students Failed: 1
Top Student: Alice
```

Sample Output #2:

```
Enter number of students: 2

Enter details for student 1:
Enter student name: Peter
Enter score for Math: 85
Enter score for English: 90
Enter score for Science: 88
Total score: 263.0
Average score: 87.67
Grade: B
Status: Pass

Enter details for student 2:
Enter student name: Alice
Enter score for Math: 87
Enter score for English: 63
Enter score for Science: 98
Total score: 248.0
Average score: 82.67
Grade: B
Status: Pass

==============================
        Summary Report
==============================

Name: Peter
Scores: [85.0, 90.0, 88.0]
Total: 263.0
Average: 87.67
Grade: B
Status: Pass

Name: Alice
Scores: [87.0, 63.0, 98.0]
Total: 248.0
Average: 82.67
Grade: B
Status: Pass

Class Average: 85.17
Students Passed: 2
Students Failed: 0
Top Student: Peter
```

Submission Instructions:

- Programs must follow the output format provided. This includes each blank line, colons (:), and other symbols.
- Programs must be working correctly.
- Programs must be written in Python.
- Programs must be submitted with the correct **.py** format.
- Programs must be saved in files with the correct file name:
  - Assignment7A.py (main file)
    - Movie_class.py
    - Customer_class.py
  - Assignment7B.py
- Programs (source code files) must be uploaded to Gradescope by the due date.