Low-Level Design (LLD) – Online Course Manager (OOPs in Python)

Difficulty Level: Easy - Intermediate | Total Marks: 25
Design Format: 1 Class with 5 Independent Methods | 5 Visible Test Cases

Summary of Design Requirements

Implement a class CourseManager using Object-Oriented Programming (OOP).

The class should:

- Track various courses and the students enrolled in them.
- Manage enrollment limits and student registrations.

Each function must be independent:

- Each method must work based on passed or internal data only.

Use simple Python data structures (dict, list).

Avoid external libraries.

Concepts Tested

- Python class and instance methods
- State management using self
- List-in-Dictionary data structures
- Membership testing and length validation
- String formatting and conditional returns

Problem Statement

Design a Python class CourseManager to handle an educational platform's course registrations with the following features:

- Core catalog management (Add courses).
- Student registration with capacity constraints.
- Real-time seat availability tracking.
- Student removal and list management.

Each course has:
- Course ID (unique identifier)
- Title (name of the course)
- Capacity (max number of students allowed)
- Enrollment List (list of names of students currently in the course)

Operations (Methods)

1. Initialize Manager (Easy)

Create an internal data structure to store courses.

```
def __init__(self):
    self.courses → {course_id: {'title': str, 'max': int, 'students': list[str]}}
```

2. Add Course (Easy)

Define a new course and set its maximum capacity.

```
def add_course(self, course_id: str, title: str, capacity: int) -> None:
    - Add to self.courses with an empty list for 'students'.
    - No return value required.
```

3. Register Student (Easy)

Add a student to a course if the course exists and is not at full capacity.

```
def register_student(self, course_id: str, name: str) -> str:
    - If course_id doesn't exist: return "Invalid ID"
    - If course is at capacity: return "Course Full"
    - Else: add name to student list, return "Enrolled in [title]"
```

4. Get Available Seats (Medium)

Calculate how many spots are left in a specific course.

```
def get_available_seats(self, course_id: str) -> int:
    - If course_id exists: return (capacity - current_enrollment_count)
    - If course_id doesn't exist: return -1
```

5. Remove Student (Medium)

Remove a student from a course and return a status message.

```
def remove_student(self, course_id: str, name: str) -> str:
  - If course exists and name is in student list:
    - Remove the name from the list.
    - Return "Removed [name] from [title]"
  - Else: return "Registration not found"
```

Test Cases & Marks Allocation

Test Case ID	**Description**	**Method**	**marks**
TC1	Initialize manager structure	__init__()	5
TC2	Add a new course to platform	add_course()	5
TC3	Enroll student with capacity check	register_student()	5
TC4	Check for remaining seats	get_available_seats()	5
TC5	Unenroll student from a course	remove_student()	5
	Total		**25**

Visible Test Case Descriptions

TC1: Instantiating CourseManager() should initialize courses as an empty dictionary.

TC2: add_course("PY101", "Python Basics", 2) created the course entry in inventory.

TC3: register_student("PY101", "Bob") appends "Bob" to the list and returns "Enrolled in Python Basics".

TC4: get_available_seats("PY101") returns 1 if one student is currently enrolled and capacity is 2.

TC5: remove_student("PY101", "Bob") removes the student and returns the confirmation string.