**Lab 8**

1. Suppose you are developing a basic bank account management system that supports different account types and banking operations. Now,
    i. Create a BankAccount class with attributes like account_number, balance, and methods deposit and withdraw. Add conditions to ensure that withdrawal does not exceed the balance and that deposits must be positive.
    ii. Using inheritance, create a SavingsAccount class that inherits from BankAccount and adds an interest rate attribute. Write a method to calculate interest based on the current balance and apply it.
    iii. Create a Customer class with attributes customer_id and accounts (a list of bank accounts). Implement methods to add accounts and display the customer's account details.
    iv. Add conditional statements in the withdraw method to handle scenarios like insufficient funds or minimum balance requirements.

2. Develop a program that generates Fibonacci numbers and can calculate specific terms in the sequence. Now complete the following tasks,
    i. Create a Fibonacci class with a method generate_sequence(n) that returns the first n Fibonacci numbers using a loop and conditional statements to handle the base cases.
    ii. Implement a method get_nth_term(n) in the Fibonacci class that returns the nth Fibonacci number. Add a condition to check if n is less than 1 and handle it by returning an error message.
    iii. Using inheritance, create a MemoizedFibonacci class that inherits from Fibonacci and uses a dictionary to store previously calculated terms for optimized calculation.
    iv. Write a conditional statement within generate_sequence(n) to avoid negative input and handle non-integer inputs with an error message.
    v. Test the class by creating an object and printing the Fibonacci sequence for the first 10 numbers and the 15th Fibonacci number.

3. Create an employee management system that handles different types of employees and calculates their salaries based on conditions. Define a Employee class with attributes name, id, and base_salary. Add a method calculate_salary() that returns the base salary. Using inheritance, create HourlyEmployee and CommissionEmployee classes. HourlyEmployee should have an additional attribute for hours worked, and CommissionEmployee should have sales and commission rate. Override calculate_salary() in both subclasses. Write conditional statements within calculate_salary() for HourlyEmployee to ensure that hours do not exceed a certain limit, e.g., 40 hours per week, with any extra hours counted as overtime. Add a condition to CommissionEmployee's calculate_salary() to provide a bonus if sales exceed a specified threshold.
4. Design a grading system that assigns letter grades to students and calculates their GPA based on different grading criteria.

i. Create a Student class with attributes name, student_id, and a dictionary to store subjects and their respective scores.
ii. Write a calculate_grade method that assigns letter grades based on conditions: A for 90-100, B for 80-89, C for 70-79, etc. Use conditional statements for the grading logic.
iii. Using inheritance, create a GraduateStudent class that inherits from Student but has stricter grading criteria. Override calculate_grade with tighter score ranges.
iv. Implement a calculate_gpa method that uses the assigned letter grades to calculate a GPA (e.g., A = 4.0, B = 3.0). Add a conditional statement to handle subjects with a grade lower than a passing grade.
v. Test the grading system by creating student objects with various scores, assigning letter grades, calculating their GPA, and printing the results.