Lab Task OOP

Question # 1

Create a Python program that simulates a student management system. Define four classes: Person, Student, Teacher, and Admin. The Person class will serve as the base class, while the Student, Teacher, and Admin classes will be derived from it using hybrid inheritance.

1. Define the Person class with attributes for name, age, and gender. This class should also have methods for displaying basic information about a person.
2. Define the Student class, which inherits from the Person class. The Student class should have additional attributes for roll number and class, as well as methods for displaying student-specific information.
3. Define the Teacher class, which also inherits from the Person class. The Teacher class should have additional attributes for subject and experience, as well as methods for displaying teacher-specific information.
4. Define the Admin class, which inherits from both the Student and Teacher classes. The Admin class should have additional attributes for salary and position, as well as methods for displaying admin-specific information.
5. Prompt the user to enter details for a student, a teacher, and an admin.
6. Create objects of the Student, Teacher, and Admin classes with the provided details.
7. Display the information of all three objects using their respective display methods.

Question # 2

The objective of this lab task is to create a class hierarchy that represents different types of bank accounts and implement basic banking operations. This will help students learn the fundamental concepts of object-oriented programming (OOP) in Python.

Tasks:

1. Create a base class 'Account':

   This class will represent the basic structure of a bank account and define common attributes and methods applicable to all types of accounts.

   a. Attributes:

- `account_number`: A unique identifier for the account.
- `balance`: The current balance of the account.
- `account_holder_name`: The name of the account holder.

b. Methods:

- `deposit(amount)`: Deposits the specified amount into the account balance.
- `withdraw(amount)`: Withdraws the specified amount from the account balance, ensuring it does not exceed the account balance.

2. Create a subclass 'SavingsAccount' that inherits from 'Account':

   This class will represent a savings account, adding an attribute for the interest rate and defining a method to calculate and add interest to the account balance.

   a. Attribute:

   - `interest_rate`: The annual interest rate applied to the account balance.

   b. Method:

   - `calculate_interest()`: Calculates the interest earned based on the interest rate and adds it to the account balance.

3. Create a subclass 'CurrentAccount' that inherits from 'Account':

   This class will represent a current account, adding an attribute for the overdraft limit and modifying the withdrawal behavior to account for overdraft scenarios.

   a. Attribute:

   - `overdraft_limit`: The maximum amount that can be withdrawn beyond the account balance.

   b. Method:

   - `check_overdraft(amount)`: Checks if the specified withdrawal amount exceeds the account balance and overdraft limit.

- `withdraw(amount)`: Overrides the `withdraw` method from the base class to check for overdraft and adjust the withdrawal accordingly.

4. Create instances of each class and perform banking operations:

   Instantiating objects of the `SavingsAccount` and `CurrentAccount` classes and calling their respective methods will demonstrate the implementation of object-oriented concepts and simulate real-world banking operations.