Section A – Short Answer Questions (2 marks each)

1. **Define Object-Oriented Programming. What are its main principles?**
   Answer: Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects", which contain data and methods.
   Main principles:

- **Encapsulation**
- **Abstraction**
- **Inheritance**
- **Polymorphism**

2. Differentiate between class and object with an example.
Answer:

- **Class** is a blueprint or template for creating objects.
- **Object** is an instance of a class.

python

```
class Car:  # Class
    def __init__(self, brand):
        self.brand = brand

my_car = Car("Toyota")  # Object
```

3. What are the three types of access specifiers in Python? Explain each briefly.
Answer:

- **Public (var)**: Accessible from anywhere.
- **Protected (_var)**: Accessible within the class and its subclasses.
- **Private (__var)**: Accessible only within the class.

4. What is a constructor in Python? What is its special method name?
Answer:
A constructor is a special method used to initialize objects.
Special method name: __init__

5. What is the purpose of the super() function in inheritance?
Answer:
The super() function allows access to methods of a parent class, especially useful in method overriding.

6. Explain the difference between method overloading and method overriding.
Answer:

- **Overloading**: Same method name with different parameters (not directly supported in Python, simulated with default arguments).
- **Overriding**: Redefining a method in a child class that exists in the parent class.

7. What is polymorphism in OOP? Give a simple example.
Answer:
Polymorphism allows methods to behave differently based on the object.

python

```
class Dog:
    def sound(self):
        print("Bark")

class Cat:
    def sound(self):
        print("Meow")

def make_sound(animal):
    animal.sound()

make_sound(Dog())  # Output: Bark
make_sound(Cat())  # Output: Meow
```

8. Can you call a private method outside its class in Python? If yes, how?
Answer:
Yes, using name mangling: _ClassName__methodName

python

```
class Test:
    def __private_method(self):
        print("Private")

obj = Test()
obj._Test__private_method()  # Accessing private method
```

Section B – Coding-Based Questions (5 marks each)
9.
Write a Python class Student with attributes name and marks. Include a constructor to initialize the attributes and a method to display student details.

```
# Expected Output:
# Name: Alice
# Marks: 90
```

Ans:

```
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Marks: {self.marks}")

# Expected Output:
student1 = Student("Alice", 90)
student1.display_details()
```

10.Create a class Vehicle with a method start(). Inherit a class Car from it and override the start() method. Call both parent and child methods using an object of Car.?

Ans:

```
class Vehicle:
    def start(self):
        print("Vehicle is starting...")

class Car(Vehicle):
    def start(self):
        # Optionally, call the parent class's start() method if required:
        super().start()
        print("Car is starting with a roar!")

# Demonstration:
my_car = Car()
my_car.start()
```

11.Demonstrate the use of public, protected, and private variables in a class using appropriate naming conventions and access.?

Ans:

```
class AccessDemo:
    def __init__(self):
        self.public_var = "I am public"         # Public member
        self._protected_var = "I am protected"   # Protected member
        self.__private_var = "I am private"      # Private member via name mangling
```

```
    def get_private(self):
        return self.__private_var

# Testing the AccessDemo class
demo = AccessDemo()
print("Public Variable:", demo.public_var)           # Accessible anywhere
print("Protected Variable:", demo._protected_var)      # Accessible by convention; should be used
with care
# The following line would normally produce an AttributeError:
# print(demo.__private_var)
# Correct way to access the private variable via a public method:
print("Private Variable via method:", demo.get_private())
```

12.Write a class hierarchy where Animal is the parent class and Dog and Cat are derived classes. Implement a method make_sound() in each class to demonstrate polymorphism.

```
class Animal:
    def make_sound(self):
        print("Some generic animal sound")

class Dog(Animal):
    def make_sound(self):
        print("Bark")

class Cat(Animal):
    def make_sound(self):
        print("Meow")

# Polymorphism demo:
animals = [Dog(), Cat(), Animal()]
for animal in animals:
    animal.make_sound()
```

Section C – Application-Based Questions (8 marks each)

13.
Create a class BankAccount with:
• private variable __balance
• methods to deposit, withdraw, and display balance
• prevent withdrawal if amount exceeds balance
Demonstrate the functionality by creating an object and calling methods.?

```
class BankAccount:
    def __init__(self, initial_balance=0):
        self.__balance = initial_balance  # Private variable

    def deposit(self, amount):
        if amount > 0:
```

```
            self.__balance += amount
            print(f"Deposited: {amount}")
        else:
            print("Invalid deposit amount.")

    def withdraw(self, amount):
        if amount > self.__balance:
            print("Insufficient balance!")
        elif amount <= 0:
            print("Enter a valid amount to withdraw.")
        else:
            self.__balance -= amount
            print(f"Withdrawn: {amount}")

    def display_balance(self):
        print(f"Current Balance: {self.__balance}")

# Demonstration:
account = BankAccount(1000)
account.display_balance()
account.deposit(500)
account.display_balance()
account.withdraw(2000)  # Should warn about insufficient balance
account.withdraw(300)
account.display_balance()
```

14.

Write a program to create a class Employee with attributes name, id, and salary. Use inheritance to create a subclass Manager that adds an additional attribute department. Use a constructor to initialize all attributes. Override a method display() to show complete info.?

Ans:

```
class Employee:
    def __init__(self, name, emp_id, salary):
        self.name = name
        self.emp_id = emp_id
        self.salary = salary

    def display(self):
        print(f"Employee Name: {self.name}")
        print(f"Employee ID: {self.emp_id}")
        print(f"Salary: {self.salary}")

class Manager(Employee):
    def __init__(self, name, emp_id, salary, department):
        super().__init__(name, emp_id, salary)
        self.department = department

    # Overriding the display method to include department info
    def display(self):
```

```
        super().display()
        print(f"Department: {self.department}")

# Testing the classes:
emp = Employee("John Doe", 101, 50000)
mgr = Manager("Alice Smith", 102, 75000, "Sales")

print("Employee Details:")
emp.display()

print("\nManager Details:")
mgr.display()
```