**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 technique based on the concept of "objects", which can contain data and code. The main principles of OOP are:

* **Encapsulation** – Hiding internal details and showing only necessary features.
* **Inheritance** – Acquiring properties of one class into another.
* **Polymorphism** – Performing a single action in different ways.
* **Abstraction** – Hiding complex implementation and showing only essential details.

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

* A **class** is a blueprint for creating objects.
* An **object** is an instance of a class.

Example:

Example of class:

class Car:

def \_\_init\_\_(self, brand):

self.brand = brand

Example of object:

car1 = Car("Toyota")

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

* **Public (name)** – Accessible from anywhere.
* **Protected (\_name)** – Meant to be accessed within the class and its subclasses.
* **Private (\_\_name)** – Can’t be accessed directly outside 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 when they are created.  
The special method name is \_\_init\_\_().

**5. What is the purpose of the super() function in inheritance?**  
**Answer:**  
The super() function is used to call a method from the parent class. It allows access to the parent class’s methods and constructors, useful for extending functionality.

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

* **Overloading**: Same method name with different parameters.
* **Overriding**: Subclass defines a method with the same name as in the parent class to change behaviour.

**7. What is polymorphism in OOP? Give a simple example.**  
**Answer:**  
Polymorphism means performing the same operation in different ways.  
Example:

class Cat:

def sound(self):

print("Meow")

class Dog:

def sound(self):

print("Bark")

def animal\_sound(animal):

animal.sound()

animal\_sound(Cat())

animal\_sound(Dog())

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

class Demo:

def \_\_private(self):

print("Private Method")

obj = Demo()

obj.\_Demo\_\_private() # 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.**  
**Answer:**

class Student:

def \_\_init\_\_(self, name, marks):

self.name = name

self.marks = marks

def display(self):

print(f"Name: {self.name}")

print(f"Marks: {self.marks}")

s1 = Student("meg", 89)

s1.display()

**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.**  
**Answer:**

class Vehicle:

def start(self):

print("Vehicle started")

class Car(Vehicle):

def start(self):

super().start()

print("Car started")

c = Car()

c.start()

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

class Sample:

def \_\_init\_\_(self):

self.public = "Public"

self.\_protected = "Protected"

self.\_\_private = "Private"

def display(self):

print(self.public)

print(self.\_protected)

print(self.\_\_private)

obj = Sample()

obj.display()

print(obj.public)

print(obj.\_protected)

print(obj.\_Sample\_\_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.**  
**Answer:**

class Animal:

def make\_sound(self):

print("Some sound")

class Dog(Animal):

def make\_sound(self):

print("Bark")

class Cat(Animal):

def make\_sound(self):

print("Meow")

animals = [Dog(), Cat()]

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  
**Answer:**

class BankAccount:

def \_\_init\_\_(self):

self.\_\_balance = 0

def deposit(self, amount):

if amount > 0:

self.\_\_balance += amount

print(f"Deposited: {amount}")

else:

print("Invalid amount")

def withdraw(self, amount):

if amount <= self.\_\_balance:

self.\_\_balance -= amount

print(f"Withdrew: {amount}")

else:

print("Insufficient balance")

def display\_balance(self):

print(f"Balance: {self.\_\_balance}")

account = BankAccount()

account.deposit(1000)

account.withdraw(500)

account.withdraw(600)

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.**  
**Answer:**

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"Name: {self.name}")

print(f"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

def display(self):

super().display()

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

m1 = Manager("John", 101, 75000, "IT")

m1.display()